Rewrite of C9*x* Subclause 7.4 `<inttypes.h>`

Douglas A. Gwyn

US Army Research Laboratory
Adelphi, MD

ABSTRACT

This edition reflects committee decisions made at the Menlo Park meeting; it supersedes Doc. No. WG14/N737 (J11/97-100) in the post-London mailing, Doc. No. WG14/N761 (J11/97-125) in the pre-Menlo Park mailing, and the preliminary ‘‘working draft’’ of Doc. No. WG14/N788 (J11/97-152) circulated at the Menlo Park meeting. With this introductory material removed, it is a ‘‘drop-in’’ replacement for the previous C9*x* (Draft 11-pre3) subclause 7.4 (source file `cl–704.mm`).

7.4 Integer types `<inttypes.h>`

The header `<inttypes.h>` defines sets of typedef names for integer types having specified widths, and defines corresponding sets of macros. It also defines macros that specify limits of integer types corresponding to typedef names defined in other standard headers, and declares four functions for converting numeric character strings to greatest-width integers.

Typedef names are defined in the following categories:

— integer types having certain exact widths;

— integer types having at least certain specified widths;

— fastest integer types having at least certain specified widths;

— integer types wide enough to hold pointers to objects;

— integer types having greatest width.

(Some of these typedef names may denote the same type.)

Corresponding macros specify limits of the defined types, construct suitable character constants, and provide conversion specifiers for use with the formatted input/output functions.

For each typedef name described herein that can be defined as a type existing in the implementation,[149] `<inttypes.h>` shall define that typedef name, and it shall define the associated macros. Conversely, for each typedef name described herein that cannot be defined as a type existing in the implementation, `<inttypes.h>` shall not define that typedef name, nor shall it define the associated macros.

7.4.1 Typedef names for integer types

When typedef names differing only in the absence or presence of the initial `u` are defined, they shall denote corresponding signed and unsigned types as described in subclause 6.1.2.5.

7.4.1.1 Exact-width integer types

Each of the following typedef names designates an integer type that has exactly the specified width. These typedef names have the general form of `int`*n*`_t` or `uint`*n*`_t` where *n* is the required width. For example, `uint8_t` denotes an unsigned integer type that has a width of exactly 8 bits.

The following designate exact-width signed integer types:

```
int8_t    int16_t    int32_t    int64_t
```

149. Some of these typedef names may denote implementation-defined extended integer types.

The following designate exact-width unsigned integer types:

```
uint8_t    uint16_t    uint32_t    uint64_t
```

(Any of these types might not exist.)

7.4.1.2 Minimum-width integer types

Each of the following typedef names designates an integer type that has at least the specified width, such that no integer type of lesser size has at least the specified width. These typedef names have the general form of `int_least`*n*`_t` or `uint_least`*n*`_t` where *n* is the minimum required width. For example, `int_least32_t` denotes a signed integer type that has a width of at least 32 bits.

The following designate minimum-width signed integer types:

```
int_least8_t      int_least16_t
int_least32_t     int_least64_t
```

The following designate minimum-width unsigned integer types:

```
uint_least8_t     uint_least16_t
uint_least32_t    uint_least64_t
```

(These types must exist.)

7.4.1.3 Fastest minimum-width integer types

Each of the following typedef names designates an integer type that is usually fastest[150] to operate with among all integer types that have at least the specified width. These typedef names have the general form of `int_fast`*n*`_t` or `uint_fast`*n*`_t` where *n* is the minimum required width. For example, `int_fast16_t` denotes the fastest signed integer type that has a width of at least 16 bits.

The following designate fastest minimum-width signed integer types:

```
int_fast8_t      int_fast16_t
int_fast32_t     int_fast64_t
```

The following designate fastest minimum-width unsigned integer types:

```
uint_fast8_t     uint_fast16_t
uint_fast32_t    uint_fast64_t
```

(These types must exist.)

7.4.1.4 Integer types capable of holding object pointers

The following typedef name designates a signed integer type with the property that any valid pointer to `void` can be converted to this type, then converted back to pointer to `void`, and the result will compare equal to the original pointer:

```
intptr_t
```

The following typedef name designates an unsigned integer type with the property that any valid pointer to `void` can be converted to this type, then converted back to pointer to `void`, and the result will compare equal to the original pointer:

```
uintptr_t
```

(Either or both of these types might not exist.)

150. The designated type is not guaranteed to be fastest for all purposes; if the implementation has no clear grounds for choosing one type over another, it will simply pick some integer type satisfying the signedness and width requirements.

7.4.1.5 Greatest-width integer types

The following typedef name designates a signed integer type capable of representing any value of any signed integer type:

```
intmax_t
```

The following typedef name designates an unsigned integer type capable of representing any value of any unsigned integer type:

```
uintmax_t
```

(These types must exist.)

7.4.2 Limits of specified-width integer types

The following object-like macros[151] specify the minimum and maximum limits of integer types corresponding to the typedef names defined in <inttypes.h>. Each macro name corresponds to a similar typedef name in subclause 7.4.1.

Each instance of any defined macro shall be replaced by a constant expression suitable for use in `#if` preprocessing directives, and this expression shall have the same type as would an expression that is an object of the corresponding type converted according to the integer promotions. Its implementation-defined value shall be equal to or greater in magnitude (absolute value) than the corresponding value given below, with the same sign.

7.4.2.1 Limits of exact-width integer types

— minimum values of exact-width signed integer types

`INT8_MIN`	–127
`INT16_MIN`	–32767
`INT32_MIN`	–2147483647
`INT64_MIN`	–9223372036854775807

(The value must be either that given or exactly 1 less.)

— maximum values of exact-width signed integer types

`INT8_MAX`	+127
`INT16_MAX`	+32767
`INT32_MAX`	+2147483647
`INT64_MAX`	+9223372036854775807

(The value must be exactly that given.)

— maximum values of exact-width unsigned integer types

`UINT8_MAX`	255
`UINT16_MAX`	65535
`UINT32_MAX`	4294967295
`UINT64_MAX`	18446744073709551615

(The value must be exactly that given.)

7.4.2.2 Limits of minimum-width integer types

— minimum values of minimum-width signed integer types

`INT_LEAST8_MIN`	–127
`INT_LEAST16_MIN`	–32767
`INT_LEAST32_MIN`	–2147483647
`INT_LEAST64_MIN`	–9223372036854775807

151. C++ implementations should define these macros only when the macro `__STDC_INTTYPES_LIMITS` is defined before <inttypes.h> is included.

— maximum values of minimum-width signed integer types

```
INT_LEAST8_MAX                     +127
INT_LEAST16_MAX                  +32767
INT_LEAST32_MAX             +2147483647
INT_LEAST64_MAX    +9223372036854775807
```

— maximum values of minimum-width unsigned integer types

```
UINT_LEAST8_MAX                     255
UINT_LEAST16_MAX                  65535
UINT_LEAST32_MAX             4294967295
UINT_LEAST64_MAX   18446744073709551615
```

7.4.2.3 Limits of fastest minimum-width integer types

— minimum values of fastest minimum-width signed integer types

```
INT_FAST8_MIN                      –127
INT_FAST16_MIN                   –32767
INT_FAST32_MIN              –2147483647
INT_FAST64_MIN     –9223372036854775807
```

— maximum values of fastest minimum-width signed integer types

```
INT_FAST8_MAX                      +127
INT_FAST16_MAX                   +32767
INT_FAST32_MAX              +2147483647
INT_FAST64_MAX     +9223372036854775807
```

— maximum values of fastest minimum-width unsigned integer types

```
UINT_FAST8_MAX                      255
UINT_FAST16_MAX                   65535
UINT_FAST32_MAX              4294967295
UINT_FAST64_MAX    18446744073709551615
```

7.4.2.4 Limits of integer types capable of holding object pointers

— minimum value of pointer-holding signed integer type

```
INTPTR_MIN                       –32767
```

— maximum value of pointer-holding signed integer type

```
INTPTR_MAX                       +32767
```

— maximum value of pointer-holding unsigned integer type

```
UINTPTR_MAX                       65535
```

7.4.2.5 Limits of greatest-width integer types

— minimum value of greatest-width signed integer type

```
INTMAX_MIN         –9223372036854775807
```

— maximum value of greatest-width signed integer type

```
INTMAX_MAX         +9223372036854775807
```

— maximum value of greatest-width unsigned integer type

```
UINTMAX_MAX        18446744073709551615
```

7.4.3 Macros for integer constants

The following function-like macros[152] expand to integer constants suitable for initializing objects that have

152. C++ implementations should define these macros only when the macro `__STDC_INTTYPES_CONSTANTS` is defined before `<inttypes.h>` is included.

integer types corresponding to typedef names defined in `<inttypes.h>`. Each macro name corresponds to a similar typedef name in subclause 7.4.1.2 or 7.4.1.5.

The argument in any instance of these macros shall be a decimal, octal, or hexadecimal constant (as defined in subclause 6.1.3.2) with a value that does not exceed the limits for the corresponding type.

7.4.3.1 Macros for minimum-width integer constants

Each of the following macros expands to an integer constant having the value specified by its argument and a type with at least the specified width. These macro names have the general form of `INT`*n*`_C` or `UINT`*n*`_C` where *n* is the minimum required width. For example, `UINT64_C(0x123)` might expand to the integer constant `0x123ULL`.

The following expand to integer constants that have signed integer types:

```
INT8_C(value)     INT16_C(value)
INT32_C(value)    INT64_C(value)
```

The following expand to integer constants that have unsigned integer types:

```
UINT8_C(value)    UINT16_C(value)
UINT32_C(value)   UINT64_C(value)
```

7.4.3.2 Macros for greatest-width integer constants

The following macro expands to an integer constant having the value specified by its argument and the type `intmax_t`:

```
INTMAX_C(value)
```

The following macro expands to an integer constant having the value specified by its argument and the type `uintmax_t`:

```
UINTMAX_C(value)
```

7.4.4 Macros for format specifiers

Each of the following object-like macros[153] expands to a string literal containing a conversion specifier, possibly modified by a prefix such as `hh`, `h`, `l`, or `ll`, suitable for use within the format argument of a formatted input/output function when converting the corresponding integer type. These macro names have the general form of `PRI` (character string literals for the `fprintf` family) or `SCN` (character string literals for the `fscanf` family),[154] followed by the conversion specifier, followed by a name corresponding to a similar typedef name in subclause 7.4.1. For example, `PRIdFAST32` can be used in a format string to print the value of an integer of type `int_fast32_t`.

The `fprintf` macros for signed integers are:

```
PRId8         PRId16         PRId32         PRId64
PRIdLEAST8    PRIdLEAST16    PRIdLEAST32    PRIdLEAST64
PRIdFAST8     PRIdFAST16     PRIdFAST32     PRIdFAST64
PRIdMAX       PRIdPTR

PRIi8         PRIi16         PRIi32         PRIi64
PRIiLEAST8    PRIiLEAST16    PRIiLEAST32    PRIiLEAST64
PRIiFAST8     PRIiFAST16     PRIiFAST32     PRIiFAST64
PRIiMAX       PRIiPTR
```

153. C++ implementations should define these macros only when the macro `__STDC_INTTYPES_PRINT_SCAN` is defined before `<inttypes.h>` is included.

154. Separate macros are given for use with `fprintf` and `fscanf` functions because, typically, different format specifiers are required for `fprintf` and `fscanf` even when the type is the same.

The `fprintf` macros for unsigned integers are:

PRIo8	PRIo16	PRIo32	PRIo64
PRIoLEAST8	PRIoLEAST16	PRIoLEAST32	PRIoLEAST64
PRIoFAST8	PRIoFAST16	PRIoFAST32	PRIoFAST64
PRIoMAX	PRIoPTR		
PRIu8	PRIu16	PRIu32	PRIu64
PRIuLEAST8	PRIuLEAST16	PRIuLEAST32	PRIuLEAST64
PRIuFAST8	PRIuFAST16	PRIuFAST32	PRIuFAST64
PRIuMAX	PRIuPTR		
PRIx8	PRIx16	PRIx32	PRIx64
PRIxLEAST8	PRIxLEAST16	PRIxLEAST32	PRIxLEAST64
PRIxFAST8	PRIxFAST16	PRIxFAST32	PRIxFAST64
PRIxMAX	PRIxPTR		
PRIX8	PRIX16	PRIX32	PRIX64
PRIXLEAST8	PRIXLEAST16	PRIXLEAST32	PRIXLEAST64
PRIXFAST8	PRIXFAST16	PRIXFAST32	PRIXFAST64
PRIXMAX	PRIXPTR		

The `fscanf` macros for signed integers are:

SCNd8	SCNd16	SCNd32	SCNd64
SCNdLEAST8	SCNdLEAST16	SCNdLEAST32	SCNdLEAST64
SCNdFAST8	SCNdFAST16	SCNdFAST32	SCNdFAST64
SCNdMAX	SCNdPTR		
SCNi8	SCNi16	SCNi32	SCNi64
SCNiLEAST8	SCNiLEAST16	SCNiLEAST32	SCNiLEAST64
SCNiFAST8	SCNiFAST16	SCNiFAST32	SCNiFAST64
SCNiMAX	SCNiPTR		

The `fscanf` macros for unsigned integers are:

SCNo8	SCNo16	SCNo32	SCNo64
SCNoLEAST8	SCNoLEAST16	SCNoLEAST32	SCNoLEAST64
SCNoFAST8	SCNoFAST16	SCNoFAST32	SCNoFAST64
SCNoMAX	SCNoPTR		
SCNu8	SCNu16	SCNu32	SCNu64
SCNuLEAST8	SCNuLEAST16	SCNuLEAST32	SCNuLEAST64
SCNuFAST8	SCNuFAST16	SCNuFAST32	SCNuFAST64
SCNuMAX	SCNuPTR		
SCNx8	SCNx16	SCNx32	SCNx64
SCNxLEAST8	SCNxLEAST16	SCNxLEAST32	SCNxLEAST64
SCNxFAST8	SCNxFAST16	SCNxFAST32	SCNxFAST64
SCNxMAX	SCNxPTR		

Because the default argument promotions do not affect pointer parameters, there might not exist suitable `fscanf` format specifiers for some of the typedef names defined in this header. Consequently, as a special exception to the requirement that the implementation shall define all macros associated with each typedef name defined in this header, in such a case the problematic `fscanf` macros may be left undefined.

Example

```
#include <inttypes.h>
#include <wchar.h>
int main(void) {
     uintmax_t i = UINTMAX_MAX;     // this type always exists
     wprintf(L"The largest integer value is %020" PRIxMAX "\n", i);
     return 0;
}
```

7.4.5 Limits of other integer types

The following object-like macros[151] specify the minimum and maximum limits of integer types corresponding to typedef names defined in other standard headers.

Each instance of these macros shall be replaced by a constant expression suitable for use in `#if` preprocessing directives, and this expression shall have the same type as would an expression that is an object of the corresponding type converted according to the integer promotions. Its implementation-defined value shall be equal to or greater in magnitude (absolute value) than the corresponding value given below, with the same sign.

— limits of `ptrdiff_t`

`PTRDIFF_MIN`	–65535
`PTRDIFF_MAX`	+65535

— limits of `sig_atomic_t`

`SIG_ATOMIC_MIN`	*see below*
`SIG_ATOMIC_MAX`	*see below*

— limit of `size_t`

`SIZE_MAX`	65535

— limits of `wchar_t`

`WCHAR_MIN`	*see below*
`WCHAR_MAX`	*see below*

— limits of `wint_t`

`WINT_MIN`	*see below*
`WINT_MAX`	*see below*

If `sig_atomic_t` is defined as a signed integer type, the value of `SIG_ATOMIC_MIN` shall be no greater than –127 and the value of `SIG_ATOMIC_MAX` shall be no less than 127; otherwise, `sig_atomic_t` is defined as an unsigned integer type, and the value of `SIG_ATOMIC_MIN` shall be 0 and the value of `SIG_ATOMIC_MAX` shall be no less than 255.

If `wchar_t` is defined as a signed integer type, the value of `WCHAR_MIN` shall be no greater than –127 and the value of `WCHAR_MAX` shall be no less than 127; otherwise, `wchar_t` is defined as an unsigned integer type, and the value of `WCHAR_MIN` shall be 0 and the value of `WCHAR_MAX` shall be no less than 255.

If `wint_t` is defined as a signed integer type, the value of `WINT_MIN` shall be no greater than –32767 and the value of `WINT_MAX` shall be no less than 32767; otherwise, `wint_t` is defined as an unsigned integer type, and the value of `WINT_MIN` shall be 0 and the value of `WINT_MAX` shall be no less than 65535.

7.4.6 Conversion functions for greatest-width integer types

7.4.6.1 The strtoimax function

Synopsis

```
#include <inttypes.h>
intmax_t strtoimax(const char * restrict nptr,
                   char ** restrict endptr, int base);
```

Description

The strtoimax function is equivalent to strtol, except that the initial portion of the string is converted to intmax_t representation.

Returns

The strtoimax function returns the converted value, if any. If no conversion could be performed zero is returned. If the correct value is outside the range of representable values, INTMAX_MAX or INTMAX_MIN is returned (according to the sign of the value), and the value of the macro ERANGE is stored in errno.

7.4.6.2 The strtoumax function

Synopsis

```
#include <inttypes.h>
uintmax_t strtoumax(const char * restrict nptr,
                    char ** restrict endptr, int base);
```

Description

The strtoumax function is equivalent to strtoul, except that the initial portion of the string is converted to uintmax_t representation.

Returns

The strtoumax function returns the converted value, if any. If no conversion could be performed zero is returned. If the correct value is outside the range of representable values, UINTMAX_MAX is returned, and the value of the macro ERANGE is stored in errno.

7.4.6.3 The wcstoimax function

Synopsis

```
#include <stddef.h>           // for wchar_t
#include <inttypes.h>
intmax_t wcstoimax(const wchar_t * restrict nptr,
                   wchar_t ** restrict endptr, int base);
```

Description

The wcstoimax function is equivalent to wcstol, except that the initial portion of the wide string is converted to intmax_t representation.

Returns

The wcstoimax function returns the converted value, if any. If no conversion could be performed zero is returned. If the correct value is outside the range of representable values, INTMAX_MAX or INTMAX_MIN is returned (according to the sign of the value), and the value of the macro ERANGE is stored in errno.

7.4.6.4 The wcstoumax function

Synopsis

```
#include <stddef.h>           // for wchar_t
#include <inttypes.h>
uintmax_t wcstoumax(const wchar_t * restrict nptr,
                    wchar_t ** restrict endptr, int base);
```

Description

The wcstoumax function is equivalent to wcstoul, except that the initial portion of the wide string is converted to uintmax_t representation.

Returns

The wcstoumax function returns the converted value, if any. If no conversion could be performed zero is returned. If the correct value is outside the range of representable values, UINTMAX_MAX is returned, and the value of the macro ERANGE is stored in errno.