WG14 N1035

L2/04-091

Disposition of Comments received during PDTR19769 ballot (SC 22 N3579)

Japan

Comment JP-1:

The name of Macro "__STDC_UTF_16" at the second paragraph of chapter 4 Encoding should be changed to "__STDC_UTF_16__".

Response JP-1: Accepted.

Netherlands

Comment NL-1:

Care should be taken to ensure that the emphasis of the proposed extension is on the support of ISO/IEC 10646, rather than on the support of the Unicode standards. The 1st paragraph of the introduction should be modified to reflect this.

Response NL-1: Accepted, answered by changes made in the text.

United States

Comment US-0 (1) :

The proposal should also include adding a -3 return code from mbrtowc, as proposed by Clive D.W. Feather. This permits general N-to-M mappings, not just 1-to-N and N-to-1.

Response US-0 (1): Accepted, answered by changes made in the text.

Comment US-1: Introduction

The sentence "The C language has matured over the last decades, yet the character concept has remained stable."

is troubling. Surely one would expect that as a language matures, its fundamental concepts would remain stable.

Response US-1: Accepted, answered by changes made in the text.

Comment US-2: Introduction

The sentence

Various code pages and multibyte libraries have been introduced in the past; however, the character data type in the C language has remained 8 bit based.

presents several problems. First, there are today C implementations for machines with 36-bit words. The C standard requires characters for such machines to be at least 9 bits (see Sections 5.2.4.2.1 and 6.2.6.1). Perhaps the text should say they are "byte based" instead of "8 bit based." Even then, there is the problem that the concrete definition of "character" given in the C standard (see Section 3.7.1) specifies single-byte characters. I fear that the different meanings of the word "character" in the standard and this TR are liable to lead to confusion and defect reports. The TR might be able to avoid this confusion by pointing out which uses of the word "character" in the C standard retain the old meaning, and which take on the new meaning. The mention of code pages seems irrelevant to the point being made. All uses of code pages I have seen have been 8 bit based.

Response US-2: Accepted, answered by changes made in the text.

Comment US-3: 2.1 Scope

The statement that the TR "specifies two character data types" conflicts with the existing C standard. The TR should either include edits to Sections **3.7** and **6.2.5** to resolve the conflicts, or it should use new terminology to refer to these new types.

Response US-3: Accepted, answered by changes made in the text.

Comment US-5: 3 The new typedefs

I assume there is nothing special about the typedefs given beyond provide more convenient names for referring to the integer types uint_least16_t and uint_least32_t. If I am wrong and special properties are tied to the typedefs, are those properties propagated through further typedefs. For example, if a user provides a typedef

```
typedef char16_t c16_t;
```

will c16_t have all the properties of char16_t.

Response US-5: Not accepted.

It is typedef and not macro. Therefore c16_t has the same type as char16_t.

Comment US-6: 4 Encoding

What does the statement
If the macro __STDC_UTF_16, the type char16_t shall have the UTF-16 encoding.

mean? According to Section C.1 of ISO/IEC 10646-1:2000, there are 16-bit values in UTF-16 that must be paired with other 16-bit values to be valid in UTF-16. Can a scalar of type char16_t take on one of those values when the macro __STDC_UTF_16 is defined? Can a value that does not correspond to a Unicode or ISO/IEC 10646 character be assigned to a variable of type char16_t?

Response US-6: Not accepted.

These types are synonyms to existing basic data types and they have the same restrictions as these types. The programmers are obliged to call library functions with only valid encodings. The wording of TR remains as it is.

Comment US-7: 4 Encoding

The statement In the absence of the mentioned macros, an implementation may define other macro's (sic) to indicate a different encoding; ...

implies but does not state that in the presence of those macros an implementation shall not define **other macros** to indicate a different encoding. Isn't such a statement void? As long as the macro chosen is a name reserved to the implementation, no strictly conforming program could tell such a macro had been provided.

Response US-7: Accepted.

Agreed with the intent of the comment and TR text has been changed accordingly.

Comment US-8: 6 Library functions

Use of the functions specified in the TR could be facilitated by providing a feature test macro for the presence of those functions.

Response US-8: Not accepted.

There is no need for feature test macro because all new features are supplied in a new header.

Comment US-9: 6 Library functions

Does the use of char16_t and char32_t place any restriction on the integer values passed to and returned from the functions beyond the restrictions that would apply if uint_least16_t and uint_least32_t had been used instead?

Response US-9: Not accepted. See Response US-6.

Comments US-10: 7 ANNEX A Unicode encoding forms: UTF-16, UTF-32

The definitions of UTF-16 and UTF-32 provided in *The Unicode Standard,* version 3.0 are incomplete in and of themselves. Additional information available online and on the CD provided with the book are needed. The definitions provided in ISO/IEC 10646, on the other hand, are complete in and of themselves. Since ISO/IEC 10646 are the normative references listed in the TR, Section 7 ANNEX A of the TR should reference the definitions given in ISO/IEC 10646.

Response US-10: Not accepted.

The problem in The Unicode Standard Version 3.0 has been resolved in the Unicode Standard 4.0. The reference in ANNEX A is updated. The reference to ISO/IEC 10646 is also now revised.

Comment US-11 (1.): Do we need to say something about uint_least16_t and uint_least32_t not being defined by <uchar.h>? Would it be better to define char16_t and char32_t in terms of the base types that uint_least*_t are defined in?

Response US-11 (1.): Accepted, answered by changes made in the text.

Comment US-12 (__x__): general

In 1, the statement that "the character data type in the C language has remained 8 bit based" is misleading at best. It would be more correct to say that the character data type *in most C implementations* has remained 8 bit based. Likewise, the statement that "wchar_t does not offer platform portability for C applications" is also misleading; it may not offer the same kind of platform portability that Unicode does, but it does offer platform portability of a different kind-that is its sole reason for existing.

Response US-12: Accepted, answered by changes made in the text.

Comment US-Editorial-01 : editorial

When types are mentioned in the running text, they generally appear in italics as opposed to appearing in courier bold as in the C standard. Since this TR is in some sense an addendum to the C standard, it may be better to use the same style.

Response US-Editorial-01: Accepted.

Comment US- Editorial-01: In the table of contents, the function names should be bold.

Response US-Editorial-01: Accepted.

Comment US- Editorial-02:

In 1, the initial sentence of the final paragraph would read better as: It is generally desirable that C applications process entire strings at once rather than processing individual characters in isolation.

Response US-Editorial-02: Accepted.

Comment US- Editorial-03:

In 2.2, the correct title of ISO/IEC 9899:1999 is "Programming languages * C".

Response US- Editorial-03: Accepted.

Comment US- Editorial-04:

In 3, the header "uchar.h" should be referred to as "<uchar.h>" for consistency with the C standard. Also, the final reference is in the wrong font.

Response US- Editorial-04: Accepted.

Comment US- Editorial-05:

In 4, it is not clear whether the user is to define the macros to influence the implementation or whether the implementation is to define the macros to document its behavior.

Response US- Editorial-05: Accepted, answered by changes made in the text.

Comment US- Editorial-06:

Also, there should be restrictions on the names of any other macros the implementation may define to prevent encroaching on the user's namespace. The typography makes it difficult to see that the underscores at the beginning and end of the macro names are doubled; a thin space should be added between them.

Response US- Editorial-06: Accepted.

Comment US- Editorial-07:

In the first paragraph, the form "yyyymmL" and the example "199712L" should be set in courier bold, as should the name of the "mbstowcs" function. Also, "Analogue" should be "Analogous".

Response US- Editorial-07: Accepted.

Comment US- Editorial-08:

In the second paragraph, "__STDC_UTF_16__" is missing its trailing underscores and the type "char16_t" appears in the wrong font, as does "char32_t" in the third paragraph.

Response US- Editorial-08: Accepted.

Comment US- Editorial-09:

In the fourth paragraph, "Analogically" should be "Analogously" (or, better still, "Similarly").

Response US- Editorial-09: Accepted

Comment US- Editorial-10:

In the fifth paragraph, "macro's" should be "macros".

Response US- Editorial-10: Accepted

Comment US- Editorial-11:

In 5.1, "analogue" should be "analogous". Also, the type "char16_t" appears in the wrong font (twice), the initial "U" and "u" in the string literal formats are in the wrong font, as are the closing quote marks (the leading quote marks may also be in the wrong font, it's hard to tell for sure). The same problem appears in 5.2 along with the initial "L" for wide string literals. And all of the examples are in the wrong font.

Response US- Editorial-11: Accepted.

Comment US- Editorial-12:

I am unable to make sense of the statement in 5.2:

... when adjacent string literals of the same format are concatenated the result is widened to the representation of the other string literal also if one of the adjacent literals is a "narrow" string.

Surely when string literals of the same format are concatenated, there is no need to widen the result, only when a "narrow" string is involved.

Response US- Editorial-12: Accepted, answered by changes made in the text.

Comment US- Editorial-13:

In 6, "the C applications" should be "C applications" and "the future enhancements" should be "future enhancements". Also, normal publishing style is to use words for small numbers like four and three rather than numerals.

Response US- Editorial-13: Accepted.

Comment US- Editorial-13:

In the various subclauses of 6, the parameter "s" frequently appears in the running text (and footnotes) in the wrong font. No semantics are given for the "ps" parameters. The "state" is sometimes referred to as just "state", sometimes as "shift state", and sometimes as "conversion state"; the terminology should be consistent. Also, in the "Returns" sections, it would be clearer to say "the resulting conversion state" rather than just "the conversion state". In the "Returns" sections of 6.1 and 6.3, "(size_t)(-2)" and "(size_t)(-1)" appear in the wrong font. In the "Returns" section of 6.1 in the text for "(size_t)(-1)", the parameter "n" appears in the wrong font.

Response US- Editorial-13: Accepted.

Comment US- Editorial-14:

On page 4, the first usage of __STDC_UTF_16__ is missing the trailing

Response US- Editorial-14: Accepted.

Comment US-0-1:

I did not understand until I read and reread the proposed draft TR several times that the types char16_t and char32_t are used solely to represent the encodings UTF-16 and UTF-32. My initial reading was that they represented Unicode characters or the characters in ISO/IEC 10646. With that understanding came the realization that the conversion functions provided are not conversions between wide characters and encodings of those wide characters, as are the similarly named functions in the C standard. Rather, they are conversions between encodings of characters.

Response US-0-1: Accepted, answered by the changes made in the text.

Comment US-0-2:

The nature of the extensions presented in the TR should be clearly stated in the introduction.

Response US-0-1: Accepted, answered by the changes made in the text.

Comment US-4-1:

4. 2.2 References
Since all of the references given are dated, the reference to undated reference should be elided.

Response US-4-1: Accepted, answered by the changes made in the text.

Comment US-Not-Titled:

The first paragraph of the TR's section 1, Introduction, makes some misstatements about the previous state of the "character concept" and platform portability in the C language. It should instead make the following argument:

(1) The C standard does not *require* the "char" type to have an 8-bit width; however, because this type is identified with the minimum addressable storage unit, most implementations have chosen to use 8 bits for "char", making it unsuitable for encoding large character sets.

(2) The type intended by the C standard for encoding large character sets is "wchar_t", but this is not *required* to have a width more than 8 bits, nor to use Unicode encoding.

(3) wchar_t does provide platform portability where details of character encoding are not essential to the function of the program.

(4) Because Unicode needs multiple encoding forms, the existing single form of "wide character" specified in the C standard is insufficient.

Response US-Not-Titled: Accepted, answered by the changes made in the text.