N1835=05-0095 2005-06-23

<stdint.h> for C++

Steve Clamage, Sun Microsystems

Header <stdint.h>, introduced in C99, provides a portable way for programmers to specify integers of a needed size. The header consists of typedefs and macros. The typedefs refer to existing standard types, and possibly to implementation-specific types. Not all typedefs and macros need to be provided by an implementation. Those which are not supported can be omitted.

Proposal N1823 “New Character Types in C++” requires some of the typedefs in <stdint.h>, so adopting <stdint.h> into C++ supports that proposal.

I don't see any conflict with C++ in the C99 specification, so I propose adopting the specification from C99 as modified by TC1 and TC2, except for the following:

Header <cstdint> puts the declarations described by the C99 standard into namespace std. The C++ version of <stdint.h> puts the declarations into both the global namespace and namespace std, as with the other headers inherited from C.

NOTE: If we later decide to do something different with namespaces for the headers inherited from C, the same would apply to <cstdint> and <stdint.h>.

The relevant pages from the C99 standard, as modified by TC1 and TC2, are attached to this paper.

7.18 Integer types `<stdint.h>`

1 The header **`<stdint.h>`** declares sets of integer types having specified widths, and defines corresponding sets of macros.[217] It also defines macros that specify limits of integer types corresponding to types defined in other standard headers.

2 Types 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 types may denote the same type.)

3 Corresponding macros specify limits of the declared types and construct suitable constants.

4 For each type described herein that the implementation provides,[218] **`<stdint.h>`** shall declare that typedef name and define the associated macros. Conversely, for each type described herein that the implementation does not provide, **`<stdint.h>`** shall not declare that typedef name nor shall it define the associated macros. An implementation shall provide those types described as ‘‘required’’, but need not provide any of the others (described as ‘‘optional’’).

7.18.1 Integer types

1 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 6.2.5; an implementation providing one of these corresponding types shall also provide the other.

2 In the following descriptions, the symbol *N* represents an unsigned decimal integer with no leading zeros (e.g., 8 or 24, but not 04 or 048).

---

217) See ‘‘future library directions’’ (7.26.8).

218) Some of these types may denote implementation-defined extended integer types.

7.18.1.1 Exact-width integer types

1 The typedef name **int**_N_**_t** designates a signed integer type with width _N_, no padding bits, and a two's complement representation. Thus, **int8_t** denotes a signed integer type with a width of exactly 8 bits.

2 The typedef name **uint**_N_**_t** designates an unsigned integer type with width _N_. Thus, **uint24_t** denotes an unsigned integer type with a width of exactly 24 bits.

3 These types are optional. However, if an implementation provides integer types with widths of 8, 16, 32, or 64 bits, no padding bits, and (for the signed types) that have a two's complement representation, it shall define the corresponding typedef names.

7.18.1.2 Minimum-width integer types

1 The typedef name **int_least**_N_**_t** designates a signed integer type with a width of at least _N_, such that no signed integer type with lesser size has at least the specified width. Thus, **int_least32_t** denotes a signed integer type with a width of at least 32 bits.

2 The typedef name **uint_least**_N_**_t** designates an unsigned integer type with a width of at least _N_, such that no unsigned integer type with lesser size has at least the specified width. Thus, **uint_least16_t** denotes an unsigned integer type with a width of at least 16 bits.

3 The following types are required:

```
int_least8_t            uint_least8_t
int_least16_t           uint_least16_t
int_least32_t           uint_least32_t
int_least64_t           uint_least64_t
```

All other types of this form are optional.

7.18.1.3 Fastest minimum-width integer types

1 Each of the following types designates an integer type that is usually fastest[219)] to operate with among all integer types that have at least the specified width.

2 The typedef name **int_fast**_N_**_t** designates the fastest signed integer type with a width of at least _N_. The typedef name **uint_fast**_N_**_t** designates the fastest unsigned integer type with a width of at least _N_.

219) 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.

3 The following types are required:

```
int_fast8_t                uint_fast8_t
int_fast16_t               uint_fast16_t
int_fast32_t               uint_fast32_t
int_fast64_t               uint_fast64_t
```

All other types of this form are optional.

7.18.1.4 Integer types capable of holding object pointers

1 The following type 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 type 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
```

These types are optional.

7.18.1.5 Greatest-width integer types

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

```
intmax_t
```

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

```
uintmax_t
```

These types are required.

7.18.2 Limits of specified-width integer types

1 The following object-like macros[220)] specify the minimum and maximum limits of the types declared in **<stdint.h>**. Each macro name corresponds to a similar type name in 7.18.1.

2 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

220) C++ implementations should define these macros only when **__STDC_LIMIT_MACROS** is defined before **<stdint.h>** is included.

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, except where stated to be exactly the given value.

7.18.2.1 Limits of exact-width integer types

1 — minimum values of exact-width signed integer types

INT*N***_MIN** exactly $-(2^{N-1})$

— maximum values of exact-width signed integer types

INT*N***_MAX** exactly $2^{N-1} - 1$

— maximum values of exact-width unsigned integer types

UINT*N***_MAX** exactly $2^{N} - 1$

7.18.2.2 Limits of minimum-width integer types

1 — minimum values of minimum-width signed integer types

INT_LEAST*N***_MIN** $-(2^{N-1} - 1)$

— maximum values of minimum-width signed integer types

INT_LEAST*N***_MAX** $2^{N-1} - 1$

— maximum values of minimum-width unsigned integer types

UINT_LEAST*N***_MAX** $2^{N} - 1$

7.18.2.3 Limits of fastest minimum-width integer types

1 — minimum values of fastest minimum-width signed integer types

INT_FAST*N***_MIN** $-(2^{N-1} - 1)$

— maximum values of fastest minimum-width signed integer types

INT_FAST*N***_MAX** $2^{N-1} - 1$

— maximum values of fastest minimum-width unsigned integer types

UINT_FAST*N***_MAX** $2^{N} - 1$

7.18.2.4 Limits of integer types capable of holding object pointers

1 — minimum value of pointer-holding signed integer type

INTPTR_MIN $-(2^{15} - 1)$

— maximum value of pointer-holding signed integer type

INTPTR_MAX $2^{15} - 1$

— maximum value of pointer-holding unsigned integer type

UINTPTR_MAX $2^{16} - 1$

7.18.2.5 Limits of greatest-width integer types

1 — minimum value of greatest-width signed integer type

INTMAX_MIN $-(2^{63} - 1)$

— maximum value of greatest-width signed integer type

INTMAX_MAX $2^{63} - 1$

— maximum value of greatest-width unsigned integer type

UINTMAX_MAX $2^{64} - 1$

7.18.3 Limits of other integer types

1 The following object-like macros[221)] specify the minimum and maximum limits of integer types corresponding to types defined in other standard headers.

2 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. An implementation shall define only the macros corresponding to those typedef names it actually provides.[222)]

— 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**

---

221) C++ implementations should define these macros only when **__STDC_LIMIT_MACROS** is defined before **<stdint.h>** is included.

222) A freestanding implementation need not provide all of these types.

WCHAR_MIN	*see below*
WCHAR_MAX	*see below*

— limits of **wint_t**

WINT_MIN	*see below*
WINT_MAX	*see below*

3 If **sig_atomic_t** (see 7.14) 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.

4 If **wchar_t** (see 7.17) 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.[223)]

5 If **wint_t** (see 7.24) 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.

7.18.4 Macros for integer constants

1 The following function-like macros[224)] expand to integer constants suitable for initializing objects that have integer types corresponding to types defined in **<stdint.h>**. Each macro name corresponds to a similar type name in 7.18.1.2 or 7.18.1.5.

2 The argument in any instance of these macros shall be a decimal, octal, or hexadecimal constant (as defined in 6.4.4.1) with a value that does not exceed the limits for the corresponding type.

3 Each invocation of one of these macros shall expand to an integer constant expression suitable for use in **#if** preprocessing directives. The type of the expression shall have the same type as would an expression of the corresponding type converted according to the integer promotions. The value of the expression shall be that of the argument.

223) The values **WCHAR_MIN** and **WCHAR_MAX** do not necessarily correspond to members of the extended character set.

224) C++ implementations should define these macros only when **__STDC_CONSTANT_MACROS** is defined before **<stdint.h>** is included.

7.18.4.1 Macros for minimum-width integer constants

1 The macro **INT***N*_C(*value*) shall expand to an integer constant expression corresponding to the type **int_least***N*_t. The macro **UINT***N*_C(*value*) shall expand to an integer constant expression corresponding to the type **uint_least***N*_t. For example, if **uint_least64_t** is a name for the type **unsigned long long int**, then **UINT64_C(0x123)** might expand to the integer constant **0x123ULL**.

7.18.4.2 Macros for greatest-width integer constants

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

INTMAX_C(*value***)**

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

UINTMAX_C(*value***)**