WG14 N3105

Title:	**Issues with CFP response to NB comments – N3101 update**
Author, affiliation:	**C FP group**
Date:	**2023-02-06**
Reference:	**N3054, N3067, N3082, N3101**

This note updates N3101 to respond to [SC22WG14.23086].

This note follows up on requests from WG14 regarding CFP response (N3082) to NB comments USA42-169, GB-286 and GB-287 (N3067).

USA42-169

The comment in N3067 is "7.12.11.4 (nexttoward): Should a 'Returns' paragraph be added?" and the proposed solution is "If added, it should be similar to the 'nextafter' one." The CFP response in N3082 is "Add the Returns paragraph as suggested. Similar 7.24.1.3 has a Returns section." Since 7.24.1.3 doesn't seem similar, WG14 requested clarification.

The only purpose of the reference to 7.24.1.3 was to show that in similar cases–where one function is described as equivalent to another function–a Returns section is provided for both functions. We support US42-169 and adding a Returns section (similar to nextafter) in 7.12.11.4 as proposed in the comment, specifically:

Returns
3 The **`nexttoward`** functions return the next representable value in the specified format after **`x`** in the direction of **`y`**.

GB-286

WG14 requested CFP provide specification for **`wcstof`***N and* **`wcstod`***N functions. To add the specification, make the changes highlighted in yellow below.*

H.12.2 String to floating

1 This subclause expands 7.24.1.5, 7.31.4.1.2, 7.24.1.6, and 7.31.4.1.3 to also include functions for the interchange and extended floating types. It adds to the synopsis in 7.24.1.5 the prototypes

```
_FloatN strtofN(const char * restrict nptr, char ** restrict endptr);
_FloatNx strtofNx(const char * restrict nptr, char ** restrict endptr);
```

It adds to the synopsis in 7.31.4.1.2 the prototypes

```
_FloatN wcstofN(const wchar_t * restrict nptr,
      wchar_t ** restrict endptr);
_FloatNx wcstofNx(const wchar_t * restrict nptr,
      wchar_t ** restrict endptr);
```

It encompasses the prototypes in 7.24.1.6 by replacing them with

```
_DecimalN strtodN(const char * restrict nptr, char ** restrict endptr);
_DecimalNx strtodNx(const char * restrict nptr,
      char ** restrict endptr);
```

It encompasses the prototypes in 7.31.4.1.3 by replacing them with

```
_DecimalN wcstodN(const wchar_t * restrict nptr,
      wchar_t ** restrict endptr);
_DecimalNx wcstodNx(const wchar_t * restrict nptr,
      wchar_t ** restrict endptr);
```

2 The descriptions and returns for the added functions are analogous to the ones in 7.24.1.5, 7.31.4.1.2, 7.24.1.6 and 7.31.4.1.3.

Paragraph 3 will need to be changed too. See further below.

Considering the principle behind GB-286, specification for wide character versions of the string to encoding functions should also be added. To add it, make the changes highlighted in yellow below.

H.12.4 String to encoding

1 An implementation shall declare the **strtoencf***N* and **wcstoencf***N* functions for each *N* equal to the width of a supported IEC 60559 arithmetic or non-arithmetic binary interchange format. An implementation shall declare the **strtoencdecd***N*, **strtoencbind***N*, **wcstoencdecd***N* and **wcstoencbind***N* functions for each *N* equal to the width of a supported IEC 60559 arithmetic or non-arithmetic decimal interchange format.

H.12.4.1 The strtoencf*N* functions

Synopsis

1
```
#define __STDC_WANT_IEC_60559_TYPES_EXT__
#include <stdlib.h>
void strtoencfN(unsigned char encptr[restrict static N/8],
      const char * restrict nptr, char ** restrict endptr);
```

Description

2 The **strtoencf***N* functions are similar to the **strtof***N* functions, except they store an IEC 60559 encoding of the result as an *N*/8 element array in the object pointed to by **encptr**. The order of bytes in the array follows the endianness specified with **__STDC_ENDIAN_NATIVE__** (7.18.2).

Returns

3 These functions return no value.

H.12.4.2 The wcstoencf*N* functions

Synopsis

1
```
#define __STDC_WANT_IEC_60559_TYPES_EXT__
#include <wchar.h>
void wcstoencfN(unsigned char encptr[restrict static N/8],
      const wchar_t * restrict nptr, wchar_t ** restrict endptr);
```

Description

2 The **wcstoencf***N* functions are similar to the **wcstof***N* functions, except they store an IEC 60559 encoding of the result as an *N*/8 element array in the object pointed to by **encptr**. The order of bytes in the array follows the endianness specified with **__STDC_ENDIAN_NATIVE__** (7.18.2).

Returns

3 These functions return no value.

H.12.4.3 The strtoencdecd*N* and strtoencbind*N* functions

Synopsis

1
```
#define __STDC_WANT_IEC_60559_TYPES_EXT__
#include <stdlib.h>
void strtoencdecdN(unsigned char encptr[restrict static N/8],
      const char * restrict nptr, char ** restrict endptr);
void strtoencbindN(unsigned char encptr[restrict static N/8],
      const char * restrict nptr, char ** restrict endptr);
```

Description

2 The **strtoencdecd***N* and **strtoencbind***N* functions are similar to the **strtod***N* functions, except they store an IEC 60559 encoding of the result as an *N*/8 element array in the object pointed to by **encptr**. The **strtoencdecd***N* functions produce an encoding in the encoding scheme based on decimal encoding of the significand. The **strtoencbind***N* functions produce an encoding in the encoding scheme based on binary encoding of the significand. The order of bytes in the array follows the endianness specified with **__STDC_ENDIAN_NATIVE__** (7.18.2).

Returns

3 These functions return no value.

H.12.4.4 The wcstoencdecd*N* and wcstoencbind*N* functions

Synopsis

1
```
#define __STDC_WANT_IEC_60559_TYPES_EXT__
#include <wchar.h>
void wcstoencdecdN(unsigned char encptr[restrict static N/8],
      const wchar_t * restrict nptr, wchar_t ** restrict endptr);
void wcstoencbindN(unsigned char encptr[restrict static N/8],
      const wchar_t * restrict nptr, wchar_t ** restrict endptr);
```

Description

2 The **wcstoencdecd***N* and **wcstoencbind***N* functions are similar to the **wcstod***N* functions, except they store an IEC 60559 encoding of the result as an *N*/8 element array in the object pointed to by **encptr**. The **wcstoencdecd***N* functions produce an encoding in the encoding scheme based on decimal encoding of the significand. The **wcstoencbind***N* functions produce an encoding in the encoding scheme based on binary encoding of the significand. The order of bytes in the array follows the endianness specified with **__STDC_ENDIAN_NATIVE__** (7.18.2).

Returns

3 These functions return no value.

Changes to H.12.2 #3 depend on the resolution of GB-287. The following change assumes support for hexadecimal input is not moved to the main body of the standard. Changes for moving hexadecimal input to the main body of the standard allow removing #3 entirely.

3 For implementations that support both binary and decimal floating types and a (binary or decimal) non-arithmetic interchange format, the **strtod***N*, **strtod***N***x**, **wcstod***N* and **wcstod***N***x** functions (and hence the **strtoencdecd***N*, **strtoencbind***N*, **wcstoencdecd***N* and **wcstoencbind***N* functions in H.12.4.3 and H.12.4.4) shall accept subject sequences that have the form of hexadecimal floating numbers (excluding any digit separators (6.4.4.1)) and otherwise meet the requirements of subject sequences (7.24.1.6). Then the decimal results shall be correctly rounded if the subject sequence has at most *M* significant hexadecimal digits, where $M \geq \lceil (P - 1)/4 \rceil + 1$ is implementation-defined, and *P* is the maximum precision of the supported binary floating types and binary non-arithmetic formats. If all subject sequences of hexadecimal form are correctly rounded, *M* may be regarded as infinite. If the subject sequence has more than *M* significant hexadecimal digits, the implementation may first round to *M* significant hexadecimal digits according to the applicable rounding direction mode, signaling exceptions as though converting from a wider format, then correctly round the result of the shortened hexadecimal input to the result type.

GB-287

WG14 requested CFP provide specification to allow hexadecimal input into ***strtod***** in the main body of the standard. To add the specification, make the changes highlighted in yellow below. Note that with this change the* ***strtod****N functions and the* ***strtof****,* ***strtod****, and* ***strtold*** *functions accept the same input.*

There was considerable WG14 and CFP email discussion about whether ***strtod****N needs to accept hexadecimal input at all. It was noted that IEC 60559 does not require conversions of hexadecimal strings to decimal formats. See [Cfp-interest 2662] Re: GB-287 . However, IEC 60559 does require correctly rounded conversions between all supported formats (arithmetic and non-arithmetic). We haven't found any other as feasible way of converting from non-arithmetic binary formats to decimal formats without the hexadecimal support in* ***strtod****N. See H.4.3 and the example in H.12.2. Difficulties with other approaches are explained in [Cfp-interest 2657] Re: GB-287 .*

7.24.1.6 The strtod*N* functions

Synopsis

1
```
#include <stdlib.h>
#ifdef __STDC_IEC_60559_DFP__
_Decimal32 strtod32(const char * restrict nptr,
      char ** restrict endptr);
_Decimal64 strtod64(const char * restrict nptr,
      char ** restrict endptr);
_Decimal128 strtod128(const char * restrict nptr,
      char ** restrict endptr);
#endif
```

Description

2 The **strtod***N* functions convert the initial portion of the string pointed to by **nptr** to decimal floating type representation. First, they decompose the input string into three parts: an initial, possibly empty, sequence of white-space characters; a subject sequence resembling a floating constant or representing an infinity or NaN; and a final string of one or more unrecognized characters, including the terminating null character of the input string. Then, they attempt to convert the subject sequence to a floating-point number, and return the result.

3 The expected form of the subject sequence is an optional plus or minus sign, then one of the following:

— a nonempty sequence of decimal digits optionally containing a decimal-point character, then an optional exponent part as defined in 6.4.4.2, excluding any digit separators (6.4.4.1)

— a **0x** or **0X**, then a nonempty sequence of hexadecimal digits optionally containing a decimal-point character, then an optional binary exponent part as defined in 6.4.4.2, excluding any digit separators (6.4.4.1)

— **INF** or **INFINITY**, ignoring case

— **NAN** or **NAN (***d-char-sequence*$_{opt}$**)** , ignoring case in the **NAN** part, where:

d-char-sequence:
digit
nondigit
d-char-sequence digit
d-char-sequence nondigit

The subject sequence is defined as the longest initial subsequence of the input string, starting with the first non-white-space character, that is of the expected form. The subject sequence contains no characters if the input string is not of the expected form.

4 If the subject sequence has the expected form for a floating-point number, the sequence of characters starting with the first digit or the decimal-point character (whichever occurs first) is interpreted as a floating constant according to the rules of 6.4.4.2, ~~including correct rounding and determination of the coefficient c and the quantum exponent q, with the following exceptions:~~
~~— It is not a hexadecimal floating number.~~

~~— The decimal-point character is used in place of a period.~~

~~— If neither an exponent part nor a decimal-point character appears in a decimal floating-point number, an exponent part of the appropriate type with value zero is assumed to follow the last digit in the string.~~

except that the decimal-point character is used in place of a period, and that if neither an exponent part nor a decimal-point character appears in a decimal floating-point number, or if a binary exponent part does not appear in a hexadecimal floating-point number, an exponent part of the appropriate type with value zero is assumed to follow the last digit in the string. If the subject sequence begins with a minus sign, the sequence is interpreted as negated before rounding and the sign s is set to −1, else s is set to 1.

5 If the subject sequence has the expected form for a decimal floating-point number, the value resulting from the conversion is correctly rounded and the coefficient c and the quantum exponent q are determined by the rules in 6.4.4.2 for a decimal floating constant of decimal type.

6 If the subject sequence has the expected form for a hexadecimal floating-point number, the value resulting from the conversion is correctly rounded provided the subject sequence has at most M significant hexadecimal digits, where $M \geq \lceil (P - 1)/4 \rceil + 1$ is implementation-defined, and P is the maximum precision of the supported radix-2 floating types and binary non-arithmetic interchange formats*). If all subject sequences of hexadecimal form are correctly rounded, M may be regarded as infinite. If the subject sequence has more than M significant hexadecimal digits, the implementation may first round to M significant hexadecimal digits according to the applicable rounding direction mode, signaling exceptions as though converting from a wider format, then correctly round the result of the shortened hexadecimal input to the result type. The preferred quantum exponent for the result is 0 if the hexadecimal number is exactly represented in the decimal type; the preferred quantum exponent for the result is the least possible if the hexadecimal number is not exactly represented in the decimal type.

*) Non-arithmetic interchange formats are an optional feature in Annex H.

7 ~~If the subject sequence begins with a minus sign, the sequence is interpreted as negated before rounding and the sign s is set to −1, else s is set to 1.~~ A character sequence **INF** or **INFINITY** is …
…

Returns
10 The **strtod***N* functions return the ~~correctly rounded~~ converted value, if any. …

11 **EXAMPLE**
…
~~"**0x1.8p+4**" (+1, 0, 0), and a pointer to "x1.8p+4" is stored in the object pointed to by **endptr**, provided **endptr** is not a null pointer~~
"**0x1.8p+4**" (+1, 24, 0)

Moving hexadecimal input into the main body of the standard allows removing H.12.2 #3:

~~3 For implementations that support both binary and decimal floating types and a (binary or decimal) non-arithmetic interchange format, the **strtod***N* and **strtod***N***x** functions (and hence the **strtoencdecd***N* and **strtoencbind***N* functions in H.12.4.2) shall accept subject sequences that have the form of hexadecimal floating numbers (excluding any digit separators (6.4.4.1)) and otherwise meet the requirements of subject sequences (7.24.1.6). Then the decimal results shall be correctly rounded if the subject sequence has at most *M* significant hexadecimal digits, where $M \geq [(P - 1)/4] + 1$ is implementation-defined, and *P* is the maximum precision of the supported binary floating types and binary non-arithmetic formats. If all subject sequences of hexadecimal form are correctly rounded, *M* may be regarded as infinite. If the subject sequence has more than *M* significant hexadecimal digits, the implementation may first round to *M* significant hexadecimal digits according to the applicable rounding direction mode, signaling exceptions as though converting from a wider format, then correctly round the result of the shortened hexadecimal input to the result type.~~