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

Douglas A. Gwyn

US Army Research Laboratory
Adelphi, MD

ABSTRACT

This is a proposed replacement for the current C9*x* (Draft 10) subclause 7.4; it supersedes Doc. No. WG14/N737 (J11/97-100), which was distributed in the post-London mailing. (Actually, that mailing contained the wrong version; in the SC22WG14 mailing list, I asked that it be replaced by the version in `ftp://ftp.arl.mil/arch/n737.ps`.) Since my original proposal, I have received much useful feedback, which has been taken into account.

In the course of this work, I noticed several apparent problems with the specification. In fixing these problems, there is a delicate balance between the force of logical necessity and invention. Vertical bars in the margin indicate innovations that I have identified as potentially controversial, or wording that could be affected by other pending proposals; I have isolated these so that if any of them is modified or rejected by the Committee, it can be quickly and accurately edited in the TROFF source file that will be provided to the Document Editor as a ‘‘plug-in’’ replacement for the old subclause 7.4.

7.4 Integer types `<inttypes.h>`

The header `<inttypes.h>` declares sets of typedef names for integer types having specified widths and defines corresponding sets of macros.

Typedef names are declared 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;

— fastest integer types with unspecified width;

— integer types large 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 declared integer types, construct suitable character constants, and provide conversion specifiers for use with the formatted input/output functions. There are also four function-like macros for converting numeric character strings to greatest-width integers.

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

Throughout this subclause, the *width* of an integer type is defined as the number of significant bits, including sign if any, in the representation of that type.

Recommended practice

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

7.4.1 Typedef names for integer types

When typedef names differing only in the absence or presence of the initial `u` are declared, 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 names designates an integer type that has exactly the specified width. These 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 is exactly 8 bits wide.

The following designate exact-width signed integer types:

```
int8_t     int16_t     int32_t     int64_t
```

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 names designates an integer type that has at least the specified width. These 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 is at least 32 bits wide.

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.)

Recommended practice

Each of these names should denote one of the *smallest* types satisfying the requirements.

7.4.1.3 Fastest minimum-width integer types

Each of the following names designates an integer type that is usually fastest to operate with among all integer types that have at least the specified width. These 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 is at least 16 bits wide.

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.)

Recommended practice

It is acknowledged that singling out one type as ‘‘usually fastest’’ can be problematic. If the implementation has no clear grounds for choosing one type over another, it may pick any integer type satisfying the signedness and width requirements.

7.4.1.4 Fastest integer types

The following name designates a signed integer type that is usually fastest among all signed integer types having widths of at least 16 bits:

```
intfast_t
```

It denotes the same type as `int_fast16_t`.

The following name designates an unsigned integer type that is usually fastest among all unsigned integer types having widths of at least 16 bits:

```
uintfast_t
```

It denotes the same type as `uint_fast16_t`.

(These types must exist.)

7.4.1.5 Integer types capable of holding object pointers

The following 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 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
```

(Support for such conversions is a common extension; however, either or both of these types might not exist.)

7.4.1.6 Greatest-width integer types

The following name designates a signed integer type capable of holding any value that can be represented by any signed integer type:

```
intmax_t
```

The following name designates an unsigned integer type capable of holding any value that can be represented by any unsigned integer type:

```
uintmax_t
```

(These types must exist.)

7.4.2 Limits of integer types

The following object-like macros specify the minimum and maximum limits of the integer types declared 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 -2^{n-1} or $1-2^{n-1}$.)

— maximum values of exact-width signed integer types

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

(The value must be $2^{n-1}-1$.)

— maximum values of exact-width unsigned integer types

```
UINT8_MAX                          255
```

```
UINT16_MAX                     65535
UINT32_MAX                4294967295
UINT64_MAX      18446744073709551615
```
(The value must be 2^n-1.)

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
```

— 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 fastest integer types

— minimum value of fastest signed integer type
```
INTFAST_MIN                     –32767
```

— maximum value of fastest signed integer type
```
INTFAST_MAX                     +32767
```

— maximum value of fastest unsigned integer type
```
UINTFAST_MAX                     65535
```

7.4.2.5 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

7.4.2.6 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 expand to integer constants suitable for initializing objects that have integer types declared in `<inttypes.h>`. Each macro name corresponds to a similar typedef name in subclause 7.4.1.2 or 7.4.1.6.

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 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 a signed integer type having width at least as great as any other signed integer type:
`INTMAX_C(`*value*`)`

The following macro expands to an integer constant having the value specified by its argument and an unsigned integer type having width at least as great as any other unsigned integer type:
`UINTMAX_C(`*value*`)`

7.4.4 Macros for format specifiers

Each of the following object-like macros expands to a string literal containing a conversion specifier, possibly modified by a prefix such as `h`, `l`, or `ll`, suitable for use within the format argument of a formatted input/output function when converting the corresponding integer type. These names have the general form of `PRI` (character string literals for the `fprintf` family), `SCN` (character string literals for the `fscanf` family),[122] `WPRI` (wide string literals for the `fwprintf` family), or `WSCN` (wide string

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

literals for the fwscanf family), 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 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
PRIdFAST     PRIdMAX       PRIdPTR

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

The fprintf macros for unsigned integers are:

```
PRIo8        PRIo16        PRIo32        PRIo64
PRIoLEAST8   PRIoLEAST16   PRIoLEAST32   PRIoLEAST64
PRIoFAST8    PRIoFAST16    PRIoFAST32    PRIoFAST64
PRIoFAST     PRIoMAX       PRIoPTR

PRIu8        PRIu16        PRIu32        PRIu64
PRIuLEAST8   PRIuLEAST16   PRIuLEAST32   PRIuLEAST64
PRIuFAST8    PRIuFAST16    PRIuFAST32    PRIuFAST64
PRIuFAST     PRIuMAX       PRIuPTR

PRIx8        PRIx16        PRIx32        PRIx64
PRIxLEAST8   PRIxLEAST16   PRIxLEAST32   PRIxLEAST64
PRIxFAST8    PRIxFAST16    PRIxFAST32    PRIxFAST64
PRIxFAST     PRIxMAX       PRIxPTR

PRIX8        PRIX16        PRIX32        PRIX64
PRIXLEAST8   PRIXLEAST16   PRIXLEAST32   PRIXLEAST64
PRIXFAST8    PRIXFAST16    PRIXFAST32    PRIXFAST64
PRIXFAST     PRIXMAX       PRIXPTR
```

The fscanf macros for signed integers are:

```
SCNd8        SCNd16        SCNd32        SCNd64
SCNdLEAST8   SCNdLEAST16   SCNdLEAST32   SCNdLEAST64
SCNdFAST8    SCNdFAST16    SCNdFAST32    SCNdFAST64
SCNdFAST     SCNdMAX       SCNdPTR

SCNi8        SCNi16        SCNi32        SCNi64
SCNiLEAST8   SCNiLEAST16   SCNiLEAST32   SCNiLEAST64
SCNiFAST8    SCNiFAST16    SCNiFAST32    SCNiFAST64
SCNiFAST     SCNiMAX       SCNiPTR
```

The fscanf macros for unsigned integers are:

```
SCNo8        SCNo16        SCNo32        SCNo64
SCNoLEAST8   SCNoLEAST16   SCNoLEAST32   SCNoLEAST64
SCNoFAST8    SCNoFAST16    SCNoFAST32    SCNoFAST64
SCNoFAST     SCNoMAX       SCNoPTR

SCNu8        SCNu16        SCNu32        SCNu64
SCNuLEAST8   SCNuLEAST16   SCNuLEAST32   SCNuLEAST64
SCNuFAST8    SCNuFAST16    SCNuFAST32    SCNuFAST64
SCNuFAST     SCNuMAX       SCNuPTR
```

SCNx8	SCNx16	SCNx32	SCNx64
SCNxLEAST8	SCNxLEAST16	SCNxLEAST32	SCNxLEAST64
SCNxFAST8	SCNxFAST16	SCNxFAST32	SCNxFAST64
SCNxFAST	SCNxMAX	SCNxPTR	

The `fwprintf` macros for signed integers are:

WPRId8	WPRId16	WPRId32	WPRId64
WPRIdLEAST8	WPRIdLEAST16	WPRIdLEAST32	WPRIdLEAST64
WPRIdFAST8	WPRIdFAST16	WPRIdFAST32	WPRIdFAST64
WPRIdFAST	WPRIdMAX	WPRIdPTR	

WPRIi8	WPRIi16	WPRIi32	WPRIi64
WPRIiLEAST8	WPRIiLEAST16	WPRIiLEAST32	WPRIiLEAST64
WPRIiFAST8	WPRIiFAST16	WPRIiFAST32	WPRIiFAST64
WPRIiFAST	WPRIiMAX	WPRIiPTR	

The `fwprintf` macros for unsigned integers are:

WPRIo8	WPRIo16	WPRIo32	WPRIo64
WPRIoLEAST8	WPRIoLEAST16	WPRIoLEAST32	WPRIoLEAST64
WPRIoFAST8	WPRIoFAST16	WPRIoFAST32	WPRIoFAST64
WPRIoFAST	WPRIoMAX	WPRIoPTR	

WPRIu8	WPRIu16	WPRIu32	WPRIu64
WPRIuLEAST8	WPRIuLEAST16	WPRIuLEAST32	WPRIuLEAST64
WPRIuFAST8	WPRIuFAST16	WPRIuFAST32	WPRIuFAST64
WPRIuFAST	WPRIuMAX	WPRIuPTR	

WPRIx8	WPRIx16	WPRIx32	WPRIx64
WPRIxLEAST8	WPRIxLEAST16	WPRIxLEAST32	WPRIxLEAST64
WPRIxFAST8	WPRIxFAST16	WPRIxFAST32	WPRIxFAST64
WPRIxFAST	WPRIxMAX	WPRIxPTR	

WPRIX8	WPRIX16	WPRIX32	WPRIX64
WPRIXLEAST8	WPRIXLEAST16	WPRIXLEAST32	WPRIXLEAST64
WPRIXFAST8	WPRIXFAST16	WPRIXFAST32	WPRIXFAST64
WPRIXFAST	WPRIXMAX	WPRIXPTR	

The `fwscanf` macros for signed integers are:

WSCNd8	WSCNd16	WSCNd32	WSCNd64
WSCNdLEAST8	WSCNdLEAST16	WSCNdLEAST32	WSCNdLEAST64
WSCNdFAST8	WSCNdFAST16	WSCNdFAST32	WSCNdFAST64
WSCNdFAST	WSCNdMAX	WSCNdPTR	

WSCNi8	WSCNi16	WSCNi32	WSCNi64
WSCNiLEAST8	WSCNiLEAST16	WSCNiLEAST32	WSCNiLEAST64
WSCNiFAST8	WSCNiFAST16	WSCNiFAST32	WSCNiFAST64
WSCNiFAST	WSCNiMAX	WSCNiPTR	

The `fwscanf` macros for unsigned integers are:

WSCNo8	WSCNo16	WSCNo32	WSCNo64
WSCNoLEAST8	WSCNoLEAST16	WSCNoLEAST32	WSCNoLEAST64
WSCNoFAST8	WSCNoFAST16	WSCNoFAST32	WSCNoFAST64
WSCNoFAST	WSCNoMAX	WSCNoPTR	

WSCNu8	WSCNu16	WSCNu32	WSCNu64
WSCNuLEAST8	WSCNuLEAST16	WSCNuLEAST32	WSCNuLEAST64
WSCNuFAST8	WSCNuFAST16	WSCNuFAST32	WSCNuFAST64

```
WSCNuFAST      WSCNuMAX       WSCNuPTR

WSCNx8         WSCNx16        WSCNx32        WSCNx64
WSCNxLEAST8    WSCNxLEAST16   WSCNxLEAST32   WSCNxLEAST64
WSCNxFAST8     WSCNxFAST16    WSCNxFAST32    WSCNxFAST64
WSCNxFAST      WSCNxMAX       WSCNxPTR
```

Because the default argument promotions do not affect pointer parameters, there might not exist suitable `fscanf` (or `fwscanf`) format specifiers for some of the declared types, particularly those with typedef names containing `8`. Consequently, as a special exception to the requirement that the implementation define all macros associated with a declared type, in such a case the problematic `fscanf` (or `fwscanf`) macros may be left undefined.

Example

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

7.4.5 Conversion macros[123] for greatest-width integer types

7.4.5.1 The `strtoimax` macro

Synopsis

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

Description

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

Returns

The `strtoimax` macro 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`.

7.4.5.2 The `strtoumax` macro

Synopsis

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

Description

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

123. `strtoimax`, `strtoumax`, `wcstoimax`, and `wcstoumax` are specified as macros rather than functions, to allow them to be implemented as synonyms for functions named in subclauses 7.13.1 and 7.18.4.1.

Returns

The `strtoumax` macro 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`.

7.4.5.3 The `wcstoimax` macro

Synopsis

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

Description

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

Returns

The `wcstoimax` macro 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`.

7.4.5.4 The `wcstoumax` macro

Synopsis

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

Description

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

Returns

The `wcstoumax` macro 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`.