Document No: WG21 N3141
INCITS / PL22.16 10-0131
Date: 2010-10-05
Project: Programming Language C++
References: WG21 N3092, SC 22 N4512: ISO/IEC FCD 14882
Reply to: Barry Hedquist
INCITS/PL22.16 IR
beh@peren.com

ISO/IEC FCD 14882, C++0X, National Body Comments

Attached is the latest revision of the complete set of National Body Comments submitted to JTC1 SC22 in response to the SC22 Letter Ballot for ISO/IEC FCD 14882, Final Committee Draft of the revision of ISO/IEC 14882:2003, aka C++0X.

This document differs from SC22 N4547 by adding editorial updates that do not change the substance of submitted comments, but correct unintended typos omissions and errors introduced when the individual source documents were converted and added to this merged listing. Prior corrections are:

- adding missing Additional Details for GB 138
- adding missing Additional Details for US 23
- deleted "US" from title on Ballot Comment form
- replaced Additional Details for US 26, 181, 194 with pages that correctly indicate the proposed changes to wording in the FCD.
- made editorial corrections caused by the deletion of bracketed text during the conversion of HTML source documents to GB 18, 37, 38, 43, 44, 58, 59, 64, 82, 91, 99, 104, 105, 108, 118, 133,142.
- made corrections to mangled text representations in CA 8, 9, 12, 14, 18 19, 20, 21.

Corrections for this revision (N3141) are the renumbering of two comments from Japan:

- JP 98, 30.3.1.5, p9 - corrected to JP 97
- JP 99, 30.3.1.5, p14 - corrected to JP 98.

None of the above corrections constitute a substantive change to the balloted comments, nor do they introduce new comments not balloted.

Document numbers referenced in the ballot comments are WG21 documents unless otherwise stated.

This document will serve as the basis for submission of WG21 responses to FCD 14882 ballot comments upon completion of ballot resolution.

Template for comments and secretariat observations

Date: 5 Oct 2010	Document: **N3141 FCD 14882 Ballot Comments**

1	2	(3)	4	5	(6)	(7)
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Secretariat observations** on each comment submitted
ITTF 01	General		ed	The ISO/IEC headers and footers should appear at the top and bottom of every page.	Insert the ISO/IEC headers and footers at the top and bottom of every page.	
US 01	1-30		ge	It appears that the C committee could possibly make some changes for C1X that we may still want to follow for C++0X to avoid gratuitous incompatibilities.	Make any appropriate changes to avoid unnecessary incompatibilities with C1X resulting from changes to the WG14 C standard draft.	
US 02	1-30		ge	The active issues identified in the CWG and LWG issues lists as if the date that the FCD was published (N3083 and N3087) must be addressed and appropriate action taken.	Appropriate action would include making changes to the FCD, identifying an issue as not requiring a change to the FCD, or deferring an issue to a later draft or a later standard.	
DE 1	1 through 15		te	Consider applying the resolutions of the active core issues in Ready status (see WG21 N3083).		
CH 1	all		ge/te	The issues on the issues lists (WG21 N3083 and N3087) shall be addressed before the standard becomes final.		
US 03	1 - 29		te	The threading model does not make basic guarantees needed to write correct programs. We should not repeat the error that POSIX made in early standards (and later corrected).	Add requirements that all no-blocked threads will (however slowly) make progress and that all visible side effects will (eventually) be seen by other threads. Possibly use the word “should” if an absolute requirement is impossible.	
US 04	all	all	ed	Many identifiers are hyphenated and broken across line boundaries. As a consequence, searching for the identifier fails.	Protect all identifiers against hyphenation.	
US 05	all	all	ed	The word "object" often copies as "ob ject".		
US 06	various	various	ed	~ (U+007E) is sometimes replaced with ~ (U+223C), causing cut and paste from the standard to fail, notably in 2.14.5.	Use U+007E consistently	
US 07	various	various	ed	' (U+0027) is consistently replaced with ’ (U+2019), causing cut and paste to fail. This is also an issue with the 1998 standard.	Use U+0027 consistently in code samples (i.e. monospace font)	
GB 1	1.1	2	Ed	The C99 standard supports inline functions, so this should not be listed as a distinguishing feature of C++.	strike "inline functions" from the list of C++ 'additional facilities'	

1 **MB** = Member body (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	(3)	4	5	(6)	(7)
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Secretariat observations** on each comment submitted
GB 2	1.2		Ge	In [intro.refs] the spec references ISO/IEC 9945:2003 even though a later revision, ISO/IEC 9945:2008 has already been released: http://www.iso.org/iso/catalogue_detail.htm?csnumber=50516	The section should be updated to reference the latest version. In addition, since POSIX is a registered trademark of the IEEE, the spec should use the registered trademark (R) symbol wherever it references it. Alternatively, it can refer to ISO/IEC 9945.	
ITTF 02	1.2		ed	The introductory text to the Normative references is not correct.	Delete the current introductory text to the Normative references and replace with the following: "The following referenced documents are indispensable for the application of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies."	
ITTF 03	1.3		ed	The title to the subclause does not accurately reflect the content.	Change the title to "Terms and definitions".	
ITTF 04	1.3		ed	3.1 of the ISO/IEC Directives, Part 2, states that the following introductory wording shall be used where all terms and definitions are given the document itself: "For the purposes of this document, the following terms and definitions apply."	Change the introductory text to: "For the purposes of this document, the following terms and definitions apply."	
ITTF 05	1.3		ed	D.1.5.3 of the ISO/IEC Directives, Part 2 states that the form of a definition shall be such that it can replace the term in context.	Delete the definite or indefinite article at the beginning of each definition. Redraft definitions 1.3.11 and 1.3.13 so that they can replace the term in context (i.e. they should not be more than one sentence).	
ITTF 06	1.3		ed	D.3.1 of the ISO/IEC Directives, Part 2, states that the definition shall not be followed by a full stop.	Remove the full stops at the end of the definitions.	

1 **MB** = Member body (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

NOTE Columns 1, 2, 4, 5 are compulsory.

Template for comments and secretariat observations

Date: 5 Oct 2010	Document: **N3141 FCD 14882 Ballot Comments**

1	2	(3)	4	5	(6)	(7)
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Secretariat observations** on each comment submitted
ITTF 07	1.3		ed	D.3.9 of the ISO/IEC Directives, Part 2, provides examples on how to present examples and notes to terms and definitions. The examples and notes to the terms and definitions are not presented in accordance with D.3.9 of the ISO/IEC Directives, Part 2.	Redraft the notes and examples to the terms and definitions in accordance with D.3.9 of the ISO/IEC Directives, Part 2.	
GB 3	1.3	1	Ed	The library stretches out to clause 30 now, and the terms in 17.3 should cover them too.	Update reference to clause 27 to say clause 30.	
JP 15	1.3	1	E	There is a description, "17.3 defines additional terms that are used only in Clauses 17 through 27 and Annex D.", but the terms defined in 17.3 are also used in Clauses 28 through 30, which are added recently. So the scope should be expanded to include them.	17.3 defines additional terms that are used only in Clauses 17 through 30 and Annex D.	
GB 4	1.3.10		Ed	The phrase "catch clause" in the 2003 standard has become "catch Clause"	Change back the "catch clause"	
RU 1	1.7	p.5, line 5 from end	ed	Reference missed	Insert reference "(2.3)" after "basic execution character set"	
GB 5	1.9	3	Ed	The evaluation of function arguments are now indeterminately sequenced, rather than left completely unspecified, as part of the new language describing the memory model. A clearer example of unspecified behavior should be used here.	[Need to identify a better example to propose]	
GB 6	1.9	4	Ed	There are core issues surrounding the undefined behavior of dereferencing a null pointer. It appears the intent is that dereferencing *is* well defined, but using the result of the dereference will yield undefined behavior. This topic is too confused to be the reference example of undefined behavior, or should be stated more precisely if it is to be retained.	[Identify a better example of undefined behavior to propose]	

1 **MB** = Member body (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	(3)	4	5	(6)	(7)
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Secretariat observations** on each comment submitted
CH 2	1.9 and 1.10		te	It's not clear whether relaxed atomic operations are observable behaviour.	Clarify it.	
GB 7	1.9.6	p6	Te	From a naive, first-time reader's point of view, 1.9.6 made me double take, as it seemed it could imply any signal would leave the program in an unspecified state after completing. I believe I understand the intent, but to clarify it, I'd change it to make it clear that the unspecified state applies only for the duration of the signal handler.	Change: 6 When the processing of the abstract machine is interrupted by receipt of a signal, the values of objects which are neither — of type volatile std::sig_atomic_t nor — lock-free atomic objects (29.4) are unspecified, and the value of any object not in either of these two categories that is modified by the handler becomes undefined. to: 6 When the processing of the abstract machine is interrupted by receipt of a signal, the values of objects which are neither — of type volatile std::sig_atomic_t nor — lock-free atomic objects (29.4) are unspecified **for the duration of the signal handler**, and the value of any object not in either of these two categories that is modified by the handler becomes undefined.	
US 08	1.9	footnote 7	te	The footnote "Overloaded operators are never assumed to be associative or commutative." is either meaningless or overly restrictive.	Change the footnote to "Overloaded operators are assumed to be non-associative and non-commutative until proven otherwise.".	
CA 23	1.10, 29	1.10, 29	Te	C1x has added new atomics C1x has added new atomics syntax, and in some cases new semantics and operations. C++0x needs to consider aligning with the new C1x atomics	Add back compatibility between C++0x and C1x atomics	

1 **MB** = Member body (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	(3)	4	5	(6)	(7)
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Secretariat observations** on each comment submitted

CA 14	1.10p4	1.10p4	ed	Initialisation of atomics
				Add the following to 1.10p4:
				[Note: There may be non-atomic writes to atomic
				objects, for example on initialization and re-
				initialization. - end note]
				Rationale: We believe the intent is that for any atomic there is a distinguished initialisation write, but that this need not happens-before all the other operations on that atomic - specifically so that the initialisation write might be non-atomic and hence give rise to a data race, and hence undefined behaviour, in examples such as this (from Hans):
				`atomic< atomic<int> * > p`
				`f()` |
				`{ atomic<int>x;` | `W_na x`
				`p.store(&x,mo_rlx);` | `W_rlx p=&x`
				`}` |
				(where na is nonatomic and rlx is relaxed). We suspect also that no other mixed atomic/nonatomic access to the same location is intended to be permitted. The possibility of non-atomic writes on atomic objects is not mentioned in 1.10, and (before talking with Hans) we didn't realise it was intended, so we suggest adding the note above to clarify things.
					Add the following to 1.10p4:
					[Note: There may be non-atomic
					writes to atomic objects, for example
					on initialization and renitialization. - end note]	

CA 12	1.10p6	1.10p6	te	The use of maximal in the definition of release sequence (proposed edit seems reasonable to Clark) We suggest that 1.10p6 be changed to:	We suggest that 1.10p6 be changed to: A release sequence from a release operation A on an atomic object M is	

1 **MB** = Member body (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	(3)	4	5	(6)	(7)
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Secretariat observations** on each comment submitted

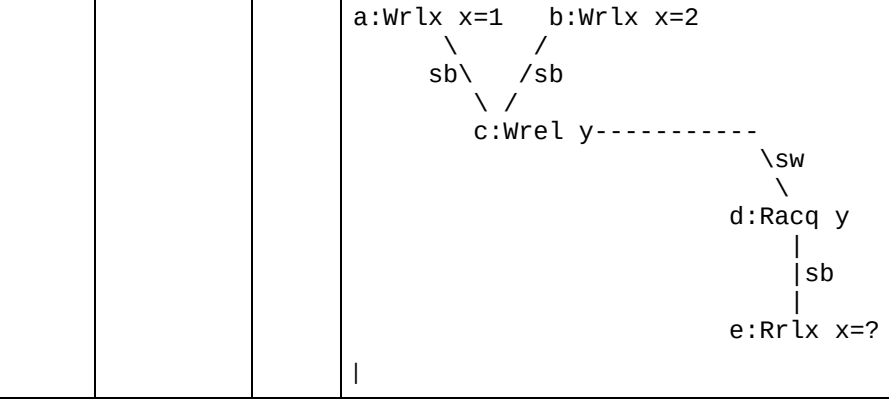
				A release sequence from a release operation A on an atomic object M is a maximal contiguous sub-sequence of side effects in the modification order of M, where the first operation is A, and every subsequent operation
				- is performed by the same thread that performed the release, or
				- is an atomic read-modify-write operation.
				Rationale: The current wording of the standard suggests that release sequences are maximal with respect to sequence inclusion, i.e. that if there are two release operations in the modification order,
				`mod mod`
				`rel1----->rel2----->w`
				then [rel1;rel2;w] is the only release sequence, as the other candidate [rel2;w] is included in it. This interpretation precludes synchronizing with releases which have other releases sequenced-before them. We believe that the intention is actually to define the maximal release sequence from a particular release operation, which would admit both [rel1;rel2;w] and [rel2;w].
					a maximal contiguous sub-sequence of side effects in the modification order of M, where the first operation is A, and every subsequent operation
					- is performed by the same thread that performed the release, or
					- is an atomic read-modify-write operation.	

US 09	1.10	para 4	te	The "operations on locks" do not provide synchronization, as locks are defined in Clause 30.	Change "operations on locks" to "locking operations".	

1 **MB** = Member body (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	(3)	4	5	(6)	(7)
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Secretariat observations** on each comment submitted
CA 20	1.10p1	1.10p1	Te	Reading from the last element in a vsse? Paul wrote: > If every element in a vsse happens-before a given value > computation, then that value computation must return > the value stored by the last element in the vsse. We wrote: We're not sure about that. Consider the following, with two relaxed writes to x on one thread that are not sequenced-before related to each other (eg in different arguments to the same function), but are followed by a release/acquire on a different variable y to another thread that then reads x. We think the final read (e) could read from either (a) or (b), regardless of how (a) and (b) are related in modification order.  	Please clarify.	

1 **MB** = Member body (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	(3)	4	5	(6)	(7)
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Secretariat observations** on each comment submitted
				Paul> In that case IIRC, the standard does not specify Paul> the order, but the code will be generated in some Paul> order, and that arbitrary choice on the part of the Paul> compiler will determine the modification order. We agree that in a normal implementation (eg where the argument evaluations are not spawned off to new threads - is that intended to be forbidden?), the two writes will indeed be ordered according to the generated-code order (and before the release fence), and hardware coherence will ensure that (e) reads the later one. But in the draft standard as written, that execution is allowed - the draft doesn't currently impose that aspect of coherence. To make the example more concrete, if there were another thread with `c --sw--> f:Racq y --sb--> g:Rrlx x` then e and g could read different values. Paul notes: > But 1.10p1 says: > > A thread of execution (also known as a thread) is a > single flow of control within a program, including > the initial invocation of a specific top-level > function, and recursively including every function		

1 **MB** = Member body (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

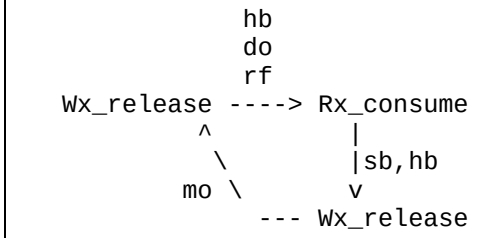
NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	(3)	4	5	(6)	(7)
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Secretariat observations** on each comment submitted
				> invocation subsequently executed by the thread. > > This excludes the possibility of the implementation > spawing off a new thread -unless- the implementation > can make things appear as if there was only one thread. > From this viewpoint, your example shows just how > careful an implementation must be if it is to fully > comply with this as-if rule. We replied >ok, thanks to this, but in fact the situation is still unclear. 1.10p1 does indeed rule out the hypothetical implementation that we mentioned, but even if (a) and (b) would be ordered by any reasonable implementation, in terms of the concepts of the standard, that doesn't introduce a sequenced-before edge between (a) and (b). It seems that Paul is assuming the individual memory accesses in function arguments are indeterminately sequenced rather than unsequenced?		
CA 19	1.10p5 1.10p13	1.10p5 1.10p13	Te	Alternative definition of the value read by an atomic operation Here's an interesting example involving a release/consume pair. We believe that in a direct implementation on hardware, this would be forbidden by	Please clarify.	

1 **MB** = Member body (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	(3)	4	5	(6)	(7)
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Secretariat observations** on each comment submitted
				coherence, but that the current text allows it. We don't know whether it should be allowed or not.  Paul claims this is forbidden by 1.10p5, but we don't see how that can be the case. We don't see much room in 1.10p5 for any other interpretation - it says: - "All modifications to a particular atomic object M occur in some particular total order, called the modification order of M" - "If A and B are modifications of an atomic object M and A happens before (as defined below) B, then A shall precede B in the modification order of M, which is defined below." Both of which seem very clear. The only wiggle room is in the Note - "[Note: This states that the modification orders must respect the "happens before" relationship]" We took that "must respect" to be a gloss rather than to add any additional constraint. Earlier we suggested a change, to the constraint on the value read by an atomic operation, that would forbid this example:		

1 **MB** = Member body (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	(3)	4	5	(6)	(7)
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Secretariat observations** on each comment submitted
				The standard introduces visible side effects, which are used first to define the values read by non-atomic operations. They are then re-used to constrain the value read by atomic operations: 1.10p13 says that an atomic operation must read from somewhere in "the" visible sequence of side effects, which must start from *a* visible side effect, i.e. a side effect that (a) happens before the read, and (b) is not happens-before-hidden. We suspect that this re-use of the notion of visible side effect may be a drafting artifact, in which case one might remove the requirement that there is a vse for atomics, and replacing the first two sentences of 1.10p13 by "An atomic operation must read from somewhere in the modification order that is not happens-before-hidden and does not follow (in modification order) any side effect that happens-after the read." Now we're not sure how this would fit in with initialisation and reading of indeterminate values; we need to think about it more.		
CA 22	1.10p8	1.10p8	Te	Control dependencies for atomics Given the examples of compilers interchanging data and control dependencies, and that control dependencies are architecturally respected on Power/ARM for load->store (and on Power for load->load with a relatively cheap isync), we're not sure why carries-a-dependency-to does not include control dependencies between atomics.	Please clarify.	
CA 15	1.10p9	1.10p9	Ed	Intra-thread dependency-ordered-before The current draft has release/acquire synchronize-with edges only between a release on one thread and an acquire on a *different* thread, whereas the definition of	We suggest changing the definition of dependency-ordered-before in 1.10p9 to the following:	

1 **MB** = Member body (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	(3)	4	5	(6)	(7)
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Secretariat observations** on each comment submitted
				dependency-ordered-before permits the release and consume to be on the same thread; it seems odd to permit the latter. (At the moment function arguments can't race or sync with each other, but they can be dependency ordered before each other.) We don't currently have an example in which this makes a real difference, but for symmetry could suggest changing the definition of dependency-ordered-before in 1.10p9 to the following: An evaluation A is dependency-ordered before an evaluation B if - A performs a release operation on an atomic object M, and on another thread, B performs a consume operation on M and reads a value written by any side effect in the release sequence headed by A, or - for some evaluation X, A is dependency-ordered before X and X carries a dependency to B.	An evaluation A is dependency- ordered before an evaluation B if - A performs a release operation on an atomic object M, and on another thread, B performs a consume operation on M and reads a value written by any side effect in the release sequence headed by A, or - for some evaluation X, A is dependency-ordered before X and X carries a dependency to B.	
CA 11	1.10p12	1.10p12	te	"Subsequent" in vsse definition Remove the word "subsequent" from the definition of visible sequence of side effects in 1.10p12. (as suggested by Hans) Rationale: if every element in a vsse happens-before a read, the read should not take the value of the visible side	Remove the word "subsequent" from the definition of visible sequence of side effects in 1.10p12.	

1 **MB** = Member body (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	(3)	4	5	(6)	(7)
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Secretariat observations** on each comment submitted
				effect.		
CA 17	1.10p12	1.10p12	Ed	1.10p12 phrasing 1.10p12 last note: "...as defined here..." should be "...as defined below...".	1.10p12 last note: "...as defined here..." should be "...as defined below...".	
CA 13	1.10p13	1.10p13	ed	Wording of the read-read coherence condition In 1.10p13 a coherence condition is stated on the values of atomic reads: "Furthermore, if a value computation A of an atomic object M happens before a value computation B of M, and the value computed by A corresponds to the value stored by side effect X, then the value computed by B shall either equal the value computed by A, or be the value stored by side effect Y, where Y follows X in the modification order of M." We suggest that this be replaced with the following: "Furthermore, if a value computation A of an atomic object M happens before a value computation B of M, and A takes its value from the side effect X, then the value computed by B shall either be the value	In 1.10p13 a coherence condition is stated on the values of atomic reads: "Furthermore, if a value computation A of an atomic object M happens before a value computation B of M, and the value computed by A corresponds to the value stored by side effect X, then the value computed by B shall either equal the value computed by A, or be the value stored by side effect Y, where Y follows X in the modification order of M." We suggest that this be replaced with the following:	

1 **MB** = Member body (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	(3)	4	5	(6)	(7)
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Secretariat observations** on each comment submitted
				stored by X, or the value stored by a side effect Y, where Y follows X in the modification order of M." Rationale: The words "corresponds to" are not used elsewhere in the standard, as far as we can see, and it is unclear whether they have a special meaning here. In addition taking the value of the read B from the value read by A seems unnecessarily indirect. B could take its value from X instead.	"Furthermore, if a value computation A of an atomic object M happens before a value computation B of M, and A takes its value from the side effect X, then the value computed by B shall either be the value stored by X, or the value stored by a side effect Y, where Y follows X in the modification order of M."	
CA 18	1.10p13	1.10p13	Te	Non-unique visible sequences of side effects and happens-before ordering In 1.10p13, replace "The visible sequence of side effects on..." by "A visible sequence of side effects on..." and "in the visible sequence of M with respect to B" by "in a visible sequence of M with respect to B" Rationale: the current standard allows multiple visible sequences of side effects (vsse's) for a given read (despite the use of "The" at the start of 1.10p13). We	In 1.10p13, replace "The visible sequence of side effects on..." by "A visible sequence of side effects on..." and "in the visible sequence of M with respect to B"	

1 **MB** = Member body (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	(3)	4	5	(6)	(7)
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Secretariat observations** on each comment submitted
				demonstrate this by constructing an execution with two vsse's. The following execution has five memory operations, four of which are read modify writes (RMW's). There are two threads, one with four operations each ordered by sequenced before (sb), the other with a single RMW release. RMW1 +---RMW3_release | / |sb do/ v / R_consume<---+ | |sb v RMW2 | |sb v RMW4 The modification order in this example is as follows: mod mod mod RMW1----->RMW2----->RMW3_release----->RMW4 With the modification order we give above, the happens-before relation has exactly these edges, according to 1.10p10: From sequenced-before: RMW1 -> R_consume, RMW2, RMW4 R_consume -> RMW2, RMW4 RMW2 -> RMW4 From ithb: From dependency-ordered-before: RMW3_release -> R_consume In particular, there are no edges	by "in a visible sequence of M with respect to B"	

1 **MB** = Member body (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	(3)	4	5	(6)	(7)
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Secretariat observations** on each comment submitted

				`RMW3_release -> RMW2 or RMW4.`
				
				`As we understand it, this is the intended absence of transitivity from dependency-ordered-before to sequenced-before.`
				
				1.10p5 says that if A happens-before B then A precedes B in the modification order, which is true for all the happens-before edges and the modification order above.
				
				RMW1 and RMW3_release are visible side effects
				
				RMW2 and RMW4 follow R_consume in happens-before, so cannot be in a visible sequence of side effects.
				
				Hence the two visible sequences of side effects are [RMW1] and [RMW3].
				
				The R_consume here must read from the later vsse in modification order for the dependency_ordered edge to exist. The existence of two vsse's relies on the lack of transitivity of happens before (which only occurs in the presence of consume operations).		

US 10	1.10	Paragraph 14	te	The definition of a data race does not take into account two overlapping atomic operations	Augment the first sentence: The execution of a program contains a data race if it contains two conflicting actions in different threads, at least one of which is not atomic (or both are atomic and operate on overlapping, but not-identical, memory locations), and neither happens before the other.	
US 11	1.10	para7	te	There is some confusion between locks and mutexes.	Change "lock" when used as a noun to "mutex".	
US 12	1.10	P4,p14, p6,p12,p13	te	Adapt N3074: http://www.open-	Proposed change in N3074: http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2010/n3074.ht	

1 **MB** = Member body (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	(3)	4	5	(6)	(7)
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Secretariat observations** on each comment submitted
				std.org/jtc1/sc22/wg21/docs/papers/2010/n3074.html	ml	
CA 2	various	various	variou s	Canada agrees with US 12, 14, 142, 145, 159	Resolve as suggested in these comments	
GB 8	1.10	4, 7	Te	The text says that the library "provides ... operations on locks". It should say "operations on mutexes", since it is the mutexes that provide the synchronization. A lock is just an abstract concept (though the library types unique_lock and lock_guard model ownership of locks) and as such cannot have operations performed on it. This mistake is carried through in the notes in that paragraph and in 1.10p7	Change 1.10p4 as follows: "The library defines a number of atomic operations (Clause 29) and operations on mutexes (Clause 30) that are specially identified as synchronization operations. These operations play a special role in making assignments in one thread visible to another. A synchronization operation on one or more memory locations is either a consume operation, an acquire operation, a release operation, or both an acquire and release operation. A synchronization operation without an associated memory location is a fence and can be either an acquire fence, a release fence, or both an acquire and release fence. In addition, there are relaxed atomic operations, which are not synchronization operations, and atomic read-modify-write operations, which have special characteristics. [Note: For example, a call that acquires a lock on a mutex will perform an acquire operation on the locations comprising the mutex. Correspondingly, a call that releases the same lock will perform a release operation on those same locations. Informally, performing a release operation on A forces prior side effects on other memory locations to become visible to other threads that later perform a consume or an acquire operation on A. “Relaxed” atomic operations are not synchronization operations even though, like synchronization operations, they cannot contribute to data races. — end note]"	

1 **MB** = Member body (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	(3)	4	5	(6)	(7)
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Secretariat observations** on each comment submitted
					Change 1.10p7 as follows: "Certain library calls synchronize with other library calls performed by another thread. In particular, an atomic operation A that performs a release operation on an atomic object M synchronizes with an atomic operation B that performs an acquire operation on M and reads a value written by any side effect in the release sequence headed by A. [Note: Except in the specified cases, reading a later value does not necessarily ensure visibility as described below. Such a requirement would sometimes interfere with efficient implementation. — end note] [Note: The specifications of the synchronization operations define when one reads the value written by another. For atomic objects, the definition is clear. All operations on a given mutex occur in a single total order. Each lock acquisition “reads the value written” by the last lock release on the same mutex. — end note]"	
GB 9	1.10	6	Te	See (B) in attachment Appendix 1 - Additional Details	Request the concurrency working group to determine if changes are needed	
GB 10	1.10	10	Te	See (C) in attachment Appendix 1 - Additional Details The GB would like WG21 to confirm there is no issue related to this. GB adds: We agree that if the read from x reads the value written by the write to x the write to x inter-thread-happens-before the write to y. However, the read from y is sequenced before the write to x, so if the read from x reads the value written by the write to x, then the read from y also inter-thread-happens-before the write to y.	Request the concurrency working group to determine if changes are needed	

1 **MB** = Member body (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	(3)	4	5	(6)	(7)
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Secretariat observations** on each comment submitted
				Consequently, the read from y cannot see the value written by the write to y. The reciprocal ordering also applies, but they cannot both apply in the same execution since if the write to x happens-before the read from x then the read from y happens-before the write to y, and vice-versa. There is thus no contradiction. [see comment below for proper formatting]		
CA 8	1.10p10	1.10p10	te	Rationale: Without this the standard permits executions with a cyclic happens-before relation that it seems clear should be forbidden, e.g. Rx_consume<--+ +-->Ry_consume | rf\ /rf | |sb X |sb v / \ v Wy_release---+ +---Wx_release One could instead impose acyclicity on happens-before; that would be equivalent.	1.10p10, before the Note, add: "The inter-thread happens-before relation of an execution must be acyclic"	
GB 11	1.10	12	Te	See (E) in attachment Appendix 1 - Additional Details The GB would like WG21 to confirm there is no issue related to this. GB adds: [see comment below for proper formatting] The variable in question has a single modification order, which is any of (a) RMW3, RMW1, RMW2, RMW4.	Request the concurrency working group to determine if changes are needed	

1 **MB** = Member body (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	(3)	4	5	(6)	(7)
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Secretariat observations** on each comment submitted
				(b) RMW1, RMW3, RMW2, RMW4. (c) RMW1, RMW2, RMW3, RMW4. (d) RMW1, RMW2, RMW4, RMW3. since RMW1, RMW2 and RMW4 occur in a single thread in that sequence, and RMW3 occurs in a separate thread with no other ordering constraints. Since the R_consume lies between RMW1 and RMW2 in that thread, it must either read the value written by RMW1 (which could happen if it immediately follows RMW1 in any of the sequences), or RMW3 (which could happen with sequence (b)). The visible sequence of side effects for R_consume is thus either RMW3, RMW1 (from (a)), RMW1 (from (b), (c) or (d)), or RMW1, RMW3 (from (b)). Which sequence applies in practice may vary from execution to execution. There is however only a single sequence on any given execution.		
GB 12	1.10	13	Te	See (F) in attachment Appendix 1 - Additional Details The GB would like WG21 to confirm there is no issue related to this. GB adds: The cycle given is clearly forbidden by the current text. The read is sequenced-before the write in the same thread. If the read sees the value written by the other thread then that write is dependency-ordered-before the read, and thus happens-before the read, and happens-before the write from the reading thread. The write from the left-hand thread thus must occur before the write from the right-hand thread in the modification order of the object by 1.10p5.	Request the concurrency working group to determine if changes are needed	

1 **MB** = Member body (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

NOTE Columns 1, 2, 4, 5 are compulsory.

Template for comments and secretariat observations

Date: 5 Oct 2010	Document: **N3141 FCD 14882 Ballot Comments**

1	2	(3)	4	5	(6)	(7)
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Secretariat observations** on each comment submitted
GB 13	1.10	13	Te	See (G) in attachment Appendix 1 - Additional Details GB suggests alternative wording to that in the attached paper: "Furthermore, if a value computation A of an atomic object M happens before a value computation B of M, and A uses the value of M from the side effect X, then the value computed by B shall either be the value stored by X, or the value stored by a side effect Y, where Y follows X in the modification order of M."	Request the concurrency working group to determine if changes are needed	
GB 14	1.10	8	Te	See (I) in attachment Appendix 1 - Additional Details GB adds: If an implementation can't guarantee the ordering it should refrain from performing the optimisation	Request the concurrency working group to determine if changes are needed.	
GB 15	1.10		Te	See (J) in attachment Appendix 1 - Additional Details	Request the concurrency working group to determine if changes are needed.	
GB 16	1.10	12	Ed	See (L) in attachment Appendix 1 - Additional Details	"...as defined here..." should be "...as defined below...".	
US 13	2.2	1	te	“Raw” strings are still only Pittsburgh-rare strings: the reversion in phase 3 only applies to an r-char-sequence.	Make the reversion apply to the entire raw-string.	
US 14	2.2 2.3 2.5 2.14.5	P1	te	Precedence of reversal and tokenization The current paper implies that determination of the characters forming an *r-char-sequence* occurs while the transformations done in phase 1 and phase 2 are still in effect. Consider these cases: • Line splicing occurred in translation phase 2; the backslash is not there on entry to phase 3 when	**In 2.14.5 [lex.string] paragraph 2:** Remove footnote 24: **In 2.2 [lex.phases] paragraph 1, phase 1; insert exception:** Physical source file characters are mapped, in an implementation-defined manner, to the basic source character set (introducing new-line characters for end-of-line indicators) if necessary. The	

1 **MB** = Member body (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	(3)	4	5	(6)	(7)
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Secretariat observations** on each comment submitted
				we try to tokenize: const char str[] = R"a()\a")a"; • Trigraph replacement occurred in phase 1. The right parenthesis is not there on entry to phase 3: const char str[] = R"(??)"; • Trigraph replacement (again). In [lex.string] paragraph 2, there is a footnote 24 in N3092. Note that this provides fuel for anti-trigraph sentiment: const char str[] = R"#()??=")#"; **Change in [lex.string] from N3077:** Escape sequences and universal-character-names in non-raw string literals have the same meaning as in character literals .... should be reflected in [lex.phases] paragraph 1, phase 5 (CD2 wording): Each source character set member and universal-character-name in a character literal or a string literal, as well as each escape sequence in a character literal or a non-raw string literal, is converted to the corresponding member of the execution character set (2.14.3, 2.14.5); if there is no corresponding member, it is converted to an implementation-defined member other than the null (wide) character and [lex.charset] paragraph 2 (CD2 wording): Additionally, if the hexadecimal value for a universal-character-name outside the *c-char-sequence*, *s-char-sequence*, or *r-char-sequence* of a character or string	set of physical source file characters accepted is implementation-defined. Trigraph sequences (2.4) are replaced by corresponding single-character internal representations. Any source file character not in the basic source character set (2.3) is replaced by the universal-character-name that designates that character. (An implementation may use any internal encoding, so long as an actual extended character encountered in the source file, and the same extended character expressed in the source file as a universal-character-name (i.e., using the \uXXXX notation), are handled equivalently except where this replacement is reverted.) **In 2.2 [lex.phases] paragraph 1, phase 3:** The source file is decomposed into preprocessing tokens (2.5) and sequences of white-space characters (including comments). A source file shall not end in a partial preprocessing token or in a partial comment. Each comment is replaced by one space character. New-line characters are retained. Whether each nonempty sequence of white-space characters other than new-line is retained or replaced by one space character is unspecified. The process of dividing a source file's characters into preprocessing tokens is context-dependent. [*Example:* see the handling of < within a #include preprocessing directive. —*end example*]	

1 **MB** = Member body (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	(3)	4	5	(6)	(7)
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Secretariat observations** on each comment submitted
				literal corresponds to a control character (in either of the ranges 0x00–0x1F or 0x7F–0x9F, both inclusive) or to a character in the basic source character set, the program is ill-formed. UCNs simply do not occur in the grammar for r-char-sequence anyway.	**In 2.2 [lex.phases] paragraph 1, phase 5:** Each source character set member in a character literal or a string literal, as well as each escape sequence and universal-character-name in a character literal or a non-raw string literal, is converted to the corresponding member of the execution character set (2.14.3, 2.14.5); if there is no corresponding member, it is converted to an implementation-defined member other than the null (wide) character. **In 2.3 [lex.charset] paragraph 2:** .... Additionally, if the hexadecimal value for a universal-character-name outside the *c-char-sequence* or *s-char-sequence* of a character or string literal corresponds to a control character (in either of the ranges 0x00–0x1F or 0x7F–0x9F, both inclusive) or to a character in the basic source character set, the program is ill-formed. [Footnote: A sequence of characters resembling a universal-character-name in an *r-char-sequence* (2.14.5 [lex.string]) does not form a universal-character-name.] **In 2.5 [lex.pptoken] paragraph 3:** If the input stream has been parsed into preprocessing tokens up to a given character: • if the next character begins a sequence of characters that could be the prefix and initial double quote of a raw string	

1 **MB** = Member body (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	(3)	4	5	(6)	(7)
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Secretariat observations** on each comment submitted
					literal, such as R", the next preprocessing token shall be a raw string literal and any transformations performed in phases 1 and 2 on this input stream (trigraphs, universal-character-names, and line splicing) are reverted for the remainder of the stream until said raw string literal (2.14.5) is matched; [Footnote: A raw string literal formed through token concatenation (16.3.3) is not parsed from an input stream and is not subject to this reverting. Destringization (16.9) involves an alternate input stream, thus there are no phase 1 or phase 2 transformations to revert.] • otherwise, the next preprocessing token is the longest sequence of characters that could constitute a preprocessing token, even if that would cause further lexical analysis to fail.	
US 15	2.6	para 2	te	The <: digraph causes problem with users unfamiliar with digraphs when passing global objects as template arguments.	Add a special hack for <:: much like the special hack for >>.	
CA 24	2.11	Various	Te	A list of issues related TR 10176:2003 1) "Combining characters should not appear as the first	Please clarify.	

1 **MB** = Member body (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	(3)	4	5	(6)	(7)
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Secretariat observations** on each comment submitted
				character of an identifier." Reference: ISO/IEC TR 10176:2003 (Annex A) This is not reflected in FCD. 2) Restrictions on the first character of an identifier are not observed as recommended in TR 10176:2003. The inclusion of digits (outside of those in the basic character set) under *identifer-nondigit* is implied by FCD. 3) It is implied that only the "main listing" from Annex A is included for C++. That is, the list ends with the Special Characters section. This is not made explicit in FCD. Existing practice in C++03 as well as WG 14 (C, as of N1425) and WG 4 (COBOL, as of N4315) is to include a list in a normative Annex. 4) Specify width sensitivity as implied by C++03: \uFF21 is not the same as A Case sensitivity is already stated in [lex.name].		
GB 17	2.14.2	Table 5	Ed	[lex.icon] 2.14.2/2 Table 5 - 'Types of integer constants' In the penultimate row for this table (for suffix `ll or LL') it gives the `Octal or hexadecimal constant' in the third column as one of: long long int unsigned long int Unless I am misunderstanding something fundamental, this second should be: unsigned long long int	Replace the entry for "ll or LL" and "Octal or hexadecimal constant" in table 5 with "long long int unsigned long long int"	
JP 16	2.14.3	2 Note	E	Typo, "wide-charater" should be "wide-character".	Correct typo. [Note: the type wchar_t is able to represent all members of the execution wide-character set (see	

1 **MB** = Member body (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	(3)	4	5	(6)	(7)
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Secretariat observations** on each comment submitted
					3.9.1).	
RU 2	2.14.3	p.23, par.3, line 1	ed	Reference missed	Insert reference "(3.9.1)" after "extended integer type"	
DE 2	2.14.4		te	C++ does not support hexadecimal floating-point literals, although they are useful to specify exact floating-point constants.	Consider supporting the C99 syntax for hexadecimal floating-point literals.	
US 16	2.14.5 [lex.string]		ge	Raw string literals have no implementation experience.	Either demonstrate a complete implementation of this feature or remove N2146 from the working paper prior the FDIS.	
DE 3	2.14.7		te	It is not sufficiently clear that std::nullptr_t is a distinct type and neither a pointer type nor a pointer-to-member type.	Add a note in 2.14.7 stating that, preferably with cross-references to the normative statements in 3.9.	
RU 5	2.14.7	p. 28	ed	Page layout bug	Move footnote 24 from page 28 to page 27	

US 17	2.14.8	6	te	In general, the parameter type of a literal operator must be the same as the argument passed to it. That is not the case for a *user-defined-character-literal*, where the argument could inadvertently match a literal operator intended for use with *user-defined-integer-literal*s:
				typedef unsigned long long ULL;
				int operator "" X(ULL);
				int i = 'c'X; // operator"" X(ULL('c'))
					Add the following phrase to the description in paragraph 6:
					S shall contain a literal operator whose parameter type is the same as the type of *ch*.	

JP 17	2.14.8	3	E	Typo, missing ",". If S contains a raw literal operator the literal L is treated as	Correct typo. If S contains a raw literal operator, the literal L is treated as	
JP 18	2.14.8	4	E	Typo, missing ",". If S contains a raw literal operator the literal L is treated as	Correct typo. If S contains a raw literal operator, the literal L is treated as	

1 **MB** = Member body (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	(3)	4	5	(6)	(7)
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Secretariat observations** on each comment submitted
US 18	2.24.8 [lex.ext]		ge	User-defined literals have no implementation experience.	Either demonstrate a complete implementation of this feature or remove N2750 from the working paper prior the FDIS.	
US 19	3	4	te	It is not always clear when the term "use" is intended as a reference to the definition in 3.2 and when it has its normal English meaning. For example, 3 paragraph 4 reads, "A *name* is a use of an identifier..."	Replace all occurrences of the word "use" that are not intended as references to 3.2 with some other term, such as "occurrence" or "appearance" or "reference to".	
US 20	3.1	para 1 bullet 4	ed	Grammatical number mismatch in "an assignment expressions".		
US 21	3.1	2	ed	using N::d; does not declare N::d.	using N::d; // declares d	
US 22	3.2	4	te	The type of the *expression* of a *decltype-specifier* is apparently required to be complete.	Make an exception so that a template specialization type is not instantiated merely because it's the type of the *expression* in decltype(*expression*)	
JP 19	3.2	4	E	Representations of reference link are not unified. Most reference links to clause (table) number, say X, are in the form "Clause X" ("Table X") capitalized, and subsection Y.Y.Y is referenced with its number only in the form "Y.Y.Y". Whether they are parenthesized or not depends on the context. However there are some notations "(Z)" consisting of only a number Z in parentheses to confer Clause or Table number Z.	Change "(10)" to "(Clause 10)".	
JP 20	3.3.2	7	E	Representations of reference link are not unified. Most reference links to clause (table) number, say X, are in the form "Clause X" ("Table X") capitalized, and subsection Y.Y.Y is referenced with its number only in the form "Y.Y.Y". Whether they are parenthesized or not depends on the context. However there are some notations "(Z)" consisting of only	Change "(9)" to "(Clause 9)".	

1 **MB** = Member body (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	(3)	4	5	(6)	(7)
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Secretariat observations** on each comment submitted
				a number Z in parentheses to confer Clause or Table number Z.		
US 23	3.4.5	para 1	te	Global class templates should not hide member templates.	Strike the end of para 1 starting with "If the lookup in the class of the object expression finds a template,". See Appendix 1 - Additional Details	
US 24	3.5	3	te	One of the critieria for giving a name internal linkage is "a variable that is explicitly declared const and neither explicitly declared extern nor previously declared to have external linkage." This should presumably apply to variables declared constexpr as well.	Add parallel wording for the constexpr specifier.	

DE 4	3.5		te	It is odd that "N" has no linkage and "g" has external linkage in this example:
				namespace {
				namespace N // has no linkage
				{
				void g(); // has external linkage
				} }		

DE 5	3.7.3		te	The term "local" was changed globally to "block-scope", but this section still contains the term "local" (see also core issue 642).	Change "local" to "block-scope" in the first paragraph.	
RU 3	3.7.4.3	p.65, line 7	ed	Reference missed	Insert reference "(5.7)" after "well-define pointer arithmetic"	
RU 4	3.7.4.3	p. 65, line 8	ed	Reference missed	Insert references "(4.10, 5.4)" after "well-define pointer conversion"	
GB 18	3.8	9	Te	It isn't clear that the comment in the example actually reflects the result of the placement new. If the intended placement operator new is supposed to be the one given by the standard library ,by including , the example is ill-formed as the placement-new expression &b is const B* which doesn't implicitly convert to void*.	Replace: new (&b) const B; With: new (const_cast<B*>(&b)) const B;	

1 **MB** = Member body (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	(3)	4	5	(6)	(7)
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Secretariat observations** on each comment submitted
US 25	3.11		Te	C/C++ compatibility problems defined in WG21/N3093.	Make the changes proposed in WG21/N3093	
US 26	3.7.4, 5.3.5, 12.5, 17.6.3.6, 18.6		te	Programmers may define a static member function operator delete that takes a size parameter indicating the size of the object to be deleted. The equivalent global operator delete is not available. This omission has unfortunate performance consequences.	Permit implementations and programmers to define sized versions of the global operator delete for use in preference to the unsized version. See Appendix 1 - Additional Details	

US 27	3.8	4	te	Related to core issue 1027, consider:
				`int f() {`
				`  union U { double d; } u1, u2;`
				`  (int&)u1.d = 1;`
				`  u2 = u1;`
				`  return (int&)u2.d;`
				`}`
				Does this involve undefined behavior? 3.8/4 seems to say that it's OK to clobber u1 with an int object. Then union assignment copies the object representation, possibly creating an int object in u2 and making the return statement well-defined. If this is well-defined, compilers are significantly limited in the assumptions they can make about type aliasing. On the other hand, the variant where U has an array of unsigned char member must be well-defined in order to support std::aligned_storage.
					Clarify that this testcase is undefined, but that adding an array of unsigned char to union U would make it well-defined--if a storage location is allocated with a particular type, it should be undefined to create an object in that storage if it would be undefined to access the stored value of the object through the allocated type.	

US 28	4.4	para 3	te	A const member function pointer could safely be applied to a non-const object without violating const correctness.	Add an implicit conversion. See Appendix 1 - Additional Details	
FI 7	4.11 [conv.mem], 5.2.9 [expr.static.cast]		te	The CD1 comment CH1 should be reconsidered. The request for being able to cast a pointer to member to a pointer to a base class (or any other implicitly convertible type) of the member is a bugfix rather than an extension. It's a safe conversion, thus it should be allowed. There are valid use cases for such conversions that are currently forbidden.	The standard should allow implicit conversions from “pointer to member of T of type cv D'' to “pointer to member of T of type cv B'', where D is of class type and B is a public base of D, It should allow explicit conversion the other way around.	

1 **MB** = Member body (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	(3)	4	5	(6)	(7)
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Secretariat observations** on each comment submitted
CH 3	4.11 and 5.2.9		te	With respect to the target type, pointer to members should behave like normal pointers. The current situation creates an inconsistency in the C++ type system and is therefore a defect in the Standard.	The standard should allow implicit conversions from ``pointer to member of T of type cv D'' to ``pointer to member of T of type cv B'', where D is of class type and B is a public base of D. It should allow explicit conversion in the other direction.	
JP 21	4.13	1	E	Representations of reference link are not unified. Most reference links to clause (table) number, say X, are in the form "Clause X" ("Table X") capitalized, and subsection Y.Y.Y is referenced with its number only in the form "Y.Y.Y". Whether they are parenthesized or not depends on the context. However there are some notations "(Z)" consisting of only a number Z in parentheses to confer Clause or Table number Z.	Change "(5)" to "(Clause 5)".	
JP 1	5	Paragraph 6	TL	The first half of the Note(before "In general") indicates that the expression "E1.E2" produces xvalue if E1 is xvalue regardless of E2's type. It will be true even if E2 is of reference type. On the other hand, according to 5.2.5 paragraph 4, if E2 is of reference type, the result of "E1.E2" is lvalue regardless of E1's type. These two descriptions contradict each other. As 5.2.5 paragraph 4 seems correct, 5 paragraph 6 should be corrected.	Modify 5 paragraph 6 so that the result of E1.E2 is lvalue instead of xvalue when E2 is of reference type.	
FI 8	5.1.2 [expr.prim.lambda]		te	As requested in JP 9 on CD, capturing by moving should be allowed for lambdas. Roshan Naik presents a very compelling use case in the Core Reflector message c++std-core-16341.	Allow specifying capture by move.	
CH 4	5.1.2	p1	ed	Document N3067 changed the position of attribute specifiers in various places. However, it left out lambda expressions as an oversight, so that the position of attribute-specifier opt in a lambda-declarator is inconsistent with a function declarator	change the rule for lambda-declarator to *lambda-declarator:* **(** *parameter-declaration-clause* **)** **mutable**$_{opt}$ *exception-specification*$_{opt}$ *attribute-specifier*$_{opt}$ *trailing-return-type*$_{opt}$	
CH 5	5.1.2	p4 first bullet	ed	typo	Change second 'if' to 'is'.	

1 **MB** = Member body (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	(3)	4	5	(6)	(7)
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Secretariat observations** on each comment submitted
US 29	5.1.2	5	te	default arguments should be allowed in lambdas (core issue 974)	See Appendix 1 - Additional Details	
US 30	5.1.2	4	te	lambda return type deduction should allow arbitrary function structure (core issue 975)	See Appendix 1 - Additional Details	
JP 22	5.1.2	7	E	Representations of reference link are not unified. Most reference links to clause (table) number, say X, are in the form "Clause X" ("Table X") capitalized, and subsection Y.Y.Y is referenced with its number only in the form "Y.Y.Y". Whether they are parenthesized or not depends on the context. However there are some notations "(Z)" consisting of only a number Z in parentheses to confer Clause or Table number Z.	Change "(5)" to "(Clause 5)".	
CH 6	5.1.2	p8 and p10	te	The current capturing rules seem too restrictive.	Consider to make those rules less restrictive.	
GB 19	5.1.2	16	Ed	[expr.prim.lambda] 5.1.2/16 has text which begins "If a lambda-expression m1 captures an entity and that entity is captured by an immediately enclosing lambda expression m2..." - that is, it describes a situation with m2 enclosing m1, and then describes the capture transformation in these terms. The example given to support this, however, turns this all around and shows m1 enclosing m2. This doesn't make either the text or the example incorrect in any sense, but I would suggest that it adds a level of confusion that is easily avoided.	All references to m1 from the beginning of 5.1.2/16 up to the last occurrence before '[Example ' to be replaced by m2, and vice versa. Rationale for suggested wording: all other examples that use the 'mN' notation for lambda expressions and which involve nesting apply increasing N (by 1, from 1) to indicate increasing nesting depth.	
GB 20	5.1.2	12	Ed	[expr.prim.lambda] 5.1.2/12. In the example code given the local struct s1 has a member function with signature int s1::work(int n) whose definition does not include an appropriate return statement; neither does it include the conventional "// ..." to indicate that the example is intended to be incomplete.	Suggested change: change the signature of this member function to void s1::work(int n), as the return of int does not contribute to the example.	
GB	5.1.2		Te	A lambda-capture can be &, which indicates it captures	Add "= identifier" to the grammar in 5.1.2p1. (The	

1 **MB** = Member body (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	(3)	4	5	(6)	(7)
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Secretariat observations** on each comment submitted
21				everything by reference (unless otherwise specified), or & /identifier/, which indicates it captures the /identifier/ by reference. It can also be =, to capture everything by value, or /identifier/, for a single thing. Why is = /identifier/ not allowed, for consistency?	wording already covers the semantics of this, since it refers to captures that "are preceded by &" or "do not contain &")	
FI 19	5.1.2 [expr.prim.lambda]	21	te	"When the lambda-expression is evaluated, the entities that are captured by copy are used to direct-initialize each corresponding non-static data member of the resulting closure object. " This apparently means that if the capture-default is to copy, entities captured by default, implicitly, are copied even in cases where the copy constructors of such entities are explicit.	Don't implicitly copy entities that have explicit copy constructors declared. Require that such entities be captured explicitly, by enumerating them in the capture list. This seems related to Core Issue 1020, so I'd like that issue to be resolved as well. See Appendix 1 - Additional Details	
GB 23	5.2.2	4	Ed	Order of initialization of arguments in a function is fundamental to the memory model of C++, and the obvious place to look for the definition is in the clause defining function call operators - which currently says nothing. The rules covering this are buried in paragraph 15 of [1.9]. A cross-reference to these words would be most helpful. In particular, it should be made clear that such initialization is indeterminately sequenced (and not unsequenced.)	Add a non-normative note with cross-reference after the first sentance of paragraph 4: "[Note - such initializations are indeterminately sequenced with respect to each other [1.9] - end note]"	
US 31	5.2.9;7.2	10	te	it is unclear from the text in 7.2 and 5.2.9 that the "values of the enumeration" term does not exclude a prvalue of an enumeration type from having other values representable in its underlying type (c++std-core-15652).	clarify this. "The value is unchanged if it is in the range of ~~enumeration~~ values of the enumeration ~~type~~; otherwise the resulting ~~enumeration~~ value is unspecified **(and might not be in that range)**." Also add a note after paragraph 7 "[Footnote: this set of values is used to define promotion and conversion semantics for the enumeration type; it does not exclude an expression of enumeration type from having a value that falls outside this range.]"	
US	5.2.5	5	te	The description of ambiguity ("...if the class of which E2 is	Change the wording to apply also to the case	

1 **MB** = Member body (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	(3)	4	5	(6)	(7)
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Secretariat observations** on each comment submitted

32				directly a member is an ambiguous base (10.2) of the naming class (11.2) of E2") does not cover the following case:
				`struct A { int i; };`
				`struct B: A { };`
				`struct C: A, B { };`
				`void f(C* p) {`
				`p->i; // Should be ambiguous`
				`}`
					when the naming class is an ambiguous base of the class of the object expression.	

JP 64	5.2.8	5	E	In some code examples, ellipsis(…) is used in ill-formed. In these cases, "…" represents omission of some codes like this: class A { /* ... */ } ; But in some cases, it is used without commented-out as below: class A { ... } ; It is an inconsistent usage. They all should be enclosed in a comment.	Change to: class D { /* ... */ };	
GB 22	5.2.10		Te	It is not legal to use reinterpret_cast<> with pointers to void.	Here's an additional paragraph to add to §5.2.10 that would fix this: * A pointer to an object type can be explicitly converted to a pointer to void, and vice versa.[1] The result of such a pointer conversion will have the same result as the standard pointer conversion described in §4.10. A value of type "pointer to object" converted to "pointer to void" and back, possibly with different cv-qualification, shall have its original value. [1] The types may have different cv-qualifiers, subject to the overall restriction that a reinterpret_cast cannot cast away constness.	
US 33	5.3.1	3	te	The resolution of issue 983 added an error for finding the named member in an ambiguous base. This is an unnecessary special case, since the value of the expression is pointer to member of base. If this value is	Revert the change for issue 983 (and in the issues list, add a link to issue 203).	

1 **MB** = Member body (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	(3)	4	5	(6)	(7)
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Secretariat observations** on each comment submitted
				then converted to pointer to member of derived, an error will be given at that point.		
GB 24	5.3.3	6	Te	The return type of the sizeof operator is defined as being of type std::size_t, defined in library clause 18.2. This, in turn, says that size_t is defined in the C standard, which in turn says that size_t is defined as the type of the result of the sizeof operator! The C definition of sizeof returns an implementation-defined unsigned integer type, recommended not to have "an integer conversion rank greater than signed long int, unless the implementation supports objects large enough to make this necessary."	The result type of the sizeof operator should explicitly be implementation defined in clause 5.3.3.	
US 34	5.3.4, 5.3.5		te	Allocations functions are missing happens-before requirements and guarantees.	Add requirements. See Appendix 1 - Additional Details	
US 35	5.3.7 [expr.unary.noexcept], 15.4 [except.spec]		ge	noexcept has no implementation experience.	Either demonstrate a complete implementation of this feature or remove N3050 from the working paper prior the FDIS.	
FI 17	5.3.7 [expr.unary.noexcept]		te	Destructors should by default be noexcept. Such a rule should, I think, be obeyed even for cases where a destructor is defaulted. Then a throwing destructor would need to be declared noexcept(false), and I think the resulting code breakage is acceptable.	Clarify the implicit generation rules and defaulting rules so that destructors are noexcept unless explicitly declared noexcept(false).	
JP 2	5.5	6	TL	Should be corrected because it contradicts with rules in 5.2.5 paragraph 4.	Add the condition that a type of e2 is not a reference type	
US 36	5.19 [expr.const]		ge	Generalized constant expressions have no implementation experience.	Either demonstrate a complete implementation of this feature or remove N2235 from the working paper prior the FDIS.	
DE 7	5.19	4, note	te	The note in paragraph 4 alludes to the possibility that compile-time and run-time evaluations of floating-point expressions might yield different results. There is no clear normative statement (other than the absence of a restriction) that gives such permission.	Move the second sentence of the note into normative text.	

1 **MB** = Member body (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	(3)	4	5	(6)	(7)
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Secretariat observations** on each comment submitted
DE 8	5.19	6	te	In the definition of "potential constant expression" in paragraph 6, it is unclear how "arbitrary" the substitution of the function parameters is. Does it mean "there exists a value for which the result is a constant expression" or does it mean "for all possible values, the result needs to be a constant expression"? Example: constexpr int f(int x){return x + 1; } is a constant expression under the first interpretation, but not under the second (because overflow occurs for x == INT_MAX, cf. 5.19p2 bullet 5). The answer also affects expressions such as: constexpr int f2(bool v) { return v ? throw 0 : 0; } constexpr int f3(bool v) { return v && (throw 0, 0); }		

GB 25	5.19		Te	In trying to pin down the behaviour of constexpr functions, it became apparent that there is disagreement over whether or not the following example is well-formed.
				
				constexpr int f() { return 42 + 84; }
				const int sz = f();
				int a[sz];
				
				This should have the same effect as
				
				const int sz = 42 + 84;
				int a[sz];
				
				otherwise constexpr functions are broken.
					Update the wording in 5.19 to make it clear that both the examples are valid.	

GB 26	5.19		Te	It is not clear how overload resolution applies within a constexpr function. In particular, if an expression in the function body yields an integral value of 0 for some parameter values, and not for others, is it usable as a null pointer constant when evaluated as a constant expression. typedef char (&One)[1];	Updated 5.19 to make it clear that overload resolution in a constexpr function is not dependent on the context of use, or the value of the arguments.	

1 **MB** = Member body (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	(3)	4	5	(6)	(7)
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Secretariat observations** on each comment submitted

				typedef char (&Two)[2];
				One f(void*); // #1
				Two f(...); // #2
				
				constexpr int test(int n) { return sizeof f(n); }
				constexpr int test2(int n) { return sizeof f(n*0); }
				
				int q = 0;
				
				#include
				
				int main() {
				char a[test(0)];
				std::cout << sizeof(a) << std::endl; // #3
				std::cout << test(q) << std::endl; // #4
				
				char b[test2(0)];
				std::cout << sizeof(b) << std::endl; // #5
				std::cout << test2(q) << std::endl; // #6
				}
				
				#3 and #4 should print 2, since n is not an integral constant expression with value 0 in the body of test() --- though it is a constant expression when test() is evaluated as a constant expression, it's value is dependent on the invocation. Permitting different results of overload resolution within the same function body in different calling contexts would violate ODR.
				
				On the other hand, in test2(), the answer is no longer dependent on the value of n, since "n*0" always evaluates to 0. However, it is not clear from the FCD whether "n*0" is thus a constant expression (and therefore a valid null pointer constant) inside the body of test2. Either way both #5 and #6 should print the same value; it would violate ODR for #5 to print "1" (indicating that "n*0" was a valid null pointer constant when test2() is evaluated in a		

1 **MB** = Member body (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	(3)	4	5	(6)	(7)
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Secretariat observations** on each comment submitted
				constant expression context) whilst #6 prints "2" (since n*0 is not a constant expression if n is not constant). #5 and #6 should thus both print "1", or both print "2".		
US 37	6.5		te	"for (auto e : range)" creates copies of elements. This seems like a gotcha for new users. Not only are copies inefficient for reading, but writing to copies won't modify the original elements. Permitting "for (identifier : expression)" and giving it the same meaning as "for (auto& identifier : expression)" would make the range-based for statement easier to teach and to use, and should be trivial to specify and to implement.	Permit "for (identifier : expression)" or similar, with the same meaning as "for (auto& identifier : expression)".	
CA 3	Various	various	various	Canada agrees with US 37, 44, 47, 85, 77, 92, 97, 102, 105, 109	Resolve as suggested in these comments	
US 38	6.5	5	te	The statement that certain infinite loops may be assumed to terminate should also apply to go-to loops and possibly infinite recursion. We expect that compiler analyses that would take advantage of this can often no longer identify the origin of such a loop.	As a strawman, replace the paragraph with "The implementation may assume that any program will eventually do one of the following: - terminate, - make a call to a library I/O function, - access or modify a volatile object, or - perform a synchronization operation (1.10) or atomic operation (Clause 29)." Possibly move this and the attached note to section 1.9, after p8.	
JP 69	6.35.1	2	E	Constant width font should be used for 3 "while"s in the paragraph as described in Syntax notation (1.6).	Change the font for "while" to constant width type. When the condition of a **while** statement is a declaration, the scope of the variable that is declared extends	

1 **MB** = Member body (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	(3)	4	5	(6)	(7)
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Secretariat observations** on each comment submitted
					from its point of declaration (3.3.2) to the end of the **while** statement. A **while** statement of the form	

GB 27	6.5.4p1		Te	6.5.4/1 requires that range-based for loops behave as if they had "{ auto&& __range = (expression); for (auto __begin = begin_expr, __end = end_expr; ..." which implies that __begin and __end must have the same type. However, this prevents stateful iterators with an end sentinel of a different type. Since range-based for loops' equivalent code already introduces a block (for the __range variable), could __begin and __end be placed there, as "auto __begin = begin_expr; auto __end = end_expr;"?
				
				Example of what this change would allow, only the relevant details shown with ctors, op*, op++, etc. omitted: (apologies if the formatting is lost)
				struct End {}; struct Counter { Counter& begin() { return *this; } // used by begin_expr End end() { return End(); } // used by end_expr bool operator!=(End) const { return _current != _end; }
				
				Counter(int begin, int end) : _current(begin), _end(end) {} int _current, _end; };
				
				void use_example() { for (auto n : Counter(0, 10)) { // n takes values from 0..9 } }
					Change the "as if" for a range-based for-loop in 6.5.4p1 to move the initialization of __begin and __end outside the loop into the enclosing block:
					"{ auto&& __range = (expression);
					auto __begin = begin_expr; auto __end = end_expr;
					for (; ..."	

CH 7	6.5.4	p1	te	The phrasing "is equivalent to" is too restrictive and might constrain future improvements.	Make clear that the specification is not necessarily the implementation, i.e. that the expressions in the specification are not necessarily called at all and that the order in which the statement is executed for different values of for-range-declaration is not necessarily the same as if the for loop would have been written the way in the specification.	
GB 28	7	1	Ed	"Attributes" is misspelled	Replace "Atrributes" with "Attributes"	

1 **MB** = Member body (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	(3)	4	5	(6)	(7)
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Secretariat observations** on each comment submitted
US 39	7.1	1	te	The current wording is, "The optional *attribute-specifier* in a *decl-specifier-seq* appertains to the type determined by the *decl-specifier-seq*." However, the rule for *decl-specifier-seq* in the grammar is recursive, and the intent is for the *attribute-specifier* to appertain to the top *decl-specifier-seq*, not the one in which the *attribute-specifier* directly appears.	Change the wording to indicate that the complete or outermost *decl-specifier-seq* is intended.	
GB 29	7.1.5		Te	A constexpr function is not permitted to return via an exception. This should be recognised, and a function declared 'constexpr' without an explicit exception specification should be treated as if declared 'noexcept(true)' rather than the usual 'noexcept(false)'. For a function template declared constexpr without an explicit exception specification, it should be considered 'noexcept(true)' if and only if the constexpr keyword is respected on a given instantiation.	Give constexpr functions an implicit non-throwing exception specification.	

US 40	7.1.6.2	4	te	The description of decltype does not specify whether the type of a parameter is the declared type or the type as adjusted in 8.3.5¶5:
				auto f(int a[])->decltype(a);
				// ill-formed or int*?
				auto g(const int i)->decltype(i);
				// int or const int?
					Clarify the wording to indicate that the type of a parameter is after the array- and function-to-pointer decay but before the removal of cv-qualification.	

DE 9	7.1.6.2	p4	te	decltype applied to a function call expression requires a complete type (5.2.2 paragraph 3 and 3.2 paragraph 4), even though decltype's result might be used in a way that does not actually require a complete type. This might cause undesired and excessive template instantiations.	When immediately applying decltype, do not require a complete type, for example for the return type of a function call.	

US 41	7.1.6.3	2	te	The current wording disallows use of *typedef-name*s in *elaborated-type-specifier*s. This prohibition should also apply to template aliases:
				struct A { };
				template<typename T> using X = A;
				struct X<int>* p2; // ill-formed
					Add the necessary wording to prohibit a specialization of a template alias in an *elaborated-type-specifier*.	

1 **MB** = Member body (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	(3)	4	5	(6)	(7)
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Secretariat observations** on each comment submitted
US 42	7.1.6.4		te	The overloaded meaning of the auto specifier is confusing and prevents possible future language enhancements.	Choose another keyword to indicate a late-specified return type. The identifiers lateret and postret have no Google code search hits. The identifiers late, latetype, and posttype have 30-40 hits.	
US 43	7.6	4	ed	The contexts in which an *attribute-specifier* can appear include statements, described in clause 6, but the cross-references to clauses describing those contexts do not include clause 6.	Add clause 6 to the list of cross-references in the first sentence.	
GB 30	7.6.1	6	Te	Making the use of [[illegal except where introducing an attribute specifier is just reintroducing the problem we had with >> for closing nested templates, albeit in a minor and less common form. As Jason Merrill commented in c++std-core-16046, there is no ambiguity in practice because code couldn't actually be well-formed under interpretation as both an attribute specifier and a lambda introducer. A small amount of lookahead would be required to differentiate the cases, but this should not be a problem. This restriction also means that lambdas in macros must be enclosed in parentheses to avoid accidental interpretation as an illegal attribute specifier if used as an array index.	Delete 7.6.1 paragraph 6.	
JP 23	7.6.1	4	E	Representations of reference link are not unified. Most reference links to clause (table) number, say X, are in the form "Clause X" ("Table X") capitalized, and subsection Y.Y.Y is referenced with its number only in the form "Y.Y.Y". Whether they are parenthesized or not depends on the context. However there are some notations "(Z)" consisting of only a number Z in parentheses to confer Clause or Table number Z.	Change "(clause 7, clause 8)" to "(Clause 7, Clause 8)".	
GB 31	7.6.2		Te	After reviewing the case for attributes, wg14 has opted not to adopt this feature, and is instead using keywords for the	Revert the changes in the initial alignment proposal that changed the 'alignas' keyword into	

1 **MB** = Member body (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	(3)	4	5	(6)	(7)
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Secretariat observations** on each comment submitted
				few important cases identified in the attributes proposal. For compatibility with C, the change of the 'alignas' keyword to the '[[align]]' attribute should be reversed.	an attribute.	
GB 32	7.6.3		Ge	C has rejected the notion of attributes, and introduced the noreturn facility as a keyword. To continue writing clean, portable code we should replace the [[noreturn]] attribute with a 'noreturn' keyword, following the usual convention that while C obfuscates new keywords with _Capital and adds a macro to map to the comfortable spelling, C++ simply adopts the all-lowercase spelling.	Replace the [[noreturn]] attribute with a new keyword, 'noreturn', with identical semantics. Note that this implies the keyword will not be something that a function can be overloaded upon.	

US 44	7.6.5		te	Even if attributes continue to be standardized over continued objections from both of the two vendors who are cited as the principal prior art, we can live with them with the exception of the virtual override controls. This result is just awful, as already shown in the example in 7.6.5 (excerpted):
				class D [[base_check]] : public B {
				void sone_func [[override]] ();
				virtual void h [[hiding]] (char*);
				};
				Here we have six keywords (not counting void and char) — three normal keywords, and three [[decorated]] keywords. There has already been public ridicule of C++0x about this ugliness. This is just a poor language design, even in the face of backward compatibility concerns (e.g., that some existing code may already use those words as identifiers) because those concerns have already been resolved in other ways in existing practice (see below).
				More importantly, this is exactly the abuse of attributes as disguised keywords that was objected to and was explicitly promised not to happen in order to get this proposal passed. The use of attributes for the virtual control keywords is the most egregious abuse of the
					Change the syntax for virtual override control to not use attributes.	

1 **MB** = Member body (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	(3)	4	5	(6)	(7)
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Secretariat observations** on each comment submitted

				attribute syntax, and at least that use of attributes must be fixed by replacing them with non-attribute syntax.
				
				These virtual override controls are language features, not annotations.
				
				It is possible to have nice names and no conflicts with existing code by using contextual keywords, such as recognizing the word as having the special meaning when it appears in a grammar position where no user identifier can appear, as demonstrated in C++/CLI which has five years of actual field experience with a large number of customers (and exactly no name conflict or programmer confusion problems reported in the field during the five years this has been available):
				
				class D : public B {
				
				void sone_func() override; // same meaning as [[override]] – explicit override
				
				virtual void h (char*) new; // same meaning as [[hiding]] – a new function, not an override
				
				};
				
				int override = 42; // ok, override is not a reserved keyword
				
				The above forms are implementable, have been implemented, have years of practical field experience, and work. Developers love them. Whether the answer is to follow this existing practice or something else, there needs to be a more natural replacement for the currently [[attributed]] keywords for virtual override control which is an ugly novelty that has no field experience and that developers have already ridiculed.		

US 45	7.6.5	6	ed	The example includes a line reading class D [[base_check]] : public B {	Change the example to read class [[base_check]] D : public B {	

1 **MB** = Member body (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	(3)	4	5	(6)	(7)
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Secretariat observations** on each comment submitted
				However, the current syntax in 9¶1 places the *attribute-specifier* before the class name.		
CH 8	8.1	p1 (syntax)	ed	'noptr-abstract-declarator:' rule misses 'opt' subscript from the constant expression within the array brackets. This seems to be an editorial oversight	change: "*noptr-abstract-declarator*$_{opt}$ **[** *constant-expression* **]** *attribute-specifier*$_{opt}$" to *"noptr-abstract-declarator*$_{opt}$ **[** *constant-expression*$_{opt}$ **]** *attribute-specifier*$_{opt}$"	
US 46	8.3.2 20.7.6.2	all Table 49	te	There is no way to create a prvalue of array type, so there ought to be no way create a (nonsensical) rvalue reference to array type.	In [dcl.ref]/2, disallow declarations of T (&&A)[]. In [dec.ref]/6 add a sentence: If a typedef, a type *template-parameter*, or a dec*ltype-specifier* denotes a type A that is an array type (of known or unknown size), an attempt to create the type "rvalue reference to *cv* A" creates the type A&. In [meta.trans.ref]/Table 49 change the third row as follows: If T names an array type, then the member typedef type shall name T&, otherwise if T names an object or function type...	
GB 33	8.3.5	5	Ed	The register keyword is deprecated, so does not make for a good example. Suggest substituting the new storage class specifier, 'thread_local', instead.	Use 'thread_local' in place of 'register' in the following sentance: "[Example: register char* becomes char* —end example]"	
US 47	8.4.2		te	8.4.2 [dcl.fct.def.default]/4 says: "A special member function is user-provided if it is user-declared and not explicitly defaulted on its first declaration. A user-provided explicitly-defaulted function is..." The second sentence here should say "A user-declared explicitly-defaulted function is...".	Change "A user-provided explicitly-defaulted function is..." to "A user-declared explicitly-defaulted function is...".	
GB 34	8.4.2	p2	Ed	It is confusing when a normative paragraph starts with a note. The note starting this paragraph, with its reference to 'this' relating to the previous paragraph and not the content that follows, should be moved into the first paragraph, or the rest of this paragraph after the note should start a new numbered paragraph.	The note starting this paragraph should be moved into the first paragraph, or the rest of this paragraph after the note should start a new numbered paragraph.	

1 **MB** = Member body (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	(3)	4	5	(6)	(7)
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Secretariat observations** on each comment submitted
FI 1	8.4.2 [dcl.fct.def.default]	Paragraph 2	te	It should be allowed to explicitly default a non-public special member function on its first declaration. It is very likely that users will want to default protected/private constructors and copy constructors without having to write such defaulting outside the class.	Strike the “it shall be public” bullet.	
FI 2	8.4.2 [dcl.fct.def.default]	Paragraph 2	te	It should be allowed to explicitly default an explicit special member function on its first declaration. It is very likely that users will want to default explicit copy constructors without having to write such defaulting outside of the class.	Strike the “it shall not be explicit” bullet. See Appendix 1 - Additional Details	
FI 3	8.4.2 [dcl.fct.def.default]	Paragraph 2	te	It should be allowed to explicitly default a virtual special member function on its first declaration. It is very likely that users will want to default virtual copy assignment operators and destructors without having to write such defaulting outside of the class.	Strike the “it shall not be virtual” bullet. See Appendix 1 - Additional Details	
GB 35	8.5.1	7	Te	With the removal of the deprecated string-literal-to-non-const-char* conversion, member 'b' of struct S should be declared as a 'const' char *.	Fix struct S as: "struct S { int a; const char* b; int c; };"	

JP 71	8.5.1	7	E	"char*" should be "const char *".
				The special rule to convert character literal to pointer has been removed from "4.2 Array-to-pointer conversion [conv.array]".
				char * p1 = "..." ; // ill-formed.(removing const'ness)
				char const *p2 = "..." ;// well-formed.
				There are many code fragments depending on the removed rule. They are ill-formed.
					Change to:
					struct S { int a; const char* b; int c; };
					S ss = { 1, "asdf" };	

JP 72	8.5.1	15	E	"char*" should be "const char *".
				The special rule to convert character literal to pointer has been removed from "4.2 Array-to-pointer conversion [conv.array]".
				char * p1 = "..." ; // ill-formed.(removing const'ness)
					Change to:
					union u { int a; const char* b; };
					u a = { 1 };
					u b = a;
					u c = 1; // error	

1 **MB** = Member body (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	(3)	4	5	(6)	(7)
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Secretariat observations** on each comment submitted

				char const *p2 = "..." ;// well-formed.
				There are many code fragments depending on the removed rule. They are ill-formed.
					u d = { 0, "asdf" }; // error
					u e = { "asdf" }; // error	

GB 36	8.5.1	17	Ed	The 'b' member of union u should be declared const char * to better illustrate the expected cause of failures.	Update union u as: "union u { int a; const char* b; };"	

US 48	8.5.3	5	te	The rule "...or the reference shall be an rvalue reference and the initializer expression shall be an rvalue or have a function type" is stated in terms of the rvalue-ness of the expression rather than the eventual target of the reference; this leads to some undesirable results, such as
				struct A { };
				struct B {
				operator A&();
				};
				
				A&& aref = B(); // binds to lvalue
				(c++std-core-16305)
					Correct the formulation to deal with the rvalue-ness of the initializer after conversion to the appropriate type.	

US 49	8.5.3	5	te	The FCD does not specify direct binding for this example:
				int i;
				int main()
				{
				int&& ir = static_cast<int&&>(i);
				ir = 42;
				return (i != 42);
				}
				(c++std-core-16181)
					See Appendix 1 - Additional Details	

GB 37	8.5.3		Te	It seems that lvalues of any sort don't bind to non-const rvalue ref args, even if an intermediate temporary would be created. See the discussion at http://stackoverflow.com/questions/2748866/c0x-rvalue-references-and-temporaries.	Possible wording: amend the second list item in 8.5.3/5: Otherwise, the reference shall be an lvalue reference to a non-volatile const type (i.e., cv1 shall be const), or the reference shall be an rvalue reference [deleted the rest of the sentence].	

1 **MB** = Member body (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	(3)	4	5	(6)	(7)
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Secretariat observations** on each comment submitted

				I'll summarise that here: Assume that std::vector has push_back overloads declared as follows, with SFINAE omitted for clarity:
				
				void push_back(const T &);
				//Copies the const T & into the vector's storage
				
				void push_back(T &&);
				//Moves the T && into the vector's storage
				
				Then this code appears to behave as commented, as of N3090:
				
				const char * cchar = "Hello, world"; std::vector<std::string> v;
				v.push_back(cchar);
				//makes a temporary string, copies the string into vector storage using push_back(const T&)
				
				v.push_back(std::string(cchar));
				//makes a temporary string, moves the string into vector storage using push_back(T&&)
				
				v.push_back(std::move(cchar));
				//makes a temporary string, moves the string into the vector using push_back(T&&)
				
				Johannes Schaub (litb) convincingly argued that the reason for this is clause 8.5.3/5 describing reference binding - it allows direct binding of lvalues, bindings that require conversion sequences to const lvalue refs, and rvalues to rvalue refs. But it doesn't allow for lvalues to ever bind to rvalue refs, even if an intermediate temporary would otherwise need to be created.
				
				This isn't what I (as a user of std::vector) expect to
					[The last example would also need to be deleted.]	

1 **MB** = Member body (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	(3)	4	5	(6)	(7)
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Secretariat observations** on each comment submitted

				happen. I expect all of these push_back calls to do the same thing, namely, select the push_back(T&&) overload, create a temporary string object from 'cchar', bind the temporary string to the argument, and hence move (not copy) the temporary string into vector's storage.
				
				It seems particularly strange that v.push_back(cchar) "requires" an extra copy, but v.push_back(std::move(cchar)) does not. It almost seems like indicating that 'cchar' is potentially-movable (by casting it to an rvalue ref using std::move) allows moving from a completely different object - the temporary string.
				
				I suggest extending the rules for initializing const lvalue refs via implicit conversions (8.5.3/5), to also apply to rvalue refs.
				
				This also raises an additional question of whether lvalues of _copyable_ types should be copied into a temporary object so that they may bind to an rvalue ref. Allowing this would not affect const T&/T&& overload pairs. But it could be potentially useful when writing functions that wish to accept a number of rvalue refs to copyable-but-not-movable types (such as all C++03 classes with user-defined copy constructors), or when writing functions that "take apart" a number their arguments in a way that is different from a straightforward move (perhaps some tree operations would want to do this).
				
				Conversely, it might seem odd that declarations such as:
				string && s = string_lvalue;
				string && s = string_rvalue_ref_variable;
				//mistake, std::move(string_rvalue_ref_variable) was intended
				
				...would both		

1 **MB** = Member body (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	(3)	4	5	(6)	(7)
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Secretariat observations** on each comment submitted
				silently copy their arguments and bind to the copy, instead of being invalid as they are now. We believe this is core issue 953		

DE 10	8.5.3		te	Reference binding rules for rvalue references should consider temporaries generated from lvalues by implicit conversions. Consider to postpone the lvalue/rvalue analysis of an expression to after the implicit conversion chain has been deduced. Example:
				`void f(std::string&&);`
				`void g() {`
				`f(std::string("hello")); // #1, ok`
				`f("hello"); // #2, error,`
				`// but should be the same as #1`
				`}`		

US 50	9	9	te	the class "struct A { const int i; };" was a POD in C++98, but is not a POD under the FCD rules because it does not have a trivial default constructor; I believe that C++0x POD was intended to be a superset of C++98 POD.	change POD to be standard layout and trivially copyable?	
FI 16	9 9 9 3.9	5 trivial 6 std-layout 9 POD 10 literal type	ge	There are definitions for these types in the text, yet it is left unclear what use these classifications have. The types are very close to each other, which makes them confusing. If the reader must rely on external references, then these references should be specified (which is undesirable, or even disallowed by ISO(?)). As it stands, there is an example for using standard-layout classes (with other programming languages). There are also uses specified for literal types. One can imagine many uses for these four/five types, so it is important to have a clear specification of the intent as to where each of these types is expected to be used.	It is necessary to have detailed information on the expected uses of standard-layout, trivial, trivially copyable, literal and POD types.	
JP 81	9	9	E	Missing description of acronym "POD", which existed in C++03: The acronym POD stands for "plain old data."	Add "The acronym POD stands for "plain old data." as an annotation.	

1 **MB** = Member body (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	(3)	4	5	(6)	(7)
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Secretariat observations** on each comment submitted
US 51	9.2 [class.mem]		ge	Non-static data member initializers have no implementation experience.	Either demonstrate a complete implementation of this feature or remove N2756 from the working paper prior the FDIS.	
US 52	9.3	7	te	The current wording allows friend declarations to name member functions "after their class has been defined." This appears to prohibit a friend declaration in a nested class defined inside its containing class that names a member function of the containing class, because the containing class is still considered to be incomplete at that point.	Change the wording to allow a friend declaration of a "previously-declared member function."	
US 53	9.3.1 [class.mfct.non-static]		ge	Move semantics for *this have no implementation experience.	Either demonstrate a complete implementation of this feature or remove N1821 from the working paper prior the FDIS.	

JP 73	9.3.1	3	E	"char*" should be "const char *".
				The special rule to convert character literal to pointer has been removed from "4.2 Array-to-pointer conversion [conv.array]".
				char * p1 = "..." ; // ill-formed.(removing const'ness)
				char const *p2 = "..." ;// well-formed.
				There are many code fragments depending on the removed rule. They are ill-formed.
					Change to:
					struct tnode {
					char tword[20];
					int count;
					tnode *left;
					tnode *right;
					void set(const char*, tnode* l, tnode* r);
					};
					void tnode::set(const char* w, tnode* l, tnode* r) {
					count = strlen(w)+1;
					if (sizeof(tword)<=count)
					perror("tnode string too long");
					strcpy(tword,w);
					left = l;
					right = r;
					}
					void f(tnode n1, tnode n2) {
					n1.set("abc",&n2,0);
					n2.set("def",0,0);
					}	

1 **MB** = Member body (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	(3)	4	5	(6)	(7)
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Secretariat observations** on each comment submitted
US 54	9.5 [class.union]		ge	Unrestricted unions have no implementation experience.	Either demonstrate a complete implementation of this feature or remove N2544 from the working paper prior the FDIS.	

JP 74	9.5	6	E	"char*" should be "const char *".
				The special rule to convert character literal to pointer has been removed from "4.2 Array-to-pointer conversion [conv.array]".
				char * p1 = "..." ; // ill-formed.(removing const'ness)
				char const *p2 = "..." ;// well-formed.
				There are many code fragments depending on the removed rule. They are ill-formed.
					Change to:
					void f() {
					union { int a; const char* p; };
					a = 1;
					p = "Jennifer";
					}	

GB 38	9.6		Te	The signedness of bit-fields is the only time when 'signed int' is any different to just 'int'. In C it is possible to remember whether a typedef uses 'signed' but in C++ it doesn't make sense and will result in ODR violations if A<long> and A<signed long> are not exactly equivalent. This also causes portability problems because it is not specified whether typedefs such as int32_t are defined with 'signed' so using the <cstdint> types in bitfields is problematic. It is common to want to guarantee a bit-field has a minimum number of bits, for which the <cstdint> types are useful, except that the signedness of a bit-field using int32_t might depend on both unspecified and implementation-defined behaviour.	'signed int' should always be equivalent to 'int' in all contexts. A possible alternative would be to specify that signed types in <cstdint> are declared with 'signed' so that using them for bit-fields has predictable results, but this doesn't address the ODR issue with A<long> and A<signed long>	
US 55	10.3	Paragraph 5	te	The following code not only does not compile on the compilers I've tested, but cannot be fixed through any combinations of forward references class B {	In the core mailing list, Daniel Krugler points out that the current wording is ambiguous as to whether this is legal (although an editorial example suggests otherwise), and observes that it should be OK as long as D2 is complete at the	

1 **MB** = Member body (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	(3)	4	5	(6)	(7)
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Secretariat observations** on each comment submitted

				public:
				  virtual B *f() = 0;
				};
				class D1 : public B {
				public:
				  virtual D2 *f();
				};
				class D2 : public B {
				public:
				  virtual D1 *f();
				};
					point of definition of D1::f. The standard should resolve the ambiguity by saying that D2 only needs to be complete at the point of definition of D1::f.
					
					The core mailing list message text is below:
					I would be happy, if the standard would just allow this, but IMO the
					current wording seems to be readable in different ways (e.g.
					Comeau rejects the code). I think the reason is that [class.virtual]/5
					just says:
					
					"The return type of an overriding function shall be either identical to
					the return type of the overridden function or covariant with the
					classes of the functions.[..]"
					
					This restriction is IMO only necessary for the ***definition*** of D::f.
					Secondly p. 6 says:
					
					"If the return type of D::f differs from the return type of B::f, the class
					type in the return type of D::f shall be complete at the point of
					declaration of D::f or shall be the class type D.[..]"
					
					and shows the following example that explicitly forbids that (simplified):
					
					struct Base {
					  virtual B* vf5();
					};	

1 **MB** = Member body (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	(3)	4	5	(6)	(7)
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Secretariat observations** on each comment submitted

					class A;
					
					struct Derived : public Base {
					A* vf5(); // error: returns pointer to incomplete class
					};	

US 56	11.3 [class.access.decl]		te	Access declarations were deprecated in the 1998 standard and have no benefits over using declarations.	Remove access declarations from the working paper.	
FI 4	12.1 [class.ctor]	Paragraph 5	te	What effect does defaulting have on triviality? Related to FI 1, non-public special members defaulted on their first declaration should retain triviality, because they shouldn't be considered user-provided. Related to FI 3, defaulted member functions that are virtual should not be considered trivial, but there's no reason why non-virtuals could not be. Furthermore, a class with a non-public explicitly-defaulted constructor isn't ever trivially constructible under the current rules. If such a class is used as a subobject, the constructor of the aggregating class should be trivial if it can access the non-public explicitly defaulted constructor of a subobject.	Change the triviality rules so that a class can have a trivial default constructor if the class has access to the default constructors of its subobjects and the default constructors of the subobjects are explicitly defaulted on first declaration, even if said defaulted constructors are non-public. See Appendix 1 - Additional Details	
FI 15	12.3.1	2	ge	12.3.1. 2: "A default constructor may be an explicit constructor; such a constructor will be used to perform default-initialization or value-initialization (8.5)." 12.3. 1 also says that an explicit ctor is different from a non-explicit ctor in that it is only invoked when direct-initialization (T a(1); T a{1}. presumably also T a;) or casting is used. What are the scenarios for the default ctor where explicit actually matters? Temporaries, arrays, ???	The difference between a no argument default constructor and an explicit no argument default constructor should be explained in the standard. If there is no difference, this should be explicitly specified.	

1 **MB** = Member body (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	(3)	4	5	(6)	(7)
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Secretariat observations** on each comment submitted
				When, if ever, is an explicit default ctor different from a non-explicit ctor?		
US 57	12.3.2 [class.conv.fct]		ge	Explicit conversion operators have no implementation experience.	Either demonstrate a complete implementation of this feature or remove N2437 from the working paper prior the FDIS.	
GB 39	12.4	4	Te	Contradiction between the note and normative language describing when defaulted function definitions might have an exception specification. (See 8.4.2p2 for requirement to provide the exception specification)	Either strike the second sentance of this note, or update it to reflect under which conditions a defaulted definition might have an exception specification.	
GB 40	12.4		Te	It is very difficult to write correct programs that do not call 'terminate' in the presence of destructors that can throw exceptions, and this practice has long been discouraged. Many implicitly declared destructors already carry noexcept declarations (mostly types with trivial destructors) and it is anticipated it will become common practice to want a user-declared destructor to be declared noexcept. This becomes important evaluating the noexcept operator, where any of the unevaluated sub-expressions may produce a temporary object. As this is expected to be the overwhelmingly common case, a user-declared destructor that does not supply an exception specification should be considered as if declared noexcept(true) rather than noexcept(false), the default for every other function.	a user-declared destructor that does not supply an exception specification should be considered as if declared noexcept(true) rather than noexcept(false), the default for every other function	
CH 9	12.4 and 15.4		te	Destructors should generally not throw exceptions. Consider giving an explicit rule for this.	Add in 12.4 or 15.4 a paragraph to the effect that all destructors not having an exception soecification are considered noexcept(true).	
US 58	12.5	foot 117	ed	Missing comma in "is not virtual the size might".	Add the comma.	
US 59	12.6.2 [class.base.init]		ge	Delegating constructors have no implementation experience.	Either demonstrate a complete implementation of this feature or remove N1986 from the working paper prior the FDIS.	

1 **MB** = Member body (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	(3)	4	5	(6)	(7)
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Secretariat observations** on each comment submitted
US 60	12.8 [class.copy]		ge	Implicitly-defined move constructors and move assignment operators have no implementation experience.	Either demonstrate a complete implementation of this feature or remove N3053 from the working paper prior the FDIS.	
DE 11	12.8		te	It is unclear whether copy elision can or cannot apply to a case like C f(C c) { return c; }, i.e. where a parameter of class type is returned. Furthermore, if copy elision cannot apply there, it should still be possible to move (instead of copy) the return value.	Amend paragraph 34 to explicitly exclude function parameters from copy elision. Amend paragraph 35 to include function parameters as eligible for move-construction.	
FI 5	12.8 [class.copy]	Paragraph 13, paragraph 27	te	Same as FI 4, the parts involving copy constructors and copy assignment operators. A class with a non-public explicitly-defaulted copy constructor isn't ever trivially copyable under the current rules. If such a class is used as a subobject, the copy constructor of the aggregating class should be trivial if it can access the non-public explicitly defaulted copy constructor of a subobject.	Change the triviality rules so that a class can have a trivial copy constructor if the class has access to the copy constructors of its subobjects and the copy constructors of the subobjects are explicitly defaulted on first declaration, even if said defaulted copy constructors are non-public. See Appendix 1 - Additional Details	
GB 41	12.8	15, 29	Te	Contradiction between the notes claiming that defaulted definitions to not have exception specifications, and the normative language in 8.4.2 which declares that they might.	Either strike the second sentance of each note, or update it to reflect under which conditions a defaulted definition might have an exception specification.	
US 61	12.8;20.2.5	17,31;table 42	ed	static_cast is broken across two lines	do not hyphenate static_cast	
US 62	12.8	16	te	The new wording describing generated copy constructors does not describe the initialization of members of reference type. (Core issue 1051 in N3083.)	Add the required description.	
US 63	12.8	16-18	te	The new wording specifies the behavior of an implicitly-defined copy constructor for a non-union class (¶16), an implicitly-defined move constructor for a non-union class (¶17), and an implicitly-defined copy constructor for a union (¶18), but not an implicitly-defined move constructor for a union. (Core issue 1064 in N3083.)	Add the required description.	
US	12.8	28	te	The current wording reads, "A copy/move assignment	Clarify the wording so that only the operator	

1 **MB** = Member body (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	(3)	4	5	(6)	(7)
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Secretariat observations** on each comment submitted
64				operator that is defaulted and not defined as deleted is *implicitly defined* when an object of its class type is assigned a value of its class type or a value of a class type derived from its class type or when it is explicitly defaulted after its first declaration." This sounds as if any assignment to a class object, regardless of whether it is a copy or a move assignment, defines both the copy and move operators. Presumably an assignment should only define the assignment operator chosen by overload resolution for the operation. (Compare the corresponding wording in ¶14 for the copy/move constructors: "...*implicitly defined* if it is used to initialize an object of its class type..." (Core issue 1066 in N3083.)	needed for the operation is implicitly defined.	
US 65	12.9 [class.inhctor]		ge	Inheriting constructors have no implementation experience.	Either demonstrate a complete implementation of this feature or remove N2540 from the working paper prior the FDIS.	

JP 65	13.1	3	E	In some code examples, ellipsis(…) is used in ill-formed. In these cases, "…" represents omission of some codes like this:
				class A { /* ... */ } ;
				But in some cases, it is used without commented-out as below:
				class A { ... } ;
				It is an inconsistent usage. They all should be enclosed in a comment.
					Change to:
					int f (int) { /* ... */ } // definition of f(int)
					int f (cInt) { /* ... */ } // error: redefinition of f(int)	

US 66	13.3.1.7	1	te	overload resolution should first look for a viable list constructor, then look for a non-list constructor if no list constructor is viable	See Appendix 1 - Additional Details	
US 67	13.3.2	3	ge	To determine whether there is an ICS, 13.3.2 uses 13.3.3.1 instead of just saying "there is an ICS if-and-only-if a copy init would be well-formed." Apparently this is desired, but to a casual reader or an implementor reading these rules for the first time for a new implementation, it's	Insert a note or annex explaining why 13.3.2 does things as it does.	

1 **MB** = Member body (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	(3)	4	5	(6)	(7)
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Secretariat observations** on each comment submitted
				not clear *why* that's desired.		

JP 75	13.2	1	E	"char*" should be "const char *".
				The special rule to convert character literal to pointer has been removed from "4.2 Array-to-pointer conversion [conv.array]".
				char * p1 = "..." ; // ill-formed.(removing const'ness)
				char const *p2 = "..." ;// well-formed.
				There are many code fragments depending on the removed rule. They are ill-formed.
					Change to:
					struct B {
					int f(int);
					};
					struct D : B {
					int f(const char*);
					};	

JP 76	13.2	2	E	"char*" should be "const char *".
				The special rule to convert character literal to pointer has been removed from "4.2 Array-to-pointer conversion [conv.array]".
				char * p1 = "..." ; // ill-formed.(removing const'ness)
				char const *p2 = "..." ;// well-formed.
				There are many code fragments depending on the removed rule. They are ill-formed.
					Change to:
					void f(const char*);
					void g() {
					extern void f(int);
					f("asdf"); // error: f(int) hides f(const char*)
					// so there is no f(const char*) in this scope
					}	

JP 77	13.3.1.2	1	E	"char*" should be "const char *".
				The special rule to convert character literal to pointer has been removed from "4.2 Array-to-pointer conversion [conv.array]".
				char * p1 = "..." ; // ill-formed.(removing const'ness)
				char const *p2 = "..." ;// well-formed.
				There are many code fragments depending on the removed rule. They are ill-formed.
					Change to:
					struct String {
					String (const String&);
					String (const char*);
					operator char* ();
					};
					String operator + (const String&, const String&);
					void f(void) {
					char* p= "one" + "two"; // ill-formed because neither
					// operand has user-defined type
					int I = 1 + 1; // Always evaluates to 2 even if
					// user-defined types exist which
					// would perform the operation.
					}	

1 **MB** = Member body (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	(3)	4	5	(6)	(7)
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Secretariat observations** on each comment submitted
US 68	13.4	1	te	Overload resolution within the operand of a unary & operator is done by selecting the function "whose type matches the target type required in the context." The criterion for determining whether the types match, however, is not defined. At least three possibilities suggest themselves: 1. The types are identical. 2. The source type can be implicitly converted to the target type. 3. The expression would be well-formed if the function under consideration were not overloaded. This question arises for pointer-to-member types, where there is an implicit conversion from a pointer-to-base-member to a pointer-to-derived-member, as well as when the context is an explicit type conversion (which allows for static_cast a conversion from pointer-to-derived-member to a pointer-to-base-member and, in the reinterpret_cast interpretation of functional and old-style casts, essentially any conversion).	Specify the intended criterion for determining whether the types match.	
JP 82	13.5.8	8	E	Typo, "lteral" should be "literal". // error: invalid lteral suffix identifier	Correct typo. // error: invalid literal suffix identifier	
US 69	14.3.2	para 1	te	The standard permits the address of thread_local variable as a non-type template parameter. The addresses of these variables are not constant, however.	Require static storage duration for non-type parameters.	
DE 12	14.3.2		te	Now that local classes can be used as template arguments, it seems odd that there are "external linkage" restrictions on non-type template parameters.	Permit addresses of objects and functions with internal linkage as arguments for non-type template parameters.	
JP 78	14.3.2	2	E	"char*" should be "const char *". If not corrected, type mismatch is another cause of error in the example below, which is not appropriate for an example here.	template<class T, const char* p> class X {	

1 **MB** = Member body (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	(3)	4	5	(6)	(7)
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Secretariat observations** on each comment submitted

				template<class T, char* p> class X {
				X();
				X(const char* q) { / ... / }
				};
				X<int, "Studebaker"> x1; *// error: string literal as template-argument*
				char p[] = "Vivisectionist";
				X<int,p> x2; *// OK*		

JP 83	14.3.2	2	E	Constructors in template declaration are not essential for this example to explain string literal error use.
				
				template<class T, char* p> class X {
				X();
				X(const char* q) { /* ... */ }
				};
				X<int, "Studebaker"> x1; // error: string literal as template-argument
				char p[] = "Vivisectionist";
				X<int,p> x2; // OK
					Delete two constructors in template declaration as follows.
					
					template<class T, const char* p> class X {
					/* ... */
					};
					X<int, "Studebaker"> x1; // error: string literal as template-argument
					char p[] = "Vivisectionist";
					X<int,p> x2; // OK	

GB 42	14.5.3	1	Ed	The name "eror" should probably be "error".	Replace: Tuple<0> eror; With: Tuple<0> error;	
JP 24	14.5.3	4	E	Representations of reference link are not unified. Most reference links to clause (table) number, say X, are in the form "Clause X" ("Table X") capitalized, and subsection Y.Y.Y is referenced with its number only in the form "Y.Y.Y". Whether they are parenthesized or not depends on the context. However there are some notations "(Z)" consisting of only a number Z in parentheses to confer Clause or Table number Z.	Change "(10)" to "(Clause 10)".	
US	14.5.6.2	2, 4	te	14.8.2.4¶3 specifies that the deduction used in partial ordering in a non-call context is based on the complete	Update the wording in 14.5.6.2 to say only that deduction is performed as described in 14.8.2.4	

1 **MB** = Member body (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	(3)	4	5	(6)	(7)
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Secretariat observations** on each comment submitted
70				function type of the function templates. The wording in 14.5.6.2¶2 (and echoed in ¶4) reflects an earlier specification, however, saying that the deduction uses only "the function parameter types, or in the case of a conversion function the return type." This is a contradiction.	and not to specify which types are used.	
US 71	unused	unused		unused		NA
CA 7	14.5.6.2p3	P3	te	r country In FCD sub-clause 14.5.6.2 [temp.func.order] paragraph 3, we are told to synthesize, "for each type, non-type, or template template parameter... a unique type, value, or class template respectively." These are then substituted for each occurrence of the respective parameter in the function type of the function template. It is not specified what the properties of said synthetics are, for example, members of a dependent type referred to in non-deduced contexts are not specified to exist, although the transformed function type would be invalid if they do not exist. We may assume, for example, that each synthetic will be given minimal properties such that the transformed function type is valid at the point of definition of the template. **Example 1:** template <typename T, typename U> struct A; template <typename T> void foo(A<T, typename T::u> *) { } // #1 // synthetic T1 has member T1::u template <typename T> void foo(A<T, typename T::u::v> *) { } // #2 // synthetic T2 has member T2::u and member T2::u::v // T in #1 deduces to synthetic T2 in partial ordering; // deduced A for the parameter is A<T2, T2::u> *--this is not necessarily compatible // with A<T2, T2::u::v> * and it does not need	This is a test of the interpretation of the resolution to Issue 214. In particular, we would like for the committee to spell out how properties of the synthetics produced for partial ordering are determined	

1 **MB** = Member body (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	(3)	4	5	(6)	(7)
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Secretariat observations** on each comment submitted

				to be. See Note 1. The effect is that // (in the call below) the compatibility of B::u and B::u::v is respected. // T in #2 cannot be successfully deduced in partial ordering from A<T1, T1::u> *; // invalid type T1::u::v will be formed when T1 is substituted into non-deduced contexts. struct B { struct u { typedef u v; }; }; int main() { foo((A<B, B::u> *)0); // calls #2 } *Note 1: Template argument deduction is an attempt to match a P and to a deduced A; however, template argument deduction is not specified to fail if the P and the deduced A are incompatible. This may occur in the presence of non-deduced contexts. Notwithstanding the parenthetical statement in [temp.deduct.partial] paragraph 9, template argument deduction may succeed in determining a template argument for every template parameter while producing a deduced A that is not compatible with the corresponding P.
				
				Example 2:
				
				template <typename T, typename U, typename V> struct A; template <typename T> void foo(A<T, struct T::u, struct T::u::u> *); #2.1 // synthetic T1 has member non-union class T1::u template <typename T, typename U> void foo(A<T, U , U> *); #2.2 // synthetic T2 and U2 has no required properties // T in #2.1 cannot be deduced in partial ordering from A<T2, U2, U2> *;// invalid types T2::u and T2::u::u will be formed when T2 is substituted in non-deduced contexts. // T and U in #2.2 deduces to, respectively, T1 and T1::u from A<T1, T1::u, struct T1::u::u> * unless // struct T1::u::u does not refer to the injected-class-name of the class T1::u (if that is possible). struct B { struct u { }; }; int main() { foo((A<B, B::u, struct B::u::u> *)0); // calls #2.1 } It is however unclear to what extent an implementation will have to go to determine these minimal properties.		

US 72	14.5.7 [temp.alias]		ge	Template aliases have no implementation experience.	Either demonstrate a complete implementation of this feature or remove N2258 from the working	

1 **MB** = Member body (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	(3)	4	5	(6)	(7)
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Secretariat observations** on each comment submitted
					paper prior the FDIS.	

US 73	14.5.7	1	te	The current wording of 7.1.3¶2 requires that the *identifier* in an *alias-declaration* "...shall not appear in the *type-id*." With template aliases, however, the name of the alias can be used indirectly:
				
				template<typename T> struct A;
				template<typename T> using B=typename A<T>::U;
				template<typename T> struct A {
				typedef B<T> U;
				};
				B<short> b;
				
				Here the instantiation of B<short> causes the instantiation of A<short>, and the typedef in A<short> ends up attempting to use B<short>, which is still in the process of being instantiated.
					Add wording to indicate that such usages in template aliases are ill-formed.	

US 74	14.5.7	1	te	An *alias-declaration* allows a class or enumeration type to be defined in its *type-id* (7.1.6¶3). However, it's not clear that this is desirable when the *alias-declaration* is part of a template alias: template<typename T> using A = struct { void f(T) { } };	Either prohibit the definition of classes and enumerations in template aliases, or prohibit the use of template parameters in such definitions, or add an example illustrating this usage.	
FI 10	14.5.7 [temp.alias]		te	Can template aliases be declared in class scope?	Allow declaring template aliases in class scope, if not allowed by the current grammar.	
FI 11	14.5.7 [temp.alias]		te	We have class templates and function templates, shouldn't we call template aliases alias templates for consistency?	Change “template alias” -> “alias template” everywhere.	
JP 25	14.5.7	1	E	Representations of reference link are not unified. Most reference links to clause (table) number, say X, are in the form "Clause X" ("Table X") capitalized, and subsection Y.Y.Y is referenced with its number only in the form "Y.Y.Y". Whether they are parenthesized or not	Change "(clause 7)" to "(Clause 7)".	

1 **MB** = Member body (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	(3)	4	5	(6)	(7)
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Secretariat observations** on each comment submitted
				depends on the context. However there are some notations "(Z)" consisting of only a number Z in parentheses to confer Clause or Table number Z.		
JP 26	14.6.2.1	1	E	Representations of reference link are not unified. Most reference links to clause (table) number, say X, are in the form "Clause X" ("Table X") capitalized, and subsection Y.Y.Y is referenced with its number only in the form "Y.Y.Y". Whether they are parenthesized or not depends on the context. However there are some notations "(Z)" consisting of only a number Z in parentheses to confer Clause or Table number Z.	Change "(9)" to "(Clause 9)".	

GB 43	14.6.2.1p3		Te	If I'm not missing something, 14.6.2.1/3 says that in the definition of a class template or member of a class template, the injected-class-name refers to the current instantiation. The template name followed by the argument list enclosed in "<..>" also refers to the current instantiation, but only in the definition of a primary class template. That results in an odd situation:
				
				template<typename T> struct A { typedef int type; void f(type); }; // here we are outside the definition of "A"
				
				template<typename T> void A::f(A::type) { } // OK: "A" is the injected-class-name
				
				template<typename T> void A::f(A<T>::type) { } // ill-formed: "A<T>" is not the injected-class-name. Needs "typename "!
				
				If you would define the member-function within the primary class template, bullet 2 would apply:
				
				template<typename T> struct A {
					Updated 14.6.2.1 [temp.dep.type] p1 bullet 2:
					"— in the definition of a primary class template or a member of a class template, the name of the class template followed by the template argument list of the primary template (as described below) enclosed in <>,"	

1 **MB** = Member body (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	(3)	4	5	(6)	(7)
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Secretariat observations** on each comment submitted

				typedef int type; void f(A<T>::type) { } // OK: name of A followed by arguments enclosed in <..>
				};
				
				I think that this state of affairs isn't any good.
				
				-> Suggested solution: Change 14.6.2.1/1 bullet2 to apply also to members of the primary class template. The same for bullet4 talking about partial specializations. Since partial specializations are also class templates, i wonder whether one could also smelt together bullet2 and bullet4 and only talk about "class template".		

GB 44	14.6p5		Te	C++0x does not allow this code template<typename T> struct id { typedef T type; }; template<typename T> void f() { int id<T>::type::*p = 0; } struct A { }; int main() { f<A>(); } The reason is that it requires "typename" before "id<T>::type", but "typename" is not allowed at that place by the syntax. Ultimately, current compilers accept this.	Change 14.6/5 to A qualified name used as the name in a mem-initializer-id, a base-specifier, an elaborated-type-specifier or the nested-name-specifier of a pointer-to-member declarator is implicitly assumed to name a type, without the use of the typename keyword.	
CA 6	14.6p5	P5	te	Given the change in N1376=02-0034 to [temp.res], found in FCD in [temp.res] paragraph 5: A qualified name used as the name in a mem-initializer-id, a base-specifier, or an elaborated-type-specifier is implicitly assumed to name a type, without the use of the typename keyword the following appears to be well-formed, with templates foo() being distinct since any type T will produce an invalid type for the second parameter for at least one foo() when T is replaced within the non-deduced context:	Please clarify as the following case appears to be expensive to implement with little functional value to the language.	

1 **MB** = Member body (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	(3)	4	5	(6)	(7)
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Secretariat observations** on each comment submitted
				template <typename T> bool *foo(T *, enum T::u_type * = 0) { return 0; } template <typename T> char *foo(T *, struct T::u_type * = 0) { return 0; } struct A { enum u_type { I }; }; int main(void) { foo((A*)0); } In particular, while determining the signature (1.3.11 [defns.signature]) for the function templates foo(), an elaborated-type-specifier qualifies as part of the decl-specifier-seq under 8.3.5 [dcl.fct] paragraph 5 in determining the type of a parameter in the parameter-type-list (absent additional wording). Also, the return type is included in the signature of a function template. A portion of the GCC 4.5.0 output: Internal compiler error: Error reporting routines re-entered. Please submit a full bug report, with preprocessed source if appropriate. See < http://gcc.gnu.org/bugs.html > for instructions. Implementations do not appear to support this case and the ability to do so brings little value since type traits such as is_enum and is_class cannot be defined using this and equivalent functionality can be achieved using the aforementioned type traits. template <typename T> struct MY_is_enum : std::false_type { }; template <typename T> struct MY_is_enum<enum T> : std::true_type { }; // ill-formed, elaborated-type-specifier resolves to typedef-name		

1 **MB** = Member body (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	(3)	4	5	(6)	(7)
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Secretariat observations** on each comment submitted
US 75	14.7		TE	As described in c++std-core-16425 and its followup messages, writing metaprograms is needlessly hard because specializing template members inside a class is (inadvertently?) not permitted. In addition, this surprising restriction makes C++ less simple and more arbitrary-seeming.	Accept the code like that in c++std-core-16425, like Visual C++ does already	

JP 79	14.7.1	10	E	"char*" should be "const char *".
				The special rule to convert character literal to pointer has been removed from "4.2 Array-to-pointer conversion [conv.array]".
				char * p1 = "..." ; // ill-formed.(removing const'ness)
				char const *p2 = "..." ;// well-formed.
				There are many code fragments depending on the removed rule. They are ill-formed.
					Change to:
					namespace N {
					template<class T> class List {
					public:
					T* get();
					};
					}
					template<class K, class V> class Map {
					public:
					N::List<V> lt;
					V get(K);
					};
					void g(Map<const char*,int>& m) {
					int i = m.get("Nicholas");
					}
					a call of lt.get() from Map<const char*,int>::get() would place List<int>::get() in the namespace N rather
					than in the global namespace. —end example]	

FI 9	14.7.3 [temp.expl.spec]	Paragraph 2	te	Explicit specializations in class scope inside class templates should be allowed. It's weird, confusing and inconsistent that they can be declared/defined in some scopes but not in others.	Allow explicit specialization of member templates inside class templates.	

JP 80	14.8.1	5	E	"char*" should be "const char *".
				The special rule to convert character literal to pointer has been removed from "4.2 Array-to-pointer conversion
					Change to:
					template<class X, class Y, class Z> X f(Y,Z);
					template<class ... Args> void f2();	

1 **MB** = Member body (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	(3)	4	5	(6)	(7)
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Secretariat observations** on each comment submitted

				[conv.array]".
				char * p1 = "..." ; // ill-formed.(removing const'ness)
				char const *p2 = "..." ;// well-formed.
				There are many code fragments depending on the removed rule. They are ill-formed.
					void g() {
					f<int,const char*,double>("aa",3.0);
					f<int,const char*>("aa",3.0); // Z is deduced to be double
					f<int>("aa",3.0); // Y is deduced to be const char*, and
					// Z is deduced to be double
					f("aa",3.0); // error: X cannot be deduced
					f2<char, short, int, long>(); // OK
					}	

US 76	14.8.2	para 9	te	"extern template" prevents inlining functions not marked inline.	Remove wording about "suppressing the implicit instantiation". See Appendix 1 - Additional Details	
US 77	14.8.2.1		te	Core Issue 1014 claims that calling f(const T&) and f(T&&) with a const int lvalue is ambiguous. It's unambiguous because the partial ordering rules consider f(const T&) to be more specialized than f(T&&), for the same reasons that they consider h(const T&) to be more specialized than h(T&). However, calling z(T&) and z(T&&) with an int lvalue is ambiguous. Because z(T&) accepts a strict subset of the things that z(T&&) accepts, it seems that the partial ordering rules should be modified to consider z(T&) to be more specialized than z(T&&). There may be additional subtleties.	Modify the partial ordering rules to consider z(T&) to be more specialized than z(T&&).	
US 78	15.2	2	te	This paragraph says that "An object that is... partially destroyed will have destructors executed... for subobjects for which the principal constructor (12.6.2) has completed execution and the destructor has not yet begun execution." This would presumably apply to an example like struct S { ~S(); } s[10];	Clarify the intent with respect to array elements and storage duration.	

1 **MB** = Member body (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	(3)	4	5	(6)	(7)
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Secretariat observations** on each comment submitted
				If the destructor for s[5] throws an exception, elements 0-4 should still be destroyed. However, the wording specifically refers to "fully constructed base classes and non-variant members," even though array elements are subobjects of the array (1.8¶2). This is presumably the effect of stack unwinding (¶1), which applies to "all automatic objects constructed since the try block was entered," but whether that should also be expected for arrays of static, thread, and dynamic storage duration is not clear.		
GB 45	15.3	16	Te	The phrasing of this clause suggests all exception-declarations produce objects. There should be some additional wording to clarify that exception-declarations that declare references bind to the exception object by appropriate initialization, and *are* allowed to be references to abstract classes. Likewise, the elipsis form does not initialize any object or temporary.	Distinguish between initializing objects, initializing references, and initializing nothing in the case of an elipsis.	
CA 5	15.3p8, 15.1p7	P8, p7	te	There is an issue with the definition of "currently handled exception" in 15.3 [except.handle] paragraph 8: > The exception with the most recently activated handler that is still active is called the *currently handled exception*. This wording implies that the currently handled exception may be changed by another thread. Thus, by 15.1 [except.throw] paragraph 7, > A throw-expression with no operand rethrows the currently handled exception (15.3). the following may throw an exception that is not of type int. try { throw 0; } catch (...) { throw; } Any solution should also specify what the currently handled exception will be for a thread that is spawned during the handling of an exception by its parent.	Clarify and fix as suggested.	

1 **MB** = Member body (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	(3)	4	5	(6)	(7)
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Secretariat observations** on each comment submitted
US 79	15.4	¶1	te	Because C has no exception mechanism, functions having “C” language linkage should implicitly be declared noexcept.	Insert a sentence such as: Any function declared to have “C” linkage shall be treated as if declared noexcept even if the declaration lacks the noexcept keyword.	
GB 46	15.4		Te	It is not entirely clear that a function-try-block on a destructor will catch exceptions for a base or member destructor destructor; whether such exceptions might be swallowed with a simple return statement rather than being rethrown; and whether such a clause might be entered multiple times if multiple bases/members throw, or if that is an automatic terminate call.	[Add words to 15.4 clarifying the problem cases.]	
CH 10	15.4	p9	te	In case of incorrect program specification, the general rule is that the behaviour is undefined. This should be true for noexcept as well.	Change the second bullet of p9 to "otherwise, the behaviour is undefined.	
GB 47	15.5.1	1	Te	15.5.1:1 [except.terminate] lists the situations in which "exception handling must be abandoned for less subtle error handling techniques". The list seems to omit some new situations added by other c++0x features.	The list should be augmented with the following: * when function std::nested_exception::rethrow_nested is called for an object that stores a null exception pointer. * when execution of a function registered with std::at_quick_exit exits using an exception. Also, after the list, add the following sentence: Function std::terminate is also called by the implementation, when the destrductor or a copy constructor of a class std::thread is called for the object that is joinable.	
GB 48	15.5.2	1-4	Te	This subclause is dealing exclusively with dynamic exception specifications, and should clearly say so.	Replace each italicised occurrence of 'exception-specification' with 'dynamic-exception-specification' in clause 15.5.2, [except.unexpected]	

1 **MB** = Member body (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	(3)	4	5	(6)	(7)
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Secretariat observations** on each comment submitted
GB 49	15.5.2	all	Ge	Dynamic exception specifications are deprecated, so clause 15.5.2 that describes how they work should move to Annex D.	Move 15.5.2 [except.unexpected] to Annex D.	
CH 11	16.1	p3	ed	constant-expression as given by C++'s grammar allows far more syntactical constructs than those that are allowed in preprocessor context. The footnote 145 provides some hints on the limitations, but in our opinion it should be made more clear that the "constant-expression" allows less than a lookup of the corresponding grammar rule suggests	Add a note or extend footnote 145 with "Note that constant-expression is much more limited than the C++ grammar rule would suggest. See the following paragraphs how it is limited in the context of conditional inclusion."	
CH 12	16.3.5	p5	ed	missing space between '~' and '5' in expansion	line "`f(2 * (2+(3,4)-0,1)) | f(2 * (~5)) & f(2 * (0,1))^m(0,1);`" should read "`f(2 * (2+(3,4)-0,1)) | f(2 * (~ 5)) & f(2 * (0,1))^m(0,1);`"	
CH 13	16.3.5	p7	ed	superfluous braces in source	change "`int j[] = { t(1,2,3), t(,4,5), t(6,{,}7), t(8,9,),`" to "`int j[] = { t(1,2,3), t(,4,5), t(6,,7), t(8,9,),`"	

CH 14	16.3.5	p9	ed	superfluous spaces after/before parentheses	change
					`fprintf(stderr, "Flag" );`
					`fprintf(stderr, "X = %d\n", x );`
					`puts( "The first, second, and third items." );`
					to
					`fprintf(stderr, "Flag");`
					`fprintf(stderr, "X = %d\n", x);`
					`puts("The first, second, and third items.");`	

DE 13	16.8		te	Committee Draft comment DE 18 has only been partially addressed, and the record of response ignores the missing item, namely the absence of a macro __STDCPP_STRICT_POINTER_SAFETY__ that indicates that a given implementation has strict pointer safety (see 3.7.4.3).	Add the macro to the list of predefined macros in 16.8.	

1 **MB** = Member body (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	(3)	4	5	(6)	(7)
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Secretariat observations** on each comment submitted
US 80	[library] 17 - 30		te	Consider applying noexcept to all of the std::lib.		
GB 60	17-30		Ge	Dyanamic exception specifications are deprecated; the library should recognise this by replacing all non-throwing exception specifications of the form 'throw()' with the 'noexcept' form.	Replace all non-throwing exception specifications of the form 'throw()' with the 'noexcept' form.	
GB 61	17-30		Te	All library types should have non-throwing move constructors and move-assignment operators unless wrapping a type with a potentially throwing move-operation. When such a type is a class-template, these operations should have a conditional noexcept specification. There are many other places where a noexcept specification may be considered, but the move operations are a special case that must be called out, to effectively support the move_if_noexcept function template.	Review every class and class template in the library. If noexcept move constructor/assignment operators can be implicitly declared, then they should be implicitly declared, or explicitly defaulted. Otherwise, a move constructor/move-assingment operator with a 'noexcept' exception specification should be provided.	
GB 62	17-30		Te	Issues with efficiency and unsatisfactory semantics mean many library functions document they do not throw exceptions with a Throws: Nothing clause, but do not advertise it with an exception specification. The semantic issues are largely resolved with the new 'noexcept' specifications, and the noexcept operator means we will want to detect these guarantees programatically in order to construct programs taking advantage of the guarantee.	Add a 'noexcept' exception specification on each libary API that offers an unconditional Throws: Nothing guarantee. Where the guarantee is conditional, add the appropriate noexcept(constant-expression) if an appropriate constant expression exists.	
GB 63	17-30		Ge	Since the newly introduced operator noexcept makes it easy (easier than previously) to detect whether or not a function has been declared with the empty exception specification (including noexcept) library functions that cannot throw should be decorated with the empty exception specification. Failing to do so and leaving it as a matter of QoI would be detrimental to portability and efficiency.	Review the whole library, and apply the noexcept specification where it is appropriate.	

1 **MB** = Member body (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	(3)	4	5	(6)	(7)
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Secretariat observations** on each comment submitted
GB 64	17-30		Ge	There are a number of unspecified types used throughout the library, such as the container iterators. Many of these unspecified types have restrictions or expectations on their behaviour in terms of exceptions. Are they permitted or required to use exception specifications, more specifically the new noexcept specification? For example, if vector<T>::iterator is implemented as a native pointer, all its operations will have an (effective) nothrow specification. If the implementation uses a class type to implement this iterator, is it permitted or required to support that same guarantee?	Clearly state the requirements for exception specifications on all unspecified library types. For example, all container iterator operations should be conditionally noexcept, with the condition matching the same operation applied to the allocator pointer_type, a certain subset of which are already required not to throw.	
GB 65	17-30		Te	Nothrowing swap operations are key to many C++ idioms, notably the common copy/swap idiom to provide the strong exception safety guarantee.	Where possible, all library types should provide a swap operation with an exception specification guaranteeing no exception shall propagate. Where noexcept(true) cannot be guaranteed to not terminate the program, and the swap in questions is a template, an exception specification with the appropriate conditional expression could be specified.	
GB 66	17-30		Ed	The syntax for attributes was altered specifically to make it legal do declare attributes on functions by making the attribute the first token in the declaration, rather than inserting it between the function name and the opening paren of the parameter list. This is much more readable, and should be the preferred style throughout the library. For example, prefer: [[noreturn]] void exit(int status); to void exit [[noreturn]] (int status);	Update all function declarations throughout the library to use the preferred placement of function-level attributes.	
CH 15	Library clauses		te	Due to the new rules about implicit copy and move constructors some library facilities are now move-only.	Make them copyable again.	
CH 16	Library clauses		te	Dynamic exception specifications are deprecated. Deprecated features shouldn't be used in the Standard.	Replace dynamic exception specifications with noexcept.	
CH	Library		te	The introduction of noexcept makes "Throws: Nothing"	Consider replacing "Throws: Nothing." clause by	

1 **MB** = Member body (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	(3)	4	5	(6)	(7)
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Secretariat observations** on each comment submitted
17	clauses			clauses looking strange.	the respective noexcept specification.	
CH 18	17		te	The general approach on moving is that a library object after moving out is in a "valid but unspecified state". But this is stated at the single object specifications, which is error prone (especially if the move operations are implicit) and unnecessary duplication.	Consider putting a general statement to the same effect into clause 17.	
JP 27	17.1	9	E	Representations of reference link are not unified. Most reference links to clause (table) number, say X, are in the form "Clause X" ("Table X") capitalized, and subsection Y.Y.Y is referenced with its number only in the form "Y.Y.Y". Whether they are parenthesized or not depends on the context. However there are some notations "(Z)" consisting of only a number Z in parentheses to confer Clause or Table number Z.	Change "(25)" to "(Clause 25)".	
GB 52	17.3.7		Te	The definition of deadlock in 17.3.7 excludes cases involving a single thread making it incorrect.	The definition should be corrected.	
GB 50	17.3.13		Te	This definition of move-assignment operator redundant and confusing now that the term move-assignment operator is defined by the core language in subclause 12.8p21.	Strike suclause 17.3.13 [defns.move.assign.op]. Add a cross-reference to (12.8) to 17.3.12.	
GB 51	17.3.14		Te	This definition of move-constructor redundant and confusing now that the term constructor is defined by the core language in subclause 12.8p3.	Strike subclause 17.3.14, [defns.move.ctor]	

JP 67	17.5.2.1.2	2	E	In some code examples, ellipsis(…) is used in ill-formed. In these cases, "…" represents omission of some codes like this:
				class A { /* ... */ } ;
				But in some cases, it is used without commented-out as below:
				class A { ... } ;
				It is an inconsistent usage. They all should be enclosed in a comment.
					Change to:
					enum bitmask {
					V0 = 1 << 0, V1 = 1 << 1, V2 = 1 << 2, V3 = 1 << 3, ...
					};
					static const bitmask C3 (V3);
					/* ... */	

1 **MB** = Member body (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	(3)	4	5	(6)	(7)
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Secretariat observations** on each comment submitted
				In addition, the number of period should be 3 rather than 5.		
GB 53	17.5.2.1.3		Te	The bitmask types defined in 27.5.2 and 28.5 contradict the bitmask type requirements in 17.5.2.1.3, and have missing or incorrectly defined operators.	See Appendix 1 - Additional Details	
GB 54	17.5.2.1.4.x		Ge	The defined terms NTC16S, NTC32S, NTWCS, char16-character sequence, null-terminated char16-character string, char32-character sequence, null-terminated char32-character string, wide-character sequence and null-terminated wide-character string do not occur at any point in the standard outside their definitional subclauses and associated footnotes.	Strike 17.5.2.1.4.3, 17.5.2.1.4.4, 17.5.2.1.4.5 and associated footnotes 170, 171, 172, 173, 174 and 175.	
GB 55	17.6.1.3	Table 15	Te	The thread header uses duration types, found in the <chrono> header, and which rely on the ratio types declared in the <ratio> header.	Add the <chrono> and <ratio> headers to the freestanding requirements. It might be necessary to address scaled-down expectations of clock support in a freestanding environment, much like <thread>.	
GB 56	17.6.1.3	Table 15	Ge	The <utility> header provides support for several important C++ idioms with move, forward and swap. Likewise, declval will be frequently used like a type trait. In order to complete cycles introduced by std::pair, the <tuple> header should also be made available. This is a similarly primitive set of functionality, with no dependency of a hosted environment, but does go beyond the minimal set of functionality otherwise suggested by the freestanding libraries. Alternatively, split the move/forward/swap/declval functions out of <utility> and into a new primitive header, requiring only that of freestanding implementation.	Add <utility> and <tuple> to table 15, headers required for a free-standing implementation.	
GB 57	17.6.1.3	Table 15	Te	The atomic operations facility is closely tied to clause 1 and the memory model. It is not easily supplied as an after-market extension, and should be trivial to implement	Add <atomic> to table 15, headers required for a free-standing implementation.	

1 **MB** = Member body (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	(3)	4	5	(6)	(7)
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Secretariat observations** on each comment submitted
				of a single-threaded serial machine. The consequence of not having this facility will be poor interoperability with future C++ libraries that memory model concerns seriously, and attempt to treat them in a portable way.		
GB 58	17.6.2		Te	It is not clear whether a library header specified in terms of a typedef name makes that same typedef name available for use, or if it simply requires that the specified type is an alias of the same type, and so the typedef name cannot be used without including the specific header that defines it. For example, is the following code required to be accepted: #include <vector> std::size_t x = 0; Most often, this question concerns the typedefs defined in header <cstddef>	Add a paragraph under 17.6.2 clarifying whether or not headers specified in terms of std::size_t can be used to access the typedef size_t, or whether the header <cstddef> must be included to reliably use this name.	

GB 59	17.6.3.6	2	Ed	The replaceable functions in header <new> are all described in clause 18.6 [support.dynamic], where it can be seen that all the listed functions have an exception specification which must be compatible with any replacement function.	Narrow the reference to (Clause 18) to (Clause 18.6).
					Add the missing exception specification on each function signature:
					void* operator new(std::size_t size) throw(std::bad_alloc);
					void* operator new(std::size_t size, const std::nothrow_t&) throw();
					void operator delete(void* ptr) throw();
					void operator delete(void* ptr, const std::nothrow_t&) throw();
					void* operator new[](std::size_t size) throw(std::bad_alloc);
					void* operator new[](std::size_t size, const std::nothrow_t&) throw();
					void operator delete[](void* ptr) throw();
					void operator delete[](void* ptr, const std::nothrow_t&) throw();
					(note that other comments might further want to replace 'throw()' with 'noexcept')	

1 **MB** = Member body (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

NOTE Columns 1, 2, 4, 5 are compulsory.

Template for comments and secretariat observations

Date: 5 Oct 2010	Document: **N3141 FCD 14882 Ballot Comments**

1	2	(3)	4	5	(6)	(7)
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Secretariat observations** on each comment submitted
US 81	17.6.3.9	¶1	ed	International Standards do not make “statements”; they “specify” or “require” instead.	s/statements/ specifications/	
US 82	17.6.3.9	¶1, bullet 3	te	The second Note can benefit by adopting recent nomenclature.	Rephrase the Note in terms of *xvalue*.	
US 83	17.6.3.10	¶2, first sent.	ed	The word “constructor” is misspelled.	s/contructor/constructor/	
GB 68	18.2	4	Te	There is no reason for the offsetof macro to invoke potentially throwing operations, so the result of noexcept(offsetof(type,member-designator)) should be true.	Add to the end of 18.2p4: "No operation invoked by the offsetof macro shall throw an exception, and noexcept(offsetof(type,member-designator)) shall be true."	
JP 68	18.3.1.5	2	E	In some code examples, ellipsis(…) is used in ill-formed. In these cases, "…" represents omission of some codes like this: class A { /* ... */ } ; But in some cases, it is used without commented-out as below: class A { ... } ; It is an inconsistent usage. They all should be enclosed in a comment. More over, in this case, "implementation-defined" would be better than "…".	Change to: inline static constexpr float infinity() throw() { return /* ... */; } inline static constexpr float quiet_NaN() throw() { return /* ... */; } inline static constexpr float signaling_NaN() throw() { return /* ... */; }	
JP 84	18.5	5 note	E	Note in paragraph 5 says "the atexit() functions shall not introduce a data race" and Note in paragraph 10 says "the at_quick_exit() functions do not introduce ...". Such different expressions in similar functions are confusing. If these notes are written for unspecified behaviors just before the sentence, "do" would be preferred.	Replace "shall" with "do".	
GB 69	18.5	14	Ed	("The function quick_exit() never returns to its caller.") should be removed as redundant. The function is already attributed with [[noreturn]].	Remove paragraph 14	

1 **MB** = Member body (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	(3)	4	5	(6)	(7)
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Secretariat observations** on each comment submitted
CH 19	18.8.5		te	It's not clear how exception_ptr is synchronized.	Make clear that accessing in different threads multiple exception_ptr objects that all refer to the same exception introduce a race.	
GB 70	18.6		Te	std::nothrow_t is a literal type (being an empty POD) so the preferred form of declaration for std::nothrow is as a constexpr literal, rather than an extern symbol.	Replace: extern const nothrow_t nothrow; with constexpr nothrow_t nothrow{};	
GB 71	18.6.2.4 / 18.8.2.2 / 18.8.3.2		Te	The thread safety of std::set_new_handler(), std::set_unexpected(), std::set_terminate(), is unspecified making the the functions impossible to use in a thread safe manner.	The thread safety guarantees for the functions must be specified and new interfaces should be provided to make it possible to query and install handlers in a thread safe way.	
DE 14	18.6.1.4		te	It is unclear how a user replacement function can simultaneously satisfy the race-free conditions imposed in this clause and query the new-handler in case of a failed allocation with the only available, mutating interface std::set_new_handler.	Offer a non-mutating interface to query the current new-handler.	
GB 72	18.8.2		Ge	Dynamic exception specifications are deprecated, so clause 18.8.2 that describes library support for this facility should move to Annex D, with the exception of the bad_exception class which is retained to indicate other failures in the exception dispatch mechanism (e.g. calling current_exception()).	With the exception of 18.8.2.1 [bad.exception], move clause 18.8.2 diectly to Annex D. [bad.exception] should simply become the new 18.8.2.	
GB 73	18.8.4		Te	The thread safety std::uncaught_exception() and the result of the function when multiple threads throw exceptions at the same time are unspecified. To make the function safe to use in the presence of exceptions in multiple threads the specification needs to be updated.	Update this clause to support safe calls from multiple threads without placing synchronization requirements on the user.	
GB 74	18.8.5	10	Te	One idea for the exception_ptr type was that a reference-counted implementation could simply 'reactivate' the same exception object in the context of a call to 'rethrow_exception'. Such an implementation would allow the same exception object to be active in multiple threads (such as when multiple threads join on a shared_future) and introduce potential data races in any exception handler that catches exceptions by reference - notably existing library code written before this capability was	Throws: a copy of the exception object to which p refers.	

1 **MB** = Member body (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	(3)	4	5	(6)	(7)
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Secretariat observations** on each comment submitted
				added. 'rethrow_exception' should *always* make a copy of the target exception object.		
US 84	18.8.6 [except.neste d]	6-7	te	The throw_with_nested spec passes in its argument as T&& (perfect forwarding pattern), but then discusses requirements on T without taking into account that T may be an lvalue-reference type. It is also not clear in the spec that t is intended to be perfectly forwarded.	Patch 6-7 to match the intent with regards to requirements on T and the use of std::forward<T>(t).	
GB 75	19		Te	None of the exception types defined in clause 19 are allowed to throw an exception on copy or move operations, but there is no clear specification that the operations have an exception specification to prove it. Note that the implicitly declared constructors, taking the exception specification from their base class (ultimately std::exception) will implicitly generate a noexcept exception specification if all of their data members similarly declare noexcept operations. As the representation is unspecified, we cannot assume non-throwing operations unless we explicitly state this as a constraint on the implementation.	Add a global guarantee that all exception types defined in clause 19 that rely on implicitly declared operations have a non-throwing exception specification on those operations.	
GB 76	19.5.1.5		Te	The C++0x FCD recommends, in a note (see 19.5.1.1/1), that users create a single error category object for each user defined error category and specifies error_category equality comparsions based on equality of addresses (19.5.1.3). The Draft apparently ignores this when specifying standard error category objects in section 19.5.1.5, by allowing the generic_category() and system_category() functions to return distinct objects for each invocation.	Append a new sentence to 19.5.1.5 [syserr.errcat.objects]/1, which reads "All calls of this function return references to the same object.", and append the same sentence to 19.5.1.5/3.	
GB 77	19.5.6.2	14	Ed	The description for system_error::what (19.5.6.2/14) changed between the C++ Working Papers N2914 and N2960. The latter document indicates	Remove the extra words from 19.5.6.2 [syserr.syserr.members]/14: "Returns: A NTBS incorporating the arguments supplied in the	

1 **MB** = Member body (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	(3)	4	5	(6)	(7)
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Secretariat observations** on each comment submitted
				that the Returns clause shall read "Returns: An NTBS incorporating the arguments supplied in the constructor.". Instead, in the FCD it now reads "Returns: An NTBS incorporating and code().message() the arguments supplied in the constructor.".	constructor."	
GB 78	19.5.6.2		Te	The FCD contains a requirement that all standard classes which are derived from std::exception have a copy constructor and copy assignment operator which essentially copy the stored what() message (See 18.8.1/2). In contrast, it is unspecified whether copies of system_error return the same error_code on calls to system_error::code().	Insert a new paragraph after 19.5.6.1 [syserr.syserr.overview]/1 which reads "The copy constructor and copy assignment operator of class system_error both have a strengthened postcondition which supplements the basic postcondition for standard library exception classes copy construction and copy assigment (18.8.1): If two objects lhs and rhs both have type system_error and lhs is a copy of rhs, then lhs.code() == rhs.code() shall hold."	
GB 79	20, 22, 24, 28		Te	The library provides several traits mechanisms intended a customization points for users. Typically, they are declared in headers that are growing quite large. This is not a problem for standard library vendors, who can manage their internal file structure to avoid large dependencies, but can be a problem for end users who have no option but to include these large headers.	Move the following traits classes into their own headers, and require the existing header to #include the traits header to support backwards compatibility: iterator_traits (plus the iterator tag-types) allocator_traits pointer_traits char_traits regex_traits	
US 85	20.2.1	Table 34		20.2.1 Table 34 "MoveConstructible requirements" says "Note: rv remains a valid object. Its state is unspecified". Some components give stronger guarantees. For example, moved-from shared_ptrs are guaranteed empty (20.9.11.2.1/25). In general, what the standard really should say (preferably as a global blanket statement) is that moved-from objects	State as a general requirement that moved-from objects can be destroyed and can be the destination of an assignment. Any other use is undefined behavior.	

1 **MB** = Member body (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	(3)	4	5	(6)	(7)
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Secretariat observations** on each comment submitted
				can be destroyed and can be the destination of an assignment. Anything else is radioactive. For example, containers can be "emptier than empty". This needs to be explicit and required generally. Note: The last time that one of us mentioned "emptier than empty" (i.e. containers missing sentinel nodes, etc.) the objection was that containers can store sentinel nodes inside themselves in order to avoid dynamically allocating them. This is unacceptable because (a) it forces existing implementations (i.e. Dinkumware's, Microsoft's, IBM's, etc.) to change for no good reason (i.e. permitting more operations on moved-from objects), and (b) it invalidates end iterators when swapping containers. (The Working Paper currently permits end iterator invalidation, which we consider to be wrong, but that's a separate argument. In any event, *mandating* end iterator invalidation is very different from permitting it.)		
GB 80	20.2.3	2	Te	See (A) in attachment std_issues.txt	as stated in the attached paper	
CA 10	20.2.3p2	20.2.3p2	te	Reads of indeterminate value result in undefined behaviour In 20.2.3p2, NullablePointer requirements [nullablepointer.requirements], the standard specifies the behaviour of programs that read indeterminate values: ... A default-initialized object of type P may have an indeterminate value. [Note: Operations involving indeterminate values may cause undefined behaviour. end note] We suggest changing the note to:	In 20.2.3p2, the standard specifies the behaviour of programs that read indeterminate values: ... A default-initialized object of type P may have an indeterminate value. [Note: Operations involving We suggest changing the note to: [Note: Operations involving indeterminate values cause undefined behaviour. end note]	

1 **MB** = Member body (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	(3)	4	5	(6)	(7)
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Secretariat observations** on each comment submitted
				[Note: Operations involving indeterminate values cause undefined behaviour. end note] Rationale: The note uses the word "may", but we believe the intention is that such reads will cause undefined behaviour, but implementations are not required to produce an error. Clark adds: > Unfortunately, this issue goes deeper than can be > addressed by deleting the word "may" from a note in > clause 20. The term "indeterminate value" and its > meaning were introduced in C99. While the term is > generally understood to be applicable to C++ (and > section 20.2.3 reflects that), the term isn't actually > defined in the C++ WD, and worse yet, there's no > statement that use of an indeterminate value can result > in undefined behavior (so the existing note can't be > deduced from the normative statements of the standard). > This is tracked by core issue 616. The wording in > 20.2.3 should be noted as relating to that issue, and		

1 **MB** = Member body (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	(3)	4	5	(6)	(7)
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Secretariat observations** on each comment submitted
				> should be handled as part thereof. Further on this, in the current draft standard, we can construct executions in which an atomic read happens-before the initialization of the atomic object, so there is no place to take a read value from. We imagine that such reads should also be of indeterminate values and result in undefined behaviour?		
US 86	20.2.5	Table 42	ed	In the row for X::propagate_on_container_move_assignment, the note says “copied” when it should say “moved”	Change the note as follows: true_type only if an allocator of type X should be ~~copied~~moved	
US 87	20.2.5	Table 42	te	reference_type should not have been removed from the allocator requirements. Even if it is always the same as value_type&, it is an important customization point for extensions and future features.	Add a row (after value_type) with columns: Expression: X::reference_type Return type: T& Assertion/note...: (empty) Default: T& [allocator.traits]: Add reference_type to allocator_traits template, defaulted to value_type&.	
US 88	20.2.5		Te	Allocator interface is not backward compatible.	See Appendix 1 - Additional Details	
US 89	20.3 [utility]		ed	make_pair is missing from the <utility> synopsis.	Add template <class T1, class T2> pair<V1, V2> make_pair(T1&&, T2&&); to the synopsis in [utility].	
GB 81	20.3	1	Ed	make_pair should be declared in the synopsis of <utility>	add to 20.3 [utility] paragraph 1: template see below make_pair(T1&&, T2&&);	
GB 82	20.3	1	Ed	The <utility> synopsis precedes the tuple_size and tuple_element declarations with a comment saying "//	correct the sub-clause reference for tuple-like access to 20.3.5.3, move the comment after the	

1 **MB** = Member body (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	(3)	4	5	(6)	(7)
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Secretariat observations** on each comment submitted
				20.3.5, tuple-like access to pair:" but the sub-clause should be 20.3.5.3 and the comment should probably be below those declarations (since they, like the tuple declaration above them, are defined in <tuple> and are not related to pair.) Also, there should be a comment above piecewise_construct_t giving the sub-clause 20.3.5.5, and it should be at the end of the synopsis,	tuple_element declaration, and add "// defined in <tuple>" to the tuple_size and tuple_element declarations. Move the piecewise_construct_t and piecewise_constrct declarations to the end of the synopsis and precede them with "// 20.3.5.5 Piecewise construction"	

US 90	20.3	3	te	In n3090, at variance with previous iterations of the idea discussed in papers and incorporated in WDs, std::forward is constrained via std::is_convertible, thus is not robust wrt access control. This causes problems in normal uses as implementation detail of member functions. For example, the following snippet leads to a compile time failure, whereas that was not the case for an implementation along the lines of n2835 (using enable_ifs instead of concepts for the constraining, of course)
				
				#include <utility>
				
				struct Base { Base(Base&&); };
				
				struct Derived
				
				: private Base
				
				{
				
				Derived(Derived&& d)
				
				: Base(std::forward<Base>(d)) { }
				
				};
				
				In other terms, LWG 1054 can be resolved in a better way, the present status is not acceptable.		

JP	20.3.1	2	E	Representations of reference link are not unified. Most reference links to clause (table) number, say X, are	Change "(31)" to "(Clause 31)".	

1 **MB** = Member body (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	(3)	4	5	(6)	(7)
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Secretariat observations** on each comment submitted
28				in the form "Clause X" ("Table X") capitalized, and subsection Y.Y.Y is referenced with its number only in the form "Y.Y.Y". Whether they are parenthesized or not depends on the context. However there are some notations "(Z)" consisting of only a number Z in parentheses to confer Clause or Table number Z.		
JP 29	20.3.1	4, 6, 8	E	Representations of reference link are not unified. Most reference links to clause (table) number, say X, are in the form "Clause X" ("Table X") capitalized, and subsection Y.Y.Y is referenced with its number only in the form "Y.Y.Y". Whether they are parenthesized or not depends on the context. However there are some notations "(Z)" consisting of only a number Z in parentheses to confer Clause or Table number Z.	Change "(32)" to "(Clause 32)".	
JP 30	20.3.2	1	E	Representations of reference link are not unified. Most reference links to clause (table) number, say X, are in the form "Clause X" ("Table X") capitalized, and subsection Y.Y.Y is referenced with its number only in the form "Y.Y.Y". Whether they are parenthesized or not depends on the context. However there are some notations "(Z)" consisting of only a number Z in parentheses to confer Clause or Table number Z.	Change "(34)" to "(Table 34)". Change "(36)" to "(Table 36)".	
US 91	Merged with US 90					
US 92	20.3.3		te	std::identity was removed from 20.3.3 [forward], apparently because std::forward() no longer needs it. However, std::identity was useful by itself (to disable template argument deduction, and to provide no transformation when one is requested).	Restore std::identity, possibly in a different section.	

1 **MB** = Member body (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	(3)	4	5	(6)	(7)
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Secretariat observations** on each comment submitted
US 93	20.3.3	6	ge	When teaching C++0x, students have consistently found the name std::move confusing because it doesn't actually move the object (it just makes it possible to move). It was also confusing for me.	Choose a name that expresses the semantics more clearly. Suggestion: std::unpin	
US 94	20.3.3	9	ed	Returns clause for move_if_noexcept d refers to a non-existent symbol, t, which should be x.	Modify the Returns clause: *Returns:* std::move(~~t~~x).	

DE 15	20.3.5.2, 20.4.2.1		te	Several function templates of pair and tuple allow for too many implicit conversions, for example:
				#include <tuple>
				std::tuple<char*> p(0); // Error?
				struct A { explicit A(int){} };
				A a = 1; // Error
				std::tuple<A> ta = std::make_tuple(1); // OK?
					Consider to add wording to constrain these function templates.	

JP 31	20.5.1	2	E	Representations of reference link are not unified. Most reference links to clause (table) number, say X, are in the form "Clause X" ("Table X") capitalized, and subsection Y.Y.Y is referenced with its number only in the form "Y.Y.Y". Whether they are parenthesized or not depends on the context. However there are some notations "(Z)" consisting of only a number Z in parentheses to confer Clause or Table number Z.	Change "(section 3.9)" to "(3.9)".	
GB 84	20.3.5.2		Ed	[pairs.pair] defines the class template pair as well as related non-member functions such as comparison operators and make_pair. The related non-member functions should be in a separate sub-clause, otherwise it's not clear that paragraphs below 17 do not refer to members of pair.	Create a new "Pair specialized algorithms" section containing everything below paragraph 17 in [pairs.pair]	
US 95	20.3.5.2	9	te	Copy-assignment for pair is defaulted and does not work for pairs with reference members. This is inconsistent with conversion-assignment, which deliberately succeeds even if one or both elements are reference types, just as for	Add to pair synopsis: pair& operator=(const pair& p);	

1 **MB** = Member body (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	(3)	4	5	(6)	(7)
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Secretariat observations** on each comment submitted
				tuple. The copy-assignment operator should be consistent with the conversion-assignment operator and with tuple's assignment operators.	Add before paragraph 9: pair& operator=(const pair& p); Requires: T1 and T2 shall satisfy the requirements of CopyAssignable. Effects: Assigns p.first to first and p.second to second. Returns: *this.	
DE 16	20.3.5.2, 20.4.2.1		te	Several pair and tuple functions in regard to move operations are incorrectly specified if the member types are references, because the result of a std::move cannot be assigned to lvalue-references. In this context the usage of the requirement sets MoveConstructible and CopyConstructible also doesn't make sense, because non-const lvalue-references cannot satisfy these requirements.	Replace the usage of std::move by that of std::forward and replace MoveConstructible and CopyConstructible requirements by other requirements.	
GB 85	20.3.5.4		Te	While std::pair may happen to hold a pair of iterators forming a valid range, this is more likely a coincidence than a feature guaranteed by the semantics of the pair template. A distinct range-type should be supplied to enable the new for-loop syntax rather than overloading an existing type with a different semantic.	Strike 20.3.5.4 and the matching declarations in 20.3 header synopsis. If a replacement facility is required for C++0x, consider n2995.	
ES 1	20.3.5.4 [pair.range]		Te	Using pair to represent a range of iterators is too general and does not provide additional useful restrictions (see N2995 and preceding papers).	Provide a separate template range<Iterator>.	
US 96	20.3.5.2 20.4.2.1 20.4.2.2	¶ 6-14 ¶ 6-20 ¶ 6-18	te	pair and tuple constructors and assignment operators use std::move when they should use std::forward. This causes lvalue references to be erroneously converted to rvalue references. Related requirements clauses are also wrong.	See Appendix 1 - Additional Details	
US 97	20.3.5.2 and 20.4.2		te	pair's class definition in N3092 20.3.5.2 [pairs.pair] contains "pair(const pair&) = default;" and "pair&	Either remove "pair(const pair&) = default;" and "pair& operator=(pair&& p);" from pair's class	

1 **MB** = Member body (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	(3)	4	5	(6)	(7)
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Secretariat observations** on each comment submitted
				operator=(pair&& p);". The latter is described by 20.3.5.2/12-13. "pair(const pair&) = default;" is a user-declared explicitly-defaulted copy constructor. According to 12.8 [class.copy]/10, this inhibits the implicitly-declared move constructor. pair should be move constructible. (12.8/7 explains that "pair(pair<U, V>&& p)" will never be instantiated to move pair<T1, T2> to pair<T1, T2>.) "pair& operator=(pair&& p);" is a user-provided move assignment operator (according to 8.4.2 [dcl.fct.def.default]/4: "A special member function is user-provided if it is user-declared and not explicitly defaulted on its first declaration."). According to 12.8/20, this inhibits the implicitly-declared copy assignment operator. pair should be copy assignable, and was in C++98/03. (Again, 12.8/7 explains that "operator=(const pair<U, V>& p)" will never be instantiated to copy pair<T1, T2> to pair<T1, T2>.) Additionally, "pair& operator=(pair&& p);" is unconditionally defined, whereas according to 12.8/25, defaulted copy/move assignment operators are defined as deleted in several situations, such as when non-static data members of reference type are present. If "pair(const pair&) = default;" and "pair& operator=(pair&& p);" were removed from pair's class definition in 20.3.5.2 and from 20.3.5.2/12-13, pair would receive implicitly-declared copy/move constructors and copy/move assignment operators, and 12.8/25 would apply. The implicitly-declared copy/move constructors would be trivial when T1 and T2 have trivial copy/move constructors, according to 12.8/13, and similarly for the assignment operators, according to 12.8/27. Notes could be added as a reminder that these functions would be implicitly-declared, but such notes would not be necessary (the Standard Library specification already assumes a	definition in 20.3.5.2 and from 20.3.5.2/12-13, or give pair explicitly-defaulted copy/move constructors and copy/move assignment operators. Change tuple to match.	

1 **MB** = Member body (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	(3)	4	5	(6)	(7)
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Secretariat observations** on each comment submitted

				high level of familiarity with the Core Language, and casual readers will simply assume that pair is copyable and movable).
				
				Alternatively, pair could be given explicitly-defaulted copy/move constructors and copy/move assignment operators. This is a matter of style.
				
				tuple is also affected. tuple's class definition in 20.4.2 [tuple.tuple] contains:
				
				tuple(const tuple&) = default;
				
				tuple(tuple&&);
				
				tuple& operator=(const tuple&);
				
				tuple& operator=(tuple&&);
				
				They should all be removed or all be explicitly-defaulted, to be consistent with pair. Additionally, 20.4.2.1 [tuple.cnstr]/8-9 specifies the behavior of an explicitly-defaulted function, which is currently inconsistent with pair.		

GB 86	20.4		Te	tuple and pair are essentially two implementations of the same type, with the same public interface, and their specification in becoming increasingly intertwined. The tuple library should be merged into the <utility> header to reduce library dependencies and simplify user expectations. The <tuple> header could optionally be retained as a deprecated alias to the <utility> header.	Merge everything declared in 20.4 into the <utility> header. Either remove the <tuple> header entirely, or move it to Annex D as a deprecated alias of the <utility> header.	
US 98	20.4.2.4	Paragraph 4	te/ed	pack_arguments is poorly named. It does not reflect the fact that it is a tuple creation function and that it forwards arguments.	Rename pack_arguments to forward_as_tuple throughout the standard.	
GB 88	20.4.2.4		Te	The tuple_cat template consists of four overloads and that can concatenate only two tuples. A single variadic signature that can concatenate an arbitrary number of tuples would be preferred.	Adopt a simplified form of the proposal in n2795, restricted to tuples and neither requiring nor outlawing support for other tuple-like types.	

1 **MB** = Member body (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	(3)	4	5	(6)	(7)
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Secretariat observations** on each comment submitted
US 99	20.4.2.4	4 - 6	te	pack_arguments is overly complex.	This issue resulted from a lack of understanding of how references are forwarded. The definition of pack_arguments should be simply: template <class... Types> tuple<~~A~~Types&&> pack_arguments(Types&&...t); ~~*Types:*Let Ti be each type in Types....~~ *Effects:* ... *Returns:* tuple<~~A~~Types&&...>(std::forward<Types>(t)...) The synopsis should also change to reflect this simpler signature.	
GB 87	20.4.2.10		Te	There is no compelling reason to assume a heterogeneous tuple of two elements holds a pair of iterators forming a valid range. Unlike std::pair, there are no functions in the standard library using this as a return type with a valid range, so there is even less reason to try to adapt this type for the new for-loop syntax.	Strike 20.4.2.10 and the matching declarations in the header synopsis in 20.4.	
US 100	20.6.1 [ratio.ratio]		te	LWG 1281 was discussed in Pittsburgh, and the decision there was to accept the typedef as proposed and move to Review. Unfortunately the issue was accidentally applied to the FCD, and incorrectly. The FCD version of the typedef refers to ratio<N, D>, but the typedef is intended to refer to ratio<num, den> which in general is not the same type.	Accept the current proposed wording of LWG 1281 which adds: typedef ratio<num, den> type;	
GB 89	20.6.2		Te	The alias representations of the ratio arithmetic templates do not allow implementations to avoid overflow, since they explicitly specify the form of the aliased template instantiation. For example ratio_multiply,ratio<2,LLONG_MAX>> is *required* to alias ratio<2*LLONG_MAX,LLONG_MAX*2>, which	Change the wording in 20.6.2p2-5 as follows: template <class R1, class R2> using ratio_add = see below; The type ratio_add<R1, R2> shall be a synonym	

1 **MB** = Member body (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	(3)	4	5	(6)	(7)
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Secretariat observations** on each comment submitted

				overflows, so is ill-formed. However, this is trivially equal to ratio<1,1>. It also contradicts the opening statement of 20.6.2p1 "implementations may use other algorithms to compute these values".	for
					~~ratio<T1,T2>~~ <ins>ratio<U, V> such that ratio<U,V>::num and
					ratio<U,V>::den are the same as the corresponding members of
					ratio<T1,T2> would be in the absence of arithmetic overflow</ins>
					where T1 has the value R1::num * R2::den + R2::num * R1::den and T2
					has the value R1::den * R2::den. <ins>If the required values of
					ratio<U,V>::num and ratio<U,V>::den cannot be represented in intmax_t
					then the program is illformed.</ins>
					
					template <class R1, class R2> using ratio_subtract = see below;
					
					The type ratio_subtract<R1, R2> shall be a synonym for
					~~ratio<T1,T2>~~ <ins>ratio<U, V> such that ratio<U,V>::num and
					ratio<U,V>::den are the same as the corresponding members of
					ratio<T1,T2> would be in the absence of arithmetic overflow</ins>
					where T1 has the value R1::num * R2::den - R2::num * R1::den and T2
					has the value R1::den * R2::den. <ins>If the required values of
					ratio<U,V>::num and ratio<U,V>::den cannot be represented in intmax_t
					then the program is illformed.</ins>
					
					template <class R1, class R2> using ratio_multiply = see below;	

1 **MB** = Member body (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	(3)	4	5	(6)	(7)
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Secretariat observations** on each comment submitted
					The type ratio_multiply<R1, R2> shall be a synonym for ~~ratio<T1,T2>~~ <ins>ratio<U, V> such that ratio<U,V>::num and ratio<U,V>::den are the same as the corresponding members of ratio<T1,T2> would be in the absence of arithmetic overflow</ins> where T1 has the value R1::num * R2::num and T2 has the value R1::den * R2::den. <ins>If the required values of ratio<U,V>::num and ratio<U,V>::den cannot be represented in intmax_t then the program is illformed.</ins> template <class R1, class R2> using ratio_divide = see below; The type ratio_divide<R1, R2> shall be a synonym for ~~ratio<T1,T2>~~ <ins>ratio<U, V> such that ratio<U,V>::num and ratio<U,V>::den are the same as the corresponding members of ratio<T1,T2> would be in the absence of arithmetic overflow</ins> where T1 has the value R1::num * R2::den and T2 has the value R1::den * R2::num. <ins>If the required values of ratio<U,V>::num and ratio<U,V>::den cannot be represented in intmax_t then the program is illformed.</ins>	
US 101	20.7		te	Paper n2965 was largely rejected after the last CD on the grounds there was no associated national body comment, so I am submitting a national body comment this time.	Consider n2965 in the context of a national body comment.	

1 **MB** = Member body (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	(3)	4	5	(6)	(7)
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Secretariat observations** on each comment submitted
GB 90	20.7		Ed	type_traits is a core support facility offered by the compiler, and exposed with a library interface that is required in a free-standing implementation. It has far more in common with numeric_limits than the utility components in clause 20, and should move to clause 18.	Move clause 20.7 into clause 18.	
DE 17	20.7		te	Speculative compilation for std::is_constructible and std::is_convertible should be limited, similar to the core language (see 14.8.2 paragraph 8).		
DE 18	20.7		te	Several type traits require compiler support, e.g. std::is_constructible or std::is_convertible. Their current specification seems to imply, that the corresponding test expressions should be well-formed, even in absense of access: class X { X(int){} }; constexpr bool test = std::is_constructible<X, int>::value; The specification does not clarify the context of this test and because it already goes beyond normal language rules, it's hard to argue by means of normal language rules what the context and outcome of the test should be.	Specify that std::is_constructible and std::is_convertible will return true only for public constructors/conversion functions.	
US 102	20.7.4		te	Despite Library Issue 520's ("Result_of and pointers to data members") resolution of CD1, the FCD's result_of supports neither pointers to member functions nor pointers to data members. It should.	Ensure result_of supports pointers to member functions and pointers to data members.	
GB 91	20.7.4.3	Table 45	Ed	It is mildly distasteful to dereference a null pointer as part of our specification, as we are playing on the edges of undefined behaviour. With the addition of the declval function template, already used in these same expressions, this is no longer necessary.	Replace the sub-expression '*(U*)0' with the sub-expression 'declval<U&>()' in the specification for has_nothrow_copy_assign and has_nothrow_move_assign type traits.	
GB 92	20.7.4.3	Table 45	Te	Trivial functions implicitly declare a noexcept exception specification, so the references to has_trivial_* traits in the has_nothrow_* traits are redundant, and should be struck for clarity.	For each of the has_nothrow_something traits, remove all references to the matching has_trivial_something traits.	
FI	20.7.4.3, Table 45		te	Related to the change proposed in FI 17, there should be	Add the following type predicate:	

1 **MB** = Member body (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	(3)	4	5	(6)	(7)
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Secretariat observations** on each comment submitted

18	(Type property predicates)			a trait for checking whether a destructor throws.	Template
					
					template <typename T> struct has_nothrow_destructor;
					
					Condition
					
					has_trivial_destructor<T>::value is true or the expression noexcept((*(U*)0).~U()) is well-formed and true, where U is remove_all_extents<T>::type.
					
					Precondition
					
					T shall be a complete type, (possibly cv-qualified) void, or an array of unknown bound.
					
					Reasoning:
					
					With this metafunction the destructor of a class template can adjust its noexcept specification depending on whether destructors of its unbound members (or unbound base classes) might throw:
					
					template <typename T>
					struct C
					{
					T t;
					~C()
					noexcept(has_nothrow_destructor<T>::value) {}
					};	

DE	20.7.4.3		te	The fundamental trait is_constructible reports false	Remove all false positives from the domain of	

1 **MB** = Member body (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	(3)	4	5	(6)	(7)
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Secretariat observations** on each comment submitted
19				positives, e.g. is_constructible<char*, void*>::value evaluates to true, even though a corresponding variable initialization would be ill-formed.	is_constructible.	
JP 32	20.7.5	2, table 47		Representations of reference link are not unified. Most reference links to clause (table) number, say X, are in the form "Clause X" ("Table X") capitalized, and subsection Y.Y.Y is referenced with its number only in the form "Y.Y.Y". Whether they are parenthesized or not depends on the context. However there are some notations "(Z)" consisting of only a number Z in parentheses to confer Clause or Table number Z.	Change "(10)" to "(Clause 10)".	
GB 93	20.7.6.2	Table 49	Te	The comments for add_rvalue_reference say "this rule reflects the semantics of reference collapsing", but reference collapsing is not defined anywhere.	Add a new sentence at the end of 8.3.2p6 "This is called reference collapsing". Add a reference to 8.3.2 to the use of "reference collapsing" in 20.7.6.2/table 49	
US 103	20.7.6.6 [meta.trans.other]		te	The current definition of result_of works for function pointers but the condition statement outlaws them. There is even an example in the WP that shows result_of working for function pointers.	Add “pointer to function” to the list of things that Fn shall be.	
US 104	28.8		te	std::basic_regex should have an allocator for all the reasons that a std::string does. For example, I can use boost::interprocess to put a string or vector in shared memory, but not a regex.	Add allocators to regexes	
GB 94	20.8		Ed	This subclause has grown large, with many components, and so should stand alone.	Promote 20.8 and all its contents up to a new numbered clause.	
GB 95	20.8		Ge	The adaptable function protocol supported by unary_function/binary_function has been superceded by lambda expressions and std::bind. Despite the name, the protocol is not very adaptable as it requires intrusive support in the adaptable types, rather than offering an external traits-like adaption mechanism. This protocol and	Move clauses 20.8.3, 20.8.9, 20.8.11 and 20.8.12 to Annex D. Remove the requirements to conditionally derive from unary/binary_function from function, reference_wrapper, and the results of calling mem_fn and bind.	

1 **MB** = Member body (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	(3)	4	5	(6)	(7)
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Secretariat observations** on each comment submitted
				related support functions should be deprecated, and we should not make onerous requirements for the specification to support this protocol for callable types introduced in this standard revision, including those adopted from TR1. It is expected that high-quality implementations will provide such support, but we should not have to write robust standard specifications mixing this restricted support with more general components such as function, bind and reference_wrapper.		
GB 96	20.8		Te	The function templates 'hash', 'less' and 'equal_to' are important customization points for user-defined types to be supported by several standard containers. These are accessed through the <functional> header which has grown significantly larger in C++0x, exposing many more facilities than a user is likely to need through there own header, simply to declare the necessary specialization. There should be a smaller header available for users to make the necessary customization.	Provide a tiny forwarding header for important functor types in the <functional> header that a user may want to specialize. This should contain the template declaration for 'equal_to', 'hash' and 'less'.	
GB 97	20.8.10		Te	The bind template is intended as a singe, simple to use replacement for the '98 adaptable function APIs and machinery. It works well in almost all respects, but lacks the ability to easily negate a predicate, or equivalently, act as a replacement for not1 and not2. Two easy ways to solve this omission would be to add a 'bind_not' function that produces a binder that negates its result. However, preference is given to requiring the unspecified bind result type to overload operator! to produce the same effect. This is preferred due to (i) its simpler usage, being the naively expected syntax, but more importantly (ii) some (limited) field experience.	Require the unspecified result of a bind expression to support unary operator! to yield another bind result that, when evaluated, yields '!res', where 'res' is the result of evaluating the original function.	

JP 3	20.8.14.2		TL	explicit default contructor is defined in std::function. Although it is allowed according to 12.3.1, it seems unnecessary to qualify the constructor as explicit.
				If it is explicit, there will be a limitation in initializer_list.
					Remove explicit.
					
					namespace std {
					template<class> class function;
					// undefined	

1 **MB** = Member body (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	(3)	4	5	(6)	(7)
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Secretariat observations** on each comment submitted

					template<class R, class... ArgTypes>
					class function<R(ArgTypes...)>
					: public unary_function<T1, R>
					// iff sizeof...(ArgTypes) == 1 and ArgTypes contains T1
					
					: public binary_function<T1, T2, R>
					// iff sizeof...(ArgTypes) == 2 and ArgTypes contains T1 andT2
					{
					public:typedef R result_type;
					
					// 20.8.14.2.1, construct/copy/destroy:function();	

JP 33	20.8.15	1	E	Representations of reference link are not unified. Most reference links to clause (table) number, say X, are in the form "Clause X" ("Table X") capitalized, and subsection Y.Y.Y is referenced with its number only in the form "Y.Y.Y". Whether they are parenthesized or not depends on the context. However there are some notations "(Z)" consisting of only a number Z in parentheses to confer Clause or Table number Z.	Change "(37)" to "(Table 37)".	
JP 34	20.8.15	1	E	Representations of reference link are not unified. Most reference links to clause (table) number, say X, are in the form "Clause X" ("Table X") capitalized, and subsection Y.Y.Y is referenced with its number only in the form "Y.Y.Y". Whether they are parenthesized or not depends on the context. However there are some notations "(Z)" consisting of only a number Z in parentheses to confer Clause or Table number Z.	Change "(33)" to "(Table 33)".	
JP 4	20.8.14.2.1	1	TL	Really does the function require that default constructor is	Remove explicit.	

1 **MB** = Member body (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	(3)	4	5	(6)	(7)
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Secretariat observations** on each comment submitted

				explicit?	function();
					template <class A>
					function(allocator_arg_t, const A& a);	

US 105	20.8.15.2		te	unique_ptr and shared_ptr are inconsistent in their handling of arrays. We can write:
				unique_ptr<int[]> p(new int[10]);
				// handles deletion correctly
				But we cannot write:
				shared_ptr<int[]> p(new int[10]);
				// incorrect
				This is an inconsistency. It is true that we have the following workaround:
				std::shared_ptr<int> s(new int[5],
				std::default_delete<int[]>());
				But this is still inconsistent, not to mention awkward and error-prone because the programmer will occasionally forget the deleter and the code will silently compile and may appear to work on some platforms.
					Support:
					shared_ptr<int[]> p(new int[10]);
					to handle deletion correctly by calling delete[] on the stored pointer.	

GB 98	20.9		Ed	This subclause has grown large, with many components, and so should stand alone.	Promote 20.9 and all of its contents to a new, top-level, numbered clause.	
GB 99	20.9	1	Te	One reason that the unique_ptr constructor taking a nullptr_t argument is not explicit is to allow conversion of nullptr to unique_ptr in contexts like equality comparison. Unfortunately operator== for unique_ptr is a little more clever than that, deducing template parameters for both arguments. This means that nullptr does not get deduced as unique_ptr type, and there are no other comparison functions to match.	Add the following signatures to 20.9p1, <memory> header synopsis: template<typename T, typename D> bool operator==(const unique_ptr<T, D> & lhs, nullptr_t); template<typename T, typename D> bool operator==(nullptr_t, const unique_ptr<T, D> &	

1 **MB** = Member body (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	(3)	4	5	(6)	(7)
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Secretariat observations** on each comment submitted

					rhs);
					
					template<typename T, typename D> bool operator!=(const unique_ptr<T, D> & lhs, nullptr_t);
					
					template<typename T, typename D> bool operator!=(nullptr_t, const unique_ptr<T, D> & rhs);	

GB 100	20.9		Te	The unique_ptr and shared_ptr constructors taking nullptr_t delegate to a constexpr constructor, and could be constexpr themselves.	In the 20.9.10.2 [unique.ptr.single] synopsis add "constexpr" to unique_ptr(nullptr_t). In the 20.9.10.3 [unique.ptr.runtime] synopsis add "constexpr" to unique_ptr(nullptr_t). In the 20.9.11.2 [util.smartptr.shared] synopsis add "constexpr" to shared_ptr(nullptr_t).	

JP 85	20.9.1		E	There are inconsistent definitions for allocator_arg.
				In 20.9 [memory] paragraph 1,
				constexpr allocator_arg_t allocator_arg = allocator_arg_t();
				and in 20.9.1,
				const allocator_arg_t allocator_arg = allocator_arg_t();
					Change "const" to "constexpr" in 20.9.1 as follows.
					constexpr allocator_arg_t allocator_arg = allocator_arg_t();	

US 106	20.9.3	all	te	pointer_traits should have a size_type member for completeness.	Add “typedef *see below* size_type;” to the generic pointer_traits template and “typedef size_t size_type;” to pointer_traits<T*>. Use pointer_traits::size_type and pointer_traits::difference_type as the defaults for allocator_traits::size_type and allocator_traits::difference_type. See Appendix 1 - Additional Details	
GB 104	20.9.5.1	13	Te	The ~ is missing from the invokation of the destructor of U.	Add the missing ~ : Effects: p->~U()	
GB 105	20.9.6	1	Te	There is a missing '_' in the piecewise-construct call for pair in the class definition.	Fix the declaration: template <class T1, class T2, class... Args1, class... Args2> void construct(pair<T1, T2>* p,	

1 **MB** = Member body (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	(3)	4	5	(6)	(7)
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Secretariat observations** on each comment submitted
					piecewise_construct_t, tuple<Args1...> x, tuple<Args2...> y);	
US 107	20.9.6		Te	scoped_allocator_adaptor should have its own header.	See Appendix 1 - Additional Details	
GB 101	20.9.10	5	Ed	The first sentence of the paragraph says "Each object of a type U instantiated form the unique_ptr template..." "form" should be "from"	Replace "form" with "from" in the opening sentence: "Each object of a type U instantiated from the unique_ptr template..."	
FI 13	20.9.10.	5	ed	typo	"form the unique_ptr" should be "from the unique_ptr."	
GB 102	20.9.10		Ed	unique_ptr is a smart pointer so [unique.ptr] should be a sub-clause of [util.smartptr]	move [unique.ptr] to a sub-clause of [util.smartptr]	
JP 35	20.9.10.2	2	E	Representations of reference link are not unified. Most reference links to clause (table) number, say X, are in the form "Clause X" ("Table X") capitalized, and subsection Y.Y.Y is referenced with its number only in the form "Y.Y.Y". Whether they are parenthesized or not depends on the context. However there are some notations "(Z)" consisting of only a number Z in parentheses to confer Clause or Table number Z.	Change "(38)" to "(Table 38)".	
JP 36	20.9.10.2.1	1, 6		Representations of reference link are not unified. Most reference links to clause (table) number, say X, are in the form "Clause X" ("Table X") capitalized, and subsection Y.Y.Y is referenced with its number only in the form "Y.Y.Y". Whether they are parenthesized or not depends on the context. However there are some notations "(Z)" consisting of only a number Z in parentheses to confer Clause or Table number Z.	Change "(33)" to "(Table 33)".	
JP 37	20.9.10.2.1	18		Representations of reference link are not unified. Most reference links to clause (table) number, say X, are in the form "Clause X" ("Table X") capitalized, and	Change "(34)" to "(Table 34)".	

1 **MB** = Member body (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	(3)	4	5	(6)	(7)
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Secretariat observations** on each comment submitted
				subsection Y.Y.Y is referenced with its number only in the form "Y.Y.Y". Whether they are parenthesized or not depends on the context. However there are some notations "(Z)" consisting of only a number Z in parentheses to confer Clause or Table number Z.		
JP 38	20.9.10.2.3	1		Representations of reference link are not unified. Most reference links to clause (table) number, say X, are in the form "Clause X" ("Table X") capitalized, and subsection Y.Y.Y is referenced with its number only in the form "Y.Y.Y". Whether they are parenthesized or not depends on the context. However there are some notations "(Z)" consisting of only a number Z in parentheses to confer Clause or Table number Z.	Change "(36)" to "(Table 36)".	

JP 5	20.9.11.2	1	TL	Hash support based on ownership sharing should be supplied for shared_ptr and weak_ptr.
				
				For two shared_ptr objects p and q, two distinct equivalence relations can be defined. One is based on equivalence of pointer values, which is derived from the expression p.get() == q.get() (hereafter called address-based equivalence relation), the other is based on equivalence of ownership sharing, which is derived from the expression !p.owner_before(q) && !q.owner_before(p) (hereafter called ownership-based equivalence relation). These two equivalence relations are independent in general. For example, a shared_ptr object created by the constructor of the signature shared_ptr(shared_ptr<U> const &, T *) could reveal a difference between these two relations. Therefore, hash support based on each equivalence relation should be supplied for shared_ptr. However, while the standard library provides the hash support for address-based one (20.9.11.6 paragraph 2), it lacks the hash support for ownership-based one. In
					Add the following non-static member functions to shared_ptr and weak_ptr class template;
					// 20.9.11.2 paragraph 1
					namespace std{
					template<class T> class shared_ptr {
					public:
					...
					size_t owner_hash() const;
					...
					};
					}
					// 20.9.11.3 paragraph 1
					namespace std{
					template<class T> class weak_ptr {
					public:
					...
					size_t owner_hash() const;
					...
					};
					}	

1 **MB** = Member body (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	(3)	4	5	(6)	(7)
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Secretariat observations** on each comment submitted
				addition, associative containers work well in combination with the shared_ptr's ownership-based comparison but unordered associative containers don't. This is inconsistent. For the case of weak_ptr, hash support for the ownership-based equivalence relation can be safely defined on weak_ptrs, and even on expired ones. The absence of hash support for the ownership-based equivalence relation is fatal, especially for expired weak_ptrs. And the absence of such hash support precludes some quite effective use-cases, e.g. erasing the unordered_map entry of an expired weak_ptr key from a customized deleter supplied to shared_ptrs. Hash support for the ownership-based equivalence relation cannot be provided by any user-defined manner because information about ownership sharing is not available to users at all. Therefore, the only way to provide ownership-based hash support is to offer it intrusively by the standard library. As far as we know, such hash support is implementable. Typical implementation of such hash function could return the hash value of the pointer of the counter object that is internally managed by shared_ptr and weak_ptr.	These functions satisfy the following requirements. Let p and q be objects of either shared_ptr or weak_ptr, H be a hypothetical function object type that satisfies the hash requirements (20.2.4) and h be an object of the type H. The expression p.owner_hash() behaves as if it were equivalent to the expression h(p). In addition, h(p) == h(q) must become true if p and q share ownership.	
CH 20	20.9.11.2	p4	te	Requiring shared_ptr and weak_ptr to always synchronize the use count makes it potentially slow and is inconsistent with the general approach to leave the synchronization to the user of a facility.	Strike 'not' from 'Changes in use_count() do not reflect modifications that can introduce data races.' Possibly add additional synchronized constructors and assignments.	
US 108	20.9.11.2.1 [util.smartptr.shared.const]		te	shared_ptr should have the same policy for constructing from auto_ptr as unique_ptr. Currently it does not.	Add “template <class Y> explicit shared_ptr(auto_ptr<Y>&); to [util.smartptr.shared.const] (and to the synopsis).	
US 109	20.9.11.2.6		te	20.9.11.2.6 [util.smartptr.shared.create]/2 says: "the placement new expression ::new (pv) T() or ::new (pv) T(std::forward<Args>(args)...)". It should simply say "the placement new expression ::new (pv)	Change "the placement new expression ::new (pv) T() or ::new (pv) T(std::forward<Args>(args)...)"	

1 **MB** = Member body (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	(3)	4	5	(6)	(7)
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Secretariat observations** on each comment submitted
				T(std::forward<Args>(args)...)", because empty parameter packs expand to nothing. This would be consistent with the requirements in paragraph 1.	to "the placement new expression ::new (pv) T(std::forward<Args>(args)...)"	
GB 103	20.9.12	12	Te	The precondition to calling declare_no_pointers is that no bytes in the range "have been previously registered" with this call. As written, this precondition includes bytes in ranges, even after they have been explicitly unregistered with a later call to 'undeclare_no_pointers'.	Replace "have been previously registered" with "are currently registered"	
GB 106	20.10.3		Te	duration is an arithmetic type, unlike time_point, and so should provide a specialization of numeric_limits.	Add a declaration of a partial specialization of numeric_limits for duration to the header synopsis in 20.10. Add 20.3.8 [time.duration.limits] "duration is an arithmetic type, and so provides an appropriate specialization of numeric_limits."	
US 110	20.10.5		te	Significant parts of the clock section are "unspecified", rather than "implementation-defined".	Make those parts "implementation-defined".	
US 111	20.10.5.2	para 1	te	What it means for monotonic_clock to be a synonym is undefined. If it may or may not be a typedef, then certain classes of programs become unportable.	Require that it be a distinct class type.	
GB 107	20.10.5.2	2	Te	1.4p9 states that which conditionally supported constructs are available should be provided in the documentation for the implementation. This doesn't help programmers trying to write portable code, as they must then rely on implementation-specific means to determine the availability of such constructs. In particular, the presence or absence of std::chrono::monotonic_clock may require different code paths to be selected. This is the only conditionally-supported library facility, and differs from the conditionally-supported language facilities in that it has standard-defined semantics rather than implementation-defined semantics.	Provide feature test macro for determining the presence of std::chrono::monotonic_clock. Add _STDCPP_HAS_MONOTONIC_CLOCK to the <chrono> header, which is defined if monotonic_clock is present, and not defined if it is not present.	
DE	20.10.5.2		te	The library component monotonic_clock is conditionally	Provide a compile-time flag (preferably a macro)	

1 **MB** = Member body (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	(3)	4	5	(6)	(7)
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Secretariat observations** on each comment submitted
20				supported, but no compile-time flag exists that allows user-code to query its existence. Further-on there exist no portable means to simulate such a query. (To do so, user code would be required to add types to namespace std::chrono.)	that can be used to query the existence of monotonic_clock.	
CH 21	20.10.5.2	p2	te	Monotonic clocks are generally easy to provide on all systems and are implicitely required by some of the library facilities anyway.	Make monotonic clocks mandatory, i.e. remove p2. Also change 30.2.4p2 accordingly.	
US 112	20.10.5.3	para 1	te	What it means for high_resolution_clock to be a synonym is undefined. If it may or may not be a typedef, then certain classes of programs become unportable.	Require that it be a distinct class type.	
JP 39	21.2.2	4	E	Representations of reference link are not unified. Most reference links to clause (table) number, say X, are in the form "Clause X" ("Table X") capitalized, and subsection Y.Y.Y is referenced with its number only in the form "Y.Y.Y". Whether they are parenthesized or not depends on the context. However there are some notations "(Z)" consisting of only a number Z in parentheses to confer Clause or Table number Z.	Change "(35)" to "(Table 35)".	
GB 108	21.2.3.1	2,3	Te	The definition of streamoff/streampos defers to the definition of off_type and pos_type in 21.2.2, which defers back to 27.2.2 for the definition of streamoff/streampos. The actual definition appears to be supplied in 27.3, the synopsis of <iosfwd>.	Update the reference in 21.2.3.1 to refer forward to 27.2.2, rather than back to 21.2.2. Add a cross-reference to from 27.2.2 to 27.3.	
GB 109	21.2.3.2/3/4		Te	It is not clear what the specification means for u16streampos, u32streampos or wstreampos when they refer to the requirements for POS_T in 21.2.2, as there are no longer any such requirements. Similarly the annex D.7 refers to the requirements of type POS_T in 27.3 that no longer exist either.	Clarify the meaning of all cross-reference to the removed type POS_T.	
JP 40	21.4	3	E	Representations of reference link are not unified. Most reference links to clause (table) number, say X, are in the form "Clause X" ("Table X") capitalized, and	Change "(96)" to "(Table 96)".	

1 **MB** = Member body (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	(3)	4	5	(6)	(7)
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Secretariat observations** on each comment submitted
				subsection Y.Y.Y is referenced with its number only in the form "Y.Y.Y". Whether they are parenthesized or not depends on the context. However there are some notations "(Z)" consisting of only a number Z in parentheses to confer Clause or Table number Z.		
GB 110	21.4.7.1		Te	data() is the function of choice for calling into legacy 'C'-like APIs. Both vector and array designed this function to be callable in a const-correct way while allowing for functions that want to use the result to designate a return buffer.	Add the following overload to basic_string data(): charT * data(); Relax the requirement that programs do not alter values in the array through the pointe retrieved through this new overload.	
GB 111	21.5		Te	Section 17.6.4.8, Data Race Avoidance, requires the C++ Standard Library to avoid data races that might otherwise result from two threads making calls to C++ Standard Library functions on distinct objects. The C standard library is part of the C++ Standard Library and some C++ Standary library functions (parts of the Localization library, as well as Numeric Conversions in 21.5), are specified to make use of the C standard library. Therefore, the C++ standard indirectly imposes a requirement on the thread safety of the C standard library. However, since the C standard does not address the concept of thread safety conforming C implementations exist that do no provide such guarantees. This conflict needs to be reconciled.	remove the requirement to make use of strtol() and sprintf() since these functions depend on the global C locale and thus cannot be made thread safe.	
JP 86	21.7	1	E	Table numbers are listed incorrectly. "74,75. and" should be "74, 75, and".	Correct typo as follows. Tables 71, 72, 73, 74, 75, and 76 describe	
JP 87	22.3.1		E	While usage of "traits" and "Traits" are explained in 21.2 as template parameters and arguments respectively, specifying "Traits" as template parameter seems misuse. template <class charT, class Traits, class Allocator> bool operator()(const basic_string<charT,Traits,Allocator>& s1, const basic_string<charT,Traits,Allocator>& s2) const;	Change "Traits" to "traits" in three places.	

1 **MB** = Member body (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	(3)	4	5	(6)	(7)
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Secretariat observations** on each comment submitted
JP 88	22.6	3, Table 91	E	Typo, duplicated "ispunct" and missing "iswpunct".	Correct as follows. isprint ispunct isspace : iswprint iswpunct iswspace	
JP 89	23.1	2, Table 92	E	Typo, "<forwardlist>" should be "<forward_list>".	Correct typo. <forward_list>	
US 113	23.2	5	te	Resolve LWG 579 one way or the other, but preferably in the direction of changing the two erase overloads to return void.		
US 114	23.2.1	Paragraph 9	te	Requirements on iterators swapping allegiance would disallow the small-string optimization.	Add an exclusion for basic_string to the sentence beginning “Every iterator referring to an element...”. Add a sentence to 21.4.6.8/2 saying that iterators and references to string elements remain valid, but it is not specified whether they refer to the same string or the other string.	
DE 21	23.2.1, 23.3.3.4		te	23.2.1/11 provides a general no-throw guarantee for erase() container functions, exceptions from this are explicitly mentioned for individual containers. Because of its different name, forward_list's erase_after() function is not ruled by this but should so.	Add a "Throws: Nothing" clause to both erase_after overloads in 23.3.3.4.	
US 115	23.2.1	Paragraph 15	te	The terms CopyConstructible, MoveConstructible, and *constructible from* are redefined, then used inconsistently and often incorrectly within the section. New terms should have been introduced and used correctly.	Better terms would be *X can copy-insert T, X can move-insert T,* and *X can construct-insert T with args*. See Appendix 1 - Additional Details	
US 116	23.2.1	Table 96	ed	The requirement for X(rv) that move construction of the allocator not throw can be misread as requiring that move construction of the whole container not throw.	Add non-normative note: *Requires:* move construction of A shall not exit via an exception. [*Note*: This requirement on allocators exists so that implementations can	

1 **MB** = Member body (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	(3)	4	5	(6)	(7)
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Secretariat observations** on each comment submitted
					(optionally) provide a nothrow guarantee on move construction of some containers. – *end note*]	
FI 12	23.2.3 [sequence.req mts]	Table 97 — Sequence container requirement s (in addition to container)	te	The requirement for insert says: "Requires:T shall be CopyConstructible. For vector and deque, T shall also be CopyAssignable.". Why must T be CopyAssignable? Is it for cases where the object may be already constructed and insert will first copy the existing data out of the way and assign to an existing object? Can't such an implementation do the same with placement-new?	CopyAssignable seems like an overly strict requirement for insert, exposing implementation details in the specification. If such implementation details are achievable without assignment, eg. with placement-new, the CopyAssignable requirement should be removed.	
JP 90	23.2.3	17, Table 98	E	In Operational semantics for "a.emplace_front(args)", <Arg> should be <Args>. Prepends an object of type T constructed with std::forward<Arg>(args)...."	Change <Arg> to <Args>.	
JP 91	23.2.5	10, Table 100	E	Typo, unnecessary new-line. a_uniq. emplace(args) Typo, unnecessary space. a_eq. emplace(args)	Remove space characters between "." and "emplace" same as other word wrapped columns in the table.	
JP 92	23.2.5	11	E	Typo, "unodified" should be "unmodified".	Correct typo. unmodified	
ES 2	23.2.5 [unord.req], Table 100 (Unordered associative container requirements (in addition to container))	Row for expression `a.erase(q)`	Te	The expression is required to return the iterator immediately following q prior to the erasure. As explained in N2023, this requirement makes it impossible to achieve average O(1) complexity for unordered associative containers implemented with singly linked lists. This has a theoretical as well as a practical impact, as reported by users of early implementations of these containers. Discussions among committee members have not found any way of remedying this deficiency (other than acknowledging it) by some smart modification of usual singly linked lists implementations.	Change the return type of the expression from `iterator` to `void`. Eliminate the sentence "Return value is the iterator immediately following q prior to the erasure". Change accordingly the appearances of "`iterator erase(const_iterator position)`" in 23.5.1, 23.5.2, 23.5.3 and 23.5.4.	

1 **MB** = Member body (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	(3)	4	5	(6)	(7)
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Secretariat observations** on each comment submitted
GB 112	23.3.1.7	p3	Te	Should the effect of calling front/back on a zero sized array really be implementation defined i.e. require the implementor to define behaviour?	Change "implementation defined" to "undefined"	
GB 113	23.3.2.2	p1	Te	There is no mention of what happens if sz==size(). While it obviously does nothing I feel a standard needs to say this explicitely.	Append "If sz == size(), does nothing" to the effects.	
US 117	23.3.3		Te	forward_list::erase_after should return an iterator.	See Appendix 1 - Additional Details	
JP 41	23.3.3	2	E	Representations of reference link are not unified. Most reference links to clause (table) number, say X, are in the form "Clause X" ("Table X") capitalized, and subsection Y.Y.Y is referenced with its number only in the form "Y.Y.Y". Whether they are parenthesized or not depends on the context. However there are some notations "(Z)" consisting of only a number Z in parentheses to confer Clause or Table number Z.	Change "(table 93)" to "(Table 93)".	
GB 114	23.3.4.1	p11 & p12	Ed	It looks like the erase/insert effects given in p11 are intended for p12.	Move the erase/insert effects down to p12	
GB 115	23.3.4.2	p1	Te	There is no mention of what happens if sz==size(). While it obviously does nothing I feel a standard needs to say this explicitly.	Express the semantics as pseudo-code similarly to the way it is done for the copying overload that follows (in p3). Include an else clause that does nothing and covers the sz==size() case.	
GB 116	23.3.5		Ed	The sequence container adaptors consume sequence containers, but are neither containers nor sequences themselves. While they clearly belong in clause 23, they should not interrupt the presentation of the sequence container themselves.	Move clause 23.3.5 out of clause 23.3. Recommending inserting as a 'new' 23.4 immediately following sequence containers, and before the current 23.4 (associative containers) which would be renumbered 23.5.	
DE 22	23.3.5.1, 23.3.5.2, 23.3.5.3		te	With the final acceptance of move operations as special members and introduction of corresponding suppression rules of implicitly generated copy operations the some library types that were copyable in C++03 are no longer copyable (only movable) in C++03, among them queue, priority_queue, and stack.		

1 **MB** = Member body (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	(3)	4	5	(6)	(7)
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Secretariat observations** on each comment submitted
JP 93	23.3.5.2	1	E	Typo, missing ";". template <class... Args> void emplace(Args&&... args)	Correct typo. template <class... Args> void emplace(Args&&... args);	
GB 117	23.3.6.2	p9	Te	(Same as for 23.3.2.2p1 i.e. deque::resize). There is no mention of what happens if sz==size(). While it obviously does nothing I feel a standard needs to say this explicitely.	Append "If sz == size(), does nothing" to the effects.	
GB 118	23.3.7		Te	vector<bool> iterators are not random access iterators because their reference type is a special class, and not 'bool &'. All standard libary operations taking iterators should treat this iterator as if it was a random access iterator, rather than a simple input iterator.	Either revise the iterator requirements to support proxy iterators (restoring functionality that was lost when the Concept facility was removed) or add an extra paragraph to the vector<bool> specification requiring the library to treat vector<bool> iterators as-if they were random access iterators, despite having the wrong reference type.	

JP 6	23.4.1	2	TL	Constructor accepting an allocator as a single parameter should be qualified as explicit.
				namespace std {
				template <class Key, class T, class Compare = less<Key>,
				class Allocator = allocator<pair<const Key, T> > >
				class map {
				public:
				...
				map(const Allocator&);
					Add explicit.
					namespace std {
					template <class Key, class T, class Compare = less<Key>,
					class Allocator = allocator<pair<const Key, T> > >
					class map {
					public:
					...
					explicit map(const Allocator&);	

JP 7	23.4.2	2	TL	Constructor accepting an allocator as a single parameter should be qualified as explicit.	Add explicit.
					namespace std {
					template <class Key, class T, class Compare = less<Key>,
					class Allocator = allocator<pair<const Key, T> > >
					class multimap {
					public:
					...
					explicit multimap(const Allocator&);	

1 **MB** = Member body (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	(3)	4	5	(6)	(7)
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Secretariat observations** on each comment submitted

JP 8	23.4.3	2	TL	Constructor accepting an allocator as a single parameter should be qualified as explicit.	Add explicit.
					namespace std {
					template <class Key, class Compare = less<Key>,
					class Allocator = allocator<Key> >
					class set {
					public:
					...
					explicit set(const Allocator&);	

JP 9	23.4.4	2	TL	Constructor accepting an allocator as a single parameter should be qualified as explicit.	Add explicit.
					namespace std {
					template <class Key, class Compare = less<Key>,
					class Allocator = allocator<Key> >
					class multiset {
					public:
					...
					explicit multiset(const Allocator&);	

US 118	23.5		te	Some unordered associative container operations have undesirable complexities when the container is implemented using singly linked lists.	See Appendix 1 - Additional Details	

JP 10	23.5.1	3	TL	Constructor accepting an allocator as a single parameter should be qualified as explicit.	Add explicit.
					namespace std {
					template <class Key,
					template <class Key,
					class T,
					class Hash = hash<Key>,
					class Pred = std::equal_to<Key>,
					class Alloc = std::allocator<std::pair<const Key,
					T> > >
					class unordered_map
					{	

1 **MB** = Member body (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	(3)	4	5	(6)	(7)
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Secretariat observations** on each comment submitted
					public: ... explicit unordered_map(const Allocator&);	

JP 11	23.5.2	3	TL	Constructor accepting an allocator as a single parameter should be qualified as explicit.	Add explicit.
					namespace std {
					template <class Key,
					class T,
					class Hash = hash<Key>,
					class Pred = std::equal_to<Key>,
					class Alloc = std::allocator<std::pair<const Key, T> > >
					class unordered_multimap
					{
					public:
					...
					explicit unordered_multimap(const Allocator&);	

JP 94	23.5.2	1	E	"see below" should be in italic and need one space between words. explicit unordered_multimap(size_type n = seebelow,	Change to: explicit unordered_multimap(size_type n = see below,	

JP 12	23.5.3	3	TL	Constructor accepting an allocator as a single parameter should be qualified as explicit.	Add explicit.
					namespace std {
					template <class Key,
					class Hash = hash<Key>,
					class Pred = std::equal_to<Key>,
					class Alloc = std::allocator<Key> >
					class unordered_set
					{
					public:
					...
					explicit unordered_set(const Allocator&);	

JP	23.5.4	3	TL	Constructor accepting an allocator as a single parameter	Add explicit.	

1 **MB** = Member body (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	(3)	4	5	(6)	(7)
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Secretariat observations** on each comment submitted

13				should be qualified as explicit.	namespace std {
					template <class Key,
					class Hash = hash<Key>,
					class Pred = std::equal_to<Key>,
					class Alloc = std::allocator<Key> >
					class unordered_multiset
					{
					public:
					...
					explicit unordered_multiset(const Allocator&);	

US 119	[input.iterators] 24.2.3	Table 104	te	Although the section talks about operator==, there is no requirement that it exist.	Add a == b to Table 104	
JP 42	25.1	8, 9	E	Representations of reference link are not unified. Most reference links to clause (table) number, say X, are in the form "Clause X" ("Table X") capitalized, and subsection Y.Y.Y is referenced with its number only in the form "Y.Y.Y". Whether they are parenthesized or not depends on the context. However there are some notations "(Z)" consisting of only a number Z in parentheses to confer Clause or Table number Z.	Change "(4)" to "(Clause 4)".	
US 120	25.2.12	para 1	te	is_permutation is underspecified for anything but the simple case where both ranges have the same value type and the comparison function is an equivalence relation.	Restrict is_permutation to the case where it is well specified. See Appendix 1 - Additional Details	
ES 3	25.2.12		Te	is_permutation does not require ForwardIterator1 and ForwardIterator2 to have the same value type. This opens the door to nonsense heterogeneous usage where both ranges have different value types	Require both iterator types to have the same value type	
JP 43	25.3.9	5	E	Representations of reference link are not unified. Most reference links to clause (table) number, say X, are	Change "(4)" to "(Clause 4)".	

1 **MB** = Member body (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	(3)	4	5	(6)	(7)
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Secretariat observations** on each comment submitted
				in the form "Clause X" ("Table X") capitalized, and subsection Y.Y.Y is referenced with its number only in the form "Y.Y.Y". Whether they are parenthesized or not depends on the context. However there are some notations "(Z)" consisting of only a number Z in parentheses to confer Clause or Table number Z.		
US 121	25.3.12 [alg.random.shuffle]	1	te	random_shuffle and shuffle should be consistent in how they accept their source of randomness: either both by rvalue reference or both by lvalue reference.	Change random_shuffle to accept its RandomNumberGenerator by lvalue reference.	
GB 119	25.3.12		Te	The functions random_shuffle and shuffle both take arguments providing a source of randomness, but one take its argument by rvalue reference, and the other requires an lvalue reference. The technical merits of which form of argument passing should be settled for this specific case, and a single preferred form used consistently.	[DEPENDS ON WHETHER RVALUE OR LVALUE REFERENCE IS THE PREFERRED FORM]	
JP 44	25.4	2	E	Representations of reference link are not unified. Most reference links to clause (table) number, say X, are in the form "Clause X" ("Table X") capitalized, and subsection Y.Y.Y is referenced with its number only in the form "Y.Y.Y". Whether they are parenthesized or not depends on the context. However there are some notations "(Z)" consisting of only a number Z in parentheses to confer Clause or Table number Z.	Change "(4)" to "(Clause 4)".	
JP 45	25.4.7	1, 10, 19	E	Representations of reference link are not unified. Most reference links to clause (table) number, say X, are in the form "Clause X" ("Table X") capitalized, and subsection Y.Y.Y is referenced with its number only in the form "Y.Y.Y". Whether they are parenthesized or not depends on the context. However there are some notations "(Z)" consisting of only a number Z in parentheses to confer Clause or Table	Change "(32)" to "(Table 32)".	

1 **MB** = Member body (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	(3)	4	5	(6)	(7)
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Secretariat observations** on each comment submitted
				number Z.		
US 122	25.4.7 [alg.min.max]		te	It was the LWG's intent in Pittsburgh that N2772 be applied to the WP	Apply N2772 to the WP.	
US 123	25.5	¶5b, and all uses of lshift	ed	N3056, as adopted, calls for each use of lshift to be followed by a subscripted value.	Adjust all occurrences of the lshift notation so as to match the notation of N3056.	
FI 14	26.3.1.	3	ed	typo	"floating-point environmnet" should be "floating-point environment."	
GB 120	26.4.7		Ge	The complex number functions added for compatibility with the C99 standard library are defined purely as a cross-reference, with no hint of what they should return. This is distinct from the style of documentation for the functions in the earlier standard. In the case of the inverse-trigonometric and hyperbolic functions, a reasonable guess of the functionality may be made from the name, this is not true of the cproj function, which apparently returns the projection on the Reimann Sphere. A single line description of each function, associated with the cross-reference, will greatly improve clarity.	[ONE LINE DESCRIPTIONS, AND ASSOCIATED PARAGRAPH NUMBERS, TO FOLLOW IF THE INTENT IS APPROVED]	
JP 46	26.5.1.6	3	E	Representations of reference link are not unified. Most reference links to clause (table) number, say X, are in the form "Clause X" ("Table X") capitalized, and subsection Y.Y.Y is referenced with its number only in the form "Y.Y.Y". Whether they are parenthesized or not depends on the context. However there are some notations "(Z)" consisting of only a number Z in parentheses to confer Clause or Table number Z.	Change "Clause 21 and 27" to "Clauses 21 and 27".	
GB 121	26.5.3		Te	All the random number engine types in this clause have a constructor taking an unsigned integer type, and a constructor template for seed sequences. This means that an attempt to create a random number engine seeded by an integer literal must remember to add the appropriate unsigned suffix to the literal, as a signed integer will	[WORDING TO FOLLOW ONCE A PREFERRED DIRECTION IS INDICATED]	

1 **MB** = Member body (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	(3)	4	5	(6)	(7)
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Secretariat observations** on each comment submitted
				attempt to use the seed sequence template, yielding undefined behaviour, as per 26.5.1.1p1a. It would be helpful if at least these anticipated cases produced a defined behaviour, either an erroneous program with diagnostic, or a conversion to unsigned int forwarding to the appropriate constructor.		
US 124	26.5.3.2	¶4	te	The Mersenne twister algorithm is meaningless for word sizes less than two, as there are then insufficient bits available to be "twisted".	Insert the following among the relations that are required to hold: 2u < w.	
US 125	26.5.4.1 [rand.adapt.disc]	3	ed	The synopsis for min() and max() is lacking "()" in the return statements.	return Engine::min(); return Engine::max();	
US 126	26.5.4.1 [rand.adapt.disc], 26.5.4.2 [rand.adapt.ibits], 26.5.4.3 [rand.adapt.shuf]	3	te	Each adaptor has a member function called base() which has no definition.	Give it the obvious definition.	
US 127	26.5.4.1	synopsis after ¶3	ed/te	Engine::min is a function and ought be invoked in the context where mentioned, as should Engine::max.	Append parentheses so as to become return Engine::min() and return Engine::max().	
US 128	26.5.4.3	synopsis after ¶3	ed/te	Engine::min is a function and ought be invoked in the context where mentioned, as should Engine::max.	Append parentheses so as to become return Engine::min() and return Engine::max().	
US 129	26.5.7.1	¶8b	ed/te	The expression begin[x+q] is incorrect since x is unspecified .	Replace x by k so as to obtain begin[k+q].	
US 130	26.5.7.1	¶8c	ed/te	The phrase "three more times" is misleading.	s/three more times,/again,/	
US 131	26.5.7.1	¶8c	ed/te	Values r_3 and r_4 are correctly specified, but in their subsequent use they are interchanged with respect to the original algorithm by Mutso Saito.	Exchange subscripts so as to read as follows: "… update begin[k + p] by xoring it with r_3, update begin[k + q] by xoring it with r_4, and …"	
US	26.5.7.1	8b	ed	The last sentence includes "begin[x + q]" but "x" is non-	Change "begin[x + q]" to "begin[k + q]"	

1 **MB** = Member body (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	(3)	4	5	(6)	(7)
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Secretariat observations** on each comment submitted
132	[rand.util.seed seq]			sensical here.		
US 133	26.5.7.1 [rand.util.seed seq]	8c	te	The use of r3 and r4 is reversed in the final sentence of 8c according to the defect report on comp.std.c++ that this specification is based on: http://groups.google.com/group/comp.std.c++/browse_thread/thread/e34cbee1932efdb8/aad523dccec12aed?q=group:comp.std.c%2B%2B+insubject:seed_seq If you follow the SFMT link to the software, the software also uses r3 and r4 in a manner inconsistent with N3092. I believe N3092 should be changed to be consistent with the defect report.	Change 8c to end: ... update begin[k + p] by xoring it with r3, update begin[k + q] by xoring it with r4, and ...	
US 134	26.5.8.5.2 [rand.dist.samp.pconst], 26.5.8.5.3 [rand.dist.samp.plinear]		te	These two distributions have a member function called densities() which returns a vector<double>. The distribution is templated on RealType. The distribution also has another member called intervals() which returns a vector<RealType>. Why doesn't densities return vector<RealType> as well? If RealType is long double, the computed densities property isn't being computed to the precision the client desires. If RealType is float, the densities vector is taking up twice as much space as the client desires.	Change the piecewise constant and linear distributions to hold / return the densities in a vector<result_type>. If this is not done, at least correct[rand.dist.samp.pconst]/13 which describes the return of densities as a vector<result_type>.	
US 135	26.5.8.5.3	10	te	This paragraph says: Let bk = xmin+k·ð for k = 0,...,n, and wk = fw(bk +ð) for k = 0,...,n. However I believe that fw(bk) would be far more desirable. I strongly suspect that this is nothing but a type-o.	Change p10 to read: Let bk = xmin+k·ð for k = 0,...,n, and wk = fw(bk) for k = 0,...,n.	
JP 47	26.7.1	2	E	Representations of reference link are not unified. Most reference links to clause (table) number, say X, are in the form "Clause X" ("Table X") capitalized, and subsection Y.Y.Y is referenced with its number only in the form "Y.Y.Y". Whether they are parenthesized or not depends on the context. However there are some notations "(Z)" consisting of only a number Z in parentheses to confer Clause or Table	Change "(35)" to "(Table 35)".	

1 **MB** = Member body (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	(3)	4	5	(6)	(7)
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Secretariat observations** on each comment submitted
				number Z.		
JP 48	26.7.2	2	E	Representations of reference link are not unified. Most reference links to clause (table) number, say X, are in the form "Clause X" ("Table X") capitalized, and subsection Y.Y.Y is referenced with its number only in the form "Y.Y.Y". Whether they are parenthesized or not depends on the context. However there are some notations "(Z)" consisting of only a number Z in parentheses to confer Clause or Table number Z.	Change "(35)" to "(Table 35)".	
JP 49	26.7.4	1	E	Representations of reference link are not unified. Most reference links to clause (table) number, say X, are in the form "Clause X" ("Table X") capitalized, and subsection Y.Y.Y is referenced with its number only in the form "Y.Y.Y". Whether they are parenthesized or not depends on the context. However there are some notations "(Z)" consisting of only a number Z in parentheses to confer Clause or Table number Z.	Change "(36)" to "(Table 36)".	
US 136	26.8		Te	Floating-point test functions are incorrectly specified.	See Appendix 1 - Additional Details	
CA 9	27.2.3p2 30.3.1.2p6 30.3.1.5p7 30.6.4p7 30.6.9p5 30.6.10.1p23	27.2.3p2 30.3.1.2p6 30.3.1.5p7 30.6.4p7 30.6.9p5 30.6.10.1p2	te	Imposed happens-before edges should be in synchronizes-with Each use of the words "happens-before" should be replaced with the words "synchronizes-with" in the following sentences: 27.2.3p2 30.3.1.2p6	Each use of the words "happens-before" should be replaced with the words "synchronizes-with" in the following sentences: 27.2.3p2 30.3.1.2p6 30.3.1.5p7	

1 **MB** = Member body (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	(3)	4	5	(6)	(7)
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Secretariat observations** on each comment submitted
		3		30.3.1.5p7 30.6.4p7 30.6.9p5 30.6.10.1p23 Rationale: Happens-before is defined in 1.10p11 in a way that (deliberately) does not make it explicitly transitively closed. Adding edges to happens-before directly, as in 27.2.3p2 etc., does not provide transitivity with sequenced-before or any other existing happens-before edge. This lack of transitivity seems to be unintentional.	30.6.4p7 30.6.9p5 30.6.10.1p23	
GB 122	27, 30		Te	See (D) in attachment Appendix 1 - Additional Details	Request the concurrency working group to determine if changes are needed	
GB 123	27.5.3.2	Table 124	Te	Several rows in table 124 specify a Return type of 'OFF_T', which does not appear to be a type defined in this standard.	Resolve outstanding references to the removed type 'OFF_T'.	
US 137	27.7		Te	Several iostreams member functions are incorrectly specified.	See Appendix 1 - Additional Details	
US 138	27.7		te	For istreams and ostreams, the move-constructor does not move-construct, the move-assignment operator does not move-assign, and the swap function does not swap because these operations do not manage the rdbuf() pointer. Useful applications of these operations are prevented both by their incorrect semantics and because they are protected.	In short: reverse the resolution of issue 900, then change the semantics to move and swap the rdbuf() pointer. Add a new protected constructor that takes an rvalue reference to a stream and a pointer to a streambuf, a new protected assign() operator that takes the same arguments, and a new protected partial_swap() function that doesn't swap rdbuf(). See Appendix 1 - Additional Details	
US 139	27.7	1.1.3	te	Resolve issue LWG 1328 one way or the other, but preferably in the direction outlined in the proposed resolution, which, however, is not complete as-is: in any case, the sentry must not set ok_ = false if is.good() ==		

1 **MB** = Member body (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	(3)	4	5	(6)	(7)
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Secretariat observations** on each comment submitted
				false, otherwise istream::seekg, being an unformatted input function, does not take any action because the sentry object returns false when converted to type bool. Thus, it remains impossible to seek away from end of file.		
US 140	27.8		te	It should be possible to construct a stringstream with a specific allocator.	Add an allocator_type and overloaded constructors that take an Allocator argument to basic_stringbuf, basic_istringstream, basic_ostringstream, and basic_stringstream. The semantics of allocator propagation should be the same as if the stringbuf contained an embedded basic_string using the same allocator.	
GB 124	27.8.1.3	3	Te	N3092 27.8.1.3 Member functions contains this text specifying the postconditions of basic_stringbuf::str(basic_string): "Postconditions: If mode & ios_base::out is true, pbase() points to the first underlying character and epptr() >= pbase() + s.size() holds; in addition, if mode & ios_base::in is true, pptr() == pbase() + s.data() holds, otherwise pptr() == pbase() is true. [...]" Firstly, there's a simple mistake: It should be pbase() + s.length(), not pbase() + s.data(). Secondly, it doesn't match existing implementations. As far as I can tell, GCC 4.5 does not test for mode & ios_base::in in the second part of that sentence, but for mode & (ios_base::app | ios_base_ate), and Visual C++ 9 for mode & ios_base::app. Besides, the wording of the C++0x draft doesn't make any sense to me. I suggest changing the second part of the sentence to one of the following: Replace ios_base::in with (ios_base::ate | ios_base::app), but this would require Visual C++ to change (replacing only with ios_base::ate would require GCC to change, and would make ios_base::app completely useless with stringstreams): in addition, if mode & (ios_base::ate | ios_base::app) is true, pptr() == pbase() + s.length() holds, otherwise pptr()		

1 **MB** = Member body (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	(3)	4	5	(6)	(7)
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Secretariat observations** on each comment submitted
				== pbase() is true. Leave pptr() unspecified if mode & ios_base::app, but not mode & ios_base::ate (implementations already differ in this case, and it's always possible to use ios_base::ate to get the effect of appending, so it's not necessary to require any implementation to change): in addition, if mode & ios_base::ate is true, pptr() == pbase() + s.length() holds, if neither mode & ios_base::ate nor mode & ios_base::app is true, pptr() == pbase() holds, otherwise pptr() >= pbase() && pptr() <= pbase() + s.length() (which of the values in this range is unspecified). Slightly stricter: in addition, if mode & ios_base::ate is true, pptr() == pbase() + s.length() holds, if neither mode & ios_base::ate nor mode & ios_base::app is true, pptr() == pbase() holds, otherwise pptr() == pbase() || pptr() == pbase() + s.length() (which of these two values is unspecified). A small table might help to better explain the three cases. BTW, at the end of the postconditions is this text: "egptr() == eback() + s.size() hold". Is there a perference for basic_string::length or basic_string::size? It doesn't really matter, but it looks a bit inconsistent.		
CA 4	27.8.2	various	te	Subclause 27.9.2 [c.files] specifies that <cinttypes> has declarations for abs() and div(); however, the signatures are not present in this subclause. The signatures proposed under TR1 ([tr.c99.inttypes]) are not present in FCD (unless if intmax_t happened to be long long). It is unclear as to which, if any of the abs() and div() functions in [c.math] are meant to be declared by <cinttypes>. This subclause mentions imaxabs() and imaxdiv(). These functions, among other things, are not specified in FCD to be the functions from Subclause 7.8 of the C Standard. Finally, <cinttypes> is not specified in FCD to include <cstdint> (whereas <inttypes.h> includes <stdint.h> in C).	Please clarify.	
JP 14	28.4		TL	Support of char16_t/char32_t is insufficient. The <regex> does not have typedefs for char16_t/char32_t.	Add typedefs below typedef basic_regex<char16_t> u16regex;	

1 **MB** = Member body (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	(3)	4	5	(6)	(7)
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Secretariat observations** on each comment submitted
				The reason we need this typedefs is, because anybody may define exactly same type with different typedef names. Doesn't <locale> offer enough operations which is required by regex implementation?	typedef basic_regex<char32_t> u32regex;	
GB 127	28.5.2		Te	The Bitmask Type requirements in 17.5.2.1.3 p3 say that all elements on a bitmask type have distinct values, but 28.5.2 defines regex_constants::match_default and regex_constants::format_default as elements of the bitmask type regex_constants::match-flag_type, both with value 0. This is a contradiction.	One of the bitmask elements should be removed from the declaration and should be defined separately, in the same manner as ios_base::adjustfield, ios_base::basefield and ios_base::floatfield are defined by 27.5.2.1.2p2 and Table 120. These are constants of a bitmask type, but are not distinct elements, they have more than one value set in the bitmask. regex_constants::format_default should be specified as a constant with the same value as regex_constants::match_default.	
JP 50	28.5.2	1	E	Representations of reference link are not unified. Most reference links to clause (table) number, say X, are in the form "Clause X" ("Table X") capitalized, and subsection Y.Y.Y is referenced with its number only in the form "Y.Y.Y". Whether they are parenthesized or not depends on the context. However there are some notations "(Z)" consisting of only a number Z in parentheses to confer Clause or Table number Z.	Change "table 136" to "Table 136".	
JP 51	28.5.3	1	E	Representations of reference link are not unified. Most reference links to clause (table) number, say X, are in the form "Clause X" ("Table X") capitalized, and subsection Y.Y.Y is referenced with its number only in the form "Y.Y.Y". Whether they are parenthesized or not depends on the context. However there are some notations "(Z)" consisting of only	Change "table 137" to "Table 137".	

1 **MB** = Member body (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	(3)	4	5	(6)	(7)
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Secretariat observations** on each comment submitted
				a number Z in parentheses to confer Clause or Table number Z.		
US 141	28.8		te	std::basic_regex should have an allocator for all the reasons that a std::string does. For example, I can use boost::interprocess to put a string or vector in shared memory, but not a regex.	Add allocators to regexes	
GB 125	28.10.3	2	Te	The term "target sequence" is not defined.	Replace "target sequence" with "string being searched/matched"	
GB 126	28.10.3		Te	It's unclear how match_results should behave if it has been default-constructed. The sub_match objects returned by operator[], prefix and suffix cannot point to the end of the sequence that was searched if no search was done. The iterators held by unmatched sub_match objects might be singular.	Add to match_results::operator[], match_results::prefix and match_results::suffix: Requires: !empty()	
JP 52	28.11.2	3	E	Representations of reference link are not unified. Most reference links to clause (table) number, say X, are in the form "Clause X" ("Table X") capitalized, and subsection Y.Y.Y is referenced with its number only in the form "Y.Y.Y". Whether they are parenthesized or not depends on the context. However there are some notations "(Z)" consisting of only a number Z in parentheses to confer Clause or Table number Z.	Change "table 139" to "Table 139".	
JP 95	28.11.3	1	E	The section number "(24.1.4)" for "Bidirectional Iterator" is wrong. The correct one is "(24.2.6)". In addition, it is written as normal text, but it should be embedded as a link to the section.	Change "(24.1.4)" to "(24.2.6)" and make it a link to section (24.2.6).	
JP 53	28.11.3	3	E	Representations of reference link are not unified. Most reference links to clause (table) number, say X, are in the form "Clause X" ("Table X") capitalized, and subsection Y.Y.Y is referenced with its number only in the form "Y.Y.Y". Whether they are parenthesized or not depends on the context.	Change "table 140" to "Table 140".	

1 **MB** = Member body (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	(3)	4	5	(6)	(7)
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Secretariat observations** on each comment submitted
				However there are some notations "(Z)" consisting of only a number Z in parentheses to confer Clause or Table number Z.		
JP 54	28.13	6	E	Representations of reference link are not unified. Most reference links to clause (table) number, say X, are in the form "Clause X" ("Table X") capitalized, and subsection Y.Y.Y is referenced with its number only in the form "Y.Y.Y". Whether they are parenthesized or not depends on the context. However there are some notations "(Z)" consisting of only a number Z in parentheses to confer Clause or Table number Z.	Change "table 135" to "Table 135".	
GB 128	29		Ge	WG14 has made some late changes to their specification of atomics, and care should be taken to ensure that we retain a common subset of language/library syntax to declare headers that are portable to both languages. Ideally, such headers would not require users to define their own macros, especially not macros that map to keywords (which remains undefined behaviour)	Depends on result of the review of WG14 work, which is expected to be out to ballot during the time wg21 is resolving its own ballot comments. Liaison may also want to file comments in WG14 to ensure compatibity from both sides.	
CH 22	29		te	WG14 currently plans to introduce atomic facilities that are intended to be compatible with the facilities of clause 29. They should be compatible.	Make sure the headers in clause 29 are defined in a way that is compatible with the planned C standard.	
GB 129	29	Table 143	Te	Table 143 lists the typedefs for various atomic types corresponding to the various standard integer typedefs, such as atomic_int_least8_t for int_least8_t, and atomic_uint_fast64_t for uint_fast64_t. However, there are no atomic typedefs corresponding to the fixed-size standard typedefs int8_t, int16_t, and so forth.	Add the following entries to table 143: atomic_int8_t => int8_t (optional), atomic_int16_t => int16_t (optional), atomic_int32_t => int32_t (optional), atomic_int64_t => int64_t (optional), atomic_uint8_t => uint8_t (optional), atomic_uint16_t => uint16_t (optional), atomic_uint32_t => uint32_t (optional), atomic_uint64_t => uint64_t (optional)	

1 **MB** = Member body (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	(3)	4	5	(6)	(7)
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Secretariat observations** on each comment submitted
					These typedefs should be available if the corresponding typedefs from are available.	
CA 16	29.1p1 29.3p8	29.1p1 footnote 343 29.3p8 footnote 344	ed	Radioactivity Footnotes 343 and 344 from 29.1p1 and 29.3p8 read: "Atomic objects are neither active nor radioactive" and "Among other implications, atomic variables shall not decay". We suggest that these be removed - the first is pretty clearly a joke, but it's not obvious that the second doesn't have some technical meaning.	Footnotes 343 and 344 from 29.1p1 and 29.3p8 should be removed.	
US 142	29.1	P2 table 141	ed	Missing 29.8 Fences	Add 29.8 Fences	
US 143	[atomics.syn] 29.2	before 1	ed	There is no free function atomic_compare_exchange_strong for volatile atomic integral types; there is one for non-volatile types.	Add atomic_compare_exchange_strong for volatile integral types to the synopsis.	
US 144	[atomics.syn] 29.2	before 1	ed	The synopsis lists the macros ATOMIC_INTEGRAL_LOCK_FREE and ATOMIC_ADDRESS_LOCK_FREE; the Lock-free Property requirements don't have ATOMIC_INTEGRAL_LOCK_FREE, but have 8 macros for the various integral types.	Change 29.2 [atomics.syn] to match 29.4 [atomics.lockfree].	
US 145	29.2		ed	missing atomic_compare_exchange_strong(volatile ....) and atomic_compare)exchange_strong_explict(...//no volatile *? > bool atomic_compare_exchange_weak(volatile atomic_itype*, integral*, integral); > bool atomic_compare_exchange_weak(atomic_itype*, integral*, integral); > bool atomic_compare_exchange_strong(atomic_itype*,	Repair as suggested	

1 **MB** = Member body (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	(3)	4	5	(6)	(7)
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Secretariat observations** on each comment submitted
				integral*, integral); > bool atomic_compare_exchange_weak_explicit(volatile atomic_itype*, integral*, > integral, memory_order, memory_order); > bool atomic_compare_exchange_weak_explicit(atomic_itype*, integral*, > integral, memory_order, memory_order); > bool atomic_compare_exchange_strong_explicit(volatile atomic_itype*, integral*, > integral, memory_order, memory_order);		
GB 130	29.2		Ed	The synopsis for the <atomic> header lists the macros ATOMIC_INTEGRAL_LOCK_FREE and ATOMIC_ADDRESS_LOCK_FREE. The ATOMIC_INTEGRAL_LOCK_FREE macro has been replaced with a set of macros for each integral type, as listed in 29.4	Replace "#define ATOMIC_INTEGRAL_LOCK_FREE unspecified" with #define ATOMIC_CHAR_LOCK_FREE implementation-defined #define ATOMIC_CHAR16_T_LOCK_FREE implementation-defined #define ATOMIC_CHAR32_T_LOCK_FREE implementation-defined #define ATOMIC_WCHAR_T_LOCK_FREE implementation-defined #define ATOMIC_SHORT_LOCK_FREE implementation-defined #define ATOMIC_INT_LOCK_FREE implementation-defined #define ATOMIC_LONG_LOCK_FREE implementation-defined #define ATOMIC_LLONG_LOCK_FREE implementation-defined	
US 146	29.2	syn 29.4	ed	The ATOMIC_..._LOCK_FREE macros have not had changes applied.	Change to match 29.4/0.	
US 147	29.2	syn 29.7	ed	The declaration of ATOMIC_VAR_INIT should be referenced to section 29.6 [atomics.types.operations], not 29.7.	Change it to 29.6.	

1 **MB** = Member body (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

NOTE Columns 1, 2, 4, 5 are compulsory.

Template for comments and secretariat observations

Date: 5 Oct 2010	Document: **N3141 FCD 14882 Ballot Comments**

1	2	(3)	4	5	(6)	(7)
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Secretariat observations** on each comment submitted
US 148	29.2	syn 29.7	ed	The definition of ATOMIC_VAR_INIT should be 'implementation defined' rather than 'see below'.	Change it to 'implementation defined'.	
US 149	29.2	syn 29.5.1	ed	The synopsis is missing the atomic_init function declarations for the bool, integral and address types.	Copy them from 29.5.1.	
US 150	29.2	syn 29.5.1	ed	There are missing function prototypes bool atomic_compare_exchange_strong(volatile atomic_itype*, integral*, integral); and integral atomic_fetch_add(volatile atomic_itype*, integral);	Add them.	
US 151	29.2	syn 29.5.1	ed	There is a duplicate function declaration of integral atomic_fetch_or(volatile atomic_itype*, integral);	Remove the volatile qualifier from the second declaration.	
US 152	29.3	para 1	ed	The table shows no disinct meaning for memory_order_seq_cst.	Add another bullet: "- memory_order_seq_cst: See below."	
GB 131	29.3	8	Te	See (H) in attachment Appendix 1 - Additional Details	Request the concurrency working group to determine if changes are needed. Consider changing the use of "sequence" in 29.3	
CA 21	29.3p8 1.9p13	29.3p8 1.9p13	Te	Overlapping evaluations are allowed 29.3p8 states: "An atomic store shall only store a value that has been computed from constants and program input values by a finite sequence of program evaluations, such that each evaluation observes the values of variables as computed by the last prior assignment in the sequence."	Please clarify.	

1 **MB** = Member body (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	(3)	4	5	(6)	(7)
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Secretariat observations** on each comment submitted

				... but 1.9p13 states:
				
				"If A is not sequenced before B and B is not
				
				sequenced before A, then A and B are unsequenced.
				
				[Note: The execution of unsequenced evaluations can
				
				overlap. -end note]"
				
				Overlapping executions can make it impossible to construct the sequence described in 29.3p8. We are not sure of the intention here and do not offer a suggestion for change, but note that 29.3p8 is the condition that prevents out-of-thin-air reads.
				
				For an example, suppose we have a function invocation f(e1,e2). The evaluations of e1 and e2 can overlap. Suppose that the evaluation of e1 writes y and reads x whereas the evaluation of e2 reads y and writes x, with reads-from edges as below (all this is within a single thread).
				
				`e1 e2`
				
				`Wrlx y-- --Wrlx x`
				`rf\ /rf`
				`X`
				`/ \`
				`Rrlx x<- ->Rrlx y`
				
				This seems like it should be allowed, but there seems to be no way to produce a sequence of evaluations with the property above.
				
				In more detail, here the two evaluations, e1 and e2, are being executed as the arguments of a function and are		

1 **MB** = Member body (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	(3)	4	5	(6)	(7)
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Secretariat observations** on each comment submitted
				consequently not sequenced-before each other. In practice we'd expect that they could overlap (as allowed by 1.9p13), with the two writes taking effect before the two reads. However, if we have to construct a linear order of evaluations, as in 29.3p8, then the execution above is not permited. Is that really intended?		
US 153	[atomics.lockfree] 29.4	before 1	ed	The macros are all specified as "implementation-deifned"; they should be "see below", since the required values are spelled out.	Change the definitions of the macros in 29.4 [atomics.lockfree] from "implementation-defined" to "see below".	
US 154	[atomics.lockfree] 29.4	before 1	te	There is no ATOMIC_BOOL_LOCK_FREE macro.	Add ATOMIC_BOOL_LOCK_FREE to 29.4 [atomics.lockfree] and to 29.2 [atomics.syn]	
US 155	29.4	para 3	ed	The 'via' 'by' word pairing is awkward.	Replace 'by' with 'via' in 'communication via memory ... and by memory'.	
CA 1	29.4, 29.6 29.7	various	te	All ATOMIC_... macros should be prefixed with STD_ as in STD_ATOMIC_... to indicate they are STD macros as other standard macros. The rationale that they all seem too long seems weak.	This covers the following macros which we suggest prepending with STD_: 29.4: #define ATOMIC_CHAR_LOCK_FREE *implementation-defined* #define ATOMIC_CHAR16_T_LOCK_FREE *implementation-defined* #define ATOMIC_CHAR32_T_LOCK_FREE *implementation-defined* #define ATOMIC_WCHAR_T_LOCK_FREE *implementation-defined* #define ATOMIC_SHORT_LOCK_FREE *implementation-defined* #define ATOMIC_INT_LOCK_FREE *implementation-defined* #define ATOMIC_LONG_LOCK_FREE *implementation-defined* #define ATOMIC_LLONG_LOCK_FREE	

1 **MB** = Member body (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	(3)	4	5	(6)	(7)
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Secretariat observations** on each comment submitted
					implementation-defined #define ATOMIC_ADDRESS_LOCK_FREE *implementation-defined* 29.6: #define ATOMIC_VAR_INIT(value) *see below* 29.7: #define ATOMIC_FLAG_INIT *see below*	
US 156	[atomics.types .integral] 29.5.1	before 1	ed	The list of member functions for atomic_bool has four versions of compare_exchange_weak taking one memory_order argument; the last two should be compare_exchange_strong.	Change the last two member functions comapare_exchange_weak taking two memory_order arguments to compare_exchange_strong.	
JP 55	29.5.1	1	E	Representations of reference link are not unified. Most reference links to clause (table) number, say X, are in the form "Clause X" ("Table X") capitalized, and subsection Y.Y.Y is referenced with its number only in the form "Y.Y.Y". Whether they are parenthesized or not depends on the context. However there are some notations "(Z)" consisting of only a number Z in parentheses to confer Clause or Table number Z.	Change "table 142" to "Table 142".	

GB 132	29.5.1		Te	The atomic_itype types and atomic_address have two overloads of operator= --- one is volatile qualified, and the other is not. atomic_bool only has the volatile qualified version:
				bool operator=(bool) volatile;
				On a non-volatile-qualified object this is ambiguous with the deleted copy-assignment operator
				atomic_bool& operator=(atomic_bool const&) = delete;
				due to the need for a single standard conversion in each case when assigning a bool to an atomic_bool as in:
				atomic_bool b;
				b=true;
					Add the "bool operator=(bool);" overload to atomic_bool in 29.5.1	

1 **MB** = Member body (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	(3)	4	5	(6)	(7)
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Secretariat observations** on each comment submitted
				The conversions are atomic_bool& -> atomic_bool volatile& vs bool -> atomic_bool		
US 157	[atomics.types.integral] 29.5.1	before 1	ed	atomic_bool has a volatile assignment operator but not a non-volatile operator The other integral types have both..	Add a non-volatile assignment operator to atomic_bool.	
US 158	29.5.1	para 0	ed	There is a space before the second & in the declaration atomic_itype& operator=(const atomic_itype &) = delete;	Remove the space.	
US 159	29.5.1		Editorial	Last 2 should be compare_exchane_strong > bool compare_exchange_weak(bool&, bool, memory_order, memory_order) volatile; > bool compare_exchange_weak(bool&, bool, memory_order, memory_order); > bool compare_exchange_strong(bool&, bool, memory_order, memory_order) volatile; > bool compare_exchange_strong(bool&, bool, memory_order, memory_order); > bool compare_exchange_weak(bool&, bool, memory_order = memory_order_seq_cst) volatile; > bool compare_exchange_weak(bool&, bool, memory_order = memory_order_seq_cst); > bool compare_exchange_weak(bool&, bool, memory_order = memory_order_seq_cst) volatile; > bool compare_exchange_weak(bool&, bool, memory_order = memory_order_seq_cst);	Fix last 2	
US 160	[atomic.types.integral] 29.5.1	1	te	The last sentence of 29.5.1 [atomics.types.integral]/1 says "Table 143 shows typedefs to atomic integral classes and the corresponding typedefs." That's nice, but nothing says these are supposed to be part of the implementation, and they are not listed in the synopsis.	Remove Table 143 and the last sentence of 29.5.1 [atomics.types.integral]/1.	
US 161	[atomic.types.address] 29.5.2	before 1	te	atomic_address has operator+= and operator-=, but no operator++ or operator--. The template specialization atomic<Ty*> has all of them.	Add operator++(int) volatile, operator++(int), operator++() volatile, operator++(), operator--(int) volatile, operator--(int), operator--() volatile, and	

1 **MB** = Member body (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	(3)	4	5	(6)	(7)
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Secretariat observations** on each comment submitted
					operator--() to atomic_address.	
US 162	[atomics.types.address] 29.5.2		te	The compare_exchange_weak and compare_exchange_strong member functions that take const void* arguments lead to a silent removal of const, because the load member function and other acessors return the stored value as a void*.	Remove the const void* overloads of compare_exchange_weak and compare_exchange_strong	

US 163	[atomics.type.address], [atomics.types.generic]
	29.5.2, 29.5.3
			te	Requiring atomic<Ty*> to be derived from atomic_address breaks type safety:
				atomic<double*> ip;
				char ch;
				atomic_store(&ip, &ch);
				*ip.load() = 3.14159;
				The last line overwrites ch with a value of type double
					Remove the requirement that atomic<Ty*> be derived from atomic_address.	

US 164	[atomics.types.address] 29.5.2	before 1	te	atomic_address has member functions compare_exchange_weak and compare_exchange_strong that take arguments of type const void*, in addition to the void* versions. If these member functions survive, there should be corresponding free functions.	Add atomic_compare_exchange_weak and atomic_compare_exchange_strong free functions taking pointers to volatile and non-volatile atomic_address objects and const void* arguments.	
JP 56	29.5.3	3	E	Representations of reference link are not unified. Most reference links to clause (table) number, say X, are in the form "Clause X" ("Table X") capitalized, and subsection Y.Y.Y is referenced with its number only in the form "Y.Y.Y". Whether they are parenthesized or not depends on the context. However there are some notations "(Z)" consisting of only a number Z in parentheses to confer Clause or Table number Z.	Change "table 142 or table 143" to "Table 142 or Table 143".	
GB 133	29.5.3		Te	The free functions that operate on atomic_address can be used to store a pointer to an unrelated type in an	Overload the atomic_store, atomic_exchange and atomic_compare_exchange_[weak/strong]	

1 **MB** = Member body (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	(3)	4	5	(6)	(7)
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Secretariat observations** on each comment submitted

				atomic<T*> without a cast. e.g.
				int i;
				atomic<int*> ai(&i);
				string s;
				atomic_store(&ai,&s);
					operations for atomic<T*> to allow storing only pointers to T:
					
					template<typename T>
					void atomic_store(atomic<T*>&,T*);
					
					template<typename T>
					void atomic_store(atomic<T*>&,void*) = delete;
					
					template<typename T>
					void
					atomic_store_explicit(atomic<T*>&,T*,memory_order);
					
					template<typename T>
					void
					atomic_store_explicit(atomic<T*>&,void*,memory_order) = delete;
					
					template<typename T>
					T* atomic_exchange(atomic<T*>&,T*);
					
					template<typename T>
					T* atomic_exchange(atomic<T*>&,void*) = delete;
					
					template<typename T>
					T*
					atomic_exchange_explicit(atomic<T*>&,T*,memory_order);
					
					template<typename T>
					T*
					atomic_exchange_explicit(atomic<T*>&,void*,memory_order) = delete;
					
					template<typename T>
					T*	

1 **MB** = Member body (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	(3)	4	5	(6)	(7)
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Secretariat observations** on each comment submitted

					atomic_compare_exchange_weak(atomic<T*>&,T**,T*);
					
					template<typename T>
					T*
					atomic_compare_exchange_weak(atomic<T*>&,void**,void*) = delete;
					
					template<typename T>
					T*
					atomic_compare_exchange_weak_explicit(atomic<T*>&,T**,T*,memory_order);
					
					template<typename T>
					T*
					atomic_compare_exchange_weak_explicit(atomic<T*>&,void**,void*,memory_order) = delete;
					
					template<typename T>
					T*
					atomic_compare_exchange_strong(atomic<T*>&,T**,T*);
					
					template<typename T>
					T*
					atomic_compare_exchange_strong(atomic<T*>&,void**,void*) = delete;
					
					template<typename T>
					T*
					atomic_compare_exchange_strong_explicit(atomic<T*>&,T**,T*,memory_order);
					
					template<typename T>
					T*
					atomic_compare_exchange_strong_explicit(atomic<T*>&,void**,void*,memory_order) = delete;	

1 **MB** = Member body (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	(3)	4	5	(6)	(7)
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Secretariat observations** on each comment submitted
US 165	29.5.3	Paragraph 23	ed	“is the same that same as that of” is not grammatical (and is not clear)		
US 166	29.6	para 2	ed	The first three bullets seem to be missing terminal punctuation.	Add semicolons to ends of the bullets.	
US 167	29.6	para 3	ed	The first three bullets seem to be missing terminal punctuation.	Add semicolons to ends of the bullets.	
US 168	29.6	para 4	te	The definition of the default constructor needs exposition.	Add a new paragraph: A::A() = default; Effects: Leaves the atomic object in an uninitialized state. [Note: These semantics ensure compatiblity with C. --end note]	
US 169	29.6	para 5	ed	The definition of ATOMIC_VAR_INIT should be 'implementation defined' rather than 'see below'.	Change it to 'implementation defined'.	
US 170	29.6	para 6	ed	The definition of atomic_init should be grouped with the value constructor.	Move the atomic_init definition to just after the constructor definition.	

GB 134	29.6	5	Ed	Some of the declarations of is_lock_free seem to be missing return types.	Replace:
					A::is_lock_free() const volatile;
					A::is_lock_free() const;
					With:
					bool A::is_lock_free() const volatile;
					bool A::is_lock_free() const;	

US 171	29.6	para 6	te	The atomic_init definition "Non-atomically assigns the value" is not quite correct, as the atomic_init purpose is initialization.	Change "Non-atomically assigns the value desired to *object." with "Initializes *object with value desired". Add the note: "[Note: This function should only be applied to objects that have been default constructed. These semantics ensure compatibility with C. --end note]"	
US 172	29.6	para 9, 13, 17, 20	ed	The order specifications are incomplete because the non-_explicit functions do not have such parameters.	Add a new sentence: "If the program does not specify an order, it shall be memory_order_seq_cst." Or perhaps: "The non-_explicit non-member functions shall affect memory as though they were _explicit with memory_order_seq_cst."	

1 **MB** = Member body (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	(3)	4	5	(6)	(7)
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Secretariat observations** on each comment submitted
US 173	29.6	para 20	ed	Misspelling.	Replace "operations ate atomic" with "operations are atomic".	
US 174	29.6	para 22	ed	The first note is about effects, not returns.	Move this note to just after the Effects paragraph.	
US 175	29.6	para 23	ed	The first sentence is grammatically incorrect.	Replace the sentence with two: "The weak compare-and-exchange operations may fail spuriously. That is, it may return false while leaving the contents of memory pointed to by expected the same as it was before the operation."	
CH 23	29.6	p23	ed	The first sentence has non-English syntax.	Change to "The weak compare-and-exchange operations may fail spuriously, that is, return false while leaving the contents of memory pointed to by expected unchanged."	
US 176	29.6	para 23	ed	Unintended paragraph break.	Proposal: Remove the paragraph break between "will be in a loop." and "When a compare-and-exchange is in a loop,".	
US 177	[atomics.types .operations] 29.6	23	ed	The first sentence of this paragraph doesn't make sense.	Figure out what it's supposed to say, and say it.	
GB 135	29.6	23	Ed	The first sentence of 29.6p23 was changed by n2992 but now makes no sense: "that is, return false while leaving the contents of memory pointed to by expected before the operation is the same that same as that of the object and the same as that of expected after the operation." There's a minor editorial difference between n2992 ("is that same as that" vs "is the same that same as that") but neither version makes sense. Also, the remark talks about "object" which should probably be "object or this" to cover the member functions which have no object parameter.	Fix the Remark to say whatever was intended.	
GB	29.6		Te	See (K) in attachment Appendix 1 - Additional Details	GB requests normative clarification in 29.6p4 that	

1 **MB** = Member body (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	(3)	4	5	(6)	(7)
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Secretariat observations** on each comment submitted
136					concurrent access constitutes a race, as already done on p6 and p7.	
US 178	29.7	para 7	ed	The sentence "The order argument shall not be memory_order_acquire nor memory_order_acq_rel." is awkwardly phrased.	Change the sentence to "The order argument shall be neither memory_order_acquire nor memory_order_acq_rel."	
US 179	29.8	para 5, 6	te	The fence functions should be extern "C", for C compatibility.	Add extern "C" to their declarations in 29.8 and in 29.2.	
GB 137	29.8	6	Te	Thread fence not only establish synchronizes with relationships, there are semantics of fences that are expressed not in terms of synchronizes with relationships (for example see 29.3p5). These semantics also need to apply to the use of atomic_signal_fence in a restricted way.	Change 29.8p6 to "Effects: equivalent to atomic_thread_fence(order), except that the resulting ordering constraints are established only between a thread and a signal handler executed in the same thread."	
US 180	30.1	para 1	ed	The introductory sentence is missing futures.	Replace "communicate conditions between threads" with "communicate conditions and values between threads".	
GB 138	30.2		Te	The FCD combines the requirements for lockable objects with those for the standard mutex objects. These should be separate. This is LWG issue 1268.	See attached Appendix 1 - Additional Details	
US 181	30.2.4	para 2	te	The timeout operations are under-specified.	Define precise semantics for timeout_until and timeout_for. See Appendix 1 - Additional Details	
US 182	[thread.req.native] 30.2.3		te	native_handle and native-handle_type should be removed. It is implementation-defined whether these names are present in the various thread support classes, and if present, it is implementation-defined what the name native_handle_type refers to. This is exactly what the implementor namespace is for. There is no benefit to programmers from providing a way to portably detect that an implementation provides non-portable extensions. The standard should not reserve these names, with unspecified semantics; it if does, the names will never become portable because implementations will differ on	Remove [thread.req.native] 30.2.3 and remove all mentions of native_handle and native_handle_type.	

1 **MB** = Member body (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	(3)	4	5	(6)	(7)
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Secretariat observations** on each comment submitted
				what they mean.		
DE 23	30.3		te	Predefined macros usually start and end with two underscores, see 16.8 and FDIS 29124 = WG21 N3060 clause 7. __STDCPP_THREADS should blend in.	Change the macro name to __STDCPP_THREADS__.	
US 183	30.3.1		te	There is no way to join a thread with a timeout.	Add join_for and join_until. Or decide one should never join a thread with a timeout since pthread_join doesn't have a timeout version.	
US 184	30.3.1.1	para 2	te	It is unclear when a thread::id ceases to be meaningful. The sentence "The library may reuse the value of a thread::id of a terminated thread that can no longer be joined." implies that some terminated threads can be joined. It says nothing about detached threads.	Require a unique thread::id for every thread that is (1) detached and not terminated or (2) has an associated std::thread object.	
JP 97	30.3.1.5	9	E	In Throw clause, both "Throws: system_error" and "Throws: std::system_error" are used. They should be in a unified way, and we propose to use system_error instead of std::system_error.	Change to: Throws: system_error	
JP 98	30.3.1.5	14	E	In Throw clause, both "Throws: system_error" and "Throws: std::system_error" are used. They should be in a unified way, and we propose to use system_error instead of std::system_error.	Change to: Throws: system_error	
CH 24	30.3.2	p1	te	What would be the value this_thread::get_id() when called from a detached thread?	Add some text to clarify that get_id() still returns the same value even after detaching.	
CH 25	30.3.2	p8 and p11	te	Clock related operations are currently not required not to throw. So "Throws: Nothing." is not always true.	Either require clock related operations not to throw (in 20.10) or change the Throws clauses in 30.3.2. Also possibly add a note that abs_time in the past or negative rel_time is allowed.	
US 185	30.4		te	Cooperate with WG14 to improve interoperability between the C++0x and C1x threads APIs. In particular, C1x mutexes should be conveniently usable with a C++0x lock_guard. Performance overheads for this combination should be considered.	Remove C++0x timed_mutex and timed_recursive_mutex if that facilitates development of more compatible APIs.	

1 **MB** = Member body (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	(3)	4	5	(6)	(7)
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Secretariat observations** on each comment submitted
CH 26	30.4		te	Specifications of unlock member functions and unlock mutex requirements are inconsistent wrt to exceptions and pre- and postconditions.	unlock should specifiy the precondition that the current thread "owns the lock", this will make calls without holding the locks "undefined behavior". unlock in [mutex.requirements] should either be noexcept(true) or be allowed to throw system_error like unique_lock::unlock, or the latter should be nothrow(true) and have the precondition owns==true. Furthermore unique_lock's postcondition is wrong in the case of a recursive mutex where owns might stay true, when it is not the last unlock needed to be called.	
CH 27	30.4.1	p18	te	The mutex requirements force try_lock to be noexcept(true). However, where they are used by the generic algorithms, those relax this requirement and say that try_lock may throw. This means the requirement is too stringent, also a non-throwing try_lock does not allow for a diagnostic such as system_error that lock() will give us.	delete p18, adjust 30.4.4 p1 and p4 accordingly	
JP 99	30.4.1	11	E	In Throw clause, both "Throws: system_error" and "Throws: std::system_error" are used. They should be in a unified way, and we propose to use system_error instead of std::system_error.	Change to: Throws: system_error	
US 186	30.4.1	14	te	try_lock does not provide a guarantee of forward progress because it is allowed to spuriously fail.	The standard mutex types must not fail spuriously in try_lock. See Appendix 1 - Additional Details	
US 187	30.4.1	14	ed	Paragraph mentions compare_exchange, which no longer exists.	Change “compare_exchange” to “compare_exchange_weak”.	
JP 57	30.4.1	14	E	Representations of reference link are not unified. Most reference links to clause (table) number, say X, are in the form "Clause X" ("Table X") capitalized, and subsection Y.Y.Y is referenced with its number only in the form "Y.Y.Y". Whether they are parenthesized or not	Change "(29)" to "(Clause 29)".	

1 **MB** = Member body (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	(3)	4	5	(6)	(7)
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Secretariat observations** on each comment submitted
				depends on the context. However there are some notations "(Z)" consisting of only a number Z in parentheses to confer Clause or Table number Z.		
US 188	30.4.1	20, 21	te	Mutex requirements should not be bound to threads	See Appendix 1 - Additional Details	
JP 58	30.4.1.1	3	E	Representations of reference link are not unified. Most reference links to clause (table) number, say X, are in the form "Clause X" ("Table X") capitalized, and subsection Y.Y.Y is referenced with its number only in the form "Y.Y.Y". Whether they are parenthesized or not depends on the context. However there are some notations "(Z)" consisting of only a number Z in parentheses to confer Clause or Table number Z.	Change "(9)" to "(Clause 9)".	
US 189	30.4.1.1 30.4.1.2		te	mutex and recursive_mutex should have an is_locked() member function. is_locked allows a user to test a lock without acquiring it and can be used to implement a light-weight try_try_lock.	Add a member function: bool is_locked() const; to std::mutex and std::recursive_mutex. These functions return true if the current thread would not be able to obtain a mutex. These functions do not synchronize with anything (and, thus, can avoid a memory fence).	
JP 59	30.4.1.2	2	E	Representations of reference link are not unified. Most reference links to clause (table) number, say X, are in the form "Clause X" ("Table X") capitalized, and subsection Y.Y.Y is referenced with its number only in the form "Y.Y.Y". Whether they are parenthesized or not depends on the context. However there are some notations "(Z)" consisting of only a number Z in parentheses to confer Clause or Table number Z.	Change "(9)" to "(Clause 9)".	

1 **MB** = Member body (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	(3)	4	5	(6)	(7)
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Secretariat observations** on each comment submitted
JP 60	30.4.2.1	2	E	Representations of reference link are not unified. Most reference links to clause (table) number, say X, are in the form "Clause X" ("Table X") capitalized, and subsection Y.Y.Y is referenced with its number only in the form "Y.Y.Y". Whether they are parenthesized or not depends on the context. However there are some notations "(Z)" consisting of only a number Z in parentheses to confer Clause or Table number Z.	Change "(9)" to "(Clause 9)".	
JP 61	30.4.2.2	2	E	Representations of reference link are not unified. Most reference links to clause (table) number, say X, are in the form "Clause X" ("Table X") capitalized, and subsection Y.Y.Y is referenced with its number only in the form "Y.Y.Y". Whether they are parenthesized or not depends on the context. However there are some notations "(Z)" consisting of only a number Z in parentheses to confer Clause or Table number Z.	Change "(9)" to "(Clause 9)".	
JP 70	30.4.3.2	18	E	Constant width font should be used for "std::system_error"s in the paragraph as described in Syntax notation (1.6).	Change the font for "std::system_error" to constant width type. Throws: **system_error** when an exception is required	
JP 100	30.4.3.2.2	18	E	In Throw clause, both "Throws: system_error" and "Throws: std::system_error" are used. They should be in a unified way, and we propose to use system_error instead of std::system_error.	Change to: Throws: system_error	
US 190	30.4.5.2	para 2, 3	te	The term "are serialized" is never defined.	Remove the sentence with "are serialized" from paragraph 2. Add "Calls to call_once on the same once_flag object shall not introduce data races (17.6.4.8)." to paragraph 3.	

1 **MB** = Member body (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	(3)	4	5	(6)	(7)
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Secretariat observations** on each comment submitted
JP 101	30.4.5.2	4	E	In Throw clause, both "Throws: system_error" and "Throws: std::system_error" are used. They should be in a unified way, and we propose to use system_error instead of std::system_error.	Change to: Throws: system_error	
US 191	30.5		te	The condition variable wait_for returning cv_status is insufficient.	Return a duration of timeout remaining instead. See Appendix 1 - Additional Details	
GB 139	30.5	7	Ed	The text says "... ownership of the lock as the current thred exits, ...", with "thread" misspelled.	Replace "thred" with "thread"	
GB 140	30.5	9	Ed	The text says "... waiting threds ..." with "threads" misspelled.	Replace "threds" with "threads".	
CH 28	30.5.1		te	Requiring wait_until makes it impossible to implement condition_variable correctly using respective objects provided by the operating system (i.e. implementing the native_handle() function) on many platforms (e.g. POSIX, Windows, MacOS X) or using the same object as for the condition variable proposed for C.	Remove the wait_until functions or make them at least conditionally supported.	
JP 102	30.5.1	3	E	In Throw clause, both "Throws: system_error" and "Throws: std::system_error" are used. They should be in a unified way, and we propose to use system_error instead of std::system_error.	Change to: Throws: system_error	
CH 30	30.5.1 and 30.5.2	p13, last bullet, and correspondin g paragraphs in all wait functions	te	If lock.lock() throws an exception, the postcondition can not be generally achieved.	Either state that the postcondition might not be achieved, depending on the error condition, or state that terminate() is called in this case.	
JP 103	30.5.1	15	E	In Throw clause, both "Throws: system_error" and "Throws: std::system_error" are used. They should be in a unified way, and we propose to use system_error instead of std::system_error.	Change to: Throws: system_error	

1 **MB** = Member body (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	(3)	4	5	(6)	(7)
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Secretariat observations** on each comment submitted
CH 31	30.5.1 and 30.5.2	p19, third bullet, and corresponding paragraphs in all wait_for/wait_until functions	ed	The sentences contain superflous "or"s.	Change "The function will unblock when signaled by a call to notify_one() or a call to notify_all(), if abs_time <= Clock::now(), or spuriously." to "The function will unblock when signaled by a call to notify_one(), a call to notify_all(), if abs_time <= Clock::now(), or spuriously."	
JP 104	30.5.1	22	E	In Throw clause, both "Throws: system_error" and "Throws: std::system_error" are used. They should be in a unified way, and we propose to use system_error instead of std::system_error.	Change to: Throws: system_error	
US 192	30.5.1	para 26	ed	The identifier cv_status::no_timeout is not in code font.	Change it to code font.	
CH 29	30.5.1 and 30.5.2	p34 and p28, respectively	te	It is unclear if a spurious wake-up during the loop and re-entering of the blocked state due to a repeated execution of the loop will adjust the timer of the blocking with the respect to the previously specified rel_time value.	Make it clear (e.g. by a note) that when re-executing the loop the waiting time when blocked will be adjusted with respect to the elapsed time of the previous loop executions.	
US 193	30.5.1, 30.5.2		te	Condition variables preclude a wakeup optimization.	Change condition_variable to allow such optimization. See Appendix 1 - Additional Details	
JP 62	30.5.1	1	E	Representations of reference link are not unified. Most reference links to clause (table) number, say X, are in the form "Clause X" ("Table X") capitalized, and subsection Y.Y.Y is referenced with its number only in the form "Y.Y.Y". Whether they are parenthesized or not depends on the context. However there are some notations "(Z)" consisting of only a number Z in parentheses to confer Clause or Table number Z.	Change "(9)" to "(Clause 9)".	
CH	30.5.2		te	Given that the lock type can be something the underlying doesn't know 'native_handle()' is probably	Consider the removal of 'native_handle()'.	

1 **MB** = Member body (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	(3)	4	5	(6)	(7)
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Secretariat observations** on each comment submitted
32				unimplementable on essentially all platforms.		
JP 105	30.5.2	12	E	In Throw clause, both "Throws: system_error" and "Throws: std::system_error" are used. They should be in a unified way, and we propose to use system_error instead of std::system_error.	Change to: Throws: system_error	
JP 106	30.5.2	18	E	In Throw clause, both "Throws: system_error" and "Throws: std::system_error" are used. They should be in a unified way, and we propose to use system_error instead of std::system_error.	Change to: Throws: system_error	
CH 33	30.5.2	before p25	ed	Template function wait_until is missing class Clock template parameter.	Change "template <class Lock, class Duration, class Predicate>" to "template <class Lock, class Clock, class Duration, class Predicate>".	

JP 96	30.5.3	2	E	Inconsistency between 30.4 paragraph 1 and 30.4.3 paragraph 2.
				In 30.4 paragraph 1:
				namespace std {
				...
				constexpr defer_lock_t defer_lock { };
				constexpr try_to_lock_t try_to_lock { };
				constexpr adopt_lock_t adopt_lock { };
				}
				In 30.4.3 paragraph 2:
				namespace std {
				...
				extern const defer_lock_t defer_lock { };
				extern const try_to_lock_t try_to_lock { };
				extern const adopt_lock_t adopt_lock { };
				}
				The writer seems to have forgotten to rewrite latter cases, so 30.4.3 paragraph 2 should be modified as this proposal.
					Change "extern const" to "constexpr".
					namespace std {
					...
					constexpr defer_lock_t defer_lock { };
					constexpr try_to_lock_t try_to_lock { };
					constexpr adopt_lock_t adopt_lock { };
					}	

US 194	30.6		te	The specification for managing associated asynchronous state is confusing, sometimes omitted, and redundantly	Define terms-of-art for releasing, making ready, and abandoning an associated asynchronous	

1 **MB** = Member body (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	(3)	4	5	(6)	(7)
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Secretariat observations** on each comment submitted
				specified.	state. Use those terms where appropriate. See Appendix 1 - Additional Details	
CH 34	30.6.1	p1	ed	The paragraph is misleading and incorrect wrt to the current specification, since an async call with launch::sync will execute in the same thread.	Change the paragraph to '30.6 describes components that a C++ program can use to retrieve the result (value or exception) of a function that has run in a (potentially different) thread.'	
US 195	30.6.4	para 8	te	The intent and meaning of the paragraph is not apparent.		
CH 35	30.6.4ff		ed/te	The term "associated asynchronous state" is long, ugly and misleading terminology. When introduced we agreed upon that we should come up with a better name. Here it is: "liaison state". Since the state is hidden and provides synchronization of a future with its corresponding promise, we believe "liaison state" is a much better and shorter name (liaison ~ (typically hidden) relationship)	Change all occurrences of "associated asynchronous state" to "liaison state".	
US 196	30.6.5	para 21, 25	te	The term "are serialized" is not defined.	Replace "are serialized" with "shall not introduce a data race (17.6.4.8)".	
US 197	30.6.5	para 21, 25	te	There is no defined synchronization between promise::set_value and future::get.	Replace "[Note: and they synchronize and serialize with other functions through the referred associated asynchronous state. --end note]" with the normative "They synchronize with (1.10) any operation on a future object with the same associated asynchronous state marked ready."	
US 198	30.6.5	22	te	promise::set_exception can be called with a null pointer, but none of the descriptions of the get() functions for the three types of futures say what happens for this case.	Add the following sentence to the end of 30.6.5/22: The behavior of a program that calls set_exception with a null pointer is undefined.	
US 199	[futures.promise] 30.6.5	26ff, 29ff	te	promise::XXX_at_thread_exit functions have no synchronization requirements. Specifying synchronization for these member functions requires coordinating with the words in 30.6.5/21 and 25, which give synchronization requirements for promise::set_value and	Change 30.6.5/21 to mention set_value_at_thread_exit and set_exception_at_thread_exit; with this text, replace 30.6.5/25 and add two new paragraphs, after 30.6.5/28 and 30.6.5/31.	

1 **MB** = Member body (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	(3)	4	5	(6)	(7)
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Secretariat observations** on each comment submitted
				promise::set_exception.		
US 200	30.6.6	para 26	ed	The paragraph is missing the "Returns:" label.	Add the label.	
US 201	[futures.unique_future], [futures.shared_future], [futures.atomic_future], [futures.task] 30.6.6 30.6.7 30.6.8 30.6.10		te	packaged_task provides operator bool() to check whether an object has an associated asynchronous state. The various future types provide a member function valid() that does the same thing. The names of these members should be the same.	Replaced the name packaged_task::operator bool() with packaged_task::valid() in the synopsis (30.6.10 [futures.task]/2) and the member function specification (before 30.6.10.1 [futures.task.members]/15).	
US 202	[futures.atomic_future] 30.6.8	18	te	The note in this paragraph says "unlike future, calling get more than once on the same atomic_future object is well defined and produces the result again." There is nothing in future that says anything negative about calling get more than once.	Remove this note, or add words to the requirements for future that reflect what this note says.	
US 203	[futures.atomic_future] 30.6.8		te	Both future and shared_future specify that calling most member functions on an object for which valid() == false produces undefined behavior. There is no such statement for atomic_future.	Add a new paragraph after 30.6.8 [futures.atomic_future]/2 with the same words as 30.6.7 [futures.shared_future]/3.	
US 204	30.6.8	Paragraph 7-8	te	According to the definition of atomic_future, all members of atomic_future are synchronizing except constructors. However, it would probably be appropriate for a move constructor to be synchronizing on the source object. If not, the postconditions on paragraphs 7-8, might not be satisfied. This may be applicable if a collection of futures are being doled out to a set of threads that process their value.	Make the move constructor for atomic future lock the source	

1 **MB** = Member body (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	(3)	4	5	(6)	(7)
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Secretariat observations** on each comment submitted
US 205	[futures.async] 30.6.9	3	te	The third sentence says "If the invocation is not deferred, a call to a waiting function on an asynchronous return object that shares the associated asynchronous state created by this async call shall block until the associated thread has completed." The next sentence says "If the invocation is not deferred, the join() on the created thread..." Blocking until a thread completes is not necessarily a join.	Decide whether the requirement is to block until finished or to call join, and rewrite to match.	
CH 36	30.6.9 and 30.6.1	<future> synopsis and p3, respectively	te	Providing only three different possible values for the enum launch and saying that launch::any means either launch::sync or launch::async is very restricting. This hinders future implementors to provide clever infrastructures that can simply by used by a call to async(launch::any,...). Also there is no hook for an implementation to provide additional alternatives to launch enumeration and no useful means to combine those (i.e. interpret them like flags). We believe something like async(launch::sync | launch::async, ...) should be allowed and can become especially useful if one could say also something like async(launch::any & ~launch::sync, ...) respectively. This flexibility might limit the features usable in the function called through async(), but it will allow a path to effortless profit from improved hardware/software without complicating the programming model when just using async(launch::any,...)	Change in 30.6.1 'enum class launch' to allow further implementation defined values and provide the following bit-operators on the launch values (operator|, operator&, operator~ delivering a launch value). Note: a possible implementation might use an unsigned value to represent the launch enums, but we shouldn't limit the standard to just 32 or 64 available bits in that case and also should keep the launch enums in their own enum namespace. Change [future.async] p3 according to the changes to enum launch. change --launch::any to "the implementation may choose any of the policies it provides." Note: this can mean that an implementation may restrict the called function to take all required information by copy in case it will be called in a different address space, or even, on a different processor type. To ensure that a call is either performed like launch::async or launch::sync describe one should call async(launch::sync|launch::async,...)	
JP 107	30.6.9	3	E	Typo, "." should be ">". decay_copy(std::forward<Args.(args))...	Correct typo. decay_copy(std::forward<Args>(args))…	
JP 108	30.6.9	3	E	<Arg> should be <Args>. launch::sync — Stores decay_copy(std::forward<F>(f)) and	Change to: launch::sync — Stores decay_copy(std::forward<F>(f)) and	

1 **MB** = Member body (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	(3)	4	5	(6)	(7)
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Secretariat observations** on each comment submitted
				decay_copy(std::forward<Arg>(args))...	decay_copy(std::forward<Args>(args))...	
US 206	[futures.task. members] 30.6.10.1	27, 28	ed	The text of paragraph 27 says that reset() moves the function object, but the text of paragraph 28 talks about exceptions thrown by the copy constructor.	Change "copy constructor" to "move constructor" in 30.6.10.1/28, bullet 2.	
US 207	[futures.task. members] 30.6.10.1	1-8	te	The constructor that takes R(*)(ArgTypes...) is not needed; the constructor that takes a callable type works for this argument type. More generally, the constructors for packaged_task should parallel those for function.	Review the constructors for packaged_task and provide the same ones as function, except where inappropriate.	
US 208	[futures.task. members] 30.6.10.1	24-26	te	packaged_task::make_ready_at_thread_exit has no synchronization requirements.	Figure out what the synchronization requirements should be and write them.	
GB 141	Appendix A [gram] paragraph 1		Ed	The links for disambiguation rules go to 6.8, 7.1 and 10.2. Section 8.2 covers ambiguity in declarators, so should be added	Added a link to 8.2 [dcl.ambig.res] to Appendix A p1	
JP 63	A.1	1	E	Representations of reference link are not unified. Most reference links to clause (table) number, say X, are in the form "Clause X" ("Table X") capitalized, and subsection Y.Y.Y is referenced with its number only in the form "Y.Y.Y". Whether they are parenthesized or not depends on the context. However there are some notations "(Z)" consisting of only a number Z in parentheses to confer Clause or Table number Z.	Change "(clause 9)" to "(Clause 9)". Change "(clause 14)" to "(Clause 14)".	
FI 6	D.2 [depr.static]	Paragraph 1	te	The use of static in namespace scope should not be deprecated. Anonymous namespaces are not a sufficient replacement for the functionality.	Strike [depr.static] completely.	
GB 142	D10		Ge	auto_ptr does not appear in the <memory> synopsis and [depr.auto.ptr] doesn't say which header declares it. Conversely, the deprecated binders bind1st etc. are in the <functional> synopsis, this is inconsistent	Either auto_ptr should be declared in the <memory> synopsis, or the deprecated binders should be removed from the <functional> synopsis and appendix D should say which header declares the binders and auto_ptr	

1 **MB** = Member body (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

NOTE Columns 1, 2, 4, 5 are compulsory.

1	2	(3)	4	5	(6)	(7)
MB[1]	**Clause No./ Subclause No./ Annex** (e.g. 3.1)	**Paragraph/ Figure/Table/ Note** (e.g. Table 1)	**Type of com-ment**[2]	**Comment (justification for change) by the MB**	**Proposed change by the MB**	**Secretariat observations** on each comment submitted
JP 109	Annex B		E	Although implementation limits for at_quick_exit() is mentioned in 18.5 paragraph 5, it is not on the list of Implementation quantities.	Add the following line. — Functions registered by at_quick_exit()[32].	
JP 110	Index		E	"local scope" has been renamed to "block scope", but the reference to "local scope" still remains in Index. block scope; see local scope, 37 "local scope" should refer to "block scope".	Change to: local scope; see block scope, 37	

1 **MB** = Member body (enter the ISO 3166 two-letter country code, e.g. CN for China; comments from the ISO/CS editing unit are identified by ******)

2 **Type of comment:** **ge** = general **te** = technical **ed** = editorial

NOTE Columns 1, 2, 4, 5 are compulsory.

APPENDIX 1

ADDITIONAL DETAILS

ISO/IEC FCD 14882, C++0X, National Body Comments

FCD 14882

ADDITIONAL DETAILS TO US BALLOT COMMENTS

FCD 14882
ADDITIONAL DETAILS TO US COMMENTS

US 23:

3.4.5: Global class templates should not hide member templates.

Technical details:

[basic.lookup.classref] p1 says:

> In a class member access expression (5.2.5), if the . or -> token is immediately followed by an identifier followed by a <, the identifier must be looked up to determine whether the < is the beginning of a template argument list (14.3) or a less-than operator. The identifier is first looked up in the class of the object expression. If the identifier is not found, it is then looked up in the context of the entire postfix-expression and shall name a class template. If the lookup in the class of the object expression finds a template, the name is also looked up in the context of the entire postfix-expression and
>
> - if the name is not found, the name found in the class of the object expression is used, otherwise
>
> - if the name is found in the context of the entire postfix-expression and does not name a class template, the name found in the class of the object expression is used, otherwise
>
> - if the name found is a class template, it shall refer to the same entity as the one found in the class of the object expression, otherwise the program is ill-formed.

This means that the following program is ill-formed:

```
#include <set>
using std::set;
struct X {
       template <typename T> void set(const T& value);
};
void foo() {
       X x;
       x.set<double>(3.2);
}
```

That's confusing and unnecessary. The compiler has already done the lookup in X's scope, and the obviously-correct resolution is that one, not the identifier from the postfix-expression's scope. Issue 305 fixed a similar issue for destructor names, but missed member functions.

Proposed resolution:
Strike the end of paragraph 1 starting with "If the lookup in the class of the object expression finds a template," and including all three bullets.

FCD 14882

ADDITIONAL DETAILS TO US COMMENTS

US 26

`Clause 3.7.4`

C++ Sized Deallocation

ISO/IEC JTC1 SC22 WG21 - 2010-05-19 - National Body Comment by Google

Lawrence Crowl, crowl@google.com, Lawrence@Crowl.org

Problem

Within the Final Committee Draft, programmers may define a static member function `operator delete` that takes a size parameter indicating the size of the object to be deleted. The equivalent global `operator delete` is not available. This omission has unfortunate performance consequences.

Modern memory allocators often allocate in size categories, and, for space efficiency reasons, do not store the size of the object near the object. Deallocation then requires searching for the size category store that contains the object. This search can be expensive, particularly as the search data structures are often not in memory caches.

Solution

Permit implementations and programmers to define sized versions of the global `operator delete`. The compiler shall the sized version in preference to the unsized version when the sized version is available.

There are two potential problems with this solution.

- When deleting an incomplete type, there is not size available. In this case, the unsized version must be used. This observation implies that calls to one version must be effectively equivalent to calls to the other version. Excepting the specific deallocation function called, we believe that any programs that would change behavior already have undefined behavior within the standard.
- Existing programs that redefine the global unsized version do not also define the sized version. When an implementation introduces a sized version, the replacement would be incomplete. In this case, the only viable option seems to be for the implementation to emit a diagnostic when the programmer provides an unsized replacement but does not provide a sized replacement. The workaround is to define a sized version that simply calls the unsized version.

  Note, however, that the converse case is not a problem. The programmer may define a both an unsized and a sized version even when the underlying implementation only provides a sized version. The reason is that the two versions must be functionally equivalent.

As a consequence of the second problem, we expect implementations to be somewhat conservative in the introduction of the pre-defined sized versions. On the other hand, programmers may aggressively define the sized versions.

Wording

The proposed wording changes are relative to the Final Committee Draft, N3092.

3.7.4 Dynamic storage duration [basic.stc.dynamic]

Edit within paragraph 2 as follows.

> .... The following allocation and deallocation functions (18.6) are implicitly declared in global scope in each translation unit of a program.
>
> ```
> void* operator new(std::size_t)
> throw(std::bad_alloc);
> void* operator new[](std::size_t)
> throw(std::bad_alloc);
> void operator delete(void*) throw();
> void operator delete[](void*) throw();
> ```
>
> Furthermore, the implementation may implicitly declare the following functions in global scope in each translation unit of a program.
>
> ```
> void operator delete(void*, std::size_t)
> throw();
> void operator delete[](void*, std::size_t)
> throw();
> ```

> If the implementation provides these functions, and if a translation unit provides a definition for an unsized version but not for a sized version, the program is ill-formed.
>
> These implicit declarations introduce only the function names `operator new`, `operator new[]`, `operator delete`, `operator delete[]`. ....

3.7.4.2 Deallocation functions [basic.stc.dynamic.deallocation]

Edit paragraph 2 as follows.

> Each deallocation function shall return `void` and its first parameter shall be `void*`. A deallocation function can have more than one parameter. If there is a declaration of global `operator delete` with exactly two parameters, the second of which has type `std::size_t` (18.2), then that function is a usual (non-placement) deallocation function. Similarly, if there is a declaration of global `operator delete[]` with exactly two parameters, the second of which has type `std::size_t`, then this function is a usual deallocation function. If a class `T` has a member deallocation function named `operator delete` with exactly one parameter, then that function is a usual ~~(non-placement)~~ deallocation function. If class `T` does not declare such an `operator delete` but does declare a member deallocation function named `operator delete` with exactly two parameters, the second of which has type `std::size_t` ~~(18.2)~~, then this function is a usual deallocation function. Similarly, if a class `T` has a member deallocation function named `operator delete[]` with exactly one parameter, then that function is a usual (non-placement) deallocation function. If class `T` does not declare such an `operator delete[]` but does declare a member deallocation function named `operator delete[]` with exactly two parameters, the second of which has type `std::size_t`, then this function is a usual deallocation function. A deallocation function can be an instance of a function template. Neither the first parameter nor the return type shall depend on a template parameter. [*Note:* that is, a deallocation function template shall have a first parameter of type `void*` and a return type of `void` (as specified above). —*end note*] A deallocation function template shall have two or more function parameters. A template instance is never a usual deallocation function, regardless of its signature.

5.3.5 Delete [expr.delete]

Paragraph 1 remains unchanged, though note the restrictions on the delete operand.

> .... The operand shall have a pointer to object type, or a class type having a single non-explicit conversion function (12.3.2) to a pointer to object type. The result has type void. [*Footnote:* This implies that an object cannot be

> deleted using a pointer of type `void*` because void is not an object type. —*end footnote*] ....

paragraph 2 remains unchanged, though note the restriction on inheritance with respect to the `delete` operand.

> .... If it is not a null pointer value, in the first alternative (*delete object*), the value of the operand of `delete` shall be a pointer to a non-array object or a pointer to a subobject (1.8) representing a base class of such an object (Clause 10). If not, the behavior is undefined. In the second alternative (*delete array*), the value of the operand of `delete` shall be the pointer value which resulted from a previous array new-expression. [*Footnote:* For non-zero-length arrays, this is the same as a pointer to the first element of the array created by that new-expression. Zero-length arrays do not have a first element. —*end footnote*] If not, the behavior is undefined. [*Note:* this means that the syntax of the *delete-expression* must match the type of the object allocated by `new`, not the syntax of the *new-expression*. —end note] ....

Paragraph 3 remains unchanged, though note the further restriction on inheritance.

> In the first alternative (*delete object*), if the static type of the object to be deleted is different from its dynamic type, the static type shall be a base class of the dynamic type of the object to be deleted and the static type shall have a virtual destructor or the behavior is undefined. In the second alternative (*delete array*) if the dynamic type of the object to be deleted differs from its static type, the behavior is undefined.

Paragraph 5 remains unchanged.

> If the object being deleted has incomplete class type at the point of deletion and the complete class has a non-trivial destructor or a deallocation function, the behavior is undefined.

Remove paragraph 9 as follows.

> When the keyword `delete` in a *delete-expression* is preceded by the unary `::` operator, the global deallocation function is used to deallocate the storage.

Add a new paragraph in place of paragraph 9 as follows.

> If a *delete-expression* begins with a unary `::` operator, the deallocation function's name is looked up in global scope. Otherwise, the lookup considers class-specific deallocations (12.5 [class.free]). If no class-specific deallocation is found, the deallocation function's name is looked

up in global scope. If the lookup selects a placement deallocation function, the program is ill-formed.

Add a new paragraph as follows.

> If deallocation function lookup finds both a usual deallocation function with one parameter and a usual deallocation function with two parameters, and then if the object being deleted has incomplete class type, the selected deallocation function shall be the one with two parameters. Otherwise, the selected deallocation function shall be the function with one parameter.

Add a new paragraph as follows. This paragraph is identical to the existing 12.5/5.

> When a *delete-expression* is executed, the selected deallocation function shall be called with the address of the block of storage to be reclaimed as its first argument and (if the two-parameter style is used) the size of the block as its second argument. [*Footnote:* If the static type of the object to be deleted is different from the dynamic type and the destructor is not virtual the size might be incorrect, but that case is already undefined, as stated above. —*end footnote*]

12.5 Free store [class.free]

Edit paragraph 4 as follows.

> Class-specific deallocation function lookup is a part of general deallocation function lookup (5.3.5 [expr.delete]) and occurs as follows. ~~If a *delete-expression* begins with a unary :: operator, the deallocation function's name is looked up in global scope. Otherwise, if~~ If the *delete-expression* is used to deallocate a class object whose static type has a virtual destructor, the deallocation function is the one selected at the point of definition of the dynamic type's virtual destructor (12.4). [*Footnote:* A similar provision is not needed for the array version of `operator delete` because 5.3.5 requires that in this situation, the static type of the object to be deleted be the same as its dynamic type. —*end footnote*] Otherwise, if the *delete-expression* is used to deallocate an object of class `T` or array thereof, the static and dynamic types of the object shall be identical and the deallocation function's name is looked up in the scope of `T`. If this lookup fails to find the name, ~~the name is looked up in the global scope.~~ the class-specific deallocation function lookup has failed and general deallocation function lookup (5.3.5 [expr.delete]) continues. If the result of the lookup is ambiguous or inaccessible, or if the lookup selects a placement deallocation function, the program is ill-formed.

Remove paragraph 5 as follows. This paragraph moves to 5.3.5/9++.

~~When a *delete-expression* is executed, the selected deallocation function shall be called with the address of the block of storage to be reclaimed as its first argument and (if the two-parameter style is used) the size of the block as its second argument. [*Footnote:* If the static type of the object to be deleted is different from the dynamic type and the destructor is not virtual the size might be incorrect, but that case is already undefined; see 5.3.5. — *end footnote*]~~

17.6.3.6 Replacement functions [replacement.functions]

Edit paragraph 2 as follows.

A C++ program may provide the definition for any of eight dynamic memory allocation function signatures declared in header `<new>` (3.7.4, ~~Clause 18~~ 18.4 [support.dynamic]):

- `operator new(std::size_t)`
- `operator new(std::size_t, const std::nothrow_t&)`
- `operator new[](std::size_t)`
- `operator new[](std::size_t, const std::nothrow_t&)`
- `operator delete(void*)`
- `operator delete(void*, const std::nothrow_t&)`
- `operator delete[](void*)`
- `operator delete[](void*, const std::nothrow_t&)`

Furthermore, a C++ program may provide the definition for any of the four dynamic memory deallocation function signatures that implementations may choose to declare in header `<new>`:

- `operator delete(void*, std::size_t)`
- `operator delete(void*, std::size_t, const std::nothrow_t&)`
- `operator delete[](void*, std::size_t)`
- `operator delete[](void*, std::size_t, const std::nothrow_t&)`

18.6 Dynamic memory management [support.dynamic]

At the end of the synopsis add the following.

The implementation may, but need not, provide the following additional group of functions.

```
operator delete(void* ptr, std::size_t size)
throw();
operator delete(void* ptr, std::size_t size,
const std::nothrow_t&) throw();
```

```
operator delete[](void* ptr, std::size_t size)
throw();
operator delete[](void* ptr, std::size_t size,
const std::nothrow_t&) throw();
```

18.6.1.1 Single-object forms [new.delete.single]

At the end of the section, add a new paragraph as follows.

```
operator delete(void* ptr, std::size_t size) throw();
operator delete(void* ptr, std::size_t size, const
std::nothrow_t&) throw();
```

Add a new paragraph as follows.

```
These functions behave as their corresponding version
without the std::size_t size parameter, with the additional
constraints:

    Requires: size shall equal that used to
    allocate ptr.

    Required behavior: Calls to the sized and
    unsized versions shall be interchangable.
```

18.6.1.2 Array forms [new.delete.array]

At the end of the section, add a new paragraph as follows.

```
operator delete[](void* ptr, std::size_t size) throw();
operator delete[](void* ptr, std::size_t size, const
std::nothrow_t&) throw();
```

Add a new paragraph as follows.

```
These functions behave as their corresponding version
without the std::size_t size parameter, with the additional
constraints:

    Requires: size shall equal that used to
    allocate ptr.

    Required behavior: Calls to the sized and
    unsized versions shall be interchangable.
```

US 28
Clause 4.4

Comment 4.4/3: A const member function pointer could safely be applied to a non-const object without violating const correctness.

Technical details:
While a cv-qualified member function can be called on a less-cv-qualified object-expression (see [class.this] 9.3.2/4), a cv-qualified member function cannot be assigned to less-cv-qualified member function pointer even though it is no more dangerous to const correctness than the rule in 9.3.2/4. This code sample illustrates:

```
struct X {
void func1() const;
void func2();
};
void f() {
void (X::*fptr1)() = &X::func1;          // ill-formed, should be well-formed
void (X::*fptr2)() = &X::func2;          // well-formed
void (X::*fptr3)() const = &X::func1; // well-formed
void (X::*fptr4)() const = &X::func2; // ill-formed, must remain so
}
```

Proposed resolution:
Informally, we want to add an implicit conversion for pointer-to-member-function types from "void (T::*)() const" to "void (T::*)()". This is slightly tricky because the cv-qualifier-seq on a function declaration isn't actually a cv-qualified function type (see 8.3.5).

One possible wording change: strike 9.3.2/4, and add to 4.4/3 "A prvalue of type 'pointer to cv-qualified member function' can be converted to a prvalue of type 'pointer to less-cv-qualified member function'."

US 29

Jason Merrill
2010-03-09
Revision 3

CORE ISSUE 974. Default arguments for lambdas

Section: 5.1.2 [expr.prim.lambda] **Status:** open **Submitter:** Jason Merrill **Date:** 4 September, 2009
Priority: 2

(From 15012 and 15148.)

There does not appear to be any technical difficulty that would require the restriction in 5.1.2 [expr.prim.lambda] paragraph 5 against default arguments in *lambda-expression*s.

Suggested Resolution (March, 2010)

Strike from 5.1.2/5:

~~Default arguments (8.3.6) shall not be specified in the parameter-declaration-clause of a lambda-declarator~~

US 30

975. Restrictions on return type deduction for lambdas

Section: 5.1.2 [expr.prim.lambda] **Status:** open **Submitter:** Jason Merrill **Date:** 4 September, 2009 **Priority:** 2

(From messages 15012, 15148, 15152, and 15170.)

There does not appear to be any technical difficulty that would require the current restriction that the return type of a lambda can be deduced only if the body of the lambda consists of a single return statement. In particular, multiple return statements could be permitted if they all return the same type.

Drafting note

It is unfortunate that there is no way of writing directly the type deduced from an expression by auto or lambda return type deduction; decltype has significantly different results.

Suggested Resolution (March, 2010)

Change 5.1.2/4 from:
..... If a *lambda-expression* does not include a *trailing-return-type*, it is as if the *trailing-return-type* denotes the following type:

- ~~if~~ the *compound-statement* ~~if of the form~~

  ~~{ return attribute-specifier$_{opt}$ expression ; }~~

  the type of the returned expression after lvalue-to-rvalue conversion (4.1), array-to-pointer conversion (4.2), and function-to-pointer conversion (4.3);

- otherwise, ~~void~~.

[*Example*:

```
   auto x1 = [](int i){ return i; }; // OK: return type is int
   auto x2 = [](){ return { 1, 2 }; }; // error: the return type is void
(a
                                       // braced-init-list is not an
expression)
```

-- *end example*]
to:
..... If a *lambda-expression* does not include a *trailing-return-type*, it is as if the *trailing-return-type* denotes the following type:

- If there are no return statements in the *compound-statement*, or all return statements return void, `void;`
- otherwise, if all return statements are of the form `return` *expression* `;` and for all return statements the type of the returned expression after lvalue-to-rvalue conversion (4.1), array-to-pointer conversion (4.2), and function-to-pointer conversion (4.3) is equivalent, that type;
- otherwise, the program is ill-formed.

[*Example*:

```
   auto x1 = [](int i){ return i; }; // OK: return type is int
   auto x2 = []{ return { 1, 2 }; }; // error (a braced-init-list is
not an expression)
   template <class T> void f () {
     [](T t, bool b){
       if (b)
         return t.fn();
       else
         return t.fn();
     };                                  // OK: return type is type of
t.fn()
     [](T t, bool b){
       if (b)
         return t.fn1();
       else
         return t.fn2();
     };                                  // error: the type of t.fn1() is
not equivalent to the type of t.fn2() (14.6.6.1)
   }
```

-- *end example*]

FCD 14882
ADDITIONAL DETAILS TO US COMMENTS

US 34

Comment 5.3.4, 5.3.5: Allocation functions are missing happens-before requirements and guarantees.

Technical details:
When the same unit of storage is allocated and deallocated repeatedly, operations on it can't be allowed to race between the allocator and the user program. But I don't see any mention of happens-before in the descriptions of allocation and deallocation functions.

Proposed resolution (not wording yet):

- The call to an allocation function returning a pointer P must happen-before the matching deallocation call with P as a parameter. Otherwise the behavior is undefined. I don't know whether receiving P with memory_order_consume fits this requirement. memory_order_relaxed does not.

- If some memory is passed to a deallocation function, the implementation must ensure that the deallocation call happens-before any allocation call that returns the same memory address.

US 49

Jason Merrill
2010-05-25
Revision 1

Direct Binding correction

The Problem

The FCD rules do not specify direct binding for this example:

```
int i;
int main()
{
  int&& ir = static_cast<int&&>(i);
  ir = 42;
  return (i != 42);
}
```

We ought to do direct binding for reference-compatible xvalues. It also seems that the array rvalue case ought to be folded in with the class rvalue case, since the only way to get an array rvalue is to refer to an array member of a class rvalue. I considered introducing the term "crvalue" for that subset of prvalues, but couldn't find any other places to use it. I also considered extending the direct binding treatment to scalar members of a class rvalue, but that seems like a larger change than necessary.

Proposed Wording

Change 8.5.3 [dcl.init.ref] paragraph 5 as follows:

...

- Otherwise, if the initializer expression is an xvalue, class prvalue or array prvalue, and "cv1 T1" is reference-compatible with "cv2 T2", the reference is bound to the object represented by the rvalue (see 3.10) or to a subobject within that object.
- Otherwise, if T2 is a class type and
  - ~~the initializer expression is an rvalue and "cv1 T1" is reference-compatible with "cv2 T2", or~~
  - T1 is not reference-related to T2 and the initializer expression can be implicitly converted to an rvalue of type "cv3 T3" where "cv1 T1" is reference-compatible with "cv3 T3" (this conversion is selected by enumerating the applicable conversion functions (13.3.1.6) and choosing the best one through overload resolution (13.3)),

then the reference is bound to ~~the initializer expression rvalue in the first case and to~~ the object that is the result of the conversion ~~in the second case~~ (or~~, in either case,~~ to the appropriate base class subobject of the object).

[*Example*:

```
   struct A { };
   struct B : A { } b;
   extern B f();
   const A& rca = f();                  // bound to the A subobject of
the B rvalue.
   A&& rcb = f();                       // same as above
   struct X {
     operator B();
   } x;
   const A& r = x;                      // bound to the A subobject of
the result of the conversion
```

-- *end example*]

- ~~If the initializer expression is an rvalue, with T2 an array type, and "cv1 T1" is reference-compatible with "cv2 T2," the reference is bound to the object represented by the rvalue (see 3.10).~~ ...

[*Example*:

```
   struct A { };
   struct B : A { } b;
   extern B f();
   const A& rca = f();                  // bound to the A subobject of
the B rvalue.
   A&& rcb = f();                       // same as above
   struct X {
     operator B();
   } x;
   const A& r = x;                      // bound to the A subobject of
the result of the conversion
```

-- *end example*]

- ~~If the initializer expression is an rvalue, with T2 an array type, and â€œcv1 T1â€□ is reference-compatible with â€œcv2 T2,â€□ the reference is bound to the object represented by the rvalue (see 3.10).~~ ....

US 66
Author: Jason Merrill
2010-05-10
Revision 2

List-construction fallback

The Problem

In core-16131, Daniel Krügler offered the following example:
a) If I have no initializer-list c'tors, I can write

```
struct A{} a;

struct S {
  S(A, A); // #1
  S(int, double, bool); // #2
};

S s1{a, a}; // OK, calls #1
S s2{12, 3.1, false}; // OK, calls #2
```

b) If I now add an initializer-list c'tor like this:

```
#include <initializer_list>

struct A{} a;
struct B{};

struct S {
  S(A, A); // #1
  S(int, double, bool); // #2
  S(std::initializer_list<B>); // #3
};

S s1{a, a}; // Error, does not match #3
S s2{12, 3.1, false};  // Error, does not match #3
```

This struck him as unfortunate; if there are list constructors but none are viable, we should subsequently try to match a non-list constructor. This is especially problematic if S has an additional constructor:

```
#include <initializer_list>

struct A{} a;
struct B{};
struct C {
  C(A, A);
};

struct S {
  S(A, A); // #1
  S(std::initializer_list<B>); // #3
  S(C); // #4
```

```
};

S s1{a, a}; // Matches #4 instead of #1
```

This was not a problem in N2385, but my changes to formalize the rules ended up oversimplifying the handling of classes with list constructors in a way that created this problem.

The right answer seems to be to do overload resolution for list-initialization of a class in two phases: first look for a suitable list constructor, and if none is found then look for a suitable non-list constructor.

Proposed Wording

Change 8.5.4 [dcl.init.list] as follows:

....

- Otherwise, if T is a class type, constructors are considered. ~~If T has an initializer-list constructor, the argument list consists of the initializer list as a single argument; otherwise, the argument list consists of the elements of the initializer list.~~ The applicable constructors are enumerated ~~(13.3.1.7)~~ and the best one is chosen through overload resolution (13.3.1.7, 13.3). If a narrowing conversion (see below) is required to convert any of the arguments, the program is ill-formed.

....

Change 13.3.1.7 [over.match.list] as follows:

When objects of non-aggregate class type `T` are list-initialized (8.5.4), overload resolution selects the constructor in two phases~~as follows, where T is the cv-unqualified class type of the object being initialized~~:

- ~~If `T` has an initializer-list constructor (8.5.4),~~ Initially, the candidate functions are the initializer-list constructors (8.5.4) of the class `T` and the argument list consists of the initializer list as a single ~~argument;~~ argument.
- ~~otherwise,~~ If no viable initializer-list constructor is found, overload resolution is performed again, where the candidate functions are all the constructors of the class `T` and the argument list consists of the elements of the initializer list.
- ~~For direct-list-initialization, the candidate functions are all the constructors of the class T.~~
- ~~For~~ In copy-list-initialization, ~~the candidate functions are all the constructors of T. However,~~ if an explicit constructor is chosen, the initialization is ill-formed. [Note: This differs from other situations (13.3.1.3, 13.3.1.4), where only converting constructors are considered for copy-initialization. This restriction only applies if this initialization is part of the final result of overload resolution -- end note]

US 76

Comment 14.8.2/9: extern template prevents inlining functions not marked inline.

Technical details:
Existing compilers often inline functions that weren't explicitly declared inline. To inline a function, they need to instantiate its template. Yet the FCD [temp.explicit]p9 says, "Except for inline functions and class template specializations, explicit instantiation declarations have the
effect of suppressing the implicit instantiation of the entity to which they refer." This means that adding an explicit instantiation declaration can affect performance, even though the user only intended to suppress out-of-line copies of functions. N1987 doesn't seem to expect any changes in inlining behavior.

Proposed resolution:

- Remove [temp.explicit] / p9.
- Replace the note in [temp.explicit] / p10 with: "The intent of this rule is to allow compilers to avoid emitting out-of-line copies of template functions, while still allowing them to inline those functions. The compiler may assume that another translation unit will supply the body."

FCD 14882
ADDITIONAL DETAILS TO US COMMENTS

US 88

Clause 20.2.5 [allocator.requirements]

Allocator Interface Not Backward Compatible

Date: 2010-05-21
Author: P.J. Plauger

Clause 20.2.5, Allocator requirements, adds functionality to the C++03 (old) allocator interface. It also endeavors to maintain compatibility with old allocators, so that old allocators can be used with new users of allocators, such as containers, function objects, regular expressions, shared pointers, string streams, etc. To supply the missing functionality for old allocators, C++0X also adds **allocator traits**, in the form of a new template class allocator_traits. This template class uses various template metaprogramming techniques to determine whether it can obtain a given type or function from an allocator, or whether it should supply a default instead. If code that uses allocators is changed to work through the intermediary of allocator_traits, it can access (most of) the functions and types defined in the old allocator interface.

Unfortunately, the new allocator requirements go too far. They permit a new allocator to omit many of the types and functions required by old allocators. allocator_traits will fill in the blanks. The new allocator requirements also eliminate some functions and types required of old allocators, presumably on the assumption that programmers don't need them any more and won't miss them. In so doing, however, the new allocator requirements fail to maintain the other half of the allocator interface -- they permit, even encourage, programmers to write **new** allocators that don't work with **old** users of allocators. Indeed, every existing container, function object, etc. listed above has to be reviewed and probably revised to work in the new world of abbreviated allocators and allocator traits.

Things that use allocators in the Standard C++ library are not a problem. Implementers are obliged to update the standard containers, etc. to match the new requirements. User-defined things that use the library-supplied default allocator are also not a problem. The default allocator retains even those features of old allocators that are no longer required by the new allocator requirements. But in general:

- Old allocators work with old allocator users, as we would expect.
- New allocators work with new allocator users, as we would expect.
- Old allocators work with new allocator users, if the new allocator users use allocator_traits, as needed.
- Old allocator users work with the new default allocator.
- But otherwise, old allocator users **do not** work with new allocators.

In designing the new allocator/user interface, committee discussions focused overwhelmingly on the problem of retaining backward compatibility between old allocators and the new containers supplied by the Standard C++ library. There was strong sentiment that old code should not have to be modified to work with the new library, even though the new library demanded more of allocators than before. The committee could have simply required that old allocators have to work only with the standard containers ("just make it work"), but it chose to go farther. By publishing the allocator_traits interface, C++0X provides tools for user-supplied containers, etc. to work with either old or new allocators.

The same consideration was not extended nearly as far to old containers, etc. If they want to survive in the C++0X world of new allocators, they will typically have to be rewritten. The rewrite is largely mechanical, but a careful review has to be done. To the best of my knowledge, the committee has not commented on this issue. There may be no sentiment for preserving old allocator users, but that weakens the case for indulging in heroics to preserve old allocators.

FCD 14882
ADDITIONAL DETAILS TO US COMMENTS

Proposal 0

If the committee is willing to mandate at least **some** rewrite of old code, it could simply suggest that all allocators be changed, if necessary, to (behave as if they) have the default allocator as a public base class, as in:

```
template<class T, class... Rest>
  class Myalloc
    : public std:allocator<T>
  {.....}
```

This solves the problem of defaulting the new functions and types that are missing in the old allocators. It ensures that old containers, etc. work with new allocators. And it provides for future additions to the allocator interface Deriving from std::allocator<T> has long been a common idiom. The author of Myalloc merely has to supply those things that change from the default allocator, and those things that cannot be inherited from a base class (such as constructors and rebind). Allocators of this form do not have to change to conform to this new requirement.

The fix is to remove the defaults from Table 41 in 20.2.5, and to restore the requirements dropped from the old allocator specification. allocator_traits can also be removed, or it can be retained for those who believe that abbreviating allocators will prove to be a good idea.

Proposal 1

If the preceding Proposal is too sweeping, at the very least the C++ Standard should fix the backward compatibility problem outlined above:

- Restore to the allocator requirements the functions and types that were dropped (primarily reference, const_reference, and address).
- Drop the default behavior for all allocator requirements inherited from C++2003.

Detailed changes for Proposal 1 are summarized below.

A salutary effect of adopting this proposal is to reduce the amount of new code that has to be added to <memory>. In its current form, allocator_traits adds roughly 200 lines of code to this header, which must be included in every program that uses a library container, **including string**. The code removed merely gives the programmer the ability to abbreviate allocators, solving a non-problem that can usually be addressed, if needed, by the one-line addition shown above.

Note that a library implementation can still choose to provide all the defaults currently listed in allocator_traits, as a conforming extension. If it turns out that there really is a market for abbreviated allocators, the evidence will be apparent after a couple of years of field experience. The C++ Standard can then be modified with a Technical Corrigendum or Amendment. But if we allow abbreviations from the start, it's much harder to later disallow abbreviations in user code.

Proposal 2

We can simplify allocator_traits even farther by noting that only two kinds of allocators really need to be supported, old and new. As before, there's no compelling need for various levels of newness. allocator_traits need only have code that recognizes the presence of **any** new feature to conclude that **all** new features are present.

Programmers sophisticated enough to write allocators know to add a base class, as above, when they are content with the defaults for some features. Adding a hundred-odd lines of code to practically **every** compile penalizes simple programs that, say, just use string, all to save sophisticated programs from writing perhaps one line of text. This is a clear violation of the rule that you don't pay for what you don't use.

The fix is simple:

In 20.2.5/2 Allocator requirements, after the sentence "Table 42 specifies a default for a given expression." ADD "If an allocator supplies any requirement that has a default, it shall supply all requirements that have a default."

Proposal 1 Details

In 20.2.5 Allocator requirements:

ADD to Table 41 the line:

mt a value of type T&

REMOVE from Table 42 defaults for pointer, const_pointer, size_type, difference_type, rebind, allocate with hint, and max_size.

Add to Table 42 the lines:

```
X::reference       (no Return)  Assertion/note: T&
X::const_reference (no Return)  Assertion/note: const T&
p.address(mt)      X::pointer   &mt
p.address(t)  X::const_pointer  &t
a.construct(p, t)  (not used)   Effect: Constructs an object of type T at p.
a.destroy(p)       (not used)   Effect: Destroys the object at p
```

REMOVE from 20.2.5/2 the sentence "A user specialization of allocator_traits may provide different defaults and may provide defaults for different requirements than the primary template."

In 20.2.5/3, DELETE: "If Allocator is a class template instantiation of the form SomeAllocator<T, Args>, where Args is zero or more type arguments, and Allocator does not supply a rebind member template, the standard allocator_traits template uses SomeAllocator<U, Args> in place of Allocator::rebind<U>::other by default. For allocator types that are not template instantiations of the above form, no default is provided."

In 20.3.5/5, CHANGE the example from:

```
template <class Tp>
  struct SimpleAllocator {
    typedef Tp value_type;
    SimpleAllocator(ctor args);
    template <class T> SimpleAllocator(const SimpleAllocator<T>& other);
    Tp *allocate(std::size_t n);
    void deallocate(Tp *p, std::size_t n);
  };
```

TO:

```
template <class Tp>
  struct SimpleAllocator
    : public std::allocator<Tp> {
  template <class T>
    SimpleAllocator(const SimpleAllocator<T>& other);
  template <class T>
    SimpleAllocator& (const SimpleAllocator<T>& other)
    { return *this; }
  template <class T>
    struct rebind {
```

```
    typedef SimpleAllocator<T>& other; }
 };
```

The allocator can add just those functions it wishes to override, such as allocate and deallocate, to obtain behavior that differs from std::allocator<Tp>.

In 20.9.4 Allocator traits, CHANGE:

```
 typedef see below pointer;
 typedef see below const_pointer;
 typedef see below void_pointer;
 typedef see below const_void_pointer;

 typedef see below difference_type;
 typedef see below size_type;
 .....
 template <class T> using rebind_alloc = see below;
```

TO:

```
 typedef Alloc::pointer pointer;
 typedef Alloc::const_pointer const_pointer;
 typedef pointer_traits<pointer>::rebind<void> void_pointer;
 typedef pointer_traits<pointer>::rebind<const void> const_void_pointer;

 typedef Alloc::difference_type difference_type;
 typedef Alloc::size_type size_type;
 .....
 template <class T> using rebind_alloc = Alloc::rebind<T>::other;
```

In 20.9.4.1 Allocator traits member types, REMOVE:

typedef see below pointer;

Type: Alloc::pointer if such a type exists; otherwise, value_type*.

typedef see below const_pointer;

Type: Alloc::const_pointer if such a type exists; otherwise, pointer_traits<pointer>::rebind<const value_type>.

typedef see below void_pointer;

Type: Alloc::void_pointer if such a type exists; otherwise, pointer_traits<pointer>::rebind<void>.

typedef see below const_void_pointer;

Type: Alloc::const_void_pointer if such a type exists; otherwise, pointer_traits<pointer>::rebind<const void>.

typedef see below difference_type;

Type: Alloc::difference_type if such a type exists; otherwise, ptrdiff_t.

typedef see below size_type;

Type: Alloc::size_type if such a type exists; otherwise, size_t.

.....

template <class T> using rebind_alloc = see below;

Template alias: Alloc::rebind<T>::other if such a type exists; otherwise, Alloc<T, Args> if Alloc is a class template instantiation of the form Alloc<U, Args>, where Args is zero or more type arguments; otherwise, the instantiation of rebind_alloc is ill-formed.

In 20.9.4.2 Allocator traits static member functions, CHANGE:

static pointer allocate(Alloc& a, size_type n, const_void_pointer hint);

Returns: a.allocate(n, hint) if that expression is well-formed; otherwise, a.allocate(n).

.....

template <class T, class... Args>
  static void construct(Alloc& a, T* p, Args&&... args);

Effects: calls a.construct(p, std::forward<Args>(args)...) if that call is well-formed; otherwise, invokes ::new (static_cast<void*>(p)) T(std::forward<Args>(args)...).

template <class T>
 static void destroy(Alloc& a, T* p);

Effects: calls a.destroy(p) if that call is well-formed; otherwise, invokes p->~T().

static size_type max_size(Alloc& a);

Returns: a.max_size() if that expression is well-formed; otherwise, numeric_limits<size_type>::max().

TO:

static pointer allocate(Alloc& a, size_type n, const_void_pointer hint);

Returns: a.allocate(n, hint).

.....

template <class T, class... Args>
  static void construct(Alloc& a, T* p, Args&&... args);

Effects: calls a.construct(p, std::forward<Args>(args)...) if that call is well-formed; otherwise, invokes ::new (static_cast<void*>(p)) T(std::forward<Args>(args)...).

[Note: Alloc always defines a signature equivalent to construct(pointer, const value_type&). -- end note]

template <class T>
 static void destroy(Alloc& a, T* p);

Effects: calls a.destroy(p) if that call is well-formed; otherwise, invokes p->~T().

[Note: Alloc always defines a signature equivalent to destroy(pointer). -- end note]

static size_type max_size(Alloc& a);

Returns: a.max_size().

FCD 14882
ADDITIONAL DETAILS TO US COMMENTS

US 96

20.3.5.2 [pairs.pair] Move/forward confusion in pair and tuple construction and assignment

Section: 20.3.5.2 [pairs.pair], 20.4.2.1 [tuple.cnstr], and 20.4.2.2/6 [tuple.assign]

Submitter: Pablo Halpern

Discussion

Note: This issue overlaps with LWG 1326.

There are actually two issues that are intertwined, so it makes sense to resolve them together.

The first issue is that the terms MoveConstructible, MoveAssignable, CopyAssignable are being misused in [pair] and [tuple] to describe heterogeneous construction or assignment, even thought the terms are defined only in terms of a single type.

The second issue is that std::move is being used where std::forward is required. In particular, std::move will erroneously convert an lvalue-reference to an rvalue-reference whereas std::forward will not. Also, the terms “move-constructs” and “move-assigns” are being used in contexts where std::forward should be used, implying the wrong semantic.

Proposed Resolution

Change [pairs.pair]/6 as follows:

template<class U, class V> pair(pair<U, V>&& p);

6 *Effects:* The constructor initializes first with std::~~move~~forward<U>(p.first) and second with std::~~move~~forward<V>(p.second).

Change [pairs.pair] paragraphs 12-15 as follows:

pair& operator=(pair&& p);

12 *Effects:* Assigns to first with std::~~move~~forward<T1>(p.first) and to second with std::~~move~~forward<T2>(p.second).

13 *Returns:* *this.

template<class U, class V> pair& operator=(pair<U, V>&& p);

14 *Effects:* Assigns to first with std::~~move~~forward<U>(p.first) and to second with std::~~move~~forward<V>(p.second).

15 *Returns:* *this.

Change [tuple.cnstr] paragraph 6 as follows:

template <class... UTypes>
explicit tuple(UTypes&&... u);

6 *Requires:* Each type in Types shall ~~satisfy the requirements of MoveConstructible (Table 34)~~be constructible from the corresponding type in UTypes&& . sizeof...(Types) == sizeof...(UTypes).

Change [tuple.cnstr] paragraphs 11-20 as follows:

tuple(tuple&& u);

10 *Requires:* Each type in Types shall ~~shall~~ satisfy the requirements of MoveConstructible (Table 34).
11 *Effects:* ~~Move~~ constructs each element of *this with the corresponding element of std::forward<Types>(u).

template <class... UTypes> tuple(const tuple<UTypes...>& u);
12 *Requires:* Each type in Types shall be constructible from the corresponding type in UTypes&&. sizeof...(Types) == sizeof...(UTypes).
13 *Effects:* Constructs each element of *this with the corresponding element of std::forward<UTypes>(u).
14 [*Note:* enable_if can be used to make the converting constructor and assignment operator exist only in the cases where the source and target have the same number of elements. —*end note*]

template <class... UTypes> tuple(tuple<UTypes...>&& u);
15 *Requires:* Each type in Types shall ~~shall the requirements of MoveConstructible (Table 34)~~be constructible from the corresponding type in UTypes&&. sizeof...(Types) == sizeof...(UTypes).
16 *Effects:* ~~Move~~ constructs each element of *this with the corresponding element of std::forward<UTypes>(u).
[*Note:* enable_if can be used to make the converting constructor and assignment operator exist only in the cases where the source and target have the same number of elements. —*end note*]

template <class U1, class U2> tuple(const pair<U1, U2>& u);
17 *Requires:* The first type in Types shall be constructible from U1 and the second type in Types shall be constructible from U2. sizeof...(Types) == 2.
18 *Effects:* Constructs the first element with u.first and the second element with u.second.

template <class U1, class U2> tuple(pair<U1, U2>&& u);
19 *Requires:* The first type in Types shall ~~shall the requirements of MoveConstructible (Table 34)~~be constructible from U1&& and the second type in Types shall be ~~move~~ constructible from U2&&. sizeof...(Types) == 2.
20 *Effects:* Constructs the first element with std::~~move~~forward<U1>(u.first) and the second element with std::~~move~~forward<U2>(u.second).

Change [tuple.assign] paragraph 6 as follows:

6 *Effects:* ~~Move~~ assigns each element of std::forward<Types>(u) to the corresponding element of *this.

Change [tuple.assign] paragraphs 11 and 12 as follows:

11 *Requires:* Each type in Types shall ~~satisfy the requirements of MoveConstructible (Table 34)~~be assignable from the corresponding type in UTypes&&. sizeof...(Types) == sizeof...(UTypes).
12 *Effects:* ~~Move~~ assigns each element of std::forward<UTypes>(u) to the corresponding element of *this.

Change [tuple.assign] paragraph 14 as follows:

14 *Requires:* The first type in Types shall ~~shall satisfy the requirements of MoveAssignable (Table 36)~~be assignable from U1 and the second type in Types shall ~~shall satisfy the requirements of MoveAssignable (Table 36)~~be assignable from U2. sizeof...(Types) == 2.

Change [tuple.assign] paragraphs 18 and 19 as follows:

18 *Requires:* The first type in Types shall be Assignable from U1&& and the second type in Types shall be Assignable from U2&&. sizeof...(Types) == 2.

19 *Effects:* Assigns std::~~move~~forward<U1>(u.first) to the first element of *this and std::~~move~~forward<U2>(u.second) to the second element of *this.

FCD 14882
ADDITIONAL DETAILS TO US COMMENTS

US 106

[pointer.traits] pointer_traits lacks size_type

Section: 20.9.3 [pointer.traits]

Submitter: Pablo Halpern

Discussion:

The pointer_traits template has a difference_type member, but not a corresponding size_type member. This asymmetry causes issues for allocators. Currently, the default type for allocator_traits<A>::difference_type is ptrdiff_t and the default type for allocator_traits<A>::size_type is size_t. However, it would be more useful and natural for allocator_traits<A>::difference_type to default to pointer_traits<A::pointer>::difference_type and allocator_traits<A>::size_type to default to pointer_traits<A::pointer>::size_type. The former is currently possible but the latter is not because of the absence of size_type in pointer_traits.

Proposed Resolution:

Add a new type to [pointer.traits]/1:

typedef *see below* difference_type;
typedef *see below* size_type;

And a paragraph after [pointer.traits]/2:

typedef *see below* difference_type;

2 Type: Ptr::difference_type if such a type exists; otherwise, std::ptrdiff_t.

typedef *see below* size_type;

Type: Ptr::size_type if such a type exists; otherwise, std::size_t.

In section [allocator.requirements], Table 42, replace the defaults for difference_type and size_type:

X::size_type	unsigned integral type	a type that can represent the size of the largest object in the allocation model.	~~size_t~~pointer_traits<pointer>::size_type
X::difference_type	signed integral type	a type that can represent the difference between any two pointers in the allocation model.	~~ptrdiff_t~~pointer_traits<pointer>::difference_type

Change the definitions of difference_type and size_type in [allocator.traits] paragraphs 5 and 6:

typedef *see below* difference_type;

5 Type: Alloc::difference_type if such a type exists; otherwise, ~~ptrdiff_t~~pointer_traits<pointer>::difference_type.

typedef *see below* size_type;

6 Type: Alloc::size_type if such a type exists; otherwise, ~~size_t~~pointer_traits<pointer>::size_type.

FCD 14882
ADDITIONAL DETAILS TO US COMMENTS

US 107

Clauses 20.1, 20.9.1, 20.9.6

Scoped Allocator Adaptor Inappropriate for <memory>

Date: 2010-05-21
Author: P.J. Plauger

Template class scoped_allocator_adaptor (20.9.6) requires a nontrivial piece of code to implement. The reference implementation (referenced in N2982) runs to nearly 800 lines. C++0X has already added significantly to the header <memory> with features such as shared_ptr and unique_ptr, but scoped allocators still represent a significant addition to a widely used header.

Dinkumware has already received push back from customers whe dislike the longer compile times brought on by C++0X features that they don't even use. This may appear to be carping, given the speed of today's computers, but it is not. If an overnight build doesn't complete overnight, scheduling problems arise. We have repartitioned our headers more than once, in response to this feedback. But that technique goes only so far.

WG21 can probably justify placing shared_ptr and unique_ptr in <memory>, since the expectation is that these template classes are also likely to be widely used, and by a broad range of programmers. But it is hard to make the same case for scoped_allocator_adaptor. The latter requires a sophisticated knowledge of allocators and how they might interact with containers and container elements.

Template class scoped_allocator_adaptor has the virtue of being self-contained. It has no required uses within the Standard C++ library, and there are no references to it from other parts of the C++ Standard. Hence, it is both easy and beneficial to move this template class to a new header.

The fix is to add the new header <scoped_allocator>:

IN 20.1 General, Table 30, ADD the line:

20.xx	Scoped allocators	<scoped_allocator>

IN 20 General Utilities Library, INSERT/APPEND a new section 20.xx, Class scoped_allocator_adaptor

FROM 20.9.1 Header <memory> synopsis, MOVE to 20.xx the new 20.xx.1 Header <scoped_allocator> synopsis:

```
// 20.9.6, scoped allocator adaptor
template <class OuterAlloc, class... InnerAlloc>
    class scoped_allocator_adaptor;
template <class OuterA1, class OuterA2, class... InnerAllocs>
bool operator==(const scoped_allocator_adaptor<OuterA1, InnerAllocs...>& a,)
const scoped_allocator_adaptor<OuterA2, InnerAllocs...>& b);
template <class OuterA1, class OuterA2, class... InnerAllocs>
bool operator!=(const scoped_allocator_adaptor<OuterA1, InnerAllocs...>& a,)
const scoped_allocator_adaptor<OuterA2, InnerAllocs...>& b);
```

MOVE 20.9.6 Scoped allocator adaptor to the new 20.xx.2.

FCD 14882
ADDITIONAL DETAILS TO US COMMENTS

US 115

23.2.1 [container.requirements.general] Terminology for constructing container elements

Section: 23.2.1 [container.requirements.general]

Submitter: Pablo Halpern

Discussion

The resolution to issue 704 added a number of requirements to the container operations in order to properly constrain the elements of the containers. Unfortunately, the current wording effectively *redefines* the terms CopyConstructible and MoveConstructible and the phrase "constructible with *args*" so that they have different meanings in the containers section than in the rest of the standard. This use of terminology is not only confusing and vague, it is also not applied correctly through the section. There are some cases, in fact, when the term CopyConstructible is used in its original meaning, but the reader would have no way to know that.

The best solution is to choose an entirely new and more precise set of terms and apply them consistently and correctly in the containers section. I nominate the terms *X can copy-insert T*, *X can move-insert T*, and *X can construct-insert T with args* as replacements for the above terms, where X is the container type, A is X's allocator type and T is its element type.

Proposed Resolution:

Replace [container.requirements.general]/15 as follows:

15 ~~The descriptions of the requirements of the type T in this section use the terms CopyConstructible, MoveConstructible, constructible from *i, and constructible from args. These terms are equivalent to the following expression using the appropriate arguments:~~

Given a container type X having an allocator_type A and a value_type T and given an lvalue m of type A, a pointer p of type T*, a value v of type T, or a value rv of type rvalue-of-T, the following terms are defined. (If X is not allocator-aware, the terms below are defined as if A were std::allocator<T>.):

X can copy-insert T means that the following expression is well-formed:

allocator_traits<A>::contruct(m, p, v);

X can move-insert T means that the following expression is well-formed:

allocator_traits<A>::contruct(m, p, rv);

A can construct-insert T from args for zero or more arguments, *args*, means that the following expression is well-formed:

allocator_traits<A>::contruct(m, p, *args*);

[*Note:* The default of contruct in std::allocator will call

::new((void*) p) T(*args*)

but specialized allocators may choose a different definition. – *end note*]

A review of the rest of section 23 shows that one can substitute the above terms in all cases where the phrase “T is CopyConstructible”, etc.. A complete resolution will need to spell out each individual case, as sometimes the wording varies as in “T shall be CopyConstructible” or “value_type is constructible from,” etc.

FCD 14882
ADDITIONAL DETAILS TO US COMMENTS

US 117

Clause 23.3.3

Problems with forward_list::erase_after

Date: 2010-05-21
Author: P.J. Plauger

The iterator returns from the two overloads of forward_list::erase_after were removed as part of the adoption of N2988. The rationale given was, in its entirety:

"I believe that erase_after should return void. Returning its argument does nothing (potentially for a price), and confuses the programmer. I think it's better that the programmer notice that forward_list is different from other containers (and better not to pass a value through a function unnecessarily)."

The first and last sentences are, by their own admission ("I believe", "I think"), unfounded conjectures.

The second sentence is misleading, since only the second overload -- erase_after(iterator first, iterator last) -- happens to return one of its arguments. The first overload -- erase_after(iterator before) -- does not. Both return an iterator designating the first element after the element(s) removed, or end() if the last element is removed.

The parenthetic remark in the second sentence happens not to be true, since the iterator to be returned arises naturally in the process of erasing the elements.

The **real price** that this change has exacted is to destroy backward compatibility with the Committee Draft, implementations of which are now widely available to programmers. It takes more than conjectures to justify making such a breaking change.

It is also worth noting that the Dinkumware implementation of forward_list calls erase_after thirteen times for internal purposes. In five of these cases, the code makes use of the return value. This is admittedly a small sample of all the code now using forward_list, but enough to refute the notion that the return value of erase_after "does nothing" or merely "confuses the programmer." Whether it is "better" to enforce a different style of programming is a matter of taste.

The fix is to restore the wording in the Committee Draft:

IN 23.3.3/3, CHANGE:

```
void erase_after(const_iterator position);
void erase_after(const_iterator position, iterator last);
```

TO:

```
iterator erase_after(const_iterator position);
iterator erase_after(const_iterator position, iterator last);
```

IN 23.3.3.5/18-21, CHANGE:

```
void erase_after(const_iterator position);
```

Requires: The iterator following position is dereferenceable.
Effects: Erases the element pointed to by the iterator following position.

void erase_after(const_iterator position, iterator last);

Requires: All iterators in the range (position,last) are dereferenceable.
Effects: Erases the elements in the range (position,last).

TO:

iterator erase_after(const_iterator position);

Requires: The iterator following position is dereferenceable.
Effects: Erases the element pointed to by the iterator following position.
Returns: An iterator pointing to the element following the one that was erased, or end() if no such element exists.

iterator erase_after(const_iterator position, iterator last);

Requires: All iterators in the range (position,last) are dereferenceable.
Effects: Erases the elements in the range (position,last).
Returns: last

US 118

Comment 23.5: Some unordered associative container operations have undesirable complexities when the container is implemented using singly linked lists.

Technical details
A common implementation technique for hash tables is as a vector of buckets, where each bucket is a separate singly linked list. The TR1 unordered associative container specification was intended to allow that implementation, among others.

This representation is desirable in some ways, but it does have a known problem: if the current element happens to be the last in a bucket, incrementing it to get to the next element requires a linear scan over the bucket array to find the next non-empty bucket. If the load factor (the ratio of elements to buckets) is very low, this means that iterator increment is expensive.

In itself this is unfortunate but tolerable ---iteration through hash tables is less common than element insertion or lookup, and there is long experience with STL hash table implementations (such as the SGI implementation) that have this property. FCD unordered associative containers, however, make the problem worse. The erase() member function is defined to return an iterator pointing to the element after the one that's erased, which means that a common hash table operation can result in invoking the potentially expensive iterator increment

Proposed resolution
None. In principle there are several possibilities, including:

- Change the signature of erase so that it returns void. This would return us to the situation with SGI STL hash tables, where iterator increment is potentially expensive but programmers can realistically avoid it in most cases.
- Impose a minimum load factor, resizing the tables when necessary. This might involve changing the iterator invalidation rules.
- Require implementations to use a doubly linked list implementation instead. This would be undesirable because it would greatly increase the space requirements, for essentially technical reasons.

FCD 14882
ADDITIONAL DETAILS TO US COMMENTS

US 120

Comment 25.2.12/1: is_permutation is underspecified for anything but the simple case where both ranges have the same value type and the comparison function is an equivalence relation.

Technical details

There are no requirements on is_permutation's template parameters beyond those implied by their names, and by the fact that the returns clause is written in terms of std::equal. It thus appears at first sight that is_permutation has the same constraints as equal, and that no further constraints are needed. This apparent simplicity is misleading.

The returns clause says that is_permutation(f1,l1,f2) is true iff there is some permutation of the elements in [f1, l1), which we can call [f1',l1'), such that equal(f1',l1',f2) is true. However, it doesn't say how to find that permutation or prove that no such permutation exists. That's obviously the crux of the algorithm, and we need to think about the constraints that are needed for it to work. So let's think about a definition that describes how we actually compute whether is_permutation(f1,l1,f2) is true or false.

The n2986 reference implementation describes the actual algorithm that's intended: loop through [f1, l1). For each value v, count how many times it appears in [f1, l1) and in [f2, l2). If the counts differ, then [f1, l1) can't be a permutation of [f2, l2). If the counts are the same for every v and the ranges have the same length, then it is. There are several variations on this technique, but all the variations I know of involve choosing some value v in one of the input ranges and looking at other instances of the same value in the same range.

This means that, unlike the case of std::equal, it isn't enough to require the existence of comparison between a value of type iterator_traits<ForwardIterator1>::value_type and one of type iterator_traits<ForwardIterator2>::value_type. At a minimum, we also also need to perform comparisons between two values of type iterator_traits<ForwardIterator1>::value_type and/or two values of type iterator_traits<ForwardIterator2>::value_type.

Even that isn't enough, though --it's enough to get is_permutation to compile, but not enough for it to give sane results. It's easy to come up with examples of two types V1 and V2 such that there are sensible comparisons between V1 and itself, V2 and itself, and V1/V2, but where is_permutation would give nonsensical results. Consider, for example:

```
struct A { int val; char tag; };
struct B { int val; string tag; };

bool operator==(A x, A y) { return x.val == y.val && x.tag == y.tag; }
bool operator==(B x, B y) { return x.val == y.val && x.tag == y.tag; }
bool operator==(A x, B y) { return x.val == y.val; }
```

Now suppose we call is_permutation on two sequences c1 and c2, where c1 is [(1, 'a'), (2, 'b'), (1, 'c')] and c2 is [(2,"x"), (1, "y"), (1, "z")]. Do we want the result to be true, or false? Taken literally the returns clause in the FCD says that in this example the return value should be true, but the algorithm described above, and the n2986 reference implementation, will return false. (And I don't know of any practical algorithm that would do otherwise.) It will examine (1, a), find that it appears once in c1 and twice in c2, and return false.

The fundamental problem with this example is is that we have three elements, x, y, and z, such that x == y, y == z, and x != z. That should look familiar. If operator== is allowed to violate the requirements of an equivalence relation, one can construct equally pathological examples using only a single value type. This algorithm has to compare multiple elements against each other, so we won't get sensible results unless operator== obeys sensible axioms.

Proposed resolution

Informally: restrict is_permutation to the case where it is well specified. More formally:
Add a new paragraph to [alg.is_permutation], before the existing paragraph 1:

> Requires: ForwardIterator1 and ForwardIterator2 shall have the same value type. The comparison function shall be an equivalence relation.

[There are other possible fixes. We could allow heterogeneous comparisons but constrain them more. Or, more radically, we could get rid of is_permutation entirely and describe in words what equality comparison should mean for hash multisets/multimaps, or even get rid of or change the definition of hash multiset/multumap equality. We recommend the resolution above because the former option would be complicated and easy to get wrong, while the latter two are larger changes than necessary.]

FCD 14882
ADDITIONAL DETAILS TO US COMMENTS

US 136

Clause 26.8

Problems with Floating-point Test Functions

Date: 2010-05-21
Author: P.J. Plauger

In 26.8 C Library, Table 116 lists as "Templates" a number of functions defined as macros in the C Standard:

Templates:
fpclassify isgreaterequal islessequal isnan isunordered
isfinite isinf islessgreater isnormal signbit
isgreater isless

This is inaccurate. Each macro is better described in C++ as three overloads, with operand(s) of type float, double, and long double. Moreover, as with the math functions described in this same section, each of the functions has "sufficient additional overloads" to simulate the effect of the C type-generic functions.

The fix is to change 26.8 as follows:

In Table 116, CHANGE "Functions:" TO "Math Functions:" and CHANGE "Templates:" TO "Classification/comparison Functions:".

BEFORE paragraph 10 ("Moreover, there shall be...") INSERT:

The classification/comparison functions behave the same as the C macros with corresponding names defined in 7.12.3, Classification macros, and 7.12.14, Comparison macros in the C Standard. Each function is overloaded for the three floating-point types, as follows:

```
namespace std {
int fpclassify(float x);
bool isfinite(float x);
bool isinf(float x);
bool isnan(float x);
bool isnormal(float x);
bool signbit(float x);

bool isgreater(float x, float y);
bool isgreaterequal(float x, float y);
bool isless(float x, float y);
bool islessequal(float x, float y);
bool islessgreater(float x, float y);
bool isunordered(float x, float y);

int fpclassify(double x);
bool isfinite(double x);
bool isinf(double x);
bool isnan(double x);
bool isnormal(double x);
bool signbit(double x);
```

```
bool isgreater(double x, double y);
bool isgreaterequal(double x, double y);
bool isless(double x, double y);
bool islessequal(double x, double y);
bool islessgreater(double x, double y);
bool isunordered(double x, double y);

int fpclassify(long double x);
bool isfinite(long double x);
bool isinf(long double x);
bool isnan(long double x);
bool isnormal(long double x);
bool signbit(long double x);

bool isgreater(long double x, long double y);
bool isgreaterequal(long double x, long double y);
bool isless(long double x, long double y);
bool islessequal(long double x, long double y);
bool islessgreater(long double x, long double y);
bool isunordered(long double x, long double y);
} // namespace std
```

DELETE paragraph 11 ("The templates defined...") and paragraph 12 ("The templates behave...").

FCD 14882
ADDITIONAL DETAILS TO US COMMENTS

US 137

Clause: 27.7

Problems with Iostreams Member Functions

Date: 2010-05-21
Author: P.J. Plauger

There are several problems with member functions in basic_istream and basic_ostream:

-- putback is obliged to fail at end of file. Moreover, the member function doesn't clear eofbit, as it should. The current wording says:

Effects: Behaves as an unformatted input function (as described in 27.7.1.3, paragraph 1). After constructing a sentry object, if !good() calls setstate(failbit) which may throw an exception, and return.

Both problems can be solved by first clearing eofbit:

CHANGE 27.7.1.3/34 (putback) first sentence from:

Behaves as an unformatted input function (as described in 27.7.1.3, paragraph 1).

TO:

Behaves as an unformatted input function (as described in 27.7.1.3, paragraph 1), except that the function first clears eofbit.

-- A similar problem exists with unget, with a similar fix:

CHANGE 27.7.1.3/36 (unget) first sentence from:

Behaves as an unformatted input function (as described in 27.7.1.3, paragraph 1).

TO:

Behaves as an unformatted input function (as described in 27.7.1.3, paragraph 1), except that the function first clears eofbit.

-- The first overload of seekg ends with the sentence:

In case of failure, the function calls setstate(failbit) (which may throw ios_basefailure).

This sentence is missing in the description of the second overload, for no good reason that I can detect. The fix is to add the sentence:

CHANGE 27.7.1.3/43 (seekg) by ADDING at the end of the paragraph:

In case of failure, the function calls setstate(failbit) (which may throw ios_basefailure).

-- The basic_ostream seek functions, seekp and tellp say nothing about constructing a sentry object. But these functions need such protection just as much as seekg and tellg, which do. The fix is to add a sentence:

FCD 14882
ADDITIONAL DETAILS TO US COMMENTS

AFTER 27.7.2.5 basic_ostream seek members, ADD:

Each seek member function begins execution by constructing an object of class sentry. It returns by destroying the sentry object.

FCD 14882
ADDITIONAL DETAILS TO US COMMENTS

US 138

27.7 [iostream.format] Move and swap for I/O streams

Section: 27.7 [iostream.format]

Submitter: Pablo Halpern

Discussion:

For basic_istream, basic_ostream, and basic_iostream, the move constructor does not do a move construction, tthe move-assignment operation does not do move-assignment and swap does not perform a swap. Moreover, these functions are protected, precluding their use in reasonable code.

The resolution to issue 900 (and related issue 911) assumes that these functions would never be called from client code. However, these classes are not abstract. They can be instanced and there are use-cases for such instances. For example, one can create a filebuf outside of an fstream, then associate it with an ostream:

```
filebuf fb("name");

ostream fstr(&fb);
```

The above ostream is a full-fledged object that should be movable, and copyable. In that case, the move and copy operations should move and copy the *whole* object, including the rdbuf() member.

However, moving the rdbuf() member poses a problem for derived classes like fstream, that contain embedded streambuf objects and which want to ensure that the base class portion of the copy container a pointer to a copy of the streambuf, not a pointer to the original streambuf. To simplify programming of these operations, we can have new constructors and a new function that perform the same actions as copy-construction, move-construction, and move-assignment, but do not move or copy the rdbuf() pointer.

Proposed Resolution:

The move() functions declared in basic_ios provides a good start. move() functions could be added to basic_istream, basic_ostream, and basic_iostream in order to support the same idiom. Alternatively, each class can have special constructors and assignment operators specific to the three operations. For example, basic_istream could have:

```
basic_istream(basic_istream&& rhs, basic_streambuf<…> *sb);

partial_move(basic_istream&& rhs, basic_streambuf<…> *sb);

partial_swap(basic_istream& other, basic_streambuf<…> *this_sb,

        basic_streambuf<…> *other_sb);
```

These functions are useful in general, and should be public. Additionally, the regular move constructor, move-assignment operator, and swap functions should be made public and changed so that they move and swap the streambuf pointer as well as the rest of the stream state.

FCD 14882
ADDITIONAL DETAILS TO US COMMENTS

US 181

Clauses 30.2.4, 30, 20.10

C++ Timeout Specification

ISO/IEC JTC1 SC22 WG21 - 2010-05-19 - National Body Comment by Google

Lawrence Crowl, crowl@google.com, Lawrence@Crowl.org

Problem

The meaning of clocks and timeouts is poorly defined within the Final Committee Draft when those clocks may be adjusted. Clocks can be adjusted by hours and many network protocols expect responses within seconds. A task regularly scheduled for midnight GMT should execute at midnight even though the clock has been adjusted to eliminate accumulated error. Failure of the standard to be precise about this distinction makes programs effectively unportable.

The root of the problem is that the current definition leaves open disparate implementations that even as implementations increase their quality, separate implementations will not converge on the same behavior.

There will necessarily be some delay in the interrupt response, function return, and scheduling of a thread waking from a timeout. Implementations can reasonably strive to approach zero delay for these activities. So, we call this delay the "quality of implementation".

Separately, there will be some delay due to contention for processor and memory resources. This delay is more under the control of the application programmer and systems administrator than it is under the implementation. So, we call this delay the "quality of management". The tradeoff between resources and responsiveness is necessarily application-dependent.

We can express the actual time of a timeout as the sum of the intended time, the quality of implementation and the quality of management. The remaining problem is to map the given timeout specifications to a common intended time.

If there are no adjustments to the clock time, the intended time may be straightforwardly determined from the manner of specification. In this case, we assume that any difference in durations between reported times and SI units is small, and thus constitutes a measure of the quality of implementation.

The problem arises with the specification of the intended timeout when the clock is adjusted in the middle of the timeout. There are two plausible strategies, the timeout is sensitive to adjustments in the clock time, or it is not. The *timeout*_until functions have straightforward definitions when they are sensitive to adjustments. The *timeout*_for functions have straightforward definitions when they are insensitive to adjustments.

Solution

Define *timeout*_until to respect reported clock time points and define *timeout*_for to respect real time durations.

A consequence of these definitions is that *timeout*_until and *timeout*_for are not functionally redundant. That is, *timeout*_until(Clock::now()+3_seconds) is not equivalent to *timeout*_for(3_seconds) when the clock is adjusted in the interval.

The implementation of the *timeout* definition necessarily depends on a *steady clock*, one that cannot be adjusted. A monotonic clock is not sufficient. While one could be implicit in the standard, below we make one explicit.

Given a steady clock, the monotonic clock seems to be of marginal utility. Still, we have preserved it in this proposal.

[Note that considered the practics of setting clocks forward in unit tests, and believe that continuing to do so even in *timeout*_for operations would be reasonable because the tests operate in a virtual world, not in the real world. The definition of time in that world need not be the same as the time in the real world.]

Wording

The proposed wording changes are relative to the Final Committee Draft, N3092.

20.10 Time utilities [time]

Edit within the header synopsis as follows.

```
// Clocks
class system_clock;
class monotonic_clock;
class steady_clock;
class high_resolution_clock;
```

20.10.1 Clock requirements [time.clock.req]

Edit within table 54 as follows.

....	....	....
`C1::is_monotonic`	`const bool`	`true` if `t1 <= t2` is always `true`, otherwise `false`. [*Note:* A clock that can be adjusted backwards is not monotonic. —*end note*]
`C1::is_steady`	`const bool`	`true` if the clock value cannot be adjusted and that the duration between ticks is close to the tick period, otherwise `false`.
`C1::now()`	`C1::time_point`	Returns a `time_point` object representing the current point in time.

Add a new paragraph 3.

> [*Note:* The relative difference in durations between those reported by the given clock and the SI definition is a measure of the quality of implementation. —*end note*]

20.10.5.1 Class `system_clock` [time.clock.system]

Edit paragraph 1 as follows.

> Objects of class `system_clock` represent wall clock time from the system-wide realtime clock.

```
class system_clock {
```

```
 public:
   typedef see below
rep;
   typedef ratio<unspecified, unspecified>
period;
   typedef chrono::duration<rep, period>
duration;
   typedef chrono::time_point<system_clock>
time_point;
   static const bool is_monotonic =
unspecified;
   static const bool is_steady =
unspecified;

   static time_point now();

   // Map to C API
   static time_t        to_time_t(const
time_point& t);
   static time_point    from_time_t(time_t t);
};
```

20.10.5.2 Class `monotonic_clock` [time.clock.monotonic]

Edit paragraph 2 as follows.

~~The class `monotonic_clock` is conditionally supported.~~

```
 class monotonic_clock {
 public:
   typedef unspecified
rep;
   typedef ratio<unspecified, unspecified>
period;
   typedef chrono::duration<rep, period>
duration;
   typedef chrono::time_point<unspecified>
time_point;
   typedef chrono::time_point<monotonic_clock>
time_point;
   static const bool is_monotonic =
true;
   static const bool is_steady =
unspecified;

   static time_point now();
};
```

20.10.5.3 Class `high_resolution_clock` [time.clock.hires]

Edit paragraph 1 as follows.

Objects of class `high_resolution_clock` represent clocks with the shortest tick period. `high_resolution_clock` may be a synonym for `system_clock` or `monotonic_clock`.

```
 class high_resolution_clock {
 public:
   typedef unspecified
rep;
   typedef ratio<unspecified, unspecified>
period;
   typedef chrono::duration<rep, period>
duration;
   typedef chrono::time_point<unspecified>
time_point;
   typedef
chrono::time_point<high_resolution_clock>
time_point;
   static const bool is_monotonic =
unspecified;
   static const bool is_steady =
unspecified;

   static time_point now();
};
```

20.10.5.4 Class `steady_clock` [time.clock.steady]

Add a new section 20.10.5.4 Class `steady_clock` [time.clock.steady] as follows.

Add a new paragraph.

Objects of class `steady_clock` represent clocks for which values of `time_point` advance at a steady rate relative to real time. That is, the clock may not be adjusted.

```
 class steady_clock {
 public:
   typedef unspecified
rep;
   typedef ratio<unspecified, unspecified>
period;
   typedef chrono::duration<rep, period>
duration;
   typedef chrono::time_point<steady_clock>
time_point;
   static const bool is_monotonic =
true;
   static const bool is_steady =
true;

   static time_point now();
```

```
};
```

30.2.4 Timing specifications [thread.req.timing]

Add a new paragraph after paragraph 1 as follows.

> Implementations necessarily have some delay in returning from a timeout. Any overhead in interrupt response, function return, and scheduling induces a "quality of implementation" delay, expressed as duration Di. Ideally, this delay would be zero. Further, any contention for processor and memory resources induces a "quality of management" delay, expressed as duration Dm. The delay durations may vary from timeout to timeout, but in all cases shorter is better.

Edit paragraph 2 as follows.

> The member functions whose names end in `_for` take an argument that specifies a ~~relative time~~ duration. These functions produce *relative timeouts*. Implementations ~~should~~ shall use a ~~monotonic~~ steady clock to measure time for these functions. ~~[*Note:* Implementations are not required to use a monotonic clock because such a clock may not be available. — *end note*]~~ [*Footnote:* All implementations for which standard time units are meaningful must necessarily have a steady clock within their hardware implementation. —*end footnote*] Given a duration argument Dt, the real-time duration of the timeout is Dt+Di+Dm.

Add a new paragraph after paragraph 2 as follows.

> The member functions whose names end in `_until` take an argument that specifies a time point. These functions produce *absolute timeouts*. Implementations shall use the clock specified in the time point to measure time for these functions. Given a clock time point argument Ct, the clock time point of the return from timeout is Ct+Di+Dm unless, during the timeout, the clock has been adjusted to a time Ca>Ct, in which case the return time is Ca+Di+Dm.

30.3.2 Namespace this_thread [thread.thread.this]

Edit paragraph 6, regarding `sleep_until`, as follows.

> *Effects:* Blocks the calling thread ~~at least until the time~~ for the absolute timeout (30.2.4 [thread.req.timing]) specified by `abs_time`.

Edit paragraph 9, regarding `sleep_for`, as follows.

Effects: Blocks the calling thread ~~for at least the time~~ for the relative timeout (30.2.4 [thread.req.timing]) specified by `rel_time`.

30.4.2 TimedMutex requirements [thread.timedmutex.requirements]

Edit paragraph 4, regarding `try_lock_for`, as follows.

Effects: The function attempts to obtain ownership of the mutex within the ~~time~~ relative timeout (30.2.4 [thread.req.timing]) specified by `rel_time`. If the time specified by `rel_time` is less than or equal to 0, the function attempts to obtain ownership without blocking (as if by calling `try_lock()`). The function shall return within the ~~time~~ timeout specified by `rel_time` only if it has obtained ownership of the mutex object. [*Note:* As with `try_lock()`, there is no guarantee that ownership will be obtained if the lock is available, but implementations are expected to make a strong effort to do so. —*end note*]

Edit paragraph 10, regarding `try_lock_until`, as follows.

Effects: The function attempts to obtain ownership of the mutex by the ~~time~~ absolute timeout (30.2.4 [thread.req.timing]) specified by `abs_time`. If `abs_time` has already passed, the function attempts to obtain ownership without blocking (as if by calling `try_lock()`). The function shall return before the ~~time~~ timeout specified by `abs_time` only if it has obtained ownership of the mutex object. [*Note:* As with `try_lock()`, there is no guarantee that ownership will be obtained if the lock is available, but implementations are expected to make a strong effort to do so. —*end note*]

30.5.1 Class condition_variable [thread.condition.condvar]

Edit within paragraph 19, regarding `wait_until`, as follows.

Effects:

- ....
- The function will unblock when signaled by a call to `notify_one()` ~~or~~ , a call to `notify_all()`, ~~if `abs_time <= Clock::now()`~~ expiration of the absolute timeout (30.2.4 [thread.req.timing]) specified by `abs_time`, or spuriously.
- ....

Edit paragraph 21, regarding `wait_until`, as follows.

Returns: `cv_status::timeout` if ~~the function unblocked because `abs_time` was reached~~ the absolute timeout (30.2.4 [thread.req.timing]) specified by `abs_time` expired, otherwise `cv_status::no_timeout`.

Edit within paragraph 25, regarding `wait_for`, as follows.

Effects:

- ....
- The function will unblock when signaled by a call to `notify_one()` ~~or~~ , a call to `notify_all()`, ~~by the elapsed time `rel_time` passing~~ expiration of the relative timeout (30.2.4 [thread.req.timing]) specified by `rel_time`, or spuriously.
- ....

Edit paragraph 26, regarding `wait_for`, as follows.

Returns: `cv_status::timeout` if ~~the function unblocked because `rel_time` elapsed~~ the relative timeout (30.2.4 [thread.req.timing]) specified by `rel_time` expired, otherwise `cv_status::no_timeout`.

Edit within paragraph 34, regarding the predicate `wait_for`, as follows.

Effects:

- ....
- The function will unblock when signaled by a call to `notify_one()` ~~or~~ , a call to `notify_all()`, ~~by the elapsed time `rel_time` passing~~ expiration of the relative timeout (30.2.4 [thread.req.timing]) specified by `rel_time`, or spuriously.
- ....
- The loop terminates when `pred()` returns true or when the ~~time duration~~ relative timeout (30.2.4 [thread.req.timing]) specified by `rel_time` has elapsed.
- ....

30.5.2 Class condition_variable_any [thread.condition.condvarany]

Edit within paragraph 15, regarding `wait_until`, as follows.

Effects:

- ....
- The function will unblock when signaled by a call to `notify_one()` ~~or~~ , a call to `notify_all()`, ~~if `abs_time` <=~~

~~`Clock::now()`~~ expiration of the absolute timeout (30.2.4 [thread.req.timing]) specified by `abs_time`, or spuriously.

- ....

Edit paragraph 17, regarding `wait_until`, as follows.

Returns: `cv_status::timeout` if ~~the function unblocked because `abs_time` was reached~~ the absolute timeout (30.2.4 [thread.req.timing]) specified by `abs_time` expired, otherwise `cv_status::no_timeout`.

Edit within paragraph 20, regarding `wait_for`, as follows.

Effects:

- ....
- The function will unblock when signaled by a call to `notify_one()` ~~or~~ , a call to `notify_all()`, ~~by the elapsed time `rel_time` passing~~ expiration of the relative timeout (30.2.4 [thread.req.timing]) specified by `rel_time`, or spuriously.
- ....

Edit paragraph 21, regarding `wait_for`, as follows.

Returns: `cv_status::timeout` if ~~the function unblocked because `rel_time` elapsed~~ the relative timeout (30.2.4 [thread.req.timing]) specified by `rel_time` expired, otherwise `cv_status::no_timeout`.

Edit within paragraph 28, regarding the predicate `wait_for`, as follows.

Effects:

- ....
- The function will unblock when signaled by a call to `notify_one()` ~~or~~ , a call to `notify_all()`, ~~by the elapsed time `rel_time` passing~~ expiration of the relative timeout (30.2.4 [thread.req.timing]) specified by `rel_time`, or spuriously.
- ....
- The loop terminates when `pred()` returns true or when the ~~time duration~~ relative timeout (30.2.4 [thread.req.timing]) specified by `rel_time` has elapsed.
- ....

30.6.6 Class template future [futures.unique_future]

Edit within paragraph 22, regarding `wait_for`, as follows.

Effects: blocks until the associated asynchronous state is ready or until the relative timeout (30.2.4 [thread.req.timing]) specified by `rel_time` has ~~elapsed~~ expired.

Edit within paragraph 23, regarding `wait_for`, as follows.

Returns:

- ....
- `future_status::timeout` if the function is returning because the ~~time period~~ relative timeout (30.2.4 [thread.req.timing]) specified by `rel_time` has ~~elapsed~~ expired.
- ....

Edit within paragraph 25, regarding `wait_until`, as follows.

Effects: blocks until the associated asynchronous state is ready or until ~~the current time exceeds~~ the absolute timeout (30.2.4 [thread.req.timing]) specified by `abs_time` has expired.

Edit within paragraph 26, regarding `wait_until`, as follows.

Returns:

- ....
- `future_status::timeout` if the function is returning because the ~~the time point~~ absolute timeout (30.2.4 [thread.req.timing]) specified by `rel_time` has ~~been reached~~ expired.
- ....

30.6.7 Class template shared_future [futures.shared_future]

Edit within paragraph 27, regarding `wait_for`, as follows.

Effects: blocks until the associated asynchronous state is ready or until the relative timeout (30.2.4 [thread.req.timing]) specified by `rel_time` has ~~elapsed~~ expired.

Edit within paragraph 28, regarding `wait_for`, as follows.

Returns:

- ....
- `future_status::timeout` if the function is returning because the ~~time period~~ relative timeout (30.2.4 [thread.req.timing]) specified by `rel_time` has ~~elapsed~~ expired.

- ....

Edit within paragraph 30, regarding `wait_until`, as follows.

> *Effects:* blocks until the associated asynchronous state is ready or until ~~the current time exceeds~~ the absolute timeout (30.2.4 [thread.req.timing]) specified by `abs_time` has expired.

Edit within paragraph 31, regarding `wait_until`, as follows.

> *Returns:*
>
> - ....
> - `future_status::timeout` if the function is returning because the ~~the time point~~ absolute timeout (30.2.4 [thread.req.timing]) specified by `rel_time` has ~~been reached~~ expired.
> - ....

30.6.8 Class template atomic_future [futures.atomic_future]

Edit within paragraph 23, regarding `wait_for`, as follows.

> *Effects:* blocks until the associated asynchronous state is ready or until the relative timeout (30.2.4 [thread.req.timing]) specified by `rel_time` has ~~elapsed~~ expired.

Edit within paragraph 24, regarding `wait_for`, as follows.

> *Returns:*
>
> - ....
> - `future_status::timeout` if the function is returning because the ~~time period~~ relative timeout (30.2.4 [thread.req.timing]) specified by `rel_time` has ~~elapsed~~ expired.
> - ....

Edit within paragraph 27, regarding `wait_until`, as follows.

> *Effects:* blocks until the associated asynchronous state is ready or until ~~the current time exceeds~~ the absolute timeout (30.2.4 [thread.req.timing]) specified by `abs_time` has expired.

Edit within paragraph 28, regarding `wait_until`, as follows.

> *Returns:*

- ....
- `future_status::timeout` if the function is returning because the ~~the time point~~ absolute timeout (30.2.4 [thread.req.timing]) specified by `rel_time` has ~~been reached~~ expired.
- ....

FCD 14882
ADDITIONAL DETAILS TO US COMMENTS

US 186

30.4.1 [thread.mutex.requirements/A] try_lock does not provide a guarantee of forward progress

Section: 30.4.1 [thread.mutex.requirements]

Submitter: Pablo Halpern (with help from Charles Leiserson, Edya Laden Mozes, and Tao B. Schardl)

Discussion

Summary

The current definition of try_lock() allows the attempt to fail spuriously, even if another thread has not acquired the lock. This definition does not provide a guarantee of forward progress without elaborate countermeasures by the programmer. We consider this to be a serious deficiency.

Background

The reasons to allow spurious failures are described in *Foundations of the C++ Concurrency Memory Model*, by Boehm and Adve, PLDI 2008: http://portal.acm.org/citation.cfm?id=1375591 (requires ACM membership). The authors describe a situation like the following:

THREAD 1	**THREAD 2**
x = 42;	while (m.try_lock()) { }
m.lock();	assert(x == 42);

In order to maintain the "sequential consistency for data-race-free programs" model and still have the assert succeed, it is necessary to have an extra fence on entering lock() for some CPU architectures. Otherwise, the try_lock() may fail before the lock() has completed and, therefore, before the memory fence within lock() has executed. To avoid this cost, it was decided that this case should be allowed to fail. One way to allow such a failure is to allow spurious false return from try_lock().

It is interesting to note that spurious failures are unlikely to occur in practice, on any architecture. POSIX mutexes, for example, are not permitted to fail spuriously. The effect of *allowing* spurious failures, however, is that the above code is permitted to fail, allowing some implementations to avoid an extra fence on lock().

Problems with the current definition

The FCD's definition of try_lock does not guarantee that forward progress will be made by any thread using try_lock. Consider code that contains no lock() calls, but any number of try_lock() calls that could be executed in parallel. Whenever threads contend on a try_lock(), they may both fail and spin indefinitely, causing livelock. In fact, a degenerate but conforming implementation of try_lock() would simply return false, thus guaranteeing livelock! (Such an implementation of try_lock() would be very efficient because it contains no fences! :-)).

Livelock is a surprising result, and is one of those traps that is easy enough to fall into without the standard lending assistance. To defend against livelock, the programmer would be forced to implement elaborate logic, such as a probabilistic back-off protocol, which most programmers are not capable of writing correctly. In contrast, if try_lock() could not fail spuriously, then one thread would be guaranteed to make progress on

the critical section. Code that did not have to defend against this kind of livelock would be simpler and less error-prone.

Allowing spurious failures is a premature optimization. People will have to write defensive code forever in the future, no matter how fast locks and memory fences become in the future (and the performance of these synchronization primitives has been improving). There is no doubt that lock acquisition can hurt performance, and that an extra fence could make a lock 20-30% slower, but if the nature of a program is such that the cost of acquiring an uncontended lock becomes a major bottleneck, then the solution is probably not a cheaper standard lock, but a cheaper algorithm – or else it is time to write your own lock with weaker guarantees. Our feeling is that the performance penalty for a well-behaved, predictable lock will make little difference to most multithreaded programs.

Proposed Resolution

We propose to ban spurious failures of try_lock and to require sequential consistency in the system-provided mutex types.

To permit the user to supply higher-performance mutexes in the requirements clause, we can permit one or both of the following in the Mutex requirements:

1. Mutexes with spurious failures on try_lock
2. Mutexes whose locking functions permit reordering of ordinary memory operations into the critical region (i.e., after the point of synchronization with a failed try block).

Each approach has as its downsides. Spurious failures are reasonably easy to reason about, but very difficult to correct-for in cases where they might cause live-lock. Allowing memory operations to be reordered across a lock is not unheard-of, but it does violate the sequential consistency rules that the standard tries hard to preserve. Either approach will give implementers of custom mutexes the flexibility they need to reduce the cost of a lock, so it is not necessary to provide both options (and therefore suffer both sets of downsides).

As per the PLDI paper, allowing spurious failures is not an implementation strategy but rather a simple way to explain the behavior of certain locks without breaking sequential consistency. Effectively, an equivalence relationship is posited such that the simpler language of spurious failures can be used instead of the more complex happens-before language. However, this equivalence is not proven and, in fact, it appears not to hold in all cases. Specifically, spurious failures could result in livelock in situations where reordering memory operations would not. According to the PLDI paper, “this is the first solution that eliminates the need to provide a fence before a lock, while still maintaining a simple definition of a race.” Indeed, a try_lock that may fail spuriously is a new invention, not vetted in common use. It is not clear how many algorithms that depend on reliable try_lock behavior would be impacted, but it is clear that the number is not zero. To our knowledge, there is no widely-used threading library that allows spurious failures in try_lock as a matter of policy.

Perhaps the most important argument against allowing for spurious failures is that the work-around for code that requires a reliable try_lock is beyond the skill of most programmers, whereas the work-around for code that relies on sequential consistency simply involves adding a well-placed fence. If a lock() call allows memory operations to be reordered into the critical section, but disallows spurious try_lock() failures, then the code example above could be made to work reliably by adding a fence before the lock():

THREAD 1	**THREAD 2**
x = 42;	while (m.try_lock()) { }
atomic_thread_fence(memory_order_release);	assert(x == 42);
m.lock();	

For these reasons, we propose wording that would **ban** spurious failure of try_lock for both system-supplied and user-supplied mutex types. For improved performance, a user-supplied mutex would be permitted to use a subtler synchronization protocol to avoid extra fences. Such a protocol might violate the "sequential consistency for data-race-free programs" model and would, therefore, be harder to reason about than mutexes that don't take such liberties. But such reasoning doesn't rely on incomplete equivalences which may yield incorrect conclusions. Critically, unlike spurious failures, there would not be the same threat of livelock. To simplify matters for the reader, however, we propose a non-normative note suggesting that such weak ordering of memory operations before a lock may manifest as an apparent spurious failure. It should be noted that the example code in the PLDI paper uses locks in a strange and not recommended fashion. Confusion about why it may fail should not be a driving force for choosing a sub-optimal solution.

FCD 14882
ADDITIONAL DETAILS TO US COMMENTS

US 188

30.4.1 [thread.mutex.requirements/B] Mutex requirements should not be bound to threads

Section: 30.4.1 [thread.mutex.requirements]

Submitter: Pablo Halpern (with help from Charles Leiserson)

Discussion:

The set of requirements called “Mutex” are intended to describe mutual exclusion locks as a set of generic requirements that permit a reasonably wide range of different mutual exclusion mechanisms. However, these requirements unnecessarily narrow the range of locks that can be called Mutexes because they define mutex ownership in terms of threads. There are, however, other agents besides threads (e.g. tasks or processes) that could meaningfully hold a mutex, and the definition of Mutex should be broad enough to encompass those types of mutex.

Some types of mutex that would not be owned by a thread:

- Models involving dynamic scheduling of tasks may have a task migrate from thread to thread. A mutex may be acquired and released within a task, but may be rescheduled in between.
- There are models involving parallelism within locked regions. There could be a type of mutex that does not protect the threads within the parallel region against one another, but rather protects the entire parallel region against other concurrent actions.
- Parallel libraries like TBB (Threading Building Blocks) may need a type of mutex that can be inherited by a child task.
- Similarly, one could conceive of a mutex that would be inherited by a packaged task or async() call.
- There are parallelism models like Cilk for which thread ownership is not well defined.
- A course-grained mutex might be owned by a process, rather than with an individual thread.
- A library might allow the owner of a mutex to delegate the release of that mutex to another entity.

None of the above would meet the requirements of a Mutex by the current definition, yet programmers would probably be surprised if they were unable to use them with the standard lock_guard, unique_lock, and generic locking algorithms. Restricting a mutex to thread-based ownership is fine for the standard-supplied mutex types, but should not be required for all mutex types.

Proposed resolution:

Change [thread.mutex.requirements] as follows:

1 A *concurrent agent* is a an entity (such as a thread, process, or packaged task) that may perform work in parallel with other concurrent agents. The *calling agent* is determined by context, e.g., the calling thread, the calling process, or the packaged task that contains the call, etc.. A mutex object facilitates protection against data races and allows ~~thread~~ safe synchronization of data between ~~threads~~ concurrent agents. An agent ~~thread~~ owns a mutex from the time it successfully calls one of the lock functions until it calls unlock. Mutexes may be either recursive or non-recursive, and may grant simultaneous ownership to one or many ~~threads~~ concurrent agents. The mutex types supplied by the standard library provide exclusive ownership semantics for threads: only one thread may own the mutex at a time. Both recursive and non-recursive mutexes are supplied. [*Note:* Some mutexes are “agent-oblivious” in that they work for any concurrent-agent model because they do not determine or store the agent’s ID (e.g., an ordinary spin-lock). – *end note*]

2 This section describes requirements on template argument types used to instantiate templates defined in the C++ standard library. The template definitions in the C++ standard library refer to the named Mutex requirements whose details are set out below. In this description, m is an object of a Mutex type.

3 A Mutex type shall be DefaultConstructible and Destructible. If initialization of an object of a Mutex type fails, an exception of type std::system_error shall be thrown. A Mutex type shall not be copyable nor movable.

4 The error conditions for error codes, if any, reported by member functions of a Mutex type shall be:

- resource_unavailable_try_again — if any native handle type manipulated is not available.
- operation_not_permitted — if the ~~thread~~ calling agent does not have the privilege to perform the operation.
- device_or_resource_busy — if any native handle type manipulated is already locked.
- invalid_argument — if any native handle type manipulated as part of mutex construction is incorrect.

5 The implementation shall provide lock and unlock operations, as described below. The implementation shall serialize those operations. *[Note:* Construction and destruction of an object of a Mutex type need not be thread-safe; other synchronization should be used to ensure that Mutex objects are initialized and visible to other threads. —*end note]*

6 The expression m.lock() shall be well-formed and have the following semantics:

7 *Effects:* Blocks the calling ~~thread~~ agent until ownership of the mutex can be obtained for the calling ~~thread~~ agent.

8 *Postcondition:* The calling ~~thread~~ agent owns the mutex.

9 *Return type:* void

10 *Synchronization:* Prior unlock() operations on the same object shall synchronize with (1.10) this operation.

11 *Throws:* std::system_error when an exception is required (30.2.2).

12 *Error conditions:*

- operation_not_permitted — if the ~~thread~~ calling agent does not have the privilege to perform the operation.
- resource_deadlock_would_occur — if the implementation detects that a deadlock would occur.
- device_or_resource_busy — if the mutex is already locked and blocking is not possible.

13 The expression m.try_lock() shall be well-formed and have the following semantics:

14 *Effects:* Attempts to obtain ownership of the mutex for the calling ~~thread~~ agent without blocking. If ownership is not obtained, there is no effect and try_lock() immediately returns. An implementation may fail to obtain the lock even if it is not held by any other ~~thread~~ concurrent agent. *[Note:* This spurious failure is normally uncommon, but allows interesting implementations based on a simple compare_exchange_weak (29). —*end note*]

15 *Return type:* bool

16 *Returns:* true if ownership of the mutex was obtained for the calling ~~thread~~ agent, otherwise false.

17 *Synchronization:* If try_lock() returns true, prior unlock() operations on the same object synchronize with (1.10) this operation. [Note: Since lock() does not synchronize with a failed

subsequent try_lock(), the visibility rules are weak enough that little would be known about the state after a failure, even in the absence of spurious failures. —end note]

18 *Throws:* Nothing.

19 The expression m.unlock() shall be well-formed and have the following semantics:

20 *Requires:* The calling ~~thread~~ agent shall own the mutex.

21 *Effects:* Releases the calling ~~thread~~ agent's ownership of the mutex.

22 *Return type:* void

23 *Synchronization:* This operation synchronizes with (1.10) subsequent lock operations that obtain ownership on the same object.

24 *Throws:* Nothing.

FCD 14882
ADDITIONAL DETAILS TO US COMMENTS

US 191

Comment 30.5: Condition variable wait_for returning cv_status is insufficient.

Technical details:
The predicate wait is defined as a loop over the non-predicate wait. The predicate wait_until is defined as a loop over the non-predicate wait_until. The predicate wait_for is not so implemented. The problem is that the return value from the non-predicate wait_for operation provides insufficient information to synthesize the behavior of the predicate wait_for. This problem indicates that the wait_for facility is under-provisioned.

Proposed resolution:
Rather than return the enumeration cv_status, the non-predicate wait_for should return the duration remaining in the timeout. Then the predicate wait_for has a constructive definition testing the duration against zero and waiting as needed for the successively smaller return values.

For consistency, the wait_until operation could return the time point of the return for comparison to the specified abs_time.

FCD 14882
ADDITIONAL DETAILS TO US COMMENTS

US 193

Comment 30.5.1: Condition variables preclude a wakeup optimization.
Technical details:
When a thread calls condition_variable::notify_all, it causes all threads blocked in condition_variable::wait(Predicate) to wake up, reacquire the lock, evaluate their predicates, and then often go back to sleep. It's more efficient to evaluate the predicates in the notifying thread, and only wake up the threads for which the predicate is true. However, wait(Predicate) doesn't allow the predicate to run in another thread, and it doesn't allow missed wakeups when the predicate would return true when evaluated in the waiting thread but returns false when evaluated in the notifying thread. Further, the predicate must run under the wait's mutex, but notify_all may be called with or without that mutex held.

I believe condition_variable_any cannot implement this optimization because it doesn't require that all waiters hold the same mutex, so there's no way for notify_all to guarantee that it won't acquire a mutex recursively when trying to guard the predicate call.

Proposed resolution:

1. To condition_variable::notify_one and condition_variable::notify_all, add:

   Requires: if any threads are concurrently executing a call to wait*(lock, ...), lock.mutex() must be locked by the calling thread. Note: this allows them to run a wait's predicate.

2. To each of the condition_variable::wait variants that takes a predicate, add to the "Effects" a line saying that

   "pred may be called in any thread that calls any notify or wait member function of this condition_variable, either inside the call to notify() or at the start of the next mutex.unlock() call, before the mutex is unlocked. If it returns false at any of these points, it is unspecified whether the preceding call to notify*() causes this wait to unblock."

FCD 14882
ADDITIONAL DETAILS TO US COMMENTS

US 194
Clause 30.6

Managing C++ Associated Asynchronous State

ISO/IEC JTC1 SC22 WG21 - 2010-05-19 - National Body Comment by Google

Lawrence Crowl, crowl@google.com, Lawrence@Crowl.org

Problem

Within the Final Committee Draft, the specification for managing associated asynchronous state [futures.state] is confusing, sometimes omitted, and redundantly specified.

Solution

Define terms-of-art for *releasing*, *making ready*, and *abandoning* an associated asynchronous state. Use those terms where appropriate.

Wording

The wording is relative to the FCD.

30.6.4 Associated asynchronous state [futures.state]

Edit paragraph 5 as follows.

> ~~When the last reference to an associated asynchronous state is given up, any resources held by that associated asynchronous state are released.~~ An asynchronous return object or an asynchronous provider *release their associated asynchronous state* as follows. If the return object or provider contains the last reference to that state, destroys that state. Destroys the reference to that state.

After paragraph 5, add a new paragraph as follows.

> An asynchronous provider *makes ready an associated asynchronous state* by marking that state ready and then unblocking any threads waiting for the associated state to become ready. An asynchronous provider *abandons an associated asynchronous state* as follows. If that state is not ready, the provider stores an exception object of type `future_error` with an error condition of `broken_promise` within that state and then makes ready that state. The provider then releases that state.

30.6.5 Class template promise [futures.promise]

Edit paragraph 7, regarding the destructor, as follows.

> *Effects:* ~~if the associated asynchronous state of `*this` is not ready, stores an exception object of type `future_error` with an error condition of `broken_promise` Any threads blocked in a function waiting for the asynchronous state associated with `*this` to become ready are unblocked. Destroys `*this` and releases its reference to its associated asynchronous state if any. If this is the last reference to that associated asynchronous state, destroys that state.~~ abandons any associated asynchronous state ([futures.state]).

Edit paragraph 8, regarding the move assignment operator, as follows.

> *Effects:* abandons any associated asynchronous state ([futures.state]) and then as if `promise<R>(std::move(rhs)).swap(*this)`.

Remove paragraph 9, as it is now redundant with the effects.

> ~~*Postcondition:* `rhs` has no associated asynchronous state. `*this` has the associated asynchronous state of `rhs` prior to the assignment.~~

Edit paragraph 18, regarding `set_value`, as follows.

> *Effects:* atomically stores `r` in the associated asynchronous state and ~~sets that state to ready. Any threads blocked in a call of a blocking function of~~

~~any future that refers to the same associated asynchronous state as `*this` are unblocked.~~ makes ready that state ([futures.state]).

Edit paragraph 22, regarding `set_exception`, as follows.

Effects: atomically stores `p` in the associated asynchronous state and ~~sets that state to ready. Any threads blocked in a call of a blocking function of any future that refers to the same associated asynchronous state as `*this` are unblocked.~~ makes ready that state ([futures.state]).

Edit paragraph 26, regarding `set_value_at_thread_exit`, as follows.

Effects: Stores `r` in the associated asynchronous state without making ready ~~the associated asynchronous~~ that state ~~ready~~ immediately. Schedules ~~the associated asynchronous~~ that state to be made ready when the current thread exits, after all objects of thread storage duration associated with the current thread have been destroyed.

Edit paragraph 29, regarding `set_exception_at_thread_exit`, as follows.

Effects: Stores `p` in the associated asynchronous state without making ready ~~the associated asynchronous~~ that state ~~ready~~ immediately. Schedules ~~the associated asynchronous~~ that state to be made ready when the current thread exits, after all objects of thread storage duration associated with the current thread have been destroyed.

30.6.6 Class template future [futures.unique_future]

Edit paragraph 10, regarding the destructor, as follows.

Effects:

- ~~gives up the reference to its~~ releases any associated asynchronous state ([futures.state]).
- destroys `*this`.

Edit paragraph 11, regarding the move assignment operator, as follows.

Effects:

- ~~if `*this` referred to an associated asynchronous state prior to the assignment it gives up this reference.~~ releases any associated asynchronous state ([futures.state]).
- move assigns the contents of `rhs` to `*this`.

30.6.7 Class template shared_future [futures.shared_future]

Edit paragraph 13, regarding the destructor, as follows.

Effects:

- ~~gives up the reference to its~~ releases any associated asynchronous state ([futures.state]).
- destroys `*this`.

Edit paragraph 14, regarding the move assignment operator, as follows.

Effects:

- ~~if `*this` refers to an associated asynchronous state it gives up this reference.~~ releases any associated asynchronous state ([futures.state]).
- assigns the contents of `rhs` to `*this`.

Edit paragraph 16, regarding the copy assignment operator, as follows.

Effects:

- ~~if `*this` refers to an associated asynchronous state it gives up this reference.~~ releases any associated asynchronous state ([futures.state]).
- assigns the contents of `rhs` to `*this`. [*Note:* as a result, `*this` refers to the same associated asynchronous state as `rhs` (if any). — *end note*]

30.6.8 Class template atomic_future [futures.atomic_future]

Edit paragraph 9, regarding the destructor, as follows.

Effects:

- ~~gives up the reference to its~~ releases any associated asynchronous state ([futures.state]).
- destroys `*this`.

Edit paragraph 10, regarding the copy assignment operator, as follows.

Effects:

- releases any associated asynchronous state.
- assigns the contents of `rhs` to `*this`. [*Note:* as a result, `*this` refers to the same associated asynchronous state as `rhs` (if any). — *end note*]

30.6.10.1 packaged_task member functions [futures.task.members]

Edit paragraph 9, regarding the move assignment operator, as follows.

Effects:

- releases any associated asynchronous state ([futures.state]).
- `packaged_task<R, ArgTypes...>(other).swap(*this).`

Edit paragraph 10, regarding the destructor, as follows.

Effects: ~~if the associated asynchronous state of `*this` is not ready, stores an exception object of type `future_error` with an error code of `broken_promise`. Any threads blocked in a function waiting for the associated asynchronous state of `*this` to become ready are unblocked. Destroys `*this` and releases its reference to its associated asynchronous state (if any). If this is the last reference to that associated asynchronous state, destroys that state.~~ abandons any associated asynchronous state ([futures.state]).

Edit within paragraph 24, regarding `make_ready_at_thread_exit`, as follows.

.... this shall be done without ~~making the state ready~~ making ready that state ([futures.state]) immediately. ....

Edit paragraph 27, regarding `reset`, as follows.

Effects: ~~returns the object to a state as if a newly-constructed instance had just been assigned to `*this` by~~ `*this = packaged_task(std::move(f))`, where `f` is the task stored in `*this`. [*Note:* this constructs a new associated asynchronous state for `*this`. The old state is ~~discarded,~~ abandoned ([futures.state]). ~~as described in the destructor for `packaged_task`. `get_future` may now be called again for `*this`.~~ —*end note*]

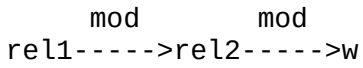
FCD 14882

UNITED KINGDOM (GB)

ADDITIONAL DETAILS TO BALLOT COMMENTS

B) The use of maximal in the definition of release sequence

The current wording of the standard suggests that release sequences are maximal with respect to sequence inclusion, i.e. that if there are two release operations in the modification order,



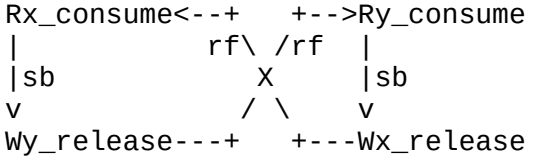
then [rel1;rel2;w] is the only release sequence, as the other candidate [rel2;w] is included in it. This interpretation precludes synchronizing with releases which have other releases sequenced-before them. We believe that the intention is actually to define the maximal release sequence from a particular release operation, which would admit both [rel1;rel2;w] and [rel2;w].

We suggest that 1.10:6 be changed to:

> A release sequence from a release operation A on an atomic object M is a maximal contiguous sub-sequence of side effects in the modification order of M, where the first operation is A, and every subsequent operation
> - is performed by the same thread that performed the release, or
> - is an atomic read-modify-write operation.

C) Inter-thread-happens-before is not acyclic

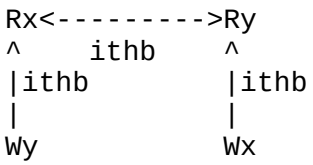
Inter-thread-happens-before (1.10:10) is not acyclic at the moment. The following example execution is valid according to the current draft and has cycles in its inter-thread-happens-before relation, but we believe the intent is to forbid it:



The diagram consists of four memory actions, each represented by either an 'R' or a 'W' for read or write, followed by a letter identifying an atomic object and a memory ordering parameter. For example the action 'Rx_consume' is a read of x that is ordered in memory as a consume.

The labelled arrows represent relationships between the operations. The vertical arrows labelled 'sb' are sequenced-before edges. The crossed diagonal arrows labelled 'rf' are reads-from edges that point from a write to the read that takes its value.

The reads-from edges make dependency-order edges with the same arrows. Transitively closing inter-thread-happens-before (ithb) gives the following relation:



Note that this is cyclic and that the execution is valid. We believe the intention is for inter thread happens before to be acyclic, and the standard will have to explicitly state this.

We suggest the following sentence be added to 1.10:

> Candidate executions with a cyclic inter-thread-happens-before should not be considered [Note: the existence of such a candidate execution does not introduce undefined behaviour].

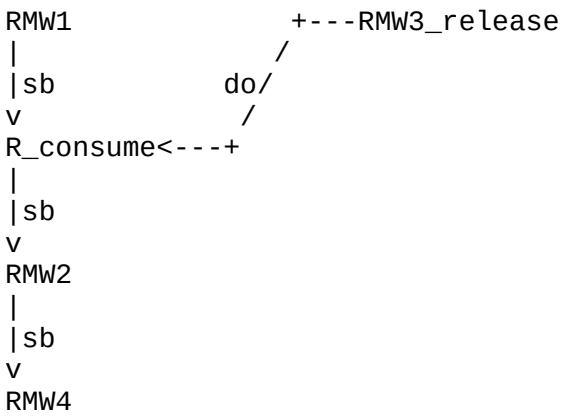
or

> The inter-thread happens-before relation shall be acyclic.

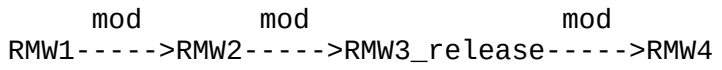
with a similar note.

E) Non-unique visible sequences of side effects and happens-before ordering
---

In 1.10:12 the standard allows multiple visible sequences of side effects (vsse's) for a given read (despite the use of "The" at the start of 1.10:13). We will demonstrate this by constructing an execution with two vsse's. The following execution has five memory operations, four of which are read modify writes (RMW's). There are two threads, one with four operations each ordered by sequenced before (sb), the other with a single RMW release.



The modification order in this example is as follows:



There are two visible sequences of side effects here for the read consume:

[RMW1,RMW2] and [RMW3,RMW4]

The R_consume here must read from the later vsse in modification order for the dependency_ordered edge to exist. The existence of two vsse's relies on the lack of transitivity of happens before (which only occurs in the presence of consume operations).

Given that there is not a unique vsse for a given side effect, the standard should be changed from defining "the" vsse to defining "a" vsse. A note can explain that we can only ever read from one vsse.

In addition to non-uniqueness, if every element in a vsse happens-before a read, the read should not take the value of the visible side effect. We can prevent this by removing the word

"subsequent" from the definition of vsse in 1.10:12 (as suggested by Hans).

Collecting the two issues together, our suggestion is to change 1.10:12 to:

> "A visible sequence of side effects on an atomic object M, with respect to a value computation B of M, is a maximal contiguous sub-sequence of side effects in the modification order of M, where the first side effect is visible with respect to B, and for every side effect, it is not the case that B happens before it. The value of an atomic object M, as determined by evaluation B, shall be the value stored by some operation in a visible sequence of M with respect to B. [...] [Note: For a given value computation B, there is only one visible sequence of side effects that B can read from, even if more than one sequence can be constructed. -end note]"

FCD 14882
GB 12

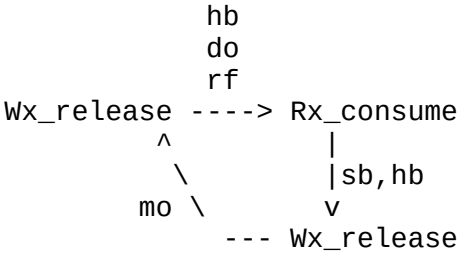
F) Alternative definition of the value read by an atomic operation
---

The standard introduces visible side effects, which are used first to define the values read by non-atomic operations. They are then re-used to constrain the value read by atomic operations: 1.10:13 says that an atomic operation must read from somewhere in "the" visible sequence of side effects, which must start from *a* visible side effect, ie a side effect that (a) happens before the read, and (b) is not happens-before-hidden.

We suspect that this re-use of the notion of visible side effect may be a drafting artifact, and tentatively suggest that it would be clearer to remove the requirement that there is a vse for atomics, replacing the first two sentences of 1.10:13 by

> "An atomic operation must read from somewhere in the modification order that is not happens-before-hidden and does not follow (in modification order) any side effect that happens-after the read."

Note that the current text allows the following, whereas the revised text would forbid it. We believe that in a direct implementation on hardware, this would be forbidden by coherence.



FCD 14882
GB 13

G) Wording of the read-read coherence condition

In 1.10:13 a coherence condition is stated on the values of atomic reads:

> "Furthermore, if a value computation A of an atomic object M happens before a value computation B of M, and the value computed by A corresponds to the value stored by side effect X, then the value computed by B shall either equal the value computed by A, or be the value stored by side effect Y, where Y follows X in the modification order of M."

The words "corresponds to" are not used elsewhere in the standard, as far as we can see, and it is unclear whether they have a special meaning here. In addition taking the value of the read B from the value read by A seems unnecessarily indirect. B could take its value from X instead. We suggest that the prose be changed to the following:

> "Furthermore, if a value computation A of an atomic object M happens before a value computation B of M, and A takes its value from the side effect X, then the value computed by B shall either be the value stored by X, or the value stored by a side effect Y, where Y follows X in the modification order of M."

FCD 14882
GB 14

I) Implementation of dependencies on Power/ARM

Carries-a-dependency includes via-memory dependencies (from a write to a program-order-later read from the same location on the same processor), but the Power and ARM architectures do not guarantee that such dependencies are respected by all processors. Hence, we wonder how implementations are expected to guarantee dependency-ordered before.

J) Control dependencies for atomics

Given the examples of compilers interchanging data and control dependencies, and that control dependencies are respected on Power/ARM for load->store (and on Power for load->load with a relatively cheap isync), we're not sure why carries-a-dependency-to does not include control dependencies between atomics.

L) Minor editorial changes

1.10:12 last note: "...as defined here..." should be "...as defined below...".

Date: 2010-06-28

Problems with bitmask types in the library

The library defines the *bitmask types* (17.5.2.1.3) `ios_base::fmtflags`, `ios_base::iostate`, `ios_base::openmode`, `regex_constants::syntax_option_type` and `regex_constants::match_flag_type`. Each is defined as an unscoped enumeration without a fixed underlying type. Each has `operator&`, `operator|` and `operator~` overloaded.

The library also defines the *enumerated types* (17.5.2.1.2) `ios_base::seekdir` and `regex_constants::error_type`.

There are several problems related to these types:

- Bitmask type operators declared as members of `ios_base`,
- bitmask type operators defined for enumerated types,
- `operator^` is missing,
- `operator~` is defined incorrectly,
- the underlying type should be fixed,
- value of `goodbit` is not unspecified,
- bitmask types are inconsistent with the requirements and over-specified.

Bitmask operators declared as members of `ios_base`

The bitwise operators for the `ios_base` bitmask types are declared in the body of `ios_base`, making them members of the class, but they should be non-member functions. This is largely an editorial issue, but is addressed by the proposed changes below.

Bitmask type operators defined for enumerated types

As specified in 27.5.2p1 and 27.5.2.1.5p1, `ios_base::seekdir` is an enumerated type (17.5.2.1.2) not a bitmask type (17.5.2.1.3) and so should not have bitwise operators defined. The operators were added by the `constexpr` proposal, n2349, but should be removed.

The same applies to `regex_constants::error_type` (28.5.3) although 28.5.3p1 incorrectly uses the term "enumeration type" instead of "enumerated type".

`operator^` is missing

n2349 proposed the following additional overload which is missing from the FCD:

```
constexpr fmtflags operator^(fmtflags lhs, fmtflags rhs)
{
```

```
   return fmtflags(int(lhs) ^ int(rhs));
}
```

This operator would be needed for all the bitmask types if they were defined as enumeration types.

`operator~` is defined incorrectly

The FCD specifies:

```
constexpr fmtflags operator~(fmtflags f)
{
   return fmtflags( f);
}
```

This seems to be an editorial error as n2349 proposed:

```
constexpr fmtflags operator~(fmtflags f)
{
   return fmtflags(~(f));
}
```

This isn't right either, since it will lead to infinite recursion. I think the intended definition is:

```
constexpr fmtflags operator~(fmtflags f)
{
   return fmtflags(~int(f));
}
```

The underlying type should be fixed

Bitmask types need `operator~` in order to clear a value from the bitmask type, as show in 17.5.2.1.3 paragraph 4:

> — To *clear* a value *Y* in an object *X* is to evaluate the expression *X &= ~Y*.

However the definition for `fmtflags` above does not have well-specified behaviour if the underlying type is smaller than `int`, because `~int(f)` is likely to produce a value outside the range of `fmtflags`.

The same problem is present in the example implementation in 17.5.2.1.3 because `int_type` is only stated to be capable of holding all the values of *bitmask* so there is no guarantee that `sizeof(int_type) == sizeof(bitmask)` and therefore no guarantee that the code given for `operator~` works as intended.

This could be solved by giving the enumeration a fixed underlying type which is the same type as used for the conversions used by the bitwise operators. An alternative solution would be to do all conversions to and from `std::underlying_type<bitmask>`.

Value of `goodbit` is not unspecified.

The synopsis in 27.5.2 includes:

```
goodbit = unspecified,
```

but 27.5.2.1.3 specifies that `goodbit` has the value zero.

Bitmasks types are inconsistent with the requirements and over-specified.

In C++03 implementions were allowed to use any of the options in 17.5.2.1.3 for the bitmask types defined by the library and e.g. `std::ios::in` is an object of bitmask type with an implementation-defined value. The FCD specifies that bitmask types are implemented as enumerations and `std::ios::in` is an enumerator not an object. Not only is this inconsistent with the stated requirements in 17.5.2.1.3 but it means that existing implementations which define bitmask types as an integer type must change, and it breaks the following valid C++03 code:

```
#include <ios>
std::ios::openmode* p = &std::ios::in;
```

If all the library's bitmask types are required to be enumeration types then there is no reason for 17.5.2.1.3 to allow integer types or a `bitset`.

Existing C++03 implementations choose to implement bitmask types as integer types for efficiency, because enumerations with overloaded operators are less efficient without `constexpr`. Since the FCD defines the bitmask types as enumerations with overloaded `constexpr` operators, implementations cannot provide a common definition for C++03 and C++0x. This is of no benefit to users or implementations.

Proposed Solution

The over-specification of the library's bitmask types should be reverted. The example bitmask in clause 17 can be updated to use `constexpr` and an enumeration type with a fixed underlying type. `constexpr` can also be used for declaring the objects of bitmask type. Proposed wording for this change is given below.

An alternative solution would be to fix all the issues listed above except the over-specification, but this would still require existing implementations to change, has the chance of breaking valid C++03 programs, and leaves the definitions of bitmask types inconsistent with the requirements in Clause 17.

Proposed Wording

Modifications to Bitmask types

Make the following changes to 17.5.2.1.3:

The bitmask type *bitmask* can be written:

```
// For exposition only.
// int_type is an integral type capable of
// representing all values of bitmask
enum bitmask : int_type {
  V0 = 1 << 0, V1 = 1 << 1, V2 = 1 << 2, V3 = 1 << 3, .....
};

static constconstexpr bitmask C0 (V0);
static constconstexpr bitmask C1 (V1);
static constconstexpr bitmask C2 (V2);
static constconstexpr bitmask C3 (V3);
  .....

// For exposition only.
// int_type is an integral type capable of
// representing all values of bitmask
constexpr bitmask operator& (bitmask X , bitmask Y ) {
  return static_cast<bitmask >(
    static_cast<int_type>(X ) &
    static_cast<int_type>(Y ));
}
constexpr bitmask operator| (bitmask X , bitmask Y ) {
  return static_cast<bitmask >(
    static_cast<int_type>(X ) |
    static_cast<int_type>(Y ));
}
constexpr bitmask operator^ (bitmask X , bitmask Y ){
  return static_cast<bitmask >(
    static_cast<int_type>(X ) ^
    static_cast<int_type>(Y ));
}
constexpr bitmask operator~ (bitmask X ){
  return static_cast<bitmask >(~static_cast<int_type>(X ));
}
constexpr bitmask & operator&=(bitmask & X , bitmask Y ){
  X = X &Y ; return X ;
}
constexpr bitmask & operator|=(bitmask & X , bitmask Y ) {
  X = X |Y ; return X ;
}
constexpr bitmask & operator^=(bitmask & X , bitmask Y ) {
  X = X ^Y ; return X ;
}
```

Modifications to Class `ios_base`

Make the following changes to the synopsis in 27.5.2:

```
class ios_base {
public:
  class failure;

  typedef T1 fmtflags;
  constexpr fmtflags boolalpha = unspecified;
  constexpr fmtflags dec = unspecified;
  constexpr fmtflags fixed = unspecified;
  constexpr fmtflags hex = unspecified;
  constexpr fmtflags internal = unspecified;
  constexpr fmtflags left = unspecified;
  constexpr fmtflags oct = unspecified;
  constexpr fmtflags right = unspecified;
  constexpr fmtflags scientific = unspecified;
  constexpr fmtflags showbase = unspecified;
  constexpr fmtflags showpoint = unspecified;
  constexpr fmtflags showpos = unspecified;
  constexpr fmtflags skipws = unspecified;
  constexpr fmtflags unitbuf = unspecified;
  constexpr fmtflags uppercase = unspecified;
  constexpr fmtflags adjustfield = unspecified;
  constexpr fmtflags basefield = unspecified;
  constexpr fmtflags floatfield = unspecified;

  // 27.5.2.1.2 fmtflags
  enum fmtflags {
    boolalpha = unspecified,
    dec = unspecified,
    fixed = unspecified,
    hex = unspecified,
    internal = unspecified,
    left = unspecified,
    oct = unspecified,
    right = unspecified,
    scientific = unspecified,
    showbase = unspecified,
    showpoint = unspecified,
    showpos = unspecified,
    skipws = unspecified,
    unitbuf = unspecified,
    uppercase = unspecified,
    adjustfield = unspecified,
    basefield = unspecified,
    floatfield = unspecified,
  };

  constexpr fmtflags operator~(fmtflags f);
  constexpr fmtflags operator&(fmtflags lhs, fmtflags rhs);
  constexpr fmtflags operator|(fmtflags lhs, fmtflags rhs);

  typedef T2 iostate;
  constexpr iostate badbit = unspecified;
```

```
  constexpr iostate eofbit = unspecified;
  constexpr iostate failbit = unspecified;
  constexpr iostate goodbit{ 0 };

  // 27.5.2.1.3 iostate
  enum iostate {
    badbit = unspecified,
    eofbit = unspecified,
    failbit = unspecified,
    goodbit = unspecified,
  };

  constexpr iostate operator~(iostate f);
  constexpr iostate operator&(iostate lhs, iostate rhs);
  constexpr iostate operator|(iostate lhs, iostate rhs);
```

```
  typedef T3 openmode;
  constexpr openmode app = unspecified;
  constexpr openmode ate = unspecified;
  constexpr openmode binary = unspecified;
  constexpr openmode in = unspecified;
  constexpr openmode out = unspecified;
  constexpr openmode trunc = unspecified;

  // 27.5.2.1.4 openmode
  enum openmode {
    app = unspecified,
    ate = unspecified,
    binary = unspecified,
    in = unspecified,
    out = unspecified,
    trunc = unspecified,
  };

  constexpr openmode operator~(openmode f);
  constexpr openmode operator&(openmode lhs, openmode rhs);
  constexpr openmode operator|(openmode lhs, openmode rhs);
```

```
  typedef T4 seekdir;
  constexpr seekdir beg = unspecified;
  constexpr seekdir cur = unspecified;
  constexpr seekdir end = unspecified;
```

```
  // 27.5.2.1.5 seekdir
  enum seekdir {
    beg = unspecified,
    cur = unspecified,
    end = unspecified,
  };

  constexpr seekdir operator~(seekdir f);
  constexpr seekdir operator&(seekdir lhs, seekdir rhs);
```

~~constexpr seekdir operator|(seekdir lhs, seekdir rhs);~~

Revert the changes from n2349, making the following changes to 27.5.2.1.2:

typedef *T1* fmtflags;
~~enum fmtflags;~~

1 The type `fmtflags` is a bitmask type (17.5.2.1.3). Setting its elements has the effects indicated in Table 112.

2 Type `fmtflags` also defines the constants indicated in Table 113.

~~constexpr fmtflags ios_base::operator~(fmtflags f);~~

~~3 *Returns:* fmtflags(f).~~

~~constexpr fmtflags ios_base::operator&(fmtflags lhs, fmtflags rhs);~~

~~4 *Returns:* fmtflags(int(lhs) & int(rhs)).~~

~~constexpr fmtflags ios_base::operator|(fmtflags lhs, fmtflags rhs);~~

~~5 *Returns:* fmtflags(int(lhs) | int(rhs)).~~

Make the following changes to 27.5.2.1.3:

typedef *T2* iostate;
~~enum iostate;~~

1 The type `iostate` is a bitmask type (17.5.2.1.3) that contains the elements indicated in Table 114.

~~constexpr iostate ios_base::operator~(iostate f);~~

~~2 *Returns:* iostate(f).~~

~~constexpr iostate ios_base::operator&(iostate lhs, iostate rhs);~~

~~3 *Returns:* iostate(int(lhs) & int(rhs)).~~

~~constexpr iostate ios_base::operator|(iostate lhs, iostate rhs);~~

~~4 *Returns:* iostate(int(lhs) | int(rhs)).~~

Make the following changes to 27.5.2.1.4:

```
typedef T3 openmode;
enum openmode;
```

1 The type openmode is a bitmask type (17.5.2.1.3). It contains the elements indicated in Table 115.

```
~~constexpr openmode ios_base::operator~(openmode f);~~

~~2 Returns: openmode(f).~~

~~constexpr openmode ios_base::operator&(openmode lhs, openmode rhs);~~

~~3 Returns: openmode(int(lhs) & int(rhs)).~~

~~constexpr openmode ios_base::operator|(openmode lhs, openmode rhs);~~

~~4 Returns: openmode(int(lhs) | int(rhs)).~~
```

Make the following changes to 27.5.2.1.5:

```
typedef T4 seekdir;
enum seekdir;
```

1 The type seekdir is an enumerated type (17.5.2.1.2) that contains the elements indicated in Table 116.

```
~~constexpr seekdir ios_base::operator~(seekdir f);~~

~~2 Returns: seekdir(f).~~

~~constexpr seekdir ios_base::operator&(seekdir lhs, seekdir rhs);~~

~~3 Returns: seekdir(int(lhs) & int(rhs)).~~

~~constexpr seekdir ios_base::operator|(seekdir lhs, seekdir rhs);~~

~~4 Returns: seekdir(int(lhs) | int(rhs)).~~
```

Modifications to Bitmask Type `regex_constants::syntax_option_type`

Make the following changes to 28.5.1:

```
namespace std {
 namespace regex_constants {
  typedef bitmask_type syntax_option_type;
```

```
  constexpr syntax_option_type icase = unspecified;
  constexpr syntax_option_type nosubs = unspecified;
  constexpr syntax_option_type optimize = unspecified;
  constexpr syntax_option_type collate = unspecified;
  constexpr syntax_option_type ECMAScript = unspecified;
  constexpr syntax_option_type basic = unspecified;
  constexpr syntax_option_type extended = unspecified;
  constexpr syntax_option_type awk = unspecified;
  constexpr syntax_option_type grep = unspecified;
  constexpr syntax_option_type egrep = unspecified;

- enum syntax_option_type {
-   icase = implementation-defined,
-   nosubs = implementation-defined,
-   optimize = implementation-defined,
-   collate = implementation-defined,
-   ECMAScript = implementation-defined,
-   basic = implementation-defined,
-   extended = implementation-defined,
-   awk = implementation-defined,
-   grep = implementation-defined,
-   egrep = implementation-defined,
- };
- constexpr syntax_option_type operator~(syntax_option_type f);
- constexpr syntax_option_type operator&(syntax_option_type lhs,
syntax_option_type rhs);
- constexpr syntax_option_type operator|(syntax_option_type lhs,
syntax_option_type rhs);
 }
}
```

1 The type `syntax_option_type` is an implementation-defined bitmask type (17.5.2.1.3). Setting its elements has the effects listed in table 128. A valid value of type `syntax_option_type` shall have exactly one of the elements `ECMAScript`, `basic`, `extended`, `awk`, `grep`, `egrep`, set.

```
- constexpr syntax_option_type operator~(syntax_option_type f);
```

2 *Returns:* `syntax_option_type(f).`

```
- constexpr syntax_option_type operator&(syntax_option_type lhs,
syntax_option_type rhs);
```

3 *Returns:* `syntax_option_type(int(lhs) & int(rhs)).`

```
- constexpr syntax_option_type operator|(syntax_option_type lhs,
syntax_option_type rhs);
```

4 *Returns:* `syntax_option_type(int(lhs) | int(rhs)).`

Modifications to Bitmask Type `regex_constants::syntax_option_type`

Make the following changes to 28.5.2:

```
namespace std {
 namespace regex_constants{
  typedef bitmask_type match_flag_type;
  constexpr match_flag_type match_default{ 0 };
  constexpr match_flag_type match_not_bol = unspecified;
  constexpr match_flag_type match_not_eol = unspecified;
  constexpr match_flag_type match_not_bow = unspecified;
  constexpr match_flag_type match_not_eow = unspecified;
  constexpr match_flag_type match_any = unspecified;
  constexpr match_flag_type match_not_null = unspecified;
  constexpr match_flag_type match_continuous = unspecified;
  constexpr match_flag_type match_prev_avail = unspecified;
  constexpr match_flag_type format_default{ 0 };
  constexpr match_flag_type format_sed = unspecified;
  constexpr match_flag_type format_no_copy = unspecified;
  constexpr match_flag_type format_first_only = unspecified;

  enum match_flag_type {
    match_default = 0,
    match_not_bol = implementation-defined,
    match_not_eol = implementation-defined,
    match_not_bow = implementation-defined,
    match_not_eow = implementation-defined,
    match_any = implementation-defined,
    match_not_null = implementation-defined,
    match_continuous = implementation-defined,
    match_prev_avail = implementation-defined,
    format_default = 0,
    format_sed = implementation-defined,
    format_no_copy = implementation-defined,
    format_first_only = implementation-defined,
  };
  constexpr match_flag_type operator~(match_flag_type f);
  constexpr match_flag_type operator&(match_flag_type lhs,
match_flag_type rhs);
  constexpr match_flag_type operator|(match_flag_type lhs,
match_flag_type rhs);
 }
}
```

1 The type `regex_constants::match_flag_type` is an implementation-defined bitmask type (17.5.2.1.3). Matching a regular expression against a sequence of characters `[first,last)` proceeds according to the rules of the grammar specified for the regular expression object, modified according to the effects listed in table 129 for any bitmask elements set.

```
  constexpr match_flag_type operator~(match_flag_type f);
```

2 *Returns:* `match_flag_type(f).`

~~constexpr match_flag_type operator&(match_flag_type lhs, match_flag_type rhs);~~

~~3 *Returns:* match_flag_type(int(lhs) & int(rhs)).~~

~~constexpr match_flag_type operator|(match_flag_type lhs, match_flag_type rhs);~~

~~4 *Returns:* match_flag_type(int(lhs) | int(rhs)).~~

Modifications to Bitmask Type `regex_constants::error_type`

Make the following changes to 28.5.3:

```
namespace std {
 namespace regex_constants {
  typedef implementation-defined error_type;
  constexpr error_type error_collate = unspecified;
  constexpr error_type error_ctype = unspecified;
  constexpr error_type error_escape = unspecified;
  constexpr error_type error_backref = unspecified;
  constexpr error_type error_brack = unspecified;
  constexpr error_type error_paren = unspecified;
  constexpr error_type error_brace = unspecified;
  constexpr error_type error_badbrace = unspecified;
  constexpr error_type error_range = unspecified;
  constexpr error_type error_space = unspecified;
  constexpr error_type error_badrepeat = unspecified;
  constexpr error_type error_complexity = unspecified;
  constexpr error_type error_stack = unspecified;

  enum error_type {
    error_collate = implementation-defined,
    error_ctype = implementation-defined,
    error_escape = implementation-defined,
    error_backref = implementation-defined,
    error_brack = implementation-defined,
    error_paren = implementation-defined,
    error_brace = implementation-defined,
    error_badbrace = implementation-defined,
    error_range = implementation-defined,
    error_space = implementation-defined,
    error_badrepeat = implementation-defined,
    error_complexity = implementation-defined,
    error_stack = implementation-defined,
  };
  constexpr error_type operator~(error_type f);
  constexpr error_type operator&(error_type lhs, error_type rhs);
  constexpr error_type operator|(error_type lhs, error_type rhs);
 }
}
```

1 The type `error_type` is an implementation-defined enumerat~~ion~~ed type (17.5.2.1.2). Values of type `error_type` represent the error conditions described in table 130:

~~`constexpr error_type operator~(error_type f);`~~

~~2 *Returns:* `error_type(f).`~~

~~`constexpr error_type operator&(error_type lhs, error_type rhs);`~~

~~3 *Returns:* `error_type(int(lhs) & int(rhs)).`~~

~~`constexpr error_type operator|(error_type lhs, error_type rhs);`~~

~~4 *Returns:* `error_type(int(lhs) | int(rhs)).`~~

FCD 14882
GB 80

A) Reads of indeterminate value result in undefined behaviour
---

In 20.2.3, NullablePointer requirements [nullablepointer.requirements], the standard specifies the behaviour of programs that read indeterminate values:

> ... A default-initialized object of type P may have an indeterminate value. [Note: Operations involving indeterminate values may cause undefined behaviour. end note]

The note uses the word "may", but we believe the intention is that such reads will cause undefined behaviour, but implementations are not required to produce an error. We suggest changing the note to:

> [Note: Operations involving indeterminate values cause undefined behaviour. end note]

D) Imposed happens-before edges are not made transitive

At various points in the standard new edges are added to happens-before, for example 27.2.3:2 adds happens-before edges between writes and reads from a stream:

> If one thread makes a library call a that writes a value to a stream and, as a result, another thread reads this value from the stream through a library call b such that this does not result in a data race, then a happens before b.

Happens-before is defined in 1.10:11 in a deliberate way that makes it not explicitly transitively closed. Adding edges to happens-before directly, as in 27.2.3:2, does not provide transitivity with sequenced-before or any other existing happens-before edge. This lack of transitivity seems to be unintentional. In order to achieve transitivity we suggest each edge be added to inter-thread-happens-before as a synchronises-with edge (as per conversation with Hans Boehm). In the standard, each use of the words "happens-before" should be replaced with the words "synchronizes-with" in the following sentences:

> 27.2.3:2
> 30.3.1.2:6
> 30.3.1.5:7
> 30.6.4:7
> 30.6.9:5
> 30.6.10.1:23

FCD 14882
GB 131

H) Overlapping evaluations are allowed

29.3:8 states:

> "An atomic store shall only store a value that has been computed from constants and program input values by a finite sequence of program evaluations, such that each evaluation observes the values of variables as computed by the last prior assignment in the sequence."

... but 1.9.13 states:

> "If A is not sequenced before B and B is not sequenced before A, then A and B are unsequenced. [Note: The execution of unsequenced evaluations can overlap. -end note]"

Overlapping executions can make it impossible to construct the sequence described in 29.3:8. We are not sure of the intention here and do not offer a suggestion for change, but note that 29.3:8 is the condition that prevents out-of-thin-air reads.

FCD 14882
GB 136

K) Initialisation of atomics

We believe the intent is that for any atomics there is a distinguished initialisation write, but that this need not happens-before all the other operations on that atomic - specifically so that the initialisation write might be non-atomic and hence give rise to a data race, and hence undefined behaviour, in examples such as this (from Hans):

```
    atomic< atomic<int> * > p
    f()                          |
    { atomic<int>x;              |   W_na  x
      p.store(&x,mo_rlx);        |   W_rlx p=&x
    }                            |
```

(where na is nonatomic and rlx is relaxed). We suspect also that no other mixed atomic/nonatomic access to the same location is intended to be permitted. Either way, a note would probably help.

FCD 14882: GB 138

Date: 2010-04-20

Author: Anthony Williams
Just Software Solutions Ltd

Lockable requirements for C++0x

This paper provides a proposed resolution for LWG issue 1268. The basic premise of that issue is that the "Mutex requirements" from the current working draft are worded as if they are requirements on all lockable types, including user-defined mutexes and instantiations of `unique_lock`. However, the requirements really only need apply to the standard mutex types such as `std::mutex`, and are too strong when applied to user-defined mutex types.

This paper therefore proposes to separate the existing requirements on the standard mutex types from the general requirements on all lockable types.

Proposed wording

Add a new section to 30.2 [thread.req] after 30.2.4 [thread.req.timing] as follows:

> 30.2.5 Requirements for Lockable types
>
> The standard library templates `unique_lock` (30.4.3.2 [thread.lock.unique]), `lock_guard` (30.4.3.1 [thread.lock.guard]), `lock`, `try_lock` (30.4.4 [thread.lock.algorithm]) and `condition_variable_any` (30.5.2 [thread.condition.condvarany]) all operate on user-supplied lockable objects. Such an object must support the member functions specified for either the `BasicLockable` requirements, the `Lockable` requirements or the `TimedLockable` requirements as appropriate to acquire or release ownership of a `lock` by a given `thread`. [*Note:* the nature of any lock ownership and any synchronization it may entail are not part of these requirements. — *end note*]
>
> 30.2.5.1 BasicLockable Requirements
>
> In order for a type `L` to qualify as a `BasicLockable` type, the following expressions must be supported, with the specified semantics, where `m` denotes a value of type `L`:
>
> The expression `m.lock()` shall be well-formed and have the following semantics:
>
> Effects:

Block until a lock can be acquired for the current thread. If an exception is thrown then a lock shall not have been acquired for the current thread.

Return type:

`void`

The expression `m.unlock()` shall be well-formed and have the following semantics:

Effects:

Release a lock on `m` held by the current thread.

Return type:

`void`

Throws:

Nothing if the current thread holds a lock on `m`.

30.2.5.2 Lockable Requirements

In order for a type `L` to qualify as a `Lockable` type, it must meet the `BasicLockable` requirements. In addition, the following expressions must be supported, with the specified semantics, where `m` denotes a value of type `L`:

The expression `m.try_lock()` shall be well-formed and have the following semantics:

Effects:

Attempt to acquire a lock for the current thread without blocking. If an exception is thrown then a lock shall not have been acquired for the current thread.

Return type:

`bool`

Returns:

`true` if the lock was acquired, `false` otherwise.

30.2.5.3 `TimedLockable` Requirements

For a type `TL` to qualify as `TimedLockable` it must meet the `Lockable` requirements, and additionally the following expressions must be well-formed, with the specified semantics, where `m` is an instance of a type `TL`, `rel_time` denotes instantiation of `duration` (20.10.3 [time.duration]) and `abs_time` denotes an instantiation of `time_point` (20.10.4 [time.point])

The expression `m.try_lock_for(rel_time)` shall be well-formed and have the following semantics:

Effects:

Attempt to acquire a lock for the current thread within the specified time period. If an exception is thrown then a lock shall not have been acquired for the current thread.

Return type:
`bool`

Returns:
`true` if the lock was acquired, `false` otherwise.

The expression `m.try_lock_until(abs_time)` shall be well-formed and have the following semantics:

Effects:
Attempt to acquire a lock for the current thread before the specified point in time. If an exception is thrown then a lock shall not have been acquired for the current thread.

Return type:
`bool`

Returns:
`true` if the lock was acquired, `false` otherwise.

Replace 30.4.1 [thread.mutex.requirements] paragraph 2 with the following:

2 This section describes requirements on ~~template argument types used to instantiate templates defined in~~ the mutex types supplied by the C++ standard library. ~~The template definitions in the C++ standard library refer~~ These types shall conform to the named `Mutex` requirements whose details are set out below. In this description, `m` is an object of ~~a `Mutex` type~~ one of the standard library mutex types `std::mutex`, `std::recursive_mutex`, `std::timed_mutex` or `std::recursive_timed_mutex`.

Add the following paragraph after 30.4.1 [thread.mutex.requirements] paragraph 2:

A `Mutex` type shall conform to the `Lockable` requirements (30.2.5.2).

Replace 30.4.2 [thread.timedmutex.requirements] paragraph 1 with the following:

The C++ standard library `TimedMutex` types `std::timed_mutex` and `std::recursive_timed_mutex` ~~A `TimedMutex` type~~ shall meet the requirements for a `Mutex` type. In addition, ~~it~~they shall meet the requirements set out ~~in this Clause 30.4.2~~below, where `rel_time` denotes an instantiation of `duration` (20.10.3 [time.duration]) and `abs_time` denotes an instantiation of `time_point` (20.10.4 [time.point]).

Add the following paragraph after 30.4.2 [thread.timedmutex.requirements] paragraph 1:

A `TimedMutex` type shall conform to the `TimedLockable` requirements (30.2.5.3).

Add the following paragraph following 30.4.3.1 [thread.lock.guard] paragraph 1:

> The supplied `Mutex` type shall meet the `Lockable` requirements (30.2.5.2).

Add the following paragraph following 30.4.3.2 [thread.lock.unique] paragraph 1:

> The supplied `Mutex` type shall meet the `Lockable` requirements (30.2.5.2). `unique_lock<Mutex>` meets the `Lockable` requirements. If `Mutex` meets the `TimedLockable` requirements (30.2.5.3) then `unique_lock<Mutex>` also meets the `TimedLockable` requirements.

Replace the use of "mutex" or "mutex object" with "lockable object" throughout clause 30.4.3.
30.4.3 [thread.mutex.locks] paragraph 1:

> 1 A lock is an object that holds a reference to a ~~mutex~~lockable object and may unlock the ~~mutex~~lockable object during the lock's destruction (such as when leaving block scope). A thread of execution may use a lock to aid in managing ~~mutex~~ ownership of a lockable object in an exception safe manner. A lock is said to own a ~~mutex~~lockable object if it is currently managing the ownership of that ~~mutex~~lockable object for a thread of execution. A lock does not manage the lifetime of the ~~mutex~~lockable object it references. [Note: Locks are intended to ease the burden of unlocking the ~~mutex~~lockable object under both normal and exceptional circumstances. — end note]

30.4.3 [thread.lock] paragaph 2:

> 2 Some lock constructors take tag types which describe what should be done with the ~~mutex~~lockable object during the lock's constuction.

30.4.3.1 [thread.lock.guard] paragaph 1:

> 1 An object of type `lock_guard` controls the ownership of a ~~mutex~~lockable object within a scope. A `lock_guard` object maintains ownership of a ~~mutex~~lockable object throughout the `lock_guard` object's lifetime. The behavior of a program is undefined if the ~~mutex~~lockable object referenced by `pm` does not exist for the entire lifetime (3.8) of the `lock_guard` object. `Mutex` shall meet the `Lockable` requirements (30.2.5.2).

30.4.3.2 [thread.lock.unique] paragaph 1:

> 1 An object of type `unique_lock` controls the ownership of a ~~mutex~~lockable object within a scope. ~~Mutex~~ ~~o~~Ownership of the lockable object may be acquired at construction or after construction, and may be transferred, after acquisition, to another `unique_lock` object. Objects of type `unique_lock` are not copyable but are movable. The behavior of a program is undefined if the contained pointer `pm` is not null and the mutex pointed to by `pm` does not exist for the entire remaining

lifetime (3.8) of the `unique_lock` object. `Mutex` shall meet the `Lockable` requirements (30.2.5.2).

Add the following to the precondition of `unique_lock(mutex_type& m, const chrono::time_point<Clock, Duration>& abs_time)` in 30.4.3.2.1 [thread.lock.unique.cons] paragraph 18:

```
template <class Clock, class Duration>
  unique_lock(mutex_type& m, const chrono::time_point<Clock,
Duration>& abs_time);
```

18 *Requires:* If `mutex_type` is not a recursive mutex the calling thread does not own the mutex. The supplied `mutex_type` type shall meet the `TimedLockable` requirements (30.2.5.3).

Add the following to the precondition of `unique_lock(mutex_type& m, const chrono::duration<Rep, Period>& rel_time)` in 30.4.3.2.1 [thread.lock.unique.cons] paragraph 22

22 *Requires:* If `mutex_type` is not a recursive mutex the calling thread does not own the mutex. The supplied `mutex_type` type shall meet the `TimedLockable` requirements (30.2.5.3).

Add the following as a precondition of `bool try_lock_until(const chrono::time_point<Clock, Duration>& abs_time)` before 30.4.3.2.2 [thread.lock.unique.locking] paragraph 10

```
template <class Clock, class Duration>
  bool try_lock_until(const chrono::time_point<Clock, Duration>&
abs_time);
```

Requires: The supplied `mutex_type` type shall meet the `TimedLockable` requirements (30.2.5.3).

Add the following as a precondition of `bool try_lock_for(const chrono::duration<Rep, Period>& rel_time)` before 30.4.3.2.2 [thread.lock.unique.locking] paragraph 15

```
template <class Rep, class Period>
  bool try_lock_for(const chrono::duration<Rep, Period>&
rel_time);
```

Requires: The supplied `mutex_type` type shall meet the `TimedLockable` requirements (30.2.5.3).

Replace 30.4.4 [thread.lock.algorithm] p1 with the following:

```
template <class L1, class L2, class... L3> int try_lock(L1&, L2&,
L3&...);
```

1 *Requires:* Each template parameter type shall meet the ~~`Mutex`~~ `Lockable` requirements (30.2.5.2).~~, except that a call to `try_lock()` may throw an exception.~~ [*Note:* The `unique_lock` class template meets these requirements when suitably instantiated. — *end note*]

Replace 30.4.4 [thread.lock.algorithm] p4 with the following:

```
template <class L1, class L2, class... L3> void lock(L1&, L2&,
L3&...);
```

4 *Requires:* Each template parameter type shall meet the ~~`Mutex`~~ `Lockable` requirements (30.2.5.2).~~, except that a call to `try_lock()` may throw an exception.~~ [*Note:* The `unique_lock` class template meets these requirements when suitably instantiated. — *end note*]

Replace 30.5.2 [thread.condition.condvarany] paragraph 1 with:

1 A `Lock` type shall meet the ~~requirements for a `Mutex` type, except that `try_lock` is not required~~ `BasicLockable` requirements (30.2.5.1). [*Note:* All of the standard mutex types meet this requirement. — *end note*]

FCD 14882

FINLAND (FI)

ADDITIONAL DETAILS TO BALLOT COMMENTS

Defaulting non-public special member functions on first declaration

Date: 2010-06-13

Version: FCD14882: FI-1, FI-2, FI-3

Authors: Ville Voutilainen <ville.voutilainen@gmail.com>

Abstract

The resolution of Core Issue 906 forbids defaulting a non-public special member function on its first declaration. I believe this resolution to be incorrect, and this document explains why. This document proposes that defaulting a non-public special member function on its first declaration is an important part of design vocabulary, and is an important facility for writing readable code. This document is to be considered as supplemental explanatory document for FCD NB comments FI-1, FI-2 and FI-3. Triviality concerns are left out of this document, focusing just on the avoidance of having to specify defaulting on the definition outside the class body. The examples illustrate cases where the special member functions are non-public, but the same arguments (readability, code that's easy to write) also apply to being able to default virtual or explicit special member functions on their first declaration.

I wish to thank Daniel Krügler for performing a sanity check for an earlier version of this paper, and Lawrence Crowl for reviewing those papers and providing feedback and suggestions for improvement.

The problem explained

I think it's problematic that non-public special member functions can't be defaulted on their first declaration. The defaulting is something that people do when they want to mark a special

member function defaulted regardless of the access of said member function, and people want to do it in a simple and concise manner, without having to write boiler-plate code. Forbidding defaulting on first declaration seems to restrict the design vocabulary considerably.

I shall repeat the use cases provided on the mailing list. Let's first consider a case where a copy constructor is protected and defaulted:

```
struct B
{
protected:
   B(const B&) = default;
};

struct D : B
{
};
```

The use case for this defaulting would be that the user is attempting to forbid slicing copies from D to B. Having to write the defaulting outside the class is tedious:

```
struct B
{
protected:
   B(const B&);
};
B::B(const B&) = default;
```

When the design evolves, the user decides that instances of the base are harmful, so she modifies the base thus:

```
struct B
{
protected:
    B() = default;
    B(const B&) = default;
};
```

Having to write the defaulting outside is once again tedious:

```
struct B
{
protected:
    B();
    B(const B&);
};
B::B(const B&) = default;
B::B() = default;
```

For other subobjects besides base classes, let's consider the following case where private constructor and copy constructor are defaulted:

```
struct part
```

```
{
friend class aggregate;
private:
    part() = default;
    part(const part&) = default;
};

struct aggregate
{
    part x;
};
```

The defaulting is equally tedious as in previous examples. Templates make the situation worse:

```
template<class T> struct TX
{
    template<class U> struct TY
    {
    protected:
        TY();
    };
protected:
    TX();
};
template<class T> TX<T>::TX() = default;
template<class T>
template<class U>
TX<T>::TY<U>::TY() = default;
```

In comparison, if the defaulting is allowed on the first declaration, the example is arguably much more readable:

```
template<class T> struct TX
{
    template<class U> struct TY
    {
    protected:
        TY() = default;
    };
protected:
    TX() = default;
};
```

According to my surveys, users understand that defaulting possibly affects e.g. triviality. Regardless of that, they want to do the defaulting concisely, and they don't want to be forced to write it outside the class. It seems to be a relatively common case to mark a special member function protected or private, default it, and move on. Forcing the defaulting to be done outside the class definition is cumbersome.

I consider forbidding defaulting non-public special member functions on first declaration to be overkill for what it's seemingly trying to achieve. Forcing users to write code outside of class definitions for defaulting, when they are never forced to do that for any other reason, is difficult to explain, difficult to teach, and is going to lead to people having to write error-prone out-of-line

declarations. For these reasons, Finnish NB comments FI-1/FI-2/FI-3 propose removing the restrictions for defaulting special member functions on their first declaration.

Triviality of non-public special member functions defaulted on first declaration

Date: 2010-06-17

Version: FCD14882: FI-4, FI-5

Authors: Ville Voutilainen <ville.voutilainen@gmail.com>

Abstract

The resolution of Core Issue 906 forbids defaulting a non-public special member function on its first declaration. I believe this resolution to be incorrect, and this document explains why. This document proposes that defaulting a non-public special member function on its first declaration should not be considered as something that affects triviality. Defaulting should retain triviality, or retain the lack of triviality, and people shouldn't expect defaulting to bring forth triviality. Furthermore, non-public special member functions defaulted on first declaration should not be considered user-provided. This document is to be considered as supplemental explanatory comment for FCD NB comments FI-4 and FI-5.

I wish to thank Daniel Krügler for performing a sanity check for this paper, and Lawrence Crowl for reviewing the paper and providing feedback and suggestions for improvement.

Background

N2346 states that "A special member function is user-provided if it is user-declared and not explicitly defaulted on its first declaration." Further, N3000 states in 12.8 Copying class objects [class.copy] p6 that "A copy constructor for class X is trivial if it is not user-provided". Therefore, I would assume that if it's allowed to default a non-public copy constructor on its first declaration, a non-public copy constructor could be trivial. For constructors, 12.1 Constructors

[class.ctor] p5 states that "A default constructor is trivial if it is not user-provided". Therefore, I'd assume that if it's allowed to default a non-public constructor on its first declaration, a non-public default constructor could be trivial.

The current status quo is that in order to be defaulted on the first declaration, a special member function must be public. During the discussion of Core Issue 906 in Santa Cruz it was suggested by John Spicer that triviality should be made more explicit by forcing the user to do the defaulting after the first declaration, assuming that defaulting a non-public member function will remove triviality because such a function would be considered user-provided.

Jason Merrill and I were of the opinion that access should not be the deciding factor when allowing defaulting on the first declaration, but it should be decided on whether the defaulted special member function is considered user-provided. It seems to me that N2346 strongly suggests that a defaulted special member function should not be considered user-provided, and thus non-public member functions should not affect triviality if they are defaulted.

Jens Maurer thought in message 15490 that a base class with a protected copy constructor makes a derived class not trivially copyable, because it has a subobject that is not trivially copyable. Mike Miller asked for a use case, which was provided later in reflector discussions, in message 15492 on the core reflector.

The problem, cases that should be allowed and trivial

As a simple rule, if an implicitly declared, or explicitly defaulted special member function is (indirectly or directly) accessible, I'd expect triviality to follow. All cases where triviality would be lost would be cases where access is lost as well and offending code becomes ill-formed, so I don't see cases where triviality would be subtly lost. Cases that make classes non-trivial today (adding a non-trivial subobject) would still be non-trivial even with defaulted non-public special member functions.

I think it's problematic that non-public special member functions can't be defaulted on their first declaration. The defaulting is something that people do when they want to retain triviality if possible. The access control is orthogonal to that, because access control in this case is a design tool. Forbidding defaulting on first declaration seems to restrict the design vocabulary considerably.

Furthermore, regarding the concern by Jens Maurer, a base class with a protected copy constructor is not CopyConstructible to begin with; a derived class can be, however, if the derived class has a public copy constructor. My take on that is that a derived class can be trivially copyable, even if the base class alone would not be, as long as the copy constructor of the base is accessible to the derived class, and as long as neither copy constructor is user-provided.

This applies to all subobjects, a class aggregating a subobject having a private copy constructor can still be trivially copyable if the aggregating class is the friend of the aggregated class.

I shall repeat the use cases provided on the mailing list. Let's first consider a case where a copy constructor is protected and defaulted:

```
struct B
{
protected:
   B(const B&) = default;
};

struct D : B
{
};
```

The use case for this defaulting would be that the user is attempting to forbid slicing copies from D to B, but still retaining triviality.

When the design evolves, the user decides that instances of the base are harmful, so she modifies the base thus:

```
struct B
{
protected:
    B() = default;
    B(const B&) = default;
};
```

There's still no reason for the user to assume that D wouldn't be trivially copyable, because there's no user-provided function in sight. D should also be trivially constructible, for the same reason.

For other subobjects besides base classes, let's consider the following:

```
struct part
{
friend class aggregate;
private:
    part() = default;
    part(const part&) = default;
};

struct aggregate
{
    part x;
};
```

I'd expect it to be perfectly reasonable for the user to assume aggregate to be trivially copyable (and trivially constructible).

If we apply the previous rule, even the following should be trivial:

```
struct B
{
protected:
    B() = default;
    B(const B&) = default;
};

struct D : B
{
protected:
    D() = default;
    D(const D&) = default;
};

struct DD : D
{
};
```

The subobjects of DD have accessible (to DD) special member functions, which are defaulted. Further, the subobjects of D have accessible (to D) special member functions, which are defaulted. This should result in DD being both trivially constructible and trivially copyable. The rule seems generic, and implementations need to walk the subobject chains anyway to check for access violations and triviality, so I'd think it's feasible to take the defaulting into account when computing whether a given class is trivial. B and D aren't trivial in isolation, and I don't propose changing that. The crux of the matter is allowing examples like DD to be trivial. Previously this was practically impossible because the only trivial things were implicitly declared special member functions. When defaulting is added, it would be good to allow triviality for cases where user-declared (but not user-provided) special member functions can retain triviality.

Some cases that should be allowed but aren't trivial

Let's consider the following example:

```
struct base
{
    std::shared_ptr<int> member;
};
```

The implicitly declared copy constructor will not be trivial, since shared_ptr's copy constructor isn't. Now, if this is a base class that people don't want to have instances of, we may write

```
struct base
{
    std::shared_ptr<int> member;
protected:
    base() = default;
```

```
    base(const base&) = default;
};
```

This shouldn't affect the triviality in any way. Yet users expect it to be allowed. While destructors work just fine with an empty definition, copy constructors don't. Thus the brevity of being able to default on first declaration seems superior to the alternatives. Consider the following:

```
struct base
{
    std::shared_ptr<int> member;
protected:
    base(const base&) {} // doesn't work at all!
};
```

Another, correct attempt would be this:

```
struct base
{
    std::shared_ptr<int> member;
protected:
    base(const base&);
};

base::base(const base&) = default; // tedious to write outside the
class definition
```

Conclusion

As suggested by this paper, I don't think a protected or even private special member function should result in loss of triviality, and most importantly, such a function should definitely not be considered user-provided just because it has different access than the implicitly declared function would have.

I'm inclined to think it's a quality-of-implementation issue to diagnose the lack of triviality, whether in the case of implicitly declared special member functions or in the case of special member functions defaulted on first declaration. I consider forbidding defaulting non-public special member functions on first declaration to be overkill for what it's seemingly trying to achieve, and I consider it grossly incorrect from the design vocabulary point of view.