ISO/IEC JTC1 SC22 WG14 N1040

Date: 2003-11-07

Reference Number of document **ISO/IEC DTR 19769**

Committee identification: ISO/IEC JTC1 SC22 WG14

SC22 Secretariat: ANSI

Information Technology —

Programming languages, their environments and system software inferfaces —

Extensions for the programming language C to support new character data types —

Warning

This document is an ISO/IEC draft Technical Report. It is not an ISO/IEC International Technical Report. It is distributed for review and comment. It is subject to change without notice and shall not be referred to as an International Technical Report or International Standard.

Recipients of this draft are invited to submit, with their comments, notification of any relevant patent rights of which they are aware and to provide supporting documentation.

Document type: Technical Report Type 2
Document subtype: n/a
Document stage: (4) Approval
Document language: E

Contents

1 Introduction

The C language has evolved over the last decades, various code pages and multibyte libraries have been introduced, and extended character set support has been introduced; however, the support for extended character data types in the C language is still limited. Today, the introduction and the success of the Unicode/ISO10646 standard and of its implementation in modern computer languages create ever increasing demands on the C language to give Unicode/ISO10646 better support. This paper addresses the introduction of new extended character data types in the C language in order to support future character encoding forms, including Unicode/ISO10646.

The Unicode standard supports 3 encoding forms:

- UTF-8
- UTF-16
- UTF-32

Each encoding form has advantages and disadvantages, so the choice of the encoding form should be left to the application. Currently, some C applications implement UTF-8 using **`char`** type, UTF-16 using **`unsigned short`** or **`wchar_t`**, and UTF-32 using **`unsigned long`** or **`wchar_t`**. The current situation, however, faces the following major problems:

- The size of **`wchar_t`** is implementation defined. While **`wchar_t`** offers a form of platform portability for C applications, Unicode offers the possibility to write platform independent applications using a platform independent data format.
- There is no string literal for 16- or 32-bit based integer types, but the Unicode encoding forms require string literals.

It is sensible to give all the Unicode encoding forms appropriate data type support. UTF-8 is normally considered as the preferred multibyte encoding, for sequences of one or more elements of type **`char`**. This paper suggests the implementation of 16 and 32 bit character data types: **`char16_t`** and **`char32_t`**. The new data types guarantee program portability through clearly defined character widths. The encoding of the new data types should be as generic as possible in order to support not only Unicode but also other character encodings.

It is generally desirable that C applications process entire strings at once rather than process individual characters in isolation. This paper does not specify the detail of library functions for the new data types, except one set of character conversion functions.

2 General

2.1 Scope

This Technical Report specifies two extended character data types as an extension to the programming language C, specified by the international standard ISO/IEC 9899:1999.

2.2 References

The following standards contain provisions which, through reference in this text, constitute provisions of this Technical Report. For dated references, subsequent amendments to, or revisions of, any of these publications do not apply. However, parties to agreements based on this Technical Report are encouraged to investigate the possibility of applying the most recent editions of the normative documents indicated below. For undated references, the latest edition of the normative document referred applies. Members of IEC and ISO maintain registers of current valid International Standards.

ISO/IEC 9899:1999, *Information technology – Programming languages, their environments an system software interfaces – Programming Languages – C*

ISO/IEC 10646-1:2000, *Universal multi-octet character set – UCS – Part 1 : Architecture and Basic Multilingual Plane*

ISO/IEC 10646-2: 2000, *Universal multi-octet character set – UCS – Part2 : CJK Unified, Ideographs Supplementary plane, General Scripts and Symbols Plane, General Purpose Plane*

3 The new typedefs

This Technical Report introduces the following two new typedefs, **char16_t** and **char32_t** :

```
typedef     T1      char16_t;
typedef     T2      char32_t;
```

where *T1* has the same type as **uint_least16_t** and *T2* has the same type as **uint_least32_t**.

The new typedefs guarantee certain widths for the data types, whereas the width of **wchar_t** is implementation defined. The data values are unsigned, while **char** and **wchar_t** could take signed values.

This Technical Report also introduces the new header:

<uchar.h>

The new typedefs, **char16_t** and **char32_t**, are defined in **<uchar.h>**

4 Encoding

C99 subclause 6.10.8 specifies that the value of the macro **__STDC_ISO_10646__** shall be "an integer constant of the form **yyyymmL** (for example, **199712L**), intended to indicate that values of type **wchar_t** are the coded representations of the characters defined by ISO/IEC 10646, along with all amendments and technical corrigenda as of the specified year and month." C99 subclause 6.4.5p5 specifies that wide string literals are initialized with a sequence of wide characters as defined by the **mbstowcs** function with an implementation-defined current locale. Analogous to this macro, this Technical Report introduces two new macros.

If the header **<uchar.h>** defines the macro **__STDC_UTF_16__**, values of type **char16_t** shall have UTF-16 encoding. This allows the use of UTF-16 in **char16_t** even when **wchar_t** uses a non-Unicode encoding. In certain cases the compile-time conversion to UTF-16 may be restricted to members of the basic character set and universal character names (\U*nnnnnnnn* and \u*nnnn*) because for these the conversion to UTF-16 is defined unambiguously.

If the header **<uchar.h>** defines the macro **__STDC_UTF_32__**, values of type **char32_t** shall have UTF-32 encoding.

If the header **<uchar.h>** does not define the macro **__STDC_UTF_16__**, the encoding of **char16_t** is implementation defined. Similarly, if the header **<uchar.h>** does not define the macro **__STDC_UTF_32__**, the encoding of **char32_t** is implementation defined.

An implementation may define other macros to indicate a different encoding.

5 String literals and character constants

5.1 String literals and character constants notations

The notations for string literals and character constants for **char16_t** are defined analogous to the wide character string literals and wide character constants:

u"*s-char-sequence***"**

denotes a **char16_t** type string literal and initializes an array of **char16_t**. The corresponding character constant is denoted by

u'*c-char-sequence***'**

and has the type **char16_t**. Likewise, the string literal and character constant for **char32_t** are,

U"*s-char-sequence***"** and

U'*c-char-sequence***'**.

5.2 The string concatenation

String literals with the new format can be concatenated. If both strings have the same format, the resulting concatenated string has that format. If one string has no prefix, it is treated as a string of the same format as the other operand. (**u"**str**"** and **U"**str**")** Any other concatenations are implementation-defined (they might or might not be supported). Here are some examples of valid concatenations:

```
u"a" u"b"→ u"ab"   U"a" U"b"→ U"ab"   L"a" L"b"→ L"ab"
u"a"  "b"→ u"ab"   U"a"  "b"→ U"ab"   L"a"  "b"→ L"ab"
 "a" u"b"→ u"ab"    "a" U"b"→ U"ab"    "a" L"b"→ L"ab"
```

6 Library functions

Speaking in general, it is desirable to free the C applications from character-based operations and encourage string-based operations. Details of the library for the new character data types are left to future enhancements of the C standard. This Technical Report specifies merely the four minimum character conversions among 3 character data types: **`char, char16_t`** and **`char32_t`**.

6.1 The `mbrtoc16` function

Synopsis

```
#include <uchar.h>
size_t mbrtoc16(char16_t * restrict pc16,
   const char * restrict s,
   size_t n,
   mbstate_t * restrict ps);
```

Description

If **`s`** is a null pointer, the **`mbrtoc16`** function is equivalent to the call:

```
mbrtoc16(NULL, "", 1, ps)
```

In this case, the values of the parameters **`pc16`** and **`n`** are ignored.

If **`s`** is not a null pointer, the **`mbrtoc16`** function inspects at most **`n`** bytes beginning with the byte pointed to by **`s`** to determine the number of bytes needed to complete the next multibyte character (including any shift sequences). If the function determines that the next multibyte character is complete and valid, it determines the value of the corresponding wide character and then, if **`pc16`** is not a null pointer, stores that value in the object pointed to by **`pc16`**. If the corresponding wide character is the null wide character, the resulting state described is the initial conversion state.

Returns

The **`mbrtoc16`** function returns the first of the following that applies (given the current conversion state):

`0` if the next **`n`** or fewer bytes complete the multibyte character that corresponds to the null wide character (which is the value stored).

between **`1`** *and* **`n`** *inclusive*

	if the next **n** or fewer bytes complete a valid multibyte character (which is the value stored); the value returned is the number of bytes that complete the multibyte character.
(size_t)(-3)	if the multibyte sequence converted more than one corresponding **char16_t** character and not all these characters have yet been stored; the next character in the sequence has now been stored and no bytes from the input have been consumed by this call.
(size_t)(-2)	if the next **n** bytes contribute to an incomplete (but potentially valid) multibyte character, and all **n** bytes have been processed (no value is stored).[1]
(size_t)(-1)	if an encoding error occurs, in which case the next **n** or fewer bytes do not contribute to a complete and valid multibyte character (no value is stored); the value of the macro **EILSEQ** is stored in **errno**, and the conversion state is unspecified.

6.2 The c16rtomb function

Synopsis

```
#include <uchar.h>
size_t c16rtomb(char * restrict s,
   char16_t c16,
   mbstate_t * restrict ps);
```

Description

If **s** is a null pointer, the **c16rtomb** function is equivalent to the call **c16rtomb(buf, L'\0', ps)** where **buf** is an internal buffer. If **s** is not a null pointer, the **c16rtomb** function determines the number of bytes needed to represent the multibyte character that corresponds to the wide character given by **c16** (including any shift sequences), and stores the multibyte character representation in the array whose first element is pointed to by **s**. At most **MB_CUR_MAX** bytes are stored. If **c16** is a null wide character, a null byte is stored, preceded by any shift sequence needed to restore the initial shift state; the resulting state described is the initial conversion state.

Returns

The **c16rtomb** function returns the number of bytes stored in the array object; this may be 0 (including any shift sequences). When **c16** is not a valid wide character, an

[1] When **n** has at least the value of the **MB_CUR_MAX** macro, this case can only occur if **s** points at a sequence of redundant shift sequences (for implementations with state-dependent encodings).

encoding error occurs: the function stores the value of the macro **EILSEQ** in **errno** and returns **(size_t)(-1)**; the conversion state is unspecified.

6.3 The mbrtoc32 function

Synopsis

```
#include <uchar.h>
size_t mbrtoc32(char32_t * restrict pc32,
   const char * restrict s,
   size_t n,
   mbstate_t * restrict ps);
```

Description

If **s** is a null pointer, the **mbrtoc32** function is equivalent to the call:

```
mbrtoc32(NULL, "", 1, ps)
```

In this case, the values of the parameters **pc32** and **n** are ignored.

If **s** is not a null pointer, the **mbrtoc32** function inspects at most **n** bytes beginning with the byte pointed to by **s** to determine the number of bytes needed to complete the next multibyte character (including any shift sequences). If the function determines that the next multibyte character is complete and valid, it determines the value of the corresponding wide character and then, if **pc32** is not a null pointer, stores that value in the object pointed to by **pc32**. If the corresponding wide character is the null wide character, the resulting state described is the initial conversion state.

Returns

The **mbrtoc32** function returns the first of the following that applies (given the current conversion state):

0 if the next **n** or fewer bytes complete the multibyte character that corresponds to the null wide character (which is the value stored).

between **1** *and* **n** *inclusive*
if the next **n** or fewer bytes complete a valid multibyte character (which is the value stored); the value returned is the number of bytes that complete the multibyte character.

(size_t)(-3) if the multibyte sequence converted more than one corresponding **char32_t** character and not all these characters have yet been stored; the next character in the sequence has now been stored and no bytes from the input have been consumed by this call.

`(size_t)(-2)`	if the next **n** bytes contribute to an incomplete (but potentially valid) multibyte character, and all **n** bytes have been processed (no value is stored).[2]
`(size_t)(-1)`	if an encoding error occurs, in which case the next **n** or fewer bytes do not contribute to a complete and valid multibyte character (no value is stored); the value of the macro **EILSEQ** is stored in **errno**, and the conversion state is unspecified.

6.4 The `c32rtomb` function

Synopsis

```
#include <uchar.h>
size_t c32rtomb(char * restrict s,
   char32_t c32,
   mbstate_t * restrict ps);
```

Description

If **s** is a null pointer, the **c32rtomb** function is equivalent to the call **c32rtomb(buf, L'\0', ps)** where **buf** is an internal buffer. If **s** is not a null pointer, the **c32rtomb** function determines the number of bytes needed to represent the multibyte character that corresponds to the wide character given by **c32** (including any shift sequences), and stores the multibyte character representation in the array whose first element is pointed to by **s**. At most **MB_CUR_MAX** bytes are stored. If **c32** is a null wide character, a null byte is stored, preceded by any shift sequence needed to restore the initial shift state; the resulting state described is the initial conversion state.

Returns

The **c32rtomb** function returns the number of bytes stored in the array object; this may be 0 (including any shift sequences). When **c32** is not a valid wide character, an encoding error occurs: the function stores the value of the macro **EILSEQ** in **errno** and returns **(size_t)(-1)**; the conversion state is unspecified.

[2] When **n** has at least the value of the **MB_CUR_MAX** macro, this case can only occur if **s** points at a sequence of redundant shift sequences (for implementations with state-dependent encodings).

7 ANNEX A Unicode encoding forms: UTF-16, UTF-32

See Section 2.5 "Encoding Forms" in *The Unicode Standard*, Version 4.0.0, defined by: *The Unicode Standard, Version 4.0* (Boston, MA, Addison-Wesley, 2003. ISBN 0-321-18578-1)

Online Edition

http://www.unicode.org/versions/Unicode4.0.0/

See also Annex C of ISO10646-1.

Online Edition

http://www.dkuug.dk/JTC1/SC2/WG2/docs/n2005/n2005-2.doc