
*Full credit is given to the above companies including the OS that this PDF file was generated!*

## *Red Hat Enterprise Linux Release 9.2 Manual Pages on 'stdint.h.0p' command*

***$ man stdint.h.0p***

stdint.h(0P) POSIX Programmer's Manual stdint.h(0P)

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

stdint.h ? integer types

SYNOPSIS

#include <stdint.h>

DESCRIPTION

Some of the functionality described on this reference page extends the ISO C standard. Applications shall define the appropriate feature test macro (see the System Interfaces volume of POSIX.1?2017, Section 2.2, The Compilation Environment) to enable the visibility of these symbols in this header.

The <stdint.h> header shall declare sets of integer types having speci? fied widths, and shall define corresponding sets of macros. It shall also define macros that specify limits of integer types corresponding to types defined in other standard headers.

Note: The ``width'' of an integer type is the number of bits used to store its value in a pure binary system; the actual type may use more bits than that (for example, a 28-bit type could

be stored in 32 bits of actual storage). An N-bit signed type has values in the range -2N-1 or 1-2N-1 to 2N-1-1, while an N-bit unsigned type has values in the range 0 to 2N-1.

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.)

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

For each type described herein that the implementation provides, the <stdint.h> header shall declare that typedef name and define the asso? ciated macros. Conversely, for each type described herein that the im? plementation does not provide, the <stdint.h> header shall not declare that typedef name, nor shall it define the associated macros. An imple? mentation shall provide those types described as required, but need not provide any of the others (described as optional).

Integer Types

When typedef names differing only in the absence or presence of the initial u are defined, they shall denote corresponding signed and un? signed types as described in the ISO/IEC 9899:1999 standard, Section 6.2.5; an implementation providing one of these corresponding types shall also provide the other.

In the following descriptions, the symbol N represents an unsigned dec? imal integer with no leading zeros (for example, 8 or 24, but not 04 or 048).

* Exact-width integer types

  The typedef name intN_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.

The typedef name uintN_t designates an unsigned integer type with width N. Thus, uint24_t denotes an unsigned integer type with a width of exactly 24 bits.

The following types are required:

int8_t

int16_t

int32_t

uint8_t

uint16_t

uint32_t

If an implementation provides integer types with width 64 that meet these requirements, then the following types are required: int64_t

uint64_t

In particular, this will be the case if any of the following are true:

-- The implementation supports the _POSIX_V7_ILP32_OFFBIG program? ming environment and the application is being built in the _POSIX_V7_ILP32_OFFBIG programming environment (see the Shell and Utilities volume of POSIX.1?2017, c99, Programming Environ? ments).

-- The implementation supports the _POSIX_V7_LP64_OFF64 program? ming environment and the application is being built in the _POSIX_V7_LP64_OFF64 programming environment.

-- The implementation supports the _POSIX_V7_LPBIG_OFFBIG program? ming environment and the application is being built in the _POSIX_V7_LPBIG_OFFBIG programming environment.

All other types of this form are optional.

* Minimum-width integer types

The typedef name int_leastN_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.

The typedef name uint_leastN_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.

The following types are required: int_least8_t int_least16_t int_least32_t int_least64_t uint_least8_t uint_least16_t uint_least32_t uint_least64_t

All other types of this form are optional.

* Fastest minimum-width integer types

Each of the following types designates an integer type that is usu? ally fastest to operate with among all integer types that have at least the specified width.

The designated type is not guaranteed to be fastest for all pur? poses; 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.

The typedef name int_fastN_t designates the fastest signed integer type with a width of at least N. The typedef name uint_fastN_t designates the fastest unsigned integer type with a width of at least N.

The following types are required: int_fast8_t int_fast16_t int_fast32_t int_fast64_t uint_fast8_t uint_fast16_t uint_fast32_t uint_fast64_t

All other types of this form are optional.

* Integer types capable of holding object pointers

The following type designates a signed integer type with the prop? erty that any valid pointer to void can be converted to this type, then converted back to a pointer to void, and the result will com? pare 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 a pointer to void, and the result will compare equal to the original pointer: uintptr_t

On XSI-conformant systems, the intptr_t and uintptr_t types are re?

quired; otherwise, they are optional.

* Greatest-width integer types

The following type designates a signed integer type capable of rep? resenting 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.

Note: Applications can test for optional types by using the corre? sponding limit macro from Limits of Specified-Width Integer Types.

Limits of Specified-Width Integer Types

The following macros specify the minimum and maximum limits of the types declared in the <stdint.h> header. Each macro name corresponds to a similar type name in Integer Types.

Each instance of any defined macro shall be replaced by a constant ex? pression suitable for use in #if preprocessing directives, and this ex? pression shall have the same type as would an expression that is an ob? ject of the corresponding type converted according to the integer pro? motions. 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.

* Limits of exact-width integer types
  - -- Minimum values of exact-width signed integer types:

    {INTN_MIN} Exactly -(2^N-1)
  - -- Maximum values of exact-width signed integer types:

    {INTN_MAX} Exactly 2^N-1 -1
  - -- Maximum values of exact-width unsigned integer types:

    {UINTN_MAX} Exactly 2^N -1
* Limits of minimum-width integer types
  - -- Minimum values of minimum-width signed integer types:

    {INT_LEASTN_MIN}

    -(2^N-1 -1)
  - -- Maximum values of minimum-width signed integer types:

{INT_LEASTN_MAX}

2^N-1 -1

-- Maximum values of minimum-width unsigned integer types:

{UINT_LEASTN_MAX}

2^N -1

* Limits of fastest minimum-width integer types

-- Minimum values of fastest minimum-width signed integer types:

{INT_FASTN_MIN} -(2^N-1 -1)

-- Maximum values of fastest minimum-width signed integer types:

{INT_FASTN_MAX} 2^N-1 -1

-- Maximum values of fastest minimum-width unsigned integer types:

{UINT_FASTN_MAX}

2^N -1

* Limits of integer types capable of holding object pointers

-- 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

* Limits of greatest-width integer types

-- 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

Limits of Other Integer Types

The following macros specify the minimum and maximum limits of integer types corresponding to types defined in other standard headers.

Each instance of these macros shall be replaced by a constant expres? sion suitable for use in #if preprocessing directives, and this expres? sion shall have the same type as would an expression that is an object

of the corresponding type converted according to the integer promo?
tions. 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 (see the <signal.h> header) is defined as a signed in?
teger 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 shall be 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 (see the <stddef.h> header) 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
shall be 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 (see the <wchar.h> header) 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 shall be 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.

Macros for Integer Constant Expressions

The following macros expand to integer constant expressions suitable for initializing objects that have integer types corresponding to types defined in the <stdint.h> header. Each macro name corresponds to a sim? ilar type name listed under Minimum-width integer types and Greatest-width integer types.

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

The argument in any instance of these macros shall be an unsuffixed in? teger constant with a value that does not exceed the limits for the corresponding type.

* Macros for minimum-width integer constant expressions

  The macro INTN_C(value) shall expand to an integer constant expres? sion corresponding to the type int_leastN_t. The macro UINTN_C(value) shall expand to an integer constant expression cor? responding to the type uint_leastN_t. For example, if uint_least64_t is a name for the type unsigned long long, then UINT64_C(0x123) might expand to the integer constant 0x123ULL.

* Macros for greatest-width integer constant expressions

  The following macro expands to an integer constant expression hav? ing the value specified by its argument and the type intmax_t: INT? MAX_C(value)

  The following macro expands to an integer constant expression hav? ing the value specified by its argument and the type uintmax_t:

UINTMAX_C(value)

The following sections are informative.

## APPLICATION USAGE

None.

## RATIONALE

The <stdint.h> header is a subset of the <inttypes.h> header more suitable for use in freestanding environments, which might not support the formatted I/O functions. In some environments, if the formatted conversion support is not wanted, using this header instead of the <inttypes.h> header avoids defining such a large number of macros.

As a consequence of adding int8_t, the following are true:

* A byte is exactly 8 bits.
* {CHAR_BIT} has the value 8, {SCHAR_MAX} has the value 127, {SCHAR_MIN} has the value -128, and {UCHAR_MAX} has the value 255.

(The POSIX standard explicitly requires 8-bit char and two's-complement arithmetic.)

## FUTURE DIRECTIONS

typedef names beginning with int or uint and ending with _t may be added to the types defined in the <stdint.h> header. Macro names beginning with INT or UINT and ending with _MAX, _MIN, or _C may be added to the macros defined in the <stdint.h> header.

## SEE ALSO

<inttypes.h>, <signal.h>, <stddef.h>, <wchar.h>

The System Interfaces volume of POSIX.1?2017, Section 2.2, The Compilation Environment

## COPYRIGHT

is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.ker?nel.org/doc/man-pages/reporting_bugs.html .