
# T-Kernel 2.0 Extension Specification

December 2012

**T-Engine Forum**

**http://www.t-engine.org/**

T-Kernel 2.0 Extension Specification (Ver.2.00.00)



For information about this specification, please contact the following:
T-Engine Forum Secretariat
In YRP Ubiquitous Networking Laboratory
28th Kowa Building, 2-20-1 Nishi-gotanda
Shinagawa, Tokyo
Japan 141-0031
+81-(0)-3-5437-0572 +81-(0)-3-5437-2399
office@t-engine.org

Note

In this specification, POSIX means Portable Operating System Interface, specifically the so-called UNIX system Operating System Interface defined in the following standards.

ISO/IEC/IEEE 9945
Information technology - Portable Operationg System Interface (POSIX)
Base Specifications, Issue 7

The standard C library referred to in the chapter for the Standard C Compatible Library means the above POSIX as well as the library functions defined in the following standard.

JIS X 3010:2003 (ISO/IEC 9899:1999) Programming Language C

Considering the programming ease and portability at some degree of affinity with POSIX, this specification follows the standard C library specifications almost as is so that programs using the standard C library may easily be ported.
This specification quotes some descriptions from the above standards with permission from IEC.
This specification is an extension of the underlying T-Kernel 2.0, which is an operating system of a totally different nature from POSIX.
This specification does not guarantee the compatibility with POSIX.
In addition, it is not guaranteed that C language programs written as per this specification are compliant with the JIS C standard.

IEC: International Electrotechnical Commission
ISO: International Organization for Standardization
JIS: Japanese Industrial Standards

The function declarations, structure definitions, and numerical values in this specification are written according to the C language syntax.

Table of Contents

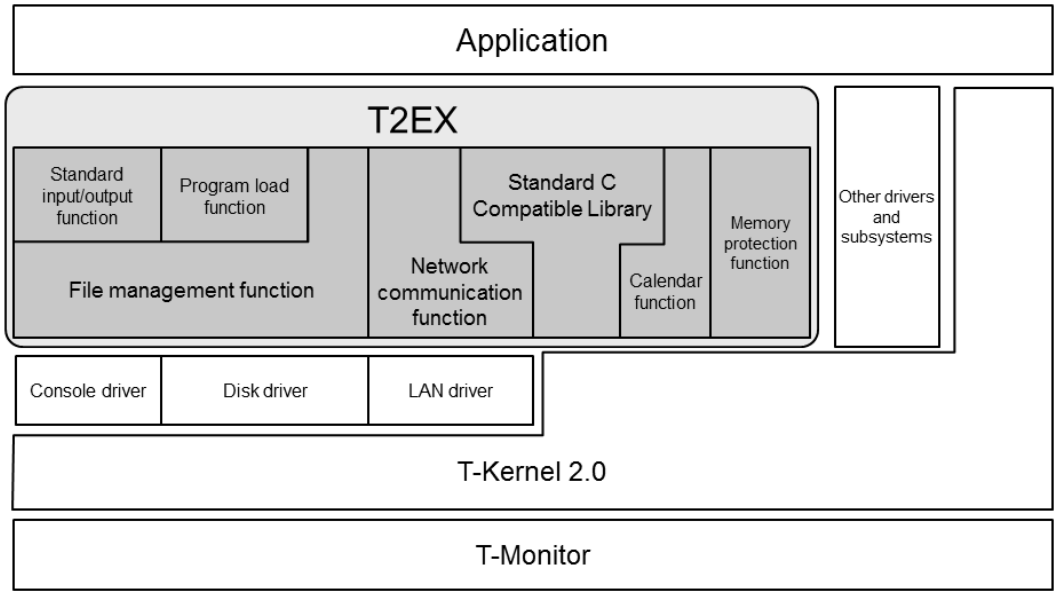
# Chapter 1 T-Kernel 2.0 Extension Overview

## 1.1. Overview

T-Kernel 2.0 Extension ("T2EX" below) is the T-Kernel 2.0 feature expansion program.
Making the most of light-weight, high-speed, and real-time properties of T-Kernel, a real-time operating system, T2EX is designed as a light-weight extension to realize advanced embedded systems.

The functions provided to applications by T2EX consist of extended SVCs (extended system call), library functions, and macros.
These functions and application interfaces altogether are called API (Application Programming Interface).
The T2EX specification is defined by the T2EX API.
Each one of the individual system calls, library functions, and macros in the API is called an "API call".
The whole of the file management function API calls, for example, is called the "file management function API".

Figure 1.1 shows the software architecture including T2EX.
This extension is positioned as an extension available as an additional T-Kernel 2.0 function (addon), allowing you to build application programs using both the T-Kernel 2.0 and T2EX APIs.




(Figure1.1: T-Kernel 2.0 Extension configuration and positioning)

To support the advanced embedded system development, T2EX provides the following functions.

- Memory Protection Function
- File Management Function
- Network Communication Function
- Calendar Function
- Program Load Function
- Standard C Compatible Library

Each function is provided as a module by functional unit, allowing you to use some of them and unuse (remove) any unnecessary function modules as needed.

The T-Engine forum implements standard T2EX codes on T-Kernel 2.0 on the T-Engine reference board as an extension and discloses the source codes together with the specifications.
This implementation is called the "T2EX reference implementation". This specification also describes how implementation-dependent items are implemented in the T2EX reference implementation.

## 1.2. T-Kernel 2.0 Extension Features

Following the existing T-Kernel 2.0 performances and maximizing its light-weight, high-speed, and real-time properties, T2EX is designed to meet the requirement for additional functions in embedded systems that has become larger and more sophisticated.
The main features of T2EX are described below.

Processless, light-weight extension

T2EX does not provide the process function to make the entire system light-weight.

The process function is effective when developing programs for relatively large systems on a module-by-module basis. However, it does not go together with light-weight and high-speed properties of the entire system, due to large overhead in inter-process communication and resource switching resulting from the split resource.

Aiming at lightweight, T2EX assumes the entire system to be built without processes, allowing various extended functions, including file management and network communication, to be used by directly calling them from the task instead of a process.

This direct availability of extended functions from the T-Kernel 2.0 task allows you to realize advanced functions while making the most use of the T-Kernel 2.0 real-time property.

Effective memory protection function independent of virtual memory

General process-based memory protections are based on multiple logical spaces with large overhead in execution when switching the spaces.

T-Kernel 2.0 task-based programs often exchange information among multiple tasks via variables (memory). Simply divided logical spaces would increase the inter-process communication overhead.

Aiming at lightweight, T2EX provides an effectively feasible two-level ring protection at the system and user levels.

This ensures necessary and sufficient reliability for a relatively complex case in a specific-purpose embedded system, which is a main target of this extension.

Modular

T2EX provides many functions including file management and network communication. Each of them is separated on a module-by-module basis, allowing you to use only the selected necessary modules and remove the rest.

This can reduce use of RAM and ROM by unnecessary functions.

More affinity with standard C and POSIX specifications

Many of the functions targeted by this extension, such as C language standard input/output and network communication functions, have the de facto standard.

The T2EX API design orients toward the optimum format as a T-Kernel task-based programming API while it considers affinity with the standard C library and the POSIX specification in terms of code reusability and reduced learning cost.

Specific elaborations include:

- Integration of the standard C library and POSIX specification error numbers (errno) into the error codes (ER type)

  The T-Kernel 2.0 error codes are extended so that the error numbers (errno) can be handled as is in the T-Kernel 2.0 error code system.

This eliminates confusions when using the standard C and POSIX specification error numbers (errno_t type) together with the error codes (ER type).

- Provision of Standard C Compatible Library

Many C language standard library functions are provided as is, including the standard input/output, except no error number (errno) output.

This greatly facilitates source codes to be shared with many other platforms.

- API call names and styles (e.g., arguments) based on the POSIX specification

  This uses the specifications based on the function names and styles of the POSIX specification, which is the de facto standard in the file management and network communication functions.

This can help code reusability with a different environment and reduce the learning cost.

- Avoidance of collision in the symbol or macro name of a functions that has a different behavior from the original specification

Any function or macro provided in T2EX with the same name as in the C language standard library or the POSIX specification does not have a different external specification, except no error number (errno) output.

This can avoid the misunderstanding of the function behavior and a mistake when the source code is reused in a different platform.

Thread-safe guarantee

The standard C library and the POSIX specification contain thread-unsafe functions, which may cause an unexpected bug depending on a race condition if misused.

In T2EX, all the API calls are thread-safe for the users to able to avoid this sort of mistake with no awareness.

## 1.3. Relationship with T-Kernel 2.0

T2EX is an additional T-Kernel 2.0 function (addon) and provides the function sets available to T-Kernel 2.0 application programs.

This means that T2EX is not an extension that builds up a different API layer (e.g., process environment) on top of T-Kernel 2.0 but provides additional functions available in T-Kernel 2.0 API-based application programs.

For instance, you can define the following function which uses both the T-Kernel 2.0 and T2EX APIs to be directly used from a T-Kernel 2.0 program.
In this example, tk_set_flg, tk_wai_flg, and tk_ext_tsk are T-Kernel 2.0 system calls, and so_recv and fprintf are API calls provided by T2EX.
This feature allows you to build an advanced real-time system that directly uses T-Kernel 2.0's synchronization and communication functions even in advanced applications including file input/output and network communication.

```
void socketReaderTask( INT stacd, void* exinf )
{
        ER ercd;
        UINT flgptn;

        for (;;) {
                /* receives data from network */
                ercd = so_recv(sd, data, sizeof(data));
                if (ercd <= 0) {
                        fprintf(stderr, "Task terminated¥n");
                        break;
                }
                len = ercd;

                /* notifies completion of reception from network */
                tk_set_flg(flgid, FLG_DATA_PRODUCED);

                /* waits until the received data are used */
                tk_wai_flg(flgid, FLG_DATA_CONSUMED, (TWF_ANDW|TWF_BITCLR), &flgptn, TMO_FEVR);
        }

        tk_ext_tsk();
}
```

Use of certain T-Kernel 2.0 functions combined with T2EX is restricted.
See Section 2.4 for details.

## 1.4. Relationship with POSIX API Specification

Many of the APIs provided by T2EX are designed to have affinity with the POSIX specification wherever possible.
Still, there are many differences because T2EX is an extension designed specifically for embedded systems with a greatly simplified and lightened POSIX API specification.

Regarding the POSIX API-derived functions in this extension, the differences from the original specification are summarized as follows.

- Process

T2EX is a system assuming the T-Kernel 2.0 task programming and thus does not provide a function equivalent to a process in POSIX.
While original POSIX functions are executed in process contexts, the T2EX system has a single execution context which corresponds to processes in POSIX.

For instance, file descriptors are assigned on a per-process basis in POSIX.
Even if a file descriptor is closed in a process, one may still be open and held in a different process.
In T2EX, file descriptor information is common to all the tasks, and thus the file descriptor status never looks differently among different tasks.

- Thread

In T-Kernel 2.0, the task function is used instead of the thread function in POSIX.
Any POSIX thread-compatible APIs are not provided.

- Signal

Any POSIX-compatible signal functions are not provided.
In T-Kernel 2.0, the task exception handling function is available instead.

- User and group

Any user- and group-related functions for POSIX are not provided.
The user and group access control is not performed as well.
This means that an operation that might require superuser privilege in POSIX can be executed from any task in T2EX.

- File
The POSIX specification is designed to abstract the file, socket, device, and synchronization object on the disk as a (broad) file to be operable through the file API.
On the other hand, T2EX does not provide such an abstracted file function. They are clearly separated as independent modules by functional unit.

Let us name the file management function and the network communication function. The descriptor is defined independently for each function, and an API set is provided the same way.
(Example: fs_read, so_read)

- Error number
In the POSIX specification, the error number is output to the thread-local variable errno.
T2EX has obsoleted the error number output to the errno variable for uniformity with T-Kernel 2.0 APIs and improvement to performance efficiency. Instead, it returns the error code that has extended to include the POSIX error number.(For details, see Section 2.5.)

Some API calls include detailed restrictions and changes.
For details, see the API call definitions.

## 1.5. Dependencies between Function Modules

The functions provided in T2EX are implemented as modules divided into functional units, allowing you to select and use some function modules as needed.
Note that there are dependencies between certain modules. When you select modules, they must meet these restrictions.
Some functions depend on device drivers and thus need the correspondent device drivers when you use them.

The dependencies between modules and between modules and device drivers are described below (outline is shown in Figure 1.1).

Dependencies between function modules
- Among the standard input/output functions, the input/output to the console file requires the file management function.
- The program load function requires the file management function.
- In the Standard C Compatible Library, the network-related function (e.g., inet.h) requires the network communication function, and the time-related function (e.g., time.h) requires the calendar function.

Dependencies between function modules and device drivers
- The file management function requires the driver (console driver or disk driver) for the physical device targeted for input and output.
- The network communication requires the LAN driver.

For the console driver, disk driver, and LAN driver specifications, see the "T-Engine Standard Device Driver Specification."

# Chapter 2 Common Rules

## 2.1. Data Types

As T2EX is used as an additional function (add-on) to T-Kernel 2.0, it inherits the data types defined in the T-Kernel 2.0 specification.
The following data types defined in section 3.1 of the T-Kernel 2.0 Specification are used as basic data types in this extension as well.

```
typedef signed char          B;     /* signed 8-bit integer */
typedef signed short         H;     /* signed 16-bit integer */
typedef signed long          W;     /* signed 32-bit integer */
typedef signed long long     D;     /* signed 64-bit integer */
typedef unsigned char       UB;     /* unsigned 8-bit integer */
typedef unsigned short      UH;     /* unsigned 16-bit integer */
typedef unsigned long       UW;     /* unsigned 32-bit integer */
typedef unsigned long long  UD;     /* unsigned 64-bit integer */

typedef char                VB;     /* 8-bit data without an intended type */
typedef short               VH;     /* 16-bit data without an intended type */
typedef long                VW;     /* 32-bit data without an intended type */
typedef long long           VD;     /* 64-bit data without an intended type */
typedef void                *VP;    /* pointer to data without an intended type */
```

Note: In T-Kernel 1.0, the data type of exinf or other was VP. In T-Kernel 2.0, VP is not used in principle in consideration of the CONST modifier, and the definition of data type "void *" is written directly in the code, instead of using VP.
The definition of VP is retained in T-Kernel 2.0 and T2EX for compatibility, but a new use of VP is not recommended.

```
typedef volatile B          _B;     /* volatile declaration */
typedef volatile H          _H;
typedef volatile W          _W;
typedef volatile D          _D;
typedef volatile UB         _UB;
typedef volatile UH         _UH;
typedef volatile UW         _UW;
typedef volatile UD         _UD;

typedef signed int          INT;    /* signed integer of processor bit width, 32 bits or more */
typedef unsigned int        UINT;   /* unsigned integer of processor bit width, 32 bits or more */

typedef INT                 ID;     /* general ID */
typedef W                   MSEC;   /* general time (in milliseconds) */

typedef void            (*FP)();    /* general function address */
typedef INT             (*FUNCP)(); /* general function address */

#define LOCAL           static      /* local symbol definition */
#define EXPORT                      /* global symbol definition */
#define IMPORT          extern      /* global symbol reference */

/*
 * Boolean values
 *      TRUE = 1 is defined, but any value other than 0 is logically TRUE.
 *      A decision such as bool == TRUE must be avoided for this reason.
 *      Instead use bool != FALSE.
 */
typedef INT             BOOL;
#define TRUE            1           /* true */
#define FALSE           0           /* false */

/*
 * TRON character codes
 */
typedef UH              TC;         /* TRON character codes */
#define TNULL           ((TC)0)     /* TRON code string termination */
```

In addition to these basic data types from T-Kernel 2.0, various type definitions are added for use in the functions provided by T2EX.
Most of them comply with the standard C library and the POSIX specification, which facilitates programs or program codes from other environment to be ported or interoperated.
All of the added data types, including the ones specific to T2EX, have POSIX style.(Example:

pm_entry_t)

Typical basic data types additionally defined in T2EX are as follows:

```
typedef signed char         int8_t;      /* signed 8-bit integer */
typedef signed short        int16_t;     /* signed 16-bit integer */
typedef signed long         int32_t;     /* signed 32-bit integer */
typedef signed long long    int64_t;     /* signed 64-bit integer */
typedef unsigned char       uint8_t;     /* unsigned 8-bit integer */
typedef unsigned short      uint16_t;    /* unsigned 16-bit integer */
typedef unsigned long       uint32_t;    /* unsigned 32-bit integer */
typedef unsigned long long  uint64_t;    /* unsigned 64-bit integer */

typedef signed long         intptr_t;    /* signed integer of pointer bit width */
typedef unsigned long       uintptr_t;   /* unsigned integer of pointer bit width */

typedef int                 errno_t;     /* integer representing an error number (described later) */

typedef unsigned long       size_t;      /* unsigned integer representing size */
typedef signed long         ssize_t;     /* signed integer representing size */


typedef signed long         time_t;      /* integer type representing time in seconds */

struct timeval {                         /* structure representing time in microseconds */
                long        tv_sec;      /* second */
                long        tv_usec;     /* microsecond */
};
```

For other types, see the sections in Chapter 4 or latter.

There are definitions of both the basic data types from T-Kernel 2.0 and the ones from the standard C library and the POSIX specification. This means that duplicate types with similar meanings are defined.
For example, the UB type and the uint8_t type have a nearly identical meaning.
It is recommended that you use them according to the semantics of this specification and the T-Kernel 2.0 specification as much as possible.
For example, uint32_t is not appropriate for the flag patterns of event flag, and UINT type should be used instead.
In contrast, it is recommended that you use off64_t for the second argument (file offset) of the file seek (fs_lseek64), in accordance with the function prototype.

## 2.2. String

The T2EX specification defines that a string represented as a char array must be encoded in UTF-8.
All the API calls defined in this extension work based on this definition.

For example, a file name specified as the argument when opening a file (fs_open) must be a UTF-8 string.
Even if you specify a string using an incorrect encoding such as ISO-8859-1 and Shift-JIS, the specified file name is interpreted as a UTF-8 string.

## 2.3. Valid Context

All API calls provided by T2EX are defined to be available in task or quasi-task portions.
These API calls cannot be used in task-independent portions.

It is possible that the implementation checks the context at runtime and returns an E_CTX error for execution in an illegal context. However, whether or not to perform such check shall be implementation-dependent, and an implementation that does not guarantee a behavior (undefined) shall be allowed.

Exceptionally, the following API calls can be used from in task-independent portions:

- fs_break (stop file operation)
- so_break (stop socket operation)

## 2.4. Usage of T-Kernel 2.0 API

As described in section 2.3, T2EX API can be used in a task or quasi-task portion.
However, when performing task management functions and task-dependent synchronization functions for tasks using T2EX, there are the following restrictions:

- tk_ter_tsk
  Unavailable.
  If used, the system behavior is not guaranteed.

- tk_sus_tsk, tk_rsm_tsk, tk_frsm_tsk
  Unavailable.
  If used, the system behavior is not guaranteed.

- tk_sig_tev, tk_wai_tev(_u)
  The task event numbers 1 to 4 are reserved for use in the T2EX system and must not be used.
  Applications can use the task event numbers 5 to 8.

- tk_dis_wai, tk_ena_wai
  Unavailable.
  If used, the system behavior is not guaranteed.

- tk_ras_tex
  Unavailable.
  If used, the system behavior is not guaranteed.

The other T-Kernel 2.0 API calls are available even when they are used with T2EX API and are guaranteed to work according to the specification.
When tk_rel_wai is issued for a task using T2EX API calls, it works as follows:

- tk_rel_wai
  Available.
  tk_rel_wai works as follows when it is executed during execution of a T2EX API call.

  - The WAITING state of the waiting T2EX API call may be released, but not necessarily.
    If the WAITING state is released, the T2EX API call returns EX_INTR, and tk_rel_wai returns E_OK.
    If the WAITING state is not released, tk_rel_wai returns E_OBJ.

In addition to tk_rel_wai, the dedicated wait-release API calls, fs_break and so_break, are provided for T2EX's file management and network communication functions, respectively.
These API calls can be used to safely release the WAITING state of the limited functions of T2EX.

## 2.5. Error Codes

As T2EX provides additional functions such as file management function and network communication function, it defines an error code system to extend the error code of T-Kernel 2.0.
In order to achieve a high affinity with the standard C library and the POSIX specification in this extension, the error number (errno) is integrated with the T-Kernel error code (ER type) system as follows:

- Defines EC_ERRNO as main error code
- Defines error codes prefixed with EX_ as the error codes corresponding to errno in the POSIX specification, as follows:
  - Uses error number names with the prefix E replaced with EX_, for error code names
  - Has EC_ERRNO as main error code
  - Has the value of errno as sub error code

For example, for the error number EBADF indicating that the file descriptor is illegal, the corresponding T2EX error code is defined as follows:

```
#define EX_BADF           ERCD(EC_ERRNO, EBADF)
```

In addition, the macros, ERRNO and ERRNOtoER, are defined to mutually convert error codes (ER type) added in T2EX and error numbers (errno).

```
#define ERRNO(er)         (MERCD(er) == EC_ERRNO ? SERCD(er) : 0)
#define ERRNOtoER(eno)    (ERCD(EC_ERRNO, (eno)))
```

For the list of definitions of added error codes, see Section 8.8.
The error codes prefixed with E_ which are defined in T-Kernel 2.0 can also be used in applications using T2EX.

With introduction of the error codes corresponding to the POSIX error numbers, several error codes

with similar meanings coexist.
For example, two types of error, E_BUSY and EX_BUSY, are defined to indicate that the target resource is being used.
One of them should be used depending on the type of the target resource.
The former should be used if the busy object is a synchronized object of T-Kernel 2.0, and the latter if a file or network is targeted.
In middleware library or other, it is also recommended that you use the T-Kernel 2.0 error codes and the error codes added in T2EX in a uniform manner as much as possible, to prevent confusions.

## 2.6. Thread-Safety

This specification defines the thread-safety of functions under the multitask environment as follows:

Thread-Safe function
A function is said to be thread-safe if it is guaranteed that its concurrent execution does not cause any problems under the condition that the memory space directly or indirectly specified by the passed arguments is not referred or changed from outside during a function call.
Otherwise, the function is said to be non-thread-safe.

The thread-safe nature is also defined for the T2EX API calls in the same manner.

As T2EX is a system assuming the task-based programming on T-Kernel, it is very important that the T2EX API calls are thread-safe in terms of reliability of the system.
For this reason, all API calls provided in T2EX are thread-safe.

Though T2EX includes many API calls based on the POSIX specification or the standard C library, alternative thread-safe API calls are provided for non-thread-safe API calls.
For example, localtime() in the standard C library is non-thread-safe and not provided in T2EX. T2EX provides the alternative thread-safe API calls, localtime_r() and localtime_r_eno().

As described in the definition of the thread-safety of API calls in this specification, a T2EX API call does not guarantee the behavior if the memory space directly or indirectly specified by the arguments is referred or changed from outside during the API call.
For example, when two tasks A and B execute the following processings concurrently for a char* type global variable "a", the behaviors are not defined.
In such case, you have the responsibility to fulfill the thread-safe conditions, for example, by preventing overlapped memory space of the arguments or using the exclusion control.

Task A
```
strcpy(a + 3, "aaaa");
```

Task B
```
strcpy(a + 2, "bbbbbb");
```

# Chapter 3 Memory Protection Function

## 3.1 Overview

T2EX provides two levels of the memory protection function based on the memory protection model of T-Kernel 2.0.

The T2EX system consists of a single logical address space and does not have task-specific spaces. This means that there is a one-to-one mapping between logical address and physical address, but it is not always necessary that they match.

Under this condition, the stability and reliability of the whole system can be improved by protecting the memory space used by the OS kernel (T2 and T2EX) and system programs (some device drivers and tasks) from user applications.

## 3.2 Memory Protection Model

### 3.2.1 Protection Levels

T-Kernel 2.0 has four protection levels from 0 to 3. In T2EX, these levels are divided into two, and the two levels of the memory protection, privileged level and user level, are provided.

- Privileged level
  This level is equivalent to the protection levels 0 to 1 in T-Kernel 2.0, and used when a task portion is executed at the protection levels 0 to 1, or when a nontask portion (task-independent portion, quasi-task portion, and so on) is executed.
  A privileged level program operates in the privileged mode of CPU.

- User level
  This level is equivalent to the protection levels 2 to 3 in T-Kernel 2.0, and used when a task portion is executed at the protection levels 2 to 3.
  A user level program operates in the user mode of CPU.

The level at which a T-Kernel 2.0 or T2EX API call can be invoked is specified using the system configuration information, independent of the memory protection boundary.
In T2EX, each protection level has the following usages:

| Protection level | Usage |
|---|---|
| 0 | Kernel, subsystems, device drivers, etc. |
| 1 | System application tasks |
| 2 | User application tasks (T-Kernel and T2EX API available) |
| 3 | [* Reserved] User application tasks (T-Kernel and T2EX API not available) |

(*) Reserved for use by upper OS functions such as processes, and not used in T2EX

As shown in the table, T-Kernel 2.0 and T2EX API can only be used in protection levels between 0 and 2. When they are used from tasks with protection level of 3, the correct behavior is not guaranteed.

### 3.2.2 Memory Access Privileges

Table 3.1 shows the access privileges to various memory spaces from privileged-level and user-level programs.
In this table, "R", "W", and "EX" indicate the read, write, and execute privileges respectively, and "-" indicates inaccessible.

Table 3.1 Memory Access Privileges in T-Kernel 2.0 Extension

| Memory space | Privileged level | User level |
|---|---|---|
| Privileged-level program area | R,EX | - |
| Privileged-level read/write static data area | R,W | - |
| Privileged-level read only static data area | R | - |
| User-level program area | R,EX | R,EX |
| User-level read/write static data area | R,W | R,W |
| User-level read only static data area | R | R |
| Privileged-level dynamically allocated memory block | R,W | - |
| User-level dynamically allocated memory block | R,W | R,W |

| | | |
|---|---|---|
| Unused (unallocated) memory | - | - |

Note that Table 3.1 shows the standard access privileges to the memory spaces. An implementation are allowed to have different access privileges than ones shown in Table 3.1, in order to accommodate the hardware limitations or the following cases.

- Add user-level read access privilege (R) to privileged-level static data area or privileged-level dynamically allocated memory block
By allowing a user-level application to directly read privileged-level data area such as device drivers, data can be passed without memory copy.

- Add execution privilege (EX) to data area or dynamically allocated memory block
To achieve JIT (Just-In-Time) compiling or other technologies, it is necessary to assign the execution privilege to data area.

- Assign write access privilege (W) to program area
A program which works by self-rewriting needs the write access privilege to the program area.

### 3.2.3 Static and Dynamic Memory Protections

There are two types of memory protection in T2EX, static memory protection and dynamic memory protection.

Static Memory Protection
This protects the memory spaces shown in Table 3.1, except the dynamically allocated memory block, by setting fixed access privileges to the memory space address determined at system generation.
Usually, the access privilege for each address area (section) is configured by the linker script at system generation, and the specific setting method depends on the implementation.

Dynamic Memory Protection
This protects the memory space dynamically obtained by T-Kernel 2.0 API call by setting access privileges at the specified protection level.
This is the dynamically allocated memory block among the memory spaces shown in Table 3.1.

### 3.2.4 Relationship with Memory-Related Functions of T-Kernel 2.0

Under the T2EX memory protection function, the API calls related to the memory protection in T-Kernel 2.0 works as follows:

tk_set_tsp
Does nothing and returns E_OK for a valid parameter.

SetTaskSpace
Only access privilege information is modified when a valid parameter is given, and task space remains unchanged.
Can be used from privileged protection level at runtime only.

LockSpace, UnlockSpace
Does nothing and returns E_OK for a valid parameter.

MapMemory
Works as per the T-Kernel 2.0 Specification if NULL is specified for paddr.
When a valid physical address is specified in paddr, E_OK is returned after it is converted to a logical address, without setting access privileges for the specified memory region.
(Access privilege settings to physical address regions are meant to be done through static memory protection configuration.)

CnvPhysicalAddr
Works as per the T-Kernel 2.0 Specification.

tk_cre_mpf, tk_cre_mpl, tk_get_smb
Allocates the memory accessible only from privileged-level programs if TA_RNG0 to 2 are specified for attr, and allocates the memory accessible from any level if TA_RNG3 is specified.

ChkSpaceBstrR, ChkSpaceBstrRW, ChkSpaceR, ChkSpaceRE, ChkSpaceRW, ChkSpaceTstrR, ChkSpaceTstrRW
Works as per the T-Kernel 2.0 Specification.

Kmalloc, Kcalloc, Krealloc, Kfree,
Vmalloc, Vcalloc, Vrealloc, Vfree

Allocates the memory at the level specified by the T2EX system configuration information (section A.1).
Can be used from privileged protection level at runtime only.

Smalloc, Scalloc, Srealloc, Sfree
Allocates the memory at the user level.
Can be used from both privileged and user protection level.

CreateLock, DeleteLock, Lock, Unlock
Can be used from privileged protection level at runtime only.
For the use in user level, following substitute functions are available:
CreateLock, DeleteULock, ULock, and UUnlock.

CreateMLock, DeleteMLock, MLock, MLockTmo, MLockTmo_u, MUnlock
Can be used from privileged protection level at runtime only.
For the use in user level, following substitute functions are available:
CreateMLock, DeleteUMLock, UMLock, UMLockTmo, UMLockTmo_u, and UMUnlock.

DI, EI, isDI, SetIntMode, EnableInt, DisableInt, ClearInt, CheckInt
Can be used from privileged protection level at runtime only.

StartPhysicalTimer, StopPhysicalTimer, GetPhysicalTimerCount, DefinePhysicalTimerHandler, GetPhysicalTimerConfig
Can be used from privileged protection level at runtime only.

GetSpaceInfo
Works as per the T-Kernel 2.0 Specification.

SetMemoryAccess
Works as per the T-Kernel 2.0 Specification.

in_d, in_w, in_h, in_b, out_w, out_d, out_w, out_h, out_b
Can be used from privileged protection level at runtime only.

WaitUsec, WaitNsec
Can be used from privileged protection level at runtime only.

### 3.2.5 Handling of Memory Protection Violation

The "memory protection violation" means a condition that a task or quasi-task portion being executed tries to access a memory space inaccessible from that level (privileged level or user level).
If a "memory protection violation" is detected, a memory protection violation exception occurs. This exception is managed internally by T2EX.

If a memory protection violation exception has occurred during execution of a task portion of a user-level task, the exception is notified to an exception handling task managed internally by T2EX. This task is called a "system exception handling task".
The system exception handling task has the priority 1 and works at the protection level 0. It calls the handling function TaskMemFaultHdr() for a memory protection violation exception in its context.

On the other hand, if a memory protection violation exception has occurred in a task-independent portion, a quasi-task portion, or in a task portion of a privileged-level task, that exception is handled by the handling function RawMemFaultHdr() which is called in the context of the task-independent portion.

You can modify these functions to perform your own exception handling such as file system synchronization and system reset, as necessary.

In addition to the standard configuration described above, T2EX supports the simple configuration which does not use system exception handling tasks.
In this configuration, a system exception handling task is not generated, and all memory protection exceptions are handled by the function RawMemFaultHdr() which is called in the context of task-independent portions.
Like with the standard configuration, you can modify this function to perform your own exception handling as necessary.

The exception handling functions have the following forms:

- void TaskMemFaultHdr(MemFaultInfo* fault);

  Handles the memory protection violation exception indicated by fault.
  It is called in the context of a system exception handling task.

```
  The contents of fault are:
        ID      tskid           ID of the task that caused the exception
        UW      vecno           Interrupt vector number of the exception
        UW      excinfo         Information about the protection violation exception
(implementation-dependent)
        void*   excaddr         Address that caused the exception (access target)
        ---(Other implementation-dependent parameters may be added beyond this point.)---

- void RawMemFaultHdr(MemFaultInfo* fault);

  Handles the memory protection violation exception indicated by fault.
  It is called in the context of a task-independent portion.

  The contents of fault are the same as those of TaskMemFaultHdr().
```

# Chapter 4 File Management Functions

## 4.1 Overview

Hereafter, a program module which provides file management functions in T2EX will be referred to as "file manager".

The file manager provides functions for accessing file systems consisting of tree-structured directories and files.
The API name prefix is "fs_" (file system).

The file manager provides an API set similar to the file input/output function of the POSIX specification. One large difference is that return codes of the T2EX APIs represent error codes if they are negative.

It also supports 64-bit large files which allow a 64-bit file size and a file offset to be directly specified. However, the size of data that can be read or written at a time is limited to 32 bits. It provides 32-bit and 64-bit APIs separately and they can be used together.

Besides the APIs that support the POSIX specification time data formats to handle time, APIs that support the T-Kernel 2.0 time data formats (SYSTIM, SYSTIM_U) are provided.

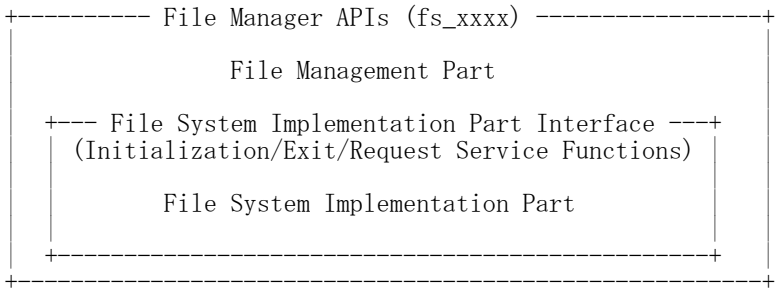



Figure 4.1 File Manager Structure

The file manager is structured as shown in Figure 4.1.

As explained in the sections below, a file manager API is interpreted by the file management part at first. The file management part then calls an appropriate function in the file system implementation part which handles files in a particular file system such as FAT and NTFS.

A file system implementation part is a set of program codes to manage a certain file system. It uses device drivers for storage devices to provide file management functions according to the file systems such as FAT, exFAT, NTFS, ext2, and files for flash ROM.

In T2EX, multiple file system implementation parts can be used. In addition to the "Basic FAT file system implementation part" which is available as a standard file system implementation part of T2EX, user-defined file system implementation parts can be used.

## 4.2 Concepts and Terminology

- File system format

In this document, a logical format used to place and manage a file structure on a device, which is generally known as FAT, NTFS, or ext2, will be refered to as file system format.

- File system

Generally, this refers to a system of files laid out on a device consisting of tree-structured directories and files, or generic name of the function to manage the system of files.

In this document, it refers to a whole directory tree on a device organized in one file system format, and this is the unit connected to the directory tree of a system.

For example, when the term "file system" is used as in the "size of a file system", "statistics of a file system", "synchronization of a file system", and "a read-only file system", it means the unit, connected to the system, organized in one file system format.

- File system implementation part (FIMP)

A set of program codes that handle access to a file system format as access to the data on an actual device is called "file system implementation part (FIMP)", and can be written by users and dynamically registered to the file manager.

T2EX provides "the Basic file system implementation part (Basic FAT FIMP)" to manage the FAT file system format. It is registered to the system and becomes usable after the initialization of the file manager.

To handle different file system formats, a FIMP to handle such different file system formats can be written and registered to the system.

Multiple FIMPs can be registered for the same file system format, and they can be switched per device or medium.

A FIMP becomes usable after it is registered to the system (fs_regist) and is attached to an actual target device (fs_attach).

These steps import the directory tree on the device into the system and file operations using path names become possible such as open (fs_open), read (fs_read), and write (fs_write).

FIMPs do not necessarily have one-to-one relationship with file system formats.
For example, a program in which one FIMP handles multiple file system formats can also be written.

- Basic FAT file system implementation part (Basic FAT FIMP)

A FIMP that supports the FAT file system format. It is provided as a standard built-in .

It can handle the FAT12, FAT16, and FAT32 file system formats and also uses VFAT long file names. It can access both devices (media) with and without partition information. It, however, can use only the primary partitions, and does not support the extended partitions.

The Basic FAT FIMP is registered after initializing the file manager with fs_main(). Registration by fs_regist() is not necessary.

To release a registered FIMP, unregister it (fs_unregist). The Basic FAT FIMP can also be unregistered.

- Directory

Special file which includes entries to identify files or other directories.

A directory can contain other directories, making a hierarchical directory structure.

Files and directories on one device form a tree structure as a whole.

- File name

Name to identify a file or directory in the file system.

A file name can contain unicode (UTF-8) characters other than the followings:

'/', ':', '¥', '*', '?', '"', '<', '>', '|', and '¥0' (null character)

The maximum length of file name is defined by NAME_MAX.
However, the maximum length of file name may be shorter than this definition in some FIMPs.

There are special file names:
"." Represents the current position in the tree structure.
".." Represents the directory immediately above the current position in the tree structure.

- Root directory

A virtual directory located at the top of all the other directories.

Immediately under the root directory, there is the top-most directory of each of the currently connected devices as a virtual subdirectory.

- Current directory

This refers to the directory which is the current reference position of file operations.

It is the start position of the search for a file or directory when a relative path name, which

described later, is given.

In T2EX, there is only one current directory in the whole system, not per task. That is, all tasks share the one current directory.

- Connection point

Subdirectory name used when connecting a file system on a device to the root directory.

When specifying a target device (or medium) by fs_attach(), specify a connection point and the name of the device (device name) on which files are located to associate them.

One connection point is used to refer to one file system.

A connection point can contain up to 14 characters including 'a'-'z', 'A'-'Z', '0'-'9', and '_', but not '/'.

- Path name

String representation of a name indicating the position of a certain file or directory in a file system.

A path name is specified as a parameter when opening a file with fs_open(). It is also used as a parameter when creating a file (fs_creat()), retrieving file status (fs_stat(), etc.), renaming or relocating a file (fs_rename()), and deleting a file (fs_unlink()).

Unicode (UTF-8) is used as the character code of a path name.
The maximum length of a path name is defined by PATH_MAX.

A path name string indicates the directory tree path from the root directory '/' as the starting point to an appropriate file on the target device, separated with '/' as shown below.

```
/connection point/directory name 1/directory name 2/.../directory name n/file name
```

A path name that indicates an absolute position from the root directory is called an absolute path name.
An absolute path name starts with "/".

A path name that indicates a relative position from the current directory is called a relative path name.
A relative path name starts with "./", which is optional. A path name that does not start with "/" is considered a relative path name.

- Device name

In the file manager, this refers to the name of a device on which files exist.
Generally, it consists of:

- Type Name which specifies the device type
- Unit Letter which specifies a physical device
- Subunit Number which specifies a logical device

Although device name is formatted as Type+Unit+Subunit, Unit and subunit may be omitted on some devices.

Unicode (UTF-8) is used for a device name string.

- File system implementation part name (FIMP name)

The name of a FIMP. This name follows the file name convention.

The following names are pre-defined and reserved in the system:

"FIMP_FAT" Basic FAT FIMP
"FIMP_AUTODETECT" Reserved for future use (automatically detecting the file system format)

- File descriptor

Zero or positive integer newly assigned to each file to identify it after a file is opened.

A file descriptor is used for read, write, or other operations on the file.

The following file descriptors are pre-assigned in the system for special purposes:

| | | |
|---|---|---|
| STDIN_FILENO | 0 | Standard input |
| STDOUT_FILENO | 1 | Standard output |
| STDERR_FILENO | 2 | Standard error output |

These file descriptors are assigned to the console device in the system.
These file descriptors cannot be used for other files since they would be automatically reopened immediately if they are closed (fs_close).

- Disk cache

To efficiently write to or read from files located on a disk or other device, many implementations create a buffer area in the memory to manage files on the device via this buffer. In such implementations, the memory area used as a buffer is called a disk cache.

T2EX utilizes a disk cache to write to or read from files efficiently.

A write operation by fs_write() only transfers write data to the disk cache and does not guarantee the completion of writing data to the acutal disk.

Call fs_close(), fs_fsync(), or fs_fdatasync() to write the file data on the disk cache into the actual disk, flushing the data in the disk cache to the physical device.

- Synchronization with the physical device

Referred to as "a state in which the disk cache and the physical device are in sync", that the content of the disk cache is written to the physical device and the content of both are exactly the same.

Because the disk cache is placed on the volatile memory, there is a possibility that data is lost due to system down unexpectedly. Therefore, it is necessary to synchronize with the physical device at appropriately timing.

For synchronization with the physical device, O_SYNC open mode is provided. Also API calls, fs_sync, fs_fsync and fs_fdatasync are provided.

- Synchronous write

An operation that also writes to the physical device as well as disk cache at the same time in order to synchronize the disk cache and the physical device at write time.

- File metadata

File management information such as the last access time, the last modified time, the last status change time, and the file size, except the file's data itself.

- File date and time specification

File metadata includes the last access time, the last modified time, and the last status change time. Besides the APIs that support the time_t type to handle these times, APIs that support the SYSTIM and SYSTIM_U types are provided.

Note that the exact meaning and the precision of a time about the file status may vary depending on the type of the file system.

Example:
- All of the last access time, the last modified time, and the last status change time may not be maintained.
- The time precision may depend on the file system.

In the case of the FAT file system, the time resolution is two seconds for modified time, and one day for access time. The last status change time does not exist. It is considered to be the same as the last modified time.

- File mode

This refers to the file type and the access mode, and is represented by mode_t type data.

mode_t
Data type to show the file type and the access mode.
The following bit masks and macros can be used for this type of data:

S_IFMT 0170000 Bit mask for the file type field

The following symbols are defined for these types:

| | | |
|---|---|---|
| S_IFBLK | 0060000 | Block device |
| S_IFCHR | 0020000 | Character device |
| S_IFREG | 0100000 | Regular file |
| S_IFDIR | 0040000 | Directory |
| S_IFLINK | 0120000 | Symbolic link |

Available file types may vary depending on the type of the file system.
* S_IFLINK is not used in the FAT file system.

The following macros are provided to determine the file type:
"m" is a value of st_mode in "stat" or other structure.
Each macro returns a non-zero value for a result of true, or 0 for false.

| | |
|---|---|
| S_ISBLK(m) | Block device |
| S_ISCHR(m) | Character device |
| S_ISDIR(m) | Directory |
| S_ISREG(m) | Regular file |
| S_ISLNK(m) | Symbolic link |

The following mode bits are defined for the access permissions:

| | | |
|---|---|---|
| S_IRWXU | 0700 | Read, write, and execution/search by owner |
| S_IRUSR | 0400 | Read permission by owner |
| S_IWUSR | 0200 | Write permission by owner |
| S_IXUSR | 0100 | Execution/search permission by owner |
| S_IRWXG | 070 | Read, write, and execution/search by group |
| S_IRGRP | 040 | Read permission by group |
| S_IWGRP | 020 | Write permission by group |
| S_IXGRP | 010 | Execution/search permission by group |
| S_IRWXO | 07 | Read, write, and execution/search by others |
| S_IROTH | 04 | Read permission by others |
| S_IWOTH | 02 | Write permission by others |
| S_IXOTH | 01 | Execution/search permission by others |
| S_ISUID | 04000 | Set user ID at runtime |
| S_ISGID | 02000 | Set group ID at runtime |
| S_ISVTX | 01000 | Sticky bit |

*The execution/search attribute is used as the execution permission when the file is a normal file, or the search permission to search the directory for files when the file is a directory.

All of these mode settings may not be supported on the type of the connected file system. In that case, unsupported settings are simply ignored.

Note that T2EX has no concept of owner, group, and others. So T2EX does not differentiate them. For this reason, when a file operation such as opening of a file or reading/writing of data is performed, it is processed as follows:

- If any of the read permission bits is set, a read operation is allowed.
- If any of the write permission bits is set, a write operation is allowed.
- If any of the execution/search permission bits is set, an execution/search operation is allowed.

- File creation flag

Flag for creating a file, used in "flag" of fs_open().

| | |
|---|---|
| O_CREAT | Creates the file if it does not exists |
| O_EXCL | Opens the file exclusively |
| O_TRUNC | Truncates the file content |

- File status flag

Flag to show the open status of a file used in fs_open() or fs_fcntl().
It is treated as an attribute of the file descriptor.

| | |
|---|---|
| O_APPEND | Append mode |
| O_NONBLOCK | Non-blocking mode |
| O_SYNC | Guarantees a synchronized file I/O at write |

- File access mode

Flag to show the access mode of a file used in fs_open() or fs_fcntl().
It is treated as an attribute of the file descriptor.

| | |
|---|---|
| O_RDONLY | Opened for read-only |
| O_RDWR | Opened for read/write |
| O_WRONLY | Opened for write-only |

The file access mode is of the mode_t data type.
The following file access mode mask is used to retrieve the file access mode from the mode_t type data:

| | |
|---|---|
| O_ACCMODE | File access mode mask |

- File offset

This refers to the current position (read/write start position) for writing to/reading from a file.
A file offset is represented by the following data types:

| | |
|---|---|
| off_t | Offset by a 32-bit integer |
| off64_t | Offset by a 64-bit integer |

4.3 Unsupported Functions

The file managerin T2EX supports a subset of the file function in POSIX.
In T2EX, no file owners nor user groups with access right exist since there is not processes and the concept of user. Additionally, symbolic links are unsupported.
Therefore, the following functions specified in POSIX are not provided:

File access privilege and owner/group related
- umask()
- chown()
- fchown()
- lchown()

Hard link
- link()
- linkat()

Symbolic link related
- symlink()
- symlinkat()
- readlink()
- lstat()

Replicate file descriptor
- dup()
- dup2()

File lock
- flock()
- lockf()

Other
- chroot()

When a file on a device contains user/group information and their access privilege information, if it is accessed and updated by T2EX, the original information of user/group/access privilege may be lost.
If handling the information of users and groups in a file managed by a FIMP is necessary, handle it by a specially implemented function in the FIMP using fs_ioctl() or others.

4.4 API

If a return code of a file manager API call is negative, it is a T2EX extended error code.

4.4.1 fs_main - Initializes and exits the file manager

C Language Interface

```
#include <t2ex/fs.h>

ER      er = fs_main(INT ac, UB* arg[]);
```

Parameter

| | | |
|---|---|---|
| INT | ac | number of elements in arg[] or a negative value |
| UB* | arg[] | array of argument strings |

Return Parameter

| | | |
|---|---|---|
| ER | er | error code |

Error Code

| | |
|---|---|
| E_OK | Normal completion |

Description

This function initializes (ac >= 0) or terminates (ac < 0) the file manager.

At the time of initialization, a number of strings can be passed to arg[] as arguments, and the total count of strings is ac.
The content of "arg" is implementation-dependent. These argument strings are not used in the T2EX reference implementation.

### 4.4.2 fs_regist - Registers a file system implementation part (FIMP)

C Language Interface

```
#include <t2ex/fs.h>

ER er = fs_regist(const char *fimpnm, const fs_fimp_t *fimp, void *info);
```

Parameter

| | | |
|---|---|---|
| const char* | fimpnm | FIMP name |
| const fs_fimp_t | fimp | FIMP definition information |
| void* | info | any parameter |

Return Parameter

| | | |
|---|---|---|
| ER | er | error code |

Error Code

| | |
|---|---|
| E_OK | Normal completion |
| EX_INVAL | Illegal parameter |
| EX_EXIST | Specified FIMP name is already registered |
| EX_ENOBUFS | Number of FIMPs exceeds the system limit |
| Others | Error code returned by registfn() of the FIMP |

Description

fs_regist() uses the name specified by "fimpnm" to register the FIMP specified by "fimp" to the system.
"fimp" is the pointer to the FIMP structure which holds the request service function of the FIMP.
"info" is a parameter to be passed to the initialization function (fimp->registfn) of the FIMP. It is used to pass the initialization information defined for each FIMP.

If fs_regist() successfully registers the FIMP, it returns E_OK.
The Basic FAT FIMP is registered when the file manager is initialized.
The registration using fs_regist()is not necessary to use the Basic FAT FIMP.

To manage a FAT file system using a FIMP other than the Basic FAT FIMP, register the different implementation part using fs_regist() with a name other than "FIMP_FAT" in "fimpnm". Then attach it with fs_attach().

Alternatively, the Basic FAT FIMP can be unregisted with fs_unregist(), then another FIMP with the name "FIMP_FAT" can be registered for use.

fs_fimp_t is a structure holding the definition information of the FIMP. For user implementation of a FIMP, see Section 4.5 "File System Implementation Part".

See Also

fs_unregist, fs_attach, fs_detach

### 4.4.3 fs_unregist - Unregisters a file system implementation part (FIMP)

C Language Interface

```
#include <t2ex/fs.h>

ER er = fs_unregist(const char *fimpnm);
```

Parameter

| | | |
|---|---|---|
| const char* | fimpnm | FIMP name |

Return Parameter

| | | |
|---|---|---|
| ER | er | error code |

Error Code

| | |
|---|---|
| E_OK | Normal completion |
| EX_INVAL | Illegal parameter |
| EX_NOENT | Non-registered FIMP is specified |
| EX_BUSY | Specified FIMP is in use |
| Others | Error code returned by unregistfn() of the FIMP |

Description

fs_unregist() unregisters the FIMP that has been registered with the name specified in "fimpnm". When the FIMP with the name specified by "fspnm" is attached to a device, use fs_detach() to detach the connection before unregistering it.

If fs_unregist() successfully unregisters the FIMP, it returns E_OK.
If the specified FIMP is attached to a device by fs_attach(), fs_unregist() returns EX_BUSY.

See Also

fs_regist, fs_attach, fs_detach

### 4.4.4 fs_attach - Connects a file system

C Language Interface

```
#include <t2ex/fs.h>

ER er = fs_attach(const char* devnm, const char* connm, iconst char *fimpnm, int flags, void *info);
```

Parameter

| | | |
|---|---|---|
| const char* | devnm | device name to connect |
| const char* | connm | connection point |
| const char* | fimpnm | FIMP name |
| int | flag | connect flag |
| | DEV_FLAG_READONLY | read-only device |
| | 0 | other device |
| void* | info | any parameter |

Return Parameter

| | | |
|---|---|---|
| ER | er | error code |

Error Code

| | |
|---|---|
| E_OK | Normal completion |
| EX_INVAL | Illegal parameter |
| EX_EXIST | Specified connection point is already registered |
| EX_ENOBUFS | Number of connections exceeded the system limit |
| EX_NOENT | Non-registered FIMP is specified |
| EX_NOTSUP | Unsupported FIMP |
| EX_BUSY | Device is connected |
| EX_IO | I/O error |
| Others | Error code returned by attachfn() of the FIMP |

Description

fs_attach() attaches the device specified by "devnm" to the FIMP with the name specified by "fimpnm" and connects it to the system directory tree using the "connm" connection point.

"devnm" is a name of the device formatted in the format specified by the FIMP.
It can be NULL for a FIMP that does not use a device.

"connm" is a connection point used to connect the FIMP.
Once fs_attach() is executed, a file on the connected device can be identified by using a path name "/<connm>/...".
For example, when a string , "usr", is specified for connm, files on the connected device will be placed under the directory "/usr".

"fimpnm" is the name of registered FIMP.

"flags" is used for specifying additional attributes for the connection such as read-only.
If DEV_FLAG_READONLY is specified here, the device will be connected as read-only even if it is writable ifself.
Set "flags" to 0 if there is no such attribute.

"info" is a parameter to be passed to the initialization function (fimp->attachfn) of the FIMP.
Applications can use it to pass initialization information specific to the FIMP.
Set it to 0 when connecting the Basic FAT FIMP.

fs_attach() returns E_OK when it is successfully connected.

If "FIMP_AUTODETECT" is specified as "fimpnm", a registered FIMP will be automatically selected to handle the data format on the connected device.

At this time, however, this function is not supported. If "FIMP_AUTODETECT" is specified as "fimpnm", fs_attach() returns EX_NOTSUP.

See Also

fs_detach

### 4.4.5 fs_detach - Disconnects a file system

C Language Interface

```
#include <t2ex/fs.h>

ER er = fs_detach(const char* connm);
```

Parameter

| | | |
|---|---|---|
| const char* | connm | connection point to disconnect |

Return Parameter

| | | |
|---|---|---|
| ER | er | error code |

Error Code

| | |
|---|---|
| E_OK | Normal completion |
| EX_BUSY | Device is in use |
| EX_FAULT | Illegal address in argument |

| | |
|---|---|
| EX_NOENT | Non-registered connection point |

Description

fs_detach() detaches the connected device specified in "connm".
If an open file exists on the device to be detached, fs_detach() returns the EX_BUSY error.

See Also

fs_attach

### 4.4.6 fs_open - Opens or creates a file

C Language Interface

```
#include <t2ex/fs.h>

int fd = fs_open(const char* path, int oflags, mode_t mode);
```

Parameter

| | | |
|---|---|---|
| const char* | path | file path name to open |
| int | oflags | open flag of file or directory |
| mode_t | mode | file mode of the file to be created (when O_CREAT) |

Return Parameter

| | | |
|---|---|---|
| int | fd | file descriptor<br>or error code |

Error Code

| | |
|---|---|
| EX_ACCES | Access privilege error<br>- File exists, but the operation specified by "oflag" is not allowed<br>- File does not exist, and the parent directory does not have write attribute |
| EX_EXIST | O_CREAT and O_EXCL are specified, but the file already exists |
| EX_INTR | Aborted by fs_break() |
| EX_INVAL | Illegal parameter |
| EX_IO | I/O error |
| EX_ISDIR | "path" is a directory for O_WRONLY or O_RDWR |
| EX_NAMETOOLONG | File name is too long<br>- Directory or file name part in "path" is too long (NAME_MAX at maximum)<br>- Whole path length is too long (PATH_MAX at maximum). |
| EX_NFILE | Number of open files exceeds the system limit |
| EX_NOENT | File does not exist<br>- O_CREAT is not specified, and the file does not exist<br>- O_CREAT is specified, and the path part in the path name is not found<br>- "path" string is empty |
| EX_NOSPC | Insufficient device space |
| EX_NOTDIR | Not a directory<br>- "path" contains something other than a directory in the prefix part<br>- O_DIRECTORY is specified, and "path" is not a directory |
| EX_ROFS | O_WRONLY, O_RDWR, O_CREAT, or O_TRUNC is specified for a file, and it exists in a read-only file system |

Description

This function opens the file or directory with the path name specified by "path" and returns its file descriptor.
The file descriptor is used to refer to the file in subsequent operations on it.

As a file descriptor, the minimum value that is not currently opened and used is returned.

The file offset, which indicates the current position of the file for reading and/or writing, is set to the beginning of the file.

The open flag is specified as follows:

```
        oflags := (O_RDONLY|O_RDWR|O_WONLY)
                        | [O_APPEND] | [O_CREAT| [O_EXCL]] | [O_DIRECTORY] | [O_NONBLOCK]
                        | [O_SYNC] | [O_TRUNC]
```

The read/write mode is either one of the followings:

O_RDONLY
Opens for read-only.
O_RDWR
Opens for read/write.
O_WRONLY
Opens for write-only.

The following values can be added with a logical OR:

O_APPEND
Before each writing (fs_write), the file offset is set to the end of file.

O_CREAT
Creates a new file if it does not exist.
If the file exists, this flag has no effect when O_EXCL is not specified.
The "mode" value is used as the mode for creating a file.

O_DIRECTORY
If "path" is not a directory, EX_NOTDIR error is returned.

O_EXCL
This is used with O_CREAT. If the file already exists, an error is returned.
The file existence check and the creation of a new file when it does not exist are atomic against other tasks which execute fs_open() with the same path name.
If O_CREAT is not specified, the result is undefined.

O_NONBLOCK
Opens in non-blocking mode if a non-blocking open is supported.
When a file is opened in non-blocking mode, fs_open() exits without waiting till the device becomes ready to use. Subsequent behaviors are device-dependent.

O_SYNC
Performs synchronous write operations to guarantee the file integrity.
The synchronous write means that a write operation on this file descriptor is performed not only to the disk cache but also to the disk as a physical device, blocking until the write operation is completed.

O_TRUNC
If the file exists, it is a normal file, and it is successfully opened with O_RDWR or O_WRONLY. The file size is truncated to 0.
If the file is a terminal device file such as the console, this has no effect.
If the file is not a normal file, the effect is implementation-dependent.
If neither O_RDWR nor O_WRONLY is specified, the behavior is undefined.

See Also

fs_close

### 4.4.7 fs_close - Closes a file

C Language Interface

```
#include <t2ex/fs.h>

ER er = fs_close(int fd);
```

Parameter

| | | |
|---|---|---|
| int | fd | file descriptor to close |

Return Parameter

| | | |
|---|---|---|
| ER | er | error code |

Error Code

| | |
|---|---|
| E_OK | Normal completion |
| EX_BADF | "fd" is invalid |
| EX_INTR | Aborted by fs_break() |
| EX_IO | I/O error |

Description

This function closes the opened file or directory specified by the file descriptor "fd".
"fd" is released and can be reused by subsequent fs_open().

If the close operation is aborted by fs_break(), fs_close() returns EX_INTR, and the "fd" remains open.
If an I/O error occurs while reading from/writing to the file system during the close operation, fs_close() returns EX_IO, and the "fd" remains open.

See Also

fs_open

### 4.4.8 fs_lseek, fs_lseek64 - Changes the file read/write offset position

C Language Interface

```
#include <t2ex/fs.h>

off_t offs = fs_lseek(int fd, off_t offset, int whence);
off64_t offs64 = fs_lseek64(int fd, off64_t offset64, int whence);
```

Parameter

| | | | |
|---|---|---|---|
| int | fd | file descriptor | |
| off_t | offset | file offset | |
| off64_t | offset64 | file offset (64 bits) | |
| int | whence | how to specify offset | |
| | | SEEK_SET | Absolute position |
| | | SEEK_CUR | Relative position from the current position |
| | | SEEK_END | Relative position from the end of file |

Return Parameter

| | | |
|---|---|---|
| off_t | offs | changed file offset or error code |
| off64_t | offs64 | changed file offset (64 bits)<br>or error code |

Error Code

| | |
|---|---|
| EX_BADF | "fd" is invalid |
| EX_INVAL | Illegal parameter<br>- File offset is not defined for the specified file descriptor<br>- Changed file offset is a negative value<br>- Changed file offset exceeds the end of file |
| EX_OVERFLOW | Changed file offset cannot be represented by off_t or off64_t |

Description

These functions change the file offset of the opened file "fd" by the method specified by "whence".

SEEK_SET
Changes the offset to the position of "offset" or "offset64".

SEEK_CUR
Changes the offset to the position of the current position + ("offset" or "offset64").

SEEK_END
Changes the offset to the position of the file size + ("offset" or "offset64").

If a file that does not support position change such as the console device is specified, it returns the EX_INVAL error.

The file offset cannot exceed the end of file. If such a position is specified, it returns the EX_INVAL error.

### 4.4.9 fs_read - Reads from a file

C Language Interface

```
#include <t2ex/fs.h>

int nb = fs_read(int fd, void* buf, size_t count);
```

Parameter

| | | |
|---|---|---|
| int | fd | file descriptor |
| void* | buf | read buffer |
| size_t | count | number of bytes to read |

Return Parameter

| | | |
|---|---|---|
| int | nb | zero or more number of bytes that have been read<br>or error code |
| void* | buf | data that have been read |

Error Code

| | |
|---|---|
| EX_AGAIN | No data can be read without block (if O_NONBLOCK is set) |
| EX_BADF | "fd" is invalid |
| EX_INTR | Aborted by fs_break() |
| EX_IO | I/O error |
| EX_ISDIR | "fd" is a directory |
| EX_OVERFLOW | "fd" is a normal file, "count" is positive, and the start position exceeds the end of file |
| EX_NOBUFS | Insufficient system resource |
| EX_NOMEM | Insufficient memory |

Description

This function reads the "count" bytes of data from the file specified by "fd" into the buffer specified by "buf".

When "count" is 0, fs_read() checks for errors and returns 0 if no error occurs.

For a file that supports seek such as a file on the disk, fs_read() starts reading from the file offset position of "fd".
The file offset is incremented by the number of bytes actually read from the file.

For a file that does not support seek such as the console, fs_read() starts reading from the current position.
The file offset is undefined for such a file.

Data is not transferred exceeding the current end of file.
If the start position exceeds the end of file, fs_read() returns 0.

If "count" is larger than SSIZE_MAX, the behavior is implementation-dependent.

When fs_read() reads from a file that supports the non-blocking read but there is currently no available data:
- it returns EX_AGAIN error if O_NONBLOCK in the file status flag is set.
- it forces the calling task to wait until some data becomes available if O_NONBLOCK in the file status flag is cleared.

If the read operation is aborted by fs_break() before data is read, fs_read() returns EX_INTR error.
If the read operation is aborted by fs_break() after even a single byte of data was read, fs_read() returns the number of read bytes, and the file offset is also incremented by the number of bytes.

4.4.10 fs_write - Writes to a file

C Language Interface

```
#include <t2ex/fs.h>

int nb = fs_write(int fd, const void* buf, size_t count);
```

Parameter

| | | |
|---|---|---|
| int | fd | file descriptor |
| const void* | buf | write buffer |
| size_t | count | number of bytes to write |

Return Parameter

| | | |
|---|---|---|
| int | nb | number of bytes that have been written<br>or error code |

Error Code

| | |
|---|---|
| EX_AGAIN | No data can be written without block (if O_NONBLOCK is set) |
| EX_BADF | "fd" is invalid |
| EX_FBIG | Position exceeds the limit of file size |
| EX_INTR | Aborted by fs_break() |
| EX_IO | I/O error |
| EX_NOBUFS | Insufficient system resource |
| EX_NOSPC | Insufficient device space |

Description

This function writes the "count" bytes of data from the buffer specified by "buf" to the file specified by "fd".

If "count" is 0 and the file is a normal file, fs_write() checks for errors and returns 0 if no error occurs.
If "count" is 0 and the file is not a normal file, the behavior is undefined.

For a file that supports seek, fs_write() starts writing data from the file offset position of the file.
The file offset is incremented by the number of bytes actually written to the file.
For a normal file, when the last position of the written data is equal to or larger than the file size, the file size becomes the position + 1.

For a file that does not support seek such as the console, fs_write() always starts writing from the current position.
The file offset is not defined for such a device.

If the file status flag O_APPEND is set, fs_write() always starts writing from the end of file by setting the file offset to the end of file before writing.

When the complete writing by fs_write() would exceed the limit of file size or the physical space of the media, it writes data up to the number of bytes for the available space and exits normally. If it attempts to write more data afterward, it cause an error.

If the write operation is aborted by fs_break() before data is written, fs_write() returns EX_INTR error.
If the write operation is aborted by fs_break() after even a single byte of data was written, fs_write() returns the number of written bytes, and the file offset is also incremented by the number of bytes.

If "count" exceeds SSIZE_MAX, the result is implementation-dependent.

When fs_write() writes to a file that supports the non-blocking write but cannot write data immediately:

- it does not force the calling task to wait if O_NONBLOCK in the file status flag is set. If some data can be written, fs_write() returns the number of written bytes. Otherwise, it returns EX_AGAIN.
- it forces the calling task to wait until data is accepted if O_NONBLOCK in the file status flag is cleared.

At the normal completion with written data (i.e., positive "count"), the last modified time and last

status change time of the file have been updated.

### 4.4.11 fs_stat, fs_fstat_us, fs_stat_ms, fs_fstat, fs_stat_ms, fs_stat_us, fs_stat64, fs_stat64_ms, fs_stat64_us, fs_fstat64, fs_stat64_ms, fs_fstat64_us - Retrieves the file status

C Language Interface

```
#include <t2ex/fs.h>

/* File size is 32 bits */
ER er = fs_stat(const char* path, struct stat* buf);
ER er = fs_stat_ms(const char* path, struct stat_ms* mbuf);
ER er = fs_stat_us(const char* path, struct stat_us* ubuf);

ER er = fs_fstat(int fd, struct stat* buf);
ER er = fs_fstat_us(int fd, struct stat_us* ubuf);
ER er = fs_fstat_ms(int fd, struct stat_ms* mbuf);


/* File size is 64 bits */
ER er = fs_stat64(const char* path, struct stat64* buf64);
ER er = fs_stat64_ms(const char* path, struct stat64_ms* mbuf64);
ER er = fs_stat64_us(const char* path, struct stat64_us* ubuf64);

ER er = fs_fstat64(int fd, struct stat64* buf64);
ER er = fs_fstat64_ms(int fd, struct stat64_ms* mbuf64);
ER er = fs_fstat64_us(int fd, struct stat64_us* ubuf64);
```

Parameter

| | | |
|---|---|---|
| const char* | path | path name of the file |
| int | fd | file descriptor |
| struct stat* | buf | file information retrieval buffer (time_t format) |
| struct stat_ms* | mbuf | file information retrieval buffer (SYSTIM format) |
| struct stat_us* | ubuf | file information retrieval buffer (SYSTIM_U format) |
| struct stat64* | buf64 | file information retrieval buffer (64-bit size, time_t format) |
| struct stat64_ms* | mbuf64 | file information retrieval buffer (64-bit size, SYSTIM format) |
| struct stat64_us* | ubuf64 | file information retrieval buffer (64-bit size, SYSTIM_U format) |

Return Parameter

| | | |
|---|---|---|
| ER | er | error code |
| struct stat* | buf | file information (time_t format) |
| struct stat_ms* | mbuf | file information (SYSTIM format) |
| struct stat_us* | ubuf | file information (SYSTIM_U format) |
| struct stat64* | buf64 | file information (64-bit size, time_t format) |
| struct stat64_ms* | mbuf64 | file information (64-bit size, SYSTIM format) |
| struct stat64_us* | ubuf64 | file information (64-bit size, SYSTIM_U format) |

Error Code

| | |
|---|---|
| E_OK | Normal completion |
| EX_ACCES | Directory included in "path" does not have the search permission attribute |
| EX_IO | I/O error |
| EX_NAMETOOLONG | File name is too long<br>- Directory or file name part in "path" is too long (NAME_MAX at maximum)<br>- Whole path length is too long (PATH_MAX at maximum). |
| EX_NOENT | File included in "path" does not exist or "path" is empty |
| EX_NOTDIR | "path" contains something other than a directory in the prefix part |
| EX_OVERFLOW | Value of st_size, st_ino, or st_blocks cannot be represented by the type of the fields in the result. |

Description

These functions store the information on the file specified by the path name "path" or the file descriptor "fd" to the area specified by "ubuf", "mbuf", "buf", "ubuf64", "mbuf64", or "buf64"

corresponding to each function.

It does not require the read, write, and execution permissions for the file specified by "path".

The structures stat, stat_us, stat_ms, stat64, stat64_us, and stat64_ms are defined as follows:

```
struct  stat    {
        dev_t           st_dev;                 /* ID of the device where the file exist */
        ino_t           st_ino;                 /* file serial number */
        mode_t          st_mode;                /* file mode */
        nlink_t         st_nlink;               /* number of links */
        uid_t           st_uid;                 /* owner ID */
        gid_t           st_gid;                 /* group ID */
        dev_t           st_rdev;                /* device type */
        off_t           st_size;                /* file size (bytes) */
        time_t          st_atime;               /* last access time */
        time_t          st_mtime;               /* last modified time */
        time_t          st_ctime;               /* last status change time */
        blksize_t       st_blksize;             /* I/O block size (bytes) */
        blkcnt_t        st_blocks;              /* number of allocated blocks */
        /* implementation-dependent additional information */
};
```

For stat_ms, the declarations of st_atime, st_mtime, and st_ctime in "stat" are replaced with the followings respectively:

```
        SYSTIM          st_atime;               /* last access time (milliseconds) */
        SYSTIM          st_mtime;               /* last modified time (milliseconds) */
        SYSTIM          st_ctime;               /* last status change time (milliseconds) */
```

For stat_us, the declarations of st_atime, st_mtime, and st_ctime in "stat" are replaced with the followings respectively:

```
        SYSTIM_U        st_atime_u;             /* last access time (microseconds) */
        SYSTIM_U        st_mtime_u;             /* last modified time (microseconds) */
        SYSTIM_U        st_ctime_u;             /* last status change time (microseconds) */
```

For stat64, st_size in "stat" is replaced with the following:

```
        off64_t         st_size;                /* 64-bit file size (bytes) */
```

For stat64_ms, the declarations of st_size, st_atime, st_mtime, and st_ctime in "stat" are replaced with the followings respectively:

```
        off64_t         st_size;                /* 64-bit file size (bytes) */
        SYSTIM          st_atime;               /* last access time (milliseconds) */
        SYSTIM          st_mtime;               /* last modified time (milliseconds) */
        SYSTIM          st_ctime;               /* last status change time (milliseconds) */
```

For stat64_us, the declarations of st_size, st_atime, st_mtime, and st_ctime in "stat" are replaced with the followings respectively:

```
        off64_t         st_size;                /* 64-bit file size (bytes) */
        SYSTIM_U        st_atime_u;             /* last access time (microseconds) */
        SYSTIM_U        st_mtime_u;             /* last modified time (microseconds) */
        SYSTIM_U        st_ctime_u;             /* last status change time (microseconds) */
```

dev_t st_dev — device ID

As device IDs are dynamically assigned when devices are registered, they are not fixed values.

ino_t st_ino; — file serial number

ID to identify the file in the system.
Its value is dependent on the FIMP.

mode_t st_mode; — file mode

File type and access mode described in the file attribute.

nlink_t st_nlink; — number of links

For a file system with hard links, it returns the number of hard links.

For a file system without hard links, it returns 1.

uid_t st_uid; — owner ID

For a file system with an owner ID, it returns the ID.
For a file system without an owner ID, it returns 0.

gid_t st_gid; — group ID

```
                                For a file system with a group ID, it returns the ID.
                                For a file system without a group ID, it returns 0.

        dev_t           st_rdev;                device type
                                If the file is a character device or block special device and the FIMP
holds the ID of the device, it returns the ID.
                                Otherwise, it returns 0.

        off_t           st_size;                file size (bytes)
        off64_t         st_size;                64-bit file size (bytes)
                                For a normal file, it returns the file size in bytes.
                                For other file types, this field is undefined.

        time_t          st_atime;               last access time
        SYSTIM          st_atime;               last access time (milliseconds)
        SYSTIM_U        st_atime;               last access time (microseconds)
                                They return the last access time of the file in a format suitable for
each type.
                                The resolution of an actually returned value depends on the file
system.

        time_t          st_mtime;               last modified time
        SYSTIM          st_mtime;               last modified time (milliseconds)
        SYSTIM_U        st_mtime;               last modified time (microseconds)
                                They return the last modified time of the file in a format suitable
for each type.
                                The resolution of an actually returned value depends on the file
system.

        time_t          st_ctime;               last status change time
        SYSTIM          st_ctime;               last status change time (milliseconds)
        SYSTIM_U        st_ctime;               last status change time (microseconds)
                                They return the last status change time of the file in a format for
each type.
                                The resolution of an actually returned value depends on the file
system.

                st_atime, st_mtime, and st_ctime are called "file time stamps", and their strict
meaning and precision depend on the file system.
                For example, the time resolutions defined in the FAT file system specification are 2
seconds for modified time, and one day for access time. The last status change time does not exist. It
is considered to be the same as the last modified time.

        blksize_t       st_blksize;             I/O block size (bytes)
                                System-specific preferable I/O block size.
                                It may vary depending on files in some file system.

        blkcnt_t        st_blocks;              number of allocated blocks
                                Number of blocks required to allocate this file.
```

### 4.4.12 fs_rename - Changes the name and position of a file

C Language Interface

```
#include <t2ex/fs.h>

ER er = fs_rename(const char* oldpath, const char* newpath);
```

Parameter

| | | |
|---|---|---|
| const char* | oldpath | file name or directory name before change |
| const char* | newpath | file name or directory name after change |

Return Parameter

| | | |
|---|---|---|
| ER | er | error code |

Error Code

| | |
|---|---|
| E_OK | Normal completion |
| EX_ACCES | Required write permission attribute does not exist |

| | |
|---|---|
| EX_BUSY | "oldpath" or "newpath" is in use |
| EX_EXIST | Directory specified by "newpath" is not empty |
| EX_INVAL | "oldpath" is included in "newpath", or the last element of either argument is "." or ".." |
| EX_IO | I/O error |
| EX_ISDIR | "newpath" is a directory but "oldpath" is a file |
| EX_NAMETOOLONG | File name is too long<br>- Directory or file name part in "oldpath" or "newpath" is too long (NAME_MAX at maximum)<br>- Whole path length is too long (PATH_MAX at maximum). |
| EX_NOENT | File does not exist<br>- "oldpath" does not exist<br>- Path part of "newpath" does not exist<br>- Either "oldpath" or "newpath" is an empty string |
| E_NOSPC | Directory containing "newpath" cannot be expanded |
| EX_NOTDIR | Not a directory<br>- Path contains something other than a directory in the prefix part<br>- "oldpath" is a directory but "newpath" is not a directory |
| EX_ROFS | Read-only file system |
| EX_XDEV | "oldpath" and "newpath" are on different file systems, and it is unsupported to move a file between different file systems |

Description

This function renames a file.
This function renames the file specified by "oldpath" to the name specified by "newpath".

If "oldpath" is a file, "newpath" must not be a directory.
If the file specified by "newpath" already exists, it is deleted first.

If "oldpath" is a directory, "newpath" must not be a file.
If the directory specified by "newpath" already exists, it is deleted first.

If the last element of either argument is "." or "..", fs_rename() fails.

"oldpath" must not be included in the path name specified by "newpath".

The write permission is required for the directories containing "oldpath" and "newpath".

If "oldpath" is a directory, the write permission of "oldpath" is required. If "newpath" exists, the write permission of "newpath" is also required.

At the normal completion, fs_rename() updates the last modified time and the last status change time of the parent directory for each argument.

If fs_rename() fails due to an error other than EX_IO, the file specified by "newpath" is not affected.

### 4.4.13 fs_unlink - Deletes a file

C Language Interface

```
#include <t2ex/fs.h>

ER er = fs_unlink(const char* path);
```

Parameter

| | | |
|---|---|---|
| const char* | path | path name of the file to delete |

Return Parameter

| | | |
|---|---|---|
| ER | er | error code |

Error Code

| | |
|---|---|
| E_OK | Normal completion |
| EX_ACCES | Write permission attribute does not exist for the parent directory |
| EX_BUSY | File is in use |

| | |
|---|---|
| EX_ISDIR | "path" is a directory |
| EX_NAMETOOLONG | File name is too long<br>- Directory or file name part in "path" is too long (NAME_MAX at maximum)<br>- Whole path length is too long (PATH_MAX at maximum). |
| EX_NOENT | File included in "path" does not exist or "path" is empty |
| EX_NOTDIR | "path" contains something other than a directory in the prefix part |
| EX_ROFS | Read-only file system |

Description

This function deletes the file with the name specified by "path".
This function cannot delete a directory.

At the normal completion, this function updates the last modified time and the last status change time of the parent directory.

See Also
fs_rmdir

### 4.4.14 fs_fsync, fs_fdatasync - Synchronizes a file

C Language Interface

```
#include <t2ex/fs.h>

ER er = fs_fsync(int fd);
ER er = fs_fdatasync(int fd);
```

Parameter

| | | |
|---|---|---|
| int | fd | file descriptor |

Return Parameter

| | | |
|---|---|---|
| ER | er | error code |

Error Code

| | |
|---|---|
| E_OK | Normal completion |
| EX_BADF | "fd" is invalid |
| EX_INTR | Aborted by fs_break() (for fs_fsync()) |
| EX_INVAL | "fd" is not a file descriptor for a device that can be synchronized |
| EX_IO | I/O error |

Other errors defined in fs_read() and fs_write() are returned.

Description

These functions complete all of the currently queued I/O requests of the file specified by "fd", and synchronize the disk cache corresponding to that file with the physical device. These functions will return after waiting for completion of synchronization.

While fs_fsync() synchronizes all data and metadata of the file, fs_fdatasync() does not synchronize metadata which is not directly related to data of the file, such as last modified time.

In some FIMPs, fs_fdatasync() may act exactly the same as the fs_fsync().

### 4.4.15 fs_chdir, fs_fchdir - Changes the current directory

C Language Interface

```
#include <t2ex/fs.h>

ER er = fs_chdir(const char* path);
ER er = fs_fchdir(int fd);
```

Parameter

| | | |
|---|---|---|
| const char* | path | directory pathname |
| int | fd | file descriptor for a directory |

Return Parameter

| | | |
|---|---|---|
| ER | er | error code |

Error Code

| | |
|---|---|
| E_OK | Normal completion |
| EX_ACCES | Search permission attribute does not exist for the directory "fd" |
| EX_NAMETOOLONG | File name is too long<br>- Directory or file name part in "path" is too long (NAME_MAX at maximum)<br>- Whole path length is too long (PATH_MAX at maximum). |
| EX_NOTDIR | "fd" is not a directory |
| EX_NOENT | Directory for "path" does not exist or "path" is empty |
| EX_INTR | Aborted by fs_break() |
| EX_IO | I/O error |

Description

fs_chdir() sets the current directory to the directory specified by the path name "path".

fs_fchdir() sets the current directory to the opened directory specified by the file descriptor "fd". A directory can be opened by fs_open() with setting O_RDONLY in oflags.

In T2EX, there is only one current directory in the whole system.

See Also

fs_getcwd

4.4.16 fs_getcwd - Retrieves the current directory

C Language Interface

```
#include <t2ex/fs.h>

ER er = fs_getcwd(char* buf, size_t size);
```

Parameter

| | | |
|---|---|---|
| char* | buf | storage space for the directory name |
| size_t | size | byte size of "buf" |

Return Parameter

| | | |
|---|---|---|
| ER | er | error code |

Error Code

| | |
|---|---|
| E_OK | Normal completion |
| EX_INVAL | "size" is 0 |
| EX_RANGE | "size" is positive, but the size is smaller than the number of bytes of the string + 1 |
| EX_ACCES | Search permission attribute does not exist for the current directory or read/search permission attribute does not exist for its upper directory |
| EX_NOMEM | Insufficient memory |

Description

This function returns the absolute path name of the current directory to the area specified by "buf". This path name does not contain the "." or ".." element.

If "buf" is NULL, the behavior is undefined.

See Also

fs_chdir, fs_fchdir

### 4.4.17 fs_creat - Creates a file

C Language Interface

```
#include <t2ex/fs.h>

int fd = fs_creat(const char* pathname, mode_t mode);
```

Parameter

| | | |
|---|---|---|
| const char* | pathname | path name of the file |
| mode_t | mode | file creation mode |

Return Parameter

| | | |
|---|---|---|
| int | fd | file descriptor |

Error Code

See fs_open()

Description

This function creates a file and opens it.
It is equivalent to fs_open(pathname, O_WRONLY|O_CREAT|O_TRUNC, mode).

For details, see fs_open().

See Also

fs_open

### 4.4.18 fs_chmod, fs_fchmod - Changes the mode of a file

C Language Interface

```
#include <t2ex/fs.h>

ER er = fs_chmod(const char *path, mode_t mode);
ER er = fs_fchmod(int fd, mode_t mode);
```

Parameter

| | | |
|---|---|---|
| const char* | path | path name of a file to change the mode |
| int | fd | file descriptor to change the mode |
| mode_t | mode | file mode |

Return Parameter

| | | |
|---|---|---|
| ER | er | error code |

Error Code

| | |
|---|---|
| E_OK | Normal completion |
| EX_ACCES | Directory included in "path" does not have the search permission attribute |
| EX_BADF | "fd" is invalid |
| EX_NAMETOOLONG | File name is too long<br>- Directory or file name part in "path" is too long (NAME_MAX at maximum)<br>- Whole path length is too long (PATH_MAX at maximum). |
| EX_NOENT | File included in "path" does not exist or "path" is empty |
| EX_NOTDIR | "path" contains something other than a directory in the prefix part |
| EX_ROFS | Read-only file system |

Description

fs_chmod() changes the mode about the access permission of the file specified by the path name "path" to another one specified by "mode".
It changes only the bits for the access permission in the file mode.

For the meaning and effect of each bit, see the term "File mode".

In case of the FAT file system, this function sets only the write permission/protection.

fs_fchmod() changes the access permission on the open file specified by the file descriptor "fd".

See Also
fs_open, fs_stat

### 4.4.19 fs_utimes, fs_utimes_ms, fs_utimes us - Changes the timestamp of a file

C Language Interface

```
#include <t2ex/fs.h>

ER er = fs_utimes(const char* path, const struct timeval tim[2]);
ER er = fs_utimes_ms(const char* path, const SYSTIM tim_m[2]);
ER er = fs_utimes_us(const char* path, const SYSTIM_U tim_u[2]);
```

Parameter

| | | |
|---|---|---|
| const char* | path | path name |
| const struct timeval | tim[2] | System time in the POSIX format and in microseconds |
| const SYSTIM | tim_m[2] | System time in milliseconds |
| const SYSTIM_U | tim_u[2] | System time in microseconds |

Return Parameter

| | | |
|---|---|---|
| ER | er | error code |

Error Code

| | |
|---|---|
| E_OK | Normal completion |
| EX_ACCES | Directory included in "path" does not have the search permission attribute |
| EX_INVAL | Specified time is not supported in the file system |
| EX_NAMETOOLONG | File name is too long<br>- Directory or file name part in "path" is too long (NAME_MAX at maximum)<br>- Whole path length is too long (PATH_MAX at maximum). |
| EX_ROFS | Read-only file system |

Description

These functions set the access time and the modified time of the file specified by "path" to the specified times.
tim_u[0], tim_m[0], and tim[0] specify a new access time.
tim_u[1], tim_m[1], and tim[1] specify a new modified time.

The structure "timeval" is defined as follows:

```
        struct timeval {
                long    tv_sec;         /* second */
                long    tv_usec;        /* microsecond */
        };
```

If tim, tim_m, or tim_u is NULL, the file access time and the modified time are set to the maximum time value not larger than the current time, supported by the FIMP.

At the time of completion, the last status change time of the file is updated.

See Also

fs_stat, fs_fstat, fs_stat64, fs_fstat64

4.4.20 fs_ioctl - Controls a device

C Language Interface

```
#include <t2ex/fs.h>

ER er = fs_ioctl(int fd, int request, ... /* arg */);
```

Parameter

| | | |
|---|---|---|
| int | fd | file descriptor |
| int | request | requested command |

Return Parameter

| | | |
|---|---|---|
| ER | er | result depending on the requested command<br>or error code |

Error Code

| | |
|---|---|
| E_OK | Normal completion |
| EX_BADF | "fd" is invalid |
| EX_INVAL | "request" or subsequent arguments are invalid |
| EX_NOTTY | "fd" does not refer to a special device of the character type |

In addition to the errors above, the FIMP returns an appropriate error depending on the type of "request".

Description

fs_ioctl() controls a file or device specified by "request".
"fd" is an open file descriptor which refers to a device.
"request" and subsequent optional arguments are passed to the FIMP corresponding to "fd" to be processed.

"arg" is additional information required to execute "request" on the target device.
"arg" is a pointer to the int type or the device-specific data structure depending on the type of "request".

The "arg" type for each "request" is described below.

FIONBIO const int*
Sets the blocking or non-blocking mode of I/O operation for the descriptor by the value pointed to by the argument int*.
*arg == 0 sets the blocking mode (the O_NONBLOCK status flag is cleared).
*arg != 0 sets the non-blocking mode (the O_NONBLOCK status flag is set).

FIONREAD int*
Returns the number of bytes ready to immediately read to the area pointed to by the argument int*.

FIONWRITE int*
Returns the number of bytes of the data stored in the send queue for the descriptor to the area pointed to by the argument int*.
Those bytes are data written to the descriptor, waiting to be processed.
How they are processed is device-dependent.

FIONSPACE int*
Returns the available space of the send queue for the descriptor to the area pointed to by the argument int*.
This value is the size of the send queue minus the size of data stored in the queue.

Other values not reserved by the system for "request" are passed to the FIMP together with the arguments to be processed.
This provides a communication path between applications and the FIMP.

Some FIMPs may return an error without EC_ERRNO as the main error code.
This can be used to directly return the error code from the device driver.

4.4.21 fs_fcntl - file control

C Language Interface

#include <t2ex/fs.h>

ER er = fs_fcntl(int fd, int cmd, ... /* arg */);

Parameter

| | | |
|---|---|---|
| int | fd | file descriptor |
| int | cmd | operation command |
| | arg | argument list may be required depending on the command (variable length) |

Return Parameter

| | | |
|---|---|---|
| ER | er | zero or positive result depending on the command<br>or error code |

Error Code

| | |
|---|---|
| EX_BADF | "fd" is invalid |
| EX_INVAL | Value of "cmd" is invalid |

Other than above, the FIMP returns an appropriate error code depending on the command.

Description

This function performs one of the following operations specified by "cmd" for the file descriptor "fd":

F_GETFL
Returns the file status flag and the access mode of the file descriptor "fd".
The file access mode is retrieved by masking the return code with O_ACCMODE.

F_SETFL
Sets the file status flag of the file descriptor "fd" with the corresponding bits in the third argument "arg".
The third argument "arg" is an int type.
The bits corresponding to the file access mode and the file creation flag are ignored and the file status flag is set.

Other bits of "arg" can be changed, but the resulting behavior is undefined.

Because the file status flag and the access mode are associated with each individual file descriptor, the setting of F_SETFL has no effect on a different file descriptor which is obtained by separately opening the same file.

Other commands and their "arg" are passed to the FIMP to be processed.

4.4.22 fs_truncate, fs_ftruncate, fs_truncate64, fs_ftruncate64 - Truncates or enlarges a file

C Language Interface

#include <t2ex/fs.h>

ER er = fs_truncate(const char* path, off_t length);
ER er = fs_ftruncate(int fd, off_t length);
ER er = fs_truncate64(const char* path, off64_t length64);
ER er = fs_ftruncate64(int fd, off64_t length64);

Parameter

| | | |
|---|---|---|
| const char* | path | path name of the file |
| int | fd | file descriptor |
| off_t | length | length of the file |
| off64_t | length64 | length of the file (by 64-bit offset) |

Return Parameter

| | | |
|---|---|---|
| ER | er | error code |

Error Code

| | |
|---|---|
| E_OK | Normal completion |
| EX_BADF | "fd" is not opened for writing |
| EX_INTR | Aborted by fs_break() |
| EX_INVAL | "length" or "length64" is negative, or "fd" is invalid |
| EX_BIG | "length" or "length64" exceeds the maximum length of a file |
| EX_IO | I/O error |
| EX_ISDIR | "path" or "fd" is a directory |
| EX_NAMETOOLONG | File name is too long<br>- Directory or file name part in "path" is too long (NAME_MAX at maximum)<br>- Whole "path" length is too long (PATH_MAX at maximum) |
| EX_NOENT | File included in "path" does not exist or "path" is empty |
| EX_ROFS | Read-only file system |

Description

These function change the length of the file named "path" or referred to by "fd" to the one specified by "length" or "length64".

"fd" must be a file descriptor opened for writing.

If the file is longer than "length" or "length64", the data exceeding "length" or "length64" is truncated.
If the file is shorter than "length" or "length64", the file size is enlarged up to "length" or "length64", filling a new space with zeros.

For an open file, the file offset is not changed.

If the size is changed, the last modified time and the last status change time of the file are updated.

fs_truncate64 and fs_ftruncate64 can be sized with a 64-bit offset.

See Also

fs_open

### 4.4.23 fs_sync - Synchronizes the file system

C Language Interface

```
#include <t2ex/fs.h>

ER er = fs_sync(void);
```

Parameter

None

Return Parameter

| | | |
|---|---|---|
| ER | er | error code |

Error Code

| | |
|---|---|
| E_OK | Normal completion |
| EX_INTR | Aborted by fs_break() |

Description

For all file systems connected to the system, this function completes all of the currently queued I/O requests, and synchronizes all disk caches with the physical devices, This function will return after waiting for completion of synchronization.

At the normal completion of fs_sync(), synchronization of all files systems connected to the system with the physical devices is guaranteed.

SEE_ALSO

fs_fsync, fs_fdatasync

4.4.24 fs_statvfs, fs_fstatvfs - Gets the file system statistics

C Language Interface

```
#include <t2ex/fs.h>

ER er = fs_statvfs(const char* path, struct statvfs* buf);
ER er = fs_fstatvfs(int fd, struct statvfs* buf);
```

Parameter

| | | |
|---|---|---|
| const char* | path | path name |
| struct statvfs* | buf | file statistics |
| int | fd | file descriptor |

Return Parameter

| | | |
|---|---|---|
| ER | er | error code |
| struct statvfs* | buf | file statistics |

Error Code

| | |
|---|---|
| E_OK | Normal completion |
| EX_BADF | "fd" is invalid (fs_fstatvfs) |
| EX_IO | I/O error |
| EX_INTR | Aborted by fs_break() |
| EX_OVERFLOW | Some values cannot be represented by the structure of "buf" |
| EX_NAMETOOLONG | File name is too long<br>- Directory or file name part in "path" is too long (NAME_MAX at maximum)<br>- Whole "path" length is too long (PATH_MAX at maximum) |
| EX_NOENT | File included in "path" does not exist or "path" is empty |
| EX_NOTDIR | "path" contains something other than a directory in the prefix part |

Description

fs_statvfs() returns information on the file system that contains the file specified by "path", to "buf".

fs_fstatvfs() returns information on the file system that contains the file referred to by the file descriptor "fd", to "buf".

The structure "statvfs" contains the following elements:

| | | |
|---|---|---|
| unsigned long | f_bsize | block size of the file system |
| unsigned long | f_frsize | block size of the fragment |
| fsblkcnt_t | f_blocks | total number of blocks in the file system in units of f_frsize |
| fsblkcnt_t | f_bfree | number of available blocks |
| fsblkcnt_t | f_bavail | equivalent to b_free |
| fsfilcnt_t | f_files | total number of files |
| fsfilcnt_t | f_ffree | number of blank files |
| fsfilcnt_t | f_favail | equivalent to f_ffree |
| unsigned long | f_fsid | file system ID |
| unsigned long | f_flag | bit mask for f_flag values |
| unsigned long | f_namemax | maximum length of the file name |

f_flag values are as follows:

| | |
|---|---|
| ST_RDONLY | Read-only file system |
| ST_NOSUID | Does not support the semantics of ST_ISUID and ST_ISGID file mode bits (Normally 1) |
| ST_REMOVABLE | Removable file system |
| ST_MEMORY | Memory file system |
| ST_NETWORK | Network file system |

### 4.4.25 fs_mkdir - Creates a directory

C Language Interface

```
#include <t2ex/fs.h>

ER er = fs_mkdir(const char* path, mode_t mode);
```

Parameter

| | | |
|---|---|---|
| const char* | path | path name of the directory to create |
| mode_t | mode | permission attribute of the directory |

Return Parameter

| | | |
|---|---|---|
| ER | er | error code |

Error Code

| | |
|---|---|
| E_OK | Normal completion |
| EX_ACCES | Write permission attribute does not exist for the parent directory |
| EX_EXIST | File with the name already exists |
| EX_NAMELOOLONG | Directory or file name part in "path" is too long (NAME_MAX at maximum) |
| EX_NOENT | File included in "path" does not exist or "path" is empty |
| EX_NOSPC | Insufficient device space |
| EX_NOTDIR | "path" contains something other than a directory in the prefix part |
| EX_ROFS | Parent directory exists in a read-only file system |

Description

This function creates a new directory with the path name specified by "path".

The access permission of the new directory is set by "mode".

Since the concept of owners, groups, and others does not exist in T2EX, the access privilege is granted if any of the bits is set.

Whether or not an access privilege is effective for read, write, or execution depends on the file system.

See Also

fs_rmdir

### 4.4.26 fs_rmdir - Removes a directory

C Language Interface

```
#include <t2ex/fs.h>

ER er = fs_rmdir(const char* path);
```

Parameter

| | | |
|---|---|---|
| const char* | path | directory pathname |

Return Parameter

| | | |
|---|---|---|
| ER | er | error code |

Error Code

| | |
|---|---|
| E_OK | Normal completion |
| EX_BUSY | Directory is in use(root directory or a connection point) |
| EX_NOTEMPTY | "path" is not an empty directory |
| EX_INVAL | "path" is invalid. |

| | |
|---|---|
| EX_IO | I/O error |
| EX_NAMELOOLONG | Directory or file name part in "path" is too long (NAME_MAX at maximum) |
| EX_NOENT | "path" does not exist |
| EX_NOTDIR | "path" contains something other than a directory in the prefix part |
| EX_ROFS | Read-only file system |

Description

This function deletes the directory with the path name specified by "path".
The directory must be empty.

At the normal completion, the last modified time and the last status change time of the parent directory of the deleted directory have been updated.

See Also

fs_mkdir, fs_rename

### 4.4.27 fs_getdents - Reads directory entries

C Language Interface

```
#include <t2ex/fs.h>

int nb = fs_getdents(int fd, struct dirent* buf, size_t bufsz);
```

Parameter

| | | |
|---|---|---|
| int | fd | file descriptor for a directory |
| struct dirent* | buf | pointer to the memory area into which the directory entry structure is read |
| size_t bufsz | buf | size of the memory area specified by "buf" (bytes) |

Return Parameter

| | | |
|---|---|---|
| int | nb | size that have been read (bytes) or error code |
| struct dirent* | buf | directory entries that have been read |

Error Code

| | |
|---|---|
| EX_BADF | "fd" is invalid |
| EX_FAULT | Illegal address in argument |
| EX_INVAL | "bufsz" is too small, no directory entry is read. |
| EX_NOENT | Such a directory does not exist |
| EX_NOTDIR | File descriptor "fd" does not refer to a directory |

Description

fs_getdents() reads directory entries from the directory specified by "fd".

"fd" is a file descriptor of a directory opened by fs_open().

"buf" is the pointer to a memory area reserved by the application to read directory entries.

"bufsz" is the size of the memory area specified by "buf" (bytes).

fs_getdents() reads one or more directory entries into the memory area specified by "buf" at a maximum of "bufsz" and returns the size (bytes) it has read.

If it reaches the end of the directories, it returns 0.

The structure of directory entry (dirent) is defined as follows:

```
struct dirent {
        ino_t           d_ino;                  /* Internal number of file */
        unsigned short  d_reclen;               /* Byte size of this entry */
        char            d_name[NAME_MAX+1];     /* Name of entry */
};
```

Directory entry is variable length, the size of which is indicated by "d_reclen".

The start address of the next entry is obtained by adding "d_reclen" to the start address of this entry.
The sum of "d_reclen" of directory entries read is the return value.

* Usually, this function should not be used to read directory entries.
  Use the readdir_r library function in Standard C Compatible Library instead.

### 4.4.28 fs_break - Breaks a file management operation

C Language Interface

```
#include <t2ex/fs.h>

int ntsk = fs_break(ID tskid);
```

Parameter

| | | |
|---|---|---|
| ID | tskid | target task ID |

Return Parameter

| | | |
|---|---|---|
| int | ntsk | number of tasks which were released from wait<br>or error code |

Error Code

| | |
|---|---|
| E_ID | Task ID is invalid (negative or exceeding TMaxTskId) |
| E_NOEXS | Task with the task ID does not exist |

Description

This function releases the waiting state of the task specified by "tskid" which involves a call of the file manager.

If the target task is waiting during execution of an API call of the file manager, it is immediately released, and the API call in progress returns EX_INTR.

If "tskid" is TSK_ALL(= (-1)), this operation applies to all the tasks.

At the normal completion, fs_break() returns the number of waiting tasks which were released.

If none of the target tasks is waiting during execution of an API call of the file manager, fs_break() returns 0.

A task wait release request by fs_break() is not queued.
That is, if a task specified by "tskid" is not waiting, the executed fs_break() has no effect on it even if it starts waiting subsequently.

## 4.5 File System Implementation Part

### 4.5.1 Overview

T2EX lets users to define program codes that manage file systems on a specified device on their own. Such program codes are called the "File System Implementation Part (FIMP)".
The FIMP defines the file system structure of files which are the target of the file manager APIs, and processing of these files.

As the pre-registered FIMPs, T2EX has the "Basic FAT FIMP" which handles the FAT file system structure, and the "Basic Console FIMP" which handles the standard input/output.

An FIMP is registered and used to operate the file system and files on a device in the following steps:

1. Create functions that execute various operations on the file system structure and the device you want to use as FIMP.
These functions are called from the file manager of T2EX.

2. Use fs_regist() to register the pointer set of the FIMP functions as well as the FIMP name in the system.
   At this time, the FIMP registration function is called.

        * For the Basic FAT FIMP and the Basic Console FIMP, fs_main() automatically carries out these steps up to this point.
Therefore, when only the Basic FAT FIMP and the Basic Console FIMP are used, the above steps 1 and 2 are not necessary.

3. Use fs_attach() to associate the device where the file system is located with the FIMP that handles the file system. This connects the file system to the specified connection point.
   At this time, the FIMP attach device function is called.

4. Execute various API calls in the file manager to manipulate the file system or files.
   At this time, the internal FIMP request service functions are invoked corresponding to these API calls.

A device is disconnected and the FIMP is unregistered in the following steps:

1. Close all the open files on the device.
    To write any still-to-be-written data to the device, execute fs_sync().

2. Use fs_detach() to disconnect the device connected to a specified connection point.
   At this time, the FIMP detach device function is called from the file manager.

3. Repeat the step 2 for all the connected devices using this FIMP.

4. Use fs_unregist() to unregister the FIMP with the specified name from the system.
   At this time, the FIMP unregistration function is called.

### 4.5.2 FIMP Structure

An FIMP consists of the following functions.

- Request service function (reqfn)
- Registration function (registfn)
- Unregistration function (unregistfn)
- Attach device function (attachfn)
- Detach device function (detachfn)
- Startup function (startupfn)
- Cleanup function (cleanupfn)
- Break function (breakfn)

These functions are called from the file manager. They return E_OK for a successful operation indicating the normal completion, and an appropriate T2EX extended error code (EX_xxx) for a failed operation, according to the cause or reason of the failure.

The file manager passes path name or other strings in the UTF-8 format character code to these functions. If a different character code system is used in the file system on the device, the character codes need to be converted in these functions.

Because the file manager is implemented as a subsystem, these functions are executed as a quasi-task portion in the context of the application task that executed the API calls of the file manager.

However, fs_break() is executable from a task-independent portion. In this particular case, the break function called in fs_break() is executed as the task-independent portion.

Because the FIMP functions are executed as a quasi-task portion in the context of the application task, they should not use the following T-Kernel 2.0 APIs which may be used in an application task to avoid unexpected behavior.

For example, when an application task calls tk_wup_tsk() to wake up another task which calls file manager APIs, tk_slp_tsk() in the file manager APIs would be incorrectly woken up.

```
tk_slp_tsk()
tk_slp_tsk_u()
tk_wup_tsk()
tk_can_wup()
```

Startup function, cleanup function and break function are optional, but all the other functions are mandatory.

- Request service function

```
ER      (*reqfn)(fimp_t *req);
```

This function is called in any API call of the file manager, except fs_main(), fs_regist(), fs_unregist(), fs_attach(), fs_detach(), and fs_break(), to execute a file operation specified by "req".
Details about the fimp_t data type and the request service function are described later.

If API calls of the file manager are executed by multiple tasks at the same time, the request service function may also be called concurrently from the quasi-task portion in the context of each task. This means that the mutual exclusion is required in the request service function.

- Registration function

```
ER      (*registfn)(fimpinf_t *fimpinf, void *info);
```

This function is called in fs_regist() to initialize the device-independent part of the FIMP itself.

The details of fimpinf_t data type is described later. fimpinf->fimpnm is the FIMP name specified by fs_regist().
fimpinf->fimpsd data can be used freely by the FIMP. An appropriate value set by this function can be used in a different FIMP function later.
Typically, it is set to the pointer of a space allocated to store specific data required by the FIMP.

A new instance of "fimpinf" is created every time fs_regist() is executed. When fs_regist() is called for the same FIMP more than once, the value of "fimpinf" is different.

"info" is the value as specified to fs_regist().

- Unregistration function

```
ER      (*unregistfn)(fimpinf_t * fimpinf);
```

This function is called in fs_unregist() to terminate the device-independent part of the FIMP itself.

"fimpinf" has the same value as "fimpinf" passed to the FIMP registration function. If memory space for storing specific data is allocated and its address is set to fimpinf->fimpsd, it needs to be released.

- Attach device function

```
ER      (*attachfn)(coninf_t * coninf, void *info);
```

This function is called in fs_attach() to initialize and connect the device.

The details of coninf_t data type are described later. coninf->fimpinf has the same value as "fimpinf" passed to the FIMP registration function.

coninf->devnm is the device name "devnm" specified by fs_attach().
The file manager makes no check regarding the device. It is necessary to perform appropriate checks on the device specified in this function for its existence, file system structure on it, and so on.

coninf->dflags is passed as the value of "flags" specified in fs_attach(). If there is any additional information regarding the device type, it needs to be ORed with the original value of dflags in this function to be notified to the file manager.

coninf->connm is the connection point, i.e., "connm" specified in fs_attach().

coninf->consd data can be used freely by the FIMP. An appropriate value set by this function can be used in a different FIMP function later.
Typically, it is set to the pointer to a memory space allocated to store connection-specific data required by the FIMP.
A new instance of "coninf" is created every time fs_attach() is executed. When fs_attach() is called for the same device and FIMP more than once, the value of "coninf" is differnt.

"info" is the value as specified in fs_attach().

- Detach device function

```
ER      (*detachfn)(coninf_t *coninf);
```

This function is called in fs_detach() to disconnect a connected device.

It writes any data left in the cache that should be written to the device.

"coninf" has the same value as "coninf" passed to the device initialization function. If the storage space for a specific data is allocated and its pointer is set in coninf->consd, it needs to be released.

- Startup function

```
ER      (*startupfn)(coninf_t *coninf, ID resid);
```

This function is called in the startup function of the file manager
invoked by the API call tk_sta_ssy() of T-Kernel 2.0, to initialize the resource management block specified by "resid".
The T2EX system does not use a resource management block, and this function is usually omitted.

- Cleanup function

```
ER      (*cleanupfn)(coninf_t *coninf, ID resid);
```

This function is called in the cleanup function of the file manager
invoked by the API call tk_cln_ssy() of T-Kernel 2.0, to release the resource management block specified by "resid".
The T2EX system does not use a resource management block, and this function is usually omitted.

- Break function

```
ER      (*breakfn)(coninf_t *coninf, ID tskid, BOOL set);
```

This function is called during the execution of fs_break().

At first, this function is called with "set" as TRUE. In this case, this function aborts and immediately terminates the execution of any unfinished FIMP request service function for the task specified by "tskid".
Upon return of the aborted request service function, this function is called again with "set" as FALSE. In this case, this function initializes the internal aborted state of the task specified by "tskid".

If a request service function is waiting during the execution, this function immediately releases the wait, and the aborted request service function returns the EX_INTR error code.
However, for a file reading/writing request service function which already has read or written some data, such state is regarded as normal operation and the request service function returns E_OK with the actually read/written number of bytes.

This function always returns E_OK.

"coninf" has the same value as "coninf" passed to the attach device function.

This function is called both from a quasi-task portion and from a task-independent portion, and thus should operate correctly in both contexts.

This function may be omitted if all the FIMP request service functions return in a short time and will not enter into long or indefinite wait.

4.5.3 Data Types

- FIMP definition (fs_fimp_t)

This structure defines an FIMP to be used as an argument of fs_regist().

```
/* FIMP definition */
typedef struct {
        ER      (*reqfn)(fimp_t * req);                                 /* Request service function */
        ER      (*registfn)(fimpinf_t * fimpinf, void *info);           /* Registration function */
        ER      (*unregistfn)(fimpinf_t * fimpinf);                     /* Unregistration function */
        ER      (*attachfn)(coninf_t * coninf, void *info);             /* Attach device function */
        ER      (*detachfn)(coninf_t * coninf);                         /* Detach device function */
        ER      (*startupfn)(coninf_t * coninf, ID resid);              /* Startup function */
        ER      (*cleanupfn)(coninf_t * coninf, ID resid);              /* Cleanup function */
        ER      (*breakfn)(coninf_t * coninf, ID tskid, BOOL set);      /* Break function */
        int     flags;                                                  /* Kind of FIMP */
        int     priority;                                               /* Priority */
} fs_fimp_t;
```

- Request service function (reqfn)
- Registration function (registfn)
- Unregistration function (unregistfn)
- Attach device function (attachfn)
- Detach device function (detachfn)
- Startup function (startupfn)
- Cleanup function (cleanupfn)
- Break function (breakfn)

Specify these function pointers in an FIMP.
When the startup, cleanup, or break function is to be omitted, specify NULL.

- Kind of FIMP (flags)

This is set to the FIMP type which is an OR of some of the following bit flags.

| | |
|---|---|
| FIMP_FLAG_READONLY | Read only file system |
| FIMP_FLAG_MEMORY | Memory file system |
| FIMP_FLAG_NETWORK | Network file system |
| FIMP_FLAG_64BIT | Supports file size of 64 bits |
| FIMP_FLAG_USEABORT | Supports fs_break() |
| 0 | Others |

If FIMP_FLAG_USEABORT is specified, the break function cannot be omitted.

- Priority (priority)

This is set to the FIMP priority in eight levels (highest: 1 to lowest: 8).
The file managemer calls FIMP startup functions in the order of this priority.
This value is not referenced if the startup function is set to NULL.

- FIMP information (fimpinf_t)

This structure is used by an FIMP to manage information necessary for operation and passed to every FIMP function.

```
/* FIMP information */
typedef struct {
        char    fimpnm[L_FIMPNM+1];     /* Name of FIMP */
        void *  fimpsd;                 /* Specific data for FIMP */
} fimpinf_t;
```

- Name of FIMP (fimpnm)

FIMP name string specified by fs_regist().
An FIMP must not change this value.

- Specific data for FIMP (fimpsd)

This data can be used freely by an FIMP. The file manager does not use this value.

- Connection information (coninf_t)

This structure is used by an FIMP to manage necessary information per connection and passed to every function except the FIMP initialization and termination functions.

```
/* Connection information */
typedef struct {
        fimpinf_t *     fimpinf;                /* FIMP information */
        UB              devnm[L_DEVNM+1];       /* Name of device */
        int             dflags;                 /* Kind of device */
        char            connm[L_CONNM+1];       /* Name of connection point */
        void *          consd;                  /* Specific data for connection */
} coninf_t;
```

- FIMP information(fimpinf)

Pointer to FIMP information created by fs_regist().

- Name of device(devnm)

Device name specified by fs_attach(). An FIMP must not change this value.

- Kind of device(dflags)

Kind of a device which is an OR of some of the following bit flags.

| | |
|---|---|
| DEV_FLAG_READONLY | Read only device |
| DEV_FLAG_REMOVABLE | Removable device |
| DEV_FLAG_MEMORY | Memory device |
| 0 | Others |

It is set to the value of "flags" specified by fs_attach(). Additional values may be set in the device initialization function.

- Name of connection point(connm)

Name of connection point specified by fs_attach().
An FIMP should not change this value.

- Specific data for connection(consd)

coninf->consd data can be used freely by an FIMP. The file manager does not use this value.

- Open file ID (fid_t)

Numerical data type for identifying a file/directory opened by an FIMP.

The value of the open file ID depends on a specific FIMP implementation. If a file/directory is opened more than once, their open file IDs shall be a same value.

4.5.4 Request Service Function

The request service function is called in any API call of the file manager, except fs_regist(), fs_unregist(), fs_attach(), fs_detach(), and fs_break(), to execute a specified file operation.

```
        ER      (*reqfn)(fimp_t *req);
```

fimp_t is the following union which indicates details of a request file operation.

```
/* Union of file request service */
union fimp {
        struct {                                /* Common structure */
                int             r_code;         /* Request service code */
                coninf_t *      coninf;         /* Pointer to connection information */
        } com;
                                                /* Parameters for each request service code */
        struct fimp_open        r_open;                 /* FIMP_OPEN */
        struct fimp_close       r_close;                /* FIMP_CLOSE */
        struct fimp_read64      r_read64;               /* FIMP_READ64 */
        struct fimp_write64     r_write64;              /* FIMP_WRITE64 */
```

```
        struct fimp_ioctl       r_ioctl;            /* FIMP_IOCTL */
        struct fimp_fsync       r_fsync;            /* FIMP_FSYNC */
        struct fimp_truncate64  r_truncate64;       /* FIMP_TRUNCATE64 */
        struct fimp_ftruncate64 r_ftruncate64;      /* FIMP_FTRUNCATE64 */
        struct fimp_unlink      r_unlink;           /* FIMP_UNLINK */
        struct fimp_rename      r_rename;           /* FIMP_RENAME */
        struct fimp_chmod       r_chmod;            /* FIMP_CHMOD */
        struct fimp_fchmod      r_fchmod;           /* FIMP_FCHMOD */
        struct fimp_mkdir       r_mkdir;            /* FIMP_MKDIR */
        struct fimp_rmdir       r_rmdir;            /* FIMP_RMDIR */
        struct fimp_chdir       r_chdir;            /* FIMP_CHDIR */
        struct fimp_fchdir      r_fchdir;           /* FIMP_FCHDIR */
        struct fimp_getdents    r_getdents;         /* FIMP_GETDENTS */
        struct fimp_fstatvfs    r_fstatvfs;         /* FIMP_FSTATVFS */
        struct fimp_statvfs     r_statvfs;          /* FIMP_STATVFS */
        struct fimp_sync        r_sync;             /* FIMP_SYNC */
        struct fimp_utimes_us   r_utimes_us;        /* FIMP_UTIMES_US */
        struct fimp_fcntl64     r_fcntl64;          /* FIMP_FCNTL64 */
        struct fimp_stat64_us   r_stat64_us;        /* FIMP_STAT64_US */
        struct fimp_fstat64_us  r_fstat64_us;       /* FIMP_FSTAT64_US */
};
typedef union fimp fimp_t;
```

The file processing request union may contain members other than the above, depending on the file managemer implementation. These members are intended to be used by the file managemer and shall not be referenced and changed by an FIMP.

The request service function code (r_code) corresponds to an API call of the file manager to distinguish it from other file operations. It takes a value from FIMP_OPEN to FIMP_STAT64_US.

The parameters and the operation of a request service function per request service function code are described in the following.

- Any file/directory path name passed to the request service function is an absolute path name which starts with "/" and does not include "." nor "..".
- In the description of a request service function, the content of a space pointed to by a pointer "p" is referred to by "*p".
- The following notations are used in the parameter comment. For a pointer, they refer to the target of the pointer.

| | |
|---|---|
| (in) | Input: A reference-only parameter which is not changed. |
| (out) | Output: A change(set)-only parameter which is not referenced. |
| (in/out) | Input/output: A parameter that is/can be referenced and changed (set). |

- FIMP_OPEN

```
struct fimp_open {
        int             r_code;         /* (in) = FIMP_OPEN */
        coninf_t *      coninf;         /* (in/out) Pointer to connection information */
        const char *    path;           /* (in) File path name */
        int             oflags;         /* (in) Open flag */
        mode_t          mode;           /* (in) File mode */
        fid_t *         fid;            /* (out) Pointer to open file ID */
};
```

This function opens a file or directory specified by "path".

If the file specified by "path" does not exist and O_CREAT is specified in "oflags", this function creates a new file with the specified path name.

"oflags" receives the value specified by "oflags" of fs_open().
"oflags" contains any one of the file access modes O_RDONLY, O_WRONLY, and O_RDWR, and may contain file creation flags O_CREAT, O_EXCL, and O_TRUNC and the file status flag O_APPEND.

"mode" receives the value specified by "mode" of fs_open().
"mode" is valid as the access permission flag for the newly created file only when the specified file does not exist and O_CREAT is specified in "oflags".

For a file system with no owner, group, or privilege, the interpretation of each bit of "mode" depends on an FIMP.
For instance, if there is no owner function and S_IWUSR is specified, whether to give the write privilege on the file or to generate an error depends on an FIMP.
Also, for a writable file, which attribute among S_IWUSR, S_IWGRP, and S_IWOTH is set for st_mode

returned by FIMP_STAT64_US and FIMP_FSTAT64_US depends on an FIMP as well.

If a file is successfully opened, the open file ID which identifies the opened file is stored in "*fid".
The open file ID is used by subsequent request service functions as the parameter to specify the opened file.

- FIMP_CLOSE

```
struct fimp_close {
        int             r_code;         /* (in) = FIMP_CLOSE */
        coninf_t*       coninf;         /* (in/out) Pointer to connection information */
        fid_t           fid;            /* (in) Open file ID */
        int             oflags;         /* (in) Open flag */
};
```

This function closes an open file or directory specified by "fid".

- FIMP_READ64

```
struct fimp_read64 {
        int             r_code;         /* (in) = FIMP_READ64 */
        coninf_t *      coninf;         /* (in/out) Pointer to connection information */
        fid_t           fid;            /* (in) Open file ID */
        int             oflags;         /* (in) Open flag */
        void *          buf;            /* (out) Pointer to data read buffer */
        size_t *        len;            /* (in/out) Pointer to number of bytes to read / read */
        off64_t *       off;            /* (in) Pointer to file offset before read */
        off64_t *       retoff;         /* (out) Pointer to file offset after read */
};
```

This function reads data for the number of bytes specified by "*len" into "*buf", from the file offset specified by "*off" in the open file specified by "fid".

At return, it stores the number of bytes actually read in "*len", and the file offset after reading in "*retoff".
This should work correctly even if the "off" and "retoff" pointers are the same.

If the end of file is reached, this function stores 0 in "*len" and returns E_OK.
If the end of the file is not reached and no bytes could not be read, it returns an appropriate error code.

If the file is a directory, its content is read.
If it is a connection point, the content in the root directory of the connected device is read likewise.

If "*off" is set to a value of 4 GB or more in an FIMP that can only
handle a file size up to 4 GB, it returns the EX_OVERFLOW error code.

"oflags" may have been changed from the value when FIMP_OPEN was called. If fs_fcntl() is executed with FIONBIO, the O_NONBLOCK bit of "oflags" will be changed. The same applies to other request service functions that have "oflags" as a parameter.

- FIMP_WRITE64

```
struct fimp_write64 {
        int             r_code;         /* (in) = FIMP_WRITE64 */
        coninf_t *      coninf;         /* (in/out) Pointer to connection information */
        fid_t           fid;            /* (in) Open file ID */
        int             oflags;         /* (in) Open flag */
        void *          buf;            /* (in) Pointer to data write buffer */
        size_t *        len;            /* (in/out) Pointer to number of bytes to write / written */
        off64_t *       off;            /* (in) Pointer to file offset before write */
        off64_t *       retoff;         /* (out) Pointer to file offset after write */
};
```

This function writes data for the number of bytes specified by "*len" from "*buf", from the file offset specified by "*off" in the open file specified by "id".

At return, it stores the number of bytes actually written in "*len", and the file offset after writing in "*retoff".

This should work correctly even if the "off" and "retoff" pointers are the same.

If no bytes could be written in the file, this function returns an appropriate error code.

If "*off" is set to a value of 4 GB or more in an FIMP that can only
handle a file size up to 4 GB, it returns the EX_OVERFLOW error code.

If O_APPEND is specified in "oflags", the "*off" value is ignored and additional data is always
written to the end of the file.

- FIMP_IOCTL

```
struct fimp_ioctl {
        int             r_code;         /* (in) = FIMP_IOCTL */
        coninf_t *      coninf;         /* (in/out) Pointer to connection information */
        fid_t           fid;            /* (in) Open file ID */
        int             oflags;         /* (in) Open flag */
        int             dcmd;           /* (in) Device control command number */
        void *          arg;            /* (in/out) Device control command parameter */
        ER *            retval;         /* (out) Pointer to return value */
};
```

This function executes the FIMP specific device operation against an open file specified by "fid".

"dcmd" is the second argument of fs_ioctl(), specifying an FIMP-specific device control command
number.
If the specified command number is not supported, this function returns the EX_NOSYS error.

"arg" is the third argument of fs_ioctl(). Its content is dependent on "dcmd".

"*retval" stores the value returned to fs_ioctl().

- FIMP_FSYNC

```
struct fimp_fsync {
        int             r_code;         /* (in) = FIMP_FSYNC */
        coninf_t *      coninf;         /* (in/out) Pointer to connection information */
        fid_t           fid;            /* (in) Open file ID */
        int             type;           /* (in) TYPE_FSYNC or TYPE_FDATASYNC */
        int             oflags;         /* (in) Open flag */
};
```

This function flushes the cache data not yet flushed to the device, regarding the open file specified
by "fid".

If "type" is TYPE_FSYNC, this function flushes all of the file data and the file management
information to the device.
If "type" is TYPE_FDATASYNC, it does not flush some management information which is not directly
related to data of the file, such as access time and modified time.

In some instances of the FIMP, the behavior of TYPE_FDATASYNC may be exactly the same as that of
TYPE_FSYNC.

If the target cache data has been flushed to the device, it simply returns E_OK.

- FIMP_TRUNCATE64

```
struct fimp_truncate64 {
        int             r_code;         /* (in) = FIMP_TRUNCATE64 */
        coninf_t *      coninf;         /* (in/out) Pointer to connection information */
        const char *    path;           /* (in) File path name */
        off64_t         len;            /* (in) File byte size */
};
```

This function truncates or enlarges the file specified by "path" to the size specified by "len".
When the file is enlarged, enlarged space is filled by zero.

If "len" is set to a value of 4 GB or more in an FIMP that can only
handle a file size up to 4 GB, it returns the EX_OVERFLOW error code.

- FIMP_FTRUNCATE64

```
struct fimp_ftruncate64 {
        int             r_code;         /* (in) = FIMP_FTRUNCATE64 */
        coninf_t *      coninf;         /* (in/out) Pointer to connection information */
        fid_t           fid;            /* (in) Open file ID */
        off64_t         len;            /* (in) File byte size */
};
```

This function truncates or enlarges the open file specified by "fid" to the length specified by "len". When the file is enlarged, enlarged space is filled by zero.

The specified file shall have been be opened for writing.

If "len" is set to a value of 4 GB or more in an FIMP that can only handle a file size up to 4 GB, it returns the EX_OVERFLOW error code.

- FIMP_UNLINK

```
struct fimp_unlink {
        int             r_code;         /* (in) = FIMP_UNLINK */
        coninf_t *      coninf;         /* (in/out) Pointer to connection information */
        const char *    path;           /* (in) File path name */
};
```

This function deletes the link of the file specified by "path".

In an FIMP that does not support links, it deletes the file itself.

- FIMP_RENAME

```
struct fimp_rename {
        int             r_code;         /* (in) = FIMP_RENAME */
        coninf_t *      coninf;         /* (in/out) Pointer to connection information */
        const char *    oldpath;        /* (in) Old file path name */
        const char *    newpath;        /* (in) New file path name */
};
```

This function changes the path name of the file specified by "oldpath" to the path name specified by "newpath".
The "oldpath" and the "newpath" shall be in a same file system, i.e., under the same connection point.

- FIMP_CHMOD

```
struct fimp_chmod {
        int             r_code;         /* (in) = FIMP_CHMOD */
        coninf_t *      coninf;         /* (in/out) Pointer to connection information */
        const char *    path;           /* (in) File path name */
        mode_t          mode;           /* (in) File mode */
};
```

This function sets the access permission mode for the file specified by "path" to the value specified by "mode".

For the explanation of "mode", see FIMP_OPEN.

- FIMP_FCHMOD

```
struct fimp_fchmod {
        int             r_code;         /* (in) = FIMP_FCHMOD */
        coninf_t *      coninf;         /* (in/out) Pointer to connection information */
        fid_t           fid;            /* (in) Open file ID */
        mode_t          mode;           /* (in) File mode */
};
```

This function sets the access permission mode for the open file specified by "fid" to the value specified by "mode".

For the explanation of "mode", see FIMP_OPEN.

- FIMP_MKDIR

```
struct fimp_mkdir {
        int             r_code;         /* (in) = FIMP_MKDIR */
        coninf_t *      coninf;         /* (in/out) Pointer to connection information */
        const char *    path;           /* (in) Directory path name */
        mode_t          mode;           /* (in) File mode */
};
```

This function creates a new directory specified by "path", and sets an access permission mode specified by "mode".

Refer to the FIMP_OPEN section for the details of "mode".

- FIMP_RMDIR

```
struct fimp_rmdir {
        int             r_code;         /* (in) FIMP_RMDIR */
        coninf_t *      coninf;         /* (in/out) Pointer to connection information */
        const char *    path;           /* (in) Directory path name */
};
```

This function removes a directory specified by "path".

- FIMP_CHDIR

```
struct fimp_chdir {
        int             r_code;         /* (in) FIMP_CHDIR */
        coninf_t *      coninf;         /* (in/out) Pointer to connection information */
        const char *    path;           /* (in) Directory path name */
};
```

This function changes the current directory to a directory specified by "path".

The current directory is managed by the file manager. The absolute path is always passed to the FIMP. This eliminates the necessity for the FIMP to manage the current directory. However, retaining the current directory in the FIMP may accelerate operations.

- FIMP_FCHDIR

```
struct fimp_chdir {
        int             r_code;         /* (in) = FIMP_CHDIR */
        coninf_t *      coninf;         /* (in/out) Pointer to connection information */
        fid_t           fid;            /* (in) Open file ID */
        char *          buf;            /* (out) Pointer to directory path name */
        int             len;            /* (in) Byte size of buf */
};
```

This function changes the current directory to the open directory specified by "fid", and stores its absolute path name in "*buf".
"len" specifies the number of bytes in the "buf" space.
If "len" is too small to store the absolute path name, this function returns an error code (EX_NAMETOOLONG).

The absolute path of the current directory stored in "*buf" is retained in the file manager to convert a relative path to the absolute path.

- FIMP_GETDENTS

```
struct fimp_getdents {
        int             r_code;         /* (in) = FIMP_GETDENTS */
        coninf_t *      coninf;         /* (in/out) Pointer to connection information */
        fid_t           fid;            /* (in) Open file ID */
        int             oflags;         /* (in) Open flag */
        struct dirent * buf;            /* (out) Pointer to directory entries read buffer */
        size_t *        len;            /* (in/out) Pointer to byte size of buf / actual read */
        off64_t *       off;            /* (in) Pointer to directory offset before read */
        off64_t *       retoff;         /* (out) Pointer to directory offset after read */
};
```

This function reads one or more directory entries in the open directory specified by "fid" from the file offset specified by "*off" into "*buf".

At return, it stores the number of bytes actually read in "*len", and the directory offset after reading in "*retoff".
This should work correctly even if the "off" and "retoff" pointers are the same.

If the end of directory is reached, this function stores 0 in "*len" and returns E_OK.

- FIMP_FSTATVFS

```
struct fimp_fstatvfs {
        int             r_code;         /* (in) = FIMP_FSTATVFS */
        coninf_t *      coninf;         /* (in/out) Pointer to connection information */
        fid_t           fid;            /* (in) Open file ID */
        struct statvfs * buf;           /* (out) Pointer to file system statistics */
};
```

This function reads the statistics information (struct statvfs) of file system that contains the file referred to by "fid" to "*buf".

- FIMP_STATVFS

```
struct fimp_statvfs {
        int             r_code;         /* (in) = FIMP_STATVFS */
        coninf_t *      coninf;         /* (in/out) Pointer to connection information */
        const char *    path;           /* (in) File path name */
        struct statvfs * buf;           /* (out) Pointer to file system statistics */
};
```

This function reads the statistics information (struct statvfs) of file system that contains the file specified by "path" to "*buf".

- FIMP_SYNC

```
struct fimp_sync {
        int             r_code;         /* (in) = FIMP_SYNC */
        coninf_t *      coninf;         /* (in/out) Pointer to connection information */
};
```

This function flushes any cache data not yet flushed to the device, for all the connected file systems supported by the FIMP.

If all the cache data has been flushed to the device, it simply returns E_OK.

- FIMP_UTIMES_US

```
struct fimp_utimes_us {
        int             r_code;         /* (in) = FIMP_UTIMES_US */
        coninf_t *      coninf;         /* (in/out) Pointer to connection information */
        const char *    path;           /* (in) File path name */
        SYSTIM_U *      times_u;        /* (in) times_u[0] Access time, times_u[1] Modified time */
};
```

This function changes the access and modified times of the file specified by "path" to the ones specified by *times_u.

- FIMP_FCNTL64

```
struct fimp_fcntl64 {
        int             r_code;         /* (in) = FIMP_FCNTL64 */
        coninf_t *      coninf;         /* (in/out) Pointer to connection information */
        fid_t           fid;            /* (in) Open file ID */
        int *           oflags;         /* (in) Pointer to open flag */
        int             fcmd;           /* (in) File control command number */
        off64_t *       off;            /* (in/out) Pointer to file offset */
        void *          arg;            /* (in/out) File control command parameter */
        ER *            retval;         /* (out) Pointer to return value */
};
```

This function executes the FIMP specific file operation against an open file specified by "fid".

"oflags" is the pointer to the Open flag specified in fs_open() and the flag value can be changed.

"fcmd" is the second argument of fs_fcntl(), specifying an FIMP-specific file control command number. If the specified command number is not supported, this function returns the EX_NOSYS error.

"arg" is the third argument of fs_fcntl(). Its content is dependent on "cmd".

"*retval" stores the error code value returned by fs_fcntl().

- FIMP_FSTAT64_US

```
struct fimp_stat64_us {
        int             r_code;         /* (in) = FIMP_STAT64_US */
        coninf_t *      coninf;         /* (in/out) Pointer to connection information */
        fid_t           fid;            /* (in) Open file ID */
        struct stat64_us * buf;         /* (out) Pointer to file information */
};
```

This function reads the file information (struct stat64_us) of an open file specified by "fid" to "*buf".

- FIMP_STAT64_US

```
struct fimp_stat64_us {
        int             r_code;         /* (in) = FIMP_STAT64_US */
        coninf_t *      coninf;         /* (in/out) Pointer to connection information */
        const char *    path;           /* (in) File path name */
        struct stat64_us * buf;         /* (out) Pointer to file information */
};
```

This function reads the file information (struct stat64_us) of a file specified by "path" to "*buf".

# Chapter 5 Network Communication Functions

## 5.1 Overview

The network communication functions provide socket interfaces for communication including TCP/IP and UDP/IP.
The API name prefix is "so_" (socket).
A program module which provides the network communication functions is referred to as "network communication manager".

The network communication manager has API calls similar to the networking service in the POSIX specification.
One major difference from POSIX specification is that return codes of the API calls represent error codes in the format defined by the T-Kernel specification if they are negative.

In the API calls that require timeout, APIs that can handle the timeout in the T-kernel 2.0 timeout format (TMO, TMO_U) have been added in addition to the API calls that handles timeout in the time data format of the POSIX specification.

## 5.2 Terms Used in This Section

### 5.2.1 Sockets

A socket is an endpoint for communication using the facilities described in this chapter. A socket is created with a specific socket type, described in 5.2.5 Socket Types, and is associated with a specific
protocol, detailed in 5.2.10 Protocols.

### 5.2.2 Socket Descriptors

0 or positive integer to identify a socket. It is allocated newly when a socket is created.
A socket descriptor is used to operate on a socket.

### 5.2.3 Stream

A stream is an abstraction used in reading and writing files and network communication, and realizes ordered sequential access.

### 5.2.4 Datagram

A unit of data transferred from one endpoint to another in connectionless mode service.

### 5.2.5 Socket Types

A socket type represents properties of a socket.
A socket is created with a specific type, which defines the communication semantics and which allows the selection of an appropriate communication protocol.

Three types are defined in T2EX: SOCK_STREAM, SOCK_DGRAM, and SOCK_RAW.

SOCK_STREAM

The SOCK_STREAM socket type provides reliable, sequenced, full-duplex octet streams between the socket and a peer to which the socket is connected. A socket of type SOCK_STREAM must be in a connected state before any data may be sent or received.

- Record boundaries are not maintained; data sent on a stream socket using output operations of one size may be received using input operations of smaller or larger sizes without loss of data.
- Data may be buffered; successful return from an output API call does not imply that the data has been delivered to the peer or even transmitted from the local system. If data cannot be successfully transmitted within a given time then the connection is considered broken, and subsequent operations shall fail.

SOCK_DGRAM

The SOCK_DGRAM socket type supports connectionless data transfer which is not necessarily acknowledged or reliable. Datagrams may be sent to the address specified (possibly multicast or broadcast) in each output operation, and incoming datagrams may be received from multiple sources. The source address of each datagram is available when receiving the datagram. An application may also pre-specify a peer address, in which case calls to output API calls that do not specify a peer address shall send to the pre-specified peer. If a peer has been specified, only datagrams from that peer shall be received. A datagram must be sent in a single output operation, and must be received in a single input operation.

- The maximum size of a datagram is protocol-specific.
- Output datagrams may be buffered within the system; thus, a successful return from an output API call does not guarantee that a datagram is actually sent or received.

SOCK_RAW
Sends and receives datagrams with packet headers.
- If an address family is AF_INET, an application sends/receives packets with IP headers. However, if IP_HDRINCL is set to 0 in so_setsockopt(), an IP header is added to the send packet automatically.
At this point, the IP header is set with the destination peer address and a protocol number being set in the socket.
If IP_HDRINCL is set to a non-zero value, the IP header is not added to the send packet automatically, so an application needs to set the IP header.
- This provides functions to manipulate the routing table when the address family is AF_ROUTE.
- The other address families do not support SOCK_RAW.

5.2.6 Address Families

All network protocols are associated with a specific address family. An address family provides basic services to the protocol implementation to allow it to function within a specific network environment. These services may include packet fragmentation and reassembly, routing, addressing, and basic transport. An address family is normally comprised of a number of protocols, one per socket type. Each protocol is characterized by an abstract socket type. It is not required that an address family support all socket types. An address family may contain multiple protocols supporting the same socket abstraction.

5.2.7 Network Address

A network-visible identifier used to designate specific endpoints in a network.
Specifically, for IP addressing, it is an identifier to designate a subnet.

Specific endpoints on host systems have addresses, and host systems may also have addresses.

5.2.8 Socket Address

An address associated with a socket or remote endpoint, including an address family identifier and addressing information specific to that address family. The address may include multiple parts, such as a network address associated with a host system and an identifier for a specific endpoint.

5.2.9 Addressing

An address family defines the format of a socket address. All network addresses are described using a general structure, called a sockaddr. However, each address family imposes finer and more specific structure, generally defining a structure with fields specific to the address family. The field sa_family in the sockaddr structure contains the address family identifier, specifying the format of the sa_data area. Example: for IPv4, it is a "sockaddr_in" structure (see chapter 8 netinet/in.h).

5.2.10 Protocols

Protocols are semantic and syntactic rules to exchange information.
Although it is sometimes called as "communication protocol" or "network protocol", this document refers it as "protocol" as long as it introduces no ambiguity.

A protocol supports one of the socket abstractions detailed in Socket Types. Selecting a protocol involves specifying the address family, socket type, and protocol number to the so_socket(). Certain semantics of the basic socket abstractions are protocol-specific. All protocols are expected to support the basic model for their particular socket type, but may, in addition, provide non-standard facilities or extensions to a mechanism.

5.2.11 Routing

Sockets provides packet routing facilities. A routing information database is maintained, which is used in selecting the appropriate network interface when transmitting packets.

5.2.12 Message

This document uses the word "message" to refer to information transmitted among sockets.

5.2.13 Network Interfaces

Each network interface in a system corresponds to a path through which messages can be sent and received. A network interface usually has a hardware device associated with it, though certain

interfaces such as the loopback interface, do not.

5.2.14 Socket I/O Mode

The I/O mode of a socket is described by the O_NONBLOCK status flag which pertains to the open socket description for the socket. This flag is initially off when a socket is created, but may be set and cleared by the use of the F_SETFL command of so_fcntl().

Basically, when a task calls a network communication API call, it waits until the processing is finished.
When the O_NONBLOCK flag is set, certain functions that would normally block until they are complete shall return immediately.

This operation mode is called "non-blocking mode".

Operation when O_NONBLOCK is set

> so_bind() initiates an address assignment and shall return without blocking when O_NONBLOCK is set; if the socket address cannot be assigned immediately, so_bind() shall return the EX_INPROGRESS error to indicate that the assignment was initiated successfully, but that it has not yet completed.
>
> so_connect() initiates a connection and shall return without blocking when O_NONBLOCK is set; it shall return the error EX_INPROGRESS to indicate that the connection was initiated successfully, but that it has not yet completed.
>
> Data transfer operations (so_read(), so_write(), so_send(), and so_recv() API calls) shall complete immediately, transfer only as much as is available, and then return without blocking, or return the EX_AGAIN error to indicate that no transfer could be made without blocking.

5.2.15 Socket Owner

T2EX sockets do not have the concept of owner.

5.2.16 Socket Queue Limits

The transmit and receive queue sizes for a socket are set when the socket is created. The default sizes used are protocol-specific. The sizes may be changed using so_setsockopt() with SO_SNDBUF or SO_RCVBUF.

Default buffer sizes for send/receive queue in each protocol are defined as system configuration information.

- SOCK_STREAM socket (AF_INET): TCP/IP
  - SoTcpTxBufSz: Send buffer size
  - SoTcpRxBufSz: Receive buffer size
- SOCK_DGRAM socket (AF_INET): UDP/IP
  - SoUdpTxBufSz: Send buffer size
  - SoUdpRxBufSz: Receive buffer size
- SOCK_RAW socket (AF_INET)
  - SoRawIPTxBufSz: Send buffer size
  - SoRawIPRxBufSz: Receive buffer size
- SOCK_RAW socket (AF_ROUTE)
  - SoRawTxBufSz: Send buffer size
  - SoRawRxBufSz: Receive buffer size

5.2.17 Pending Error

Errors may occur asynchronously, and be reported to the socket in response to input from the network protocol. The socket stores the pending error to be reported to a user of the socket at the next opportunity. The error is returned in response to a subsequent so_send(), so_recv(), or so_getsockopt() operation on the socket, and the pending error is then cleared.

5.2.18 Socket Receive Queue

A socket has a receive queue that buffers data when it is received by the system until it is removed by a receive call. Depending on the type of the socket and the communication provider, the receive queue may also contain ancillary data such as the addressing and other protocol data associated with the normal data in the queue, and may contain out-of-band or expedited data.

5.2.19 Socket Out-of-Band Data State

Out-of-band data is delivered through a transmission channel which is logically independent from the normal data and transmitted to users separately.
There are two ways to handle out-of-band data when receiving it; store it in a receive queue of normal data or store it in the other queue.

If out-of-band data may be placed in the socket receive queue;
Out-of-band data may be placed either at the end of the queue or before all normal data in the queue. In this case, out-of-band data is returned to an application program by a normal receive call.

If out-of-band data may be queued separately rather than being placed in the socket receive queue;
Out-of-band data shall be returned only in response to a receive call that requests out-of-band data. It is protocol-specific whether an out-of-band data mark is placed in the receive queue to demarcate data preceding the out-of-band data and following the out-of-band data. An out-of-band data mark is logically an empty data segment that cannot be merged with other segments in the queue. An out-of-band data mark is never returned in response to an input operation. so_sockatmark() can be used to test whether an out-of-band data mark is the first element in the queue. If an out-of-band data mark is the first element in the queue when an input API call is called without the MSG_PEEK option, the mark is removed from the queue and the following data (if any) is processed as if the mark had not been present.

5.2.20 Connection Indication Queue

Sockets that are used to accept incoming connections maintain a queue of outstanding connection indications. This queue is a list of connections that are awaiting acceptance by the application.

5.2.21 Asynchronous Errors

If any of the following conditions occur asynchronously for a socket, the corresponding value listed below shall become the pending error for the socket:

| | |
|---|---|
| EX_CONNABORTED | The connection was aborted locally. |
| EX_CONNREFUSED | For a connection-mode socket attempting a non-blocking connection, the attempt to connect was forcefully rejected. For a connectionless-mode socket, an attempt to deliver a datagram was forcefully rejected. |
| EX_CONNRESET | The peer has aborted the connection. |
| EX_HOSTDOWN | The destination host has been determined to be down or disconnected. |
| EX_HOSTUNREACH | The destination host is not reachable. |
| EX_MSGSIZE | For a connectionless-mode socket, the size of a previously sent datagram prevented delivery. |
| EX_NETDOWN | The local network connection is not operational. |
| EX_NETRESET | The connection was aborted by the network. |
| EX_NETUNREACH | The destination network is not reachable. |

5.2.22 Socket Options

There are a number of socket options which either specialize the behavior of a socket or provide useful information. These options may be set at different protocol levels and are always present at the uppermost "socket" level. Socket options are manipulated by two API calls, so_getsockopt() and so_setsockopt().

Protocol levels can be defined as follows.

| | |
|---|---|
| IPPROTO_IP | The IP level |
| IPPROTO_TCP | The TCP level |
| SOL_SOCKET | The socket level |

R- or RW in the description of the following options means the ability to get (so_getsockopt()) only or to both get and set (so_getsockopt() and so_setsockopt()) respectively.

For IP level options given below, the following can be specified.

IP_OPTIONS RW IP options set in the IP header of each packet to be sent.
Specify the buffer of size 0 to disable previously specified options.
For more information about IP options, see RFC-791.
By default, IP options embedded in the IP header are not specified.

IP_HDRINCL RW Enables/disables the addition of the IP header to the sent data by an application
Type of the option: int
Value of the option:
- When the value is non-zero, an application adds the IP header to the sent data before sending it.

- When the value is 0, an application does not adds the IP header to sent data; the system does instead.
This option is valid only for SOCK_RAW type sockets.
By default, the system adds the IP header.

For TCP levels options given below, the following can be specified.

| | | |
|---|---|---|
| TCP_NODELAY | RW | Enables/disables the immediate transmission of data.<br>Type of the option: int<br>Value of the option:<br>- When the value is non-zero, the Nagle's algorithm is not used.<br>- When the value is 0, the Nagle's algorithm is used.<br>To utilize the network more effectively, TCP uses the Nagle's algorithm to buffer packets before sending them in one go instead of frequently sending small packets. Small packets are sent immediately when the Nagle's algorithm is not used.<br>By default, the Nagle's algorithm is used. |
| TCP_MAXSEG | RW | Maximum length of a segment<br>Type of the option: int<br>By default, it is 536 bytes. |

For Socket level options given below, the following can be specified.

| | | |
|---|---|---|
| SO_DEBUG | RW | Debugging in the underlying protocol modules.<br>Type of the option: int<br>Value of the option:<br>- If the value is non-zero, debugging is on.<br>- If the value is 0, debugging is off.<br>The default value for SO_DEBUG is for debugging to be turned off. |
| SO_REUSEADDR | RW | Reuse of local addresses.<br>Type of the option: int<br>Value of the option:<br>- If the value is non-zero, reuse of the local address is permitted.<br>- If it is 0, reuse of the local address is not permitted.<br>The default value for SO_REUSEADDR is off; that is, reuse of local addresses is not permitted. |
| SO_KEEPALIVE | RW | Periodic transmission of keepalive messages.<br>Type of the option: int<br>Value of the option:<br>- If the value is non-zero, keep-alive messages are sent periodically.<br>- If the value is 0, keep-alive messages are not sent periodically.<br>The default value for SO_KEEPALIVE is zero, specifying that this capability is turned off. |
| SO_DONTROUTE | RW | Bypass of normal routing; route based on destination address only.<br>Type of the option: int<br>Value of the option:<br>- If the value is non-zero, this capability is enabled..<br>- If the value is 0, this capability is turned off.<br>The destination must be on a directly-connected network, and messages are directed to the appropriate network interface according to the destination address.<br>If invalid, messages are sent by using standard routing functions.<br>The default value for SO_DONTROUTE is zero, specifying that this capability is turned off. |
| SO_BROADCAST | RW | Permission to transmit broadcast datagrams.<br>Type of the option: int<br>Value of the option:<br>- If the value is non-zero, broadcast messages can be sent.<br>- If the value is 0, broadcast messages cannot be sent.<br>The default for SO_BROADCAST is that the ability to send broadcast datagrams on a socket is disabled. |
| SO_USELOOPBACK | RW | Enables/disables functions to communicate bypassing the hardware<br>Type of the option: int<br>Value of the option:<br>- If the value is non-zero, communication is performed bypassing the hardware when possible.<br>- If the value is 0, communication is never performed bypassing the |

```
hardware.
                                By default, communication is never performed bypassing the hardware.

        SO_LINGER       RW      Actions to be taken for queued, unsent data on so_close()
                                Type of the option: struct linger
                                Value of the option:
                                 - If the value of l_onoff is non-zero and l_linger is positive, the
system shall block the calling thread during so_close() until it can transmit the data or until the
end of the interval indicated by the l_linger member, whichever comes first.
                                 - If l_onoff is 0 or l_linger is 0, the system handles so_close() in
a way that allows the calling thread to continue as quickly as possible.
                                The default value for SO_LINGER is zero, or off, for the l_onoff
                                element of the option value and zero seconds for the linger time
                                specified by the l_linger element.

        SO_OOBINLINE    RW      Out-of-band data be placed into normal data input queue as received.
                                Type of the option: int
                                Value of the option:
                                 - If the value is non-zero, out-of-band data is then accessible using
API calls such as so_read() and so_recv without the MSG_OOB flag set.
                                 - If the value is 0, out-of-band data is not placed into the standard
receive queue.
                                The default for SO_OOBINLINE is off; that is, for out-of-band data
                                not to be placed in the normal data input queue.

        SO_REUSEPORT    RW      Enables/disables reuse of the local address and the port
                                Type of the option: int
                                Value of the option:
                                 - If the value is non-zero, reuse of the local address and the port
is permitted.
                                 - If the value is 0, reuse of the local address and the port is not
permitted.
                                By default, reuse of the local address and the port is not permitted.

        SO_TIMESTAMP    RW      Enables/disables the addition of timestamps to the received datagram
                                Type of the option: int
                                Value of the option:
                                 - If the value is other non-zero, timestamp is added to the received
datagram.
                                 - If the value is 0, timestamp is not added to the received datagram.
                                If this option is enabled, timestamp is stored in auxiliary data, and
cmsg_len, cmsg_level, and cmsg_type are the size of "timeval" structure (the number of byte),
SOL_SOCKET, and SCM_TIMESTAMP, respectively.
                                By default, timestamp is not added to the received datagram.

        SO_SNDBUF       RW      Size of send buffer (in bytes)
                                Type of the option: int
                                The default value for SO_SNDBUF option value is
                                protocol-dependent.
                                 - TCP/IP: 32768 bytes
                                 - UDP/IP: 9216 bytes
                                 - SOCK_RAW socket (AF_INET): 8192 bytes
                                 - SOCK_RAW socket (AF_ROUTE): 8192 bytes

        SO_RCVBUF       RW      Size of receive buffer (in bytes)
                                Type of the option: int
                                The default value for SO_RCVBUF option value is
                                protocol-dependent.
                                 - TCP/IP: 32768 bytes
                                 - UDP/IP: 41600 bytes
                                 - SOCK_RAW socket (AF_INET): 8192 bytes
                                 - SOCK_RAW socket (AF_ROUTE): 8192 bytes

        SO_SNDLOWAT     RW      Minimum amount of data to send for output operations (in bytes)
                                Type of the option: int
                                The default value for SO_SNDLOWAT is 2048 bytes.

        SO_RCVLOWAT     RW      Minimum amount of data to return to application for input operations
(in bytes).
                                Type of the option: int
                                The default value for SO_RCVLOWAT is 1 byte.

        SO_SNDTIMEO     RW      Timeout value for a socket send operation
```

```
                                Type of the option: struct timeval
                                Value of the option:
                                 - When the value of optval is non-zero second, the send API call
waits until the specified time passes.
When the timeout occurs, if no data has been written, error EX_AGAIN is returned. If more than one
byte of data has been written, the data size (the number of byte(s)) is returned.
                                 - If the value is 0 second, the timeout already specified in the send
API call is released.
                                The default for this option is the value zero, which indicates that
                                a send operation will not time out.

        SO_RCVTIMEO     RW      Timeout value for a socket receive operation
                                Type of the option: struct timeval
                                Value of the option:
                                 - When the value of optval is non-zero second, the received API call
waits until the specified time passes.
When the timeout occurs, if no data has been read, error EX_AGAIN is returned. If more than one byte
of data has been read, the data size (the number of byte(s)) is returned.
                                 - If the value of optval is 0 second, the timeout already specified
in the receive API call is released.
                                The default for this option is the value zero, which indicates that
                                a receive operation will not time out.

        SO_ERROR        R-      Pending error information on the socket
                                Type of the option: int
                                Value of the option:
                                 - The non-zero value indicates an asynchronous error.
                                 - If the value is zero, there are no pending errors.

        SO_TYPE         R-      Socket type
                                Type of the option: int
```

5.2.23 Use of Sockets over Internet Protocols

When a socket is created in the Internet family with a protocol value of zero, the implementation shall use the protocol listed below for the type of socket created.

```
        SOCK_STREAM             IPPROTO_TCP
        SOCK_DGRAM              IPPROTO_UDP
        SOCK_RAW                IPPROTO_RAW
```

The default protocols for type SOCK_STREAM and SOCK_DGRAM are TCP and UDP respectively.

A raw interface to IP is available by creating an Internet socket of type SOCK_RAW. The default protocol for type SOCK_RAW shall be identified in the IP header with the value IPPROTO_RAW. Applications should not use the default protocol when creating a socket with type SOCK_RAW, but should identify a specific protocol by value. The ICMP control protocol is accessible from a raw socket by specifying a value of IPPROTO_ICMP for protocol.

5.2.24 Host Name Table

A Table in which addresses, host names, and host aliases are defined.
It corresponds to the setting file /etc/hosts of the TCP/IP protocol stack implemented on PC.
Setting in the host name table is shared throughout the system.

An entry of the host name table consists of at least an address and a host name. In addition, multiple host aliases can be defined in an entry.

5.2.25 Name Resolution

It refers to the translation of a domain name to a network address.

T2EX provides a name resolution function based on the host name table and the name resolution servers.

5.2.26 Routing Table

A routing table is a routing information database which is used to find a route to a destination.
This table consists of routing information, which includes information of the network interface to be used for every destination and the gateway to which the host should send first to reach that destination.

The destination is a specific host or all hosts belonging to a network.
All hosts belonging to a network can be represented using a network addresses and a net mask.

A special address whose net mask consists of only 0 (zero) is used when the destination is neither a specific host nor a network address.
The gateway to be used in this case is called a "default gateway".

5.2.27 Routing Socket

A routing socket is a special socket for the routing table operation.

Manipulation of the routing sockets is described in 5.6.

5.3 Unsupported Functions

Although the T2EX networking function offers subsets of the networking services of the POSIX specification, some differences exist as follows.

(1) Because T2EX does not have a process or a concept of user, "socket owners" or "accessible user groups" has no meaning, and features related to those are not provided.

(2) The T2EX network function is independent from the file management function.
Therefore, unlike the POSIX specification, single T2EX API call does not perform both network communication and file management.
The network communication feature are provided by API calls with names like "so_XXXX()", and those of the file management feature are provided by API calls with names like "fs_XXXX()". They are provided as independent API calls.
(3) Following features are not provided.
However, these may be supported as part of T2EX in the future.

- IPv6
- IPsec
- Multicast
- Unix-domain socket
- Protocols above the transport layer except for DHCP and DNS

5.4 Data Type Definitions

5.4.1 Address Families

```
#define AF_UNSPEC       0               /* unspecified */
#define AF_INET         2               /* IPv4 */
#define AF_ROUTE        17              /* internal routing protocol */
#define AF_LINK         18              /* link layer interface */
```

5.4.2 Protocol Families

```
#define PF_UNSPEC       0               /* unspecified */
#define PF_INET         2               /* IPv4 */
#define PF_ROUTE        17              /* internal routing protocol */
#define PF_LINK         18              /* link layer interface */
```

5.4.3 Socket Type

```
#define SOCK_STREAM     1               /* stream socket */
#define SOCK_DGRAM      2               /* datagram socket */
#define SOCK_RAW        3               /* raw-protocol interface */
```

5.4.4 Protocol

```
#define IPPROTO_IP      0               /* IP */
#define IPPROTO_ICMP    1               /* ICMP */
#define IPPROTO_TCP     6               /* TCP */
#define IPPROTO_UDP     17              /* UDP */
#define IPPROTO_RAW     255             /* raw IP packet */
```

5.4.5 General Structure Describing Network Addresses

Struct "sockaddr" is a general structure representing a socket address.
It is used in the APIs handling socket addresses.

```
struct sockaddr {
        unsigned char   sa_len;         /* total length (in bytes) */
        unsigned char   sa_family;      /* address family */
        char            sa_data[14];    /* address value */
};
```

sa_family, which represents address family, defines the form of the sa_data field. sa_data field is an area specific to an address family.

If an application uses addresses belonging to a specific address family, casting a structure representing that address. (Example: For IPv4, casts the "sockaddr_in" structure (see 8.14 netinet/in.h)) to the "sockaddr" structure and the casting in the reverse direction need to be performed.

Additionally, note that the size of the sa_data area is 14 bytes when the address is stored in the "sockaddr" structure.
Although an address of address families provided in T2EX does not exceed this size, the size may be exceeded if any originally extended address is used.
In such cases, use the "sockaddr_storage" structure to store the address of arbitrary size.

5.4.6 Generic Structure to Store an Address of Any Family

Struct "sockaddr_storage" is a structure that has a large enough size to store any address of any address family.
It is used to handle an address that is too long to be stored in the "sockaddr" structure.

```
#define _SS_MAXSIZE     128
#define _SS_ALIGNSIZE   (sizeof(int64_t))
#define _SS_PAD1SIZE    (_SS_ALIGNSIZE - 2)
#define _SS_PAD2SIZE    (_SS_MAXSIZE - 2 - _SS_PAD1SIZE - _SS_ALIGNSIZE)

struct sockaddr_storage {
        uint8_t         ss_len;                         /* total length (in bytes) */
        sa_family_t     ss_family;                      /* address family */
        char            __ss_pad1[_SS_PAD1SIZE];        /* 6 bytes padding
                                                           (to align __ss_align in an 8 byte boundary)
*/
        int64_t         __ss_align;                     /* force desired structure storage alignment
*/
        char            __ss_pad2[_SS_PAD2SIZE];        /* 112 bytes padding
                                                           (to make the whole structure 128 bytes) */
};
```

5.4.7 Address Structure of the Data Link Layer

```
struct sockaddr_dl {
        uint8_t         sdl_len;                /* total length (in bytes) */
        sa_family_t     sdl_family;             /* address family (always AF_LINK) */
        uint16_t        sdl_index;              /* index for interface */
        uint8_t         sdl_type;               /* interface type */
        uint8_t         sdl_nlen;               /* interface name length (in bytes) */
        uint8_t         sdl_alen;               /* link level address length (in bytes) */
        uint8_t         sdl_slen;               /* link layer selector length (in bytes) */
        char            sdl_data[12];           /* minimum work area for both if name and ll address
*/
};
```

5.4.8 Structure to Store Statistics and Network Interface Information

```
struct if_data {
        /* interface information */
        unsigned char   ifi_type;               /* interface type */
        unsigned char   ifi_addrlen;            /* media address length (in bytes) */
        unsigned char   ifi_hdrlen;             /* media header length (in bytes) */
        int             ifi_link_state;         /* current link state */
        uint64_t        ifi_mtu;                /* maximum transmission unit */
        uint64_t        ifi_metric;             /* routing metric */
        uint64_t        ifi_baudrate;           /* linespeed */
        /* statistical data */
        uint64_t        ifi_ipackets;           /* packets received on interface */
        uint64_t        ifi_ierrors;            /* input errors on interface */
        uint64_t        ifi_opackets;           /* packets sent on interface */
        uint64_t        ifi_oerrors;            /* output errors on interface */
        uint64_t        ifi_collisions;         /* collisions on csma interfaces */
        uint64_t        ifi_ibytes;             /* total number of octets received (in octets) */
        uint64_t        ifi_obytes;             /* total number of octets sent (in octets) */
        uint64_t        ifi_imcasts;            /* packets received via multicast */
        uint64_t        ifi_omcasts;            /* packets sent via multicast */
```

```
        uint64_t        ifi_iqdrops;            /* dropped on input, this interface */
        uint64_t        ifi_noproto;            /* destined for unsupported protocol */
        struct timeval  ifi_lastchange;         /* last operational state change */
};
```

5.4.9 Scatter/Gather Structure

```
struct iovec {
        void*           iov_base;       /* base address */
        size_t          iov_len;        /* length (in bytes) */
};
```

5.4.10 Message Header for so_recvmsg() and so_sendmsg()

```
struct msghdr {
        void*           msg_name;               /* optional address */
        socklen_t       msg_namelen;            /* size of address (in bytes) */
        struct iovec*   msg_iov;                /* scatter/gather array */
        int             msg_iovlen;             /* the number of elements in msg_iov */
        void*           msg_control;            /* ancillary data */
        socklen_t       msg_controllen;         /* ancillary data buffer length (in bytes) */
        int             msg_flags;              /* flags on received message */
};
```

5.4.11 Structure for Auxiliary Data of so_recvmsg() and so_sendmsg()

```
struct cmsghdr {
        socklen_t       cmsg_len;               /* data byte count, including hdr (in bytes) */
        int             cmsg_level;             /* originating protocol */
        int             cmsg_type;              /* protocol-specific type */
};
```

Auxiliary data is stored in the area following the "cmsghdr" structure.

5.4.12 Structure for SO_LINGER Option

```
struct linger {
        int     l_onoff;        /* option on/off */
        int     l_linger;       /* linger time (in seconds) */
};
```

5.4.13 Set of Socket Descriptors

```
#define FD_SETSIZE      256

typedef struct fd_set {
        int     fds_bits[((FD_SETSIZE + (sizeof(int)*8-1)) / sizeof(int)*8)];
} fd_set;
```

5.4.14 Structure for Address Information of so_getaddrinfo()

```
struct addrinfo {
        int                     ai_flags;               /* input flag */
        int                     ai_family;              /* address family */
        int                     ai_socktype;            /* Socket type */
        int                     ai_protocol;            /* Protocol */
        socklen_t               ai_addrlen;             /* size of the socket address (in bytes) */
        struct sockaddr*        ai_addr;                /* socket address */
        char*                   ai_canonname;           /* standard name for the service location */
        struct addrinfo*        ai_next;                /* pointer to the next list */
};
```

5.4.15 Network Interface Operation Structure

```
struct ifreq {
#define IFNAMSIZ        16
        char    ifr_name[IFNAMSIZ];     /* Device name */
        union {
                struct sockaddr         ifru_addr;              /* host address */
                struct sockaddr         ifru_dstaddr;           /* destination address */
                struct sockaddr         ifru_broadaddr;         /* broadcast address */
                short                   ifru_flags;             /* flag */
                short                   ifru_metric;            /* metric */
```

```
                void*                   ifru_data;              /* area used by the interface */
        } ifr_ifru;
#define ifr_addr                ifr_ifru.ifru_addr
#define ifr_dstaddr             ifr_ifru.ifru_dstaddr
#define ifr_broadaddr           ifr_ifru.ifru_broadaddr
#define ifr_flags               ifr_ifru.ifru_flags
#define ifr_metric              ifr_ifru.ifru_metric
#define ifr_data                ifr_ifru.ifru_data
};
```

5.4.16 Structure for SIOCAIFADDR and SIOCDIFADDR of so_ioctl()

```
struct ifaliasreq {
        char                    ifrac_name[IFNAMSIZ];   /* Device name */
        struct sockaddr         ifra_addr;              /* host address */
        struct sockaddr         ifra_dstaddr;           /* destination address */
#define ifra_broadaddr          ifra_dstaddr            /* broadcast address */
        struct sockaddr         ifra_mask;              /* network mask */
};
```

5.4.17 Network Interface

```
struct ifaddrs {
        struct ifaddrs*         ifa_next;               /* Pointer to next struct */
        char*                   ifa_name;               /* interface name */
        unsigned int            ifa_flags;              /* flag */
        struct sockaddr*        ifa_addr;               /* Address */
        struct sockaddr*        ifa_netmask;            /* net mask */
        struct sockaddr*        ifa_broadaddr;          /* broadcast address */
        struct sockaddr*        ifa_dstaddr;            /* destination address of the P2P interface */
        void*                   ifa_data;               /* address family specific data */
};
```

5.4.18 Host Name Table Structure

```
struct hosttable {
        struct sockaddr*        addr;           /* Address */
        char*                   host;           /* host name */
        char*                   aliases;        /* aliases of host */
};
```

5.4.19 Routing Metric Structure

```
struct rt_metrics {
        unsigned long   rmx_locks;      /* flag to specify the routing
        metric not changed by the network communication manager */
        unsigned long   rmx_mtu;        /* route MTU */
        unsigned long   rmx_hopcount;   /* maximum number of hops */
        unsigned long   rmx_expire;     /* expiration of route (in seconds) (length of time until an
ARP entry is deleted) */
        unsigned long   rmx_recvpipe;   /* size of the receive socket buffer (inbound delay-bandwidth
product) */
        unsigned long   rmx_sendpipe;   /* size of the send socket buffer (outbound delay-bandwidth
product) */
        unsigned long   rmx_ssthresh;   /* buffer size of the gateway (outbound gateway buffer limit)
*/
        unsigned long   rmx_rtt;        /* round trip time (in milliseconds) */
        unsigned long   rmx_rttvar;     /* distribution of round trip time (in milliseconds) */
        unsigned long   rmx_pksent;     /* number of packets sent over the route */
};
```

5.4.20 Routing Message Header Structure

```
struct rt_msghdr {
        unsigned short  rtm_msglen;     /* message size (in bytes) */
        unsigned char   rtm_version;    /* version */
        unsigned char   rtm_type;       /* message type */
        unsigned short  rtm_index;      /* interface index */
        int             rtm_flags;      /* flag */
        int             rtm_addrs;      /* bit mask representing an address included in the message */
        ID              rtm_tid;        /* task ID of send task */
        int             rtm_seq;        /* sequence number */
        int             rtm_errno;      /* error that occurred while processing messages */
```

```
        int             rtm_use;        /* transmission counts using the corresponding route */
        unsigned long   rtm_inits;      /* flag to specify routing metric to be initialized */
        struct rt_metrics rtm_rmx;      /* routing metric */
};
```

5.4.21 Routing Message Header Structure (for RTM_IFINFO)

```
struct if_msghdr {
        unsigned short  ifm_msglen;     /* message size (in bytes) */
        unsigned char   ifm_version;    /* version */
        unsigned char   ifm_type;       /* message type */
        int             ifm_addrs;      /* bit mask representing an address in the message */
        int             ifm_flags;      /* flag */
        unsigned short  ifm_index;      /* interface index */
        struct if_data  ifm_data;       /* additional information about statistics and interface */
};
```

RTM_IFINFO message uses this structure for its header.

5.4.22 Routing Message Header Structure (for RTM_NEWADDR/RTM_DELADDR)

```
struct ifa_msghdr {
        unsigned short  ifam_msglen;    /* message size (in bytes) */
        unsigned char   ifam_version;   /* version */
        unsigned char   ifam_type;      /* message size */
        int             ifam_addrs;     /* bit mask representing an address included in the message */
        int             ifam_flags;     /* flag */
        unsigned short  ifam_index;     /* interface index */
        int             ifam_metric;    /* communication cost of the interface */
};
```

RTM_NEWADDR, RTM_DELADDR message uses this structure for its header.

5.4.23 Routing Message Header Structure (for RTM_IFANNOUNCE)

```
struct if_announcemsghdr {
        unsigned short  ifan_msglen;            /* message size (in bytes) */
        unsigned char   ifan_version;           /* version */
        unsigned char   ifan_type;              /* message type */
        unsigned short  ifan_index;             /* interface index */
        char            ifan_name[IFNAMSIZ];    /* interface name */
        unsigned short  ifan_what;              /* type of notification */
};
```

RTM_IFANNOUNCE message uses this structure for its header.

5.5 API

5.5.1 so_main - Initialize/Terminate the Socket System Service

C Language Interface

```
#include <t2ex/socket.h>

ER ercd = so_main(INT ac, UB* arg[]);
```

Parameter

| | | |
|---|---|---|
| INT | ac | Number of elements in arg[] or a negative value |
| UB* | arg[] | Array of argument strings |

Return Parameter

| | | |
|---|---|---|
| ER | ercd | Error code |

Error Code

| | |
|---|---|
| E_OK | Normal completion |
| EX_INVAL | Invalid parameters |

Description

This function initializes (ac >= 0) or terminates (ac < 0) the network communication manager in T2EX.

A number of strings can be passed to arg[] as arguments, and its total count is ac.

Content of argument is implemenatation-dependent.

### 5.5.2 so_socket - Create an Endpoint for Communication

C Language Interface

```
#include <t2ex/socket.h>

int sd = so_socket(int domain, int type, int protocol);
```

Parameter

| | | |
|---|---|---|
| int | domain | Communication domain |
| int | type | Socket type |
| int | protocol | Protocol |

Return Parameter

| | | |
|---|---|---|
| int | sd | Socket descriptor<br>or Error code |

Error Code

| | |
|---|---|
| EX_AFNOSUPPORT | The implementation does not support the specified address family. |
| EX_NFILE | No more socket descriptors are available for the system. |
| EX_PROTONOSUPPORT | The protocol is unsupported.<br>- The protocol is unsupported by the address family.<br>- The protocol is unsupported by the implementation. |
| EX_PROTOTYPE | The socket type is unsupported by the protocol. |
| EX_NOBUFS | Insufficient resources were available in the system to perform the operation. |

Description

so_socket() shall create an unbound socket in a communications domain, and return a socket descriptor that can be used in later API calls that operate on sockets.

protocol specifies a particular protocol to be used with the socket. Specifying a protocol of 0 causes so_socket() to use an unspecified default protocol appropriate for the requested socket type.

domain specifies the address family used in the communications domain. The following address families are defined;

| | |
|---|---|
| AF_INET | IPv4 protocol |
| AF_ROUTE | For the operation of a routing information database |

type specifies the socket type, which determines the semantics of communication over the socket. The following socket types are defined;

SOCK_STREAM
: Provides sequenced, reliable, bidirectional, connection-mode byte streams, and may provide a transmission mechanism for out-of-band data.

SOCK_DGRAM
: Provides datagrams, which are connectionless-mode, unreliable messages of fixed maximum length.

SOCK_RAW
: Provides interfaces to manipulate a raw packet including its header directly.

If protocol is non-zero, it shall specify a protocol that is supported by the address family. If protocol is zero, the default protocol for this address family and type shall be used.

### 5.5.3 so_close - Close a Socket Descriptor

C Language Interface

```
#include <t2ex/socket.h>

ER ercd = so_close(int sd);
```

Parameter

| | | |
|---|---|---|
| int | sd | Socket descriptor |

Return Parameter

| | | |
|---|---|---|
| ER | ercd | Error code |

Error Code

| | |
|---|---|
| E_OK | Normal completion |
| EX_BADF | sd is not a valid socket descriptor. |

Description

so_close() shall deallocate the socket descriptor pointed to by sd.

To deallocate means to make the socket descriptor available for return by subsequent calls to so_socket() or other API calls that allocate socket descriptors.
This function discards name information allocated to the socket and data in the queue.
so_close() shall cause the socket to be destroyed.

If the socket is in connection-mode, and the SO_LINGER option is set for the socket with non-zero linger time, and the socket has untransmitted data, then so_close() shall block for up to the current linger interval until all data is transmitted.
In this case, the waiting state starts even if the socket descriptor sd is set to the non-blocking mode.

If the operation is aborted by so_break(), so_close() completes normally and releases the waiting state. The socket is closed with delay.
Additionally, if the linger interval is set, the same processing occurs as when a timeout happens.

See Also

so_accept, so_getsockopt, so_socket, so_socketpair, so_setsockopt

### 5.5.4 so_accept - Accept a New Connection on a Socket

C Language Interface

```
#include <t2ex/socket.h>

int rsd = so_accept(int sd, struct sockaddr* addr, socklen_t* addrlen);
```

Parameter

| | | |
|---|---|---|
| int | sd | Socket descriptor |
| struct sockaddr* | addr | Peer address |
| socklen_t* | addrlen | Size of the peer address (in bytes) |

Return Parameter

| | | |
|---|---|---|
| int | rsd | Socket descriptor of the accepted socket, or Error code |
| struct sockaddr* | addr | Peer address |
| socklen_t* | addrlen | Actual size of the returned peer address (in bytes) |

Error Code

| | |
|---|---|
| EX_AGAIN or EX_WOULDBLOCK | O_NONBLOCK is set for sd and no connections are present to be accepted. |
| EX_BADF | sd is not a valid socket descriptor. |
| EX_CONNABORTED | A connection has been aborted. |
| EX_FAULT | addr is not in the writable address space |

| | |
|---|---|
| EX_INTR | Aborted by so_break() |
| EX_INVAL | sd is not accepting connections. |
| EX_NFILE | The maximum number of socket descriptors in the system are already open. |
| EX_OPNOTSUPP | The socket type of sd does not support accepting connections. |

Description

so_accept() accepts a new connection on sd.

so_accept() shall extract the first connection on the queue of unfinished connections, create a new socket with the same socket type protocol and address family as sd, and allocate a new socket descriptor for that socket.
This API call is available to only SOCK_STREAM type sockets.

sd is a socket that was created with so_socket(), has been bound to an address with so_bind(), and has issued a successful call to so_listen().

addr is either a null pointer, or a pointer to a sockaddr structure where the address of the connecting socket shall be returned.

addrlen points to a socklen_t structure which on input specifies the length of the supplied sockaddr structure, and on output specifies the length of the stored address.

If addr is not a null pointer, the address of the peer for the accepted connection shall be stored in the sockaddr structure pointed to by addr, and the length of this address shall be stored in the object pointed to by addrlen.

If the actual length of the address is greater than the length of the supplied sockaddr structure, the stored address shall be truncated.

If the listen queue is empty of connection requests, the following operations are performed based on the sd state flag.

If O_NONBLOCK is not set on sd;
so_accept() shall block until a connection becomes present.

If O_NONBLOCK is set on sd (non-blocking mode);
so_accept() shall fail and return the error EX_AGAIN. When a connection is available, so_select() indicates that the socket descriptor for the socket is ready for reading.

The accepted socket cannot itself accept more connections. The original socket remains open and can accept more connections.

See Also

so_bind, so_connect, so_listen, so_select, so_socket

### 5.5.5 so_bind - Bind a Name to a Socket

C Language Interface

```
#include <t2ex/socket.h>

ER ercd = so_bind(int sd, const struct sockaddr* addr, socklen_t addrlen);
```

Parameter

| | | |
|---|---|---|
| int | sd | Socket descriptor |
| const struct sockaddr* | addr | Address |
| socklen_t | addrlen | Size of the address (in bytes) |

Return Parameter

| | | |
|---|---|---|
| ER | ercd | Error code |

Error Code

| | |
|---|---|
| E_OK | Normal completion |
| EX_ADDRINUSE | The specified address is already in use. |
| EX_ADDRNOTAVAIL | The specified address is not available from the local machine. |

| | |
|---|---|
| EX_BADF | sd is not a valid socket descriptor. |
| EX_INVAL | Invalid parameters<br>- sd is already bound to an address, and the protocol does not support binding to a new address.<br>- sd has been shut down.<br>- addrlen is invalid for the address family |
| EX_FAULT | addr is not in the valid address space |

Description

so_bind() shall assign a local socket address address to a socket identified by descriptor sd that has no local socket address assigned. Sockets created with so_socket() are initially unnamed; they are identified only by their address family.

sd specifies the socket descriptor of the socket to be bound.

addr points to a sockaddr structure containing the address to be bound to the socket. The length and format of the address depend on the address family of the socket.

addrlen specifies the length of the sockaddr structure pointed to by addr.

See Also

so_connect, so_getsockname, so_listen, so_socket

### 5.5.6 so_connect - Connect a Socket

C Language Interface

```
#include <t2ex/socket.h>

ER ercd = so_connect(int sd, const struct sockaddr* addr, socklen_t addrlen);
```

Parameter

| | | |
|---|---|---|
| int | sd | Socket descriptor |
| const struct sockddr* | addr | Address |
| socklen_t | addrlen | Size of the address (in bytes) |

Return Parameter

| | | |
|---|---|---|
| ER | ercd | Error code |

Error Code

| | |
|---|---|
| E_OK | Normal completion |
| EX_ADDRNOTAVAIL | The specified address is not available from the local machine. |
| EX_AFNOSUPPORT | The specified address is not a valid address for the address family of the specified socket. |
| EX_ALREADY | A connection request is already in progress for the specified socket. |
| EX_BADF | sd is not a valid socket descriptor. |
| EX_CONNREFUSED | The target address was not listening for connections or refused the connection request. |
| EX_INPROGRESS | O_NONBLOCK is set for the socket descriptor for the socket and the connection cannot be immediately established; the connection shall be established asynchronously. |
| EX_INTR | Aborted by so_break() (the connection is established asynchronously) |
| EX_ISCONN | The specified socket is connection-mode and is already connected. |
| EX_NETUNREACH | No route to the network is present. |
| EX_TIMEDOUT | The attempt to connect timed out before a connection was made. |
| EX_ADDRINUSE | Attempt to establish a connection that uses addresses that are already in use. |
| EX_FAULT | addr is not in the valid address space |
| EX_INVAL | Invalid parameters |

Description

so_connect() shall attempt to make a connection on a connection-mode socket or to set or reset the peer address of a connectionless-mode socket.

sd specifies the socket descriptor associated with the socket.

addr points to a sockaddr structure containing the peer address. The length and format of the address depend on the address family of the socket.

addrlen specifies the length of the sockaddr structure pointed to by addr.

If the socket has not already been bound to a local address, so_connect() shall bind it to an address which is an unused local address.

For SOCK_DGRAM sockets,

so_connect() shall set the peer address of the socket's, and no connection is made. The peer address identifies where all datagrams are sent on subsequent so_send(), and limits the remote sender for subsequent so_recv(). The socket's peer address shall be reset if any of the following conditions are satisfied:
- The sa_family member of address is AF_UNSPEC.
- The address is an invalid address such as null address.

For SOCK_STREAM sockets,

so_connect() shall attempt to establish a connection to the address specified by addr.
If the connection cannot be established immediately, the following operations are performed according to the setting of the sd state flag.

If O_NONBLOCK is not set for the socket descriptor for the socket,
so_connect() shall block for up to an unspecified timeout interval until the connection is established. If the timeout interval expires before the connection is established, so_connect() shall fail and the connection attempt shall be aborted. If so_connect() is interrupted by so_break() that is caught while blocked waiting to establish a connection, so_connect() shall fail and return an error EX_INTR, but the connection request shall not be aborted, and the connection shall be established asynchronously.

If O_NONBLOCK is set for the socket descriptor for the socket,
so_connect() shall fail and return an error EX_INPROGRESS, but the connection request shall not be aborted, and the connection shall be established asynchronously. Subsequent calls to so_connect() for the same socket, before the connection is established, shall fail and return an error EX_ALREADY.

When the connection has been established asynchronously, so_select() shall indicate that the socket descriptor for the socket is ready for writing.

See Also

so_accept, so_bind, so_close, so_getsockname, so_send, so_shutdown, so_socket

### 5.5.7 so_listen - Listen for Socket Connections

C Language Interface

```
#include <t2ex/socket.h>

ER ercd = so_listen(int sd, int backlog);
```

Parameter

| | | |
|---|---|---|
| int | sd | Socket descriptor |
| int | backlog | The maximum length of the connection indication queue |

Return Parameter

| | | |
|---|---|---|
| ER | ercd | Error code |

Error Code

| | |
|---|---|
| E_OK | Normal completion |
| EX_BADF | sd is not a valid socket descriptor. |
| EX_OPNOTSUPP | The socket protocol does not support so_listen(). |
| EX_INVAL | Invalid parameters |

Description

so_listen() shall mark a connection-mode socket, specified by sd, as accepting connections.

The maximum number of connection requests held in the listening queue as yet-to-be-handled is specified by backlog.
The maximum value to be specified in backlog is defined by SOMAXCONN.
If the value more than SOMAXCONN is specified to backlog, this API call assumes that SOMAXCON is specified to backlog.
If a connection request arrives when no free space is available in the connection request waiting queue, the client receives an error indicating EX_CONNREFUSED, or the request may be ignored so that later retrials may succeed if lower protocols support retransmission.

If so_listen() is called with a backlog value that is less than 0, the API call behaves as if it had been called with a backlog argument value of 0.

See Also

so_accept, so_connect, so_socket

### 5.5.8 so_select, so_select_ms, so_select_us - Synchronous I/O Multiplexing

C Language Interface

```
#include <t2ex/socket.h>

int nfd = so_select(int nfds, fd_set* readfds, fd_set* writefds, fd_set* exceptfds, struct timeval*
tv);
int nfd = so_select_ms(int nfds, fd_set* readfds, fd_set* writefds, fd_set* exceptfds, TMO tmout);
int nfd = so_select_us(int nfds, fd_set* readfds, fd_set* writefds, fd_set* exceptfds, TMO_U tmout_u);
```

Parameter

| | | |
|---|---|---|
| int | nfds | Range of socket descriptors |
| fd_set* | readfds | Socket descriptors to be checked for being ready to read |
| fd_set* | writefds | Socket descriptors to be checked for being ready to write |
| fd_set* | exceptfds | Socket descriptors to be checked for pending error conditions |
| struct timeval* | tv | Timeout (timeval format) |
| TMO | tmout | Timeout (in milliseconds) |
| TMO_U | tmout_u | Timeout (in microseconds) |

Return Parameter

| | | |
|---|---|---|
| int | nfd | The number of ready descriptors,<br>or Error code |

Error Code

| | |
|---|---|
| EX_BADF | One or more of the socket descriptor sets specified a socket descriptor that is not a valid open socket descriptor. |
| EX_INTR | Aborted by so_break() |
| EX_INVAL | Invalid parameters<br>- An invalid timeout interval was specified.<br>- nfds is less than 0 or greater than FD_SETSIZE. |
| EX_FAULT | At least one of readfds, writefds, and exceptfds does not point at the valid address space |

Description

so_select(), so_select_ms(), and so_select_us() shall examine the socket descriptor sets whose addresses are passed in readfds, writefds, and errorfds to see whether some of their descriptors are ready for reading, are ready for writing, or have a pending exceptional condition, respectively.

nfds specifies the range of descriptors to be tested. The first nfds descriptors shall be checked in each set; that is, the descriptors from zero through nfds-1 in the descriptor sets shall be examined.

If readfds is not a null pointer, it points to an object of type fd_set that on input specifies the socket descriptors to be checked for being ready to read, and on output indicates which socket descriptors are ready to read.

If writefds is not a null pointer, it points to an object of type fd_set that on input specifies the socket descriptors to be checked for being ready to write, and on output indicates which socket descriptors are ready to write.

If errorfds is not a null pointer, it points to an object of type fd_set that on input specifies the socket descriptors to be checked for pending error conditions, and on output indicates which socket descriptors have pending error conditions.

Upon successful completion, these API calls shall modify the objects pointed to by readfds, writefds, and errorfds to indicate which socket descriptors are ready for reading, ready for writing, or have pending error conditions, respectively, and shall return the total number of ready descriptors in all the output sets. For each socket descriptor less than nfds, the corresponding bit shall be set upon successful completion if it was set on input and the associated condition is true for that socket descriptor.

If none of the selected descriptors are ready for the requested operation, these API calls shall block
- until at least one of the requested operations becomes ready,
- until the timeout occurs, or
- until interrupted by so_break().

The timeout interval until the socket descriptor meets the request is set to tv, tmout, or tmout_u, which are then used by so_select(), so_select_ms(), or so_select_us() respectively.
Specify the relative time until the timeout occurs in tv using the "timeout" structure, tmout using milliseconds, and tmout_u using microseconds.

If tv, tmout, or tmout_u indicates more than 0 seconds, it specifies a maximum interval to wait for the selection to complete. If the specified time interval expires without any requested operation becoming ready, these API calls shall return. If timout or tmout_u is TMO_FEVR or tv is a null pointer, then the call to these API calls shall block indefinitely until at least one descriptor meets the specified criteria. To effect a poll, tmout or tmout_u should be TMO_POL, or timeout should not be a null pointer, and should point to a zero-valued timespec structure.

so_select() does not change the value of tv and the value is reusable in the subsequent API call. However, the POSIX specification claims that the value of tv can be changed or unchanged (for example, FreeBSD and NetBSD do not change timeout while Linux does).
Upon considering the application portability, it is recommended that a user program initializes the value of "tv" on each call of so_select().

A descriptor shall be considered ready for reading,

in the case of a socket passed to so_recvmsg
so_recvmsg() with parameters requesting normal and ancillary data, such that the presence of either type shall cause the socket to be marked as readable. The presence of out-of-band data shall be checked if the socket option SO_OOBINLINE has been enabled, as out-of-band data is enqueued with normal data.

in the case of a socket passed to so_accept
If the socket is currently listening, then it shall be marked as readable if an incoming connection request has been received, and a call to so_accept() shall complete without blocking.

A descriptor shall be considered ready for writing,

in the case of a socket passed to so_sendmsg
If so_sendmsg() supplies an amount of normal data equal to the current value of the SO_SNDLOWAT option for the socket, the socket shall be marked as writable.

in the case of a socket passed to so_connect
If a non-blocking call to so_connect() has been made for a socket, and the connection attempt has either succeeded or failed leaving a pending error, the socket shall be marked as writable.

A socket shall be considered to have a pending exceptional condition

- If a receive operation with O_NONBLOCK clear for the open socket description and with the MSG_OOB flag set would return out-of-band data without blocking. (It is protocol-specific whether the MSG_OOB flag would be used to read out-of-band data.)
- If an out-of-band data mark is present in the receive queue.
- Other circumstances under which a socket may be considered to have a pending exceptional condition are protocol-specific.

When NULL is specified in all of readfds, writefds, and exceptfds and a positive value is set in "tmout_u" and "tmout", so_select() makes the task wait until the specified time elapses or so_break() aborts the operation.

When NULL is specified in all of readfds, writefds, and exceptfds and unlimited waiting is set on "tmout_u" and "tmout" (when "tmout_u" and "tmout" are TMO_FEVR and tv is NULL), so_select() makes the task wait until so_break() aborts the operation.

On failure, the objects pointed to by the readfds, writefds, and errorfds shall not be modified. If the timeout interval expires without the specified condition being true for any of the specified socket descriptors, the objects pointed to by the readfds, writefds, and errorfds shall have all bits set to 0.

Socket descriptor masks of type fd_set can be initialized and tested with FD_CLR(), FD_ISSET(), FD_SET(), and FD_ZERO().

FD_CLR(fd, fdsetp)
	FD_CLR shall remove the socket descriptor fd from the set specified by fdsetp. If fd is not a member of this set, there shall be no effect on the set, nor will an error be returned.

FD_ISSET(fd, fdsetp)
	FD_ISSET shall evaluate to non-zero if the socket descriptor fd is a member of the set specified by fdsetp, and shall evaluate to zero otherwise.

FD_SET(fd, fdsetp)
	FD_SET shall add the socket descriptor fd to the set specified by fdsetp. If the socket descriptor fd is already in this set, there shall be no effect on the set, nor will an error be returned.

FD_ZERO(fdsetp)
	FD_ZERO shall initialize the descriptor set specified by fdsetp to the null set. No error is returned if the set is not empty at the time FD_ZERO is invoked.

The behavior of these macros is undefined,
	- if fd is less than 0 or greater than or equal to FD_SETSIZE,
	- if fd is not a valid socket descriptor, or
	- if any of the arguments are expressions with side-effects.

### 5.5.9 so_read - Read from a Socket

C Language Interface

```
#include <t2ex/socket.h>

int nb = so_read(int sd, void* buf, size_t count);
```

Parameter

| | | |
|---|---|---|
| int | sd | Socket descriptor |
| void* | buf | Receive buffer |
| size_t | count | Size of the receive buffer (in bytes) |

Return Parameter

| | | |
|---|---|---|
| int | nb | The number of bytes actually read,<br>or Error code |
| void* | buf | Received data. |

Error Code

| | |
|---|---|
| EX_AGAIN, or EX_WOULDBLOCK | The O_NONBLOCK flag is set for the socket descriptor and no data is waiting to be received. |
| EX_BADF | sd is not a valid socket descriptor. |
| EX_INTR | Aborted by so_break() (no data has been received) |
| EX_IO | I/O error while reading from the socket |
| EX_FAULT | buf is not in the valid address space |
| EX_INVAL | Invalid parameters |
| EX_NOTCONN | A read was attempted on a socket that is not connected. |

Description

so_read() shall attempt to read count bytes from the socket associated with sd, into the buffer pointed to by buf.

If count is 0, no operation is performed.
Completes successfully if there is no parameter error. Returns a corresponding error code if there is any error.

If the value of count is greater than SSIZE_MAX, so_read() returns an error EX_INVAL.

If no data is currently available, the following operation is performed based on the sd state flag.

If O_NONBLOCK is clear,
so_read() shall block the calling thread until some data becomes available.

If O_NONBLOCK is set (non-blocking mode),
so_read() shall return an error EX_AGAIN.

The use of the O_NONBLOCK flag has no effect if there is some data available.

Upon successful completion, where count is greater than 0, so_read() shall return the number of bytes read. This number shall never be greater than count. The value returned may be less than count if the socket has fewer than counts bytes immediately available for reading.

If so_read() is interrupted by so_break() before it reads any data, it shall return an error EX_INTR.
If the operation is aborted after reading one or more bytes of data, the number of bytes read so far is returned.

so_read() shall be equivalent to so_recv() with no flags set.

See Also

so_fcntl, so_ioctl, so_recv, so_select, so_socket, so_sockpair

### 5.5.10 so_recv, so_recvfrom, so_recvmsg - Receive a Message from a Connected Socket

C Language Interface

```
#include <t2ex/socket.h>

int nb = so_recv(int sd, void* buf, size_t len, int flags);
int nb = so_recvfrom(int sd, void* buf, size_t len, int flags, struct sockaddr* src_addr, socklen_t*
addrlen);
int nb = so_recvmsg(int sd, struct msghdr* msg, int flags);
```

Parameter

| | | |
|---|---|---|
| int | sd | Socket descriptor |
| void* | buf | Receive buffer |
| size_t | len | Size of the receive buffer (in bytes) |
| struct msghdr* | msg | Message header |
| int | flags | Flags |
| struct sockaddr* | src_addr | Source address |
| socklen_t* | addrlen | Size of the source address (in bytes) |

Return Parameter

| | | |
|---|---|---|
| int | nb | The length of the message (in bytes),<br>or Error code |
| void* | buf | Received data |

Error Code

| | |
|---|---|
| EX_AGAIN or EX_WOULDBLOCK | The socket descriptor of the socket is marked O_NONBLOCK and no data is waiting to be received; or MSG_OOB is set and no out-of-band data is available and either the socket descriptor of the socket is marked O_NONBLOCK or the socket does not support blocking to await out-of-band data. |
| EX_BADF | sd is not a valid socket descriptor. |

| | |
|---|---|
| EX_INTR | Aborted by so_break() (no data has been received) |
| EX_INVAL | Invalid parameter.<br>- The sum of the iov_len values is greater than SSIZE_MAX.<br>- The MSG_OOB flag is set and no out-of-band data is available. |
| EX_NOTCONN | A receive is attempted on a connection-mode socket that is not connected. |
| EX_OPNOTSUPP | The specified flags are unsupported for this socket type or protocol. |
| EX_FAULT | Receive buffer is not in a valid address space |

For so_recvmsg(),

| | |
|---|---|
| EX_MSGSIZE | The msg_iovlen member of the msghdr structure pointed to by msg is less than or equal to 0, or is greater than IOV_MAX. |

Description

so_recv(), so_recvfrom(), and so_recvmsg() shall receive a message from a connection-mode or connectionless-mode socket.

so_recvfrom() and so_recvmsg() shall receive a message from a connection-mode or connectionless-mode socket. It is normally used with connectionless-mode sockets because it permits the application to retrieve the source address of received data.

so_recv() is normally used with connected sockets because it does not permit the application to retrieve
the source address of received data.

sd specifies the socket descriptor.

buf points to a buffer where the message should be stored.

len is the length in bytes of the buffer pointed to by buf.

msg points to a msghdr structure, containing both the buffer to store the source address and the buffers for the incoming message. The length and format of the address depend on the address family of the socket.

In the msghdr structure, the msg_name and msg_namelen members specify the source address if the socket is unconnected. If the socket is connected, the msg_name and msg_namelen members shall be ignored. The msg_name member may be a null pointer if no names are desired or required.
The msg_flags member is ignored on input, but may contain meaningful values on output. Upon successful completion, the msg_flags member of the message header shall be the bitwise-inclusive OR of all of the following flags that indicate conditions detected for the received message:

| | |
|---|---|
| MSG_EOR | End-of-record was received (if supported by the protocol). |
| MSG_OOB | Out-of-band data was received. |
| MSG_TRUNC | Normal data was truncated. |
| MSG_CTRUNC | Control data was truncated. |

The msg_iov and msg_iovlen fields are used to specify where the received data shall be stored. msg_iov points to an array of iovec structures; msg_iovlen shall be set to the dimension of this array.

In each iovec structure, the iov_base field specifies a storage area and the iov_len field gives its size in bytes. Each storage area indicated by msg_iov is filled with received data in turn until all of the received data is stored or all of the areas have been filled.

Specify a pointer to the "cmsghdr" structure to write an auxiliary data of the protocol in msg_control of the "msghdr" structure, and the size of msg_control in msg_controllen in bytes.

flags specifies the type of message reception. Values of flags are formed by logically OR'ing zero or more of the following values:

MSG_PEEK
Peeks at an incoming message. The data is treated as unread and the next so_recv(), so_recvfrom(), so_recvmsg() shall still return this data.

MSG_OOB
Requests out-of-band data. The significance and semantics of out-of-band data are protocol-specific.

MSG_WAITALL
On SOCK_STREAM sockets this requests that the API call block until the full amount

of data can be returned. The API call may return the smaller amount of data,
- if the socket is a message-based socket,
- if aborted by so_break(),
- if the connection is terminated,
- if MSG_PEEK was specified, or
- if an error is pending for the socket.

src_addr is a null pointer, or points to a sockaddr structure in which the sending address is to be stored. The length and format of the address depend on the address family of the socket.

addrlen is the length of the sockaddr structure pointed to by src_addr.

so_recv() and so_recvfrom() shall return the length of the message written to the buffer pointed to by buf, and so_recvmsg() shall return the length of the message in bytes. For message-based sockets, such as SOCK_DGRAM, the entire message shall be read in a single operation. If a message is too long to fit in the supplied buffer, and MSG_PEEK is not set in flags, the excess bytes shall be discarded. For stream-based sockets, such as SOCK_STREAM, message boundaries shall be ignored. In this case, data shall be returned to the user as soon as it becomes available, and no data shall be discarded.

If the MSG_WAITALL flag is not set, data shall be returned only up to the end of the first message.

If no messages are available at the socket,

if O_NONBLOCK is not set on sd,
so_recv(), so_recvfrom(), and so_recvmsg() shall block until a message arrives.

if O_NONBLOCK is set on sd,
so_recv(), so_recvfrom(), and so_recvmsg() shall fail and return an error EX_AGAIN.

Not all protocols provide the source address for messages. If src_addr is not a null pointer and the protocol provides the source address of messages, the source address of the received message shall be stored in the sockaddr structure pointed to by src_addr, and the length of this address shall be stored in the object pointed to by addrlen.
If the actual length of the address is greater than the length of the supplied sockaddr structure, the stored address shall be truncated.

If src_addr is not a null pointer and the protocol does not provide the source address of messages, the value stored in the object pointed to by src_addr is unspecified.

### 5.5.11 so_write - Write on a Socket

C Language Interface

```
#include <t2ex/socket.h>

int nb = so_write(int sd, const void* buf, size_t count);
```

Parameter

| | | |
|---|---|---|
| int | sd | Socket descriptor |
| const void* | buf | Send buffer |
| size_t | count | Size of the send buffer (in bytes) |

Return Parameter

| | | |
|---|---|---|
| int | nb | The number of bytes actually written,<br>or Error code |

Error Code

| | |
|---|---|
| EX_AGAIN, or EX_WOULDBLOCK | The O_NONBLOCK flag is set for the socket descriptor and the task would be delayed in the so_write() operation. |
| EX_BADF | sd is not a valid socket descriptor open for writing. |
| EX_INTR | Aborted by so_break() |
| EX_FAULT | The area specified in buf is not in the valid address space |
| EX_HOSTUNREACH | Message cannot reach the destination |
| EX_HOSTDOWN | The destination is on the local subnet and does not respond to arp |
| EX_INVAL | Invalid parameters. |
| EX_NOTCONN | Attempted to send using a connection type socket for which the connection is |

not established

Description

so_write() shall attempt to write count bytes from the buffer pointed to by buf to the socket associated with sd.

If count is zero, it completes normally without sending any data.

If so_write() is interrupted by so_break() before it writes any data, it shall return an error EX_INTR.

If so_write() is interrupted by so_break() after it successfully writes some data, it shall return the number of bytes written.

If the value of count is greater than SSIZE_MAX, so_write() shall return an error EX_INVAL.

When attempting to write to a socket descriptor and cannot accept the data immediately:

If the O_NONBLOCK flag is clear,
so_write() shall block the calling task until the data can be accepted.

If the O_NONBLOCK flag is set,
so_write() shall return an error EX_AGAIN.

so_write() shall be equivalent to so_send() with no flags set.

See Also

so_read, so_select

### 5.5.12 so_send, so_sendto, so_sendmsg - Send a Message on a Socket

C Language Interface

```
#include <t2ex/socket.h>

int nb = so_send(int sd, const void* buf, size_t len, int flags);
int nb = so_sendto(int sd, const void* buf, size_t len, int flags, const struct sockaddr* dest_addr,
socklen_t addrlen);
int nb = so_sendmsg(int sd, const struct msghdr* msg, int flags);
```

Parameter

| | | |
|---|---|---|
| int | sd | Socket descriptor |
| const void* | buf | Send buffer |
| size_t | len | Size of the send buffer (in bytes) |
| const struct msghdr* | msg | Message header |
| int | flags | Flags |
| const struct sockaddr* | dest_addr | Destination address |
| socklen_t | addrlen | Size of the destination address (in bytes) |

Return Parameter

| | | |
|---|---|---|
| int | nb | The number of bytes sent,<br>or Error code |

Error Code

| | |
|---|---|
| EX_AGAIN, or EX_WOULDBLOCK | The socket descriptor of the socket is marked O_NONBLOCK and the requested operation would block. |
| EX_BADF | sd is not a valid socket descriptor. |
| EX_INTR | Aborted by so_break() |
| EX_INVAL | Invalid parameters<br>- The sum of the iov_len values is greater than SSIZE_MAX. |
| EX_NOBUFS | Insufficient resources were available in the system to perform the operation. |
| EX_FAULT | The area specified in the argument is not in the valid address space |
| EX_DESTADDRREQ | The socket is not a connection type and a communication address is not set |
| EX_HOSTUNREACH | Message cannot reach the destination |

EX_HOSTDOWN The destination is on the local subnet and does not respond to arp
EX_AFNOSUPPORT Specified address family cannot be used by this socket.
EX_NOTCONN The socket is not connected.

For so_sendto(),
EX_ISCONN A destination address was specified and the socket is already connected.

For so_sendmsg(),
EX_MSGSIZE The msg_iovlen member of the msghdr structure pointed to by msg is non-positive
or is greater than IOV_MAX.

Description

so_send(), so_sendto(), and so_sendmsg() shall initiate transmission of a message from the specified socket to its peer.

so_send() shall send a message only when the socket is connected. If the socket is a connectionless-mode socket, the message shall be sent to the pre-specified peer address.

so_sendto() shall send a message through a connection-mode or connectionless-mode socket. If the socket is a connectionless-mode socket, the message shall be sent to the address specified by dest_addr if no pre-specified peer address has been set. If a peer address has been pre-specified, so_sendto() shall return an error EX_ISCONN. If the socket is connection-mode, dest_addr shall be ignored.

so_sendmsg() shall send a message through a connection-mode or connectionless-mode socket. If the socket is a connectionless-mode socket, the message shall be sent to the address specified by msghdr if no pre-specified peer address has been set. If a peer address has been pre-specified, the message shall be sent to the address specified in msghdr (overriding the pre-specified peer address). If the socket is connection-mode, the destination address in msghdr shall be ignored.

sd specifies the socket descriptor.

buf points to the buffer containing the message to send.

len specifies the length of the message in bytes.

msg points to a msghdr structure, containing both the destination address and the buffers for the outgoing message. The length and format of the address depend on the address family of the socket. The msg_flags member is ignored.

The msg_iov and msg_iovlen fields of msg specify zero or more buffers containing the data to be sent. msg_iov points to an array of iovec structures; msg_iovlen shall be set to the dimension of this array. In each iovec structure, the iov_base field specifies a storage area and the iov_len field gives its size in bytes. Some of these sizes can be zero. The data from each storage area specified by msg_iov is sent in turn.

flags specifies the type of message transmission. Values of flags are formed by logically OR'ing zero or more of the following flags:

MSG_EOR
Terminates a record (if supported by the protocol).
MSG_OOB
Sends out-of-band data on sockets that support out-of-band communications. The significance and semantics of out-of-band data are protocol-specific.

dest_addr points to a sockaddr structure containing the destination address. The length and format of the address depend on the address family of the socket.

addrlen is the length of the sockaddr structure pointed to by dest_addr.

The length of the message to be sent is specified by len. If the message is too long to pass through the underlying protocol, so_send() shall fail and no data shall be transmitted.

Successful completion of a call to so_send(), so_sendto(), or so_sendmsg() does not guarantee delivery of the message. A return value of a negative value indicates only locally-detected errors.

If space is not available at the sending socket to hold the message to be transmitted,

if the socket descriptor does not have O_NONBLOCK set,
so_send(), so_sendto(), and so_sendmsg() shall block until space is available.

if the socket descriptor does have O_NONBLOCK set,
so_send(), so_sendto(), and so_sendmsg() shall fail. so_select() can be used to determine when it is possible to send more data.

If the socket protocol supports broadcast and the specified address is a broadcast address for the socket protocol, so_sendmsg() and so_sendto() shall fail if the SO_BROADCAST option is not set for the socket.

### 5.5.13 so_getpeername - Get the Name of the Peer Socket

C Language Interface

```
#include <t2ex/socket.h>

ER ercd = so_getpeername(int sd, struct sockaddr* addr, socklen_t* addrlen);
```

Parameter

| | | |
|---|---|---|
| int | sd | Socket descriptor |
| struct sockaddr* | addr | Peer address |
| socklen_t* | addrlen | Size of the peer address (in bytes) |

Return Parameter

| | | |
|---|---|---|
| ER | ercd | Error code |
| struct sockaddr* | addr | Peer address |
| socklen_t* | addrlen | Actual size of the returned peer address (in bytes) |

Error Code

| | |
|---|---|
| E_OK | Normal completion |
| EX_BADF | sd is not a valid socket descriptor. |
| EX_NOTCONN | The socket is not connected or otherwise has not had the peer pre-specified. |
| EX_NOBUFS | Insufficient resources were available in the system to complete the call. |
| EX_FAULT | addr is not in the valid address space |
| EX_INVAL | Invalid parameters |

Description

so_getpeername() shall retrieve the peer address of the specified socket, store this address in the sockaddr structure pointed to by addr, and store the length of this address in the object pointed to by addrlen.

The socket descriptor of the socket to search the address of the destination is specified in sd.

addr is a NULL pointer or a pointer to the "sockaddr" structure.
If addr is a pointer to the "sockaddr" structure, the connected socket address is written to addr.

addrlen is a pointer to socklen_t data type and used for both calling this API call and returning from this API call.
When this API call is invoked, the size of the "sockaddr" structure passed to the API call is specified in addrlen.
When this API call returns, the written address size is stored in addrlen.

If the actual length of the address is greater than the length of the supplied sockaddr structure, the stored address shall be truncated.

See Also

so_accept, so_bind, so_getsockname, so_socket

### 5.5.14 so_getsockname - Get a Socket Name

C Language Interface

```
#include <t2ex/socket.h>

ER ercd = so_getsockname(int sd, struct sockaddr* addr, socklen_t* addrlen);
```

Parameter

| | | |
|---|---|---|
| int | sd | Socket descriptor |
| struct sockaddr* | addr | Address |
| socklen_t* | addrlen | Size of the address (in bytes) |

Return Parameter

| | | |
|---|---|---|
| ER | ercd | Error code |
| struct sockaddr* | addr | Address |
| socklen_t* | addrlen | Actual size of the returned address (in bytes) |

Error Code

| | |
|---|---|
| E_OK | Normal completion |
| EX_BADF | sd is not a valid socket descriptor. |
| EX_INVAL | The socket has been shut down. |
| EX_NOBUFS | Insufficient resources were available in the system to complete the function. |
| EX_FAULT | addr is not in the valid address space |

Description

so_getsockname() shall retrieve the locally-bound name of the specified socket, store this address in the sockaddr structure pointed to by addr, and store the length of this address in the object pointed to by addrlen.

The socket descriptor of the socket to search the local name is specified in sd.

addr is a NULL pointer or a pointer to the "sockaddr" structure.
If addr is a pointer to the "sockaddr" structure, the connected socket address is written.

addrlen is a pointer to socklen_t data type and used for both calling this API call and returning from this API call.
When this API call is invoked, the length of the "sockaddr" structure passed to the API call is specified in addrlen.
When this API call returns, the written address size is stored in addrlen.

If the actual length of the address is greater than the length of the supplied sockaddr structure, the stored address shall be truncated.

If the socket has not been bound to a local name, the value stored in the object pointed to by address is unspecified.

See Also

so_accept, so_bind, so_getpeername, so_socket

### 5.5.15 so_getsockopt - Get a Socket Option

C Language Interface

```
#include <t2ex/socket.h>

ER ercd = so_getsockopt(int sd, int level, int optname, void* optval, socklen_t* optlen);
```

Parameter

| | | |
|---|---|---|
| int | sd | Socket descriptor |
| int | level | Level |
| int | optname | Option |
| void* | optval | Option value |
| socklen_t* | optlen | Size of the option value (in bytes) |

Return Parameter

| | | |
|---|---|---|
| ER | ercd | Error code |
| void* | optval | Returned option value |
| socklen_t* | optlen | Actual size of the returned option value (in bytes) |

Error Code

| | |
|---|---|
| E_OK | Normal completion |
| EX_BADF | sd is not a valid socket descriptor. |
| EX_INVAL | Invalid parameters<br>- The specified option is invalid at the specified socket level.<br>- The socket has been shut down. |
| EX_NOPROTOOPT | The option is unsupported by the protocol. |
| EX_FAULT | optval or optlen are not in the valid address space |

Description

so_getsockopt() shall retrieve the value for the option specified by optname for the socket specified by sd.

If the size of the option value is greater than optlen, the value stored in the object pointed to by optval shall be silently truncated. Otherwise, the object pointed to by the optlen shall be modified to indicate the actual length of the value.

level specifies the protocol level at which the option resides. To retrieve options at the socket level, specify level as SOL_SOCKET. To retrieve options at other levels, supply the appropriate level identifier for the protocol controlling the option. For example, to indicate that an option is interpreted by the TCP, set level to IPPROTO_TCP.
The following level identifiers can be used.

| | |
|---|---|
| IPPROTO_IP | The IP level |
| IPPROTO_TCP | The TCP level |
| SOL_SOCKET | The socket level |

optname specifies a single option to be retrieved.

If IPPROTO_IP is specified in level, the following options can be specified in optname.

| | |
|---|---|
| IP_OPTIONS | IP options embedded in the IP header of each packet to be sent |
| IP_HDRINCL | Enables/disables addition of the IP header to the sent data by an application |

If IPPROTO_TCP is specified in level, the following options can be specified in optname.

| | |
|---|---|
| TCP_NODELAY | Enables/disables the immediate transmission of data immediately |
| TCP_MAXSEG | Maximum length of a segment |

If SOL_SOCKET is specified in level, the following options can be specified in optname.

| | |
|---|---|
| SO_DEBUG | Debugging in the underlying protocol modules. |
| SO_ACCEPTCONN | Waiting state of so_listen(). |
| SO_REUSEADDR | Reuse of local addresses. |
| SO_KEEPALIVE | Periodic transmission of keepalive messages. |
| SO_DONTROUTE | Bypass of normal routing; route based on destination address only. |
| SO_BROADCAST | Permission to transmit broadcast datagrams. |
| SO_USELOOPBACK | Enables/disables functions to communicate bypassing the hardware |
| SO_LINGER | Actions to be taken for queued, unsent data on so_close(). |
| SO_OOBINLINE | Out-of-band data be placed into normal data input queue as received. |
| SO_REUSEPORT | Enables/disables to reuse of the local address and the port |
| SO_TIMESTAMP | Enables/disables to add timestamps to the received datagram |
| SO_SNDBUF | Size of send buffer (in bytes) |
| SO_RCVBUF | Size of receive buffer (in bytes) |
| SO_SNDLOWAT | Minimum amount of data to send for output operations (in bytes) |
| SO_RCVLOWAT | Minimum amount of data to return to application for input operations (in bytes). |
| SO_SNDTIMEO | Timeout value for a socket send operation. |
| SO_RCVTIMEO | Timeout value for a socket receive operation |
| SO_ERROR | Pending error information on the socket |
| SO_TYPE | Socket type |

See Also

so_ioctl, so_select, so_socket

5.5.16 so_setsockopt - Set a Socket Option

C Language Interface

```
#include <t2ex/socket.h>

ER ercd = so_setsockopt(int sd, int level, int optname, const void* optval, socklen_t optlen);
```

Parameter

| | | |
|---|---|---|
| int | sd | Socket descriptor |
| int | level | Level |
| int | optname | Option |
| const void* | optval | Option value |
| socklen_t | optlen | Size of the option value (in bytes) |

Return Parameter

| | | |
|---|---|---|
| ER | ercd | Error code |

Error Code

| | |
|---|---|
| E_OK | Normal completion |
| EX_BADF | sd is not a valid socket descriptor. |
| EX_INVAL | Invalid parameters<br>- The specified option is invalid at the specified socket level<br>- The socket has been shut down. |
| EX_NOPROTOOPT | The option is unsupported by the protocol. |
| EX_FAULT | optval or optlen are not in the valid address space |

Description

so_setsockopt() shall set the option specified by the optname, at the protocol level specified by level, to the value pointed to by optval for the socket associated with the socket descriptor specified by sd.

level specifies the protocol level at which the option resides. To set options at the socket level, specify level as SOL_SOCKET. To set options at other levels, supply the appropriate level identifier for the protocol controlling the option. For example, to indicate that an option is interpreted by the TCP.
The following level identifiers can be used.

| | |
|---|---|
| IPPROTO_IP | The IP level |
| IPPROTO_TCP | The TCP level |
| SOL_SOCKET | The socket level |

optname specifies a single option to set.

If IPPROTO_IP is specified in level, the following options can be specified in optname.

| | |
|---|---|
| IP_OPTIONS | IP options embedded in the IP header of each packet to be sent |
| IP_HDRINCL | Enables/disables to assign the IP header to the sent data by an application |

If IPPROTO_TCP is specified in level, the following options can be specified in optname.

| | |
|---|---|
| TCP_NODELAY | Enables/disables the immediate transmission of data |
| TCP_MAXSEG | Maximum length of a segment |

If SOL_SOCKET is specified in level, the following options can be specified in optname.

| | |
|---|---|
| SO_DEBUG | Debugging in the underlying protocol modules. |
| SO_REUSEADDR | Reuse of local addresses. |
| SO_KEEPALIVE | Periodic transmission of keepalive messages. |
| SO_DONTROUTE | Bypass of normal routing; route based on destination address only. |
| SO_BROADCAST | Permission to transmit broadcast datagrams. |
| SO_USELOOPBACK | Enables/disables functions to communicate bypassing the hardware |
| SO_LINGER | Actions to be taken for queued, unsent data on so_close(). |
| SO_OOBINLINE | Out-of-band data be placed into normal data input queue as received. |
| SO_REUSEPORT | Enables/disables the reuse of the local address and the port |
| SO_TIMESTAMP | Enables/disables the addition of timestamps to the received datagram |
| SO_SNDBUF | Size of send buffer (in bytes) |
| SO_RCVBUF | Size of receive buffer (in bytes) |
| SO_SNDLOWAT | Minimum amount of data to send for output operations (in bytes) |
| SO_RCVLOWAT | Minimum amount of data to return to application for input operations (in bytes). |
| SO_SNDTIMEO | Timeout value for a socket send operation. |
| SO_RCVTIMEO | Timeout value for a socket receive operation |

5.5.17 so_gethostname - Get the Current Host Name

C Language Interface

```
#include <t2ex/socket.h>

ER ercd = so_gethostname(char* name, size_t len);
```

Parameter

| | | |
|---|---|---|
| char* | name | Pointer to the area to return the host name |
| size_t | len | Size of the area pointed by name (in bytes) |

Return Parameter

| | | |
|---|---|---|
| ER | ercd | Error code |
| char* | name | Host name |

Error Code

| | |
|---|---|
| E_OK | Normal completion |
| EX_FAULT | The area specified in name is not in the valid address space |
| EX_INVAL | Invalid parameters |

Description

so_gethostname() shall return the standard host name for the current machine.

len is the size of the array pointed to by name. The returned name is null-terminated, except that if len is too short hold the host name, then the returned name shall be truncated and it is unspecified whether the returned name is null-terminated.

Host names are limited to HOST_NAME_MAX bytes.

See Also

so_sethostname

5.5.18 so_sethostname - Set a Host Name

C Language Interface

```
#include <t2ex/socket.h>

ER ercd = so_sethostname(const char* name, size_t len);
```

Parameter

| | | |
|---|---|---|
| const char* | name | Name of host |
| size_t | len | Size of the host name (in bytes) |

Return Parameter

| | | |
|---|---|---|
| ER | ercd | Error code |

Error Code

| | |
|---|---|
| E_OK | Normal completion |
| EX_FAULT | The area specified in name is not in the valid address space |
| EX_INVAL | Illegal parameter |

Description

This function sets the host name specified by name to the current machine.

The size of array represented by name is specified in len.
Usually, this API call is used immediately after startup.

See Also

so_gethostname

### 5.5.19 so_getaddrinfo, so_getaddrinfo_ms, so_getaddrinfo_us - Get Address Information

C Language Interface

```
#include <t2ex/socket.h>

int len = so_getaddrinfo(const char* node, const char* service, const struct addrinfo* hints, struct
addrinfo** res, void* buf, size_t bufsz, struct timeval* timeout);
int len = so_getaddrinfo_ms(const char* node, const char* service, const struct addrinfo* hints,
struct addrinfo** res, void* buf, size_t bufsz, TMO tmout);
int len = so_getaddrinfo_us(const char* node, const char* service, const struct addrinfo* hints,
struct addrinfo** res, void* buf, size_t bufsz, TMO_U tmout_u);
```

Parameter

| | | |
|---|---|---|
| const char* | node | Service location |
| const char* | service | Service |
| const struct addrinfo* | hints | Hints containing input values that directs the operation |
| struct addrinfo** | res | Pointer to the linked list of results |
| void* | buf | Pointer to the area to store results |
| size_t | bufsz | Size of the area pointed by buf (in bytes) |
| | | |
| struct timeval* | timeout | Timeout (timeval format) |
| TMO | tmout | Timeout (in milliseconds) |
| TMO_U | tmout_u | Timeout (in microseconds) |

Return Parameter

| | | |
|---|---|---|
| int | len | Size of the buffer to store results (in bytes),<br>or Error code |
| struct addrinfo** | res | Pointer to the linked list of results |
| void* | buf | The linked list of results |

Error Code

| | |
|---|---|
| EX_TIMEDOUT | Timeout occured before completing the operation |
| EX_INTR | Aborted by so_break() |
| EX_INVAL | Invalid parameters |
| EX_AI_AGAIN | The name could not be resolved at this time. Future attempts may succeed. |
| EX_AI_BADFLAGS | ai_flags had an invalid value. |
| EX_AI_FAIL | A non-recoverable error occurred when attempting to resolve the name. |
| EX_AI_FAMILY | The address family was not recognized. |
| EX_AI_NONAME | The name does not resolve for the supplied parameters.<br>(Neither node nor service were supplied. At least one of these shall be supplied.) |
| EX_AI_SERVICE | The service passed was not recognized for the specified socket type. |
| EX_AI_SOCKTYPE | The intended socket type was not recognized. |

Description

so_getaddrinfo() shall translate the name of a service location (for example, a host name) and/or a service name and shall return a set of socket addresses and associated information to be used in creating a socket with which to address the specified service.

node and service are either null pointers or pointers to null-terminated strings. One or both of these two arguments shall be supplied by the application as a non-null pointer.

The format of a valid name depends on the address family or families. If a specific family is not given and the name could be interpreted as valid within multiple supported families, the implementation shall attempt to resolve the name in all supported families and, in absence of errors, one or more results shall be returned.

If node is not null, it can be a descriptive name or can be an address string. If the specified address family is AF_INET or AF_UNSPEC, valid descriptive names include host names. If the specified address family is AF_INET or AF_UNSPEC, address strings using Internet standard dot notation as specified in inet_addr are valid.

If node is not null, the requested service location is named by node; otherwise, the requested service location is local to the caller.

If service is null, the call shall return network-level addresses for the specified node. If service is not null, it is a null-terminated character string identifying the requested service. This can be either a descriptive name or a numeric representation suitable for use with the address family or families. If the specified address family is AF_INET or AF_UNSPEC, the service can be specified as a string specifying a decimal port number.

If hints is not null, it refers to a structure containing input values that directs the operation by providing options and by limiting the returned information to a specific socket type, address family, and/or protocol, as described below. In this hints structure every member other than ai_flags, ai_family, ai_socktype, and ai_protocol shall be set to zero or a null pointer. If hints is a null pointer, the behavior shall be as if it referred to a structure containing the value zero for the ai_flags, ai_socktype, and ai_protocol fields, and AF_UNSPEC for the ai_family field.

A value of zero for ai_protocol means that the caller shall accept any protocol.

ai_family
: The ai_family field to which argument hints points specifies the address familly for the service.

  If the ai_family field to which hints points has the value AF_UNSPEC, addresses shall be returned for use with any address family that can be used with the specified node and/or service. Otherwise, addresses shall be returned for use only with the specified address family. If ai_family is not AF_UNSPEC and ai_protocol is not zero, then addresses shall be returned for use only with the specified address family and protocol.

ai_socktype
: The ai_socktype field to which argument hints points specifies the socket type for the service, as defined in socket.
  0, SOCK_STREAM, SOCK_DGRAM, or SOCK_RAW can be specified in the field.

  If a specific socket type is not given (for example, a value of zero) and the service name could be interpreted as valid with multiple supported socket types, the implementation shall attempt to resolve the service name for all supported socket types and, in the absence of errors, all possible results shall be returned. A non-zero socket type value shall limit the returned information to values with the specified socket type.

ai_protocol
: The protocol required by the service is specified in the ai_protocol field.
  IPPROTO_UDP or IPPROTO_TCP can be specified in the field.

  A value of zero for ai_protocol means that the caller shall accept any protocol.

  If ai_family is not AF_UNSPEC and ai_protocol is not zero, then addresses shall be returned for use only with the specified address family and protocol.

ai_flags
: The ai_flags field to which the hints parameter points shall be set to zero or be the bitwise-inclusive OR of one or more of the values AI_PASSIVE, AI_CANONNAME, AI_NUMERICHOST, and AI_NUMERICSERV.

  AI_CANONNAME
  : If the AI_CANONNAME flag is specified and node is not null, the API call shall attempt to determine the canonical name corresponding to node.

  AI_NUMERICHOST
  : If the AI_NUMERICHOST flag is specified, then a non-null node string supplied shall be a numeric host address string. Otherwise, an EX_AI_NONAME error is returned. This flag shall prevent any type of name resolution service (for example, the DNS) from being invoked.

  AI_NUMERICSERV
  : If the AI_NUMERICSERV flag is specified, then a non-null service string supplied shall be a numeric port string. Otherwise, an EX_AI_NONAME error shall be returned. This flag shall prevent any type of name resolution service (for example, NIS+) from being invoked.

  AI_PASSIVE
  : If the AI_PASSIVE flag is specified, the returned address information shall be suitable for use in binding a socket for accepting incoming connections for the

> specified service. In this case, if node is null, then the IP address portion of the socket address structure shall be set to INADDR_ANY for an IPv4 address. The AI_PASSIVE flag shall be ignored if node is not null.
>
> If the AI_PASSIVE flag is not specified, the returned address information shall be suitable for a call to so_connect() (for a connection-mode protocol) or for a call to so_connect(), so_sendto(), or so_sendmsg() (for a connectionless protocol). In this case, if node is null, then the IP address portion of the socket address structure shall be set to the loopback address.

buf is a pointer to the buffer area to store the result. Its size is specified in bufsz in bytes.
getaddrinfo() in the POSIX specification allocates a memory space dynamically, store the result there, and set res as its pointer.
On the other hand, so_getaddrinfo(), so_getaddrinfo_ms(), and so_getaddrinfo_us() store results in the area allocated by buf to avoid any dynamic allocation of memory through an API call.
They return buffer size required to store all the result entries as return code when the name resolution completes normally.

If the memory size required to store the result is larger than "bufsz", entries are stored in buf as many as possible.
If no entries can be stored in buf, res is set to NULL.

In addition, because the memory space is not allocated dynamically in API calls as stated above, the memory space storing the result of so_getaddrinfo() does not need to be released.
Therefore, unlike the POSIX specification, there is no API call equivalent to freeaddrinfo() to release the result memory area.

Timeout interval for completing the name resolution operation is specified in timeout, tmout, or tmout_u, which is then used in so_getaddrinfo(), so_getaddrinfo_ms(), or so_getaddrinfo_us() respectively.
Specify the relative time until the timeout occurs in timeout using the "timeout" structure, tmout using milliseconds, and tmout_u using microseconds.

If timeout, tmout, or tmout_u is positive, and the name resolution does not complete within the specified time, the name resolution process is aborted and EX_TIMEDOUT is returned.
To try to resolve the same name again in this case, start from scratch instead of continuing this name resolution process.
If tmout and tmout_u are set to TMO_FEVR and timeout is set to NULL, the behavior is the same as that of the getaddrinfo() in the POSIX specification except for memory space allocation for the result.

Upon successful return of so_getaddrinfo(), the location to which res points shall refer to a linked list of addrinfo structures, each of which shall specify a socket address and information for use in creating a socket with which to use that socket address. The list shall include at least one addrinfo structure. The ai_next field of each structure contains a pointer to the next structure on the list, or a null pointer if it is the last structure on the list. Each structure on the list shall include values for use with a call to so_socket(), and a socket address for use with so_connect() or, if the AI_PASSIVE flag was specified, for use with so_bind(). The fields ai_family, ai_socktype, and ai_protocol shall be usable as the arguments to so_socket() to create a socket suitable for use with the returned address. The fields ai_addr and ai_addrlen are usable as the arguments to so_connect() or so_bind() with such a socket, according to the AI_PASSIVE flag.

If node is not null, and if requested by the AI_CANONNAME flag, the ai_canonname field of the first returned addrinfo structure shall point to a null-terminated string containing the canonical name corresponding to the input node; if the canonical name is not available, then ai_canonname shall refer to node or a string with the same contents. The contents of the ai_flags field of the returned structures are undefined.

If so_break() is issued for the task waiting for completion of this API call process before the so_getaddrinfo() operation completes, this API call aborts the name resolution process and returns EX_INTR.
To try to resolve the same name again in this case, start from the scratch instead of continuing this name resolution process.

Because gethostbyname and gethostbyaddr defined in the POSIX specification are non-thread-safe, and they can be replaced by this function, T2EX does not provide API calls equivalent to these.

See Also

so_bind, so_connect, so_getnameinfo, so_socket

### 5.5.20 so_getnameinfo, so_getnameinfo_ms, so_getnameinfo_us - Get Name Information

C Language Interface

```
#include <t2ex/socket.h>

ER ercd = so_getnameinfo(const struct sockaddr* sa, socklen_t salen, char* host, size_t hostlen, char*
serv, size_t servlen, int flags, struct timeval* timeout);
ER ercd = so_getnameinfo_ms(const struct sockaddr* sa, socklen_t salen, char* host, size_t hostlen,
char* serv, size_t servlen, int flags, TMO tmout);
ER ercd = so_getnameinfo_us(const struct sockaddr* sa, socklen_t salen, char* host, size_t hostlen,
char* serv, size_t servlen, int flags, TMO_U tmout_u);
```

Parameter

| | | |
|---|---|---|
| const struct sockaddr* | sa | Address to be translated |
| socklen_t | salen | Size of the address to be translated (in bytes) |
| char* | host | Pointer to the area to return the host name. |
| size_t | hostlen | Size of the area pointed by host (in bytes) |
| char* | serv | Pointer to the area to return the service name. |
| size_t | servlen | Size of the area pointed by serv (in bytes) |
| int | flags | Flags |
| struct timeval* | timeout | Timeout (timeval format) |
| TMO | tmout | Timeout (in milliseconds) |
| TMO_U | tmout_u | Timeout (in microseconds) |

Return Parameter

| | | |
|---|---|---|
| ER | ercd | Error code |
| char* | host | Host name |
| char* | serv | Service name |

Error Code

| | |
|---|---|
| E_OK | Normal completion |
| EX_TIMEDOUT | Timeout occured before completing the operation |
| EX_INTR | Aborted by so_break() |
| EX_INVAL | Invalid parameters |
| EX_AI_AGAIN | The name could not be resolved at this time. Future attempts may succeed. |
| EX_AI_BADFLAGS | ai_flags had an invalid value. |
| EX_AI_FAIL | A non-recoverable error occurred when attempting to resolve the name. |
| EX_AI_FAMILY | Invalid address family<br>- The address family was not recognized<br>- The address length was invalid. |
| EX_AI_NONAME | The name does not resolve for the supplied parameters.<br>- NI_NAMEREQD is set and the host's name cannot be located<br>- Both nodename and servname were null. |
| EX_AI_OVERFLOW | An argument buffer overflowed.<br>- The buffer pointed to by node or service was too small. |
| EX_AI_SYSTEM | An internal error occurred.<br>- Converting an address to a string is failed. |
| EX_AI_SOCKTYPE | The intended socket type was not recognized. |

Description

so_getnameinfo() shall translate a socket address to a node name and service location.

sa points to a socket address structure to be translated.

If host is non-NULL and hostlen is non-zero, then host points to a buffer able to contain up to hostlen characters that receives the node name as a null-terminated string. If host is NULL or hostlen is zero, the node name shall not be returned. If the node's name cannot be located, the numeric form of the address contained in the socket address structure pointed to by sa is returned instead of its name.

If serv is non-NULL and servlen is non-zero, then serv points to a buffer able to contain up to servlen bytes that receives the service name as a null-terminated string. If serv is NULL or servlen is zero, the service name shall not be returned. If the service's name cannot be located, the numeric form of the service address (for example, its port number) shall be returned instead of its name.

flags is a flag that changes the default actions of the API call. By default the fully-qualified

domain name (FQDN) for the host shall be returned, but:

NI_NOFQDN
If the flag bit NI_NOFQDN is set, only the node name portion of the FQDN shall be returned for local hosts.

NI_NUMERICHOST
If the flag bit NI_NUMERICHOST is set, the numeric form of the address contained in the socket address structure pointed to by sa shall be returned instead of its name.

NI_NAMEREQD
If the flag bit NI_NAMEREQD is set, an error shall be returned if the host's name cannot be located.

NI_NUMERICSERV
If the flag bit NI_NUMERICSERV is set, the numeric form of the service address shall be returned (for example, its port number) instead of its name.

NI_DGRAM
If the flag bit NI_DGRAM is set, this indicates that the service is a datagram service (SOCK_DGRAM). The default behavior shall assume that the service is a stream service (SOCK_STREAM).

The NI_DGRAM flag is required for the few AF_INET port numbers (for example, [512,514]) that represent different services for UDP and TCP.

Timeout interval for completing the name resolution operation is specified in timeout, tmout, or tmout_u, which is then used in so_getnameinfo(), so_getnameinfo_ms(), or so_getnameinfo_us() respectively.
Specify the relative time until the timeout occurs in timeout using the "timeout" structure, tmout using milliseconds, and tmout_u using microseconds.

If timeout, tmout, or tmout_u is positive, and the name resolution does not complete within the specified time, the name resolution process is aborted and EX_TIMEDOUT is returned.
To try to resolve the same name again in this case, start from scratch instead of continuing this name resolution process.
If tmout or tmout_u is set to TMO_FEVR and timeout is set to NULL, the behavior is the same as that of getnameinfo() in the POSIX specification except for memory space allocation for the result.

If so_break() is issued for the task waiting for the completion of this API call before so_getnameinfo() operation completes, this API call aborts the name resolution processing and returns EX_INTR.
To try to resolve the same name again in this case, start from scratch instead of continuing this name resolution process.

Because getservbyname and getservbyport defined in the POSIX specification are non-thread-safe and they can be replaced by this function, T2EX does not provide API calls equivalent to these.

### 5.5.21 so_resctl - Operation Related to Name Resolution

C Language Interface

```
#include <t2ex/socket.h>

int len = so_resctl(int cmd, void* buf, size_t bufsz);
```

Parameter

| | | |
|---|---|---|
| int | cmd | Command |
| void* | buf | Buffer for storing data |
| int | bufsz | Buffer size |

Return Parameter

| | | |
|---|---|---|
| int | len | Buffer size (in bytes) to store data or the error code |
| void* | buf | Result of the operation |

Error Code

| | |
|---|---|
| EX_INVAL | Illegal parameter |

Description

Performs the operation related to the name resolution specified in cmd.

Any of the following commands is specified in cmd.

| | |
|---|---|
| SO_RES_ADD_TABLE | Adds entries to the host table |
| SO_RES_DEL_TABLE | Deletes entries from the host table |
| SO_RES_GET_TABLES | Gets the host name table |
| SO_RES_FLUSH_TABLES | Deletes all entries from the host table |
| SO_RES_ADD_SERVER | Adds the name resolution server |
| SO_RES_DEL_SERVER | Deletes the name resolution server |
| SO_RES_GET_SERVERS | Gets the name server list |
| SO_RES_FLUSH_SERVERS | Clears the name server list |
| SO_RES_ADD_DOMAIN | Adds the search domain |
| SO_RES_DEL_DOMAIN | Deletes the search domain |
| SO_RES_GET_DOMAINS | Gets the search domain list |
| SO_RES_FLUSH_DOMAINS | Clears the search domain list |

The buffer to store data is specified in buf and its size is specified in bufsz.
buf is used both to pass data from an application to this function and return data from this function to the application.
When data is passed from the application to this function, bufsz bytes of data from the start address of buf is stored.
When data is returned from this function to the application, the len bytes (where len is a return value) of data from the start address of buf is used.
If NULL is specified in buf, the buffer size required to store data can be obtained as the return code.

This API call does not provide name resolution itself.
Name resolution is provided by so_getaddrinfo(), and the name resolution is performed using the settings of this API call.

SO_RES_ADD_TABLE
Type of buf: const struct hosttable*

The content to add to the host name table is specified in buf.
The size of buf is specified in bufsz.
The pointer to the "sockaddr" structure is stored in addr of the "hosttable" structure, and the null-terminated host name is specified in "host" member.
The pointer to the null-terminated host aliases delimited by space (" ") is specified in "aliases" member.
Member "aliases" can also be set to NULL.

SO_RES_DEL_TABLE
Type of buf: const struct hosttable*

The content to delete from the host name table is specified in buf.
The size of buf is specified in bufsz.

SO_RES_GET_TABLES
Type of buf: struct hosttable*

The buffer to store the host name table is specified in buf.
The size of buf is specified in bufsz.
buf is used to store a host name table in the form of an array of "hosttable" structure. The required area for the members of addr, host, and aliases is available from buf.
The addr of the last element is set to NULL.
An address pointed to by buf should be a memory area that is aligned properly to store the "hosttable" structure.
If the memory size required to store the result is larger than "bufsz", maximum possible entries of the host name tables are stored in buf.
Because the addr member of the last element of the array is set to NULL, bufsz needs to have a size enough to store at least one pointer.
If bufsz is too small to store at least one pointer, error EX_INVAL is returned.

```
/* dummy code */
union {
        struct hosttable top;
        UB c[256];
} buf;
struct hosttable *res;
int len, i;
```

```
        len = so_resctl(SO_RES_GET_TABLES, &buf.top, sizeof(buf));
        res = &buf.top;
        for( i=0; res[i].addr != NULL; i++ ) {
                /*
                 * operation to res
                 *   struct sockaddr *addr = res[i].addr;
                 *   char *host            = res[i].host;
                 *   char *alias           = res[i].aliases;
                 */
        }
```

SO_RES_FLUSH_TABLES
        Type of buf: void*

        buf is set to NULL and bufsz is set to 0.
        This function deletes all entries in the host name table.

SO_RES_ADD_SERVER
        Type of buf: const struct sockaddr*

        buf is set to the address of the name resolution server to add.
        The size of buf is specified in bufsz.
        Name resolution servers can be set up to MAXNS as maximum.
        When multiple name resolution servers are specified, queries are sent to the name resolution servers in the order of their addition.

        Name resolution algorithm
         - If a query to the first name resolution server times out, the query is sent to the next name resolution server.
This operation is performed for all name resolution servers.
         - Above described queries for all name resolution servers are repeated until the maximum number of trials is reached.

        #define MAXNS   (3)     /* The maximum number of name resolution servers that can be registered */

SO_RES_DEL_SERVER
        Type of buf: const struct sockaddr*

        buf is set to the address of the name resolution server to delete.
        The size of buf is specified in bufsz.

SO_RES_GET_SERVERS
        Type of buf: struct sockaddr**

        The buffer to store the name resolution server list is specified in buf.
        The size of buf is specified in bufsz.
        buf is used to store the list of name resolution servers in the form of an array of pointers to "sockaddr" structures. The required area to store each "sockaddr" structure is taken from buf.
Size of each "sockaddr" structure is indicated by sa_len, which is a member of "sockaddr" structure.
The last element of the array of pointers is set to NULL.
        An address indicated by buf should be a memory area that is aligned properly to store the pointer to the "sockaddr" structure.
        If the memory size required to store the result is larger than "bufsz", names of name resolution servers as many as possible are stored in buf.
Because the last element of the array is set to NULL, bufsz needs to have a size large enough to store at least one pointer.
If bufsz is too small to store at least one pointer, error EX_INVAL is returned.

```
        /* dummy code */
        union {
                struct sockaddr *top;
                UB c[256];
        } buf;
        struct sockaddr **res;
        int len, i;

        len = so_resctl(SO_RES_GET_SERVERS, &buf.top, sizeof(buf));
        res = &buf.top;
        for( i=0; res[i] != NULL; i++ ) {
                /*
                 * operation to res
```

```
                *   struct sockaddr *addr = res[i];
                */
        }
```

SO_RES_FLUSH_SERVERS
        Type of buf: void*

        buf is set to NULL and bufsz is set to 0.
        Deletes all lists of name resolution server.

SO_RES_ADD_DOMAIN
        Type of buf: const char*

        The search domain to add is set in buf.
        The size of buf is specified in bufsz.
        buf is a pointer to a null-terminated string.
To resolve names, an abbreviated name (without the domain part) can be used for hosts belonging to the search domain.
Such search domains can be registered up to MAXDNSRCH as maximum.
If multiple search domains are registered, each element of the search domain list is tested in the order of registration until the matched name is found in resolving name.
Usually, a local domain name is given first in the list of search domains.
        If the name server for the registered domain is not local, large amount of network traffic may occur due to the query for each server.

        If only one domain is set as a search domain, name searching is performed up to MAXDFLSRCH and to the hierarchy level of LOCALDOMAINPARTS.
For example, when only domain "www.xxx.yyy.zzz" is set, name searching is performed for "www.xxx.yyy.zzz", "xxx.yyy.zzz", and "yyy.zzz".

```
        #define MAXDNSRCH               6       /* Maximum number of domains that can be registered.
*/
        #define MAXDFLSRCH              3       /* Maximum number of subdomains to complete the full
domain name. */
        #define LOCALDOMAINPARTS        2       /* Minimum hierarchy level to regard as a local domain
*/
```

SO_RES_DEL_DOMAIN
        Type of buf: const char*

        The search domain to delete is set in buf.
        The size of buf is specified in bufsz.

SO_RES_GET_DOMAINS
        Type of buf: char**

        The buffer to store the search domain list is specified in buf.
        The size of buf is specified in bufsz.
        buf is used to store domain names in the form of an array of pointers to null-terminated strings. Required area to store each sockaddr structure is taken from buf.
The last element of the array of pointers is set to NULL.
        An address indicated by buf should be a memory area that is aligned properly to store the pointer to the null-terminated string.
        If the memory size required to store the result is larger than "bufsz", as many domain names as possible are stored in buf.
Because the last element of the array is set to NULL, bufsz needs to have a size large enough to store at least one pointer.
If bufsz is too small to store at least one pointer, error EX_INVAL is returned.

```
        /* dummy code */
        union {
                char *top;
                UB c[256];
        } buf;
        char **res;
        int len, i;

        len = so_resctl(SO_RES_GET_SERVERS, &buf.top, sizeof(buf));
        res = &buf.top;
        for( i=0; res[i] != NULL; i++ ) {
                /*
                 * operation to res
                 *   char *domain = res[i];
```

```
                */
        }
```

SO_RES_FLUSH_DOMAINS
        Type of buf: void*

        buf is set to NULL and bufsz is set to 0.
        Deletes all entries of search domain list.

See Also

so_getnameinfo

### 5.5.22 so_rtlist - Get a List of Routing Table Entries

C Language Interface

```
#include <t2ex/socket.h>

int len = so_rtlist(int af, int cmd, int flags, void* buf, size_t bufsz);
```

Parameter

| | | |
|---|---|---|
| int | af | Address family |
| int | cmd | Command |
| int | flags | Flag |
| void* | buf | Buffer to store a routing table |
| size_t | bufsz | Buffer size (in bytes) |

Return Parameter

| | | |
|---|---|---|
| int | len | Buffer size (in bytes) to store the routing table or the error code |
| void* | buf | Routing table |

Error Code

| | |
|---|---|
| EX_AFNOSUPPORT | Specified address family is not implemented |
| EX_INVAL | Illegal parameter |

Description

Gets a routing table entries that meets criteria specified by the arguments.

The address family is specified in af to get a route corresponding to the address family specified by this argument.
If af is set to AF_UNSPEC, gets a route corresponding any address family.

One of the following values is specified in cmd.

| | |
|---|---|
| NET_RT_DUMP | Gets all routes corresponding to the specified address family. |
| NET_RT_FLAGS | Gets all routes having a flag for the specified address family. |
| NET_RT_IFLIST | Gets all routes corresponding to the address family specified for each network interface. |

Logical ORs of flags as follows is set to flags.
Routes that has any bits specified in flags are obtained.
flags is valid only when cmd is set to NET_RT_FLAGS. Otherwise, it is invalid.

| | |
|---|---|
| RTF_UP | Route is available. |
| RTF_GATEWAY | The destination is a gateway. |
| RTF_HOST | The destination is a host. |
| RTF_REJECT | The destination is unreachable. |
| RTF_DYNAMIC | The route to the destination was generated by the ICMP redirect. |
| RTF_MODIFIED | The route to the destination was changed by the ICMP redirect. |
| RTF_CLONING | A new route is generated by duplication. |
| RTF_LLINFO | Information about valid data link layer exists. |
| RTF_STATIC | The route has been added manually. |
| RTF_BLACKHOLE | Destroys packets to the destination. |
| RTF_CLONED | The route is generated by duplication. |
| RTF_PROTO2 | A protocol specific flag |
| RTF_PROTO1 | A protocol specific flag |

buf is set to a pointer to the buffer to store the routing information, and bufsz is set to the size of the buffer (in bytes) to which buf points.
If buf is set to NULL, the required buffer size to store the routing table acquired through the specified arguments is returned.

buf stores routing messages as many as possible.
rtm_msglen of the header of the last routing message is set to 0.
So, bufsz needs to have the size large enough to store one rtm_msglen.
If bufsz is too small to store at least one rtm_msglen, error EX_INVAL is returned.

The retrieved route can be referred by using following codes.

```
/* dummy code */
size_t                  needed;
struct rt_msghdr        *rtm;
char                    *buf;

needed = so_rtlist(AF_UNSPEC, NET_RT_DUMP, 0, NULL, 0);
buf = malloc(needed);
needed = so_rtlist(AF_UNSPEC, NET_RT_DUMP, 0, buf, needed);
for( rtm = (struct rt_msghdr *)buf;
     rtm->rtm_msglen != 0;
     rtm = (struct rt_msghdr *)((void*)rtm + rtm->rtm_msglen) ) {
        /*
         * operation for rtm
         *  (casts rtm to the appropriate header based on rtm->rtm_type.
         *   See 5.6.1 routing message)
         */
}
free(buf);
```

### 5.5.23 so_ifattach - Attach a Device Driver

C Language Interface

```
#include <t2ex/socket.h>

ER ercd = so_ifattach(const char* devnm);
```

Parameter

| | | |
|---|---|---|
| const char* | devnm | Device name |

Return Parameter

| | | |
|---|---|---|
| ER | ercd | Error code |

Error Code

| | |
|---|---|
| E_OK | Normal completion |
| EX_BUSY | Device Driver is busy |
| EX_INVAL | Illegal parameter |

Description

Attaches the device driver specified in devnm that complies with the T-Engine Standard Device Driver Specification to the network communication manager.
Set a device name specified in the T-Engine Standard Device Driver Specification (e.g. Neta, Netb) in devnm.

After this API call completes successfully, addressing, activation and other operations are possible for devnm.

See Also

so_ifdetach, so_ioctl, so_socket

### 5.5.24 so_ifdetach - Detach a Device Driver

C Language Interface

```
#include <t2ex/socket.h>

ER ercd = so_ifdetach(const char* devnm);
```

Parameter

| | | |
|---|---|---|
| const char* | devnm | Device name |

Return Parameter

| | | |
|---|---|---|
| ER | ercd | Error code |

Error Code

| | |
|---|---|
| E_OK | Normal completion |
| EX_NOENT | Device Driver is not connected |
| EX_BUSY | Device Driver is busy |
| EX_INVAL | Illegal parameter |

Description

Detaches the device driver specified in devnm that complies with the T-Engine Standard Device Driver Specification from the network communication manager.
Set a device name specified in the T-Engine Standard Device Driver Specification (e.g. Neta, Netb) in devnm.

See Also

so_ifattach

### 5.5.25 so_getifaddrs - Get Interface Address Information

C Language Interface

```
#include <t2ex/socket.h>

int len = so_getifaddrs(struct ifaddrs** ifap, void* buf, size_t bufsz);
```

Parameter

| | | |
|---|---|---|
| struct ifaddrs** | ifap | The pointer to the head of the linked list of the address information of the interface |
| void* | buf | Buffer to store the address information of the interface |
| size_t | bufsz | The size of the buffer to store address information of interface (in bytes) |

Return Parameter

| | | |
|---|---|---|
| int | len | Buffer size (in bytes) to store interface address information or the error code |
| struct ifaddrs** | ifap | The pointer to the head of the linked list of the address information of the interface |
| void* | buf | Linked list of the address information of the interface |

Error Code

| | |
|---|---|
| EX_INVAL | Illegal parameter |

Description

Gets the address information of registered network interfaces in the form of a linked list.

ifap is a pointer reference to the "ifaddrs" structure.

```
    struct ifaddrs   *ifa_next;         /* Pointer to next struct */
    char             *ifa_name;         /* Interface name */
    unsigned int     ifa_flags;         /* Interface flags */
    struct sockaddr  *ifa_addr;         /* Interface address */
```

```
    struct sockaddr  *ifa_netmask;      /* Interface netmask */
    struct sockaddr  *ifa_broadaddr;    /* Interface broadcast address */
    struct sockaddr  *ifa_dstaddr;      /* P2P interface destination */
    void             *ifa_data;         /* Address specific data */
```

The ifa_next field is a pointer to the next element of linked list.
When it is the last element of the linked list, ifa_next field is NULL.

The name of the interface is stored into ifa_name field.

Logical OR of the following flags is stored into ifa_flags field.

R- or RW in the following description means the ability to refer something only or to both refer and change something respectively.

| | | |
|---|---|---|
| IFF_UP | RW | The interface is running. |
| IFF_BROADCAST | R- | The broadcast address is enabled. |
| IFF_DEBUG | RW | It is in debug mode. |
| IFF_LOOPBACK | RW | It is a loop-back. |
| IFF_POINTOPOINT | R- | It is a point-to-point link. |
| IFF_NOTRAILERS | RW | Does not use a trailer. |
| IFF_RUNNING | R- | Resourses have already been assigned. |
| IFF_NOARP | RW | The address resolution for network is disabled. |
| IFF_PROMISC | R- | Receives all packets. |
| IFF_ALLMULTI | R- | Receives all multicast packet. |
| IFF_OACTIVE | R- | Currently sending. |
| IFF_SIMPLEX | R- | Simplex mode communication. |
| IFF_LINK0 | RW | Control flag of link layer |
| IFF_LINK1 | RW | Control flag of link layer |
| IFF_LINK2 | RW | Control flag of link layer |
| IFF_MULTICAST | R- | Supports multicast. |

An address of the interface or an address of the data link layer level is stored in ifa_addr field.
If neither of addresses does not exist, the field is set to NULL.

A net mask associated with ifa_addr is stored in the ifa_network field.
If no net mask exists, the field is set to NULL.

For non P2P interface, a broadcast address associated with ifa_addr is stored in the ifa_broadaddr field.
Otherwise, the field is set to NULL.

For the P2P interface, a destination address is stored in the ifa_dstaddr field.
Otherwise, the field is set to NULL.

The address specific information is stored in ifa_data field.
For AF_LINK, the "if_data" structure which includes the interface information and the statistics is stored in the field.
For other address families, it is set to NULL.

buf is a pointer to the buffer area to store address information of the network interface, and its size is specified in bufsz in bytes.
The return code at normal completion indicates the size of buffer required to store the address information.

If the memory size required to store address information is larger than "bufsz", entries are stored in buf as many as possible.
If no "ifaddrs" structure can be stored in buf, ifap is set to NULL.

See Also

so_ifattach, so_ifdetach, so_ioctl

### 5.5.26 so_ifindextoname - Convert an Interface Index to Interface Name

C Language Interface

```
#include <t2ex/socket.h>

ER ercd = so_ifindextoname(unsigned int ifindex, char* ifname);
```

Parameter

| | | |
|---|---|---|
| unsigned int | ifindex | Interface index |
| char* | ifname | Buffer to store the name of the interface |

Return Parameter

| | | |
|---|---|---|
| ER | ercd | Error code |
| char* | ifname | Interface name |

Error Code

| | |
|---|---|
| E_OK | Normal completion |
| EX_NXIO | No interface exist |

Description

Convert an interface index to interface name.

The interface index is specified in ifindex.
An interface index is an integer that is retrieved through so_getifaddrs() and routing messages.

At least IF_NAMESIZE bytes of buffer should be specified for ifname.
If this API call completes normally, the null-terminated interface name is stored in ifname.

```
#define IF_NAMESIZE     16      /* the maximum size of the interface name (in bytes, including NULL
character) */
```

See Also

so_ifnametoindex

### 5.5.27 so_ifnametoindex - Convert an Interface Name to Interface Index

C Language Interface

```
#include <t2ex/socket.h>

int ifindex = so_ifnametoindex(const char* ifname);
```

Parameter

| | | |
|---|---|---|
| const char* | ifname | Interface name |

Return Parameter

| | | |
|---|---|---|
| int | ifindex | The interface index or the error code |

Error Code

| | |
|---|---|
| EX_NXIO | No interface exist |

Description

Convert an interface name to interface index.

The null-terminated interface name is specified in ifname.

See Also

so_ifindextoname

### 5.5.28 so_ioctl - Control a Device

C Language Interface

```
#include <t2ex/socket.h>

ER ercd = so_ioctl(int sd, int request, ... /* arg */);
```

Parameter

| | | |
|---|---|---|
| int | sd | Socket descriptor |
| int | request | Request |

Return Parameter

| | | |
|---|---|---|
| ER | ercd | Error code |

Error Code

| | |
|---|---|
| E_OK | Normal completion |
| EX_BADF | sd is not a valid socket descriptor. |
| EX_INVAL | The request or arg argument is not valid for this device. |
| EX_FAULT | An argument of the request is not in the valid address space |

Description

so_ioctl() shall perform a variety of control functions on socket devices.

sd is an open socket descriptor that refers to a device.

request selects the control function to be performed and shall depend on the device being addressed.

arg represents additional information that is needed by this specific device to perform the requested function. The type of arg depends upon the particular control request, but it shall be either an integer or a pointer to a device-specific data structure.

Values that can be specified for request and the description about their arg are as follows.

FIONBIO const int*
Sets the blocking or non-blocking mode of I/O operation for the descriptor by the value pointed to by the argument int*.
*arg == 0 sets the blocking mode (the O_NONBLOCK status flag is cleared).
*arg != 0 sets the non-blocking mode (the O_NONBLOCK status flag is set).

FIONREAD int*
SIOCINQ int*
Returns the number of bytes ready to be immediately read, in the area pointed to by the argument int*.

FIONWRITE int*
SIOCOUTQ int*
Returns the number of bytes of the data stored in the send queue for the descriptor, in the area pointed to by the argument int*.
Those bytes are data already written to the descriptor, waiting to be processed.
How they are processed is device-dependent.

FIONSPACE int*
Returns the available free space of the send queue for the descriptor, in the area pointed to by the argument int*.
This value is the size of the send queue minus the size of data stored in the queue.

SIOCATMARK int*
Check whether out-of-band data has been received or not, and return the result in the area indicated by the argument int*.
If this value is 1, the socket is marked as having out-of-band data.
In this case, out-of-band data can be read by specifying the MSG_OOB flag on so_recv().
If this value is 0, the socket is not marked as having out-of-band data.

SIOCSIFADDR const struct ifreq*
Sets the specified address to the network interface.

SIOCAIFADDR const struct ifaliasreq*
Sets or adds the specified address to the network interface.
If the specified address has already been set up, updates the information about the address.
This command can set a host address, a destination address, a broadcast address, and a net mask at the same time.

SIOCDIFADDR const struct ifaliasreq*

```
                Deletes the specified address from the network interface.

SIOCGIFADDR     struct ifreq*
                Obtains the address of the specified network interface.

SIOCSIFNETMASK  const struct ifreq*
                Sets the net mask to the specified network interface.

SIOCGIFNETMASK  struct ifreq*
                Obtains the net mask of the specified network interface.

SIOCSIFFLAGS    const struct ifreq*
                Sets the flag to the specified network interface.

SIOCGIFFLAGS    struct ifreq*
                Obtains the flag of the specified network interface.

SIOCSIFBRDADDR  const struct ifreq*
                Sets the broadcast address to the specified network interface.
                If IFF_BROADCAST flag is not set for the network interface, returns error EX_INVAL.

SIOCGIFBRDADDR  struct ifreq*
                Obtains the broadcast address of the specified network interface.
                If IFF_BROADCAST flag is not set for the network interface, returns error EX_INVAL.

SIOCSIFDSTADDR  const struct ifreq*
                Sets the destination address of the specified network interface.
                If IFF_POINTOPOINT flag is not set for the network interface, returns error EX_INVAL.

SIOCGIFDSTADDR  struct ifreq*
                Obtains the destination address of the specified network interface.
                If IFF_POINTOPOINT flag is not set for the network interface, returns error EX_INVAL.

See Also

so_read, so_recv, so_sockatmark, so_write
```

### 5.5.29 so_fcntl - Socket Control

```
C Language Interface

#include <t2ex/socket.h>

ER ercd = so_fcntl(int sd, int cmd, ... /* arg */);

Parameter

int                     sd              Socket descriptor
int                     cmd             Command
                        arg             Required arguments (variable number) depending on the command

Return Parameter

ER                      ercd            Zero or positive result depending on the command
                                        or error code

Error Code

E_OK                    Normal completion
EX_BADF                 sd is not a valid socket descriptor.
EX_INVAL                cmd is invalid.

Description

so_fcntl() shall perform the operations described below on open socket.

F_GETFL
        Get the socket status flags for the socket descriptor associated with sd. so_fcntl() returns
        the socket status flag.

F_SETFL
```

Set the socket status flags for the socket descriptor associated with sd from the corresponding bits in the third argument, arg, taken as type int.

Flag available for F_GETFL and F_SETFL commands is O_NONBLOCK.

O_NONBLOCK
Non-blocking mode.
If this flag is set, the socket descriptor is in non-blocking mode.

See Also

so_accept, so_close, so_connect

### 5.5.30 so_sockatmark - Determine Whether a Socket Is at the Out-of-band Mark

C Language Interface

```
#include <t2ex/socket.h>

int mark = so_sockatmark(int sd);
```

Parameter

| | | |
|---|---|---|
| int | sd | Socket descriptor |

Return Parameter

| | | |
|---|---|---|
| int | mark | Value indicating whether the socket is at an out-of-band data mark<br>or Error code |

Error Code

| | |
|---|---|
| EX_BADF | sd is not a valid socket descriptor. |
| EX_INVAL | Invalid parameters. |

Description

so_sockatmark() shall determine whether the socket specified by the descriptor sd is at the out-of-band data mark.

If the protocol for the socket supports out-of-band data by marking the stream with an out-of-band data mark, so_sockatmark() shall return 1 when all data preceding the mark has been read and the out-of-band data mark is the first element in the receive queue. so_sockatmark() shall not remove the mark from the stream.

See Also

so_ioctl, so_recvmsg

### 5.5.31 so_shutdown - Shut Down Socket Send and Receive Operations

C Language Interface

```
#include <t2ex/socket.h>

ER ercd = so_shutdown(int sd, int how);
```

Parameter

| | | |
|---|---|---|
| int | sd | Socket descriptor |
| int | how | Type of shutdown |

Return Parameter

| | | |
|---|---|---|
| ER | ercd | Error code. |

Error Code

| | |
|---|---|
| E_OK | Normal completion |
| EX_BADF | sd is not a valid socket descriptor. |
| EX_INVAL | how is invalid. |
| EX_NOTCONN | The socket is not connected. |

Description

so_shutdown() shall cause all or part of a full-duplex connection on the socket associated with the socket descriptor sd to be shut down.

sd specifies the socket descriptor of the socket.

how specifies the type of shutdown. The values are as follows:

SHUT_RD
Disables further receive operations.

SHUT_WR
Disables further send operations.

SHUT_RDWR
Disables further send and receive operations.

so_shutdown() disables subsequent send and/or receive operations on a socket, depending on the value of how.

### 5.5.32 so_break - Stop Socket Operation

C Language Interface

```
#include <t2ex/socket.h>

int ntsk = so_break(ID tskid);
```

Parameter

| | | |
|---|---|---|
| ID | tskid | The ID of the task which is to be released from WAIT state |

Return Parameter

| | | |
|---|---|---|
| int | ntsk | Number of tasks which were released from the WAIT state<br>or error code |

Error Code

| | |
|---|---|
| E_ID | Task ID is invalid (negative or exceeding TMaxTskId) |
| E_NOEXS | Task with the task ID does not exist |

Description

This function forcibly releases the task specified by "tskid" from a waiting state caused by an API call of the network communication manager.
If tskid is TSK_ALL(= -1), releases the waiting state of the network communication functions for all tasks.

Releases the waiting state by the socket operation immediately for the specified task. If the interrupted API call is in the wait state, it returns EX_INTR. Otherwise it returns the processing result of up to the point when so_break() is issued.

If the task is released from the wait status by so_read(), so_recv(), so_recvmsg(), or so_recvfrom() and the API call has not read any data, it returns the error EX_INTR.
If any of them has read at least one byte of data, it returns the number of data read.

If the task is released from the wait status by so_write(), so_send(), so_sendmsg(), or so_sendto() and the API call has not written any data, it returns the error EX_INTR.
If any of them has written at least one byte of data, it returns the number of data written.

If the task is released from the wait state by so_accept(), so_connect(), so_select(), so_select_ms(), or so_select_us(), the API call returns the error EX_INTR.
The processing of these API calls which has been aborted by so_break can be later probed for completion by using so_select(), so_select_ms(), and so_select_us().

If the task in wait state by so_getaddrinfo(), so_getaddrinfo_ms(), so_getaddrinfo_us(), so_getnameinfo(), so_getnameinfo_ms(), or so_getnameinfo_us() is released, the API call returns the error EX_INTR.
Operations of these API calls aborted by so_break() cannot be resumed.

If the waiting state of the task waiting for so_close() is released, this API call completes successfully.
The specified socket is closed with a delay.

At the normal completion, so_break() returns the number of tasks which are released from wait.
If the target task does not call the network communication function, this API call exits without any operations and returns the return code E_OK.

A wait release request by so_break() is not queued.

See Also

so_accept, so_close, so_connect, so_getaddrinfo, so_getaddrinfo_ms, so_getaddrinfo_us, so_getnameinfo, so_getnameinfo_ms, so_getnameinfo_us, so_read, so_recv, so_recvfrom, so_recvmsg, so_select, so_send, so_sendmsg, so_sendto, so_write

5.6 Operation for Routing Socket

An application manipulates a routing table by using routing messages to routing sockets.
It can also obtain event notifications related to the routing table from the network communication manager by receiving routing messages using routing sockets.

A routing socket is created by passing the following arguments to the API call so_socket().

```
  s = socket( PF_ROUTE, SOCK_RAW, protocol );
```

If "protocol" is set to AF_UNSPEC, all routing messages can be sent and received.
If a specific address family is specified in "protocol", routing messages related to the specific address family can be sent and received.
For example, if "protocol" is specified as AF_INET, only messages related to IP are received.

Messages sent to the network communication manager through a routing socket are sent to all routing sockets.
Therefore, when a message is sent through a routing socket, copy of the sent message is inserted into the receive queue of the socket and the socket receives the message sent by itself.
Insertion of the message, sent by itself, into the receive queue can be avoided by using so_setsockopt() to disable SO_USELOOPBACK.
Also, so_shutdown() can be used to avoid subsequent routing messages being inserted into the receive queue of the socket.

5.6.1 Routing Message

A routing message is used for operations of the routing table such as adding, deleting, and retrieving routes, and for notification of events from the network communication manager related to the routing table such as adding and deleting a route and failing to find a route.

A routing message consists of a header and a series of addresses, which follows it immediately, of "sockaddr" structures.
There are following message types of routing messages.

| | | |
|---|---|---|
| RTM_ADD | RW | Adds a route |
| RTM_DELETE | RW | Deletes a route |
| RTM_CHANGE | RW | Changes a route |
| RTM_GET | RW | Retreives a route |
| RTM_LOSING | R- | Failed to reach to the destination |
| RTM_REDIRECT | R- | Notify the ICMP redirect |
| RTM_MISS | R- | Failed to find a route |
| RTM_LOCK | RW | Locks the specified routing metric |
| RTM_NEWADDR | R- | Adds an address to the interface |
| RTM_DELADDR | R- | Deletes an address from the interface |
| RTM_IFINFO | R- | Changes the status of the interface/link |
| RTM_IFANNOUNCE | R- | Notifies of attached or detached interface |

RW: Allows setting/reference
R-: Allows reference only

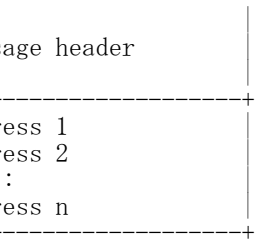



Figure: Structure of a routing message

5.6.1.1 Message Header Part

Structure of the header of a routing message depends on the message type.

| | |
|---|---|
| RTM_ADD | "rt_msghdr" structure |
| RTM_DELETE | "rt_msghdr" structure |
| RTM_CHANGE | "rt_msghdr" structure |
| RTM_GET | "rt_msghdr" structure |
| RTM_LOSING | "rt_msghdr" structure |
| RTM_REDIRECT | "rt_msghdr" structure |
| RTM_MISS | "rt_msghdr" structure |
| RTM_LOCK | "rt_msghdr" structure |
| RTM_NEWADDR | "ifa_msghdr" structure |
| RTM_DELADDR | "ifa_msgdhr" structure |
| RTM_IFINFO | "if_msghdr" structure |
| RTM_IFANNOUNCE | "if_announcemsghdr" structure |

5.6.1.2 Address Part

An address specified in the header is stored immediate after the header part in the routing message.
The member name of the structure indicating an address in a message is different for each header structure: rtm_addrs, ifm_addrs, and ifam_addrs.
These store the value of logical OR of the following flags
"if_announcemsghdr" structure does not have this member.

| | |
|---|---|
| RTA_DST | destination address |
| RTA_GATEWAY | gateway address |
| RTA_NETMASK | net mask |
| RTA_GENMASK | mask used for duplication |
| RTA_IFP | data link layer address of the network interface |
| RTA_IFA | address of network interface |
| RTA_AUTHOR | node address of the node that sent an ICMP redirect |
| RTA_BRD | broadcast address |

Addresses are stored in the address part in ascending order of the above flags.
For example, in case of a logical OR of RTA_DST, RTA_GATEWAY, and RTA_NETMASK, address are stored after the header in the order of the destination address, the gateway address, and the net mask.

An address other than the one for RTA_IFP is stored in the form of the "sockaddr" structure based on the address family.
An address for RTA_IFP is stored in a "sockaddr_dl" structure for AF_LINK.
The size of each address is specified by sa_len of the "sockaddr" structure.

5.6.2 Detail of a Routing Message

Each member of the "rt_msghdr", "ifa_msgdhr", "if_msghdr", and "if_announcemsghdr" structures has the following meaning.

rtm_msglen, ifm_msglen, ifam_msglen, and ifan_msglen are the whole size of the message including the message header and the address that follows.

rtm_version, ifm_version, ifam_version, and ifan_version are used to check binary compatibility.
Their value is RTM_VERSION.

```
#define RTM_VERSION     3       /* Version */
```

rtm_type, ifm_type, ifam_type, and ifan_type are message types.
rtm_index, ifm_index, ifam_index, and ifan_index are indices of interface structures.

rtm_flags, ifm_flags, and ifam_flags are flags of routing messages.
These values are logical ORs of flags.
Flags can be set only by the RTM_ADD messages.

```
        RTF_UP                  R-      Route is available.
        RTF_GATEWAY             RW      The destination is a gateway.
        RTF_HOST                RW      The destination is a host.
        RTF_REJECT              RW      The destination is unreachable.
        RTF_DYNAMIC             R-      The route was generated by the ICMP redirect.
        RTF_MODIFIED            R-      The route was changed by the ICMP redirect.
        RTF_DONE                R-      Notification of completion from the network communication
manager
        RTF_CLONING             RW      Creates a new route by duplication (cloning).
        RTF_LLINFO              R-      Information about valid data link layer exists.
        RTF_STATIC              RW      The route has been added manually.
        RTF_BLACKHOLE           RW      Destroys packets to the destination.
        RTF_CLONED              RW      A route was generated by duplication.
        RTF_PROTO2              RW      A protocol specific flag
        RTF_PROTO1              RW      A protocol specific flag

RW: Allows setting/reference
R-: Allows reference only

RTF_UP
        RTF_UP indicates that the route is available.
This is a flag set by the network communication manager.
        Setting this flag on writing is ignored.

RTF_GATEWAY
        RTF_GATEWAY indicates that the destination is a gateway.
If RTF_GATEWAY is not set, messages reach the destination directly.
It is a flag that can be set by an application.

RTF_HOST
        RTF_HOST indicates that the destination is an address of a host.
IF RTF_HOST is not set, the destination is an address of a network.

It is a flag that can be set by an application.

RTF_REJECT
        RTF_REJECT indicates that the destination via the route is unreachable.
        Though this flag is settable from an application, it is assumed to be used by ARP mainly.
        When the destination does not respond to an ARP request, this flag is set to discard packets
to the destination.
        If the destination to send matches the route for which RTF_REJECT is set, sending APIs such as
so_send() return EX_HOSTUNREACH.

RTF_DYNAMIC
        RTF_DYNAMIC indicates that the route is added by the ICMP redirect.
        This is a flag set by the network communication manager.
Setting this flag on writing is ignored.

RTF_MODIFIED
        RTF_MODIFIED indicates that the gateway of existing route is changed by the ICMP redirect.
        This is a flag set by the network communication manager.
Setting this flag on writing is ignored.

RTF_DONE
        RTF_DONE indicates that the processing of a routing message from an application has completed.
        This is a flag set by the network communication manager.
Setting this flag on writing is ignored.
        This flag is used only in routing messages and not stored as routing information.

RTF_CLONING
        RTF_CLONING indicates that a new route to the destination has been created by cloning.
        For example, when RTF_CLONING is set on the route whose destination is an address of network,
the route to the host address that matches with this route is cloned and added to the routing table.
        RTF_CLONED is set on the cloned route.
When deleting a route for which RTF_CLONING is set, those routes that have been created by cloning
that route and RTF_CLONED is set are deleted together.

RTF_LLINFO
        RTF_LLINFO indicates that the route has information about valid data link layer level.
        This is a flag set by the network communication manager.
Specially, ARP in the network communication manager sets this.
```

RTF_STATIC
RTF_STATIC indicates that the route is added manually.
This flag can be set by an application, and set when the application adds a route.

RTF_BLACKHOLE
RTF_BLACKHOLE discards packets to the destination.
Unlike RTF_REJECT, it only discards packets and does not return any error.
It is a flag that can be set by an application.

RTF_CLONED
RTF_CLONED indicates that the route is created by cloning based on the setting of RTF_CLONING.
It is a flag that can be set by an application.
Bulk deletion of cloned routes along with the deletion of the original route, for which RTF_CLONING is set, can be prevented by clearing RTF_CLONED setting.

RTF_PROTO2, RTF_PROTO1
RTF_PROTO1 and RTF_PROTO2 are flags that any protocol can define the usage and use accordingly.

rtm_addrs, ifm_addrs, and ifam_addrs are addresses included in a routing message.
These contain the value of logical ORs of flags constants whose name starts with "RTA_".

rtm_tid and rtm_seq are used to identify a message. They shall be set by a task sending routing messages appropriately.

rtm_errno indicates that an error has occurred during processing a routing message.

| | |
|---|---|
| EEXIST | Attempted to add an existing route |
| ESRCH | Attempted to delete a route that does not exist |
| ENOBUFS | Not enough resource to add a new route |

There are multiple routing metrics represented by the "rt_metrics" structure.
rtm_inits specifies routing metrics to be initialized, which contains the value of logical OR of flag constants whose name starts with "RTV_".
The initial values for routing metrics of the routing information are specified by values of rtm_rmx corresponding to bits of rtm_inits.

rtm_rmx is routing metrics of the routing information.
For rmx_locks of the "rt_metric" structure, specify the routing metric which the network communication manager is now allowed to change, and set a value of logical OR of the following flags.

| | |
|---|---|
| RTV_MTU | Flags corresponding to rmx_mtu |
| RTV_HOPCOUNT | Flags corresponding to rmx_hopcount |
| RTV_EXPIRE | Flags corresponding to rmx_expire |
| RTV_RPIPE | Flags corresponding to rmx_recvpipe |
| RTV_SPIPE | Flags corresponding to rmx_sendpipe |
| RTV_SSTHRESH | Flags corresponding to rmx_ssthresh |
| RTV_RTT | Flags corresponding to rmx_rtt |
| RTV_RTTVAR | Flags corresponding to rmx_rttvar |

ifan_name is a device name and stores a device name like Neta.

ifan_what is a type of notification that relates to interfaces.
Its value is either IFAN_ARRIVAL or IFAN_DEPARTURE.

| | |
|---|---|
| IFAN_ARRIVAL | Registration an interface |
| IFAN_DEPARTURE | Disconnection of an interface |

5.6.2.1 RTM_ADD - Add a Route

The RTM_ADD message adds a route.
This message is sent from an application to the network communication manager.

rtm_msglen, rtm_version, and rtm_type are set to the total size of the routing message, RTM_VERSION, RTM_ADD respectively.

For rtm_addrs, at least RTA_DST and RTA_GATEWAY should be set to specify the route to add.
If a network address is specified as the address, RTA_NETMASK needs to be specified also.
Additionally, the network interface can be specified explicitly by setting RTA_IFP or RTA_IFA.
If a network interface is not specified explicitly, the system selects an appropriate network interface.

At least RTF_STATIC needs to be specified for rtm_flags.

The task ID and the sequence number shall be set appropriately in rtm_tid and rtm_seq respectively.

For rtm_inits, specify flags corresponding to the routing metrics to be initialized and set values of routing metrics to rtm_rmx.
Otherwise, 0 is used for initial values for routing metrics which are not specified by rtm_inits.

For the other members of "rt_msghdr" structure, set zero before transmission.

5.6.2.2 RTM_DELETE - Delete a Route

The RTM_DELETE message deletes a route.
At the same time, routes generated by cloning the specified route (RTF_CLONED is set as the flag) are also deleted.
This message is sent from an application to the network communication manager.

rtm_msglen, rtm_version, and rtm_type are set to the total size of the routing messages, RTM_VERSION, RTM_DELETE respectively.

For rtm_addrs, set RTA_DST and specify the route to delete.
If a network address is specified as the address, RTA_NETMASK needs to be specified also.

The task ID and the sequence number shall be set appropriately in rtm_tid and rtm_seq respectively.

For the other members of "rt_msghdr" structure, set zero before transmission.

5.6.2.3 RTM_CHANGE - Change a Route

The RTM_CHANGE messages changes the gateway, the network interface, or the flag of the specified gateway.
This message is sent from an application to the network communication manager.

rtm_msglen, rtm_version, and rtm_type are set to the total size of the routing messages, RTM_VERSION, RTM_CHANGE respectively.

For rtm_addrs, at least RTA_DST should be set to specify the route to change.
If a network address is specified as the address, RTA_NETMASK needs to be specified also.
The gateway and the network interface is set as necessary.

For rtm_inits, specify flags corresponding to the routing metrics to be changed and set values of routing metrics to rtm_rmx.

The task ID and the sequence number shall be set appropriately in rtm_tid and rtm_seq respectively.

For the other members of "rt_msghdr" structure, set zero before transmission.

5.6.2.4 RTM_GET - Retrieve a Route

The RTM_GET message retrieves a routing information.
This message is sent from an application to the network communication manager, which then stores the specified route information in a message and sends it back to the application.

rtm_msglen, rtm_version, and rtm_type are set to the total size of the routing messages, RTM_VERSION, RTM_GET respectively.

For rtm_addrs, at least RTA_DST should be set to specify the route to retreive.
If a network address is specified as the address, RTA_NETMASK needs to be specified also.
To get information about the data link layer, set RTA_IFP and add the address of the "sockaddr_dl" structure following the message header.
sdl_len and sdl_family members of the "sockaddr_dl" structure are set to the size of the "sockaddr_dl" structure and AF_LINK respectively while other members are set to zero.

The task ID and the sequence number shall be set appropriately in rtm_tid and rtm_seq respectively.

For the other members of "rt_msghdr" structure, set zero before transmission.

5.6.2.5 RTM_LOSING - Notify a Failure to Reach the Destination

The RTM_LOSING message notifies that the destination is unreachable.
Specifically, this message is generated when a TCP segment cannot reach the destination after trying to resend for the specified times.
This might indicate that a wrong gateway is selected as the route to the destination.
This message is sent to an application from the network communication manager.

The RTM_LOSING message stores route information that failed.

5.6.2.6 RTM_REDIRECT - Notify an ICMP Redirect

The RTM_REDIRECT message notifies that the route was generated or changed by the ICMP redirect message.
This message is sent to an application from the network communication manager.

An RTM_REDIRECT message stores a route added by the ICMP redirect message or a route after the change.
RTA_AUTHOR is set to the node address that sent the ICMP redirect message.

5.6.2.7 RTM_MISS - Notify a Failure to Find a Route

The RTM_MISS message notifies the failure of searching a route to the destination.
If no entry in the routing table matches the destination, a search failure occurs.
This message is sent to an application from the network communication manager.

The RTM_MISS message includes a destination address that failed the route search.

5.6.2.8 RTM_LOCK - Lock the Specified Routing Metric

The RTM_LOCK message specifies that a certain routing metric of the specified route is locked and not to be changed by the the network communication manager or is unlocked and to be changed.
It does not change other items.
This message is sent from an application to the network communication manager.

rtm_msglen, rtm_version, and rtm_type are set to the total size of the routing messages, RTM_VERSION, RTM_LOCK respectively.

Value to change is set in rtm_rmx of rmx_locks.

The task ID and the sequence number shall be set appropriately in rtm_tid and rtm_seq respectively.

For the other members of "rt_msghdr" structure, set zero before transmission.

5.6.2.9 RTM_NEWADDR - Add an Address to the Interface

The RTM_NEWADDR message notifies that an address has been added to the interface.
This message is sent to an application from the network communication manager.

The added route is stored in the message whose header is the "ifa_msghdr" structure.

5.6.2.10 RTM_DELADDR - Delete an Address from the Interface

The RTM_DELETE message notifies that an address has been deleted from the interface.
This message is sent to an application from the network communication manager.

The deleted route is stored in the message whose header is the "ifa_msghdr" structure.

5.6.2.11 RTM_IFINFO - Change the Status of the Interface/Link

The RTM_IFINFO message notifies that the states of the interface or the link has changed.
This message is sent to an application from the network communication manager.

The status of the interface and the link is stored in the message whose header is the "if_msghdr" structure.

5.6.2.12 RTM_IFANNOUNCE - Notify a Attachment or Detachment of an Interface

The RTM_IFANNOUNCE message notifies that the interface has been attached or detached.
This message is sent to an application from the network communication manager.

The notification of attachment or detachment of the interface is stored in the message whose header is the "if_announcemsghdr" structure.

# Chapter 6 Calendar Function

## 6.1 Overview

The calendar function converts between the system time (SYSTIM and SYSTIM_U) and time_t type (calendar time) in T-Kernel 2.0 Extension.
The API name prefix is "dt_" (date/time).

In T-Kernel 2.0 Extension, the system time (SYSTIM and SYSTIM_U) is expressed as total milliseconds (SYSTIM) and total microseconds (SYSTIM_U) from 0:00:00 UTC, January 1, 1985.

## 6.2 Definition

### time_t

The time_t type is an integer data type representing the time in seconds, used in POSIX, and called calendar time.
The time_t type expresses the time as total seconds from 0:00:00 UTC, January 1, 1970.
The start time of this total seconds (0:00:00 UTC, January 1, 1970) is called the "epoch".

For mutual conversion between calendar time and system time, the API calls corresponding to all of the time_t, SYSTIM, and SYSTIM_U types are provided.

### struct tm

The structure struct tm is defined for representing calendar time elements, as shown below.
tm_usec is added as a structure member specific to T-Kernel 2.0 Extension, which allows you to include the information about the time in millisecond or microsecond level.

```
#include <t2ex/datetime.h>

struct tm {
        int     tm_usec;                /* microseconds [0,999999] */
        int     tm_sec;                 /* seconds [0,60] */
        int     tm_min;                 /* minutes [0,59] */
        int     tm_hour;                /* hour [0,23]*/
        int     tm_mday;                /* day of the month [1,31] */
        int     tm_mon;                 /* month [0,11] */
        int     tm_year;                /* years since 1900 */
        int     tm_wday;                /* day of the week [0,6] (0 is Sunday) */
        int     tm_yday;                /* days since January 1 [0,365] */
        int     tm_isdst;               /* daylight saving time (positive: on DST, 0: not DST,
negative: unknown) */
};
```

### Time Zone

Time zone is the time of a whole region that uses a certain time difference from the Coordinated Universal Time (UTC) as the standard time for the region.
The structure struct tzinfo represents the time zone of a region as the difference from the Coordinated Universal Time (UTC).

```
#define TZNAME_MAX 8                            /* maximum number of characters of time zone name */

struct tzinfo {
        char                    tzname[2][TZNAME_MAX+1];        /* time zone name */
        long                    offset;                         /* offset in seconds from UTC
(positive for west of UTC) */
        int                     daylight;                       /* daylight saving time */

        /* The followings are specific to daylight saving time (valid only if daylight > 0) */
        long                    dst_offset;                     /* offset in seconds from UTC during
daylight saving time */
        union dsttimespec       dst_start;                      /* start date/time of daylight saving
time */
        union dsttimespec       dst_end;                        /* end date/time of daylight saving
time */
};
```

daylight  If positive, the daylight saving time is in effect.
If 0, the daylight saving time is not in effect.

```
        dst_start, dst_end
                        Indicates the time period of daylight saving time.
The dsttimespec union is defined as follows:

union dsttimespec {
        uint32_t                v;
        struct julian {
                unsigned int    type: 2,                        /* use total seconds since 0:00,
January 1, local time (value is 0 or 1) */
                int             offset: 30,                     /* total seconds [0,31622400] */
        } j;
        struct monthweekday {
                unsigned int    type: 2,                        /* use month, week number and day
number (value is 2) */
                unsigned int    m: 4,                           /* month [1,12] */
                unsigned int    n: 4,                           /* week number [1,5] */
                unsigned int    d: 4,                           /* day number [0,6] */
                int             offset: 18,                     /* seconds [0,86400] */
        } m;
};
```

When using the total seconds since 0:00, January 1, local time, type = 0 indicates the usual total seconds, and type = 1 indicates the total seconds without counting the leap day.
In the latter case, February 29 cannot be referred to explicitly.

When using the month, week number, and day number (type = 2), the local date and time when the offset seconds have elapsed since 0:00 on the dth day of the nth week of the mth month of the year is specified.

System Time Zone

The only one current time zone of the system is called the "system time zone".
This defines the time difference from the Coordinated Universal Time (UTC) and can be used to represent the local time of the currently running system.
Normally, dt_setsystz sets the time zone of the current region as the system time zone, when starting up the system.
The initial value of the system time zone depends on the implementation. It is set to UTC (Coordinated Universal Time) in the T2EX reference implementation.

In an API with the time zone address argument, if NULL is specified for "struct tzinfo *", then the system time zone is used.

Local Time

The time of a region that is defined by the time zone is called the "local time" of that time zone, in contrast to the Coordinated Universal Time (UTC).
When simply saying the "local time", it means the local time of the system time zone.

Time Zone String

The time zone string is a string representation of the time zone information in one of the following forms:

1. :characters
   String beginning with ":". "characters" depends on the implementation.
   The T2EX reference implementation does not support this form of time zone string.
   If it is used in the dt_tzset() API call, an error occurs.

2. std offset dst offset, rule

   std  Name of standard time
   dst  Name of alternative time

   offset  Value added to the local time to obtain the Coordinated Universal Time (UTC)
           In the form of hh[:mm[:ss]]

           hh  hour [0,24]
           mm  minutes [0,59]
           ss  seconds [0,59]

offset after std is required.
offset after dst can be omitted.
If offset after dst is omitted, dst is assumed to be one hour ahead of offset of std.

If the time zone string begins with "-", it means east of the Greenwich meridian.
For west of the Greenwich meridian, "+" is optional.

rule Indicates when to change to and back from the alternative time.
rule has the following form:
date[/time],date[/time]

The first date indicates when to change from the standard time to the alternative time.
The second date indicates when to back from the alternative time to the standard time.
Each time field indicates when to change to the other time, in local time.

date has the following form:
Jn
Julian day n (1 <= n <= 365).
The leap day must not be counted.
In all years including leap years, n = 59 for February 28 and n = 60 for March 1.
February 29 cannot be referred to explicitly.
n
Zero-based Julian day (0 <= n <= 365).
The leap day is counted, and February 29 can be referred to.
Mm.n.d
dth day of week n of month m of the year (1 <= m <= 12, 1 <= n <= 5, 0 <= d <=6).
n = 1 indicates the first week in which the dth day occurs.
d = 0 indicates Sunday.

time has the same form as offset, except that it has no leading sign ("+" or "-").
If time is omitted, it is assumed to be 02:00:00.

Examples of Time Zone String:
- For New Zealand where there is an offset from UTC and the daylight saving time is applied:
"NZST-12:00:00NZDT-13:00:00,M10.1.0,M3.3.0"

Name of standard time NZST (New Zealand standard time)
Time difference -12 hours (12 hours earlier than UTC)
Name of alternative time NZDT (New Zealand daylight saving time)
Time difference -13 hours (13 hours earlier than UTC)
Date to change to the alternative time The first Sunday of October
Date to back from the alternative time The third Sunday of March

- For Japan:
"JST-9"

There is only the standard name "JST" without alternative name.
The time is 9 hours earlier than UTC, and the daylight saving time is not applied.

System Locale

The locale defines the behavior specific to a country or region, for each category including date and time formatting, character collation order, numeric conventions, and currency symbol.
T2EX does not have a function for setting a locale dynamically and freely, and uses a fixed default locale.
This locale is called the "system locale".
Normally, the system locale uses the fixed values that are customized when configuring the system, for each category depending on the destination country or region of T2EX or depending on the purpose of use. The T2EX reference implementation uses the ISO C locale ("C").
Therefore, API calls for getting error messages will return ASCII strings in English. When the time is converted to a string representation, the month and the day of the week are returned in English.

6.3 API

Only the two functions, dt_setsystz and dt_getsystz, are implemented as system calls, and the other functions are implemented as library functions.

### 6.3.1 dt_main - Initializes and exits the calendar function

C Language Interface

```
#include <t2ex/datetime.h>

ER      er = dt_main(INT ac, UB* arg[]);
```

Parameter

| | | |
|---|---|---|
| INT | ac | number of elements in arg[] or a negative value |
| UB* | arg[] | array of argument strings |

Return Parameter

| | | |
|---|---|---|
| ER | er | error code |

Error Code

| | |
|---|---|
| E_OK | Normal completion |

Description

This function initializes (ac >= 0) or terminates (ac < 0) the calendar function.

At the time of initialization, a number of strings can be passed to arg[] as arguments, and the total count of strings is ac.
The content of "arg" is implementation-dependent. These argument strings are not used in the T2EX reference implementation.

### 6.3.2 dt_setsystz - Set System Time Zone

C Language Interface

```
#include <t2ex/datetime.h>

ER er = dt_setsystz(const struct tzinfo* tz);
```

Parameter

| | | |
|---|---|---|
| const struct tzinfo* | tz | Time zone to set |

Return Parameter

| | | |
|---|---|---|
| ER | er | Error code |

Error Code

| | |
|---|---|
| E_OK | Normal completion |
| EX_FAULT | Illegal tz address |
| EX_INVAL | Illegal tz content |

Description

Changes the system time zone to the time zone specified by tz.

See Also

dt_getsystz,dt_tzset

### 6.3.3 dt_getsystz - Get System Time Zone

C Language Interface

```
#include <t2ex/datetime.h>

ER er = dt_getsystz(struct tzinfo* tz);
```

Parameter

| | | |
|---|---|---|
| struct tzinfo* | tz | Time zone to get |

Return Parameter

| | | |
|---|---|---|
| ER | er | Error code |
| struct tzinfo* | tz | System Time Zone |

Error Code

| | |
|---|---|
| E_OK | Normal completion |
| EX_FAULT | Illegal tz address |

Description

Returns the value of the system time zone to the area specified by tz.

See Also

dt_setsystz,dt_tzset

### 6.3.4 dt_tzset - Initialize Time Zone Structure

C Language Interface

```
#include <t2ex/datetime.h>

ER er = dt_tzset(struct tzinfo* tz, const char* spec);
```

Parameter

| | | |
|---|---|---|
| struct tzinfo* | tz | Address of the time zone to store |
| const char* | spec | Time zone string |

Return Parameter

| | | |
|---|---|---|
| ER | er | Error code |
| struct tzinfo* | tz | Time zone |

Error Code

| | |
|---|---|
| E_OK | Normal completion |
| EX_INVAL | spec is illegal as time zone string |

Description

Initializes the time zone structure tz based on the time zone string specified by spec.

See Also

dt_getsystz, dt_setsystz

### 6.3.5 dt_localtime, dt_localtime_ms, dt_localtime_us - Convert to Local Time

C Language Interface

```
#include <t2ex/datetime.h>
```

```
ER er = dt_localtime(time_t tims, const struct tzinfo* tz, struct tm* result);

ER er = dt_localtime_ms(const SYSTIM* tim, const struct tzinfo* tz, struct tm* result);

ER er = dt_localtime_us(SYSTIM_U tim_u, const struct tzinfo* tz, struct tm* result);
```

Parameter

| | | |
|---|---|---|
| time_t | tims | System time in seconds |
| const SYSTIM* | tim | Pointer to the system time in milliseconds |
| SYSTIM_U | tim_u | System time in microseconds |
| struct tzinfo* | tz | Pointer to the time zone |
| struct tm | *result | Pointer to the area to store the local time |

Return Parameter

| | | |
|---|---|---|
| ER | er | Error code |
| struct tm* | result | Converted local time |

Error Code

| | |
|---|---|
| E_OK | Normal completion |
| EX_FAULT | Illegal parameter address |
| EX_OVERFLOW | Result cannot be expressed in tm |

Description

Converts the system time indicated by tims, tim, or tim_u to the local time of the time zone specified by tz, and stores the result to the tm structure indicated by result.

If NULL is specified for tz, the system time zone is used.

See Also

dt_setsystz, dt_getsystz, dt_tzset, dt_strftime, dt_strptime, dt_mktime, dt_gmtime

### 6.3.6 dt_strftime - Convert Date and Time to String

C Language Interface

```
#include <t2ex/datetime.h>

int nb = dt_strftime(char *s, size_t max, const char* format, const struct tm* tm, const struct
tzinfo* tz);
```

Parameter

| | | |
|---|---|---|
| char* | s | Address of the string to store the result |
| size_t | max | Size of s (in bytes) |
| const char* | format | Conversion format specifier string |
| const struct tm* | tm | Time to be converted to a string |
| const struct tzinfo | tz | Time zone |

Return Parameter

| | | |
|---|---|---|
| int | nb | Number of bytes of the result not including the null character, or error code |
| char* | s | Converted string |

Error Code

| | |
|---|---|
| EX_INVAL | Illegal parameter |

Description

dt_strftime() converts the time represented by struct tm to a string according to the format specified

by format and the time zone specified by tz, and writes the result to the string s of up to max characters including the null character.
When converting the time to a string representation, the conversion result varies depending on the locale information.
dt_strftime() uses the information from the default locale of the system (system locale).

format is a character string representing the conversion format, consisting of 0 or more conversion specification strings (conversion specifiers) and ordinary characters.
Each conversion specifier begins with the character "%", followed by the sequence of the following characters in this order:

1. E or O modifier (optional). Details are given later.

2. Conversion specifier character which determines the conversion type

The ordinary characters including the terminating null character are copied to the result.
If copying takes place between overlapping objects, the result is undefined. (For example, s and format have the same value)

No more than max bytes are written to the string s.
Each conversion specifier is replaced by an appropriate string described in the list below.
The appropriate string is determined by 0 or more time elements pointed by the system locale and tm.
If specified values are out of the normal ranges, the stored string is undefined.
If tz is NULL, the system time zone is used when interpreting the "%Z" and "%z" specifiers.

The following conversion specifier characters are supported.
"[ ]" indicate the range of number when the result is a numeric string.
"{ }" indicate the tm structure member to be referred to.

a
Abbreviated name of day of the week {tm_wday}
A
Full name of day of the week {tm_wday}
b
Abbreviated name of month {tm_mon}
B
Full name of month {tm_mon}
c
General date and time in the system locale
C
The year divided by 100 and truncated to an integer, as a decimal number. [00,99] {tm_year}
d
Day of the month [01,31] {tm_mday}
D
Equivalent to %m/%d/%y {tm_mon, tm_mday, tm_year}
e
Day of the month.
If single digit, a white-space is prepended [1,31] {tm_mday}
F
Equivalent to %Y-%m-%d.{tm_mon, tm_mday, tm_year}
g
Last two digits of the week-based year (see below) [00,99] {tm_year, tm_wday, tm_yday}
G
Week-based year {tm_year, tm_wday, tm_yday}
h
Equivalent to %b {tm_mon}
H
Hour (24-hour clock) [00,23] {tm_hour}
I
Hour (12-hour clock) [01,12] {tm_hour}
j
days since January 1 [001,366] {tm_yday}
m
Month [01,12] {tm_mon}
M
Minutes [00,59] {tm_min}
n
New line
p
String equivalent to a.m. or p.m. in the system locale
r
Time in a.m. and p.m. notation {tm_hour, tm_min, tm_sec}
R

```
        Time in 24-hour notation (%H:%M) {tm_hour, tm_min}
S
        Seconds [00,60] {tm_sec}
t
        Tab
T
        Time (%H:%M:%S) {tm_hour, tm_min, tm_sec}
u
        Day of the week where 1 is Monday [1,7] {tm_wday}
U
        Week number of the year [00,53]
        The first Sunday of January is the first day of the week number 1.
        Days before it in the new year have the week number 0 {tm_year, tm_wday, tm_yday}
V
        Week number of the year (Monday as the first day of the week) [01,53]
        If the week containing January 1 has four or more days, it is considered the week number 1.
        Otherwise, the last week of the previous year and the next week is the week number 1.
        January 4 and the first Thursday of January always have the week number 1.
{tm_year, tm_wday, tm_yday}
w
        Day number of the week [0,6].
0 is Sunday.
{tm_wday}
W
        Week number of the year [00,53].
The first Monday of January has the week number 1.
Days before it have the week number 0.
        {tm_year, tm_wday, tm_yday}
x
        Date representation in the system locale
X
        Time representation in the system locale
y
        Last two digits of the year [00,99] {tm_year}
Y
        Year {tm_year}
z
        Offset from UTC in the ISO 8601:2000 standard format
Z
        System time zone name
%
        "%" character

E or O Modifier

Some conversion specifiers can be modified with the E or O modifier.
When modified with this modifier, the result is output using an alternative format, if any.
If the alternative format does not exist for the system locale, the behavior is the same as the
specifier without modifier.

%Ec
        Locale's alternative date and time
%EC
        Name of the base year (period) in the locale's alternative representation
$Ex
        Locale's alternative date
%Ey
        Locale's alternative time
%EY
        Full alternative representation of year
%Od
        Day of the month using the locale's alternative numeric symbols.
        If there is any alternative symbol for zero, the beginning is filled with zeros as needed;
otherwise, filled with spaces.
%Oe
        Day of the month using the locale's alternative numeric symbols.
        The beginning is filled with spaces, if necessary.
%OH
        Hour (24-hour clock) using the locale's alternative numeric symbols
%OI
        Hour (12-hour clock) using the locale's alternative numeric symbols
%Om
        Month using the locale's alternative numeric symbols
```

%OM
        Minutes using the locale's alternative numeric symbols
%OS
        Seconds using the locale's alternative numeric symbols
%Ou
        Day number of the week (Monday = 1) using the locale's alternative numeric symbols
%OU
        Week number of the year using the locale's alternative numeric symbols
        (Sunday as the first day of the week, similar to %U.)
%OVV
        Week number of the year using the locale's alternative numeric symbols
        (Monday as the first day of the week, similar to %V.)
%Ow
        Day number of the week (Sunday = 0) using the locale's alternative numeric symbols
%OW
        Week number of the year using the locale's alternative numeric symbols (Monday as the first
day of the week)
%Oy
        Year (offset from %C) using the locale's alternative numeric symbols


%g, %G and %V use the week number since the beginning of the year according to the ISO 8601:2000 standard.
In this system, a week begins on Monday, and the first week of the year contains January 4 and also contains the first Thursday of the year.
In addition, the first week of the year contains at least four days.

If the first Monday of January is the 2nd, 3rd, or 4th day, the first 1 to 3 days are part of the last week of the previous year.
Therefore, for Saturday, January 2, 1999, %G is replaced with 1998, and %V is replaced with 53.
If December 29, 30, or 31 is Monday, those days are part of the first week of the next year.
Therefore, for Tuesday, December 30, 1997, %G is replaced with 1998, and %V is replaced with 01.

If a conversion specifier is other than the above ones, the behavior is undefined.

See Also

dt_strptime, dt_mktime

### 6.3.7 dt_strptime - Convert String to Date and Time

```
#include <t2ex/datetime.h>

int index = dt_strptime(const char *str, const char *format, struct tm *tm, const struct tzinfo* tz);
```

Parameter

| | | |
|---|---|---|
| const char* | str | String to be converted |
| const char* | format | Conversion format string |
| struct tm* | tm | tm structure to store the result date and time |
| const struct tzinfo* | tz | Time zone |

Return Parameter

| | | |
|---|---|---|
| int | index | Index to the first unprocessed character in str, or error code |
| struct tm* | tm | Time converted from string |

Error Code

None

Description

dt_strptime() converts the character string indicated by str to the time in tm structure according to the format specified by format and the time zone specified by tz, and stores the result into tm.

format consists of 0 or more directives, and each directive is one of the following:
 - One or more white-space characters for which isspace() is true
 - Ordinary character excluding "%" and white-space characters described above
 - Conversion specifier

Each conversion specifier begins with "%", followed by the following in this order:
1. Flag character (optional)
        "0" or "+", but ignored if exists.
2. Field width (optional)
        Decimal numeric string specifying the maximum number of bytes to convert.
        This takes priority over the number of bytes required by the conversion specifier.
3. E or O modifier (optional)
4. Conversion specifier which determines the conversion type

The system locale is used as the locale information required for conversion.

An application shall ensure that there are white-space or other non-alphanumeric characters between any two conversion specifiers, unless all of the adjacent conversion specifiers convert a known fixed number of characters.

The maximum number of scanned characters (excluding the one matching the next directive) is determined as follows:
 - If the field width is specified, then that number.
 - Otherwise, the pattern "{x}" indicates that the maximum is x.
 - Otherwise, the pattern "[x,y]" indicates that the value falls within the specified range (including both bounds), and the maximum number of scanned characters is the maximum that can represent any value within the range without leading zeros or a leading plus sign.

The following conversion specifiers are supported.
If a modifier is specified with a flag or a minimum field width, or if the field width is specified for any conversion specifier other than C, F, or Y, the result is undefined.

a
        Day of the week using the locale's name of day of the week.
Either the abbreviated or full name may be specified.
A
        Equivalent to %a.
b
        Month using the locale's name of month.
Either the abbreviated or full name may be specified.
B
        Equivalent to %b.
c
        Locale's standard date and time.
C
        The century number [00,99]; leading zeros are permitted but not required.
        Leading zeros are permitted, but not required.
        A leading "+" or "-" is permitted, but not required.
d
        Day of the month [01,31].
Leading zeros are not required.
D
        Same as %m/%d/%y.
e
        Equivalent to %d.
h
        Equivalent to %b.
H
        Hour [00,23].
Leading zeros are not required.
I
        Hour [01,12].
Leading zeros are not required.
j
        Day number of the year [001,366].
Leading zeros are not required.
m
        Month [01,12].
Leading zeros are not required.
M
        Minutes [00,59].
Leading zeros are not required.
n
        White-space character
p
        Locale's representation equivalent to a.m. or p.m.

r
        12-hour clock time in a.m./p.m. notation.
R
        Same as %H:%M.
S
        Seconds [00,60].
Leading zeros are not required.
t
        White-space character.
T
        Same as %H:%M:%S.
U
        Week number of the year (Sunday as the first day of the week) [00,53].
Leading zeros are not required.
w
        Day number of the week [0,6].
0 is Sunday.
W
        Week number of the year (Monday as the first day of the week) [00,53].
Leading zeros are not required.
x
        Date using the locale's date format.
X
        Time using the locale's time format.
y
        Last two digits of the year.
When format contains neither C nor Y, a value [69,99] means a year from 1969 to 1999, and a value [00,68] means a year from 2000 to 2068.
Leading zeros are not required.
        A leading "+" or "-" character is permitted, but not required.
Y
        Year represented by four digits.
Leading zeros are not required.
        A leading "+" or "-" character is permitted, but not required.
%
        "%" character.

E or O Modifier

Some conversion specifiers can be modified by the E and O modifier so that an alternative format is used.
If the alternative format does not exist in the system locale, the behavior is the same as the specifier without modifier.

%Ec
        Locale's alternative date and time
%EC
        Name of the base year (period) in the locale's alternative representation
%Ex
        Locale's alternative date
%EX
        Locale's alternative time
%Ey
        Offset from %EC (year only) using the locale's alternative representation
%EY
        Full alternative representation of year
%Od
        Day of the month using the locale's alternative numeric symbols.
Leading zeros are not required.
%Oe
        Equivalent to %Od
%OH
        Hour (24-hour clock) using the locale's alternative numeric symbols
%OI
        Hour (12-hour clock) using the locale's alternative numeric symbols
%Om
        Month using the locale's alternative numeric symbols
%OM
        Minutes using the locale's alternative numeric symbols
%OS
        Seconds using the locale's alternative numeric symbols
%OU
        Week number of the year (Sunday as the first day of the week) using the locale's alternative

numeric symbols
%Ow
Day number of the week (Sunday = 0) using the locale's alternative numeric symbols
%OW
Week number of the year (Monday as the first day of the week) using the locale's alternative numeric symbols
%Oy
Year (offset from %C) using the locale's alternative numeric symbols

If a directive consists of white-space characters, characters are scanned until the first character that is not white-space or the last one.

If a directive is an ordinary character, the next character is scanned from the buffer.
If the scanned character differs from the one for the directive, the directive fails, and that character and the subsequent characters remain unscanned.

If a conversion specification consists of %n, %t, white-space characters, or any combination of them, characters are scanned until the first character that is not white-space (which remains unscanned) or the last one.

For any other directive, characters are scanned until a character matching the next directive or the last one.
These characters, except the one matching the next directive, are compared to the locale values associated with the conversion specifier.
If a match is found, the value of an appropriate tm structure member is set to the value corresponding to the locale information.
Case is ignored when matching items in str such as month name or day name of the week.
If no match is found, dt_strptime() fails, and no more characters are scanned.

See Also

dt_strftime

### 6.3.8 dt_mktime, dt_mktime_ms, dt_mktime_us - Convert to System Time

C Language Interface

```
#include <t2ex/datetime.h>

ER er = dt_mktime(struct tm* tm, const struct tzinfo* tz, time_t* result);
ER er = dt_mktime_ms(struct tm* tm, const struct tzinfo* tz, SYSTIM* result);
ER er = dt_mktime_us(struct tm* tm, const struct tzinfo* tz, SYSTIM_U* result);
```

Parameter

| | | |
|---|---|---|
| struct tm* | tm | Time data to be converted |
| const struct tzinfo* | tz | Time zone |
| time_t* | result | Converted system time (in seconds) |
| SYSTIM* | result | Converted system time (in milliseconds) |
| SYSTIM_U* | result | Converted system time (in microseconds) |

Return Parameter

| | | |
|---|---|---|
| ER | er | Error code |
| struct tm* | tm | tm_wday and tm_yday are set appropriately |
| time_t* | result | Converted system time (in seconds) |
| SYSTIM* | result | Converted system time (in milliseconds) |
| SYSTIM_U* | result | Converted system time (in microseconds) |

Error Code

| | |
|---|---|
| E_OK | Normal completion |
| EX_OVERFLOW | The conversion result cannot be represented in the specified format |

Description

Converts the local time elements specified by tm to the system time in the format specified by the API call under the time zone specified by tz.

dt_mktime converts the time to the time_t format in seconds.
dt_mktime_ms converts the time to the SYSTIM format in milliseconds.
dt_mktime_us converts the time to the SYSTIM_U format in microseconds.
If tz is NULL, the system time zone is used.
Among the original values of tm, tm_wday and tm_yday are ignored.

If successful, the tm_wday and tm_yday elements are set appropriately, and other elements are set to the values from the start time of the system time (0:00:00 (GMT), January 1, 1985).
NULL can be specified for result.
Even in this case, the elements of tm are set appropriately.

If the result cannot be represented in the result type, the contents of tm are not changed, and the EX_OVERFLOW error is returned.
See Also

dt_gmtime

### 6.3.9 dt_gmtime, dt_gmtime_ms, dt_gmtime_us - Convert to UTC Time

C Language Interface

```
#include <t2ex/datetime.h>

ER er = dt_gmtime(time_t tim, struct tm* result);
ER er = dt_gmtime_ms(const SYSTIM* tim_m, struct tm* result);
ER er = dt_gmtime_us(SYSTIM_U tim_u, struct tm* result);
```

Parameter

| | | |
|---|---|---|
| time_t | tim | System time (in seconds) |
| const SYSTIM* | tim_m | System time (in milliseconds) |
| SYSTIM_U | tim_u | System time (in microseconds) |
| struct tm* | result | Result of the conversion to UTC time |

Return Parameter

| | | |
|---|---|---|
| ER | er | Error code |
| struct tm* | tm | tm structure representing the UTC time |

Error Code

| | |
|---|---|
| E_OK | Normal completion |
| EX_OVERFLOW | Conversion result cannot be expressed |

Description

Converts the system time to the time elements expressed as Coordinated Universal Time (UTC) and stores them in result.
dt_gmtime gives the system time in seconds.
dt_gmtime_ms gives the system time in milliseconds.
dt_gmtime_us gives the system time in microseconds.
The resolution of result is the same as for tim_u, tim_m, or tim.

See Also

dt_mktime

# Chapter 7 Program Load Function

## 7.1 Overview

The program load function loads and executes the program modules in T2EX.
The API name prefix is "pm_" (program module).

Program modules for this function are classified into the following two types.

Regular Program Module
This program module runs at the same protection level as for the user side when it is used from an application program or a system-level component.
When using the module after loading, it is executed as a function call without an SVC interruption.

System Program Module
This means the privileged program module located in the system memory space and refers to subsystems and device drivers in T-Kernel 2.0.
This is used assuming that the user-level program provides the system-related, privileged level operations to applications in a safe and organized form.
The system program function is used via an SVC interruption using a device driver interface or subsystem entry point.

The essential difference between the regular program and system program modules is the protection level at which the program module is executed.
The former module function is executed at the caller's protection level, and the latter always at the system level regardless of the caller's runtime protection level.
Generally, it is desirable to use the regular program module for the purpose of dividing an application program into module units, in terms of performance efficiency and safety.
The system program module should be used only for the purpose of providing a system-level interface including device drivers and subsystems.

## 7.2 Regular Program Module

### 7.2.1 Regular Program Module Interface

The entry point for a regular program module is named module_main and has the following format.

```
int module_main(BOOL startup, void* arg)
{
        if ( startup ) {
                /* regular program startup processing */
        }
        else {
                /* regular program termination processing */
        }

        return ercd;
}
```

If startup is TRUE, it means to execute the startup processing, and thus the module must be initialized to be available.
If an error (negative value) is returned, the startup is considered to be failed, and the resources reserved by the startup processing must be released.
The argument arg can be used as startup processing parameters.

If startup is FALSE, the termination processing is executed.
The termination processing must release the resources allocated by the module.
If an error occurs during the termination processing, it must not be aborted, and the resources shall be released wherever possible.
If some part of the termination processing could not be executed normally, an error is returned as the return code.
The argument arg can be used as termination processing parameters.

The entry point itself shall only perform the startup and termination processings, and the service interfaces of the module shall be provided separately.
This specification does not specifically limit the methodology. A typical one is to set the pointers of the module interface functions in the area indicated by the startup argument arg.
For specific module definition examples, see Appendix A.1.2.

### 7.2.2 Usage of Regular Program Module

To start using a regular program module, follow the procedure below.

1. Load the regular program module
	Load the target program module onto the memory.
This can be performed using the API call pm_load in this function.

2. Call the startup processing
	Call the entry point obtained by the pm_load API call, setting startup is TRUE.
	This executes the startup processing of the module.

To terminate a regular program module, follow the procedure below.

1. Call the termination processing
	Call the entry point obtained by the pm_load API call, setting startup is FALSE.
	This executes the termination processing of the module to release the resources allocated by the module.
2. Unload the regular program module
	Unload the target program module from the memory.
This can be performed using the API call pm_unload in this function.

For specific module usage examples, see Appendix A.1.2.

## 7.3 System Program Module

The interface specification of a system program complies with the T-Kernel 2.0 subsystem and device driver interfaces.
The entry point format is compliant with it, as follows.
For specification details, see Section 5.11 "Subsystem and Device Driver Starting" in the T-Kernel 2.0 Specification.

```
ER main(INT ac, UB* av[])
{
	if ( ac >= 0 ) {
		/* system program startup processing */
	}
	else {
		/* system program termination processing */
	}

	return ercd;
}
```

The loading and startup of a system program module are an indivisible processing. The same goes for its termination and unloading. These are executed together with the API calls pm_loadspg and pm_unload in this function.
For details about them, see Section 7.5.

## 7.4 Data Type Definition

- pm_entry_t
	Type of the entry point (module_main) of the regular program module

```
	typedef int pm_entry_t(BOOL startup, void* arg);
```

- struct pm
	Program module to load

```
	struct pm {
		ATR	pmtype;  /* program type (PM_FILE, PM_PTR) */
		void*	pmhdr;   /* program to load */

		/* other implementation-dependent information */
	};
```

	pmtype is specified as follows.
		pmtype := (PM_FILE || PM_PTR)

	When pmtype is PM_FILE, the program module to load is a file, and the pointer to the string (char* type) indicating the file path name is specified in pmhdr.
When pmtype is PM_PTR, the target program module is assumed to be located on the memory, and the top address of the memory space where the module locates is specified in pmhdr.

The format of the program module to be loaded is not defined in this specification and shall depend on the implementation.

## 7.5 API

All the API calls in this function are implemented as system calls.
If the return code of an API call provided by this function is negative, the return code shall be interpreted as an ER type extended error code.

### 7.5.1 pm_main - Initializes and exits the program load function

C Language Interface

```
#include <t2ex/load.h>

ER      er = pm_main(INT ac, UB* arg[]);
```

Parameter

| | | |
|---|---|---|
| INT | ac | number of elements in arg[] or a negative value |
| UB* | arg[] | array of argument strings |

Return Parameter

| | | |
|---|---|---|
| ER | er | error code |

Error Code

| | |
|---|---|
| E_OK | Normal completion |

Description

This function initializes (ac >= 0) or terminates (ac < 0) the program load function.

At the time of initialization, a number of strings can be passed to arg[] as arguments, and the total count of strings is ac.
The content of "arg" is implementation-dependent. These argument strings are not used in the T2EX reference implementation.

### 7.5.2 pm_load - Load Regular Program Module

C Language Interface

```
#include <t2ex/load.h>

ID progid = pm_load(const struct pm* prog, UINT attr, pm_entry_t** entry);
```

Parameter

| | | |
|---|---|---|
| const struct pm* | prog | Program to load |
| UINT | attr | Attribute of the destination memory space |
| pm_entry_t** | entry | Pointer to the area to return the program entry point |

Return Parameter

| | | |
|---|---|---|
| ID | progid | Program ID (> 0) for normal completion, or negative error code |
| pm_entry_t* | entry | Program entry point |

Error Code

| | |
|---|---|
| E_LIMIT | The number of loaded programs reached the upper limit |
| E_MACV | The memory address given as an argument is inaccessible (prog, entry) |
| EX_ACCES | There is no permission to load the target program |
| EX_FBIG | The target program file is too big |
| EX_INTR | Aborted by fs_break() |
| EX_INVAL | Illegal parameter<br>- ac < 0 |

| | |
|---|---|
| | - prog or attr is invalid |
| EX_IO | I/O error |
| EX_ISDIR | prog is a directory |
| EX_NFILE | The number of opened files in the system exceeded the limit |
| EX_NODEV | The connection name of the file to load does not exist |
| EX_NOENT | The file to load does not exist |
| EX_NOEXEC | The format of the program to load is illegal |
| EX_PERM | There is no permission to load the target program |

Description

Loads the regular program module prog onto the memory.
The module startup processing (entry point) is not called.

The attribute of the destination memory space is specified using attr as follows:
attr := (TA_RNG0 || TA_RNG1 || TA_RNG2 || TA_RNG3)

For successful loading, the program ID is returned as the return code.
The pointer to the entry point of the loaded regular program module is returned to *entry.
By giving TRUE and FALSE as the first argument of the obtained function pointer, the initialization and termination processings can be executed for the module, respectively.

The loaded program module can be executed repeatedly by simple function calls via the function pointer of the obtained entry point until it is unloaded.

See Also

pm_loadspg, pm_unload

### 7.5.3 pm_loadspg - Load System Program Module

C Language Interface

```
#include <t2ex/load.h>

ID progid = pm_loadspg(const struct pm* prog, INT ac, UB* av[]);
```

Parameter

| | | |
|---|---|---|
| const struct pm* | prog | Program to load |
| INT | ac | Number of arguments passed to the entry point at startup processing (>= 0) |
| UB* | av[] | String argument passed to the entry point at startup processing |

Return Parameter

| | | |
|---|---|---|
| ID | progid | Program ID (> 0) for normal completion, or negative error code |

Error Code

| | |
|---|---|
| E_LIMIT | The number of loaded programs reached the upper limit |
| E_MACV | The memory address given as an argument is inaccessible (prog, av) |
| EX_ACCES | There is no permission to load the target program |
| EX_FBIG | The target program file is too big |
| EX_INTR | Aborted by fs_break() |
| EX_INVAL | Illegal parameter<br>- ac < 0<br>- prog or av is invalid |
| EX_IO | I/O error |
| EX_ISDIR | prog is a directory |
| EX_NFILE | The number of opened files in the system exceeded the limit |
| EX_NODEV | The connection name of the file to load does not exist |
| EX_NOENT | The file to load does not exist |
| EX_NOEXEC | The format of the program to load is illegal |
| EX_PERM | There is no permission to load the target program |

Description

Loads the system program module prog onto the privileged level memory space, then executes the startup processing as a quasi-task portion of this API's caller task.

When the entry point returns 0 or a positive value in the startup processing, the startup processing is assumed to have been executed normally, and the program ID is returned as the return code.
When the entry point returns a negative value, the system program load is considered to be failed, and the return code from the entry point is returned as is.

An execution of this API call itself completes at the termination of the startup processing (return from the entry point), and the program remains mapped to the memory until it is unloaded.

See Also

pm_load, pm_unload

### 7.5.4 pm_status - Refer to Program Module Information

C Language Interface

```
#include <t2ex/load.h>

ER ercd = pm_status(ID progid, struct pm_stat* status);
```

Parameter

| | | |
|---|---|---|
| ID | progid | Target program ID |
| struct pm_stat* | status | Pointer to the area to return the program status |

Return Parameter

| | | |
|---|---|---|
| ER | ercd | Error code |
| Contents of status: | | |
| BOOL | sysprg | Program attribute (TRUE: system program, FALSE: regular program) |
| UINT | attr | Attribute of the destination memory |
| void* | entry | Program entry point address |
| void* | start | Start address of the program |
| void* | end | End address of the program |

Error Code

| | |
|---|---|
| E_OK | Normal completion |
| E_ID | The program ID is illegal |
| E_MACV | The memory address given as an argument is inaccessible (status) |

Description

Refers to the information of the program indicated by the program module ID progid and stores the information in *status.

See Also

pm_loadspg, pm_load, pm_unload

### 7.5.5 pm_unload - Unload Program Module

C Language Interface

```
#include <t2ex/load.h>

ER ercd = pm_unload(ID progid);
```

Parameter

| | | |
|---|---|---|
| ID | progid | Target program ID |

Return Parameter

| | | |
|---|---|---|
| ER | ercd | Error code |

Error Code

| | |
|---|---|
| E_OK | Normal completion |
| E_ID | The program ID is illegal |

Description

Unloads the program module indicated by the program module ID progid.

If the unloaded program module is executed or the memory space of the unloaded program is referred to, the behavior is not guaranteed.

When this API call is executed for the system program module loaded by pm_loadspg(), it works as follows:

- The entry point of the target system program (main function) is called for the termination processing.
  At this time, the first argument ac is (-1) to distinguish the call from the initialization processing at loading.
  The called main function is executed as a quasi-task portion of the task issuing pm_unload().

- If the main function returns 0 or a positive value, the termination processing is assumed to be successful, and the unloading of the program is executed.
If the main function returns a negative value, the termination processing is assumed to be unsuccessful, and the negative return code from the main function is returned as the return code of the pm_unload() as is without unloading.

If this API call is executed for a regular program module loaded by pm_load(), only the unloading of the program is executed without calling the termination processing.
Therefore, the termination processing (entry point) must always be called before executing the unloading using this API call.

See Also

pm_loadspg, pm_load

# Chapter 8 Standard C Compatible Library

## 8.1 Overview

The Standard C Compatible Library is a library group highly compatible with the Standard C Library (ISO/IEC 9899:1999).
T2EX places importance on thread-safety of the API.
Any thread-unsafe Standard C Library functions are excluded, replaced with different ones with equivalent capabilities, or added with arguments to ensure thread safety.
In addition, functions related to capabilities not provided by T2EX such as locale are deleted.
Therefore, this is not fully compatible with the Standard C Library.

T2EX provides the following header files and the functions defined therein.

| | |
|---|---|
| arpa/inet.h | BSD socket * 1 |
| assert.h | Testing function |
| complex.h | Complex calculation |
| ctype.h | Character type classification |
| dirent.h | Directory reading * 2 |
| errno.h | Error number definition |
| float.h | Floating point type properties |
| inttypes.h | Integer type format conversion |
| iso646.h | Alternate spellings |
| limits.h | Various limit values |
| math.h | Numeric operation |
| netinet/in.h | BSD socket * 1 |
| search.h | Search * 2 |
| stdarg.h | Variable number actual argument |
| stdbool.h | Boolean type and boolean value |
| stddef.h | Common definition |
| stdint.h | Integer type |
| stdio.h | Standard input/output |
| stdlib.h | General utility |
| string.h | String operation |
| time.h | Date and time |
| wchar.h | Multibyte and wide character extension |

* 1 The headers defined for the BSD socket are added in order to handle a socket that realizes TCP/IP.

* 2 This is added based on POSIX (IEEE Std 1003.1-2008) as versatile functions.

The following headers defined in the Standard C Library (ISO/IEC 9899:1999) are removed from the Standard C Compatible Library.

| | |
|---|---|
| fenv.h | Floating point environment<br>* T2EX does not define floating point exceptions. |
| locale.h | Locale operation<br>* T2EX does not support functions for locale operation and uses the system locale only. |
| setjmp.h | Non-local jump<br>* Generally, it has restriction to be used under multitask. |
| signal.h | Signal operation<br>* There is no signal in T2EX. |
| tgmath.h | Type-generic mathematical function<br>* There is no need of it in T2EX. |
| wctype.h | Wide character type classification and case conversion<br>* T2EX does not support the wide character/multibyte character libraries. |

## 8.2 Compatibility

T2EX does not support some of the functions defined in each Standard C Library header.
Some function names are added with a postfix to add arguments.
Therefore, the T2EX Standard C Compatible Library is not fully compatible with the Standard C Library.
It has the following differences from the Standard C Library.

### File and Socket

In the Standard C Library, files and sockets can be handled in the same way as a stream.
In the T2EX Standard C Compatible Library, files and sockets cannot be handled with the same functions.
Separate functions exist for handling files and sockets.

Particularly in the C language standard input/output library defined in stdio.h, the functions can be used only with files.
The standard input/output functions cannot be used with sockets.

Error Code and Error Number

In the T2EX system call group, all the API calls have ER type return codes, which tells the error details.
The Standard C Library functions use -1 or NULL as the return codes to notify the occurrence of an error, which do not give detailed error information.
Usually a POSIX specification-compliant operating system gets error information by reading the content of the symbol errno.
An errno value is a positive integer indicating the detailed error information. This positive integer is called an "error number" which is distinguished from a negative error code of the ER type in T2EX.

Supplement: T2EX does not introduce static variables equivalent to errno in the POSIX specification.
A possible specification or implementation may define errno using macros and functions to provide errno with task-by-task error information, which is not adapted by T2EX due to complication and runtime overhead.

In the Standard C Compatible Library functions described in this chapter, their return codes notify of an error occurrence but do not tell detailed information of the error.
In addition, there is no function that returns an ER type error code as for the API calls in the system call group.

With the Standard C Compatible Library functions, the error number equivalent to errno is retrieved as follows:
1. Call the function that returns the error number occurred.
   ferror() function.
2. Add the function itself with an argument that specifies the area to which the error number is returned, then return the error number to the area specified by the argument.
   fopen_eno(), and so on.

The error number is represented by the following type.

```
typedef int             errno_t;
```

The error number is mapped to the T2EX error code system and handled in a unified manner as the ER type, as follows:
An error code is derived for an error number as follows:
- EC_ERRNO as the main error code.
- Error number as the sub error code.
- EX_.... is defined as the error code symbol corresponding to the error number.

An error code that has EC_ERRNO as the main error code is called an "extended error code".
The following macro is used to convert an error number value (eno) to a T2EX ER type extended error code.

```
#define ERRNOtoER(eno)  (ERCD(EC_ERRNO, (eno))
```

By the following macro, an errno_t type error number value can be obtained from an ER type extended error code (er).

```
#define ERRNO(er)               (MERCD(er) == EC_ERRNO ? SERCD(er) : 0)
```

- In T2EX, error numbers are not retrieved by errno defined in the Standard C library. Therefore, certain Standard C Library functions are added with an argument that returns the error number, with the _eno postfix added to their names.
  They include the following functions.
Their features are equivalent to the original name functions.

```
fopen_eno
fdopen_eno
freopen_eno
fclose_eno
opendir_eno
closedir_eno
gmtime_r_eno
localtime_r_eno
mktime_eno
realpath_eno
```

realpath2_eno

For details about the error number, see the errno.h section.

Thread-Safe

T2EX does not support the following thread-unsafe functions.

asctime
ctime
gets
gmtime
localtime
rand
readdir
srand
strerror
strtok

The following thread-unsafe functions are changed to be thread-safe by adding arguments and the _r postfix to the function name.

lgamma_r
lgammaf_r
lgammal_r

drand48_r
lrand48_r
mrand48_r
srand48_r
seed48_r
lcong48_r

Large file

T2EX supports 64-bit size large files.
The following functions are added to handle 64-bit file offsets, using different names than the conventional ones to make available both of them.

fgetpos64
fsetpos64
fseek64
ftell64

File lock

T2EX does not support the file lock-related functions.

flockfile
ftrylockfile
funlockfile
_unlocked

Pipe

T2EX does not support the pipe handling functions because there is no process.

popen
pclose

Locale

T2EX does not provide locale.h.
Therefore, it does not provide the functions to arbitrarily set a locale and get the locale information in the lconv structure.
However, it refers to the fixed default locale of the system in case it needs the locale information for character matching or other purpose.
This default locale is called the "system locale".
The system locale value is implementation-dependent.
In the T2EX reference implementation, the system locale is USA (en_US).

Other

The function that implicitly needs errno is not supported.

perror

Function notation

Unlike in the previous chapters, this chapter describes each function in the following format.

[Name]
[Format]
[Description]
[Return code]
[Error]
[Related item]
[Additional notes]

The error number described in the [Error] section is obtained by the following ways when the function notifies of an error occurrence by returning a value specific to the function such as -1 and NULL.
- Return code of the ferror() function
- Error number stored in the errno_t* argument if the function has this type of argument

API call

The functions and macros provided by the Standard C Compatible Library are part of the T2EX API, and each of them corresponds to a T2EX API call.
In this chapter, they are not referred to as API calls, in terms of compatibility, but as "functions" and "macros" according to the general Standard C Library description.

Character and string

T2EX assumes that UTF-8 is used as the character code.
In UTF-8, one character is not always one byte. It uses multibyte characters represented by multiple bytes.
To represent every one character, there is the wide character wchar_t type.
However, the T2EX Standard C Compatible Library does not provide the library functions related to multibyte and wide characters.
In this chapter, "character" means one byte.
"String" means a byte sequence that ends with NULL.

Floating point

As a rule, the floating point representation and operation shall comply with the IEC 60559 (IEEE 754).
Implementations that cannot comply with it for architectural reasons are also allowed.
However, the floating point exceptions (INVALID, DENORMAL, ZERIDIVIDE, OVERFLOW, UNDERFLOW, and INEXACT) defined in this specification do not occur in T2EX.
For operations that generate such an exception, NAN, INFINITY, HUGE_VAL, or other appropriate value is returned.

Other

[0,1] is used to represent a numerical range.
This is the range between 0 and 1 inclusive.
[0.0, 1.0) means 0.0 or larger and less than 1.0.
(0.0, 1.0] means larger than 0.0 and 1.0 or less.
+/-0 means +0 or -0.

## 8.3 arpa/inet.h

The header arpa/inet.h defines the following macros, structures, and function prototype declarations.

Types

in_port_t
: The type defined in <netinet/in.h>.

in_addr_t
: The type defined in <netinet/in.h>.

in_addr
: The structure defined in <netinet/in.h>.

Macros

INET_ADDRSTRLEN
: The macro defined in <netinet/in.h>.

Functions or Macros

### 8.3.1 htonl, htons, ntohl, ntohs - convert values between host and network byte order

C Language Interface

```
#include <arpa/inet.h>

uint32_t        htonl(uint32_t hostlong);
uint16_t        htons(uint16_t hostshort);
uint32_t        ntohl(uint32_t netlong);
uint16_t        ntohs(uint16_t netshort);
```

Description

These functions shall convert 16-bit and 32-bit quantities between network byte order and host byte order.
Though htonl(), htons(), ntohl(), and ntohs() are implemented as functions in the T2EX reference implementation, these can be also implemented as macros.

Return Parameter

The htonl() and htons() functions shall return the argument value converted from host to network byte order.
The ntohl() and ntohs() functions shall return the argument value converted from network to host byte order.

Error Code

None.

### 8.3.2 inet_addr - IPv4 address manipulation

C Language Interface

```
#include <arpa/inet.h>

in_addr_t       inet_addr(const char *cp);
```

Description

The inet_addr() function shall convert the string pointed to by cp, in the standard IPv4 dotted decimal notation, to an integer value suitable for use as an Internet address.

All Internet addresses shall be returned in network order (bytes ordered from left to right).

Values specified using IPv4 dotted decimal notation take one of the following forms:

a.b.c.d
When four parts are specified, each shall be interpreted as a byte of data and assigned, from left to right, to the four bytes of an Internet address.

a.b.c
When a three-part address is specified, the last part shall be interpreted as a 16-bit quantity and placed in the rightmost two bytes of the network address. This makes the three-part address format convenient for specifying Class B network addresses as "128.net.host" .

a.b
When a two-part address is supplied, the last part shall be interpreted as a 24-bit quantity and placed in the rightmost three bytes of the network address. This makes the two-part address format convenient for specifying Class A network addresses as "net.host" .

a
When only one part is given, the value shall be stored directly in the network address without any byte rearrangement.

All numbers supplied as parts in IPv4 dotted decimal notation may be decimal, octal, or hexadecimal, as specified in the ISO C standard (that is, a leading 0x or 0X implies hexadecimal; otherwise, a leading '0' implies octal; otherwise, the number is interpreted as decimal).

Return Parameter

Upon successful completion, inet_addr() shall return the Internet address. Otherwise, it shall return (in_addr_t)(-1).

Error Code

None.

Additional Notes

The inet_ntoa() function in the standard C library is non-thread-safe and thus is not provided in T2EX.
inet_ntop() function substitutes this function.

### 8.3.3 inet_ntop, inet_pton - convert IPv4 addresses between binary and text form

C Language Interface

```
#include <arpa/inet.h>

const char      *inet_ntop(int af, const void *src, char *dst, socklen_t size);
int             inet_pton(int af, const char *src, void *dst);
```

Description

The inet_ntop() function shall convert a numeric address into a text string suitable for presentation.

The af argument shall specify the family of the address. This can be AF_INET.
The src argument points to a buffer holding an IPv4 address if the af argument is AF_INET; the address must be in network byte order.
The dst argument points to a buffer where the function stores the resulting text string; it shall not be NULL.
The size argument specifies the size of this buffer, which shall be large enough to hold the text string (INET_ADDRSTRLEN characters for IPv4).

The inet_pton() function shall convert an address in its standard text presentation form into its numeric binary form.

The af argument shall specify the family of the address. The AF_INET address families shall be supported.
The src argument points to the string being passed in.
The dst argument points to a buffer into which the function stores the numeric address; this shall be large enough to hold the numeric address (32 bits for AF_INET).

If the af argument of inet_pton() is AF_INET, the src string shall be in the standard IPv4 dotted-decimal form:

ddd.ddd.ddd.ddd

where "ddd" is a one to three digit decimal number between 0 and 255 (see inet_addr ). The inet_pton() function does not accept other formats (such as the octal numbers, hexadecimal numbers, and fewer than four numbers that inet_addr() accepts).

Return Parameter

The inet_ntop() function shall return a pointer to the buffer containing the text string if the conversion succeeds, and NULL otherwise.

The inet_pton() function shall return 1 if the conversion succeeds, with the address pointed to by dst in network byte order. It shall return 0 if the input is not a valid IPv4 dotted-decimal string or -1 if the af argument is unknown.

Error Code

None.

## 8.4 assert.h

The header assert.h defines the following test macro.

Macro

assert()
The <assert.h> header shall define the assert() macro. It refers to the macro NDEBUG which is not defined in the header.
If NDEBUG is defined as a macro name before the inclusion of this header, the assert() macro shall be defined simply as:

```
#define assert(ignore)((void) 0)
```

Otherwise, the macro behaves as described in assert() shown below.

The assert() macro shall be redefined according to the current state of NDEBUG each time <assert.h> is included.

The assert() macro shall be implemented as a macro, not as a function.

### 8.4.1 assert - insert program diagnostics

C Language Interface

```
#include <assert.h>

void    assert(scalar expression);
```

Description

The assert() macro shall insert diagnostics into programs; it shall expand to a void expression.

When it is executed, if expression (which shall have a scalar type) is false (that is, compares equal to 0), assert() shall write information about the particular call that failed on stderr and shall call abort().

The information written about the call that failed shall include the text of the argument, the name of the source file, the source file line number, and the name of the enclosing function; the latter are, respectively, the values of the preprocessing macros __FILE__ and __LINE__ and of the identifier __func__.

Forcing a definition of the name NDEBUG, either from the compiler command line or with the preprocessor control statement #define NDEBUG ahead of the #include <assert.h> statement, shall stop assertions from being compiled into the program.

Return Parameter

None.

## 8.5 complex.h

The header complex.h defines the following complex-related macros and function prototype declarations.

Macros

complex
: Expands to _Complex.

_Complex_I
: Expands to a constant expression of type const float _Complex, with the value of the imaginary unit (that is, a number i such that i^2 = -1) ('^' represents the power).

imaginary
: Expands to _Imaginary.

_Imaginary_I
: Expands to a constant expression of type const float _Imaginary with the value of the imaginary unit.

I
: Expands to either _Imaginary_I or _Complex_I. If _Imaginary_I is not defined, I expands to _Complex_I.

The macros imaginary and _Imaginary_I shall be defined if and only if the implementation supports imaginary types.
In the T2EX reference implementation, the imaginary type is not supported and thus these are not defined.

Functions

### 8.5.1 cabs, cabsf, cabsl - return a complex absolute value

C Language Interface

```
#include <complex.h>

double                  cabs(double complex z);
float                   cabsf(float complex z);
long double             cabsl(long double complex z);
```

Description

These functions shall compute the complex absolute value (also called norm, modulus, or magnitude) of z.

Return Parameter

These functions shall return the complex absolute value.

Error Code

None.

### 8.5.2 cacos, cacosf, cacosl - complex arc cosine functions

C Language Interface

```
#include <complex.h>
```

```
double complex          cacos(double complex z);
float complex           cacosf(float complex z);
long double complex     cacosl(long double complex z);
```

Description

These functions shall compute the complex arc cosine of z, with branch cuts outside the interval [-1, +1] along the real axis.

Return Parameter

These functions shall return the complex arc cosine value, in the range of a strip mathematically unbounded along the imaginary axis and in the interval [0,+ π] along the real axis.

Error Code

None.

See Also

ccos

### 8.5.3 cacosh, cacoshf, cacoshl - complex arc hyperbolic cosine functions

C Language Interface

```
#include <complex.h>

double complex          cacosh(double complex z);
float complex           cacoshf(float complex z);
long double complex     cacoshl(long double complex z);
```

Description

These functions shall compute the complex arc hyperbolic cosine of z, with a branch cut at values less than 1 along the real axis.

Return Parameter

These functions shall return the complex arc hyperbolic cosine value, in the range of a half-strip of non-negative values along the real axis and in the interval [-i*π, +i*π] along the imaginary axis.

Error Code

None.

See Also

ccosh

### 8.5.4 carg, cargf, cargl - complex argument functions

C Language Interface

```
#include <complex.h>

double                  carg(double complex z);
float                   cargf(float complex z);
long double             cargl(long double complex z);
```

Description

These functions shall compute the argument (also called phase angle) of z, with a branch cut along the negative real axis.

Return Parameter

These functions shall return the value of the argument in the interval [-π, +π].

Error Code

None.

See Also

cimag, conj, cproj

### 8.5.5 casin, casinf, casinl - complex arc sine functions

C Language Interface

```
#include <complex.h>

double complex          casin(double complex z);
float complex           casinf(float complex z);
long double complex     casinl(long double complex z);
```

Description

These functions shall compute the complex arc sine of z, with branch cuts outside the interval [-1, +1] along the real axis.

Return Parameter

These functions shall return the complex arc sine value, in the range of a strip mathematically unbounded along the imaginary axis and in the interval [-π/2, +π/2] along the real axis.

Error Code

None.

See Also

csin

### 8.5.6 casinh, casinhf, casinhl - complex arc hyperbolic sine functions

C Language Interface

```
#include <complex.h>

double complex          casinh(double complex z);
float complex           casinhf(float complex z);
long double complex     casinhl(long double complex z);
```

Description

These functions shall compute the complex arc hyperbolic sine of z, with branch cuts outside the interval [-i, +i] along the imaginary axis.

Return Parameter

These functions shall return the complex arc hyperbolic sine value, in the range of a strip mathematically unbounded along the real axis and in the interval [-i*π/2, +i*π/2] along the imaginary axis.

Error Code

None.

See Also

csinh

### 8.5.7 catan, catanf, catanl - complex arc tangent functions

C Language Interface

```
#include <complex.h>

double complex          catan(double complex z);
float complex           catanf(float complex z);
long double complex     catanl(long double complex z);
```

Description

These functions shall compute the complex arc tangent of z, with branch cuts outside the interval [-i, +i] along the imaginary axis.

Return Parameter

These functions shall return the complex arc tangent value, in the range of a strip mathematically unbounded along the imaginary axis and in the interval [-π/2, +π/2] along the real axis.

Error Code

None.

See Also

ctan

### 8.5.8 catanh, catanhf, catanhl - complex arc hyperbolic tangent functions

C Language Interface

```
#include <complex.h>

double complex          catanh(double complex z);
float complex           catanhf(float complex z);
long double complex     catanhl(long double complex z);
```

Description

These functions shall compute the complex arc hyperbolic tangent of z, with branch cuts outside the interval [-1, +1] along the real axis.

Return Parameter

These functions shall return the complex arc hyperbolic tangent value, in the range of a strip mathematically unbounded along the real axis and in the interval [-i*π/2, +i*π/2] along the imaginary axis.

Error Code

None.

See Also

ctanh

### 8.5.9 ccos, ccosf, ccosl - complex cosine functions

C Language Interface

```
#include <complex.h>

double complex          ccos(double complex z);
float complex           ccosf(float complex z);
long double complex     ccosl(long double complex z);
```

Description

These functions shall compute the complex cosine of z.

Return Parameter

These functions shall return the complex cosine value.

Error Code

None.

See Also

cacos

### 8.5.10 ccosh, cconshf, cconshl - complex hyperbolic cosine functions

C Language Interface

```
#include <complex.h>

double complex          ccosh(double complex z);
float complex           ccoshf(float complex z);
long double complex     ccoshl(long double complex z);
```

Description

These functions shall compute the complex hyperbolic cosine of z.

Return Parameter

hese functions shall return the complex hyperbolic cosine value.

Error Code

None.

See Also

cacosh

### 8.5.11 cexp, cexpf, cexpl - complex exponential functions

C Language Interface

```
#include <complex.h>

double complex          cexp(double complex z);
float complex           cexpf(float complex z);
long double complex     cexpl(long double complex z);
```

Description

These functions shall compute the complex exponent of z, defined as e^z ('^' represents the power).

Return Parameter

These functions shall return the complex exponential value of z.

Error Code

None.

See Also

clog

### 8.5.12 cimag, cimagf, cimagl - complex imaginary functions

C Language Interface

```
#include <complex.h>

double                  cimag(double complex z);
float                   cimagf(float complex z);
long double             cimagl(long double complex z);
```

Description

These functions shall compute the imaginary part of z.

Return Parameter

These functions shall return the imaginary part value (as a real).

Error Code

None.

See Also

carg, conj, cproj, creal

### 8.5.13 clog, clogf, clogl - complex natural logarithm functions

C Language Interface

```
#include <complex.h>

double complex          clog(double complex z);
float complex           clogf(float complex z);
long double complex     clogl(long double complex z);
```

Description

These functions shall compute the complex natural (base-e) logarithm of z, with a branch cut along the negative real axis.

Return Parameter

These functions shall return the complex natural logarithm value, in the range of a strip mathematically unbounded along the real axis and in the interval [-i*π, +i*π] along the imaginary axis.

Error Code

None.

See Also

cexp

### 8.5.14 conj, conjf, conjl - complex conjugate functions

C Language Interface

```
#include <complex.h>

double complex          conj(double complex z);
float complex           conjf(float complex z);
long double complex     conjl(long double complex z);
```

Description

These functions shall compute the complex conjugate of z, by reversing the sign of its imaginary part.

Return Parameter

These functions return the complex conjugate value.

Error Code

None.

See Also

carg, cimag, cproj, creal

### 8.5.15 cpow, cpowf, cpowl - complex power functions

C Language Interface

```
#include <complex.h>

double complex          cpow(double complex x, double complex y);
float complex           cpowf(float complex x, float complex y);
long double complex     cpowl(long double complex x, long double complex y);
```

Description

These functions shall compute the complex power function xˆy ('ˆ' represents the power), with a branch cut for the first parameter along the negative real axis.

Return Parameter

These functions shall return the complex power function value.

Error Code

None.

See Also

cabs, csqrt

### 8.5.16 cproj, cprojf, cprojl - complex projection functions

C Language Interface

```
#include <complex.h>

double complex          cproj(double complex z);
float complex           cprojf(float complex z);
long double complex     cprojl(long double complex z);
```

Description

These functions shall compute a projection of z onto the Riemann sphere: z projects to z, except that all complex infinities (even those with one infinite part and one NaN part) project to positive infinity on the real axis.
If z has an infinite part, then cproj(z) shall be equivalent to:

```
        INFINITY + I * copysign(0.0, cimag(z))
```

Return Parameter

These functions shall return the value of the projection onto the Riemann sphere.

Error Code

None.

See Also

carg, cimag, conj, creal

### 8.5.17 creal, crealf, creall - complex real functions

C Language Interface

```
#include <complex.h>

double                  creal(double complex z);
float                   crealf(float complex z);
long double             creall(long double complex z);
```

Description

These functions shall compute the real part of z.

Return Parameter

These functions shall return the real part value.

Error Code

None.

See Also

carg, cimag, conj, cproj

### 8.5.18 csin, csinf, csinl - complex sine functions

C Language Interface

```
#include <complex.h>

double complex          csin(double complex z);
float complex           csinf(float complex z);
long double complex     csinl(long double complex z);
```

Description

These functions shall compute the complex sine of z.

Return Parameter

These functions shall return the complex sine value.

Error Code

None.

See Also

casin

### 8.5.19 csinh, csinhf, csinhl - complex hyperbolic sine functions

C Language Interface

```
#include <complex.h>

double complex          csinh(double complex z);
float complex           csinhf(float complex z);
long double complex     csinhl(long double complex z);
```

Description

These functions shall compute the complex hyperbolic sine of z.

Return Parameter

These functions shall return the complex hyperbolic sine value.

Error Code

None.

See Also

casinh

### 8.5.20 csqrt, csqrtf, csqrtl - complex square root functions

C Language Interface

```
#include <complex.h>

double complex          csqrt(double complex z);
float complex           csqrtf(float complex z);
long double complex     csqrtl(long double complex z);
```

Description

These functions shall compute the complex square root of z, with a branch cut along the negative real axis.

Return Parameter

These functions shall return the complex square root value, in the range of the right half-plane (including the imaginary axis).

Error Code

None.

See Also

cabs, cpow

### 8.5.21 ctan, ctanf, ctanl - complex tangent functions

C Language Interface

```
#include <complex.h>

double complex          ctan(double complex z);
float complex           ctanf(float complex z);
long double complex     ctanl(long double complex z);
```

Description

These functions shall compute the complex tangent of z.

Return Parameter

These functions shall return the complex tangent value.

Error Code

None.

See Also

catan

8.5.22 ctanh, ctanhf, ctanhl - complex hyperbolic tangent functions

C Language Interface

```
#include <complex.h>

complex                 ctanh(double complex z);
float complex           ctanhf(float complex z);
long double complex     ctanhl(long double complex z);
```

Description

These functions shall compute the complex hyperbolic tangent of z.

Return Parameter

These functions shall return the complex hyperbolic tangent value.

Error Code

None.

See Also

catanh

8.6 ctype.h

The header ctype.h defines the following character type determination functions and macros.

Functions or macros

```
int   isalnum(int c);
                shall return non-zero if c is an alphanumeric character.

int   isalpha(int c);
                shall return non-zero if c is an alphabetic character.

int   isascii(int c);
                shall return non-zero if c is a 7-bit US-ASCII character code between 0 and
                octal 0177 inclusive.

int   isblank(int c);
                shall return non-zero if c is a blank (space or tab).

int   iscntrl(int c);
                shall return non-zero if c is a control character.

int   isdigit(int c);
                shall return non-zero if c is a decimal digit.

int   isgraph(int c);
                shall return non-zero if c is a character with a visible representation.

int   islower(int c);
                shall return non-zero if c is a lowercase letter.

int   isprint(int c);
                shall return non-zero if c is a printable character.

int   ispunct(int c);
                shall return non-zero if c is a punctuation character.

int   isspace(int c);
                shall return non-zero if c is a white-space character.

int   isupper(int c);
                shall return non-zero if c is an uppercase letter.

int   isxdigit(int c);
                shall return non-zero if c is a hexadecimal digit.
```

In the above function or macro, the c argument is an int, the value of which the application shall ensure is representable as an unsigned char or equal to the value of the macro EOF.
If the argument has any other value, the behavior is undefined.

```
int   toascii(int c);
                shall return the value (c &0x7f).

int   tolower(int c);
                shall return the lowercase letter corresponding to the argument passed;
                otherwise, they shall return the argument unchanged.

int   toupper(int c);
                shall return the uppercase letter corresponding to the argument passed;
                otherwise, they shall return the argument unchanged.
```

The followings are required to be defined as macros.

```
int   _toupper(int c);
                shall return the uppercase letter corresponding to the argument passed.

int   _tolower(int c);
                shall return the lowercase letter corresponding to the argument passed.
```

The character type is determined based on the system locale.

## 8.7 dirent.h

The header dirent.h defines the following structure and function prototype declaration to read the directory entry in the same way as a stream.

### Types

DIR type
	The type that expresses the directory stream.
	In the T2EX reference implementation, the DIR type is a structure including file descriptors.

struct "dirent" structure
	This structure represents a directory entry and includes the following elements.

```
	ino_t   d_ino                           file serial number
	char    d_name[NAME_MAX+1]              Entry name
```

ino_t type
	File serial number or file identification number in the file system.

### Functions

The readdir() in the standard C library is non-thread-safe and thus is not provided in T2EX.
The readdir_r() is used instead.

### 8.7.1 closedir_eno, closedir - Closes the directory stream

C Language Interface

```
#include <dirent.h>

int     closedir_eno(DIR *dirp, errno_t* enop);
int     closedir(DIR *dirp);
```

Description

The closedir_eno() closes the directory stream referred to by dirp.
In case of an error, its error number is stored in the area pointed to by enop.
If enop is NULL, the error number is not stored.
At completion, the value of dirp no longer points to the accessible DIR type object.
For a DIR type implementation using a file descriptor, the file descriptor is closed.

The closedir() is equivalent to closedir_eno(dirp, NULL).

Return Parameter

If successful, these functions return 0.
In case of an error, these functions return -1.

Error Code

In case of an error, the followings are stored in the area pointed to by enop.

```
EBADF           The dirp does not refer to the opened directory stream
EINTR           Aborted by fs_break()
```

See Also

opendir, opendir_eno, readdir_r

### 8.7.2 opendir_eno, opendir - Opens the directory stream

C Language Interface

```
#include <dirent.h>

DIR     *opendir_eno(const char *path, errno_t* enop);
DIR     *opendir(const char *path);
```

Description

The opendir_eno() opens the directory specified by path and returns the DIR type directory stream.
The opened directory stream points the first entry.
If the DIR type is implemented using a file descriptor, directories cannot be opened beyond the number of file descriptors that can be opened by the system.
The file descriptor used in the DIR type must be obtained by specifying O_DIRECTORY as the flag of fs_open().
In case of an error, the error number is stored in the area pointed to by enop.
If enop is NULL, the error number is not stored.

The opendir(path) is equivalent to opendir_eno(path, NULL).

Return Parameter

If successful, opendir() and opendir_eno() return the pointer to the DIR type.
In case of an error, these functions return NULL.

Error Code

In case of an error, the followings are stored in the area pointed to by enop.

| | |
|---|---|
| EACCES | Read permission attribute does not exist for a directory in "path" |
| ENAMETOOLONG | File name is too long<br>- The directory or file name part in "path" is too long (NAME_MAX at maximum).<br>- Whole "path" length is too long (PATH_MAX at maximum). |
| ENOENT | "path" does not exist |
| ENOTDIR | "path" is not a directory |
| ENFILE | The limit of number of file descriptors being opened in the system is exceeded the limit |

See Also

closedir, closedir_eno, readdir_r

### 8.7.3 readdir_r - Reads directory entries

C Language Interface

```
#include <dirent.h>

errno_t readdir_r(DIR *dirp, struct dirent *entry, struct dirent **result);
```

Description

The readdir_r() reads the current directory entry of the directory stream specified by dirp.
Then, it initializes the struct dirent type data indicated by entry with the read content, stores the pointer to this data in the location pointed to by result, and moves the directory stream to the next position.

The area pointed to by entry must be large enough to be able to store the dirent type whose element "char d_name[]" size is over NAME_MAX+1 character.

If successful, the value of *result is same as the one of entry.
If the end of the directory stream is reached, *result becomes NULL.

The readdir_r() can buffer several directory entries for a single, actual reading processing.

Return Parameter

If successful, readdir_r() returns 0.
Otherwise, it returns an error number.

Error Code

| | |
|---|---|
| EOVERFLOW | An unrepresentable value has been generated in one of structures that set the value |
| EBADF | The dirp does not refer to the opened directory stream |
| ENOENT | The current directory stream position is illegal |

See Also

opendir_eno, closedir_eno

### 8.7.4 rewinddir - Resets the directory stream position to the beginning

C Language Interface

```
#include <dirent.h>

void    rewinddir(DIR *dirp);
```

Description

The rewindir() resets the position of the directory stream referred to by dirp to the beginning of the directory.
If dirp does not point to any directory stream, its action shall be undefined.

Return Parameter

The rewindir() does not return a value.

Error Code

None

See Also

seekdir, telldir

### 8.7.5 seekdir - Move the directory stream position

C Language Interface

```
#include <dirent.h>

void    seekdir(DIR *dirp, long loc);
```

Description

Regarding the directory stream specified by dirp, the seekdir() sets the next readdir_r() position to the position specified by loc.
The loc value must be the one that was previously returned as a result of calling telldir().
The new position is the one when telldir() was previously executed.

Return Parameter

The seekdir() does not return a value.

Error Code

None

See Also

readdir, rewinddir, telldir

8.7.6 telldir - The current directory stream position

C Language Interface

```
#include <dirent.h>

long    telldir(DIR *dirp);
```

Description

The telldir() returns the current position of the directory stream specified by dirp.
The behavior is undefined if dirp is illegal.

Return Parameter

The telldir() returns the current position of the directory stream specified.

Error Code

None

See Also

seekdir, readdir, rewinddir

## 8.8 errno.h

The header errno.h defines the following macros that relate to types for error handling and error numbers.
For the error number, see Section 8.2.

### Type

```
typedef int errno_t;
```
Error number type

### Macros

```
#define EC_ERRNO        (-73)
```
T2EX main error code corresponding to the error number

```
#define ERRNOtoER(eno)  (ERCD(EC_ERRNO, (eno)))
```
Macro for converting an error number into the T2EX extended error code

```
#define ERRNO(er)       (MERCD(er) == EC_ERRNO ? SERCD(er) : 0)
```
Macro for extracting the error number from the T2EX error code

### Error Numbers

The following shows the list of error numbers and their meanings.
For details of the meaning of the errors, see the description of each function that returns that error number.

| | |
|---|---|
| EPERM | Operation not permitted. |
| ENOENT | No such file or directory. |
| ESRCH | No such process.* 1 |
| EINTR | Interrupted function. |
| EIO | I/O error. |
| ENXIO | No such device or address. |
| E2BIG | Argument list too long. |
| ENOEXEC | Executable file format error. |
| EBADF | Bad file descriptor. |
| ECHILD | No child processes.* 1 |
| EAGAIN | Resource unavailable, try again. (may be the same value as EWOULDBLOCK.) |
| EDEADLK | Resource deadlock would occur. |
| ENOMEM | Not enough space. |
| EACCES | Permission denied. |
| EFAULT | Bad address. |
| EBUSY | Device or resource busy. |
| EEXIST | File exists. |
| EXDEV | Cross-device link. |
| ENODEV | No such device. |
| ENOTDIR | Not a directory. |
| EISDIR | Is a directory. |
| EINVAL | Invalid argument. |
| ENFILE | Too many files open in system. |
| EMFILE | File descriptor value too large.* 1 |
| ENOTTY | Inappropriate I/O control operation. |
| EFBIG | File too large. |
| ENOSPC | No space left on device. |
| ESPIPE | Invalid seek. |
| EROFS | Read-only file system. |
| EMLINK | Too many links.* 1 |
| EPIPE | Broken pipe.* 1 |
| EDOM | Mathematics argument out of domain of function. |
| ERANGE | Result too large. |
| EWOULDBLOCK | Operation would block. (may be the same value as EAGAIN.) |
| EINPROGRESS | Operation in progress. |
| EALREADY | Connection already in progress. |
| ENOTSOCK | Not a socket. |
| EDESTADDRREQ | Destination address required. |
| EMSGSIZE | Message too large. |
| EPROTOTYPE | Protocol wrong type for socket. |
| ENOPROTOOPT | Protocol not available. |
| EPROTONOSUPPORT | Unsupported protocol. |
| ESOCKTNOSUPPORT | Unsupported socket type. |
| EOPNOTSUPP | Operation unsupported on socket. |

| | |
|---|---|
| EPFNOSUPPORT | Unsupported protocol family. |
| EAFNOSUPPORT | Address family unsupported by protocol. |
| EADDRINUSE | Address in use. |
| EADDRNOTAVAIL | Address not available. |
| ENETDOWN | Network is down. |
| ENETUNREACH | Network unreachable. |
| ENETRESET | Connection aborted by network. |
| ECONNABORTED | Connection aborted. (The problem of the local host side.) |
| ECONNRESET | Connection reset. (Reset by peer.) |
| ENOBUFS | No buffer space available. |
| EISCONN | Socket is connected. |
| ENOTCONN | The socket is not connected. |
| ESHUTDOWN | Cannot send after transport endpoint shutdown. |
| ETIMEDOUT | Connection timed out. |
| ECONNREFUSED | Connection refused. (Rejected by peer.) |
| ELOOP | Too many levels of symbolic links.* This does not occur. |
| ENAMETOOLONG | Filename too long. |
| EHOSTDOWN | Remote host is down. |
| EHOSTUNREACH | Host is unreachable. |
| ENOTEMPTY | Directory not empty. |
| EDQUOT | Disk quota exceeded.* 1 |
| ENOLCK | No locks available.* 1 |
| ENOSYS | Unsupported function. |
| EOVERFLOW | Value too large to be stored in data type. |
| EFTYPE | Inappropriate file type or format. |
| EILSEQ | Illegal byte sequence. |
| ENOTSUP | Unsupported. |

* 1 For the compatibility with POSIX, macro names for error numbers are defined for errors that do not actually occur in the T2EX reference implementation.
(If a custom file system implementation part is implemented, approrpriate error macro names can be selected among them.)

The following error numbers are only used by API calls of the network communication functions.

| | |
|---|---|
| EAI_AGAIN | The name could not be resolved at this time. |
| EAI_BADFLAGS | ai_flags had an invalid value. |
| EAI_FAIL | A non-recoverable error occurred when attempting to resolve the name. |
| EAI_FAMILY | Invalid address family. |
| EAI_MEMORY | Memory allocation failure. |
| EAI_NODATA | No address is associated with the specified host name. |
| EAI_NONAME | Neither a host name nor a service name are supplied. Or the name cannot be resolved. |
| EAI_SERVICE | The service passed is not recognized for the specified socket type. |
| EAI_SOCKTYPE | The intended socket type is not recognized. |
| EAI_SYSTEM | An internal error is occurred. |
| EAI_BADHINTS | Invalid value for hints |
| EAI_OVERFLOW | Argument buffer overflow. |

The extended error codes (EX_...) corresponding to the POSIX error numbers are defined as follows.

```
#define EX_PERM                 ERCD(EC_ERRNO, EPERM)
#define EX_NOENT                ERCD(EC_ERRNO, ENOENT)
#define EX_SRCH                 ERCD(EC_ERRNO, ESRCH)
#define EX_INTR                 ERCD(EC_ERRNO, EINTR)
#define EX_IO                   ERCD(EC_ERRNO, EIO)
#define EX_NXIO                 ERCD(EC_ERRNO, ENXIO)
#define EX_2BIG                 ERCD(EC_ERRNO, E2BIG)
#define EX_NOEXEC               ERCD(EC_ERRNO, ENOEXEC)
#define EX_BADF                 ERCD(EC_ERRNO, EBADF)
#define EX_CHILD                ERCD(EC_ERRNO, ECHILD)
#define EX_AGAIN                ERCD(EC_ERRNO, EAGAIN)
#define EX_DEADLK               ERCD(EC_ERRNO, EDEADLK)
#define EX_NOMEM                ERCD(EC_ERRNO, ENOMEM)
#define EX_ACCES                ERCD(EC_ERRNO, EACCES)
#define EX_FAULT                ERCD(EC_ERRNO, EFAULT)
#define EX_BUSY                 ERCD(EC_ERRNO, EBUSY)
#define EX_EXIST                ERCD(EC_ERRNO, EEXIST)
#define EX_XDEV                 ERCD(EC_ERRNO, EXDEV)
#define EX_NODEV                ERCD(EC_ERRNO, ENODEV)
#define EX_NOTDIR               ERCD(EC_ERRNO, ENOTDIR)
#define EX_ISDIR                ERCD(EC_ERRNO, EISDIR)
#define EX_INVAL                ERCD(EC_ERRNO, EINVAL)
```

```
#define EX_NFILE                ERCD(EC_ERRNO, ENFILE)
#define EX_MFILE                ERCD(EC_ERRNO, EMFILE)
#define EX_NOTTY                ERCD(EC_ERRNO, ENOTTY)
#define EX_FBIG                 ERCD(EC_ERRNO, EFBIG)
#define EX_NOSPC                ERCD(EC_ERRNO, ENOSPC)
#define EX_SPIPE                ERCD(EC_ERRNO, ESPIPE)
#define EX_ROFS                 ERCD(EC_ERRNO, EROFS)
#define EX_MLINK                ERCD(EC_ERRNO, EMLINK)
#define EX_PIPE                 ERCD(EC_ERRNO, EPIPE)
#define EX_DOM                  ERCD(EC_ERRNO, EDOM)
#define EX_RANGE                ERCD(EC_ERRNO, ERANGE)
#define EX_WOULDBLOCK           ERCD(EC_ERRNO, EWOULDBLOCK)
#define EX_INPROGRESS           ERCD(EC_ERRNO, EINPROGRESS)
#define EX_ALREADY              ERCD(EC_ERRNO, EALREADY)
#define EX_NOTSOCK              ERCD(EC_ERRNO, ENOTSOCK)
#define EX_DESTADDRREQ          ERCD(EC_ERRNO, EDESTADDRREQ)
#define EX_MSGSIZE              ERCD(EC_ERRNO, EMSGSIZE)
#define EX_PROTOTYPE            ERCD(EC_ERRNO, EPROTOTYPE)
#define EX_NOPROTOOPT           ERCD(EC_ERRNO, ENOPROTOOPT)
#define EX_PROTONOSUPPORT       ERCD(EC_ERRNO, EPROTONOSUPPORT)
#define EX_SOCKTNOSUPPORT       ERCD(EC_ERRNO, ESOCKTNOSUPPORT)
#define EX_OPNOTSUPP            ERCD(EC_ERRNO, EOPNOTSUPP)
#define EX_PFNOSUPPORT          ERCD(EC_ERRNO, EPFNOSUPPORT)
#define EX_AFNOSUPPORT          ERCD(EC_ERRNO, EAFNOSUPPORT)
#define EX_ADDRINUSE            ERCD(EC_ERRNO, EADDRINUSE)
#define EX_ADDRNOTAVAIL         ERCD(EC_ERRNO, EADDRNOTAVAIL)
#define EX_NETDOWN              ERCD(EC_ERRNO, ENETDOWN)
#define EX_NETUNREACH           ERCD(EC_ERRNO, ENETUNREACH)
#define EX_NETRESET             ERCD(EC_ERRNO, ENETRESET)
#define EX_CONNABORTED          ERCD(EC_ERRNO, ECONNABORTED)
#define EX_CONNRESET            ERCD(EC_ERRNO, ECONNRESET)
#define EX_NOBUFS               ERCD(EC_ERRNO, ENOBUFS)
#define EX_ISCONN               ERCD(EC_ERRNO, EISCONN)
#define EX_NOTCONN              ERCD(EC_ERRNO, ENOTCONN)
#define EX_SHUTDOWN             ERCD(EC_ERRNO, ESHUTDOWN)
#define EX_TIMEDOUT             ERCD(EC_ERRNO, ETIMEDOUT)
#define EX_CONNREFUSED          ERCD(EC_ERRNO, ECONNREFUSED)
#define EX_LOOP                 ERCD(EC_ERRNO, ELOOP)
#define EX_NAMETOOLONG          ERCD(EC_ERRNO, ENAMETOOLONG)
#define EX_HOSTDOWN             ERCD(EC_ERRNO, EHOSTDOWN)
#define EX_HOSTUNREACH          ERCD(EC_ERRNO, EHOSTUNREACH)
#define EX_NOTEMPTY             ERCD(EC_ERRNO, ENOTEMPTY)
#define EX_DQUOT                ERCD(EC_ERRNO, EDQUOT)
#define EX_NOLCK                ERCD(EC_ERRNO, ENOLCK)
#define EX_OVERFLOW             ERCD(EC_ERRNO, EOVERFLOW)
#define EX_NOSYS                ERCD(EC_ERRNO, ENOSYS)
#define EX_FTYPE                ERCD(EC_ERRNO, EFTYPE)
#define EX_ILSEQ                ERCD(EC_ERRNO, EILSEQ)
#define EX_NOTSUP               ERCD(EC_ERRNO, ENOTSUP)
#define EX_AI_ADDRFAMILY        ERCD(EC_ERRNO, EAI_ADDRFAMILY)
#define EX_AI_AGAIN             ERCD(EC_ERRNO, EAI_AGAIN)
#define EX_AI_BADFLAGS          ERCD(EC_ERRNO, EAI_BADFLAGS)
#define EX_AI_FAIL              ERCD(EC_ERRNO, EAI_FAIL)
#define EX_AI_FAMILY            ERCD(EC_ERRNO, EAI_FAMILY)
#define EX_AI_MEMORY            ERCD(EC_ERRNO, EAI_MEMORY)
#define EX_AI_NODATA            ERCD(EC_ERRNO, EAI_NODATA)
#define EX_AI_NONAME            ERCD(EC_ERRNO, EAI_NONAME)
#define EX_AI_SERVICE           ERCD(EC_ERRNO, EAI_SERVICE)
#define EX_AI_SOCKTYPE          ERCD(EC_ERRNO, EAI_SOCKTYPE)
#define EX_AI_SYSTEM            ERCD(EC_ERRNO, EAI_SYSTEM)
#define EX_AI_BADHINTS          ERCD(EC_ERRNO, EAI_BADHINTS)
#define EX_AI_PROTOCOL          ERCD(EC_ERRNO, EAI_PROTOCOL)
#define EX_AI_OVERFLOW          ERCD(EC_ERRNO, EAI_OVERFLOW)
```

8.9 float.h

The header float.h defines the following macros for the attributes and limit values relating to the floating point number.

Macros

FLT_ROUNDS
The rounding mode for floating-point addition is characterized by the implementation-defined value of FLT_ROUNDS:

| | |
|---|---|
| -1 | Indeterminable. |
| 0 | Toward zero. |
| 1 | To nearest. |
| 2 | Toward positive infinity. |
| 3 | Toward negative infinity. |

All other values for FLT_ROUNDS characterize implementation-defined rounding behavior.
FLT_ROUNDS may be a non-constant.
In the T2EX reference implementation, FLT_ROUNDS is 1.

FLT_EVAL_METHOD
The values of operations with floating operands and values subject to the usual arithmetic conversions and of floating constants are evaluated to a format whose range and precision may be greater than required by the type.
The use of evaluation formats is characterized by the implementation-defined value of FLT_EVAL_METHOD:

| | |
|---|---|
| -1 | Indeterminable. |
| 0 | Evaluate all operations and constants just to the range and precision of the type. |
| 1 | Evaluate operations and constants of type float and double to the range and precision of the double type; evaluate long double operations and constants to the range and precision of the long double type. |
| 2 | Evaluate all operations and constants to the range and precision of the long double type. |

All other values for FLT_EVAL_METHOD characterize implementation-defined behavior.
In the T2EX reference implementation, FLT_EVAL_METHOD is 0.

The following symbols are constant expressions with implementation-defined values that are greater or equal in magnitude (absolute value) to those shown on right, with the same sign, if the value on the right is defined.

FLT_RADIX 2
Radix of exponent representation.
In the T2EX reference implementation, FLT_RADIX is 2.

FLT_MANT_DIG
DBL_MANT_DIG
LDBL_MANT_DIG
Number of base-FLT_RADIX digits in the floating-point significand.
In the T2EX reference implementation, FLT_MANT_DIG is 24 and both DBL_MANT_DIG and LDBL_MANT_DIG are 53.

DECIMAL_DIG 10
Number of decimal digits, n, such that any floating-point number in the widest supported floating type can be rounded to a floating-point number with n decimal digits and back again without change to the value.
This is 17 in the T2EX reference implementation.

FLT_DIG 6
DBL_DIG 10
LDBL_DIG 10
The decimal numbers consisting of these numbers of digits can be rounded to float, double, and long double respectively, and will maintain the original value when converted back.
In the T2EX reference implementation, FLT_DIG is 6 and both DBL_DIG and LDBL_DIG are 15.

FLT_MIN_EXP
DBL_MIN_EXP

LDBL_MIN_EXP
Minimum negative integer such that FLT_RADIX raised to that power minus 1 is a normalized floating-point number.
In the T2EX reference implementation, FLT_MIN_EXP is -125 and both DBL_MIN_EXP and LDBL_MIN_EXP are -1021.

FLT_MIN_10_EXP -37
DBL_MIN_10_EXP -37
LDBL_MIN_10_EXP -37
Minimum negative integer such that 10 raised to that power is in the range of normalized floating-point numbers.
In the T2EX reference implementation, FLT_MIN_10_EXP is -37 and both DBL_MIN_10_EXP and LDBL_MIN_10_EXP are -307.

FLT_MAX_EXP
DBL_MAX_EXP
LDBL_MAX_EXP
Maximum integer such that FLT_RADIX raised to that power minus 1 is a representable finite floating-point number.
In the T2EX reference implementation, FLT_MAX_EXP is 128 and both DBL_MAX_EXP and LDBL_MAX_EXP are 1024.

FLT_MAX_10_EXP +37
DBL_MAX_10_EXP +37
LDBL_MAX_10_EXP +37
Maximum integer such that 10 raised to that power is in the range of representable finite floating-point numbers.
In the T2EX reference implementation, FLT_MAX_10_EXP is 38 and both DBL_MAX_10_EXP and LDBL_MAX_10_EXP are 308.

The following symbols are constant expressions with implementation-defined values that are greater or equal to those shown on right.

FLT_MAX 1E+37
DBL_MAX 1E+37
LDBL_MAX 1E+37
Maximum representable finite floating-point number.
These take the following values in the T2EX reference implementation.
FLT_MAX 3.40282347e+38F
DBL_MAX 1.7976931348623157e+308
LDB_MAX 1.7976931348623157e+308L

The following symbols are constant expressions with implementation-defined values that are less or equal to those shown on right.

FLT_MIN 1E-37
DBL_MIN 1E-37
LDBL_MIN 1E-37
Minimum normalized positive floating-point number.
These take the following values in the T2EX reference implementation.
FLT_MIN 1.17549435e-38F
DBL_MIN 2.2250738585072014e-308
LDBL_MIN 2.2250738585072014e-308L

FLT_EPSILON 1E-5
DBL_EPSILON 1E-9
LDBL_EPSILON 1E-9
The difference between 1 and the least value greater than 1 that is representable in the given floating-point type.
These take the following values in the T2EX reference implementation.
FLT_EPSILON 1.19209290e-7F
DBL_EPSILON 2.2204460492503131e-16
LDBL_EPSILON 2.2204460492503131e-16L

## 8.10 inttypes.h

The header inttypes.h defines the type, functions, and macros related to the handling of fixed-size integers.
The header inttypes.h includes <stdint.h> internally.

### Type

imaxdiv_t
Structure type to store the value returned by imaxdiv()

### Macro

The following macros are for use in format (the formatted input/output function) when converting the corresponding integer type.
Each of these macros includes a conversion specifier and is expanded to a string literal modified by a length modifier as needed.

General forms of these macros begin with one of the followings:
- PRI (string literal for fprintf() series functions)
- SCN (string literal for fscanf() series functions)

This is followed by a specifier (d, i, o, u, x, X), and then a name corresponding to the type name similar to what is defined in stdint.h.

In these names, N represents the width of the type described in stdint.h.
For instance, PRIdFAST32 can be used as the format string for outputting the int_fast32_t type integer value.

Macros for fprintf() for signed integer:

| | | | | |
|---|---|---|---|---|
| PRIdN | PRIdLEASTN | PRIdFASTN | PRIdMAX | PRIdPTR |
| PRIiN | PRIiLEASTN | PRIiFASTN | PRIiMAX | PRIiPTR |

Macros for fprintf() for unsigned integer:

| | | | | |
|---|---|---|---|---|
| PRIoN | PRIoLEASTN | PRIoFASTN | PRIoMAX | PRIoPTR |
| PRIuN | PRIuLEASTN | PRIuFASTN | PRIuMAX | PRIuPTR |
| PRIxN | PRIxLEASTN | PRIxFASTN | PRIxMAX | PRIxPTR |
| PRIXN | PRIXLEASTN | PRIXFASTN | PRIXMAX | PRIXPTR |

Macros for fscanf() for signed integer:

| | | | | |
|---|---|---|---|---|
| SCNdN | SCNdLEASTN | SCNdFASTN | SCNdMAX | SCNdPTR |
| SCNiN | SCNiLEASTN | SCNiFASTN | SCNiMAX | SCNiPTR |

Macros for fscanf() for unsigned integer:

| | | | | |
|---|---|---|---|---|
| SCNoN | SCNoLEASTN | SCNoFASTN | SCNoMAX | SCNoPTR |
| SCNuN | SCNuLEASTN | SCNuFASTN | SCNuMAX | SCNuPTR |
| SCNxN | SCNxLEASTN | SCNxFASTN | SCNxMAX | SCNxPTR |

The processor needs to define a corresponding fprintf macro for each type provided by stdint.h.
If the processor has an appropriate length modifier for the type, it also needs to define a corresponding fscanf macro.

### Functions (can also be defined as macro)

#### 8.10.1 imaxabs - Absolute value of an integer

C Language Interface

```
#include <inttypes.h>

intmax_t        imaxabs(intmax_t j);
```

Description

imaxabs() calculates the absolute value of the integer j.
If the result is unrepresentable, the behavior shall be undefined.

Return Parameter

imaxabs() returns an absolute value.

Error Code

None

See Also

imaxdiv

8.10.2 imaxdiv - Quotient and remainder of integers

C Language Interface

```
#include <inttypes.h>

imaxdiv_t       imaxdiv(intmax_t numer, intmax_t denom);
```

Description

The imaxdiv() calculates "numer / denom" and "number % denom" in a single operation.

Return Parameter

The imbxdiv() returns an imaxdiv_t type structure in which the quotient and remainder are stored.
This structure needs to contain the members of intmax_t type quot (quotient) and rem (remainder)
(order is irrelevant).
If either one in the result is unrepresentable, the behavior shall be undefined.

Error Code

None

See Also

imaxabs

8.10.3 strtoimax, strtoumax - Converts a string to an integer type

C Language Interface

```
#include <inttypes.h>

intmax_t        strtoimax(const char *nptr, char **endptr, int base);
uintmax_t       strtoumax(const char *nptr, char **endptr, int base);
```

Description

These functions are equivalent to strtol(), strtoll(), strtoul(), and strtoull() except that the types
of the convertion result are intmax_t and unitmax_t, respectively.

Return Parameter

These functions return a converted value once the conversion takes place.
If conversion is not performed, 0 is returned.
If the result exceeds the range of values representable, INTMAX_MAX, INTMAX_MIN, or UINTMAX_MAX is

returned according to the sign and type of the value.

Error Code

None

See Also

strtol, strtoul

## 8.11 iso646.h

The header iso646.h defines the following macros that provide alternative descriptions regarding operators.

### Macros

The following macros on the left expand to the corresponding tokens on the right:

| | |
|---|---|
| and | && |
| and_eq | &= |
| bitand | & |
| bitor | \| |
| compl | ~ |
| not | ! |
| not_eq | != |
| or | \|\| |
| or_eq | \|= |
| xor | ^ |
| xor_eq | ^= |

8.12 limits.h

The header limits.h defines macros regarding various limits.

Constants

Numerical Limits:
If the value on the right is positive, the limit must be this value or less. If it is negative, the limit must be this value or more.

If the values of type char are treated as signed integers when used in an expression, the value of CHAR_MIN is the same as that of SCHAR_MIN and the value of CHAR_MAX is the same as that of SCHAR_MAX. Otherwise, the value of CHAR_MIN is 0 and the value of CHAR_MAX is the same as that of UCHAR_MAX.

CHAR_BIT 8
Number of bits in a type char.

CHAR_MAX UCHAR_MAX or SCHAR_MAX
Maximum value for an object of type char.

CHAR_MIN 0 or SCHAR_MIN
Minimum value for an object of type char.

INT_MAX +2147483647
Maximum value for an object of type int.

INT_MIN -2147483647
Minimum value for an object of type int.

LLONG_MAX +9223372036854775807
Maximum value for an object of type long long.

LLONG_MIN -9223372036854775807
Minimum value for an object of type long long.

LONG_BIT 32
Number of bits in an object of type long.

LONG_MAX +2147483647
Maximum value for an object of type long.

LONG_MIN -2147483647
Minimum value for an object of type long.

MB_LEN_MAX 1
Maximum number of bytes in a character, for the system locale.

SCHAR_MAX +127
Maximum value for an object of type signed char.

SCHAR_MIN -128
Minimum value for an object of type signed char.

SHRT_MAX +32767
Maximum value for an object of type short.

SHRT_MIN -32767
Minimum value for an object of type short.

SSIZE_MAX 32767
Maximum value for an object of type ssize_t.

UCHAR_MAX 255
Maximum value for an object of type unsigned char.

UINT_MAX 4294967295
Maximum value for an object of type unsigned int.

ULLONG_MAX 18446744073709551615
Maximum value for an object of type unsigned long long.

ULONG_MAX 4294967295

Maximum value for an object of type unsigned long.

USHRT_MAX 65535
Maximum value for an object of type unsigned short.

WORD_BIT 32
Number of bits in an object of type int.

Limits on filename and pathname:
The value of the symbol should be greater or equal than the value shown on the right side.

NAME_MAX 255
Maximum number of bytes in a filename (not including the terminating null).

PATH_MAX 1024
Maximum number of bytes the implementation will store as a pathname in a user-supplied buffer of unspecified size, including the terminating null character.

## 8.13 math.h

The header math.h defines the following macros and function prototype declarations regarding mathematical items.

### Macros

```
int fpclassify(real-floating x);
```

The fpclassify() macro shall classify its argument value as NaN, infinite, normal, subnormal, zero, or into another implementation-defined category.
The fpclassify() macro shall return the value of the number classification macro (described below) appropriate to the value of its argument.
No implementation-defined categories are provided in the T2EX reference implementation.

```
int isfinite(real-floating x);
```

The isfinite() macro shall determine whether its argument has a finite value (zero, subnormal, or normal, and not infinite or NaN).
The isfinite() macro shall return a non-zero value if and only if its argument has a finite value.

```
int isinf(real-floating x);
```

The isinf() macro shall determine whether its argument value is an infinity (positive or negative).
The isinf() macro shall return a non-zero value if and only if its argument has an infinite value.

```
int isnan(real-floating x);
```

The isnan() macro shall determine whether its argument value is a NaN.
The isnan() macro shall return a non-zero value if and only if its argument has a NaN value.

```
int isnormal(real-floating x);
```

The isnormal() macro shall determine whether its argument value is normal (neither zero, subnormal, infinite, nor NaN).
The isnormal() macro shall return a non-zero value if and only if its argument has a normal value.

```
int signbit(real-floating x);
```

The signbit() macro shall determine whether the sign of its argument value is negative. NaNs, zeros, and infinities have a sign bit.
The signbit() macro shall return a non-zero value if and only if the sign of its argument value is negative.

```
int isgreater(real-floating x, real-floating y);
```

The isgreater() macro shall determine whether its first argument is greater than its second argument. The value of isgreater( x, y) shall be equal to (x) > (y); however, unlike (x) > (y), isgreater( x, y) shall not raise the invalid floating-point exception when x and y are unordered.
Upon successful completion, the isgreater() macro shall return the value of (x) > (y).
If x or y is NaN, 0 shall be returned.

```
int isgreaterequal(real-floating x, real-floating y);
```

The isgreaterequal() macro shall determine whether its first argument is greater than or equal to its second argument. The value of isgreaterequal( x, y) shall be equal to (x) >= (y); however, unlike (x) >= (y), isgreaterequal( x, y) shall not raise the invalid floating-point exception when x and y are unordered.
Upon successful completion, the isgreaterequal() macro shall return the value of (x) >= (y).
If x or y is NaN, 0 shall be returned.

```
int isless(real-floating x, real-floating y);
```

The isless() macro shall determine whether its first argument is less than its second

argument. The value of isless( x, y) shall be equal to (x) < (y); however, unlike (x) < (y), isless( x, y) shall not raise the invalid floating-point exception when x and y are unordered.
Upon successful completion, the isless() macro shall return the value of (x) < (y).
If x or y is NaN, 0 shall be returned.

int islessequal(real-floating x, real-floating y);

The islessequal() macro shall determine whether its first argument is less than or equal to its second argument. The value of islessequal( x, y) shall be equal to (x) <= (y); however, unlike (x) <= (y), islessequal( x, y) shall not raise the invalid floating-point exception when x and y are unordered.
Upon successful completion, the islessequal() macro shall return the value of (x) <= (y).
If x or y is NaN, 0 shall be returned.

int islessgreater(real-floating x, real-floating y);

The islessgreater() macro shall determine whether its first argument is less than or greater than its second argument. The islessgreater( x, y) macro is similar to (x) < (y) || (x) > (y); however, islessgreater( x, y) shall not raise the invalid floating-point exception when x and y are unordered (nor shall it evaluate x and y twice).
Upon successful completion, the islessgreater() macro shall return the value of (x) < (y) || (x) > (y).
If x or y is NaN, 0 shall be returned.

int isunordered(real-floating x, real-floating y);

The isunordered() macro shall determine whether its arguments are unordered.
Upon successful completion, the isunordered() macro shall return 1 if its arguments are unordered, and 0 otherwise.
If x or y is NaN, 1 shall be returned.

Number classification macro values

Number classification macro values shall be defined for number classification. They expand to
integer constant expressions with distinct values.
The followings are defined in the T2EX reference implementation.
Additional implementation-defined floating-point classifications, with macro definitions beginning with FP_ and an uppercase letter, may also be specified by the implementation.

| | |
|---|---|
| FP_INFINITE | Positive or negative infinity.<br>(Denoted as +Inf, -Inf, or +/-Inf) |
| FP_NAN | Not-a-number.<br>(Denoted as NaN) |
| FP_NORMAL | Normal number. |
| FP_SUBNORMAL | Subnormal number. |
| FP_ZERO | Zero. (+0 or -0) |

Returned value of ilogb(x)
The following macros shall expand to integer constant expressions whose values are returned by ilogb(x).

| | |
|---|---|
| FP_ILOGB0 | Returned value if x is zero.<br>The value shall be either INT_MIN or -INT_MAX. |
| FP_ILOGBNAN | Returned value if x is NaN.<br>The value shall be either INT_MAX or INT_MIN. |

Symbolic constants.
The values shall have type double and shall be accurate within the precision of the double type.

| | |
|---|---|
| M_E | Value of e |
| M_LOG2E | Value of log2(e) |
| M_LOG10E | Value of log10(e) |
| M_LN2 | Value of loge(2) |
| M_LN10 | Value of loge(10) |
| M_PI | Value of $\pi$ |
| M_PI_2 | Value of $\pi/2$ |
| M_PI_4 | Value of $\pi/4$ |

| | |
|---|---|
| M_1_PI | Value of 1/π |
| M_2_PI | Value of 2/π |
| M_2_SQRTPI | Value of 2/sqrt(π) |
| M_SQRT2 | Value of sqrt(2) |
| M_SQRT1_2 | Value of 1/(sqrt(2) |
| MAXFLOAT | Same value as FLT_MAX in <float.h>. |

Special symbolic constants:
The <math.h> header shall define the following macros:

| | |
|---|---|
| HUGE_VAL | A positive double constant expression, not necessarily representable as a float. Used as an error value returned by the mathematics library. |
| HUGE_VALF | A positive float constant expression. Used as an error value returned by the mathematics library. |
| HUGE_VALL | A positive long double constant expression. Used as an error value returned by the mathematics library. |
| INFINITY | A constant expression of type float representing positive or unsigned infinity, if available; else a positive constant of type float that overflows at translation time. |
| NAN | A constant expression of type float representing a quiet NaN. This macro is only defined if the implementation supports quiet NaNs for the float type. Quiet NaN is outputted as an operation result without raising any exception when the operation is performed on itself. As a rule, T2EX supports NaN and any NaN shall be a quiet NaN. However, implementation that does not support NaN is also allowed. |

Functions

## 8.13.1 acos, acosf, acosl - arc cosine functions

C Language Interface

```
#include <math.h>

double          acos(double x);
float           acosf(float x);
long double     acosl(long double x);
```

Description

These functions shall compute the principal value of the arc cosine of their argument x.
The value of x should be in the range [-1,1].

Return Parameter

Upon successful completion, these functions shall return the arc cosine of x, in the range [0, π] radians.
For finite values of x not in the range [-1,1], a NaN shall be returned.
If x is NaN, a NaN shall be returned.
If x is +1, +0 shall be returned.
If x is +/-Inf, a NaN shall be returned.

Error Code

None.

See Also

cos

### 8.13.2 acosh, acoshf, acoshl - inverse hyperbolic cosine functions

C Language Interface

```
#include <math.h>

double          acosh(double x);
float           acoshf(float x);
long double     acoshl(long double x);
```

Description

These functions shall compute the inverse hyperbolic cosine of their argument x.

Return Parameter

Upon successful completion, these functions shall return the inverse hyperbolic cosine of their argument.
For finite values of x < 1, a NaN shall be returned.
If x is NaN, a NaN shall be returned.
If x is +1, +0 shall be returned.
If x is +Inf, +Inf shall be returned.
If x is -Inf, a NaN shall be returned.

Error Code

None.

See Also

cosh

### 8.13.3 asin, asinf, asinl - arc sine function

C Language Interface

```
#include <math.h>

double          asin(double x);
float           asinf(float x);
long double     asinl(long double x);
```

Description

These functions shall compute the principal value of the arc sine of their argument x.
The value of x should be in the range [-1,1].

Return Parameter

Upon successful completion, these functions shall return the arc sine of x, in the range $[-\pi/2, \pi/2]$ radians.
For finite values of x not in the range [-1,1], a NaN shall be returned.
If x is NaN, a NaN shall be returned.
If x is +/-0, x shall be returned.
If x is +/-Inf, a NaN shall be returned.
If x is subnormal, x should be returned.

Error Code

None.

See Also

sin

8.13.4 asinh, asinhf, asinhl - inverse hyperbolic sine functions

C Language Interface

```
#include <math.h>

double          asinh(double x);
float           asinhf(float x);
long double     asinhl(long double x);
```

Description

These functions shall compute the inverse hyperbolic sine of their argument x.

Return Parameter

Upon successful completion, these functions shall return the inverse hyperbolic sine of their argument.
If x is NaN, a NaN shall be returned.
Otherwise, x shall be returned.

Error Code

None.

See Also

sinh

8.13.5 atan, atanf, atanl - arc tangent function

C Language Interface

```
#include <math.h>

double          atan(double x);
float           atanf(float x);
long double     atanl(long double x);
```

Description

These functions shall compute the principal value of the arc tangent of their argument x.

Return Parameter

Upon successful completion, these functions shall return the arc tangent of x in the range [-π/2, π/2] radians.
If x is NaN, a NaN shall be returned.
If x is +/-Inf, +/-π/2 shall be returned.
Otherwise, x shall be returned.

Error Code

None.

See Also

tan

### 8.13.6 atan2, ata2f, atan2l - arc tangent functions

C Language Interface

```
#include <math.h>

double          atan2(double y, double x);
float           atan2f(float y, float x);
long double     atan2l(long double y, long double x);
```

Description

These functions shall compute the principal value of the arc tangent of y/x, using the signs of both arguments to determine the quadrant of the return value.

Return Parameter

Upon successful completion, these functions shall return the arc tangent of y/x in the range [-π, π] radians.
If y is +/-0 and x is < 0, +/-π shall be returned.
If y is +/-0 and x is > 0, +/-0 shall be returned.
If y is < 0 and x is +/-0, -π/2 shall be returned.
If y is > 0 and x is +/-0, π/2 shall be returned.
If either x or y is NaN, a NaN shall be returned.
If the result underflows, y/x should be returned.
If y is +/-0 and x is -0, +/-π shall be returned.
If y is +/-0 and x is +0, +/-0 shall be returned.
For finite values of +/- y > 0, if x is -Inf, +/-π shall be returned.
For finite values of +/- y > 0, if x is +Inf, +/-0 shall be returned.
For finite values of x, if y is +/-Inf, +/-π/2 shall be returned.
If y is +/-Inf and x is -Inf, +/-3*π/4 shall be returned.
If y is +/-Inf and x is +Inf, +/-π/4 shall be returned.

Error Code

None.

See Also

tan, atan

### 8.13.7 atanh, atanhf, atanhl - inverse hyperbolic tangent functions

C Language Interface

```
#include <math.h>

double          atanh(double x);
float           atanhf(float x);
long double     atanhl(long double x);
```

Description

These functions shall compute the inverse hyperbolic tangent of their argument x.

Return Parameter

Upon successful completion, these functions shall return the inverse hyperbolic tangent of their argument.
If x is +/-1, atanh(), atanhf(), and atanhl() shall return the value of the macro HUGE_VAL, HUGE_VALF, and HUGE_VALL, respectively, with the same sign as the correct value of the function.

For finite |x|>1, a NaN shall be returned.
If x is NaN, a NaN shall be returned.
If x is +/-0, x shall be returned.
If x is +/-Inf, a NaN shall be returned.
If x is subnormal, x should be returned.

Error Code

None.

See Also

tanh

8.13.8 cbrt, cbrtf, cbrtl - cube root functions

C Language Interface

```
#include <math.h>

double          cbrt(double x);
float           cbrtf(float x);
long double     cbrtl(long double x);
```

Description

These functions shall compute the real cube root of their argument x.

Return Parameter

Upon successful completion, these functions shall return the cube root of x.
If x is NaN, a NaN shall be returned.
If x is +/-0 or +/-Inf, x shall be returned.

Error Code

None.

8.13.9 ceil, ceilf, ceill - ceiling value function

C Language Interface

```
#include <math.h>

double          ceil(double x);
float           ceilf(float x);
long double     ceill(long double x);
```

Description

These functions shall compute the smallest integral value not less than x.

Return Parameter

Upon successful completion, ceil(), ceilf(), and ceill() shall return the smallest integral value not less than x, expressed as a type double, float, or long double, respectively.
If x is NaN, a NaN shall be returned.
If x is +/-0 or +/-Inf, x shall be returned.
If the correct value would cause overflow, ceil(), ceilf(), and ceill() shall return the value of the macro HUGE_VAL, HUGE_VALF, and HUGE_VALL, respectively.

Error Code

None.

### 8.13.10 copysign, copysignf, copysignl - number manipulation function

C Language Interface

```
#include <math.h>

double          copysign(double x, double y);
float           copysignf(float x, float y);
long double     copysignl(long double x, long double y);
```

Description

These functions shall produce a value with the magnitude of x and the sign of y.

Return Parameter

Upon successful completion, these functions shall return a value with the magnitude of x and the sign of y.

Error Code

None.

See Also

floor

### 8.13.11 cos, cosf, cosl - cosine function

C Language Interface

```
#include <math.h>

double          cos(double x);
float           cosf(float x);
long double     cosl(long double x);
```

Description

These functions shall compute the cosine of their argument x, measured in radians.

Return Parameter

Upon successful completion, these functions shall return the cosine of x.
If x is NaN, a NaN shall be returned.
If x is +/-0, the value 1.0 shall be returned.
If x is +/-Inf, a NaN shall be returned.

Error Code

None.

See Also

sin, tan

8.13.12 cosh, coshf, coshl - hyperbolic cosine functions

C Language Interface

```
#include <math.h>

double          cosh(double x);
float           coshf(float x);
long double     coshl(long double x);
```

Description

These functions shall compute the hyperbolic cosine of their argument x.

Return Parameter

Upon successful completion, these functions shall return the hyperbolic cosine of x.
If x is NaN, a NaN shall be returned.
If x is +/-0, the value 1.0 shall be returned.
If x is +/-Inf, +Inf shall be returned.

Error Code

None.

See Also

sinh, tanh

8.13.13 erf, erff, erfl - error functions

C Language Interface

```
#include <math.h>

double          erf(double x);
float           erff(float x);
long double     erfl(long double x);
```

Description

These functions shall compute the error function of their argument x, defined as:

```
        erf(x) = 2/sqrt(pi) * integral from 0 to x of exp(-t*t) dt
```

Return Parameter

Upon successful completion, these functions shall return the value of the error function.
If x is NaN, a NaN shall be returned.
If x is +/-0, +/-0 shall be returned.
If x is +/-Inf, +/-1 shall be returned.
If x is subnormal, 2 * x/ sqrt(π) should be returned.

Error Code

None.

See Also

erfc

8.13.14 erfc, erfcf, erfcl - complementary error functions

C Language Interface

```
#include <math.h>

double          erfc(double x);
float           erfcf(float x);
long double     erfcl(long double x);
```

Description

These functions shall compute the complementary error function 1.0 - erf(x).

Return Parameter

Upon successful completion, these functions shall return the value of the complementary error function.
If x is NaN, a NaN shall be returned.
If x is +/-0, +1 shall be returned.
If x is -Inf, +2 shall be returned.
If x is +Inf, +0 shall be returned.

Error Code

None.

See Also

erf

8.13.15 exp, expf, expl - exponential function

C Language Interface

```
#include <math.h>

double          exp(double x);
float           expf(float x);
long double     expl(long double x);
```

Description

These functions shall compute the base-e exponential of x.

Return Parameter

Upon successful completion, these functions shall return the exponential value of x.
If the correct value would cause overflow, exp(), expf(), and expl() shall return the value of the macro HUGE_VAL, HUGE_VALF, and HUGE_VALL, respectively.
If the correct value would cause underflow, and is not representable, 0.0 shall be returned.
If x is NaN, a NaN shall be returned.
If x is +/-0, 1 shall be returned.
If x is -Inf, +0 shall be returned.
If x is +Inf, x shall be returned.

Error Code

None.

See Also

log

8.13.16 exp2, exp2f, exp2l - exponential base-2 functions

C Language Interface

```
#include <math.h>

double          exp2(double x);
float           exp2f(float x);
long double     exp2l(long double x);
```

Description

These functions shall compute the base-2 exponential of x.

Return Parameter

Upon successful completion, these functions shall return 2^x ('^' represents the power).
If the correct value would cause overflow, exp2(), exp2f(), and exp2l() shall return the value of the macro HUGE_VAL, HUGE_VALF, and HUGE_VALL, respectively.
If the correct value would cause underflow, and is not representable, 0.0 shall be returned.
If x is NaN, a NaN shall be returned.
If x is +/-0, 1 shall be returned.
If x is -Inf, +0 shall be returned.
If x is +Inf, x shall be returned.

Error Code

None.

See Also

exp, log

8.13.17 expm1, expm1f, expm1l - compute exponential functions

C Language Interface

```
#include <math.h>

double          expm1(double x);
float           expm1f(float x);
long double     expm1l(long double x);
```

Description

These functions shall compute e^x-1.0 ('^' represents the power).

Return Parameter

Upon successful completion, these functions return e^x-1.0.
If the correct value would cause overflow, expm1(), expm1f(), and expm1l() shall return the value of the macro HUGE_VAL, HUGE_VALF, and HUGE_VALL, respectively.
If x is NaN, a NaN shall be returned.
If x is +/-0, +/-0 shall be returned.
If x is -Inf, -1 shall be returned.
If x is +Inf, x shall be returned.
If x is subnormal, x should be returned.

Error Code

None.

See Also

exp, ilogb, log1p

8.13.18 fabs, fabsf, fabsl - absolute value function

C Language Interface

```
#include <math.h>

double          fabs(double x);
float           fabsf(float x);
long double     fabsl(long double x);
```

Description

These functions shall compute the absolute value of their argument x, |x|.

Return Parameter

Upon successful completion, these functions shall return the absolute value of x.
If x is NaN, a NaN shall be returned.
If x is +/-0, +0 shall be returned.
If x is +/-Inf, +Inf shall be returned.

Error Code

None.

See Also

isnan

8.13.19 fdim, fdimf, fdiml - compute positive difference between two floating-point numbers

C Language Interface

```
#include <math.h>

double          fdim(double x, double y);
float           fdimf(float x, float y);
long double     fdiml(long double x, long double y);
```

Description

These functions shall determine the positive difference between their arguments. If x is greater than y, x - y is returned. If x is less than or equal to y, +0 is returned.

Return Parameter

Upon successful completion, these functions shall return the positive difference value.
If x - y is positive and overflows, fdim(), fdimf(), and fdiml() shall return the value of the macro HUGE_VAL, HUGE_VALF, and HUGE_VALL, respectively.
If x - y is positive and underflows, 0.0 shall be returned.
If x or y is NaN, a NaN shall be returned.

Error Code

None.

8.13.20 floor, floorf, floorl - floor function

C Language Interface

```
#include <math.h>

double          floor(double x);
float           floorf(float x);
long double     floorl(long double x);
```

Description

These functions shall compute the largest integral value not greater than x.

Return Parameter

Upon successful completion, these functions shall return the largest integral value not greater than x, expressed as a double, float, or long double, as appropriate for the return type of the function.
If x is NaN, a NaN shall be returned.
If x is +/-0 or +/-Inf, x shall be returned.
If the correct value would cause overflow, floor(), floorf(), and floorl() shall return the value of the macro -HUGE_VAL, -HUGE_VALF, and -HUGE_VALL, respectively.

Error Code

None.

See Also

fmax, fmin

### 8.13.21 fma, fmaf, fmal - floating-point multiply-add

C Language Interface

```
#include <math.h>

double          fma(double x, double y, double z);
float           fmaf(float x, float y, float z);
long double     fmal(long double x, long double y, long double z);
```

Description

These functions shall compute (x * y) + z, rounded as one ternary operation: they shall compute the value (as if) to infinite precision and round once to the result format, according to the rounding mode characterized by the value of FLT_ROUNDS.

Return Parameter

Upon successful completion, these functions shall return (x * y) + z, rounded as one ternary operation.
If x or y are NaN, a NaN shall be returned.
If x multiplied by y is an exact infinity and z is also an infinity but with the opposite sign, a NaN shall be returned.
If one of x and y is infinite, the other is zero, and z is not a NaN, a NaN shall be returned.
If one of x and y is infinite, the other is zero, and z is a NaN, a NaN shall be returned.
If x*y is not 0*Inf nor Inf*0 and z is a NaN, a NaN shall be returned.

Error Code

None.

### 8.13.22 fmax, fmaxf, fmaxl - determine maximum numeric value of two floating-point numbers

C Language Interface

```
#include <math.h>

double          fmax(double x, double y);
float           fmaxf(float x, float y);
long double     fmaxl(long double x, long double y);
```

Description

These functions shall determine the maximum numeric value of their arguments.

Return Parameter

Upon successful completion, these functions shall return the maximum numeric value of their arguments.
If just one argument is a NaN, the other argument shall be returned.
If x and y are NaN, a NaN shall be returned.

Error Code

None.

See Also

fdim, fmin

### 8.13.23 fmin, fminl, fminf - determine minimum numeric value of two floating-point numbers

C Language Interface

```
#include <math.h>

double          fmin(double x, double y);
float           fminf(float x, float y);
long double     fminl(long double x, long double y);
```

Description

These functions shall determine the minimum numeric value of their arguments.

Return Parameter

Upon successful completion, these functions shall return the minimum numeric value of their arguments.
If just one argument is a NaN, the other argument shall be returned.
If x and y are NaN, a NaN shall be returned.

Error Code

None.

See Also

fdim, fmax

### 8.13.24 fmod, fmodf, fmodl - floating-point remainder value function

C Language Interface

```
#include <math.h>

double          fmod(double x, double y);
float           fmodf(float x, float y);
long double     fmodl(long double x, long double y);
```

Description

These functions shall return the floating-point remainder of the division of x by y.

Return Parameter

These functions shall return the value x-i*y, for some integer i such that, if y is non-zero, the result has the same sign as x and magnitude less than the magnitude of y.
If the correct value would cause underflow, and is not representable, 0.0 shall be returned.
If x or y is NaN, a NaN shall be returned.
If y is zero, a NaN shall be returned.
If x is infinite, a NaN shall be returned.
If x is +/-0 and y is not zero, +/-0 shall be returned.
If x is not infinite and y is +/-Inf, x shall be returned.

Error Code

None.

See Also

isnan

8.13.25 frexp, frexpf, frexpl - extract mantissa and exponent from a double precision number

C Language Interface

```
#include <math.h>

double          frexp(double num, int *exp);
float           frexpf(float num, int *exp);
long double     frexpl(long double num, int *exp);
```

Description

These functions shall break a floating-point number num into a normalized fraction and an integral power of 2. The integer exponent shall be stored in the int object pointed to by exp.

Return Parameter

For finite arguments, these functions shall return the value x, such that x has a magnitude in the interval [1/2,1) or 0, and num equals x times 2 raised to the power *exp.
If num is NaN, a NaN shall be returned, and the value of *exp is unspecified.
If num is +/-0, +/-0 shall be returned, and the value of *exp shall be 0.
If num is +/-Inf, num shall be returned, and the value of *exp is unspecified.

Error Code

None.

See Also

ldexp, modf

8.13.26 hypot, hypotf, hypotl - Euclidean distance function

C Language Interface

```
#include <math.h>

double          hypot(double x, double y);
float           hypotf(float x, float y);
long double     hypotl(long double x, long double y);
```

Description

These functions shall compute the value of the square root of x*x + y*y without undue overflow or underflow.

Return Parameter

Upon successful completion, these functions shall return the length of the hypotenuse of a right-angled triangle with sides of length x and y.
If the correct value would cause overflow, hypot(), hypotf(), and hypotl() shall return the value of the macro HUGE_VAL, HUGE_VALF, and HUGE_VALL, respectively.
If x or y is +/-Inf, +Inf shall be returned (even if one of x or y is NaN).
If x or y is NaN, and the other is not +/-Inf, a NaN shall be returned.

Error Code

None.

See Also

atan2, sqrt

8.13.27 ilogb, ilogbf, ilogbl - return an unbiased exponent

C Language Interface

```
#include <math.h>

int             ilogb(double x);
int             ilogbf(float x);
int             ilogbl(long double x);
```

Description

These functions shall return the exponent part of their argument x as a signed integer value.

Return Parameter

Upon successful completion, these functions shall return the exponent part of x as a signed integer value.
If x is 0, the value FP_ILOGB0 shall be returned.
If x is +/-Inf, the value INT_MAX shall be returned.
If x is a NaN, the value FP_ILOGBNAN shall be returned.
If the correct value is greater than INT_MAX, INT_MAX shall be returned.
If the correct value is less than INT_MIN, INT_MIN shall be returned.

Error Code

None.

See Also

logb, scalbln

8.13.28 j0, j1, jn - Bessel functions of the first kind

C Language Interface

```
#include <math.h>

double          j0(double x);
```

```
double          j1(double x);
double          jn(int n, double x);
```

Description

The j0(), j1(), and jn() functions shall compute Bessel functions of x of the first kind of orders 0, 1, and n, respectively.

Return Parameter

Upon successful completion, these functions shall return the relevant Bessel value of x of the first kind.
If the x argument is too large in magnitude, or the correct result would cause underflow, 0 shall be returned.
If x is NaN, a NaN shall be returned.

Error Code

None.

See Also

y0

8.13.29 ldexp, ldexpf, ldexpl - load exponent of a floating-point number

C Language Interface

```
#include <math.h>

double          ldexp(double x, int exp);
float           ldexpf(float x, int exp);
long double     ldexpl(long double x, int exp);
```

Description

These functions shall compute the quantity x * 2^exp ('^' represents the power).

Return Parameter

Upon successful completion, these functions shall return x multiplied by 2, raised to the power exp.
If these functions would cause overflow, ldexp(), ldexpf(), and ldexpl() shall return +/-HUGE_VAL, +/-HUGE_VALF, and +/-HUGE_VALL (according to the sign of x), respectively.
If the correct value would cause underflow, and is not representable, 0.0 shall be returned.
If x is NaN, a NaN shall be returned.
If x is +/-0 or +/-Inf, x shall be returned.
If exp is 0, x shall be returned.

Error Code

None.
frexp(), isnan()

8.13.30 llrint, llrintf, llrintl - round to the nearest integer value using current rounding direction

C Language Interface

```
#include <math.h>

long long       llrint(double x);
long long       llrintf(float x);
long long       llrintl(long double x);
```

Description

These functions shall round their argument to the nearest integer value, rounding according to the current rounding direction.

Return Parameter

Upon successful completion, these functions shall return the rounded integer value.
If the correct value is positive and too large to represent as a long long value, an unspecified value shall be returned.
If the correct value is negative and too large to represent as a long long value, an unspecified value shall be returned.
If x is NaN, an unspecified value is returned.
If x is +Inf, an unspecified value is returned.
If x is -Inf, an unspecified value is returned.

Error Code

None.

See Also

lrint

8.13.31 llround, llroundr, llroundl - round to nearest integer value

C Language Interface

```
#include <math.h>

long long       llround(double x);
long long       llroundf(float x);
long long       llroundl(long double x);
```

Description

These functions shall round their argument to the nearest integer value, rounding halfway cases away from zero, regardless of the current rounding direction.

Return Parameter

Upon successful completion, these functions shall return the rounded integer value.
If the correct value is positive and too large to represent as a long long value, an unspecified value shall be returned.
If the correct value is negative and too large to represent as a long long value, an unspecified value shall be returned.
If x is NaN, an unspecified value is returned.
If x is +Inf, an unspecified value is returned.
If x is -Inf, an unspecified value is returned.

Error Code

None.

See Also

lround

8.13.32 log, logf, logl - natural logarithm function

C Language Interface

```
#include <math.h>

double          log(double x);
float           logf(float x);
long double     logl(long double x);
```

Description

These functions shall compute the natural logarithm of their argument x, loge(x).

Return Parameter

Upon successful completion, these functions shall return the natural logarithm of x.
If x is +/-0, log(), logf(), and logl() shall return -HUGE_VAL, -HUGE_VALF, and -HUGE_VALL, respectively.
For finite values of x that are less than 0, or if x is -Inf, a NaN shall be returned.
If x is NaN, a NaN shall be returned.
If x is 1, +0 shall be returned.
If x is +Inf, x shall be returned.

Error Code

None.

See Also

exp, log10, log1p

8.13.33 log10, log10f, log10l - base-10 logarithm function

C Language Interface

```
#include <math.h>

double          log10(double x);
float           log10f(float x);
long double     log10l(long double x);
```

Description

These functions shall compute the base-10 logarithm of their argument x, log10(x).

Return Parameter

Upon successful completion, these functions shall return the base-10 logarithm of x.
If x is +/-0, log10(), log10f(), and log10l() shall return -HUGE_VAL, -HUGE_VALF, and -HUGE_VALL, respectively.
For finite values of x that are less than 0, or if x is -Inf, a NaN shall be returned.
If x is NaN, a NaN shall be returned.
If x is 1, +0 shall be returned.
If x is +Inf, +Inf shall be returned.

Error Code

None.

See Also

log, pow

8.13.34 log1p, log1pf, log1pl - compute a natural logarithm

C Language Interface

```
#include <math.h>

double          log1p(double x);
float           log1pf(float x);
long double     log1pl(long double x);
```

Description

These functions shall compute log(1.0 + x).

Return Parameter

Upon successful completion, these functions shall return the natural logarithm of 1.0 + x.
If x is -1, log1p(), log1pf(), and log1pl() shall return -HUGE_VAL, -HUGE_VALF, and -HUGE_VALL, respectively.
For finite values of x that are less than -1, or if x is -Inf, a NaN shall be returned.
If x is NaN, a NaN shall be returned.
If x is +/-0, or +Inf, x shall be returned.

Error Code

None.

See Also

log

8.13.35 log2, log2f, log2l - compute base-2 logarithm functions

C Language Interface

```
#include <math.h>

double          log2(double x);
float           log2f(float x);
long double     log2l(long double x);
```

Description

These functions shall compute the base-2 logarithm of their argument x, log2(x).

Return Parameter

Upon successful completion, these functions shall return the base-2 logarithm of x.
If x is +/-0, log2(), log2f(), and log2l() shall return -HUGE_VAL, -HUGE_VALF, and -HUGE_VALL, respectively.
For finite values of x that are less than 0, or if x is -Inf, a NaN shall be returned.
If x is NaN, a NaN shall be returned.
If x is 1, +0 shall be returned.
If x is +Inf, x shall be returned.

Error Code

None.

See Also

log

8.13.36 logb, lobf, lobl - radix-independent exponent

C Language Interface

```
#include <math.h>

double          logb(double x);
float           logbf(float x);
long double     logbl(long double x);
```

Description

These functions shall compute the exponent of x, which is the integral part of logr |x|, as a signed floating-point value, for non-zero x, where r is the radix of the machine's floating-point arithmetic, which is the value of FLT_RADIX defined in the <float.h> header.

If x is subnormal it is treated as though it were normalized; thus for finite positive x:

1 <= x * FLT_RADIX-logb(x) < FLT_RADIX

Return Parameter

Upon successful completion, these functions shall return the exponent of x.

If x is +/-0, logb(), logbf(), and logbl() shall return -HUGE_VAL, -HUGE_VALF, and -HUGE_VALL, respectively.
If x is NaN, a NaN shall be returned.
If x is +/-Inf, +Inf shall be returned.

Error Code

None.

See Also

ilogb, scalbln

8.13.37 lrint, lrintf, lrintl - round to nearest integer value using current rounding direction

C Language Interface

```
#include <math.h>

long            lrint(double x);
long            lrintf(float x);
long            lrintl(long double x);
```

Description

These functions shall round their argument to the nearest integer value, rounding according to the current rounding direction.

Return Parameter

Upon successful completion, these functions shall return the rounded integer value.
If x is NaN, an unspecified value is returned.
If x is +Inf, an unspecified value is returned.
If x is -Inf, an unspecified value is returned.
If the correct value is positive and too large to represent as a long, an unspecified value shall be returned.
If the correct value is negative and too large to represent as a long, an unspecified value shall be returned.

Error Code

None.

See Also

llrint

### 8.13.38 lround, lroundr, lroundl - round to nearest integer value

C Language Interface

```
#include <math.h>

long            lround(double x);
long            lroundf(float x);
long            lroundl(long double x);
```

Description

These functions shall round their argument to the nearest integer value, rounding halfway cases away from zero, regardless of the current rounding direction.

Return Parameter

Upon successful completion, these functions shall return the rounded integer value.
If x is NaN, an unspecified value is returned.
If x is +Inf, an unspecified value is returned.
If x is -Inf, an unspecified value is returned.
If the correct value is positive and too large to represent as a long, an unspecified value shall be returned.
If the correct value is negative and too large to represent as a long, an unspecified value shall be returned.

Error Code

None.

### 8.13.39 modf, modff, modfl - decompose a floating-point number

C Language Interface

```
#include <math.h>

double          modf(double x, double *iptr);
float           modff(float x, float *iptr);
long double     modfl(long double x, long double *iptr);
```

Description

These functions shall break the argument x into integral and fractional parts, each of which has the same sign as the argument. It stores the integral part as a double (for the modf() function), a float (for the modff() function), or a long double (for the modfl() function), in the object pointed to by iptr.

Return Parameter

Upon successful completion, these functions shall return the signed fractional part of x.
If x is NaN, a NaN shall be returned, and *iptr shall be set to a NaN.
If x is +/-Inf, +/-0 shall be returned, and *iptr shall be set to +/-Inf.

Error Code

None.

See Also

llround

8.13.40 nan, nanf, nanl - return quiet NaN

C Language Interface

```
#include <math.h>

double          nan(const char *tagp);
float           nanf(const char *tagp);
long double     nanl(const char *tagp);
```

Description

The function call nan("n-char-sequence") shall be equivalent to:

```
strtod("NAN(n-char-sequence)", (char **) NULL);
```

The function call nan("") shall be equivalent to:

```
strtod("NAN()", (char **) NULL)
```

If tagp does not point to an n- char sequence or an empty string, the function call shall be equivalent to:

```
strtod("NAN", (char **) NULL)
```

Function calls to nanf() and nanl() are equivalent to the corresponding function calls to strtof() and strtold().

The n-char-sequence information is stored in the available area in the binary representation of NaN and used to describe the reason why that NaN has been generated.

Return Parameter

These functions shall return a quiet NaN, if available, with content indicated through tagp.

T2EX supports a quiet NaN, but if the implementation does not support quiet NaNs, these functions shall return zero.

Error Code

None.

See Also

strtod

8.13.41 nearbyint, nearbyintf, nearbyintl - floating-point rounding functions

C Language Interface

```
#include <math.h>

double          nearbyint(double x);
float           nearbyintf(float x);
long double     nearbyintl(long double x);
```

Description

These functions shall round their argument to an integer value in floating-point format, using the current rounding direction and without raising the inexact floating-point exception.

Return Parameter

Upon successful completion, these functions shall return the rounded integer value.
If x is NaN, a NaN shall be returned.
If x is +/-0, +/-0 shall be returned.
If x is +/-Inf, x shall be returned.

Error Code

None.

8.13.42 nextfter, nextafterf, nextafterl, nexttoward, nexttowordf, nexttowordl - next representable floating-point number

C Language Interface

```
#include <math.h>

double          nextafter(double x, double y);
float           nextafterf(float x, float y);
long double     nextafterl(long double x, long double y);
double          nexttoward(double x, long double y);
float           nexttowardf(float x, long double y);
long double     nexttowardl(long double x, long double y);
```

Description

The nextafter(), nextafterf(), and nextafterl() functions shall compute the next representable floating-point value following x in the direction of y. Thus, if y is less than x, nextafter() shall return the largest representable floating-point number less than x. The nextafter(), nextafterf(), and nextafterl() functions shall return y if x equals y.

The nexttoward(), nexttowardf(), and nexttowardl() functions shall be equivalent to the corresponding nextafter() functions, except that the second parameter shall have type long double and the functions shall return y converted to the type of the function if x equals y.

Return Parameter

Upon successful completion, these functions shall return the next representable floating-point value following x in the direction of y.
If x== y, y (of the type x) shall be returned.

If x is finite and the correct function value would overflow, +/-HUGE_VAL, +/-HUGE_VALF, and +/-HUGE_VALL (with the same sign as x) shall be returned as appropriate for the return type of the function.
If x or y is NaN, a NaN shall be returned.

Error Code

None.

8.13.43 pow, powf, powl - power function

C Language Interface

```
#include <math.h>

double          pow(double x, double y);
float           powf(float x, float y);
long double     powl(long double x, long double y);
```

Description

These functions shall compute the value of x raised to the power y. If x is negative, the application shall ensure that y is an integer value.

Return Parameter

Upon successful completion, these functions shall return the value of x raised to the power y.

For finite values of x < 0, and finite non-integer values of y, a NaN shall be returned.
If the correct value would cause overflow, pow(), powf(), and powl() shall return +/-HUGE_VAL, +/-HUGE_VALF, and +/-HUGE_VALL, respectively, with the same sign as the correct value of the function.
If the correct value would cause underflow, and is not representable, 0.0 shall be returned.

For any value of y (including NaN), if x is +1, 1.0 shall be returned.
For any value of x (including NaN), if y is +/-0, 1.0 shall be returned.
For any odd integer value of y > 0, if x is +/-0, +/-0 shall be returned.
For y > 0 and not an odd integer, if x is +/-0, +0 shall be returned.
If x is -1, and y is +/-Inf, 1.0 shall be returned.
For |x| < 1, if y is -Inf, +Inf shall be returned.
For |x| > 1, if y is -Inf, +0 shall be returned.
For |x| < 1, if y is +Inf, +0 shall be returned.
For |x| > 1, if y is +Inf, +Inf shall be returned.
For y an odd integer < 0, if x is -Inf, -0 shall be returned.
For y < 0 and not an odd integer, if x is -Inf, +0 shall be returned.
For y an odd integer > 0, if x is -Inf, -Inf shall be returned.
For y > 0 and not an odd integer, if x is -Inf, +Inf shall be returned.
For y < 0, if x is +Inf, +0 shall be returned.
For y > 0, if x is +Inf, +Inf shall be returned.

Error Code

None.

See Also

exp, isnan

8.13.44 remainder, remainderf, remainderl - remainder function

C Language Interface

```
#include <math.h>

double          remainder(double x, double y);
float           remainderf(float x, float y);
long double     remainderl(long double x, long double y);
```

Description

These functions shall return the floating-point remainder r = x - n * y when y is non-zero.
The value n is the integral value nearest the exact value x / y. When |n - x / y| = 1/2, the value n is chosen to be even.

The behavior of remainder() shall be independent of the rounding mode.

Return Parameter

Upon successful completion, these functions shall return the floating-point remainder r = x- n * y when y is non-zero.
On systems that do not support the IEC 60559 Floating-Point option, if y is zero, zero is returned.

Error Code

None.

See Also

abs, div, ldiv

8.13.45 remquo, remquof, remquol - remainder functions

C Language Interface

```
#include <math.h>

double          remquo(double x, double y, int *quo);
float           remquof(float x, float y, int *quo);
long double     remquol(long double x, long double y, int *quo);
```

Description

The remquo(), remquof(), and remquol() functions shall compute the same remainder as the remainder(), remainderf(), and remainderl() functions, respectively. In the object pointed to by quo, they store a value whose sign is the sign of x / y and whose magnitude is congruent modulo 2^n ('^' represents the power) to the magnitude of the integral quotient of x / y, where n is an implementation-defined integer greater than or equal to 3.
In the T2EX reference implementation, n = 31.
If y is zero, the value stored in the object pointed to by quo is unspecified.

Return Parameter

These functions shall return x REM y.

On systems that do not support the IEC 60559 Floating-Point option, if y is zero, zero is returned.

Error Code

None.

See Also

remainder

8.13.46 rint, rintf, rintl - round-to-nearest integral value

C Language Interface

```
#include <math.h>

double          rint(double x);
float           rintf(float x);
long double     rintl(long double x);
```

Description

These functions shall return the integral value (represented as a double) nearest x in the direction of the current rounding mode.

If the current rounding mode rounds toward negative infinity, then rint() shall be equivalent to floor.
If the current rounding mode rounds toward positive infinity, then rint() shall be equivalent to ceil._RETURNV
Upon successful completion, these functions shall return the integer (represented as a double precision number) nearest x in the direction of the current rounding mode.
If x is NaN, a NaN shall be returned.
If x is +/-0 or +/-Inf, x shall be returned.
If the correct value would cause overflow, rint(), rintf(), and rintl() shall return the value of the macro +/-HUGE_VAL, +/-HUGE_VALF, and +/-HUGE_VALL (with the same sign as x), respectively.

Error Code

None.

See Also

abs, ceil, floor, nearbyint

8.13.47 round, roundf, roundl - round to the nearest integer value in a floating-point format

C Language Interface

```
#include <math.h>

double          round(double x);
float           roundf(float x);
long double     roundl(long double x);
```

Description

These functions shall round their argument to the nearest integer value in floating-point format, rounding halfway cases away from zero, regardless of the current rounding direction.

Return Parameter

Upon successful completion, these functions shall return the rounded integer value.
If x is NaN, a NaN shall be returned.
If x is +/-0 or +/-Inf, x shall be returned.
If the correct value would cause overflow, round(), roundf(), and roundl() shall return the value of the macro +/-HUGE_VAL, +/-HUGE_VALF, and +/-HUGE_VALL (with the same sign as x), respectively.

Error Code

None.

8.13.48 scanbln, scanblnf, scanblnl, scanbn, scanbnf, scanbnl - compute exponent using FLT_RADIX

C Language Interface

```
#include <math.h>

double          scalbln(double x, long n);
float           scalblnf(float x, long n);
long double     scalblnl(long double x, long n);
double          scalbn(double x, int n);
float           scalbnf(float x, int n);
long double     scalbnl(long double x, int n);
```

Description

These functions shall compute x * FLT_RADIXˆn efficiently, not normally by computing FLT_RADIXˆn explicitly ('ˆ' represents the power).

Return Parameter

Upon successful completion, these functions shall return x * FLT_RADIXˆn.

If the result would cause overflow, these functions shall return +/-HUGE_VAL, +/-HUGE_VALF, and +/-HUGE_VALL (according to the sign of x) as appropriate for the return type of the function.
If the correct value would cause underflow, and is not representable, 0.0 shall be returned.
If x is NaN, a NaN shall be returned.
If x is +/-0 or +/-Inf, x shall be returned.
If n is 0, x shall be returned.

Error Code

None.

8.13.49 sin, sinf, sinl - sine function

C Language Interface

```
#include <math.h>

double          sin(double x);
float           sinf(float x);
long double     sinl(long double x);
```

Description

These functions shall compute the sine of their argument x, measured in radians.

Return Parameter

Upon successful completion, these functions shall return the sine of x.
If x is NaN, a NaN shall be returned.
If x is +/-0, x shall be returned.
If x is subnormal, a range error may occur and x should be returned.
If x is +/-Inf, a NaN shall be returned.

Error Code

None.

See Also

cos, tan, asin

8.13.50 sinh, sinhf, sinhl - hyperbolic sine functions

C Language Interface

```
#include <math.h>

double          sinh(double x);
float           sinhf(float x);
long double     sinhl(long double x);
```

Description

These functions shall compute the hyperbolic sine of their argument x.

Return Parameter

Upon successful completion, these functions shall return the hyperbolic sine of x.
If the result would cause an overflow, +/-HUGE_VAL, +/-HUGE_VALF, and +/-HUGE_VALL (with the same sign as x) shall be returned as appropriate for the type of the function.
If x is NaN, a NaN shall be returned.
If x is +/-0 or +/-Inf, x shall be returned.
If x is subnormal, a range error may occur and x should be returned.

Error Code

None.

See Also

asinh, cosh, tanh

8.13.51 sqrt, sqrtf, sqrtl - square root function

C Language Interface

```
#include <math.h>

double          sqrt(double x);
float           sqrtf(float x);
long double     sqrtl(long double x);
```

Description

These functions shall compute the square root of their argument x.

Return Parameter

Upon successful completion, these functions shall return the square root of x.
For finite values of x < -0, a NaN shall be returned.
If x is NaN, a NaN shall be returned.
If x is +/-0 or +Inf, x shall be returned.
If x is -Inf, a NaN shall be returned.

Error Code

None.

8.13.52 tan, tanf, tanl - tangent function

C Language Interface

```
#include <math.h>

double          tan(double x);
float           tanf(float x);
long double     tanl(long double x);
```

Description

These functions shall compute the tangent of their argument x, measured in radians.

Return Parameter

Upon successful completion, these functions shall return the tangent of x.
If the correct value would cause underflow, and is not representable, 0.0 shall be returned.
If x is NaN, a NaN shall be returned.
If x is +/-0, x shall be returned.
If x is subnormal, x should be returned.
If x is +/-Inf, a NaN shall be returned.
If the correct value would cause overflow, tan(), tanf(), and tanl() shall return +/-HUGE_VAL, +/-HUGE_VALF, and +/-HUGE_VALL, respectively, with the same sign as the correct value of the function.

Error Code

None.

See Also

atan, sin, cos

8.13.53 tanh, tanhf, tanhl - hyperbolic tangent functions

C Language Interface

```
#include <math.h>

double          tanh(double x);
float           tanhf(float x);
long double     tanhl(long double x);
```

Description

These functions shall compute the hyperbolic tangent of their argument x.

Return Parameter

Upon successful completion, these functions shall return the hyperbolic tangent of x.
If x is NaN, a NaN shall be returned.
If x is +/-0, x shall be returned.
If x is +/-Inf, +/-1 shall be returned.
If x is subnormal, x should be returned.

Error Code

None.

See Also

atanh, tan

8.13.54 tgamma, tgammaf, tgammal - compute gamma() function

C Language Interface

```
#include <math.h>

double          tgamma(double x);
float           tgammaf(float x);
long double     tgammal(long double x);
```

Description

These functions shall compute the gamma() function of x.

Return Parameter

Upon successful completion, these functions shall return Gamma(x).
If x is a negative integer, a NaN shall be returned. On systems that support the IEC 60559 Floating-Point option, a NaN shall be returned.
If x is +/-0, tgamma(), tgammaf(), and tgammal() shall return +/-HUGE_VAL, +/-HUGE_VALF, and +/-HUGE_VALL, respectively.
If the correct value would cause overflow, tgamma(), tgammaf(), and tgammal() shall return +/-HUGE_VAL, +/-HUGE_VALF, or +/-HUGE_VALL, respectively, with the same sign as the correct value of the function.
If x is NaN, a NaN shall be returned.
If x is +Inf, x shall be returned.
If x is -Inf, a NaN shall be returned.

Error Code

None.

See Also

lgamma_r

8.13.55 trunc, truncf, truncl - round to truncated integer value

C Language Interface

```
#include <math.h>

double          trunc(double x);
float           truncf(float x);
long double     truncl(long double x);
```

Description

These functions shall round their argument to the integer value, in floating format, nearest to but no larger in magnitude than the argument.

Return Parameter

Upon successful completion, these functions shall return the truncated integer value.
If x is NaN, a NaN shall be returned.
If x is +/-0 or +/-Inf, x shall be returned.

Error Code

None.

8.13.56 y0, y1, yn - Bessel functions of the second kind

C Language Interface

```
#include <math.h>

double          y0(double x);
double          y1(double x);
double          yn(int n, double x);
```

Description

The y0(), y1(), and yn() functions shall compute Bessel functions of x of the second kind of orders 0, 1, and n, respectively.

Return Parameter

Upon successful completion, these functions shall return the relevant Bessel value of x of the second kind.
If x is NaN, NaN shall be returned.
If the x argument to these functions is negative, -HUGE_VAL or NaN shall be returned.
If x is 0.0, -HUGE_VAL shall be returned.
If the correct result would cause underflow, 0.0 shall be returned.
If the correct result would cause overflow, -HUGE_VAL or 0.0 shall be returned.

Error Code

None.

See Also

j0

8.13.57 lgamma_r, lgammaf_r, lgammal_r - log gamma function

C Language Interface

```
#include <math.h>

double          lgamma_r(double x, int* signp);
float           lgammaf_r(float x, int* signp);
long double     lgammal_r(long double x, int* signp);
```

Description

These functions shall compute the log( |gamma(x)| ).
The argument x need not be a non-positive integer ( is defined over the reals, except the non-positive integers).

Return Parameter

Upon successful completion, these functions shall return the logarithmic gamma of x.
The sign of gamma(x) is stored in the object pointed to by signp.

If x is a non-positive integer, lgamma_r(), lgammaf_r(), and lgammal_r() shall return +HUGE_VAL, +HUGE_VALF, and +HUGE_VALL, respectively.
If the correct value would cause overflow, lgamma_r(), lgammaf_r(), and lgammal_r() shall return +/-HUGE_VAL, +/-HUGE_VALF, and +/-HUGE_VALL (having the same sign as the correct value), respectively.
If x is NaN, a NaN shall be returned.
If x is 1 or 2, +0 shall be returned.
If x is +/-Inf, +Inf shall be returned.
ERROR
None.

See Also

exp

Additional Notes

These functions are thread-safe replacement of lgamma, lgammaf, and lgammal in the standard C library.
The static variable ″extern int signum″ does not exist in T2EX.
The sign obtained for signum is stored in the area specified by signp of each function instead.

## 8.14 netinet/in.h

The header netinet/in.h defines the following network communication-related macros, structures, and function prototype declarations.

Type

in_port_t
: Type used for port numbers of network communication.
Equivalent to uint16_t defined in stdint.h.

in_addr_t
: Type representing the IPv4 address of network communication.
Equivalent to uint32_t defined in stdint.h.

struct in_addr
: Structure representing the network communication address at the higher abstraction level than in_addr_t.
This structure needs to include at least the following members.

| | |
|---|---|
| in_addr_t | s_addr |

struct sockaddr_in
: Structure representing the socket address for the network communication.
This structure needs to include at least the following members.

| | | |
|---|---|---|
| sa_family_t | sin_family | Address family (AF_INET) |
| in_port_t | sin_port | Port number |
| struct in_addr | sin_addr | IP address |

The sin_port and sin_addr need to be stored in network byte order.

The following symbol constants are defined for use as level values of the getsockopt() and setsockopt() functions.

IPPROTO_IP
: Internet protocol

IPPROTO_ICMP
: Control message protocol

IPPROTO_RAW
: Raw IP packet protocol

IPPROTO_TCP
: TCP protocol

IPPROTO_UDP
: UDP protocol

The following symbol constants are defined for use asdestination addresses for the so_connect(), so_sendmsg(), and sendto() functions.

INADDR_ANY
: IPv4 local host address

INADDR_BROADCAST
: IPv4 broadcast address

The following symbol constant is defined as the size for storing the IPv4 address as a string.

INET_ADDRSTRLEN
: 16 Length of IPv4 address in a string format

## 8.15 search.h

The header search.h defines the following types and function prototype declarations for searching through various tables.

### Types

ENTRY type

The ENTRY type for structure entry which shall include the following members:

```
char            *key                    Search key string ending with a null character
void            *data                   Data corresponding to key
```

ACTION type

This value is for specifying the search operation and is defined as follows.

```
enum {
        FIND,                   /* Performs searching only */
        ENTER                   /* Inserted if not found */
} ACTION;
```

VISIT type

This value is for indicating the binary search state and is defined as follows.

```
enum {
        preorder,               /* First visit to the node */
        postorder,              /* Second visit to the node */
        endorder,               /* Third visit to the node */
        leaf                    /* This is not a node but a leaf */
} VISIT;
```

hsearch_data

Structure representing a hash table.
The contents are implementation-dependent.
This is defined as follows in the T2EX reference implementation.

```
struct hsearch_data {
        void    *htable;
        int     htablesize;
};
```

### Functions

#### 8.15.1 hcreate_r, hdestroy_r, hsearch_r - Hash table management

C Language Interface

```
#include <search.h>

int     hcreate_r(size_t nel, struct hsearch_data *htab);
void    hdestroy_r(struct hsearch_data* htab);
int     hsearch_r(ENTRY item, ACTION action, ENTRY **result, struct hsearch_data *htab);
```

Description

These functions manage the hash table.

The hcreate_r() allocates a sufficient area for the table and initializes the data htab for managing the hash table.
The contents of htab need to be initialized to 0 before calling hcreate_r() first time.
The application needs to guarantee that hcreate_r() has been called before using hsearch_r().
The nel is the estimated maximum number of entries included in the table.
This value may be modified to a larger value by the algorithm in order to get a specific, mathematically-convenient situation.

The hdestroy_r() releases the area which was created by hcreate_r() and allocated to the hash table htab.

Regarding the hash table specified by htab, the hsearch_r() searches for an item having the same key value as item.key. If the item is found, the hsearch_r() stores the pointer to that item in the area pointed to by result.
The item.data is the pointer to the data associated with item.key.
The strcmp() is used to compare keys.

If no item is found, the following behavior specified by ACTION is executed.
 - ENTER instructs to insert a copy of the item into an appropriate location of the table.
 - FIND instructs not to operate any entries.

Return Parameter

If successful, hcreate_r() returns a non-zero value.
Otherwise, it returns 0.

If successful, hsearch_r() returns a non-zero value.
Zero (0) is returned if the action is ENTER and the hash table is full, or the action is FIND and no item is found.

The hdestroy_r() does not return a value.

Error Code

None

See Also

bsearch, lsearch, tsearch

8.15.2 insque, remque - insert or remove an element in a queue

C Language Interface

```
#include <search.h>

void    insque(void *element, void *prev);
void    remque(void *element);
```

Description

The insque() and remque() functions shall manipulate queues built from doubly-linked lists. The queue can be either circular or linear. An application using insque() or remque() shall ensure it defines a structure in which the first two members of the structure are pointers to the same type of structure, and any further members are application-specific. The first member of the structure is a forward pointer to the next entry in the queue. The second member is a backward pointer to the previous entry in the queue. If the queue is linear, the queue is terminated with null pointers. The names of the structure and of the pointer members are not subject to any special restriction.

The insque() function shall insert the element pointed to by element into a queue immediately after the element pointed to by prev.

The remque() function shall remove the element pointed to by element from a queue.

If the queue is to be used as a linear list, invoking insque(&element, NULL), where element is the initial element of the queue, shall initialize the forward and backward pointers of element to null pointers.

If the queue is to be used as a circular list, the application shall ensure it initializes the forward pointer and the backward pointer of the initial element of the queue to the element's own address.

Return Parameter

The insque() and remque() functions do not return a value.

Error Code

None.

8.15.3 lfind, lsearch - linear search and update

C Language Interface

#include <search.h>

```
void    *lfind(const void *key, const void *base, size_t *nelp, size_t width, int (*compar)(const void
*, const void *));
void    *lsearch(const void *key, void *base, size_t *nelp, size_t width, int (*compar)(const void *,
const void *));
```

Description

The lsearch() function shall linearly search the table and return a pointer into the table for the matching entry. If the entry does not occur, it shall be added at the end of the table. The key argument points to the entry to be sought in the table. The base argument points to the first element in the table. The width argument is the size of an element in bytes. The nelp argument points to an integer containing the current number of elements in the table. The integer to which nelp points shall be incremented if the entry is added to the table.

The compar argument points to a comparison function which the application shall supply (for example, strcmp()). It is called with two arguments that point to the elements being compared. The application shall ensure that the function returns 0 if the elements are equal, and non-zero otherwise.

The lfind() function shall be equivalent to lsearch(), except that if the entry is not found, it is not added to the table.

Return Parameter

If the searched for entry is found, both lsearch() and lfind() shall return a pointer to it.
Otherwise, lfind() shall return a null pointer and lsearch() shall return a pointer to the newly added element.
Both functions shall return a null pointer in case of error.

Error Code

None.

See Also

bsearch, hsearch_r, tsearch

8.15.4 tdelete, tfind, tsearch, twalk - manage a binary search tree

C Language Interface

#include <search.h>

```
void   *tdelete(const void *key, void **rootp, int(*compar)(const void *, const void *));
void   *tfind(const void *key, void *const *rootp, int(*compar)(const void *, const void *));
void   *tsearch(const void *key, void **rootp, int(*compar)(const void *, const void *));
void    twalk(const void *root, void (*action)(const void *nodep, VISIT which, int depth));
```

Description

The tdelete(), tfind(), tsearch(), and twalk() functions manipulate binary search trees. Comparisons are made with a user-supplied routine, the address of which is passed as the compar argument. This routine is called with two arguments, which are the pointers to the elements being compared. The application shall ensure that the user-supplied routine returns an integer less than, equal to, or

greater than 0, according to whether the first argument is to be considered less than, equal to, or greater than the second argument.

The tsearch() function shall build and access the tree. The key argument is a pointer to an element to be accessed or stored. If there is a node in the tree whose element is equal to the value pointed to by key, a pointer to this found node shall be returned.
Otherwise, the value pointed to by key shall be inserted (that is, a new node is created and the value of key is copied to this node), and a pointer to this node returned. Only pointers are copied, so the application shall ensure that the calling routine stores the data. The rootp argument points to a variable that points to the root node of the tree. A null pointer value for the variable pointed to by rootp denotes an empty tree; in this case, the variable shall be set to point to the node which shall be at the root of the new tree.

Like tsearch(), tfind() shall search for a node in the tree, returning a pointer to it if found. However, if it is not found, tfind() shall return a null pointer. The arguments for tfind() are the same as for tsearch().

The tdelete() function shall delete a node from a binary search tree. The arguments are the same as for tsearch(). The variable pointed to by rootp shall be changed if the deleted node was the root of the tree. The tdelete() function shall return a pointer to the parent of the deleted node, or an unspecified non-null pointer if the deleted node was the root node, or a null pointer if the node is not found.

If tsearch() adds an element to a tree, or tdelete() successfully deletes an element from a tree, the concurrent use of that tree in another thread, or use of pointers produced by a previous call to tfind() or tsearch(), produces undefined results.

The twalk() function shall traverse a binary search tree. The root argument is a pointer to the root node of the tree to be traversed. (Any node in a tree may be used as the root for a walk below that node.)

The argument action is the name of a routine to be invoked at each node. This routine is, in turn, called with three arguments. The first argument shall be the address of the node being visited. The structure pointed to by this argument is unspecified and shall not be modified by the application, but it shall be possible to cast a pointer-to-node into a pointer-to-pointer-to-element to access the element stored in the node.

The second argument nodep shall be a value from an enumeration data type:

```
typedef enum { preorder, postorder, endorder, leaf } VISIT;
```

(defined in <search.h>), depending on whether this is the first, second, or third time that the node is visited (during a depth-first, left-to-right traversal of the tree), or whether the node is a leaf.

The third argument depth shall be the level of the node in the tree, with the root being level 0.

If the calling function alters the pointer to the root, the result is undefined.

If the functions pointed to by action or compar (for any of these binary search functions) change the tree, the results are undefined.

These functions are thread-safe only as long as multiple threads do not access the same tree.

Return Parameter

If the node is found, both tsearch() and tfind() shall return a pointer to it. If not, tfind() shall return a null pointer, and tsearch() shall return a pointer to the inserted item.

A null pointer shall be returned by tsearch() if there is not enough space available to create a new node.

A null pointer shall be returned by tdelete(), tfind(), and tsearch() if rootp is a null pointer on entry.
The tdelete(), tfind(), and tsearch() return NULL if rootp is NULL.

The tdelete() function shall return a pointer to the parent of the deleted node, or an unspecified non-null pointer if the deleted node was the root node, or a null pointer if the node is not found.

The twalk() function shall not return a value.

Error Code

None.

See Also

bsearch, hsearch_r, lsearch

8.16 stdarg.h

The header stdarg.h defines a set of macros that enables programmers to describe portable functions which accept variable argument lists.
If a function with variable number of arguments (e.g. printf()) is created without using the macros defined here, it may lose portability because the variable argument list may not be correctly referenced depending on the argument passing convention used on different systems.

Type

va_list
The type that represents the argument list for the function with a variable number of arguments.

The following macros are defined.

void va_start(va_list ap, argN);
The va_start() macro initializes ap used in the va_arg() and va_end() macros.
Therefore, this must be called before va_arg() and va_end().
The ap is the variable used for processing the argument list.
This is initialized by the va_start() macro.
The argN is the argument immediately before the list of variable number arguments represented by "...".
The va_start() macro must not be called for reinitializing its ap, without calling the va_end() macro for ap.

type va_arg(va_list ap, type);
The va_arg() macro returns a single argument which has a type specified by type from ap.
when va_arg(ap, type) is called for the first time after the va_start(), argument following argN is returned.

Every time the va_arg() macro is called, ap is updated. At the next call, the argument after that is returned.
The ap is the variable used for processing the argument list.
This needs to be initialized by the va_start() or va_copy() macro.
The type specifies the data type of the argument to be taken out.
The name of the type must show the type of a pointer to the object of that type by simply appending "*" at the end of the type name.

If the next argument does not actually exist or the type is not compatible with the type of the next actual argument, the behavior shall be undefined except the following cases.
- One type is a signed integer, the other is the corresponding unsigned integer, and the value can be represented in either of the types.
- One type is a pointer to void and the other is a pointerto the character type.

void va_copy(va_list dest, va_list src);
The va_copy() macro initializes dest as a copy of src.
The va_start macro is applied to dest and then va_arg is applied in the same manner as va_arg is applied to src until src becomes the current state.
Do not simply assign dest = src;.

Both the va_copy() and va_start() macros must not be called for reinitializing its dest, without calling the va_end() macro for dest.

void va_end(va_list ap);
The va_end() macro is used for cleanup.
This keeps ap illegal until it is reinitialized by calling va_start() or va_copy() macro again.
Each call of the va_start() and va_copy() macros must correspond one-to-one with the call of the corresponding va_end() macro in the same function.
The va_start() ... va_end() can be used in a nested-manner..

Sample usage:

```
void func1(int narg, ...)
{
        va_list ap;
        int             i;

        va_start(ap, narg);
        for (i = 0; i < narg; i++) {
```

```
                xxx = va_arg(ap, type);
                ...
        }
        va_end(ap);
}
```

## 8.17 stdbool.h

The header stdbool.h defines the macro for handling the boolean type (logical type).

Macros

bool
: Expands to _Bool.

true
: Expands to the integer constant 1.

false
: Expands to the integer constant 0.

__bool_true_false_are_defined
: Expands to the integer constant 1.

## 8.18 stddef.h

The header stddef.h defines the following commonly used constants, macros, and types.

### Constants

NULL
: Null pointer constant. The macro shall expand to an integer constant expression with the value 0 cast to type void *.

### Macros

offsetof(type, member-designator)
: Integer constant expression of type size_t, the value of which is the offset in bytes to the structure member (member-designator), from the beginning of its structure (type).

### Types

ptrdiff_t
: Signed integer type of the result of subtracting two pointers.

wchar_t
: Integer type whose range of values can represent distinct codes for all members of the largest extended character set specified among the system locales.
For instance, multi-byte characters are used in UTF-8 which represents the Unicode as a byte sequence. This type means the integer type capable of representing this maximum number of bytes.
The null character shall have the code value zero.
This is the type when each element in a character set (which may consist of multiple bytes in UTF-8) is used as a one-character integer character constant and is called "wide character type".

size_t
: Unsigned integer type of the result of the sizeof operator.

## 8.19 stdint.h

The header stdint.h defines the types regarding integers with a specific width and macros for their limit values.

### Integer Types

Exact-width integer types:

int8_t
int16_t
int32_t
int64_t
: 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.

uint8_t
uint16_t
uint32_t
uint64_t
: The typedef name uintN_t designates an unsigned integer type with width N. Thus, uint32_t denotes an unsigned integer type with a width of exactly 32 bits.

Minimum-width integer types:

int_least8_t
int_least16_t
int_least32_t
int_least64_t
: 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.

uint_least8_t
uint_least16_t
uint_least32_t
uint_least64_t
: 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.

Fastest minimum-width integer types:

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

int_fast8_t
int_fast16_t
int_fast32_t
int_fast64_t
: The typedef name int_fastN_t designates the fastest signed integer type with a width of at least N.

uint_fast8_t
uint_fast16_t
uint_fast32_t
uint_fast64_t
: The typedef name uint_fastN_t designates the fastest unsigned integer type with a width of at least N.

Integer types capable of holding object pointers:

intptr_t
: This 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 a pointer to void, and the result will compare equal to the original pointer.

uintptr_t
: This 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.

Greatest-width integer types:

intmax_t
This type designates a signed integer type capable of representing any value of any signed integer type.

uintmax_t
This type designates an unsigned integer type capable of representing any value of any unsigned integer type.

Macros

Limits of exact-width integer types:

INT8_MIN
INT16_MIN
INT32_MIN
INT64_MIN
Minimum values of exact n-bit width signed integer types.
-(2 ^ (n-1)) for INTn_MIN. ('^' represents the power.)

INT8_MAX
INT16_MAX
INT32_MAX
INT64_MAX
Maximum values of exact n-bit width signed integer types.
2 ^ (n-1) - 1 for INTn_MAX. ('^' represents the power.)

UINT8_MAX
UINT16_MAX
UINT32_MAX
UINT64_MAX
Maximum values of exact n-bit width unsigned integer types.
(2 ^ n) - 1 for UINTn_MAX. ('^' represents the power.)

Limits of minimum-width integer types:

INT_LEAST8_MIN
INT_LEAST16_MIN
INT_LEAST32_MIN
INT_LEAST64_MIN
Minimum values of minimum n-bit width signed integer types.
-(2 ^ (n-1)) for INT_LEASTn_MIN. ('^' represents the power.)

INT_LEAST8_MAX
INT_LEAST16_MAX
INT_LEAST32_MAX
INT_LEAST64_MAX
Maximum values of minimum n-bit width signed integer types.
2 ^ (n-1) - 1 for INT_LEASTn_MAX. ('^' represents the power.)

UINT_LEAST8_MAX
UINT_LEAST16_MAX
UINT_LEAST32_MAX
UINT_LEAST64_MAX
Maximum values of minimum n-bit width unsigned integer types.
(2 - n) - 1 for UINT_LEASTn_MAX. ('^' represents the power.)

Limits of fastest minimum-width integer types:

INT_FAST8_MIN
INT_FAST16_MIN
INT_FAST32_MIN
INT_FAST64_MIN
Minimum values of fastest minimum n-bit width signed integer types:
-(2 ^ (n-1)) for INT_FASTn_MIN. ('^' represents the power.)

INT_FAST8_MAX
INT_FAST16_MAX

INT_FAST32_MAX
INT_FAST64_MAX
Maximum values of fastest minimum n-bit width signed integer types.
(2 ^ (n-1)) - 1 for INT_FASTn_MAX. ('^' represents the power.)

UINT_FAST8_MAX
UINT_FAST16_MAX
UINT_FAST32_MAX
UINT_FAST64_MAX
Maximum values of fastest minimum n-bit width unsigned integer types.
(2 ^ n) - 1 for UINT_FASTn_MAX. ('^' represents the power.)

Limits of integer types capable of holding object pointers:

INTPTR_MIN
Minimum value of pointer-holding signed integer type.
-(2 ^ 31) ('^' represents the power.)

INTPTR_MAX
Maximum value of pointer-holding signed integer type.
(2 ^ 31) - 1 ('^' represents the power.)

UINTPTR_MAX
Maximum value of pointer-holding unsigned integer type.
(2 ^ 32) - 1 ('^' represents the power.)

Limits of greatest-width integer types:

INTMAX_MIN
Minimum value of greatest-width signed integer type.
-((2 ^ 63) ('^' represents the power.)

INTMAX_MAX
Maximum value of greatest-width signed integer type.
(2 ^ 63) - 1 ('^' represents the power.)

UINTMAX_MAX
Maximum value of greatest-width unsigned integer type.
(2 ^ 64) - 1 ('^' represents the power.)

Limits of Other Integer Types:

PTRDIFF_MIN
Minimum limits of ptrdiff_t type.
INT32_MIN

PTRDIFF_MAX
Maximum limits of ptrdiff_t type.
INT32_MAX

SIZE_MAX
Maximum limits of size_t type.
UINT32_MAX

WCHAR_MIN
Minimum limits of wchar_t type.
0

WCHAR_MAX
Maximum limits of wchar_t type.
UINT32_MAX

## 8.20 stdio.h

The header stdio.h defines types, macros, and function prototype declarations for the following standard input/output:

While the standard C library can handle files and sockets uniformly as I/O stream, the standard I/O of the T2EX standard C compatible library handles only files as I/O stream and does not handle sockets.

### Type

FILE
The type of structure to manage a file.

Its elements are implementation-dependent. Usually, it contains a file descriptor, pointer to memory area to buffer data for read/write operation, file position to read/write (different from the file offset), flag to indicate that the end of file is reached (end-of-file flag), area to record an error number when errors occur during I/O (error information), and all other information required to control the file I/O stream.
Elements in the structure shall be accessed via functions described in this section and not directly be accessed by users.

```
typedef struct {
        /* Implementation-dependent */
} FILE;
```

For T2EX reference implementation, the implementation-dependent part is defined as int opaque[23]; and its elements shall not be directly accessed by users.

Sequential input and output for a file are called as a stream and specified by "FILE*".

errno_t
An integer type representing an error number.

fpos_t
A type representing a position in a file.
With T2EX reference implementation, it is a 32-bit signed integer type, but non-integer type implementation is also possible.

fpos64_t
A type representing a position in a file using 64-bit.
With T2EX reference implementation, it is a 64-bit signed integer type, but non-integer type implementation is also possible.

off_t
An integer type representing a file size in 32-bit.

off64_t
An integer type representing a file size in 64-bit.

size_t
An unsigned integer type representing the result of sizeof operator.

ssize_t
A type representing 0 or more number of bytes or a negative error code.

va_list
A type representing the argument list for the function with a variable number of arguments.

### Macro

BUFSIZ
The size of I/O buffer (in bytes).

Macros indicating the buffering status of the file I/O:

| | |
|---|---|
| _IOFBF | Indicates that the I/O is completely buffered. |
| _IOLBF | Indicates that the I/O is buffered line by line. |

```
        _IONBF          Indicates that the I/O is not buffered.

Macros indicating a reference position when the current offset of the file is moved:

        SEEK_CUR        Indicates that the starting point of the move is the current position of the
file.
        SEEK_END        Indicates that the starting point of the move is the end of the file.
        SEEK_SET        Indicates that the starting point of the move is the beginning of the file.


FILENAME_MAX
                The maximum string length of a file name (in bytes).

FOPEN_MAX
                The maximum number of files that can be opened at the same time.

EOF
                The end of a file.


The following three values of FILE* type are defined for the console I/O:

        stdin                   The standard console input.
        stdout                  The standard console output.
        stderr                  The standard console error output.



Function
```

### 8.20.1 libc_stdio_init - Initializes standard I/O

C Language Interface

```
#include <stdio.h>

ER      libc_stdio_init(void)
```

Description

It initializes the standard input/output library.
After calling fs_main() of the file management function, this API call must explicitly be called.
This API call initializes stdin, stdout, and stderr so that they can immediately be used.
This API call must be called before using other standard input/output library.

Return Parameter

Error Code

Return code is an error code.

| | |
|---|---|
| E_OK | Normal completion |

See Also

libc_stdio_cleanup

Additional Notes

libc_stdio_init() is a T2EX-specific API call.

### 8.20.2 libc_stdio_cleanup - Terminates standard I/O

C Language Interface

#include <stdio.h>

ER libc_stdio_cleanup(void)

Description

It terminates the standard input/output library.
Before terminating the file management function by fs_main(), this API must explicitly be called.

Return Parameter

Error Code

Return code is an error code.
E_OK Normal completion

See Also

libc_stdio_init

Additional Notes

libc_stdio_cleanup() is a T2EX-specific API call.

### 8.20.3 clearerr - Clears the error number of stream

C Language Interface

#include <stdio.h>

void clearerr(FILE* stream);

Description

clearerr() clears the end-of-file indicator and error information pointed by "stream".

Return Parameter

The clearerr() does not return a value.

Error Code

None

See Also

feof, ferror, fileno, fopen

### 8.20.4 feof - Checks the end of file

C Language Interface

#include <stdio.h>

int feof(FILE* stream);

Description

feof() checks whether or not a stream pointed by "stream" has reached the end of file.
If the "stream" has reached the end of file, feof() returns a non-zero value.

Return Parameter

If the "stream" has reached the end of file, feof() returns a non-zero value.

Error Code

None

See Also

clearerr, ferror, fileno, fopen

### 8.20.5 ferror - Tests error status of stream

C Language Interface

```
#include <stdio.h>

errno_t ferror(FILE* stream);
```

Description

ferror() checks whether or not the stream pointed by "stream" is in the error state.
If an error is recorded in the "stream", ferror() returns its error number.

Return Parameter

If an error is recorded in the "stream", ferror() returns its error number.

Error Code

None

See Also

clearerr, feof, fileno, fopen

### 8.20.6 fileno - Gets the file descriptor of stream

C Language Interface

```
#include <stdio.h>

int     fileno(FILE* stream);
```

Description

fileno() returns the value of the file descriptor corresponding to the stream pointed by "stream".

Return Parameter

If successful, fileno() returns the file descriptor value (non-negative) of "stream".
If an error occurs, it records the error number in the stream and returns -1.

Error Code

If an error occurs, calling ferror() may return the followings:

EBADF "stream" is invalid

See Also

clearerr, feof, ferror, fopen

### 8.20.7 fgetc, getc, getchar - Reads one character from stream

C Language Interface

```
#include <stdio.h>

int     fgetc(FILE* stream);
int     getc(FILE *stream);
int     getchar(void);
```

Description

fgetc() gets the next character (1 byte) from the stream pointed by "stream" as unsigned char type and returns it after converting it to int type.

If the stream has reached the end of file, it returns -1.

getc() is equivalent to fgetc() and is implemented as a macro.
Therefore, "stream" may be evaluated more than once.

getchar() is equivalent to getc(stdin).

Return Parameter

If successful, fgetc(), getc(), or getchar() returns one character read from the current file position of the stream.
If the stream has reached the end of file, it sets the end-of-file indicator in the stream and returns EOF.
If an error occurs, it records the error number in the stream and returns EOF.

Error Code

If an error occurs, calling ferror() may return the followings:

EAGAIN Since O_NONBLOCK flag of the file descriptor for the stream is set and writing will have caused a wait, the function returned immediately.
EBADF The file descriptor of the stream is not a correct file descriptor opened for read
EINTR Aborted by fs_break()
EIO I/O error

See Also

feof, ferror, fgets, fread, ungetc, fopen

### 8.20.8 fgets - Reads one line from stream

C Language Interface

```
#include <stdio.h>

char*   fgets(char* s, int size, FILE* stream);
```

Description

fgets() reads a string from "stream" for "size" -1 bytes or until it detects a new line ('¥n') or the end of file, and stores the string read in the area pointed by "s" and appends a null character to the end of the string.

Return Parameter

If successful, fgets() returns s.
If the stream has reached the end of file, it sets the end-of-file indicator in the stream and returns NULL.
If an error occurs, it records the error number in the stream and returns NULL.

Error Code

See fgetc().

See Also

feof, ferror, fgetc, fread, ungetc, fopen

### 8.20.9 ungetc - Pushes one character back to input stream

C Language Interface

```
#include <stdio.h>

int     ungetc(int c, FILE* stream);
```

Description

ungetc() pushes a character (1 byte) converted from "c" to unsigned char back to the input stream pointed to by "stream".

When the pushed back characters are read again, they are read in the reverse order of the push back.

When fseek(), fsetpos(), or rewind() is executed on this stream, the pushed back character is discarded.

Even if data is pushed back, data on the disk of the file corresponding to the stream does not change.

While pushing back one character is guaranteed to succeed, whether or not the subsequent push back is successful is implementation-dependent.
In the T2EX reference implementation, any number of characters can be pushed back unless the memory is exhausted.

If the value of "c" is EOF, the state of the stream does not change and ungetc() fails.

If successful, the end-of-file indicator in the stream is cleared.
The file position in the stream is decremented by 1 every time the push back is performed.
If the file position is 0 before calling ungetc(), the resulting file position is undetermined.

Return Parameter

If successful, ungetc() returns the value of "c".
If failed, it returns EOF.

Error Code

None

See Also

fseek, fgetc, fopen, fsetpos, rewind, setbuf

8.20.10 fputc, putc, putchar - Outputs one character to stream

C Language Interface

```
#include <stdio.h>

int     fputc(int c, FILE* stream);
int     putc(int c, FILE* stream);
int     putchar(int c);
```

Description

fputc() writes "c" converted to an unsigned char to the output stream pointed to by "stream".
If the file position is defined on the stream, the writing takes placein that position,
and the file position is icremented by one.

If a file does not support the positioning or a stream is opened with the append mode, the character is written at the end of the stream.

putc() is equivalent to fputc() and is implemented as a macro.
Therefore, "stream" may be evaluated more than once.

putchar() is equivalent to putc(c, stdout).

Return Parameter

If successful, fputc(), putc(), or putchar() returns the value of the written character.
If an error occurs, it records the error number in the stream and returns EOF.

Error Code

If an error occurs, calling ferror() may return the followings:

| | |
|---|---|
| EAGAIN | Since O_NONBLOCK flag of the file descriptor for the stream is set and writing will have caused a wait, the function returned immediately. |
| EBADF | File descriptor corresponding to the stream is invalid |
| EFBIG | Position exceeds the limit of the file size |
| EINTR | Aborted by fs_break() |
| EIO | I/O error |
| ENOSPC | Insufficient device space |

See Also

ferror, fopen, fputs, setbuf

8.20.11 fputs, puts - Outputs string to stream

C Language Interface

```
#include <stdio.h>

int     fputs(const char* s, FILE* stream);
int     puts(const char* s);
```

Description

fputs() writes a string ending with a null character, pointed to by "s", to the output stream pointed to by "stream".
The last null character is not written.

puts() is equivalent to fputs(s, stdout).

Return Parameter

If successful, fputs(), puts() returns 0 or a positive value.
If an error occurs, it records the error number in the stream and returns EOF.

Error Code

See fputc()

See Also

fputc, fopen, ferror

### 8.20.12 fgetpos, fgetpos64 - Gets the current file position

C Language Interface

```
#include <stdio.h>

int     fgetpos(FILE* stream, fpos_t* pos);
int     fgetpos64(FILE* stream, fpos64_t* pos);
```

Description

fgetpos() stores the current file position of the stream pointed to by "stream" in the area pointed to by "pos".

fgetpos64() uses fpos64_t type for arguments to handle the 64-bit file position.

Return Parameter

If successful, fgetpos(), fgetpos64() returns 0.
If an error occurs, it records the error number in the stream and returns a non-zero value.

Error Code

If an error occurs, calling ferror() may return the followings:

| | |
|---|---|
| EOVERFLOW | The current position cannot be represented by fpos_t |
| EBADF | The file descriptor used for the stream is invalid |

See Also

fopen, fseek, ftell, rewind, ungetc

### 8.20.13 fsetpos, fsetpos64 - Sets the current file position

C Language Interface

```
#include <stdio.h>

int     fsetpos(FILE* stream, const fpos_t* pos);
int     fsetpos64(FILE* stream, const fpos64_t* pos); // Non-standard function
```

Description

fsetpos() sets the current file position of the stream specified by "stream" to the value pointed to by "pos".

"pos" is a value acquired by a previously executed fgetpos().

If successful, it clears the end-of-file indicator in the stream and discards the character pushed back by ungetc().

fsetpos64() uses fpos64_t for its arguments and the value of "pos" is acquired by fgetpos64().

Return Parameter

If successful, fsetpos() or fsetpos64() returns 0.
If an error occurs, it records the error number in the stream and returns a non-zero value.

Error Code

If an error occurs, calling ferror() may return the followings:

EAGAIN Since O_NONBLOCK flag of the file descriptor for the stream is set and writing will have caused a wait, the function returned immediately.
EBADF File descriptor for the stream is invalid
EFBIG Position exceeds the limit of file size
EINTR Aborted by fs_break()
EIO I/O error
ENOSPC Insufficient device space

See Also

fopen, fseek, ftell, rewind, ungetc

## 8.20.14 fseek, fseek64, rewind, ftell, ftell64 - Changes or gets the current file position

C Language Interface

```
#include <stdio.h>

int     fseek(FILE* stream, long offset, int whence);
int     fseek64(FILE* stream, int64_t offset, int whence);
void    rewind(FILE* stream);
long    ftell(FILE* stream);
int64_t ftell64(FILE* stream);
```

Description

fseek() changes the file position of the stream pointed to by "stream".
The new position of the stream in bytes is "offset" added to the position specified by "whence".
whence can have the following values specifying the positions shown on the right:

| | |
|---|---|
| SEEK_SET | Start position of the file |
| SEEK_CUR | Current position of the file |
| SEEK_END | End position of the file |

If successful, it clears the end-of-file indicator in the stream and discards the data pushed back by ungetc().

Attempting to set a file position exceeding the end position of the file causes an error.
In this case, ferror(stream) returns EINVAL.

For non-seekable file such as console whose file position is not defined, fseek() causes an error. In this case, ferror(stream) returns ESPIPE.

fseek64(), whose "offset" to specify a position is an int64_t type, is equivalent to fseek() except that it can specify a position using a 64-bit value.

rewind() is equivalent to (void) fseek(stream, 0, SEEK_SET).
Therefore it sets the file position of the stream to the start of the file.
rewind() also clears the error information of the stream.

ftell() returns the current file position of the stream pointed to by "stream".

ftell64() is equivalent to ftell() except that its return code is a 64-bit value.

Return Parameter

If successful, fseek() or fseek64() returns 0.
If an error occurs, it records the error number in the stream and returns -1.

rewind() does not return a value.

If successful, ftell() or ftell64() returns the current file position.
If an error occurs, it records the error number in the stream and returns -1.

Error Code

If an error occurs, calling ferror() may return the followings:

| | |
|---|---|
| EAGAIN | Since O_NONBLOCK flag of the file descriptor for the stream isset and writing will have caused a wait, the function returned immediately. |
| EBADF | File descriptor corresponding to the stream is invalid |
| EFBIG | Position exceeds the limit of file size |
| EINVAL | Specified position is invalid<br>- Resulting file position is negative<br>- Position exceeding the end of file |
| EINTR | Aborted by fs_break() |
| EIO | I/O error |
| ESPIPE | Non-seekable file |
| EOVERFLOW | File position cannot be represented by the resulting data |

ftell() and ftell64() may return only EBADF and EOVERFLOW.

See Also

fgetpos, fsetpos, ftell, rewind, ungetc

### 8.20.15 fread - Input from binary stream

C Language Interface

```
#include <stdio.h>

size_t  fread(void* ptr, size_t size, size_t nmemb, FILE* stream);
```

Description

fread() reads "nmemb" elements of data, size of which is "size" bytes each, from the stream pointed to by "stream" into the area pointed to by "ptr".
The file position specified by "stream" is advanced by the number of bytes of the data that has been read.

Return Parameter

fread() returns the number of elements that have been read successfully.
If an read error occurs or EOF is reached, this value will be smaller than "nmemb".
If "size" or "nmemb" is 0, it returns 0 and the state of "stream" does not change.
If an error occurs, it records the error number in the stream.

Error Code

See fgetc().

See Also

fopen, fscanf, fgetc, feof, ferror

### 8.20.16 fwrite - Output to binary stream

C Language Interface

```
#include <stdio.h>

size_t  fwrite(const void* ptr, size_t size, size_t nmemb, FILE* stream);
```

Description

fwirte() writes "nmemb" elements of data, each size of which is "size" bytes, from the area pointed to by "ptr" to the stream pointed to by "stream".
The position of the file pointed to by "stream" is advanced by the number of bytes of the data that has been written.

Return Parameter

fwrite() returns the number of elements that have been written successfully.
If a write error occurs, this value will be smaller than "nmemb".
If "size" or "nmemb" is 0, it returns 0 and the state of "stream" does not change.
If a write error occurs, it records the error number in the stream.

Error Code

See fputc().

See Also

fopen, fprintf, fputc, ferror

### 8.20.17 fflush - Flushes stream

C Language Interface

```
#include <stdio.h>

int     fflush(FILE* stream);
```

Description

fflush() writes (flushes) unwritten data to a file if any data is not output yet to the output stream specified by "stream".
If "stream" is NULL, fflush() flushes all opened output streams.

Return Parameter

If successful, fflush() returns 0.
If an error occurs, it records the error number in the stream and returns EOF.

Error Code

If an error occurs, calling ferror() may return the followings:

| | |
|---|---|
| EAGAIN | Since O_NONBLOCK flag of the file descriptor for the stream is set and writing will have caused a wait, the function returned immediately. |
| EBADF | File descriptor for the stream is invalid |
| EFBIG | Position exceeds the limit of file size |
| EINTR | Aborted by fs_break() |
| EIO | I/O error |
| ENOSPC | Insufficient device space |

See Also

setbuf, fopen

8.20.18 fprintf, printf, snprintf, sprintf - Formatted output

C Language Interface

```
#include <stdio.h>

int     fprintf(FILE* stream, const char* format, ...);
int     printf(const char* format, ...);
int     snprintf(char* str, size_t size, const char* format, ...);
int     sprintf(char* str, const char* format, ...);
```

Description

fprintf() converts data according to the format pointed to by "format" and writes it to the output stream specified by "stream".
In printf(), output stream is fixed to the standard output.
sprintf() writes output as a string ending with a null character to the area pointed to by "str".
Users must guarantee that the area pointed to by "str" has a large enough size.
snprintf() is equivalent to sprintf() and "size" represents the number of bytes of the buffer pointed to by "str".
If "size" is 0, output is not performed and "str" can be NULL.
Otherwise, output bytes exceeding "size" - 1 are discarded and a null character is appended at the end of characters actually written to buffer pointed to by "str".
In the output of sprintf() or snprintf(), the result is undefined when copying of the overlapped area occurs.
(For example, when the same area is referenced by both the output destination and argument, etc.,)

fprintf(), printf(), snprintf(), or sprintf() converts subsequent arguments according to the format pointed to by "format" and outputs them.

"format" consists of 0 or more directives:
They are normal characters that are simply copied to the output stream, and conversion specifications, each of which directs the retrieval of 0 or more arguments.

The result is undefined when the number of arguments for "format" is insufficient.
If "format" is used up while any arguments still remain, the extra arguments are only evaluated when these API calls are executed and ignored in these API calls.

If the format string contains any conversion specification with % format, each conversion specification uses the first unused argument in the argument list.

Each conversion specification begins with '%' character, followed by the following items listed in 1 to 5 in this order:

Format of conversion specification

```
%[flag][minimum field width][.precision][length modifier]conversion specifier
```

1. Flags (unordered, optional)
  Flag is a character to modify the meaning of 0 or more conversion specifications.

2. Minimum field width (optional)
  The minimum field width is a decimal integer string or <asterisk> ('*') to specify the minimum field width.
  If the converted value is smaller than the minimum field width, left side is filled with <white-space> characters by default.
  If the left-alignment flag ('-') is given to the minimum field width, right side is filled with <white-space> characters.
  If the minimum field width is specified as the <asterisk> ('*'), an argument with an int type is used to specify the minimum field width.
  If an argument for the minimum field width is negative, positive field width with '-' flag is assumed.

3. Precision (optional)
  Precision specifies a <period>('.') followed by a decimal numeric string or an <asterisk> ('*').
  Decimal numeric string is optional and is assumed to be 0 when it is omitted.
  The behavior shall be undefined when precision is used with other conversion specifications.
  If precision is specified as <asterisk> ('*'), an argument of an int type is used to specify the precision.
  If an argument for precision is negative, the precision is assumed to be omitted.

4. Length modifier
  Length modifier is a character to specify the length of the type indicated by the conversion specifier.

5. Conversion specifier
  Conversion specifier is a character to specify the type of conversion to apply.

When the field width and precision are specified by the <asterisk> ('*'), the arguments corresponding to the minimum field width and precision must be provided in the order of the field width and precision before the arguments to be converted by the conversion specification.

Flag characters and their meanings:

-
	The conversion result is left-justified in the field.
	If this flag is not specified, the conversion result is right-justified in the field.

+
	The signed conversion result shall always begin with a sign ('+' or '-').
	If this flag is not specified, the conversion result is signed only when it is a negative value.

<Space>
	If the first character of the signed conversion is not signed, or the result of the signed conversion is empty, a <space> is prefixed to the result.
This means that if both a <space> and '+' flags are specified, the <space> flag shall be ignored.

#
	Specifies that the value is converted to an alternate form.
	For o conversion specifier:
		When the first character of the conversion result is not 0, 0 is added.
		Increase the precision if necessary.
	For x or X conversion specifier:
		If the conversion result is non-zero, 0x or 0X (for X conversion) is prefixed to the beginning of the result.
	For a, A, e, E, f, F, g, or G conversion specifier:
		Even if a number does not exist after the decimal point, the decimal point is always output.
	For g or G conversion specifier:
		The trailing 0 is not deleted from the conversion result.
	The behavior is undefined for other conversion specifiers.

0
	For d, i, o, u, x ,X, a, A, e, E, f, F, g, or G conversion, except for infinity or NaN conversion, leading zeros (following a sign or radix representation) are used instead of spaces to fill the field width.
	If both '0' and '-' flags are used, '0' flag is ignored.
	If precision is specified for d, i, o, u, x, or X conversion, '0' flag is ignored.
	The behavior is undefined for other conversion specifications.

Precision and their meanings:

	For d, i, o, u, x, or X conversion specifier:
		Minimum number of digits to output.
		The default is 1.

	For a, A, e, E, f, or F conversion specifier:
		The number of digits in decimal point part.
		The default is 6.

	For g or G conversion specifier:
		Maximum number of significant digits.
		The default is 6.

	For s conversion specifier:
		Maximum number of characters to output (in bytes).

Length modifiers and their meanings:

hh
	If d, i, o, u, x, or X conversion specifier follows, it specifies that the argument to convert

is a signed char or unsigned char.
(Though the integer argument has been promoted according to the argument promotion rule, the value is converted to signed char or unsigned char type prior to the formatting.)
If n conversion specification follows, it indicates that the argument to convert is a pointer to a signed char.

h
If d, i, o, u, x, or X conversion specifier follows, it specifies that the argument to convert is a short or unsigned short.
(Though the integer argument has been promoted according to the argument promotion rule, the value is converted to short or unsigned short type prior to the formatting.)
If n conversion specification follows, it specifies that the argument to convert is a pointer to a short argument.

l
If d, i, o, u, x, or X conversion specifier follows, it specifies that the argument to convert is a long or unsigned long.
If n conversion specification follows, it specifies that the argument to convert is a pointer to a long argument.
If a, A, e, E, f, F, g, or G conversion specification follows, this modifier is ignored.

ll
If d, i, o, u, x, or X conversion specifier follows, it specifies that the argument to convert is a long long or unsigned long long.
If n conversion specification follows, it specifies that the argument to convert is a pointer to a long long argument.

j
If d, i, o, u, x, or X conversion specifier follows, it specifies that the argument to convert is an intmax_t or uintmax_t.
If n conversion specification follows, it indicates that the argument to convert is a pointer to an intmax_t argument.

z
If d, i, o, u, x, or X conversion specifier follows, it specifies that the argument to convert is a size_t or the corresponding signed integer type value.
If n conversion specification follows, it specifies that the argument to convert is a pointer to a size_t or the corresponding signed integer type argument.

t
If d, i, o, u, x, or X conversion specifier follows, it specifies that the argument to convert is ptrdiff_t or the corresponding unsigned integer type.
If n conversion specification follows, it specifies that the argument to convert is a pointer to a ptrdiff_t or the corresponding unsigned integer type argument.

L
If a, A, e, E, f, F, g, or G conversion specifier follows, it indicates that the argument to convert is a long double.

The behavior is undefined when the length modifier is specified for conversion specifications other than above.

Conversion specifiers and their meanings:

d,i
Converts an int type argument to a signed decimal number in "[-]dddd" format.
The precision specifies the minimum number of digits to appear.
If a value is represented with fewer number of digits, leading space up to the precision position is filled with zeros.
Default precision is 1.
If the precision is 0 and the conversion result is 0, nothing is output.

o
Converts an unsigned type argument to an unsigned octal number in "dddd" style.
The precision specifies the minimum number of digits to appear.
If a value is represented with fewer number of digits, leading space up to the precision position is filled with zeros.
Default precision is 1.
If the precision is 0 and the conversion result is 0, no character is output.

u
Converts an unsigned type argument to an unsigned decimal number with "dddd" style.
The precision specifies the minimum number of digits to appear.

If a value is represented with fewer number of digits, leading space up to the precision position is filled with zeros.
Default precision is 1.
If the precision is 0 and the conversion result is 0, no character is output.

x
Converts an unsigned type argument to an unsigned hexadecimal number with "dddd" style (characters "abcdef" are used to represent 10 to 15).
The precision specifies the minimum number of digits to appear.
If a value is represented with fewer number of digits, leading space up to the precision position is filled with zeros.
Default precision is 1.
If the precision is 0 and the conversion result is 0, no character is output.

X
It is equivalent to x conversion specifier except that it uses the uppercases "ABCDEF" instead of the characters "abcdef".

f,F
Converts a double type argument to a decimal number notation in "[-]ddd.ddd" style.
The number of digits after the decimal point is equal to the precision specification.
If the precision is omitted, 6 is assumed.
If the precision is 0 and the '#' flag is not specified, the decimal point does not appear.
When the decimal point appears, a number with at least one digit appears before the decimal point.
Though lower digits are rounded using round half up in the T2EX reference implementation, other implementation-defined rounding is allowed.

In the T2EX reference implementation, the infinity double type argument is converted to "[-]inf", and depending on the implementation, it can be converted to "[-]infinity".
The double type argument representing NaN is converted to "[-]nan(n-char-sequence)" or "[-]nan".
For (n-char-sequence), see nan().
F conversion specifier generates "INF", "INFINITY", and "NAN" instead of "inf", "infinity", and "nan", respectively.

e,E
Converts a double type argument in "[-]d.ddde+/-dd" style.
If the argument is not 0, a single digit number exists before the decimal point and the number of digits after the decimal point is equal to the number of digits specified by the precision.
If the precision is omitted, 6 is assumed.
If the precision is 0 (zero) and the '#' flag is not specified, the decimal point does not appear.
The lower digits are rounded by the implementation-defined method.
E conversion specifier uses 'E' instead of 'e' as the first character in the exponent part.
The exponent part has always two or more digits.
If a value is 0, the exponent becomes "00".

A double type argument representing infinity and NaN is converted as in the case of f and F conversion specifier.

g,G
Depending on the converted value and precision, the double type argument is converted to f or e format (F or E for G conversion specifier).

The number of digits after the decimal point is equal to the number of digits specified by the precision.
If 0 is specified as the precision, it assumed to be 1.
The format results in e or f format depending on the converted value.
If the exponent part of the conversion result is -4 or less, or equal to or more than the precision, e format is used.
The trailing zeros in the decimal part of the conversion result are removed.
The decimal point character is output only when at least one digit number exists after the decimal point or the # flag is specified.

A double type argument representing infinity and NaN is converted as in the case of f and F conversion specifier.

a,A
Converts a double type argument in "[-]0xh.hhhhp+/-d" style.
A single digit hexadecimal number exists before the decimal point and the number of digits of hexadecimal number after the decimal point is equal to the number of digits specified by the precision.

The number before the decimal point is non-zero for the normalized floating point number, and undetermined for the non-normalized.

If the precision is not specified and FLT_RADIX is the power of 2, the precision becomes a value required to precisely represent the value.
If the precision is not specified and FLT_RADIX is other than the power of 2, the precision becomes a value that can precisely represent the double type value.
When the trailing zeros exist, the consecutive zeros are omitted.

If the precision is 0 and the '#' flag is not specified, the decimal point does not appear.
The characters "abcdef" are used for the "a" conversion specifier while the uppercases "ABCDEF" are used for the "A" conversion specifier.
"A" conversion generates a number in the format of 'X' conversions instead of 'x' and 'p'.
The exponent value is always one or more digits and has the minimum number of digits required to represent the binary exponent as a decimal exponent.

A double type argument representing infinity and NaN is converted as in the case of f and F conversion specifier.

c
It converts an int type argument to an unsigned char and outputs the resulting character (one byte).

s
The argument is assumed to point the char type array.
It outputs characters from the beginning of array to a trailing null character (excluding the null character).
If a precision is specified, no character exceeding the precision is output.
If the precision is not specified or it is larger than the size of array, application must guarantee that the array includes a null character.

p
The argument must be a pointer to a void type.
It converts a pointer value to a hexadecimal number as if the pointer value is converted by %#x or %#lx.

n
The argument must be a pointer to an integer.
The number of bytes to have been output so far is stored in the integer by calling this fprintf().
The argument is not converted.

%
Outputs a character '%'.
Complete conversion specifier is %%.

The behavior is undefined when the conversion specifier does not match any of the above formats.
The behavior is undefined when the argument is not a correct type corresponding to the conversion specification.

Even if the minimum field width is not specified or is small, the field is not truncated.
If the conversion result is larger than the minimum field width, the field is extended until the conversion result can be included.
Characters generated by fprintf() and printf() are output as if fputc() and putc() was called respectively.
Therefore, an error that can occur in fputc() also may occur in fprintf(), and an error that can occur in putc() also may occur in printf().

If FLT_RADIX is not a power of 2 for the "a" and "A" conversion specifications and thus disabling the result precisely be represented in the given precision, the result must be one of the two adjacent numbers in the format of hexadecimal floating point with the given precision.
In that case, the error must have the appropriate sign correctly reflecting the current rounding direction.

If the significant digits is DECIMAL_DIG or less, the result must be rounded correctly for e, E, f, F, g, and G conversion specifier.

Return Parameter

If successful, fprintf() or printf() returns the number of bytes to have been output.

If successful, sprintf() returns the number of bytes of the output which was written to "s" excluding the trailing null character.

If successful and "size" is large enough, snprintf() returns the number of bytes of the output that is supposed to be written to "str" excluding the trailing null character.

If an output error occurs, printf(), fprintf(), snprintf(), or sprintf() returns a negative value.

While snprintf() writes nothing when "size" is 0, it returns the number of bytes of the output that is supposed to be written to "str" excluding the trailing null character when "size" is large enough. In this case, "str" is allowed to be NULL.

Error Code

If an error occurs, calling ferror() may return the followings:

Error numbers occurred in fputc()
EOVERFLOW Overflow
- In snprintf(), the value of "size" is larger than INT_MAX
- The number of bytes required to hold the output excluding the trailing null character exceeds the INT_MAX

See Also

fputc, fscanf

Additional Notes

Since there is no multibyte character library, there is no conversion specification for wchar_t and multibyte.

8.20.19 vfprintf, vprintf, vsnprintf, vsprintf - Formatted output by the list of variable number arguments

C Language Interface

```
#include <stdio.h>

int     vfprintf(FILE *stream, const char *format, va_list ap);
int     vprintf(const char *format, va_list ap);
int     vsnprintf(char *str, size_t size, const char *format, va_list ap);
int     vsprintf(char *str, const char *format, va_list ap);
```

Description

vfprintf(), vprintf(), vsnprintf(), and vsprintf() are equivalent to fprintf(), printf(), snprintf(), and sprintf(), respectively, except that they receive the va_list type argument list as the argument instead of the variable number of arguments.

vfprintf(), vprintf(), vsnprintf(), and vsprintf() do not call the va_end macro.
Since vfprintf(), vprintf(), vsnprintf(), and vsprintf() call the va_arg macro, the value of "ap" at the time of completion is undetermined.

Return Parameter

See fprintf().

Error Code

See fprintf().

See Also

fprintf

8.20.20 fscanf, scanf, sscanf - Formatted input conversion

C Language Interface

```
#include <stdio.h>

int     fscanf(FILE* stream, const char* format, ...);
int     scanf(const char* format, ...);
int     sscanf(const char* str, const char* format, ...);
```

Description

fscanf() reads from the input stream pointed to by "stream".
scanf() reads from the standard input.
sscanf() reads from the string pointed to by "str".

These functions read one byte at a time from the specified stream, convert them according to the format specified by "format", and then store the result in the area pointed to by the argument.

Each function has the variable number of arguments of control strings specified by "format" and the pointers indicating the storage area of the converted input.
If the actual number of arguments when the function is called is fewer than the number of arguments required by the format specified by "format", the result is undefined.
If "format" is used up while any arguments still remain, the extra arguments are only evaluated when these API calls are executed, and ignored in fscanf(), scanf(), or sscanf().

fscanf() inputs from the stream as if fgetc() was called.
Therefore, an error that can occur in fgetc() also may occur in fscanf().

"format" is a string consisting of 0 or more directives.
Each directive corresponds to one of (a), (b), and (c):

(a) One or more white-space characters (<space>, <tab>, <new line>, <vertical tab>, or <page break>)
(b) Ordinary characters (neither '%' nor white-space)
(c) Conversion specification

fscanf() executes the directives in "format" sequentially.
If a directive fails, the function terminates there.
Failures of directives are classified into the input error (no available input) or the matching error (improper input).

The directive consisting of one or more white-space characters described in (a) is skipped until the input is exhausted or non-white-space character is reached.
At that time, the reached non-white-space character is deemed to be not read yet.

The directive consisting of the ordinary characters described in (b) is executed as follows:
The next character is read from the input and compared with the character consisting the directive.
If matched, the character is skipped.
If unmatched, the directive results in the failure of matching and the subsequent characters including the current character are left unread.
If the end of file is reached or read error occurs, the directive results in the failure of input without reading characters.

The conversion specification described in (c) starts with '%' character and the characters explained below in the numerical order given.

1. Assignment suppression character '*' (optional)
2. Non-zero decimal integer specifying the maximum field width (optional)
3. Length modifier specifying the size of object to accept (optional)
4. Conversion specification character specifying the type of conversion to apply

Format of conversion specification

```
%[assignment suppression character][maximum field width][length modifier]conversion specifier
```

1. Assignment suppression character (optional)
   Assignment suppression character is an <asterisk> ('*').

2. Maximum field width (optional)
   The maximum field width is a non-zero decimal integer string to specify the maximum field width.

3. Length modifier (optional)
   Length modifier is a character to specify the size of type indicated by the conversion specifier.

4. Conversion specifier

Conversion specifier is a character to specify the type of conversion to apply.

The directive consisting of conversion specifications defines a set of strings to be compared to the input for each conversion specification character.

The conversion specification is executed in the following steps:

If the conversion specification does not contain any one of '[', 'c', 'C', and 'n', a white-space within the input is skipped.

If the conversion specification does not contain "n", one item is read from the input (input item) at first.
The input item is defined as the longest input string that matches the string set specified by the conversion specification as a result of comparison starting from the beginning of the input to the maximum field width.

A byte immediately after the input item, if any, is left unread.
If the length of input item is 0, the execution of the conversion specification fails.
This means a matching error unless any input error including a reaching end of file and a read error has occurred.

Except for the %n conversion specification, the input item (the number of input bytes for %n conversion specification) is converted to the type specified by the conversion specification.
If the input item is not contained in the set of matching strings indicated by the conversion specification, execution of the conversion specification fails.
This means a matching error.

If the assignment suppression by '*' is not specified and the conversion specification starts with %, the conversion corresponding to the conversion specification is performed and the conversion result is stored in the object indicated by the first argument that does not receive the conversion result yet after "format" argument.
The behavior is undefined when this object is improper type or the conversion result cannot be represented by the area provided by the argument.
If the assignment suppression by '*' is specified, the conversion corresponding to the conversion specification is performed, but the pointer argument is not used and the conversion result is discarded.

Length modifiers and their meanings:

hh
	If d, i, o, u, x, X, or n conversion specifier follows, it indicates that the argument is a pointer type to signed char or unsigned char.

h
	If d, i, o, u, x, X, or n conversion specifier follows, it indicates that the argument is a pointer type to short or unsigned short.

l
	If d, i, o, u, x, X, or n conversion specifier follows, it indicates that the argument is a pointer type to long or unsigned long.
	If a, A, e, E, f, F, g, or G conversion specifier follows, it indicates that the argument is a pointer type to double.

ll
	If d, i, o, u, x, X, or n conversion specifier follows, it indicates that the argument is a pointer type to long long or unsigned long long.

j
	If d, i, o, u, x, X, or n conversion specifier follows, it indicates that the argument is a pointer type to intmax_t or uintmax_t.

z
	If d, i, o, u, x, X, or n conversion specifier follows, it indicates that the argument is a pointer type to size_t.

t
	If d, i, o, u, x, X, or n conversion specifier follows, it indicates that the argument is a pointer type to ptrdiff_t or the corresponding unsigned type.

L
	If a, A, e, E, f, F, g, or G conversion specifier follows, it indicates that the argument is a

pointer type to long double.

The behavior is undefined when the length modifier appears in the conversion specifier other than the above.

Conversion specifiers and their meanings:

d
	Converts to a signed decimal integer.
	It matches the signed decimal integer with the same format as the conversion target of strtol() whose "base" is 10.
	If the length modifier is not specified, the corresponding argument must be a pointer to an int type.

i
	Converts to a signed octal, decimal or hexadecimal integer.
	It matches the signed integer with the same format as the conversion target of strtol() whose "base" is 0.
	If the length modifier is not specified, the corresponding argument must be a pointer to an int type.

o
	Converts to a signed octal integer.
	It matches the signed octal integer with the same format as the conversion target of strtol() whose "base" is 8.
	If the length modifier is not specified, the corresponding argument must be a pointer to an unsigned integer type.

u
	Converts to an unsigned decimal integer.
	It matches the unsigned decimal integer with the same format as the conversion target of strtol() whose "base" is 10.
	If the length modifier is not specified, the corresponding argument must be a pointer to an unsigned integer type.

x
	Converts to an unsigned hexadecimal integer.
	It matches the unsigned hexadecimal integer with the same format as the conversion target of strtol() whose "base" is 16.
	If the length modifier is not specified, the corresponding argument must be a pointer to an unsigned integer type.

a,e,f,g
	Converts to a signed decimal floating point number.
	It matches the floating point constant, infinity, or NaN with the same format as the conversion target of strtod().
	If the length modifier is not specified, the corresponding argument must be a pointer to a float type.

s
	Converts to a string excluding the white-space character.
	It matches the string other than the white-space character.
	Application must guarantee that the corresponding argument is a pointer to the beginning of the array of char, signed char, or unsigned char type, which is large enough to store the string and trailing null character.

[scanned character sequence],[^scanned character sequence]
	Converts to a set of expected characters (scanned character set).
	It matches the non-empty string consisting of a set of the expected characters (scanned character set).
	In this case, the skip of white-space characters, which usually takes place, is not performed.
	Application must guarantee that the corresponding argument is a pointer to the beginning of the array of char, signed char, or unsigned char type, which is large enough to store the string and an automatically appended trailing null character.

	Conversion specifier contains all the string so far, including the corresponding ']', in the "format" string.
	If the character immediately after '[' is not '^', the string between '[' and ']' (scanned character sequence) configures the scanned character set.
	If the character immediately after '[' is '^', the scanned character set consists of characters that do not appear in the scanned character sequence between '^' and ']'.

If the conversion specifier starts with "[]" or "[^]", ']' is included in the scanned character sequence and the conversion specifier is up to the next ']'.
Otherwise, the conversion specifier ends with the first ']'.

For example, "[0-9]" means a set of numbers from '0' to '9'.
"[^]0-9-]" means a set of all characters except for three patterns: ']', '0' to '9', and '-'.

c

Converts to a characters including the white-space character.
It matches with a string of length specified by a number in the maximum field width (1 when the maximum field width is not specified).
The null character is not added.
In this case, the skip of white-space characters, which usually takes place, is suppressed.
Application must guarantee that the corresponding argument is a pointer to the array of char, signed char, or unsigned char type, which is large enough to accept the string.

p

Converts to a pointer.
It matches with the string representation of the pointer value generated by the %p conversion specification of the corresponding fprintf(), printf(), snprintf(), or sprintf().
Application must guarantee that the corresponding argument is a pointer to a pointer to void type.
Interpretation of the input item is implementation-dependent.

If the input item has a value previously converted during the execution of the same program, the resulting pointer must match that value.
Otherwise, the behavior of %p conversion is undefined.
In the T2EX reference implementation, hexadecimal representation of the pointer (%#x) is interpreted.
NULL is represented as 0x0.

n

Converts to the input number of bytes.
The input is not consumed.
The number of characters (in bytes) read from the input before %n arrives is stored in the corresponding argument.
Application must guarantee that the corresponding argument is a pointer to integer.
Even when executing %n conversion specification, the number of input items, which is the return value of this function, does not increase.
The behavior is undefined when the conversion specification contains an assignment suppression character or has the maximum field width.

%

Converts to '%'.
It matches one '%' character.
Neither conversion nor assignment is performed.
Complete conversion specifier is %%.

The behavior is undefined when the conversion specifier is invalid.

The conversion specifiers A, E, F, G, and X are equivalent to a, e, f, g, and x, respectively.

If the end of file is reached during input, the conversion terminates.
If the end of file is reached before the character (excluding the leading white-space characters) matching the current conversion specification other than %n is read, execution of the current conversion specification terminates as an input error.
If the execution of the current conversion specification does not terminate due to a matching error, execution of the subsequent conversion specification terminates as an input error.

When the end of string is reached by sscanf(), it is assumed to be equivalent to when the end of file is reached by fscanf().

If the conversion terminates due to an improper input character, the improper input is left unread in the input stream.
The subsequent white-space characters (including <new line> character) unmatched in the conversion specification is left unread in the input stream.

Return Parameter

When successful, fscanf(), scanf(), or sscanf() returns the number of input items that are assigned by successful matching.
If the matching fails first, this value is 0.

If the input terminates before the matching or conversion fails, it returns EOF.
If an error occurs in the input stream, it records the error number in the stream and returns EOF.

Error Code

If an error occurs, calling ferror() may return the followings:

Error numbers occurred in fgetc()
ENOMEM  Insufficient memory
EINVAL  Arguments are insufficient

See Also

fgetc, fprintf

Additional Notes

Since there is no multibyte character library, length modifiers cannot be used for c or s conversion specifiers.

### 8.20.21 vfscanf, vscanf, vsscanf - Formatted input by the list of variable number arguments

C Language Interface

```
#include <stdio.h>

int     vfscanf(FILE* stream, const char* format, va_list ap);
int     vscanf(const char* format, va_list ap);
int     vsscanf(const char* str, const char* format, va_list ap);
```

Description

vfscanf(), vscanf(), and vsscanf() are equivalent to fscanf(), scanf(), and sscanf() respectively, except that they are called with the argument list defined in stdarg.h instead of variable number of arguments.
vfscanf(), vscanf(), and vsscanf() do not call the va_end macro.
Since vfscanf(), vscanf(), and vsscanf() call the va_arg macro, the value of "ap" after the completion of these functions is undeterminated.

Return Parameter

See fscanf().

Error Code

See fscanf().

See Also

fscanf

### 8.20.22 setbuf, setvbuf - Sets buffer of stream

C Language Interface

```
#include <stdio.h>

void    setbuf(FILE* stream, char* buf);
int     setvbuf(FILE* stream, char* buf, int mode, size_t size);
```

Description

setvbuf() sets the buffering of the stream specified by "stream" to "buf".
setvbuf() must be executed before performing other operations after opening the stream.
Use "mode" to specify the type of buffering.

| | |
|---|---|
| _IOFBF | In both input and output, complete buffering is performed. |
| _IOLBF | In both input and output, line buffering is performed. |
| _IONBF | In both input and output, buffering is not performed. |

If "buf" is not NULL, area of "size" bytes starting at "buf" is used for the buffer of the stream.
If "buf" is NULL, setvbuf() allocates the buffer of "size" bytes internally.

If "buf" is not NULL, setbuf() is equivalent to setvbuf(stream, buf, _IOFBF, BUFSIZ).
If "buf" is NULL, it is equivalent to setvbuf(stream, buf, _IONBF, BUFSIZ).

Return Parameter

If successful, setvbuf() returns 0.
If the value of "mode" is invalid or execution fails, it returns a non-zero value.
The setbuf() does not return a value.

Error Code

If an error occurs, calling ferror() may return the followings:

| | |
|---|---|
| EBADF | File descriptor for the stream is invalid |

See Also

fopen

8.20.23 fopen, fopen_eno - Opens stream (of a file)

C Language Interface

```
#include <stdio.h>

FILE*   fopen(const char* path, const char* mode);
FILE*   fopen_eno(const char* path, const char* mode, errno_t* eno); // Additional function
```

Description

fopen() or fopen_eno() opens the file and associates a stream with it.
Opens the file whose name pointed to by "path" with the open mode pointed to by "mode", attaches it to the stream, and returns the stream.

"mode" is a string to indicate the open mode and has one of the following values:

| | |
|---|---|
| "r" | Open with text file read mode. |
| "rb" | Open with binary file read mode. |
| "w" | Opens with the text file write mode.<br>A new file will be created when the file does not exist. When it exists, the file size will be set to 0. |
| "wb" | Opens with the binary file write mode.<br>A new file will be created when the file does not exist. When it exists, the file size will be set to 0. |
| "a" | Opens with the text file write by append mode.<br>A new file will be created when the file does not exist. When it exists, the file offset will be set to the end of the file. |
| "ab" | Opens with the binary file write by append mode.<br>A new file will be created when the file does not exist. When it exists, the file offset will be set to the end of the file. |
| "r+" | Opens with the text file update (read/write) mode. |

"rb+", "r+b"
Opens with the binary file update (read/write) mode.
"w+"
Opens with the text file update (read/write) mode.
A new file will be created when the file does not exist. When it exists, the file size will be set to 0.
"wb+", "w+b"
Opens with the binary file update (read/write) mode.
A new file will be created when the file does not exist. When it exists, the file size will be set to 0.
"a+"
Open with the text file update by append (read/write) mode.
A new file will be created when the file does not exist. When it exists, the file offset will be set to the end of the file.
"ab+", "a+b"
Open with the binary file update by append (read/write) mode.
A new file will be created when the file does not exist. When it exists, the file offset will be set to the end of the file.

Character 'b' means the binary mode.
"mode" without 'b' specification means the text mode.
Although this mode is prepared for a system which treats the read/write in a special way during the text mode, difference of the effect due to "b" does not occur since no special treatment is made to the text mode in this system.

The behavior is undefined when specifications other than the above are made to "mode".

Write operation for the file opened with append mode (mode starting with 'a') always appends the data to the end of file even when fseek() is called.

When opening file with update mode (mode including '+'), input and output can be performed for the stream.

- When performing input after performing output, fflush() or fseek(), fseek64(), fsetpos(), rewind() must be called between the two processes.
- When performing output after performing input, fseek(), fseek64(), fsetpos(), rewid() must be called between the two processes.

When the file is opened, the stream is in a mode to perform the complete buffering unless the file is an interactive device such as a console.
Error information of the stream and the end-of-file indicator are cleared.

If an error occurs and "eno" is not NULL, fopen_eno() stores the error number in the area pointed to by "eno".

fopen() is equivalent to fopen_eno(path, mode, NULL).

Return Parameter

If successful, fopen() or fopen_eno() returns the pointer corresponding to the stream.
If an error occurs, it returns NULL.

Error Code

If an error occurs, the following error numbers are set in "eno":

| | |
|---|---|
| EACCES | Attempted to open the read-only file with the write or update mode |
| EINTR | Aborted by fs_break() |
| EISDIR | "path" is a directory and "mode" is write or update mode |
| EMFILE | Number of opened streams exceeded FOPEN_MAX |
| ENAMETOOLONG | File name is too long<br>- The directory or file name part in "path" is too long (NAME_MAX at maximum)<br>- Whole path length is too long (PATH_MAX at maximum) |
| ENFILE | Number of open files exceeded the system limit |
| ENOENT | File does not exist<br>- File specified by "path" does not exist<br>- "path" is an empty string |
| ENOMEM | Insufficient memory |
| EROFS | Attempted to open a file on the read-only file system for write |

See Also

fclose, fdopen, freopen

### 8.20.24 fdopen, fdopen_eno - Associate a stream with a file descriptor

C Language Interface

```
#include <stdio.h>

FILE*   fdopen(int fd, const char* mode);
FILE*   fdopen_eno(int fd, const char* mode, errno_t* eno); // Additional function
```

Description

fdopen() or fdopen_eno() associates a stream with an existing file descriptor.

For the specification of "mode", see fopen(), fopen_eno().
However, if 'w' is used, do not set the file size to 0.

The specification of "mode" must be compatible with the read/write mode of the file descriptor "fd" opened by fs_open() or fs_creat().
The file position of the newly created stream is set to the file offset value of the file descriptor "fd".
Error information of the stream and the end-of-file indicator are cleared.

If an error occurs and "eno" is not NULL, it stores the error number in the area pointed to by "eno".

fdopen() is equivalent to fdopen_eno(fd, mode, NULL).

Return Parameter

If successful, fdopen() or fdopen_eno() returns the pointer corresponding to the stream.
If an error occurs, it returns NULL.

Error Code

If an error occurs, the following error numbers are set in "eno":

| | |
|---|---|
| EMFILE | Number of opened streams exceeded FOPEN_MAX |
| EBADF | File descriptor is invalid |
| EINVAL | "mode" is invalid |
| ENOMEM | Insufficient memory |

See Also

fopen, fclose, freopen

### 8.20.25 freopen, freopen_eno - Reopens stream

C Language Interface

```
#include <stdio.h>

FILE*   freopen(const char* path, const char* mode, FILE* stream);
FILE*   freopen_eno(const char* path, const char* mode, FILE* stream, errno_t* eno); // Additional function
```

Description

freopen_eno() flushes stream specified by "stream" as if fflush(stream) was called first.
Failure in flushing stream is ignored.
If "path" is not NULL, freopen_eno() closes the file descriptor attached to the stream.
Failure in closing file descriptor is ignored.

Error information of the stream and the end-of-file indicator are cleared.

freopen_eno() next opens the file whose name specified by "path" and attaches it to the stream specified by "stream".
In this case, "mode" is used exactly in the same manner as fopen().

The first specified stream is closed whether or not the last open is successful.

If "path" is NULL, freopen_eno() attempts to change the mode of the stream to the mode specified by "mode" as if the file name attached to the current stream was used.
In this case, if calling freopen_eno() is successful, the file descriptor attached to the stream does not need to be closed.
Available mode change in each situation is implementation-dependent.
In the T2EX reference implementation, change of mode is not allowed.

If an error occurs and "eno" is not NULL, freopen_eno() stores the error number in the area pointed to by "eno".

freopen(path, mode, stream) is equivalent to freopen_eno(path, mode, stream, NULL).

Return Parameter

If successful, freopen(), freopen_eno() returns the stream value.
If an error occurs, it returns NULL.

Error Code

If an error occurs, the following error numbers are set in "eno":

| | |
|---|---|
| EACCES | Access permission specified by "mode" does not exist for file |
| EBADF | File descriptor corresponding to the stream is invalid |
| EINTR | Aborted by fs_break() |
| EISDIR | Though "path" is a directory, write request is specified as "mode" |
| EMFILE | Number of opened streams exceeded FOPEN_MAX |
| ENAMETOOLONG | File name is too long<br>- Length of "path" exceeds the PATH_MAX<br>- The directory or file name part in "path" is too long (NAME_MAX at maximum) |
| ENFILE | Number of open files exceeded the system limit |
| ENOENT | File does not exist<br>- File specified by "path" does not exist<br>- "path" is an empty string |
| ENOTDIR | "path" contains something other than a directory in the prefix part |
| EROFS | Attempted to open a file on the read-only file system for write |

See Also

fopen, fdopen, fclose

8.20.26 fclose_eno, fclose - Closes stream

C Language Interface

```
#include <stdio.h>

int     fclose(FILE* fp);
int     fclose_eno(FILE* fp, errno_t* eno); // Additional function
```

Description

fclose_eno() flushes the stream specified by "stream" and closes the file attached to the stream.
Unwritten data in the buffer of the stream is written to the file while the unread data within the buffer is discarded.
Whether or not the calling is successful, association between the stream and file is released and the buffer set by setbuf() or setvbuf() is detached from the stream.
If the buffer has been automatically allocated, it is released.

If an error occurs and "eno" is not NULL, fclose_eno() stores the error number in the area pointed to

by "eno".

fclose(fp) is equivalent to fclose_eno(fp, NULL).

Return Parameter

When successful, fclose_eno() or fclose() returns 0. When an error occurs, it returns EOF.

Error Code

If an error occurs, the following error numbers are set in "eno":

| | |
|---|---|
| EAGAIN | Since O_NONBLOCK flag of the file descriptor for the stream is set and writing will have caused a wait, the function returned immediately |
| EBADF | File descriptor corresponding to the stream is invalid |
| EFBIG | Position exceeds the limit of file size |
| EINTR | Aborted by fs_break() |
| EIO | I/O error |
| ENOSPC | Insufficient device space |

See Also

fopen

## 8.21 stdlib.h

The header stdlib.h defines the following macros and types.

Macros

NULL
Null pointer constant. The macro shall expand to an integer constant expression with the value 0 cast to type (void *).

RAND_MAX
Maximum value returned by rand(); at least 32767.
In the T2EX reference implementation, this shall be 0x7fffffff.

Types

div_t
Structure type returned by the div() function.
The structure consists of the quotient and remainder of int type.

ldiv_t
Structure type returned by the ldiv() function.
The structure consists of the quotient and remainder of long type.

lldiv_t
Structure type returned by the lldiv() function.
The structure consists of the quotient and remainder of long long type.

size_t
Unsigned integer type of the result of the sizeof operator.

wchar_t
Integer type whose range of values can represent distinct codes for all members of the largest extended character set specified among the system locales.
For instance, multi-byte characters are used in UTF-8 which represents the Unicode as a byte sequence. This type means the integer type capable of representing this maximum number of bytes.
The null character shall have the code value zero.
This is the type when each element in a character set (which may be multiple bytes in UTF-8) is used as a one-character integer character constant and is called "wide character type".

Functions

### 8.21.1 abort - Abnormal system termination

C Language Interface

```
#include <stdlib.h>

void    abort(void);
```

Description

The abort() terminates the system abnormally.
The standard behavior is that an error message is output and tm_monitor() is executed. However, by executing setabort(), the processing in case of system anomaly can be changed.

Return Parameter

None.

Error Code

None.

See Also

setabort

### 8.21.2 abs, labs, llabs - return a integer absolute value

C Language Interface

```
#include <stdlib.h>

int             abs(int i);
long            labs(long i);
long long       llabs(long long i);
```

Description

The abs(), labs(), and llabs() functions shall compute the absolute value of its integer parameter, i. If the result cannot be represented, the behavior is undefined.

Return Parameter

The abs(), labs(), and llabs() function shall return the absolute value of its integer parameter.

Error Code

None.

See Also

fabs

### 8.21.3 atof - convert a string to a double-precision number

C Language Interface

```
#include <stdlib.h>

double  atof(const char *str);
```

Description

The atof() function shall convert a string str to a double-precision number
The call atof(str) shall be equivalent to:

```
        strtod(str,(char **)NULL),
```

except that the handling of errors may differ. If the value cannot be represented, the behavior is undefined.

Return Parameter

The atof() function shall return the converted value if the value can be represented.

Error Code

None.

See Also

strtod

8.21.4 atoi, atol, atoll - convert a string to an integer

C Language Interface

```
#include <stdlib.h>

int             atoi(const char *str);
long            atol(const char *str);
long long       atoll(const char *str);
```

Description

The atoi(), atol(), and atoll() functions shall convert a string str to an integer.
The call atoi(str) and atol(str) shall be equivalent to:

```
        (int) strtol(str, (char **)NULL, 10)
```

except that the handling of errors may differ. If the value cannot be represented, the behavior is undefined.

The call atoll(str) shall be equivalent to:

```
        strtoll(str, (char **)NULL, 10)
```

except that the handling of errors may differ. If the value cannot be represented, the behavior is undefined.

Return Parameter

These functions shall return the converted value if the value can be represented.

Error Code

None.

See Also

strtol

8.21.5 bsearch - Binary-tree search

C Language Interface

```
#include <stdlib.h>

void    *bsearch(const void *key, const void *base, size_t nel, size_t width,
                 int (*compare)(const void *, const void *));
```

Description

The bsearch() function shall search an array of nel objects, the first element of which is pointed to by base, for an element that matches the object pointed to by key. The size of each element in the array is specified by width. If the nel argument has the value zero, the comparison function pointed to by compare shall not be called and no match shall be found.

The comparison function pointed to by compare shall be called with two arguments that point at the key object and to an array element, in that order.

The application shall ensure that the comparison function pointed to by compare does not alter the contents of the array. The bsearch() may reorder elements of the array between calls to the comparison function, but shall not alter the contents of any individual element.

When the same objects (consisting of width bytes, irrespective of their current positions in the array) are passed more than once to the comparison function, the results shall be consistent with one another. That is, the same object shall always compare the same way with the key.

The application shall ensure that the comparison function returns an integer less than, equal to, or greater than 0 if the key object is considered, respectively, to be less than, to match, or to be greater than the array element. The application shall ensure that the array consists of all the elements of the array must be sorted according to the ascending order by the comparison function.

Return Parameter

The bsearch() function shall return a pointer to a matching member of the array, or a null pointer if no member match. If two or more members match, which member is returned is unspecified.

Error Code

None.

See Also

lsearch, qsort

### 8.21.6 calloc - Allocates user-level memory

C Language Interface

```
#include <stdlib.h>

void    *calloc(size_t nelem, size_t elsize);
```

Description

The calloc() function shall allocate unused memory area for an array of nelem elements each of whose size in bytes is elsize. The memory area shall be initialized to all zero bytes.

The protection level of the allocated memory area is user-level.

The pointer returned if the allocation succeeds shall be suitably aligned so that it may be assigned to a pointer to any type of object and then used to access such an object or an array of such objects in the memory area allocated (until the memory area is explicitly freed or reallocated). Each such allocation shall yield a pointer to an object disjoint from any other object.

The pointer returned shall point to the start (lowest byte address) of the allocated memory area. If the memory area cannot be allocated, a null pointer shall be returned. If the size of the memory area requested is 0, the behavior is implementation-defined: the value returned shall be either a null pointer or a unique pointer.
In the T2EX reference implementation, NULL is returned if the area size is 0.

Return Parameter

Upon successful completion with both nelem and elsize non-zero, calloc() shall return a pointer to the allocated memory area. If either nelem or elsize is 0, then either a null pointer or a unique pointer that can be successfully passed to free() shall be returned. Otherwise, it shall return a null pointer.

Error Code

None.

See Also

malloc, realloc, free

### 8.21.7 div, ldiv, lldiv - compute the quotient and remainder of an integer division

C Language Interface

```
#include <stdlib.h>

div_t           div(int n, int d);
ldiv_t          ldiv(long n, long d);
lldiv_t         lldiv(long long n, long long d);
```

Description

The div(), ldiv(), and lldiv() functions shall compute the quotient and remainder of the division of the numerator n by the denominator d. If the division is inexact, the resulting quotient is the integer of lesser magnitude that is the nearest to the algebraic quotient. If the result cannot be represented, the behavior is undefined; otherwise, (quot * d + rem) shall equal numerator n.

Return Parameter

div(), ldiv(), and lldiv() functions shall return a structure of type div_t, ldiv_t, and lldivt respectively, comprising both the quotient and the remainder.
These structures includes the following members, in any order:

| | |
|---|---|
| quot | quotient: int, long, and long long types according to the function |
| rem | remainder: int, long, and long long types according to the function |

Error Code

None.

See Also

ldiv

8.21.8 free - free allocated memory

C Language Interface

```
#include <stdlib.h>

void    free(void *ptr);
```

Description

The free() function shall cause the memory area pointed to by ptr to be deallocated; that is, made available for further allocation. If ptr is a null pointer, no action shall occur. Otherwise, if the argument does not match a pointer earlier returned by a function that allocates memory by calloc(), malloc(), and realloc(), or if the memory area has been deallocated by a call to free() or realloc(), the behavior is undefined.

Any use of a pointer that refers to freed memory area results in undefined behavior.

Return Parameter

The free() function shall not return a value.

Error Code

None.

See Also

calloc, malloc, realloc

8.21.9 malloc - Allocates user-level memory area

C Language Interface

#include <stdlib.h>

void *malloc(size_t size);

Description

The malloc() function shall allocate unused memory area for an object whose size in bytes is specified by size and whose value is unspecified.
The protection level of the allocated memory area is user-level.

The order and contiguity of memory area allocated by successive calls to malloc() is unspecified.
The pointer returned if the allocation succeeds shall be suitably aligned so that it may be assigned to a pointer to any type of object and then used to access such an object in the memory area allocated (until the memory area is explicitly freed or reallocated). Each such allocation shall yield a pointer to an object disjoint from any other object. The pointer returned points to the start (lowest byte address) of the allocated memory area.
If the memory area cannot be allocated, a null pointer shall be returned. If the size of the memory area requested is 0, the behavior is implementation-defined: the value returned shall be either a null pointer or a unique pointer.

Return Parameter

Upon successful completion with size not equal to 0, malloc() shall return a pointer to the allocated memory area. If size is 0, either a null pointer or a unique pointer that can be successfully passed to free() shall be returned. Otherwise, it shall return a null pointer
In the T2EX reference implementation, NULL is returned if size is 0.

Error Code

None.

See Also

calloc, realloc, free

8.21.10 qsort - sort a table of data

C Language Interface

#include <stdlib.h>

void qsort(void *base, size_t nel, size_t width, int (*compar)(const void *, const void *));

Description

The qsort() function shall sort an array of nel objects, the first element of which is pointed to by base. The size of each object, in bytes, is specified by the width argument. If the nel argument has the value zero, the comparison function pointed to by compar shall not be called and no rearrangement shall take place.

The application shall ensure that the comparison function pointed to by compar does not alter the contents of the array. The qsort() may reorder elements of the array between calls to the comparison function, but shall not alter the contents of any individual element.

When the same objects (consisting of width bytes, irrespective of their current positions in the array) are passed more than once to the comparison function, the results shall be consistent with one another. That is, they shall define a total ordering on the array.

The contents of the array shall be sorted in ascending order according to a comparison function. The compar argument is a pointer to the comparison function, which is called with two arguments that point to the elements being compared. The application shall ensure that the function returns an integer less than, equal to, or greater than 0, if the first argument is considered respectively less than, equal to, or greater than the second. If two members compare as equal, their order in the sorted array is unspecified.

Return Parameter

The qsort() function shall not return a value.

Error Code

None.

### 8.21.11 rand_r - pseudo-random number generator

C Language Interface

```
#include <stdlib.h>

int     rand_r(unsigned int *seed);
```

Description

The rand() function shall compute a sequence of pseudo-random integers in the range [0, RAND_MAX].

If rand_r() is called with the same initial value for the object pointed to by seed and that object is not modified between successive returns and calls to rand_r(), the same sequence shall be generated.

Return Parameter

The rand_r() function shall return a pseudo-random integer.

Error Code

None.

See Also

drand48_r

### 8.21.12 drand48_r, lrand48_r, mrand48_r, srand48_r, seed48_r, lcong48_r - Generates uniformly-distributed pseudo random numbers

C Language Interface

```
#include <stdlib.h>

double  drand48_r(struct rand48_data *buffer);
long    lrand48_r(struct rand48_data *buffer);
long    mrand48_r(struct rand48_data *buffer);
void    srand48_r(long int seedval, struct rand48_data *buffer);
void    seed48_r(unsigned short int seed16v[3], struct rand48_data *buffer, unsigned short oldxi[3]);
void    lcong48_r(unsigned short int param[7], struct rand48_data *buffer);
```

Description

The drand48_r(), lrand48_r(), mrand48_r(), srand48_r(), seed48_r(), and lcong48_r() generate pseudo random numbers by linear congruential method and 48-bit integer calculation.

The drand48_r() returns a non-negative, double-precision floating point value uniformly distributed in the [0.0, 1.0) range.
The lrand48_r() returns a non-negative integer uniformly distributed in the [0, 2^31) range ('^' represents the power).
The mrand48_r() returns a signed integer uniformly distributed in the [-2^31, 2^31) range ('^' represents the power).

The srand48_r(), seed48_r() and lcong48_r() are functions to initialize the "rand48_data" structure before one of them calls drand48_r(), lrand48_r(), or mrand48_r().

The drand48_r(), lrand48_r(), mrand48_r(), srand48_r(), seed48_r(), and lcong48_r() operate by generating the 48-bit integer column Xi as the following linear congruent expression.
For the nth X, the (n+1)th X is obtained by the following expression.

```
Xn+1 = (a * Xn + c) mod m     n>=0
```

The parameter m is 2^48, and thus 48-bit integer calculation is performed ('^' represents the power).
Unless lcong48_r() is called, the following values are used for the multiplier a and the addend c.

```
a = 0x5DEECE66D
c = 0xB
```

The value returned by drand48_r(), lrand48_r(), or mrand48_r() is obtained by first generating the next 48-bit Xi in the integer column.
Next, a number whose number of bits is appropriate for the type of return code is copied from Xi starting from the most significant bit and converted to the return code type.

The structure rand48_data includes the following data.

```
struct rand48_data {
        unsigned short  xi[3];          /* Current Xi value : xi[0] is the least significant 16 bits
*/
        unsigned short  mult[3];        /* Multiplier a : mult[0] is the least significant 16 bits */
        unsigned short  add;            /* Addend c */
        /* Implementation-dependent element */          /* The T2EX reference implementation does not
have this element */
};
```

The calculation of random numbers uses the above rand48_data.
Therefore, the rand48_data structure needs to be initialized first using the srand48_r(), seed48_r(), or lcong48_r().

The drand48_r(), lrand48_r(), and mrand48_r() use the Xi, m , and a given by rand48_data to calculate Xi+1 = (a * Xi + c ) mod.

The generated last 48-bit value is defined to be the next XI and stored in xi[3] of the buffer.

The initialization function srand48_r() initializes the Xi value of rand48_data as follows.
The higher 32 bits of xi are set to the lower 32 bits of seedval.
The lower 16 bits of xi (i.e., xi[0]) are set to 0x330E.
The "mult" and "add" are set to the above default values.

The initialization function seed48_r() sets the Xi value of rand48_data to the value specified by seed16v[3].
The lower 16 bits of Xi (xi[0]) are set to the lower 16 bits of seed16v[0].
The middle 16 bits of Xi (xi[1]) are set to the lower 16 bits of seed16v[1].
The higher 16 bits of Xi (xi[2]) are set to the lower 16 bits of seed16v[2].
In addition, the previous Xi value (before being called) is copied to the area specified by oldxi.
The "mult" and "add" are set to the above default values.

The initialization function lcong48_r() sets the initial number Xi, multiplier a, and addend c of rand48_data to the specified value.
The param[0-2] specifies Xi, the param[3-5] specifies multiplier a, and the param[6] specifies 16-bit addend c.

Return Parameter

The drand48_r(), lrand48_r(), mrand48_r(), srand48_r(), seed48_r(), and lcong48_r() return the values mentioned above.

Error Code

None.

See Also

rand_r

8.21.13 realloc - Reallocates user-level memory

C Language Interface

```
#include <stdlib.h>

void    *realloc(void *ptr , size_t size);
```

Description

The realloc() resizes the memory object of user-level protection pointed to by "ptr" to the size specified by "size".
The memory object content is maintained unless it is resized smaller than whichever smaller of the previous size or "size".
If the memory object needs to be moved, the area previously allocated to the object is released.
If the size is enlarged, the content of the newly allocated portion of the object is undefined.
If the size is 0 and ptr is not NULL, the area pointed to by ptr is released.
If no area is allocated, the object remains as is without being released.

If ptr is NULL, realloc() is equivalent to malloc(size).

If ptr is not the return code of the previously executed calloc(), malloc(), or realloc(), or if the area is released by free() or realloc(), the behavior shall be undefined.

The pointer returned at a successful allocation is properly aligned. This pointer can always access the array of objects in the allocated area (until the area is released or reallocated) regardless of the type of object pointer it is substituted to.
The pointer by this allocation is a pointer to the object distinguished from any other objects.
The returned pointer is the beginning of the allocated area.
NULL is returned if no area is allocated.
The protection level of the allocated memory area is user-level.

Return Parameter

The realloc() returns a pointer to the allocated area if "size" is not 0 and allocation is successful.
If "size" is 0, NULL or a unique pointer that can be passed to free() is returned.
If memory is insufficient, NULL is returned.
In the T2EX reference implementation, NULL is returned if size is 0.

Error Code

None.

See Also

calloc, malloc, free

8.21.14 realpath_eno - Normalizes path name

C Language Interface

```
#include <stdlib.h>

char    *realpath_eno(const char *path, char *resolved_path, errno_t* eno);
```

Description

The realpath_eno() generates an absolute path name that does not contain '.', '..', or extra '/' based on the string of the path name specified by "path".
The generated path name is stored as a null-terminated string in an area pointed to by resolved_path whose maximum size is PATH_MAX bytes.
If resolved_path is NULL, the generated path name is stored as a string ending with a null character in the buffer allocated by malloc().

If an error occurs and "eno" is not NULL, realpath_eno() stores the error number in the area pointed to by "eno".

Return Parameter

If successful, realpath_eno() returns the pointer to the buffer containing the generated path name.
If failed, it returns NULL.

If resolved_path is NULL, the pointer returned by realpath_eno() can be passed to free().
If resolved_path is not NULL and realpath_eno() fails, the content of the buffer pointed to by resolved_path shall be undefined.

Error Code

The following error numbers may be stored in the area pointed to by eno.

| | |
|---|---|
| EINVAL | Either path or resolved_path is NULL |
| EIO | I/O error |
| ENAMETOOLONG | File name is too long<br>- The directory or file name part in "path" is too long (NAME_MAX at maximum).<br>- Whole pathname length is too long (PATH_MAX at maximum). |
| ENOENT | File specified by "path" does not exist |
| ENOTDIR | "path" contains something other than a directory in the prefix part |

See Also

realpath2_eno

8.21.15 realpath2_eno - Normalizes path name

C Language Interface

```
#include <stdlib.h>

char    *realpath2_eno(const char *path1, const char* path2, char *resolved_path, errno_t *eno);
```

Description

The realpath2() regards path1 as the current working directory and generates an absolute path name that does not contain '.', '..', or extra '/' based on the string of the path name specified by path2.
The generated path name is stored as a null-terminated string in an area pointed to by resolved_path whose maximum size is PATH_MAX bytes.
If resolved_path is NULL, the generated path name is stored as a string ending with a null character in the buffer allocated by malloc().
If eno is not NULL in case of an error, the error number is stored in the area pointed to by eno.

Return Parameter

If successful, realpath2_eno() returns the pointer to the buffer containing the generated path name.
If failed, it returns NULL.

If resolved_path is NULL, the pointer returned by realpath2_eno() can be passed to free().
If resolved_path is not NULL and realpath2_eno() fails, the content of the buffer pointed to by resolved_path shall be undefined.

Error Code

The following error numbers may be stored in the area pointed to by eno.

| | |
|---|---|
| EINVAL | Either path or resolved_path is NULL |
| EIO | I/O error |
| ENAMETOOLONG | File name is too long<br>- Directory or file name part in pathname is too long (NAME_MAX at maximum)<br>- Whole pathname length is too long (PATH_MAX at maximum) |
| ENOENT | File specified by "path" does not exist |
| ENOTDIR | "path" contains something other than a directory in the prefix part |

8.21.16 setabort - Registers the function that processes abnormal system termination

C Language Interface

```
#include <stdlib.h>

int     setabort(void (*func)(void));
```

Description

This registers the function func executed at the abnormal system termination generated by abort().
The already registered functions are released making only the last registered function valid.

If func is NULL, it means that the library standard abnormality handling function is to be registered.
The standard abnormality handling function outputs an error message and executes tm_monitor().

Return Parameter

The setabort() is always successful and returns 0.

Error Code

None.

See Also

abort

Additional Notes

This function is a T2EX-specific function.

8.21.17 strtod, strtof, strtold - Converts from string to double, float, and long double type number

C Language Interface

```
#include <stdlib.h>

double          strtod(const char *nptr, char **endptr);
float           strtof(const char *nptr, char **endptr);
long double     strtold(const char *nptr, char **endptr);
```

Description

These functions shall convert the initial portion of the string pointed to by nptr to double, float, and long double representation, respectively.

First, they decompose the input string into three parts:
(1) An initial, possibly empty, sequence of white-space characters (as specified by isspace())
(2) A subject sequence interpreted as a floating-point constant or representing infinity or NaN
(3) A final string of one or more unrecognized characters, including the terminating NUL character of the input string

Then these API calls shall attempt to convert the subject sequence to a floating-point number, and return the result.

The expected form of the subject sequence is an optional '+' or '-' sign, then one of the following:

- A non-empty sequence of decimal digits optionally containing a radix character; then an optional exponent part

- A 0x or 0X, then a non-empty sequence of hexadecimal digits optionally containing a radix character; then an optional binary exponent part

 - One of INF or INFINITY, ignoring case

 - One of NAN or NAN(n-char-sequence), ignoring case in the NAN part,

An exponent part has the form of one of the following:
 - The decimal exponent part has a following form:
        'e' or 'E', followed by an optional '+' or '-' sign, then a non-empty numeric string that represents the exponent value.
        The exponent value means how many power of 10.
 - The binary exponent part has a following form:
        'p' or 'P', followed by an optional '+' or '-' sign, then a non-empty numeric string that represents the exponent value.
        The exponent value means how many power of 2.

The subject sequence is defined as the longest initial subsequence of the input string, starting with the first non-white-space character, that is of the expected form. The subject sequence contains no characters if the input string is not of the expected form.

A character sequence INF or INFINITY shall be interpreted as an infinity, if representable in the return type, else as if it were a floating constant that is too large for the range of the return type.

A character sequence NAN or NAN(n-char-sequence) shall be interpreted as a quiet NaN, if supported in the return type, else as if it were a subject sequence part that does not have the expected form.

A pointer to the final string is stored in the memory area pointed to by endptr, if endptr is not a null pointer.

If the subject sequence has the hexadecimal form and FLT_RADIX is a power of 2, the value resulting from the conversion is correctly rounded.

If the subject sequence is empty or does not have the expected form, no conversion shall be performed; the value of nptr is stored in the memory area pointed to by endptr, if endptr is not a null pointer.

Return Parameter

Upon successful completion, these functions shall return the converted value.
If no conversion could be performed, 0 shall be returned.

If the correct value is outside the range of representable values, +/-HUGE_VAL, +/-HUGE_VALF, or +/-HUGE_VALL shall be returned (according to the sign of the value).

If the correct value would cause an underflow, a value whose magnitude is no greater than the smallest normalized positive number in the return type (usually 0) shall be returned.

Error Code

None.

See Also

strtol, fscanf, isspace

8.21.18 strtol, strtoll - convert a string to a long integer

C Language Interface

```
#include <stdlib.h>

long            strtol(const char *str, char **endptr, int base);
long long       strtoll(const char *str, char **endptr, int base);
```

Description

These functions shall convert the initial portion of the string pointed to by str to a type long and long long representation, respectively.

First, they decompose the input string into three parts:

(1) An initial, possibly empty, sequence of white-space characters (as specified by isspace())
(2) A subject sequence interpreted as an integer represented in some radix determined by the value of base
(3) A final string of one or more unrecognized characters, including the terminating NUL character of the input string.

Then they shall attempt to convert the subject sequence to an integer, and return the result.

If the value of base is 0, the expected form of the subject sequence is that of a decimal constant, octal constant, or hexadecimal constant, any of which may be preceded by a '+' or '-' sign.
 - A decimal constant begins with a non-zero digit, and consists of a sequence of decimal digits.
 - An octal constant consists of the prefix '0' optionally followed by a sequence of the digits '0' to '7' only.
 - A hexadecimal constant consists of the prefix 0x or 0X followed by a sequence of the decimal digits and letters 'a' (or 'A') to 'f' (or 'F') with values 10 to 15 respectively.

If the value of base is between 2 and 36, the expected form of the subject sequence is a sequence of letters and digits representing an integer with the radix specified by base, optionally preceded by a '+' or '-' sign.
The letters from 'a' (or 'A') to 'z' (or 'Z') inclusive are ascribed the values 10 to 35; only letters whose ascribed values are less than that of base are permitted.
If the value of base is 16, the characters 0x or 0X may optionally precede the sequence of letters and digits, following the sign if present.

The subject sequence is defined as the longest initial subsequence of the input string, starting with the first non-white-space character that is of the expected form. The subject sequence shall contain no characters if the input string is empty or consists entirely of white-space characters, or if the first non-white-space character is other than a sign or a permissible letter or digit.

If the subject sequence has the expected form and the value of base is 0, the sequence of characters starting with the first digit shall be interpreted as an integer constant.
If the subject sequence has the expected form and the value of base is between 2 and 36, it shall be used as the base for conversion, ascribing to each letter its value as given above.
If the subject sequence begins with a minus-sign, the value resulting from the conversion shall be negated.

A pointer to the final string shall be stored in the object pointed to by endptr, fi endptr is not a null pointer.

If the subject sequence is empty or does not have the expected form, no conversion is performed; the value of str is stored in the object pointed to by endptr, if endptr is not a null pointer.

Return Parameter

Upon successful completion, these functions shall return the converted value, if any.

If no conversion could be performed, 0 shall be returned.

If the correct value is outside the range of representable values, LONG_MIN, LONG_MAX, LLONG_MIN, or LLONG_MAX shall be returned (according to the sign of the value).

Error Code

None.

See Also

strtod, fscanf, isalpha

8.21.19 strtoul, strtoull - convert a string to an unsigned long integer

C Language Interface

```
#include <stdlib.h>

unsigned long          strtoul(const char *str, char **endptr, int base);
unsigned long long     strtoull(const char *str, char **endptr, int base);
```

Description

These functions shall convert the initial portion of the string pointed to by str to a type unsigned long and unsigned long long representation, respectively.
First, they decompose the input string into three parts:
(1) An initial, possibly empty, sequence of white-space characters (as specified by isspace())
(2) A subject sequence interpreted as an integer represented in some radix determined by the value of base
(3) A final string of one or more unrecognized characters, including the terminating NUL character of the input string

Then they shall attempt to convert the subject sequence to an unsigned integer, and return the result.

If the value of base is 0, the expected form of the subject sequence is that of a decimal constant, octal constant, or hexadecimal constant, any of which may be preceded by a '+' or '-' sign.
 - A decimal constant begins with a non-zero digit, and consists of a sequence of decimal digits.
 - An octal constant consists of the prefix '0' optionally followed by a sequence of the digits '0' to '7' only.
 - A hexadecimal constant consists of the prefix 0x or 0X followed by a sequence of the decimal digits and letters 'a' (or 'A') to 'f' (or 'F') with values 10 to 15 respectively.

If the value of base is between 2 and 36, the expected form of the subject sequence is a sequence of letters and digits representing an integer with the radix specified by base, optionally preceded by a '+' or '-' sign.
The letters from 'a' (or 'A') to 'z' (or 'Z') inclusive are ascribed the values 10 to 35; only letters whose ascribed values are less than that of base are permitted.
If the value of base is 16, the characters 0x or 0X may optionally precede the sequence of letters and digits, following the sign if present.

The subject sequence is defined as the longest initial subsequence of the input string, starting with the first non-white-space character that is of the expected form. The subject sequence shall contain no characters if the input string is empty or consists entirely of white-space characters, or if the first non-white-space character is other than a sign or a permissible letter or digit.

If the subject sequence has the expected form and the value of base is 0, the sequence of characters starting with the first digit shall be interpreted as an integer constant. If the subject sequence has the expected form and the value of base is between 2 and 36, it shall be used as the base for conversion, ascribing to each letter its value as given above. If the subject sequence begins with a minus-sign, the value resulting from the conversion shall be negated.

A pointer to the final string shall be stored in the object pointed to by endptr, if endptr is not a null pointer.

If the subject sequence is empty or does not have the expected form, no conversion shall be performed; the value of str shall be stored in the object pointed to by endptr, if endptr is not a null pointer.

Return Parameter

Upon successful completion, these functions shall return the converted value, if any.

If no conversion could be performed, 0 shall be returned

If the correct value is outside the range of representable values, ULONG_MAX or ULLONG_MAX shall be returned.

Error Code

None.

See Also

strtod, strtol, fscanf, isalpha

## 8.22 string.h

The header string.h defines the following functions that manipulate strings.
T2EX excludes the strerror and strtok functions which are non-thread-safe for the standard C library string operation function group.

Functions

### 8.22.1 memccpy - copy bytes in memory

C Language Interface

```
#include <string.h>

void    *memccpy(void *dst, const void *src, int c, size_t n);
```

Description

The memccpy() function shall copy bytes from memory area src into dst, stopping after the first occurrence of byte c (converted to an unsigned char) is copied, or after n bytes are copied, whichever comes first. If copying takes place between objects that overlap, the behavior is undefined.

Return Parameter

The memccpy() function shall return a pointer to the byte after the copy of c in dst, or a null pointer if c was not found in the first n bytes of src.

Error Code

None.

See Also

bcopy, memmove, memccpy

### 8.22.2 memchr - find byte in memory

C Language Interface

```
#include <string.h>

void    *memchr(const void *s, int c, size_t n);
```

Description

The memchr() function shall locate the first occurrence of c (converted to an unsigned char) in the initial n bytes (each interpreted as unsigned char) of the object pointed to by s.

Return Parameter

The memchr() function shall return a pointer to the located byte, or a null pointer if the byte does not occur in the object.

Error Code

None.

See Also

strchr, index

8.22.3 memcmp - compare bytes in memory

C Language Interface

#include <string.h>

int     memcmp(const void *s1, const void *s2, size_t n);

Description

The memcmp() function shall compare the first n bytes (each interpreted as unsigned char) of the object pointed to by s1 to the first n bytes of the object pointed to by s2.

Return Parameter

The memcmp() function shall return an integer greater than, equal to, or less than 0, if the object pointed to by s1 is greater than, equal to, or less than the object pointed to by s2, respectively.

Error Code

None.

See Also

strcmp

8.22.4 memcpy - copy bytes in memory

C Language Interface

#include <string.h>

void    *memcpy(void *dst, const void *src, size_t n);

Description

The memcpy() function shall copy n bytes from the object pointed to by src into the object pointed to by dst. If copying takes place between objects that overlap, the behavior is undefined.

Return Parameter

The memcpy() function shall return dst.

Error Code

None.

See Also

bcopy, memmove

8.22.5 memmove - copy bytes in memory with overlapping areas

C Language Interface

#include <string.h>

void    *memmove(void *dst, const void *src, size_t n);

Description

The memmove() function shall copy n bytes from the object pointed to by src into the object pointed to by dst. Copying takes place as if the n bytes from the object pointed to by src are first copied into a temporary array of n bytes that does not overlap the objects pointed to by dst and src, and then the n bytes from the temporary array are copied into the object pointed to by dst.

Return Parameter

The memmove() function shall return dst.

Error Code

None.

See Also

bcopy, memcpy

8.22.6 memset - set bytes in memory

C Language Interface

#include <string.h>

void *memset(void *s, int c, size_t n);

Description

The memset() function shall copy c (converted to an unsigned char) into each of the first n bytes of the object pointed to by s.

Return Parameter

The memset() function shall return s.

Error Code

None.

See Also

bzero

8.22.7 strcat - concatenate two strings

C Language Interface

#include <string.h>

char *strcat(char *dst, const char *src);

Description

The strcat() function shall append a copy of the string pointed to by src (including the terminating NUL character) to the end of the string pointed to by dst. The initial byte of src overwrites the NUL character at the end of dst. If copying takes place between objects that overlap, the behavior is undefined.

Return Parameter

The strcat() function shall return dst.

Error Code

None.

See Also

strncat, strcpy, strncpy

8.22.8 strchr - string scanning operation

C Language Interface

#include <string.h>

char *strchr(const char *s, int c);

Description

The strchr() function shall locate the first occurrence of c (converted to a char) in the string pointed to by s. The terminating NUL character is considered to be part of the string.

Return Parameter

Upon completion, strchr() shall return a pointer to the searched character c, or a null pointer if the character c was not found.

Error Code

None.

See Also

strrchr, memchr, index

8.22.9 strcmp - compare two strings

C Language Interface

#include <string.h>

int strcmp(const char *s1, const char *s2);

Description

The strcmp() function shall compare the string pointed to by s1 to the string pointed to by s2.

Return Parameter

Upon completion, strcmp() shall return an integer greater than, equal to, or less than 0, if the string pointed to by s1 is greater than, equal to, or less than the string pointed to by s2, respectively.

Error Code

None.

See Also

bcmp, memcmp, strncmp

8.22.10 strcoll - string comparison using collating information

C Language Interface

#include <string.h>

int     strcoll(const char *s1, const char *s2);

Description

The strcoll() functions shall compare the string pointed to by s1 to the string pointed to by s2, both interpreted as appropriate to the collation order of the system locale.

Return Parameter

Upon successful completion, strcoll() shall return an integer greater than, equal to, or less than 0, according to whether the string pointed to by s1 is greater than, equal to, or less than the string pointed to by s2 when both are interpreted as appropriate to the system locale.

Error Code

None.

See Also

bcmp, memcmp, strcmp

8.22.11 strcpy - copy a string and return a pointer to the end of the result

C Language Interface

#include <string.h>

char    *strcpy(char *dst, const char *src);

Description

The strcpy() function shall copy the string pointed to by src (including the terminating NUL character) into the array pointed to by dst.
If copying takes place between objects that overlap, the behavior is undefined.

Return Parameter

The strcpy() function shall return dst.

Error Code

None.

See Also

bcopy, memcpy, memmove, strncpy

8.22.12 strcspn - get the length of a complementary substring

C Language Interface

#include <string.h>

size_t  strcspn(const char *s1, const char *s2);

Description

The strcspn() function shall compute the length (in bytes) of the maximum initial segment of the string pointed to by s1 which consists entirely of bytes not from the string pointed to by s2.

Return Parameter

The strcspn() function shall return the length of the computed segment of the string pointed to by s1.

Error Code

None.

See Also

strspn, memchr, index, strchr, strstr

8.22.13 strdup - duplicate a specific number of bytes from a string

C Language Interface

```
#include <string.h>

char    *strdup(const char *s);
```

Description

The strdup() function shall return a pointer to a new string, which is a duplicate of the string pointed to by s. The returned pointer can be passed to free(). A null pointer is returned if the new string cannot be created.

Return Parameter

The strdup() function shall return a pointer to a new string on success. Otherwise, it shall return a null pointer.

Error Code

None.

See Also

malloc, calloc, realloc, free

8.22.14 strlen - get length of fixed size string

C Language Interface

```
#include <string.h>

size_t  strlen(const char *s);
```

Description

The strlen() function shall compute the number of bytes in the string to which s points, not including the terminating NUL character.

Return Parameter

The strlen() function shall return the length of s.

Error Code

None.

### 8.22.15 strncat - concatenate a string with part of another

C Language Interface

```
#include <string.h>

char    *strncat(char *dst, const char *src, size_t n);
```

Description

The strncat() function shall append not more than n bytes (a NUL character and bytes that follow it are not appended) from the array pointed to by src to the end of the string pointed to by dst. The initial byte of src overwrites the NUL character at the end of dst. A terminating NUL character is always appended to the result. If copying takes place between objects that overlap, the behavior is undefined.

Return Parameter

The strncat() function shall return dst.

Error Code

None.

See Also

strcat, strcpy, strncpy

### 8.22.16 strncmp - compare part of two strings

C Language Interface

```
#include <string.h>

int     strncmp(const char *s1, const char *s2, size_t n);
```

Description

The strncmp() function shall compare not more than n bytes (bytes that follow a NUL character are not compared) from the array pointed to by s1 to the array pointed to by s2.

Return Parameter

Upon successful completion, strncmp() shall return an integer greater than, equal to, or less than 0, if the possibly null-terminated array pointed to by s1 is greater than, equal to, or less than the possibly null-terminated array pointed to by s2 respectively.

Error Code

None.

See Also

strcmp, memcmp, bcmp

8.22.17 strncpy - copy fixed length string

C Language Interface

#include <string.h>

char *strncpy(char *dst, const char *src, size_t n);

Description

The strncpy() function shall copy not more than n bytes (bytes that follow a NUL character are not copied) from the array pointed to by src to the array pointed to by dst.

If the length of the string pointed to by src is shorter than n bytes, NULL characters shall be appended to tje copy in the array pointed to by dst, until n bytes in all are written.

If copying takes place between objects that overlap, the behavior is undefined.

Return Parameter

The strncpy() function shall return dst.

Error Code

None.

See Also

strcpy, bcopy, memcpy, memmove

8.22.18 strpbrk - scan a string for a byte

C Language Interface

#include <string.h>

char *strpbrk(const char *s1, const char *s2);

Description

The strpbrk() function shall locate the first occurrence in the string pointed to by s1 of any character from the string pointed to by s2.

Return Parameter

Upon successful completion, strpbrk() shall return a pointer to the character or a null pointer if no character from s2 occurs in s1.

Error Code

None.

See Also

strchr, strrchr, strspn

8.22.19 strrchr - string scanning operation

C Language Interface

#include <string.h>

char *strrchr(const char *s, int c);

Description

The strrchr() function shall locate the last occurrence of c (converted to a char) in the string pointed to by s. The terminating NUL character is considered to be part of the string.

Return Parameter

Upon successful completion, strrchr() shall return a pointer to the byte or a null pointer if c does not occur in the string.

Error Code

None.

See Also

strchr, index, rindex

8.22.20 strspn - get length of a substring

C Language Interface

#include <string.h>

size_t  strspn(const char *s1, const char *s2);

Description

The strspn() function shall compute the length (in bytes) of the maximum initial segment of the string pointed to by s1 which consists entirely of bytes from the string pointed to by s2.

Return Parameter

The strspn() function shall return the computed length.

Error Code

None.

See Also

strcspn, strchr, strpbrk, strstr

8.22.21 strstr - find a substring

C Language Interface

#include <string.h>

char    *strstr(const char *s1, const char *s2);

Description

The strstr() function shall locate the first occurrence in the string pointed to by s1 of the sequence of bytes (excluding the terminating NUL character) in the string pointed to by s2.

Return Parameter

Upon successful completion, strstr() shall return a pointer to the located string or a null pointer if the string is not found.

Error Code

None.

See Also

strspn, strpbrk

8.22.22 strtok_r - split string into tokens

C Language Interface

#include <string.h>

char    *strtok_r(char *str, const char *sep, char **lasts);

Description

The strtok_r() function considers the null-terminated string str as a sequence of zero or more text tokens separated by spans of one or more characters from the separator string sep. The argument lasts points to a user-provided pointer which points to stored information necessary for strtok_r() to continue scanning the same string.

In the first call to strtok_r(), str points to a null-terminated string, sep to a null-terminated string of separator characters, and the value pointed to by lasts is ignored. The strtok_r() function shall return a pointer to the first character of the first token, write a null character into str immediately following the returned token, and update the pointer to which lasts points.

In subsequent calls, str must be a null pointer and lasts shall be unchanged from the previous call so that subsequent calls shall move through the string str, returning successive tokens until no tokens remain. The separator string sep may be different from call to call. When no token remains in the string, a null pointer shall be returned.

Return Parameter

The strtok_r() function shall return a pointer to the token found, or a null pointer when no token is found.

Error Code

None.

8.22.23 strxfrm - string transformation

C Language Interface

#include <string.h>

size_t  strxfrm(char *dst, const char *src, size_t n);

Description

The strxfrm() function shall transform the string pointed to by src and place the resulting string into the area pointed to by dst.

The transformation is such that if strcmp() is applied to the two transformed strings, it shall return a value greater than, equal to, or less than 0, corresponding to the result of strcoll() respectively, applied to the two original strings.

No more than n bytes are placed into the resulting array pointed to by dst, including the terminating NUL character. If n is 0, dst is permitted to be a null pointer.

If copying takes place between objects that overlap, the behavior is undefined.

Return Parameter

Upon successful completion, strxfrm() shall return the length of the transformed string (not including the terminating NUL character).
If the value returned is n or more, the contents of the array pointed to by dst are unspecified.

Error Code

None.

See Also

strncpy

8.22.24 strerror_r - get error message string

C Language Interface

#include <string.h>

int	strerror_r(int errnum, char *buf, size_t buflen);

Description

The strerror_r() function shall map the error number in errnum to a error message string in system locale and shall return the string in the buffer pointed to by buf, with length buflen.

Return Parameter

Upon successful completion, strerror_r() shall return 0. Otherwise, an error number shall be returned to indicate the error.

Error Code

| | |
|---|---|
| EINVAL | The value of errnum is not a valid error number. |
| ERANGE | Insufficient storage was supplied via buf and buflen to contain the generated message string. |

8.22.25 strercd_r - get error message string for error code

C Language Interface

#include <string.h>

int	strercd_r(ER ercd, char *buf, size_t buflen);

Description

The strercd_r() function shall map the error code in ercd to a error message string in system locale and shall return the string in the buffer pointed to by buf, with length buflen.

Return Parameter

Upon successful completion, strercd_r() shall return 0. Otherwise, an error number shall be returned to indicate the error.

Error Code

| | |
|---|---|
| EINVAL | The value of ercd is not a valid error code. |
| ERANGE | Insufficient storage was supplied via buf and buflen to contain the generated message string. |

Additional Notes

strercd_r() is a T2EX-specific function.

## 8.23 strings.h

The header strings.h defines the following functions.

Those functions that have been deleted from the latest POSIX (IEEE Std 1003.1-2008) specification but are still used generally in UNIX operating systems, are provided without being deleted for portability.
These functions are commented as "/* LEGACY */".

### Functions

### 8.23.1 bcmp - compare byte sequences

C Language Interface

```
#include <strings.h>

int     bcmp(const void *s1, const void *s2, size_t n); /* LEGACY */
```

Description

The bcmp() function shall compare the first n bytes of the object pointed to by s1 to the first n bytes of the object pointed to by s2.
If they are equal, and in particular if n is zero, bcmp() returns 0.

Return Parameter

bcmp() function returns 0 if the n byte sequences are equal, otherwise a nonzero result is returned.

Error Code

None.

See Also

memcmp, strcmp

Additional Notes

This function is removed in POSIX(IEEE Std 1003.1-2008).
It is desirable to use memcmp() instead.

### 8.23.2 bcopy - copy byte sequence

C Language Interface

```
#include <strings.h>

void    bcopy(const void *dst, void *src, size_t n); /* LEGACY */
```

Description

The bcopy() function shall copy n bytes from the object pointed to by src into the object pointed to by dst.
If copying takes place between objects that overlap, the result is correct.

Return Parameter

None.

Error Code

None.

See Also

memmove, memcpy, strcpy

Additional Notes

This function is removed in POSIX(IEEE Std 1003.1-2008).
It is desirable to use memmove() instead.

8.23.3 bzero - write zero-valued bytes in memory

C Language Interface

#include <strings.h>

void    bzero(void *s, size_t n); /* LEGACY */

Description

The bzero() function shall copy zero into each of the first n bytes of the object pointed to by s.

Return Parameter

None.

Error Code

None.

See Also

memset

Additional Notes

This function is removed in POSIX(IEEE Std 1003.1-2008).
It is desirable to use memset() instead.

8.23.4 ffs - find first set bit

C Language Interface

#include <strings.h>

int     ffs(int i);

Description

The ffs() function shall find the first bit set (beginning with the least significant bit) in i, and return the index of that bit. Bits are numbered starting at one (the least significant bit).

Return Parameter

ffs() function shall return the index of the first bit set. If i is 0, then ffs() shall return 0.

Error Code

None.

8.23.5 index, rindex - string scanning operation

C Language Interface

```
#include <strings.h>

char    *index(const char *s, int c);   /* LEGACY */
char    *rindex(const char *s, int c);  /* LEGACY */
```

Description

The index() function shall locate the first occurrence of c (converted to a char) in the string pointed to by s.
The rindex() function shall locate the last occurrence of c (converted to a char) in the string pointed to by s.
The terminating NUL character is considered to be part of the string.

Return Parameter

Upon successful completion, index() and rindex() shall return a pointer to the byte or a null pointer if c does not occur in the string.

Error Code

None.

See Also

strchr, strrchr

Additional Notes

These functions are removed in POSIX(IEEE Std 1003.1-2008).
It is desirable to use strchr() or strrchr() instead.

8.23.6 strcasecmp, strncasecmp - case-insensitive string comparisons

C Language Interface

```
#include <strings.h>

int     strcasecmp(const char *s1, const char *s2);
int     strncasecmp(const char *s1, const char *s2, size_t n);
```

Description

The strcasecmp() functions shall compare, while ignoring differences in case, the string pointed to by s1 to the string pointed to by s2.
The strncasecmp() functions shall compare, while ignoring differences in case, not more than n bytes from the string pointed to by s1 to the string pointed to by s2.
Comparison is performed case-insensitively and in the character collation order based on the system locale.

Return Parameter

Upon completion, strcasecmp() and strncasecmp() shall return an integer greater than, equal to, or less than 0, if the string pointed to by s1 is, ignoring case, greater than, equal to, or less than the string pointed to by s2, respectively.

Error Code

None.

See Also

strcmp, strncmp

## 8.24 time.h

The header time.h defines the following time-related types and functions.

The time zone referred to by the following functions is always the system time zone.
The system time zone will never be set automatically.
Unless the system time zone has been set by executing the calendar function dt_setsystz() in advance, the initial value of the system time zone takes the implementation-dependent value.

The following types are defined.

"struct tm" structure
This structure represents the calendar time by elements.
See the "struct tm" structure in the calendar function.

time_t
Integer type representing time in seconds.
Used to represent the calendar time (total elapsed seconds from 00:00:00 Coordinated Universal Time (UTC), January 1, 1970).

Functions

### 8.24.1 asctime_r - Converts from time to string

C Language Interface

```
#include <time.h>

char    *asctime_r(const struct tm *tm, char *buf);
```

Description

The asctime_r() converts the time represented by elements in the structure pointed to by tm to the string format.
The string format is "Day of the week Month Day Hour:Minute:Second The dominical year" as below.

```
        "Thu Dec 16 09:24:58 2011¥n"
```

The string is added '¥n' at its end.

The day of the week uses "Sun", "Mon", "Tue", "Wed", "Thu", "Fri", and "Sat".
The month uses "Jan", "Feb", "Mar", "Apr", "May", "Jun", "Jul", "Aug", "Sep", "Oct", "Nov", and "Dec".

This stores the converted string in the area pointed to by buf (requires 26 bytes or more) and returns buf.
The behavior shall be undefined if the element-separated day of the week (tm_wday) or month (tm_mon) in tm is an out-of-range value, if (tm_year - 1900) exceeds INT_MAX, or if the result length including the terminating null character exceeds 26.

Return Parameter

If successful, asctime_r() returns buf.
If conversion is not possible, NULL is returned.

Error Code

None

See Also

ctime_r, gmtime_r, localtime_r

8.24.2 ctime_r - Converts from calendar time to string

C Language Interface

```
#include <time.h>

char    *ctime_r(const time_t *clock, char *buf);
```

Description

The ctime_r() converts the calendar time pointed to by "clock" (total elapsed seconds from 00:00:00 UTC, January 1, 1970) to a string format as the local time in the system time zone and stores it in the area pointed to by buf (requires 26 bytes or more).

For string format, see asctime_r().

Return Parameter

If successful, ctime_r() returns buf.
If conversion is not possible, NULL is returned.

Error Code

None

See Also

asctime_r, gmtime_r, localtime_r

8.24.3 gmtime_r_eno, gmtime_r - Converts from calendar time to element-separated UTC time

C Language Interface

```
#include <time.h>

struct tm       *gmtime_r_eno(const time_t *clock, struct tm *result, errno_t *enop);
struct tm       *gmtime_r(const time_t *clock, struct tm *result);
```

Description

The gmtime_r_eno() and gmtime_r() convert the calendar time pointed to by clock (total elapsed seconds from 00:00:00 UTC, January 1, 1970) to the element-separated time in the Coordinated Universal Time(UTC) and store it in the struct tm type data pointed to by result.
If conversion is not possible, the error number is stored in the area pointed to by enop.
If enop is NULL, the error number is not stored.

The gmtime_r() is equivalent to gmtime_r_eno(clock, tm, NULL).

Return Parameter

If successful, gmtime_r_eno() and gmtime_r() returns result.
If conversion is not possible, the error number is stored in the area pointed to by enop and NULL is returned.

Error Code

Error number returned to the area pointed to by enop:

| | |
|---|---|
| EOVERFLOW | The result is not in the representable range |

See Also

asctime_r, ctime_r, localtime_r

8.24.4 localtime_r_eno, localtime_r - Converts from calendar time to element-separated local time

C Language Interface

```
#include <time.h>

struct tm       *localtime_r_eno(const time_t *clock, struct tm *result, errno_t *enop);
struct tm       *localtime_r(const time_t *clock, struct tm *result);
```

Description

The localtime_r_eno() converts the calendar time pointed to by clock (total elapsed seconds from 00:00:00 UTC, January 1, 1970) to the element-separated local time in the system time zone, stores it in the struct tm type data pointed to by result, and returns the result.
If conversion is not possible, the error number is stored in the area pointed to by enop.
If enop is NULL, the error number is not stored.

The localtime_r() is equivalent to localtime_r_eno(clock, result, NULL).

Return Parameter

If successful, localtime_r_eno() and localtime_r() returns result.
If conversion is not possible, NULL is returned.

Error Code

Error number returned to the area pointed to by enop:

| | |
|---|---|
| EOVERFLOW | The result is not in the representable range |

See Also

asctime_r, ctime_r, gmtime_r

8.24.5 mktime_eno, mktime - Converts from element-separated local time to calendar time

C Language Interface

```
#include <time.h>

time_t  mktime_eno(struct tm *, errno_t* enop);
time_t  mktime(struct tm *tm);
```

Description

The mktime_eno() converts the time elements represented as the local time pointed to by tm to the calendar time (total elapsed seconds from 00:00:00 UTC, January 1, 1970).
This ignores the initial values of tm_wday and tm_yday in tm.
Other elements also do not need to have their initial values inside the correct range.
When the conversion is successful, tm_wday and tm_yday are set to the appropriate values and other elements set to inside the correct range.
The tm_mday is not set until tm_mon and tm_year are decided.

If conversion is not possible, the error number is stored in the area pointed to by enop.
If enop is NULL, the error number is not stored.

mktime() is equivalent to mktime_eno(tm, NULL).

Return Parameter

The mktime_eno() and mktime() return the calendar time (total elapsed seconds from 00:00:00 UTC, January 1, 1970).
They return ((time_t) (-1)) if the calendar time (total elapsed seconds from 00:00:00 UTC, January 1, 1970) is not representable.

Error Code

Error number returned to the area pointed to by enop:
EOVERFLOW  The result is not in the representable range

See Also

ctime_r, gmtime_r, localtime_r

8.24.6 time - Obtains current time

C Language Interface

#include <time.h>

time_t  time(time_t *tloc);

Description

The time() returns the current time as the total elapsed seconds from 00:00:00 UTC, January 1, 1970.
If the argument tloc is not NULL, the return code is stored also in the area pointed to by tloc.

Return Parameter

If successful, time() returns the current time.
If failed, it returns ((time_t) (-1)).

Error Code

None

8.24.7 difftime - Calculates time difference between calendar times

C Language Interface

#include <time.h>

double  difftime(time_t time1, time_t time0);

Description

The difftime() obtains the difference between two calendar times (time1 - time0).

Return Parameter

The difftime() returns the difference between the numbers of seconds in the double type.

Error Code

None

8.24.8 strftime - Converts from date and time to string

C Language Interface

#include <time.h>

size_t  strftime(char *s, size_t max, const char *format, const struct tm *tm);

Description

The strftime() converts the time pointed to by tm to a string according to the format specified by format and writes the result to the string s of up to max characters including the null character.

format is a character string representing the conversion format, containing 0 or more conversion specification strings (conversion specifiers) and ordinary characters.
The details of format is identical with format for dt_strftime() in Chapter 6 Calendar Functions.

Return Parameter

If successful, strftime() returns the number of bytes of the result which was stored to "s" excluding the null character.
Otherwise, it returns 0 and the area pointed to by s becomes undefined.

Error Code

None

See Also

dt_strftime

### 8.24.9 strptime - Converts from string to date and time

C Language Interface

```
#include <time.h>

char    *strptime(const char *str, const char *format, struct tm *tm);
```

Description

The strptime() converts the string pointed to by str to the time in tm structure according to the format specified by format and stores it into tm.
The details of format is identical with format for dt_strptime() in Chapter 6 Calendar Functions.

Return Parameter

If successful, strptime() returns the pointer to the character next to the character last analyzed.
Otherwise, it returns NULL.

Error Code

None

See Also

dt_strptime

## 8.25 wchar.h

The header wchar.h provides the following wide character-related definition.

Only the following type is defined.
T2EX does not provide wide character-related library functions and thus defines only the wide character type.
Many other definitions defined in the standard C library are not defined in T2EX.

wchar_t
: An integer type that has the range of values representing the code capable of distinguishing all the members of system locale extended character set.
This is also defined in stddef.h.

Appendix

A.1 System Configuration Information

T2EX uses the T-Kernel 2.0 system configuration information management function in order to hold and manage the system-related setting information.
This section describes the standard system management information defined by T2EX.

For the system configuration information management function itself, see Section 5.7 in the T-Kernel 2.0 Specification.

○ T-Kernel 2.0

In the T-Kernel 2.0 specification, TSVCLimit sets both of the callable system call level and the protection level of the memory allocated by Kmalloc/Vmalloc.
In T2EX, only the former is set by TSVCLimit, and the latter is set independently by the parameter TKMallocLevel.

| | | |
|---|---|---|
| N | TSVCLimit | Lowest protection level for T-Kernel 2.0 and T2EX system calls<br>T2EX takes this value as 2 and does not guarantee the behavior if this value is changed. |
| N | TKmallocLvl | Protection level of the memory allocated by Kmalloc/Vmalloc<br>T2EX takes this value as 1 and does not guarantee the behavior if this value is changed.<br>The value of TSVCLimit is used if this parameter is not set. |

For other T-Kernel 2.0-derived standard system configuration information, see Section 5.7 in the T-Kernel 2.0 Specification.

○ File Management Function

System Configuration Information of Entire File Management Function

| | | |
|---|---|---|
| N | FsMaxFile | Number of files that can be opened at the same time (system-wide) |
| N | FsMaxFIMP | Number of file system implementation parts that can be registered at the same time |
| N | FsMaxCON | Number of connections that can be connected at the same time |
| N | FsAccessTime | Whether or not to allow updating the last access time |

System Configuration Information of FAT File System Implementation Part

| | | |
|---|---|---|
| N | FiFAT_TskPri | Task priority |
| N | FiFAT_StkSz | Task stack size |
| N | FiFAT_FCacheSz | FAT cache size (bytes) |
| N | FiFAT_FCacheNs | Number of FAT cache unit sectors |
| N | FiFAT_RCacheSz | Root directory cache size (bytes) |
| N | FiFAT_RCacheNs | Number of root directory cache unit sectors |
| N | FiFAT_DCacheSz | Data cache size (bytes) |
| N | FiFAT_DCacheNs | Number of data cache unit sectors |

○ Network Communication Function

| | | |
|---|---|---|
| N | SoMaxSocket | Maximum number of sockets that can be opened at the same time |
| N | SoTaskBasePri | Maximum priority of tasks created by the network communication manager.<br>The network communication manager creates tasks whose priorites are ranging from SoTaskBasePri to (SoTaskBasePri + 4). |
| N | SoDrvRxBufNum | Number of buffers registered in the LAN driver. |
| N | SoTcpSendBufSz | Default buffer size for TCP/IP send queue (in bytes) |
| N | SoTcpRecvBufSz | Default buffer size for TCP/IP receive queue (in bytes) |
| N | SoUdpSendBufSz | Default buffer size for UDP/IP send queue (in bytes) |
| N | SoUdpRecvBufSz | Default buffer size for UDP/IP receive queue (in bytes) |
| N | SoRawIPSendBufSz | Default buffer size for SOCK_RAW type socket (AF_INET) send queue (in bytes) |
| N | SoRawIPRecvBufSz | Default buffer size for SOCK_RAW type socket (AF_INET) receive queue (in bytes) |
| N | SoRawSendBufSz | Default buffer size for SOCK_RAW type socket (AF_ROUTE) send queue (in bytes) |
| N | SoRawRecvBufSz | Default buffer size for SOCK_RAW type socket (AF_ROUTE) receive queue (in bytes) |

```
   N    SoTcpDoAutoSendBuf      Allow automatic resizing of TCP/IP send buffer size
                                Allows automatic resizing if the value is other than 0.
   N    SoTcpIncAutoSendBufSz   Incremental size of automatic resizing of TCP/IP send buffer size (in
bytes)
   N    SoTcpMaxAutoSendBufSz   Maximum size after automatic resizing of TCP/IP send buffer size (in
bytes)
   N    SoTcpDoAutoRecvBuf      Allow automatic resizing of TCP/IP receive buffer size
                                Allows automatic resizing if the value is other than 0.
   N    SoTcpIncAutoRecvBufSz   Incremental size of automatic resizing of TCP/IP receive buffer size
(in bytes)
   N    SoTcpMaxAutoRecvBufSz   Maximum size after automatic resizing of TCP/IP receive buffer size
(in bytes)
```

## A.2 Usage Example of Break API Call (fs_break, so_break)

This section shows a usage example of the break API calls (fs_break, so_break) in the file management and network communication functions.

The break API calls release a wait for an input/output processing to/from a file or network to safely abort the input/output processing.
They are used to safely terminate the input/output processing being executed by a task when the processing is cancelled by a user operation or due to a system factor such as low remaining power and shutdown.
They are equivalent to the T-Kernel 2.0 API tk_rel_wai, but release only a wait for file and network management functions respectively.

As a usage example of the break API calls, this section shows how to abort a reading processing that reads cnt bytes from the socket descriptor sd.
Section A.2.1 shows the implementation example of the reading processing itself. Section A.1.2 shows the implementation example of the abort processing of it.

### A.2.1 Implementation Example of Reading Processing

This example assumes that the socket reading processing is executed by a task that has the priority SOCKET_READ_TASK_PRI.
To support external abort, the following implementation example of the startup function defines and uses the variables "interrupted" which indicates whether abort occurs or not and "finished" which indicates the completion of abort.

```
LOCAL BOOL finished = FALSE;
LOCAL BOOL interrupted = FALSE;

void socketReadTask( INT stacd, void* exinf )
{
    int sd = (int)stacd;
    int pos;
    ER er;

    for (pos = 0; pos < cnt; !interrupted) {
        /* so_read may begin waiting */
        er = so_read(sd, buf + pos, cnt - pos);
        if (er <= 0)
            break;
        pos += er;
    }

    /* notify the abort processing side that the abort processing was normally executed */
    finished = TRUE;

    /* terminate the task */
    tk_ext_tsk();
}
```

### A.2.2 Implementation Example of Abort Processing

For the implementation in Section A.2.1, an abort processing using the break API call can be implemented as follows.

```
void interruptSocketReadTask(ID tskid)
```

```
{
    ER er;

    /* reduce the priority of the task for which the abort processing is executed
       (execute so_break only when the target task is waiting) */
    tk_chg_pri(TSK_SELF, SOCKET_READ_TASK_PRI + 1);

    /* abort processing of the task */
    while (!finished) {
        interrupted = TRUE;
        so_break(tskid);
    }
}
```

The basic concept is to use so_break to release so_read from waiting and abort the processing.
Note that a release request by so_break is not queued and does not take effect when the break API call is invoked before or after executing so_read.

The implementation example in this section repeatedly executes so_break until the variable finished becomes true to confirm the processing was normally aborted.
This guarantees the target processing is surely be aborted.

## A.3 Usage Example of Regular Program Module Function

As a usage example of the regular program module function, this section shows an example of implementing the data structure stack as a regular program module.

### A.3.1 Definition Example of Regular Program Module

The following shows an example of defining a regular program module that realizes the push and pop operations of the stack for the int type element.
Though the error handling is omitted for simplification, it is recommended to handle errors correctly in an actual program module.

○ stack.h

```
/*
 * Stack module external interface definition (stack.h)
 */
#ifndef DEFINE_H_STACK
#define DEFINE_H_STACK

typedef struct stack_module {
    /* parameter for initializing the module */
    int size;

    /* interface functions of the module */
    void (*push)(int data);
    void (*pop)(int* pdata);
} STACK_MODULE;

#endif
```

○ stack.c

```
/*
 * Stack module implementation (stack.c)
 */
#include <tk/tkernel.h>
#include "stack.h"

LOCAL FastLock lock;
LOCAL int*     stack;
LOCAL int      sp;

LOCAL void stack_push(STACK_MODULE* sm, int data)
{
    Lock(&lock);
    stack[sp++] = data;
    Unlock(&lock);
}
```

```
LOCAL void stack_pop(STACK_MODULE* sm, int* pdata)
{
    Lock(&lock);
    *pdata = stack[--sp];
    Unlock(&lock);
}

EXPORT int module_main(BOOL startup, void* arg)
{
    STACK_MODULE* sm = (STACK_MODULE*)arg;

    if (startup) {
        /* initialization */
        CreateLock(&lock, "stk1");
        stack = (int*)malloc(sm->size * sizeof(int));
        sp = 0;
        sm->push = stack_push;
        sm->pop = stack_pop;
    }
    else {
        /* termination */
        free(stack);
        DeleteLock(&lock);
    }

    return 0;
}
```

### A.3.2 Usage Example of Regular Program Module

This section shows a code example of using the stack module defined in Section A.3.1 from an application program.

○ sample.c

```
/*
 * Stack usage (sample.c)
 */
#include <basic.h>
#include <t2ex/load.h>
#include "stack.h"

LOCAL ID smId = E_NOEXS;
LOCAL STACK_MODULE smif;
LOCAL pm_entry_t* sm_main = NULL;

EXPORT void sample()
{
    ER er;
    int data1, data2;
    struct pm target = {
        .pmatr = PM_FILE,
        .pmhdr = "/sysdsk/module/stack.obj",
    };

    /* load the module*/
    er = pm_load(&target, &sm_main);
    if (er < E_OK) {
        fprintf(stderr, "pm_load error: %d¥n", er);
        return;
    }
    smId = er;

    /* initialize the module */
    smif.size = 100;
    sm_main(TRUE, &smif);

    /* use the module */
    smif.push(1);
    smif.push(2);
    smif.pop(&data1);
```

```
    smif.push(3);
    smif.pop(&data2);

    /* terminate the module */
    sm_main(FALSE, &smif);

    pm_unload(smId);
    sm_main = NULL;
    smId = E_NOEXS;
}
```