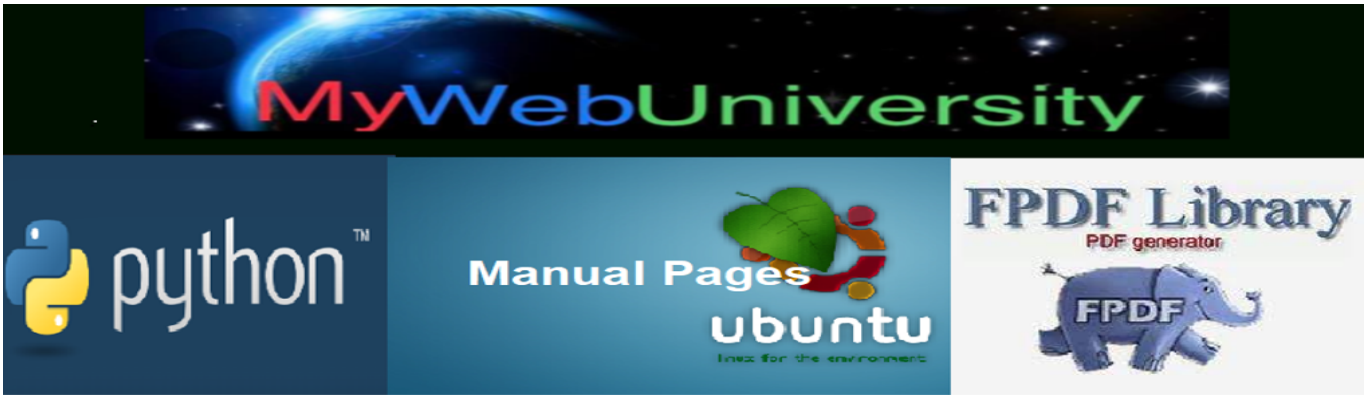

Rocky Enterprise Linux 9.2 Manual Pages on command 'clockid_t.3'

$ man clockid_t.3

SYSTEM_DATA_TYPES(7) Linux Programmer's Manual SYSTEM_DATA_TYPES(7)

NAME

system_data_types - overview of system data types

DESCRIPTION

aiocb

Include: <aio.h>.

```
struct aiocb {
    int           aio_fildes;    /* File descriptor */
    off_t         aio_offset;    /* File offset */
    volatile void  *aio_buf;      /* Location of buffer */
    size_t        aio_nbytes;    /* Length of transfer */
    int           aio_reqprio;   /* Request priority offset */
    struct sigevent aio_sigevent;  /* Signal number and value */
    int           aio_lio_opcode;/* Operation to be performed */
};
```

For further information about this structure, see aio(7).

Conforming to: POSIX.1-2001 and later.

See also: aio_cancel(3), aio_error(3), aio_fsync(3), aio_read(3), aio_return(3), aio_suspend(3), aio_write(3), lio_listio(3)

clock_t

Include: <time.h> or <sys/types.h>. Alternatively, <sys/time.h>.

Used for system time in clock ticks or CLOCKS_PER_SEC (defined in <time.h>). Ac? cording to POSIX, it shall be an integer type or a real-floating type.

Conforming to: C99 and later; POSIX.1-2001 and later.

See also: times(2), clock(3)

clockid_t

Include: <sys/types.h>. Alternatively, <time.h>.

Used for clock ID type in the clock and timer functions. According to POSIX, it shall be defined as an arithmetic type.

Conforming to: POSIX.1-2001 and later.

See also: clock_adjtime(2), clock_getres(2), clock_nanosleep(2), timer_create(2), clock_getcpuclockid(3)

dev_t

Include: <sys/types.h>. Alternatively, <sys/stat.h>.

Used for device IDs. According to POSIX, it shall be an integer type. For further details of this type, see makedev(3).

Conforming to: POSIX.1-2001 and later.

See also: mknod(2), stat(2)

div_t

Include: <stdlib.h>.

```
typedef struct {
    int quot; /* Quotient */
    int rem;  /* Remainder */
} div_t;
```

It is the type of the value returned by the div(3) function.

Conforming to: C99 and later; POSIX.1-2001 and later.

See also: div(3)

double_t

Include: <math.h>.

The implementation's most efficient floating type at least as wide as double. Its type depends on the value of the macro FLT_EVAL_METHOD (defined in <float.h>):

0 double_t is double.

1 double_t is double.

2 double_t is long double.

For other values of FLT_EVAL_METHOD, the type of double_t is implementation-de? fined.

Conforming to: C99 and later; POSIX.1-2001 and later.

See also: the float_t type in this page.

fd_set

Include: <sys/select.h>. Alternatively, <sys/time.h>.

A structure type that can represent a set of file descriptors. According to POSIX, the maximum number of file descriptors in an fd_set structure is the value of the macro FD_SETSIZE.

Conforming to: POSIX.1-2001 and later.

See also: select(2)

fenv_t

Include: <fenv.h>.

This type represents the entire floating-point environment, including control modes and status flags; for further details, see fenv(3).

Conforming to: C99 and later; POSIX.1-2001 and later.

See also: fenv(3)

fexcept_t

Include: <fenv.h>.

This type represents the floating-point status flags collectively; for further de? tails see fenv(3).

Conforming to: C99 and later; POSIX.1-2001 and later.

See also: fenv(3)

FILE

Include: <stdio.h>. Alternatively, <wchar.h>.

An object type used for streams.

Conforming to: C99 and later; POSIX.1-2001 and later.

See also: fclose(3), flockfile(3), fopen(3), fprintf(3), fread(3), fscanf(3), stdin(3), stdio(3)

float_t

Include: <math.h>.

The implementation's most efficient floating type at least as wide as float. Its type depends on the value of the macro FLT_EVAL_METHOD (defined in <float.h>):

0 float_t is float.

1 float_t is double.

2 float_t is long double.

For other values of FLT_EVAL_METHOD, the type of float_t is implementation-defined.

Conforming to: C99 and later; POSIX.1-2001 and later.

See also: the double_t type in this page.

gid_t

Include: <sys/types.h>. Alternatively, <grp.h>, <pwd.h>, <signal.h>, <stropts.h>, <sys/ipc.h>, <sys/stat.h>, or <unistd.h>.

A type used to hold group IDs. According to POSIX, this shall be an integer type.

Conforming to: POSIX.1-2001 and later.

See also: chown(2), getgid(2), getegid(2), getgroups(2), getresgid(2), getgrnam(2), credentials(7)

id_t

Include: <sys/types.h>. Alternatively, <sys/resource.h>.

A type used to hold a general identifier. According to POSIX, this shall be an in? teger type that can be used to contain a pid_t, uid_t, or gid_t.

Conforming to: POSIX.1-2001 and later.

See also: getpriority(2), waitid(2)

imaxdiv_t

Include: <inttypes.h>.

```
typedef struct {
    intmax_t    quot; /* Quotient */
    intmax_t    rem;  /* Remainder */
} imaxdiv_t;
```

It is the type of the value returned by the imaxdiv(3) function.

Conforming to: C99 and later; POSIX.1-2001 and later.

See also: imaxdiv(3)

intmax_t

Include: <stdint.h>. Alternatively, <inttypes.h>.

A signed integer type capable of representing any value of any signed integer type supported by the implementation. According to the C language standard, it shall be capable of storing values in the range [INTMAX_MIN, INTMAX_MAX].

The macro INTMAX_C() expands its argument to an integer constant of type intmax_t.

The length modifier for intmax_t for the printf(3) and the scanf(3) families of

functions is j; resulting commonly in %jd or %ji for printing intmax_t values.

Conforming to: C99 and later; POSIX.1-2001 and later.

Bugs: intmax_t is not large enough to represent values of type __int128 in imple?

mentations where __int128 is defined and long long is less than 128 bits wide.

See also: the uintmax_t type in this page.

intN_t

Include: <stdint.h>. Alternatively, <inttypes.h>.

int8_t, int16_t, int32_t, int64_t

A signed integer type of a fixed width of exactly N bits, N being the value speci?

fied in its type name. According to the C language standard, they shall be capable

of storing values in the range [INTN_MIN, INTN_MAX], substituting N by the appro?

priate number.

According to POSIX, int8_t, int16_t, and int32_t are required; int64_t is only re?

quired in implementations that provide integer types with width 64; and all other

types of this form are optional.

The length modifiers for the intN_t types for the printf(3) family of functions are

expanded by macros of the forms PRIdN and PRIiN (defined in <inttypes.h>); result?

ing for example in %"PRId64" or %"PRIi64" for printing int64_t values. The length

modifiers for the intN_t types for the scanf(3) family of functions are expanded by

macros of the forms SCNdN and SCNiN, (defined in <inttypes.h>); resulting for exam?

ple in %"SCNd8" or %"SCNi8" for scanning int8_t values.

Conforming to: C99 and later; POSIX.1-2001 and later.

See also: the intmax_t, uintN_t, and uintmax_t types in this page.

intptr_t

Include: <stdint.h>. Alternatively, <inttypes.h>.

A signed integer type such that any valid (void *) value can be converted to this

type and back. According to the C language standard, it shall be capable of stor?

ing values in the range [INTPTR_MIN, INTPTR_MAX].

The length modifier for intptr_t for the printf(3) family of functions is expanded

by the macros PRIdPTR and PRIiPTR (defined in <inttypes.h>); resulting commonly in

%"PRIdPTR" or %"PRIiPTR" for printing intptr_t values. The length modifier for

intptr_t for the scanf(3) family of functions is expanded by the macros SCNdPTR and

SCNiPTR, (defined in <inttypes.h>); resulting commonly in %"SCNdPTR" or %"SCNiPTR"

for scanning intptr_t values.

Conforming to: C99 and later; POSIX.1-2001 and later.

See also: the uintptr_t and void * types in this page.

lconv

Include: <locale.h>.

```
struct lconv {                  /* Values in the "C" locale: */
    char   *decimal_point;      /* "." */
    char   *thousands_sep;      /* "" */
    char   *grouping;           /* "" */
    char   *mon_decimal_point;  /* "" */
    char   *mon_thousands_sep;  /* "" */
    char   *mon_grouping;       /* "" */
    char   *positive_sign;      /* "" */
    char   *negative_sign;      /* "" */
    char   *currency_symbol;    /* "" */
    char    frac_digits;        /* CHAR_MAX */
    char    p_cs_precedes;      /* CHAR_MAX */
    char    n_cs_precedes;      /* CHAR_MAX */
    char    p_sep_by_space;     /* CHAR_MAX */
    char    n_sep_by_space;     /* CHAR_MAX */
    char    p_sign_posn;        /* CHAR_MAX */
    char    n_sign_posn;        /* CHAR_MAX */
    char   *int_curr_symbol;    /* "" */
    char    int_frac_digits;    /* CHAR_MAX */
    char    int_p_cs_precedes;  /* CHAR_MAX */
    char    int_n_cs_precedes;  /* CHAR_MAX */
    char    int_p_sep_by_space; /* CHAR_MAX */
    char    int_n_sep_by_space; /* CHAR_MAX */
    char    int_p_sign_posn;    /* CHAR_MAX */
    char    int_n_sign_posn;    /* CHAR_MAX */
};
```

Contains members related to the formatting of numeric values. In the "C" locale, its members have the values shown in the comments above.

Conforming to: C11 and later; POSIX.1-2001 and later.

See also: setlocale(3), localeconv(3), charsets(5), locale(7)

ldiv_t

Include: <stdlib.h>.

```
typedef struct {
    long    quot; /* Quotient */
    long    rem;  /* Remainder */
} ldiv_t;
```

It is the type of the value returned by the ldiv(3) function.

Conforming to: C99 and later; POSIX.1-2001 and later.

See also: ldiv(3)

lldiv_t

Include: <stdlib.h>.

```
typedef struct {
    long long   quot; /* Quotient */
    long long   rem;  /* Remainder */
} lldiv_t;
```

It is the type of the value returned by the lldiv(3) function.

Conforming to: C99 and later; POSIX.1-2001 and later.

See also: lldiv(3)

off_t

Include: <sys/types.h>. Alternatively, <aio.h>, <fcntl.h>, <stdio.h>, <sys/mman.h>, <sys/stat.h.h>, or <unistd.h>.

Used for file sizes. According to POSIX, this shall be a signed integer type.

Versions: <aio.h> and <stdio.h> define off_t since POSIX.1-2008.

Conforming to: POSIX.1-2001 and later.

Notes: On some architectures, the width of this type can be controlled with the feature test macro _FILE_OFFSET_BITS.

See also: lseek(2), mmap(2), posix_fadvise(2), pread(2), truncate(2), fseeko(3), lockf(3), posix_fallocate(3), feature_test_macros(7)

pid_t

Include: <sys/types.h>. Alternatively, <fcntl.h>, <sched.h>, <signal.h>, <spawn.h>, <sys/msg.h>, <sys/sem.h>, <sys/shm.h>, <sys/wait.h>, <termios.h>,

<time.h>, <unistd.h>, or <utmpx.h>.

This type is used for storing process IDs, process group IDs, and session IDs. Ac?

cording to POSIX, it shall be a signed integer type, and the implementation shall

support one or more programming environments where the width of pid_t is no greater

than the width of the type long.

Conforming to: POSIX.1-2001 and later.

See also: fork(2), getpid(2), getppid(2), getsid(2), gettid(2), getpgid(2),

kill(2), pidfd_open(2), sched_setscheduler(2), waitpid(2), sigqueue(3), creden?

tials(7),

ptrdiff_t

Include: <stddef.h>.

Used for a count of elements, and array indices. It is the result of subtracting

two pointers. According to the C language standard, it shall be a signed integer

type capable of storing values in the range [PTRDIFF_MIN, PTRDIFF_MAX].

The length modifier for ptrdiff_t for the printf(3) and the scanf(3) families of

functions is t; resulting commonly in %td or %ti for printing ptrdiff_t values.

Conforming to: C99 and later; POSIX.1-2001 and later.

See also: the size_t and ssize_t types in this page.

regex_t

Include: <regex.h>.

```
typedef struct {
    size_t  re_nsub; /* Number of parenthesized subexpressions. */
} regex_t;
```

This is a structure type used in regular expression matching. It holds a compiled

regular expression, compiled with regcomp(3).

Conforming to: POSIX.1-2001 and later.

See also: regex(3)

regmatch_t

Include: <regex.h>.

```
typedef struct {
    regoff_t    rm_so; /* Byte offset from start of string
                          to start of substring */
    regoff_t    rm_eo; /* Byte offset from start of string of
```

```
                the first character after the end of
                substring */
} regmatch_t;
```

This is a structure type used in regular expression matching.

Conforming to: POSIX.1-2001 and later.

See also: regexec(3)

regoff_t

Include: <regex.h>.

According to POSIX, it shall be a signed integer type capable of storing the largest value that can be stored in either a ptrdiff_t type or a ssize_t type.

Versions: Prior to POSIX.1-2008, the type was capable of storing the largest value that can be stored in either an off_t type or a ssize_t type.

Conforming to: POSIX.1-2001 and later.

See also: the regmatch_t structure and the ptrdiff_t and ssize_t types in this page.

sigevent

Include: <signal.h>. Alternatively, <aio.h>, <mqueue.h>, or <time.h>.

```
struct sigevent {
   int          sigev_notify; /* Notification type */
   int          sigev_signo;  /* Signal number */
   union sigval    sigev_value;  /* Signal value */
   void         (*sigev_notify_function)(union sigval);
                        /* Notification function */
   pthread_attr_t *sigev_notify_attributes;
                        /* Notification attributes */
};
```

For further details about this type, see sigevent(7).

Versions: <aio.h> and <time.h> define sigevent since POSIX.1-2008.

Conforming to: POSIX.1-2001 and later.

See also: timer_create(2), getaddrinfo_a(3), lio_listio(3), mq_notify(3)

See also the aiocb structure in this page.

siginfo_t

Include: <signal.h>. Alternatively, <sys/wait.h>.

typedef struct {

int si_signo; /* Signal number */

int si_code; /* Signal code */

pid_t si_pid; /* Sending process ID */

uid_t si_uid; /* Real user ID of sending process */

void *si_addr; /* Address of faulting instruction */

int si_status; /* Exit value or signal */

union sigval si_value; /* Signal value */

} siginfo_t;

Information associated with a signal. For further details on this structure (in?

cluding additional, Linux-specific fields), see sigaction(2).

Conforming to: POSIX.1-2001 and later.

See also: pidfd_send_signal(2), rt_sigqueueinfo(2), sigaction(2), sigwaitinfo(2),

psiginfo(3)

sigset_t

Include: <signal.h>. Alternatively, <spawn.h>, or <sys/select.h>.

This is a type that represents a set of signals. According to POSIX, this shall be

an integer or structure type.

Conforming to: POSIX.1-2001 and later.

See also: epoll_pwait(2), ppoll(2), pselect(2), sigaction(2), signalfd(2), sigpend?

ing(2), sigprocmask(2), sigsuspend(2), sigwaitinfo(2), signal(7)

sigval

Include: <signal.h>.

union sigval {

int sigval_int; /* Integer value */

void *sigval_ptr; /* Pointer value */

};

Data passed with a signal.

Conforming to: POSIX.1-2001 and later.

See also: pthread_sigqueue(3), sigqueue(3), sigevent(7)

See also the sigevent structure and the siginfo_t type in this page.

size_t

Include: <stddef.h> or <sys/types.h>. Alternatively, <aio.h>, <glob.h>, <grp.h>,

<iconv.h>, <monetary.h>, <mqueue.h>, <ndbm.h>, <pwd.h>, <regex.h>, <search.h>, <signal.h>, <stdio.h>, <stdlib.h>, <string.h>, <strings.h>, <sys/mman.h>, <sys/msg.h>, <sys/sem.h>, <sys/shm.h>, <sys/socket.h>, <sys/uio.h>, <time.h>, <unistd.h>, <wchar.h>, or <wordexp.h>.

Used for a count of bytes. It is the result of the sizeof operator. According to the C language standard, it shall be an unsigned integer type capable of storing values in the range [0, SIZE_MAX]. According to POSIX, the implementation shall support one or more programming environments where the width of size_t is no greater than the width of the type long.

The length modifier for size_t for the printf(3) and the scanf(3) families of func? tions is z; resulting commonly in %zu or %zx for printing size_t values.

Versions: <aio.h>, <glob.h>, <grp.h>, <iconv.h>, <mqueue.h>, <pwd.h>, <signal.h>, and <sys/socket.h> define size_t since POSIX.1-2008.

Conforming to: C99 and later; POSIX.1-2001 and later.

See also: read(2), write(2), fread(3), fwrite(3), memcmp(3), memcpy(3), memset(3), offsetof(3)

See also the ptrdiff_t and ssize_t types in this page.

ssize_t

Include: <sys/types.h>. Alternatively, <aio.h>, <monetary.h>, <mqueue.h>, <stdio.h>, <sys/msg.h>, <sys/socket.h>, <sys/uio.h>, or <unistd.h>.

Used for a count of bytes or an error indication. According to POSIX, it shall be a signed integer type capable of storing values at least in the range [-1, SSIZE_MAX], and the implementation shall support one or more programming environ? ments where the width of ssize_t is no greater than the width of the type long.

Glibc and most other implementations provide a length modifier for ssize_t for the printf(3) and the scanf(3) families of functions, which is z; resulting commonly in %zd or %zi for printing ssize_t values. Although z works for ssize_t on most im? plementations, portable POSIX programs should avoid using it?for example, by con? verting the value to intmax_t and using its length modifier (j).

Conforming to: POSIX.1-2001 and later.

See also: read(2), readlink(2), readv(2), recv(2), send(2), write(2)

See also the ptrdiff_t and size_t types in this page.

suseconds_t

Include: <sys/types.h>. Alternatively, <sys/select.h>, or <sys/time.h>.

Used for time in microseconds. According to POSIX, it shall be a signed integer type capable of storing values at least in the range [-1, 1000000], and the imple? mentation shall support one or more programming environments where the width of suseconds_t is no greater than the width of the type long.

Conforming to: POSIX.1-2001 and later.

See also: the timeval structure in this page.

time_t

Include: <time.h> or <sys/types.h>. Alternatively, <sched.h>, <sys/msg.h>, <sys/select.h>, <sys/sem.h>, <sys/shm.h>, <sys/stat.h>, <sys/time.h>, or <utime.h>.

Used for time in seconds. According to POSIX, it shall be an integer type.

Versions: <sched.h> defines time_t since POSIX.1-2008.

Conforming to: C99 and later; POSIX.1-2001 and later.

See also: stime(2), time(2), ctime(3), difftime(3)

timer_t

Include: <sys/types.h>. Alternatively, <time.h>.

Used for timer ID returned by timer_create(2). According to POSIX, there are no defined comparison or assignment operators for this type.

Conforming to: POSIX.1-2001 and later.

See also: timer_create(2), timer_delete(2), timer_getoverrun(2), timer_settime(2)

timespec

Include: <time.h>. Alternatively, <aio.h>, <mqueue.h>, <sched.h>, <signal.h>, <sys/select.h>, or <sys/stat.h>.

```
struct timespec {
   time_t  tv_sec;  /* Seconds */
   long    tv_nsec; /* Nanoseconds */
};
```

Describes times in seconds and nanoseconds.

Conforming to: C11 and later; POSIX.1-2001 and later.

See also: clock_gettime(2), clock_nanosleep(2), nanosleep(2), timerfd_gettime(2), timer_gettime(2)

timeval

Include: <sys/time.h>. Alternatively, <sys/resource.h>, <sys/select.h>, or

<utmpx.h>.

struct timeval {

time_t tv_sec; /* Seconds */

suseconds_t tv_usec; /* Microseconds */

};

Describes times in seconds and microseconds.

Conforming to: POSIX.1-2001 and later.

See also: gettimeofday(2), select(2), utimes(2), adjtime(3), futimes(3), timer?

add(3)

uid_t

Include: <sys/types.h>. Alternatively, <pwd.h>, <signal.h>, <stropts.h>,

<sys/ipc.h>, <sys/stat.h>, or <unistd.h>.

A type used to hold user IDs. According to POSIX, this shall be an integer type.

Conforming to: POSIX.1-2001 and later.

See also: chown(2), getuid(2), geteuid(2), getresuid(2), getpwnam(2), creden?

tials(7)

uintmax_t

Include: <stdint.h>. Alternatively, <inttypes.h>.

An unsigned integer type capable of representing any value of any unsigned integer

type supported by the implementation. According to the C language standard, it

shall be capable of storing values in the range [0, UINTMAX_MAX].

The macro UINTMAX_C() expands its argument to an integer constant of type uint?

max_t.

The length modifier for uintmax_t for the printf(3) and the scanf(3) families of

functions is j; resulting commonly in %ju or %jx for printing uintmax_t values.

Conforming to: C99 and later; POSIX.1-2001 and later.

Bugs: uintmax_t is not large enough to represent values of type unsigned __int128

in implementations where unsigned __int128 is defined and unsigned long long is

less than 128 bits wide.

See also: the intmax_t type in this page.

uintN_t

Include: <stdint.h>. Alternatively, <inttypes.h>.

uint8_t, uint16_t, uint32_t, uint64_t

An unsigned integer type of a fixed width of exactly N bits, N being the value specified in its type name. According to the C language standard, they shall be capable of storing values in the range [0, UINTN_MAX], substituting N by the appro? priate number.

According to POSIX, uint8_t, uint16_t, and uint32_t are required; uint64_t is only required in implementations that provide integer types with width 64; and all other types of this form are optional.

The length modifiers for the uintN_t types for the printf(3) family of functions are expanded by macros of the forms PRIuN, PRIoN, PRIxN, and PRIXN (defined in <inttypes.h>); resulting for example in %"PRIu32" or %"PRIx32" for printing uint32_t values. The length modifiers for the uintN_t types for the scanf(3) fam? ily of functions are expanded by macros of the forms SCNuN, SCNoN, SCNxN, and SCNXN (defined in <inttypes.h>); resulting for example in %"SCNu16" or %"SCNx16" for scanning uint16_t values.

Conforming to: C99 and later; POSIX.1-2001 and later.

See also: the intmax_t, intN_t, and uintmax_t types in this page.

uintptr_t

Include: <stdint.h>. Alternatively, <inttypes.h>.

An unsigned integer type such that any valid (void *) value can be converted to this type and back. According to the C language standard, it shall be capable of storing values in the range [0, UINTPTR_MAX].

The length modifier for uintptr_t for the printf(3) family of functions is expanded by the macros PRIuPTR, PRIoPTR, PRIxPTR, and PRIXPTR (defined in <inttypes.h>); re? sulting commonly in %"PRIuPTR" or %"PRIxPTR" for printing uintptr_t values. The length modifier for uintptr_t for the scanf(3) family of functions is expanded by the macros SCNuPTR, SCNoPTR, SCNxPTR, and SCNXPTR (defined in <inttypes.h>); re? sulting commonly in %"SCNuPTR" or %"SCNxPTR" for scanning uintptr_t values.

Conforming to: C99 and later; POSIX.1-2001 and later.

See also: the intptr_t and void * types in this page.

va_list

Include: <stdarg>. Alternatively, <stdio.h>, or <wchar.h>.

Used by functions with a varying number of arguments of varying types. The func? tion must declare an object of type va_list which is used by the macros

va_start(3), va_arg(3), va_copy(3), and va_end(3) to traverse the list of argu? ments.

Conforming to: C99 and later; POSIX.1-2001 and later.

See also: va_start(3), va_arg(3), va_copy(3), va_end(3)

void *

According to the C language standard, a pointer to any object type may be converted to a pointer to void and back. POSIX further requires that any pointer, including pointers to functions, may be converted to a pointer to void and back.

Conversions from and to any other pointer type are done implicitly, not requiring casts at all. Note that this feature prevents any kind of type checking: the pro? grammer should be careful not to convert a void * value to a type incompatible to that of the underlying data, because that would result in undefined behavior.

This type is useful in function parameters and return value to allow passing values of any type. The function will typically use some mechanism to know the real type of the data being passed via a pointer to void.

A value of this type can't be dereferenced, as it would give a value of type void, which is not possible. Likewise, pointer arithmetic is not possible with this type. However, in GNU C, pointer arithmetic is allowed as an extension to the standard; this is done by treating the size of a void or of a function as 1. A consequence of this is that sizeof is also allowed on void and on function types, and returns 1.

The conversion specifier for void * for the printf(3) and the scanf(3) families of functions is p.

Versions: The POSIX requirement about compatibility between void * and function pointers was added in POSIX.1-2008 Technical Corrigendum 1 (2013).

Conforming to: C99 and later; POSIX.1-2001 and later.

See also: malloc(3), memcmp(3), memcpy(3), memset(3)

See also the intptr_t and uintptr_t types in this page.

NOTES

The structures described in this manual page shall contain, at least, the members shown in their definition, in no particular order.

Most of the integer types described in this page don't have a corresponding length modi? fier for the printf(3) and the scanf(3) families of functions. To print a value of an in?

teger type that doesn't have a length modifier, it should be converted to intmax_t or uintmax_t by an explicit cast. To scan into a variable of an integer type that doesn't have a length modifier, an intermediate temporary variable of type intmax_t or uintmax_t should be used. When copying from the temporary variable to the destination variable, the value could overflow. If the type has upper and lower limits, the user should check that the value is within those limits, before actually copying the value. The example below shows how these conversions should be done.

Conventions used in this page

In "Conforming to" we only concern ourselves with C99 and later and POSIX.1-2001 and later. Some types may be specified in earlier versions of one of these standards, but in the interests of simplicity we omit details from earlier standards.

In "Include", we first note the "primary" header(s) that define the type according to ei? ther the C or POSIX.1 standards. Under "Alternatively", we note additional headers that the standards specify shall define the type.

EXAMPLES

The program shown below scans from a string and prints a value stored in a variable of an integer type that doesn't have a length modifier. The appropriate conversions from and to intmax_t, and the appropriate range checks, are used as explained in the notes section above.

```
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
int
main (void)
{
    static const char *const str = "500000 us in half a second";
    suseconds_t us;
    intmax_t    tmp;
    /* Scan the number from the string into the temporary variable */
    sscanf(str, "%jd", &tmp);
    /* Check that the value is within the valid range of suseconds_t */
    if (tmp < -1 || tmp > 1000000) {
```

```
            fprintf(stderr, "Scanned value outside valid range!\n");
            exit(EXIT_FAILURE);
        }
        /* Copy the value to the suseconds_t variable 'us' */
        us = tmp;
        /* Even though suseconds_t can hold the value -1, this isn't
           a sensible number of microseconds */
        if (us < 0) {
            fprintf(stderr, "Scanned value shouldn't be negative!\n");
            exit(EXIT_FAILURE);
        }
        /* Print the value */
        printf("There are %jd microseconds in half a second.\n",
                (intmax_t) us);
        exit(EXIT_SUCCESS);
    }
```

SEE ALSO

feature_test_macros(7), standards(7)

COLOPHON

This page is part of release 5.10 of the Linux man-pages project. A description of the project, information about reporting bugs, and the latest version of this page, can be found at https://www.kernel.org/doc/man-pages/.