This header declares sets of integral types having specified precisions and defines corresponding sets of macros.

Integral types are declared in the following categories:

— integral types having certain exact precisions;

— integral types having at least certain specified precisions;

— fastest integral types having at least certain specified precisions.

— fastest integral types with unspecified precision;

— integral types large enough to hold pointers to objects;

— integral types having greatest precision;

The macros specify the limits of the declared integral types and provide corresponding conversion specifiers for use with the formatted input/output functions.

For each type described herein that exists in the implementation, this header shall declare that type, and it shall define the associated macros. Conversely, for each of the types described herein that is not implemented, this header shall not declare that type, nor shall it define the associated macros.

0.0.1 Typedef names for integral types

0.0.1.1 Exact-precision integral types The following typedef names designate integral types having exactly the specified precisions. These names have the general format of **int***n***_t** or **uint***n***_t** where *n* is the number of significant bits, including sign if any, in the representation of that type. For example, **int8_t** denotes a signed integral type with exactly 8 significant bits in its representation.

The following designate exact-precision signed integral types: int8_t int16_t int32_t int64_t

The following designate exact-precision unsigned integral types: uint8_t uint16_t uint32_t uint64_t

0.0.1.2 Minimum-precision integral types The following typedef names designate integral types having at least the specified precisions. These names have the general format of **int_least***n***_t** or **uint_least***n***_t** where *n* is the minimum required number of significant bits, including sign if any, in the representation of that type. For example, **int_least32_t** denotes a signed integral type with at least 32 significant bits in its representation.

The following designate minimum-precision signed integral types: int_least8_t int_least16_t int_least32_t int_least64_t

The following designate minimum-precision unsigned integral types: uint_least8_t uint_least16_t uint_least32_t uint_least64_t

0.0.1.3 Fastest minimum-precision integral types The following typedef names designate integral types that are usually fastest to operate with among all integral types having at least the specified precision. These names have the general format of **int_fast***n***_t** or **uint_fast***n***_t** where *n* is the minimum required number of significant bits, including sign if any, in the representation of that type. For example, **int_fast16_t** denotes a fast signed integral type with at least 16 significant bits in its representation.

The following designate fastest minimum-precision signed integral types: int_fast8_t int_fast16_t int_fast32_t int_fast64_t

The following designate fastest minimum-precision unsigned integral types: uint_fast8_t uint_fast16_t uint_fast32_t uint_fast64_t

0.0.1.4 Fastest integral types A signed integral type that is usually fastest among all those having at least 16 bits, including sign, in their representations is denoted by intfast_t

An unsigned integral type that is usually fastest among all those having at least 16 bits in their representations is denoted by uintfast_t

0.0.1.5 Integer types capable of holding object pointers A signed integral type large enough to hold any pointer to **`void`** is designated by intptr_t

An unsigned integral type large enough to hold any pointer to **`void`** is designated by uintptr_t

0.0.1.6 Greatest-precision integral types A signed integral type having greatest precision is designated by intmax_t

An unsigned integral type having greatest precision is designated by uintmax_t

0.0.2 Limits of integral types The following macros specify the minimum and maximum limits of the integral types declared in <inttypes.h>. Each macro name corresponds to a similar typedef name in subclause 7.4.1.

The values given below shall be replaced by constant expressions suitable for use in preprocessing directives, and these expressions shall have the same type as would an expression that is an object of the corresponding type converted according to the integral promotions. Their implementation-defined values shall be equal or greater in magnitude (absolute value) to those shown, with the same sign.

0.0.2.1 Limits of exact-precision integral types

— minimum values of exact-precision signed integral types

— maximum values of exact-precision signed integral types

— maximum values of exact-precision unsigned integral types

0.0.2.2 Limits of minimum-precision integral types

— minimum values of minimum-precision signed integral types

— maximum values of minimum-precision signed integral types

— maximum values of minimum-precision unsigned integral types

0.0.2.3 Limits of fastest minimum-precision integral types

— minimum values of fastest minimum-precision signed integral types

— maximum values of fastest minimum-precision signed integral types

— maximum values of fastest minimum-precision unsigned integral types

0.0.2.4 Limits of fastest integral types

— minimum value of fastest signed integral type

— maximum value of fastest signed integral type

— maximum value of fastest unsigned integral type

0.0.2.5 Limits of integer types capable of holding object pointers

— maximum value of pointer-holding signed integral type

— maximum value of pointer-holding unsigned integral type

0.0.2.6 Limits of greatest-precision integral types

— minimum value of greatest-precision signed integral type

— maximum value of greatest-precision signed integral type

— maximum value of greatest-precision unsigned integral type

0.0.3 Macros for creating integer constants

0.0.3.1 Macros for minimum-precision integer constants Each of the following function-like macros expands to an integer constant having the value specified by its argument and a type with at least the specified precision. These names have the general format of **INT***n***_C** or **UINT***n***_C** where *n* is the minimum required number of significant bits, including sign if any, in the representation of that type. For example, **INT16_C(123)** expands to an integer constant with value 123, with signed integral type having at least 16 significant bits in its representation.

The following expand to signed integer constants: INT8_C(*value*) **INT16_C(***value***)** **INT32_C(***value***)** **INT64_C(***value***)**

The following expand to unsigned integer constants: **UINT8_C(***value***)** **UINT16_C(***value***)** **UINT32_C(***value***)** **UINT64_C(***value***)**

0.0.3.2 Macros for greatest-precision integer constants Each of the following function-like macros expands to an integer constant having the value specified by its argument and an integral type having greatest precision.

The following expands to a signed integer constant: INTMAX_C(*value*)

The following expands to an unsigned integer constant: **UINTMAX_C(***value***)**

0.0.4 Macros for integer format specifiers Each of the following macros expands to a string literal containing a conversion specifier, possibly modified by a prefix such as **h**, **l**, or **ll**, suitable for use with the corresponding integral type within the format argument of a formatted input/output function. These names have the general format of **PRI** (for the **printf** family) or **SCN** (for the **scanf** family), followed by the conversion specifier, followed by a name corresponding to a similar typedef name in subclause 7.4.1. For example, **PRIdFAST32** is used as a format string to print an integer of type **int_fast32_t**.[1]

The **printf** 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 **printf** 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

---

1. Separate macros are given for **printf** and **scanf** functions because, typically, different format specifiers are required for **printf** and **scanf** even when the type is the same.

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 **scanf** 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 **scanf** 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

0.0.5 Conversion macros for largest integral types

*0.0.5.1 The***strtoimax**

5Synopsis

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

5Description

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

5Returns

The 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, or is returned (according to the sign of the value), and the value of the macro is stored in **errno**.

*0.3.0.1 The***strtoumax**

5Synopsis

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

5Description

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

5Returns

The 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, is returned, and the value of the macro is stored in **`errno`**.