librudp Reference

PlayStation®Vita Programmer Tool Runtime Library Release 3.50

Table of Contents

Initialization, Configuration, Termination APIs

sceRudpInit

Initialize librudp

Definition

```
#include <rudp.h>
int sceRudpInit(
        void *pool,
        size_t poolSize
)
```

Calling Conditions

Multithread safe.

Arguments

pool	Pointer to the memory pool to be used by librudp
poolSize	Size of the memory pool to be used by librudp (bytes)

Return Values

Returns `SCE_RUDP_SUCCESS(0)` for normal termination.

Returns a negative value for errors. The main error codes are shown below. Note, however, that the application must not malfunction even if other error codes are returned.

Value	Description
`SCE_RUDP_ERROR_ALREADY_INITIALIZED`	Already initialized. `sceRudpInit()` may have been called again before `sceRudpEnd()`. Check the calling order.
`SCE_RUDP_ERROR_MEMORY`	Could not allocate memory from the heap. This error does not occur during normal operation.

Description

This function initializes librudp. Allocate a memory pool in advance by the application and provide it to librudp. Since the necessary memory varies significantly depending on usage, it is necessary to find out memory used space at the stage of development. Check the maximum value of allocated memory size (*memPeak*) using `sceRudpGetStatus()`.

The memory pool provided to librudp will be used until `sceRudpEnd()` is called.

Examples

```
#define RUDP_POOL_SIZE (500*1024);
uint8_t rudp_pool[RUDP_POOL_SIZE];

ret = sceRudpInit(rudp_pool, RUDP_POOL_SIZE);
if ( sceRudpInit(&myAllocator ) < 0 ) {
        // Error handling
}
```

See Also

`sceRudpEnd()`

sceRudpEnd

Terminate librudp

Definition

```
#include <rudp.h>
int sceRudpEnd(
        void
)
```

Calling Conditions

Multithread safe.

Arguments

None

Return Values

Returns `SCE_RUDP_SUCCESS(0)` for normal termination.

Returns a negative value for errors. The main error codes are shown below. Note, however, that the application must not malfunction even if other error codes are returned.

Value	Description
`SCE_RUDP_ERROR_NOT_INITIALIZED`	Not initialized. `sceRudpInit()` has not been called, or `sceRudpEnd()` has already been called. Check the calling order.

Description

This function terminates the library.

All threads in blocking state will be unblocked as soon as this function is called, and the blocking functions will return `SCE_RUDP_ERROR_CANCELLED`.

See Also

`sceRudpInit()`

sceRudpEnableInternalIOThread

Start internal network I/O thread

Definition

```
#include <rudp.h>
int sceRudpEnableInternalIOThread(
        uint32_t stackSize,
        uint32_t priority
)
```

Calling Conditions

Multithread safe.

Arguments

stackSize	Stack size of internal network I/O thread
priority	Priority of internal network I/O thread

Return Values

Returns `SCE_RUDP_SUCCESS(0)` for normal termination.

Returns a negative value for errors. The main error codes are shown below. Note, however, that the application must not malfunction even if other error codes are returned.

Value	Description
`SCE_RUDP_ERROR_NOT_INITIALIZED`	Not initialized. `sceRudpInit()` has not been called, or `sceRudpEnd()` has already been called. Check the calling order.
`SCE_RUDP_ERROR_NOT_ACCEPTABLE`	Operation is not permitted. `sceRudpCreateContext()` may have been called already. Check the calling order.
`SCE_RUDP_ERROR_THREAD_IN_USE`	Internal I/O thread is already being used. This function may have been called already.
`SCE_RUDP_ERROR_THREAD`	An error was detected while creating the internal I/O thread. This error does not occur during normal operation.
`SCE_RUDP_ERROR_MEMORY`	Could not allocate memory from the heap. This error does not occur during normal operation.
`SCE_RUDP_ERROR_INVALID_SOCKET`	An error was detected in the initial settings of the internal network. Check that libnet was correctly initialized.

Description

This function starts up the internal network I/O thread.

To *`stackSize`*, specify the size of the stack memory to be used by the internal network I/O thread. Because all callbacks in librudp will be called from this thread, it is also necessary to take into account the size of the stack memory used by the application during callbacks. librudp uses a stack size of up to 4 KB, so the value to set to *`stackSize`* is approximately 4 KB plus the stack size for the callback. If a value under 4 KB is specified, *`stackSize`* will be set internally to 4 KB.

To *`priority`*, specify the priority of the internal network I/O thread. Specify a value in the range described in the "sceKernelCreateThread" section of the "Kernel Reference" document.

The values set to *`stackSize`* and *`priority`* are passed to the arguments *`stackSize`* and *`initPriority`*, respectively, of the kernel system call `sceKernelCreateThread()`.

Notes

The internal network I/O thread will be terminated (JOINED) when `sceRudpEnd()` is called.

sceRudpSetEventHandler

Register a common event handler

Definition

```
#include <rudp.h>
int sceRudpSetEventHandler(
        SceRudpEventHandler handler,
        void *arg
)
```

Calling Conditions

Multithread safe.

Arguments

handler	Callback function
arg	Address to pass as an argument to the callback function

Return Values

Returns `SCE_RUDP_SUCCESS(0)` for normal termination.

Returns a negative value for errors. The main error codes are shown below. Note, however, that the application must not malfunction even if other error codes are returned.

Value	Description
`SCE_RUDP_ERROR_NOT_INITIALIZED`	Not initialized. `sceRudpInit()` has not been called, or `sceRudpEnd()` has already been called. Check the calling order.
`SCE_RUDP_ERROR_NO_EVENT_HANDLER`	Using NULL in *handler*.

Description

This function registers a common event handler.

A handler must always be registered. Otherwise, the functions `sceRudpInitiate()` and `sceRudpActivate()` will fail, returning the code `SCE_RUDP_ERROR_NO_EVENT_HANDLER`.

Examples

```
extern SceRudpEventHandler myHandler;
extern void *arg;

int ret;

ret = sceRudpSetEventHandler(myHandler, arg)
if ( ret < 0 ) {
        // Error handling
}
```

See Also

`SceRudpEventHandler`

SceRudpEventHandler

Common event handler

Definition

```
#include <rudp.h>
typedef int ( *SceRudpEventHandler )(
        int eventId,
        int soc,
        uint8_t const *data,
        size_t dataLen,
        struct SceNetSockaddr const *addr,
        SceNetSocklen_t addrLen,
        void *arg
);
```

Calling Conditions

Multithread safe.

Arguments

eventId	Common event ID
soc	UDPP2P socket ID
data	Data
dataLen	Size of *data*
addr	Socket address of peer
addrLen	Size of *addr*
arg	Address passed to `sceRudpSetEventHandler()`

One of the following values will be set to *eventId*.

Please note that other values may be added in the future. The application must not malfunction even if other values are passed.

Common Event ID	(Number)	Description
`SCE_RUDP_EVENT_SEND`	1	Request to send UDP data
`SCE_RUDP_EVENT_SOCKET_RELEASED`	2	Socket was freed
`SCE_RUDP_EVENT_DIAG_SENT`	100	(For testing) UDP send data (only when the internal network I/O is used)
`SCE_RUDP_EVENT_DIAG_RCVD`	101	(For testing) UDP receive data (only when the internal network I/O is used)

Description

This is a prototype of the callback function that is called when a common event occurs. The application must be designed to behave as follows, according to the value passed to *`eventId`*.

SCE_RUDP_EVENT_SEND

This event is notified when UDP send data is to be passed to the application without using an internal network I/O thread. The information necessary for sending data will be passed to the arguments of the callback: *`soc`*, *`data`*, *`dataLen`*, *`addr`*, and *`addrLen`*. Under normal circumstances, use this information to call `sendto()` in the callback.

If the return value of `sendto()` is 0 or over, return the value as is. If the return value of `sendto()` is a negative value (indicating an error), return `SCE_RUDP_ERROR_WOULDBLOCK` if `sys_net_errno` is `SYS_NET_EAGAIN` or `SYS_NET_EWOULDBLOCK`. If `sys_net_errno` is another value, return `SCE_RUDP_ERROR_INVALID_SOCKET`.

SCE_RUDP_EVENT_SOCKET_RELEASED

This event is notified when all the contexts that used the socket ID stored to *`soc`* have been deleted. Only the argument *`soc`* is valid; do not access any other arguments.

Always return `SCE_RUDP_SUCCESS(0)`.

SCE_RUDP_EVENT_DIAG_SENT / SCE_RUDP_EVENT_DIAG_RCVD

When an internal network I/O thread is used, these events are notified immediately before `sendto()` is called, or immediately after `recvfrom()` is called, respectively. Valid values are stored to *`soc`*, *`data`*, *`dataLen`*, *`addr`*, and *`addrLen`*.

Under normal circumstances, return immediately with the return value `SCE_RUDP_SUCCESS(0)`. During testing, it is possible to emulate a packet loss by returning a negative value, since this will cause the internal network I/O thread to skip the send/receive operation.

Other Values

Common events may be added in the future. Thus it is possible for other values to be passed to *`eventId`*. In such cases, always return `SCE_RUDP_SUCCESS(0)`.

See Also

`sceRudpSetEventHandler()`

sceRudpSetMaxSegmentSize

Set the maximum segment size (MSS)

Definition

```
#include <rudp.h>
int sceRudpSetMaxSegmentSize(
        uint16_t mss
)
```

Calling Conditions

Multithread safe.

Arguments

mss	Maximum size of the RUDP segment

Return Values

Returns `SCE_RUDP_SUCCESS(0)` for normal termination.

Returns a negative value for errors. The main error codes are shown below. Note, however, that the application must not malfunction even if other error codes are returned.

Value	Description
`SCE_RUDP_ERROR_NOT_INITIALIZED`	Not initialized. `sceRudpInit()` has not been called, or `sceRudpEnd()` has already been called. Check the calling order.
`SCE_RUDP_ERROR_NOT_ACCEPTABLE`	Operation is not permitted. `sceRudpCreateContext()` may have been called already. Check the calling order.

Description

This function sets the maximum segment length sent by librudp (in other words, the maximum payload size in UDP).

This function must be called before a context is created with `sceRudpCreateContext()`. The default for the maximum segment size is 1410 bytes.

See Also

`sceRudpGetMaxSegmentSize()`

sceRudpGetMaxSegmentSize

Get the maximum segment size (MSS)

Definition

```
#include <rudp.h>
int sceRudpGetMaxSegmentSize(
        uint16_t *mss
)
```

Calling Conditions

Multithread safe.

Arguments

mss	Pointer to area storing the maximum segment size

Return Values

Returns `SCE_RUDP_SUCCESS(0)` for normal termination.

Returns a negative value for errors. The main error codes are shown below. Note, however, that the application must not malfunction even if other error codes are returned.

Value	Description
`SCE_RUDP_ERROR_NOT_INITIALIZED`	Not initialized. `sceRudpInit()` has not been called, or `sceRudpEnd()` has already been called. Check the calling order.
`SCE_RUDP_ERROR_INVALID_ARGUMENT`	NULL was specified to *mss*

Description

This function gets the value set to librudp indicating the maximum segment length that can be sent by the library (in other words, the maximum payload size in UDP).

See Also

`sceRudpSetMaxSegmentSize()`

Context and Option Configuration APIs

sceRudpCreateContext

Create a context

Definition

```
#include <rudp.h>
int sceRudpCreateContext(
        SceRudpContextEventHandler handler,
        void *arg,
        int *ctxId
)
```

Calling Conditions

Multithread safe.

Arguments

handler	Callback function from the context
arg	Address to pass as an argument to the callback function
ctxId	Pointer to area to store the context ID

Return Values

Returns `SCE_RUDP_SUCCESS(0)` for normal termination.

Returns a negative value for errors. The main error codes are shown below. Note, however, that the application must not malfunction even if other error codes are returned.

Value	Description
`SCE_RUDP_ERROR_NOT_INITIALIZED`	Not initialized. `sceRudpInit()` has not been called, or `sceRudpEnd()` has already been called. Check the calling order.
`SCE_RUDP_ERROR_INVALID_ARGUMENT`	NULL was specified to *ctxId*
`SCE_RUDP_ERROR_MEMORY`	Could not allocate memory from the heap. This error does not occur during normal operation.

Description

This function creates a context that acts as an endpoint in RUDP communication. It is the equivalent of a socket in TCP/IP.

To *handler*, set a callback function if callbacks from the context will be necessary. If the polling functions and blocking mode provided by librudp will be used instead, specify NULL to *handler*, as callbacks from the context will not be used.

Upon normal termination, a context ID (a value of 0 or over) will be stored to **ctxId*. If an error occurs, the value of the area pointed to by *ctxId* is undefined.

Examples

```
extern SceRudpContextEventHandler ctxHandler;
extern void *arg;
extern int ctxId;

int ret;

ret = sceRudpCreateContext(ctxHandler, arg, &ctxId)
if ( ret < 0 ) {
        // Error handling
}
```

See Also

`SceRudpContextEventHandler`, `sceRudpTerminate()`

SceRudpContextEventHandler

Context event handler

Definition

```
#include <rudp.h>
typedef void ( *SceRudpContextEventHandler )(
        int ctxId,
        int eventId,
        int errorCode,
        void *arg
);
```

Calling Conditions

Multithread safe.

Arguments

ctxId	Context ID
eventId	Event ID
errorCode	Error code
arg	Address passed to `sceRudpCreateContext()`

One of the following values will be set to *eventId*.

Please note that other values may be added in the future. The application must not malfunction even if other values are passed.

Context Event ID	(Number)	Description
`SCE_RUDP_CONTEXT_EVENT_CLOSED`	1	Connection failed, or was closed
`SCE_RUDP_CONTEXT_EVENT_ESTABLISHED`	2	Connection was established
`SCE_RUDP_CONTEXT_EVENT_ERROR`	3	An error occurred
`SCE_RUDP_CONTEXT_EVENT_WRITABLE`	4	Writes became enabled
`SCE_RUDP_CONTEXT_EVENT_READABLE`	5	Reads became enabled
`SCE_RUDP_CONTEXT_EVENT_FLUSHED`	6	Send data was flushed

Return Values

None

Description

Context communication events are notified to this callback function.

To *ctxId*, the context ID of the event will be passed.

An error code (`SCE_RUDP_ERROR_XXX`) will be passed to *errorCode* only if *eventId* is `SCE_RUDP_CONTEXT_EVENT_CLOSED` or `SCE_RUDP_CONTEXT_EVENT_ERROR`.

See Also

`sceRudpCreateContext()`

sceRudpSetOption

Set context options

Definition

```
#include <rudp.h>
int sceRudpSetOption(
        int ctxId,
        int option,
        void const *optVal,
        size_t optLen
)
```

Calling Conditions

Multithread safe.

Arguments

ctxId	Context ID
option	Context option
optVal	Pointer to area storing the context option value
optLen	Size of the context option value

Return Values

Returns `SCE_RUDP_SUCCESS(0)` for normal termination.

Returns a negative value for errors. The main error codes are shown below. Note, however, that the application must not malfunction even if other error codes are returned.

Value	Description
SCE_RUDP_ERROR_NOT_INITIALIZED	Not initialized. `sceRudpInit()` has not been called, or `sceRudpEnd()` has already been called. Check the calling order.
SCE_RUDP_ERROR_INVALID_ARGUMENT	A negative value was set to *ctxId*, NULL was specified to *optVal*, or an invalid value was set to *optLen*.
SCE_RUDP_ERROR_INVALID_CONTEXT_ID	The context ID is invalid. Check the value specified to *ctxId*. Also make sure that the context itself has not already been deleted with `sceRudpTerminate()`.
SCE_RUDP_ERROR_INVALID_OPTION	An invalid value was set to *option*
SCE_RUDP_ERROR_NOT_ACCEPTABLE	The specified option cannot be set in the context's current state

Description

This function sets context options to a context.

The following context options are supported.

Option Name	`SCE_RUDP_OPTION_MAX_PAYLOAD`
Description	Maximum payload size of RUDP segment
Area Type	`uint32_t`
Area Size	`sizeof(uint32_t)`
Value	Byte size of maximum payload of RUDP segment (default: 1346 bytes)
Notes	This value must be smaller than the size of the send buffer or receive buffer. Also it must be at least 64 bytes smaller than the maximum segment size set with `sceRudpSetMaxSegmentSize()`. This option must be set in IDLE state (before starting a connection). Also refer to note in the "Maximum Size of Sendable Messages" section in the Chapter 4 "Notes" of the "librudp Overview" document.

Option Name	`SCE_RUDP_OPTION_SNDBUF`
Description	Send buffer size
Area Type	`uint32_t`
Area Size	`sizeof(uint32_t)`
Value	Byte size of send buffer (default: 65536 bytes)
Notes	This value must be larger than the maximum segment size specified with `sceRudpSetMaxSegmentSize()`. There is no upper limit. Depending on the communication status, it is possible for parts of the buffer to remain unused. This option must be set in IDLE state (before starting a connection).

Option Name	`SCE_RUDP_OPTION_RCVBUF`
Description	Receive buffer size
Area Type	`uint32_t`
Area Size	`sizeof(uint32_t)`
Value	Byte size of receive buffer (default: 65536 bytes)
Notes	This value must be larger than the maximum segment size specified with `sceRudpSetMaxSegmentSize()`. There is no upper limit. Depending on the communication status, it is possible for parts of the buffer to remain unused. This option must be set in IDLE state (before starting a connection).

Option Name	`SCE_RUDP_OPTION_NODELAY`
Description	Message aggregation for context
Area Type	`int`
Area Size	`sizeof(int)`
Value	0 Aggregate messages (default) 1 No message aggregation
Notes	Setting this option to 1 is the equivalent of setting a `SCE_RUDP_MSG_LATENCY_CRITICAL` flag to each outgoing message, although this may cause network bandwidth efficiency to deteriorate. This setting can be changed at any time.

Option Name	`SCE_RUDP_OPTION_DELIVERY_CRITICAL`
Description	Delivery Critical (DC) option
Area Type	`int`
Area Size	`sizeof(int)`
Value	0 Set DC to Off 1 Set DC to On (default)
Notes	This option must be set in IDLE state (before starting a connection).

Option Name	`SCE_RUDP_OPTION_ORDER_CRITICAL`
Description	Order Critical (OC) option
Area Type	`int`
Area Size	`sizeof(int)`
Value	0 Set OC to Off 1 Set OC to On (default)
Notes	This option must be set in IDLE state (before starting a connection).

Option Name	`SCE_RUDP_OPTION_NONBLOCK`
Description	Nonblocking mode
Area Type	`int`
Area Size	`sizeof(int)`
Value	0 Use blocking mode 1 Use nonblocking mode (default)
Notes	This option specifies whether to execute the functions `sceRudpInitiate()`, `sceRudpActivate()`, `sceRudpRead()`, `sceRudpWrite()`, and `sceRudpFlush()` as blocking or nonblocking functions. It is always possible to switch the mode. If nonblocking mode is specified while a thread is blocking, the thread will be released from blocking state, and the blocking function will return `SCE_RUDP_ERROR_CANCELLED`.

Option Name	`SCE_RUDP_OPTION_STREAM`
Description	Transport type
Area Type	`int`
Area Size	`sizeof(int)`
Value	0 Set the transport type to DGRAM (default) 1 Set the transport type to STREAM
Notes	This option must be set in IDLE state (before starting a connection).

Option Name	`SCE_RUDP_OPTION_CONNECTION_TIMEOUT`
Description	Connection timeout
Area Type	`uint32_t`
Area Size	`sizeof(uint32_t)`
Value	Time (in milliseconds) before `sceRudpInitiate()` or `sceRudpActivate()` times out (default: 60 seconds)
Notes	This option must be set in IDLE state (before starting a connection).

Option Name	`SCE_RUDP_OPTION_CLOSE_WAIT_TIMEOUT`
Description	Close-wait timeout
Area Type	`uint32_t`
Area Size	`sizeof(uint32_t)`
Value	Time (in milliseconds) for a close-wait timeout (default: 0)
Notes	This option must be set in IDLE state (before starting a connection).

Option Name	`SCE_RUDP_OPTION_AGGREGATION_TIMEOUT`
Description	Aggregation timeout
Area Type	`uint32_t`
Area Size	`sizeof(uint32_t)`
Value	Time (in milliseconds) for an aggregation timeout (default: 30)
Notes	This option causes a delay of the specified time when sending data, for message aggregation. If `SCE_RUDP_OPTION_NODELAY` is set to 1, messages will not be aggregated. (This option will be ignored.) This option must be set in IDLE state (before starting a connection).

Option Name	`SCE_RUDP_OPTION_READ_TIMEOUT`
Description	Read timeout
Area Type	`uint32_t`
Area Size	`sizeof(uint32_t)`
Value	Time (in milliseconds) before a read times out (default: 0)
Notes	This is the maximum blocking time by `sceRudpRead()` in blocking mode. If 0 is specified, the operation will not time out. In nonblocking mode, this value is meaningless. The timeout value can be changed at any time.

Option Name	`SCE_RUDP_OPTION_WRITE_TIMEOUT`
Description	Write timeout
Area Type	`uint32_t`
Area Size	`sizeof(uint32_t)`
Value	Time (in milliseconds) before a write times out (default: 0)
Notes	This is the maximum blocking time by `sceRudpWrite()` in blocking mode. If 0 is specified, the operation will not time out. In nonblocking mode, this value is meaningless. The timeout value can be changed at any time.

Option Name	`SCE_RUDP_OPTION_FLUSH_TIMEOUT`
Description	Flush timeout
Area Type	`uint32_t`
Area Size	`sizeof(uint32_t)`
Value	Time (in milliseconds) before a flush times out (default: 0)
Notes	This is the maximum blocking time by `sceRudpFlush()` in blocking mode. If 0 is specified, the operation will not time out. In nonblocking mode, this value is meaningless. The timeout value can be changed at any time.

Option Name	`SCE_RUDP_OPTION_KEEP_ALIVE_INTERVAL`
Description	Keep-alive interval
Area Type	`uint32_t`
Area Size	`sizeof(uint32_t)`
Value	Time (in milliseconds) after the last send/receive before starting the transmission of keep-alive packets (default: 0, no keep-alive)
Notes	If 0 is specified, the keep-alive feature will be disabled. The setting for this option can be changed at any time.

Option Name	SCE_RUDP_OPTION_KEEP_ALIVE_TIMEOUT
Description	Keep-alive timeout
Area Type	`uint32_t`
Area Size	`sizeof(uint32_t)`
Value	Time (in milliseconds) after starting the transmission of a keep-alive packet and no response is received (the connection is considered closed) (default: 0 – no timeout)
Notes	A keep-alive packet will continue to be resent until a timeout occurs or a response is received and another keep-alive is sent. The interval for the first resend is the larger of two values: the actual RTT measured by librudp or 1 second. Each interval for a subsequent resend will be twice as long as the previous interval. The timeout value can be changed at any time.

See Also

`sceRudpGetOption()`

sceRudpGetOption

Get context options

Definition

```
#include <rudp.h>
int sceRudpGetOption(
        int ctxId,
        int option,
        void *optVal,
        size_t optLen
)
```

Calling Conditions

Multithread safe.

Arguments

ctxId	Context ID
option	Context option
optVal	Pointer to area storing the context option value
optLen	Size of the context option value

Return Values

Returns `SCE_RUDP_SUCCESS(0)` for normal termination.

Returns a negative value for errors. The main error codes are shown below. Note, however, that the application must not malfunction even if other error codes are returned.

Value	Description
`SCE_RUDP_ERROR_NOT_INITIALIZED`	Not initialized. `sceRudpInit()` has not been called, or `sceRudpEnd()` has already been called. Check the calling order.
`SCE_RUDP_ERROR_INVALID_ARGUMENT`	A negative value was set to *ctxId*, NULL was specified to *optVal*, or an invalid value was set to *optLen*.
`SCE_RUDP_ERROR_INVALID_CONTEXT_ID`	The context ID is invalid. Check the value specified to *ctxId*. Also make sure that the context itself has not already been deleted with `sceRudpTerminate()`.
`SCE_RUDP_ERROR_INVALID_OPTION`	An invalid value was set to *option*

Description

This function gets the context options currently set to a context.

The following context options are supported.

Option Name	`SCE_RUDP_OPTION_MAX_PAYLOAD`
Description	Maximum payload size of RUDP segment
Area Type	`uint32_t`
Area Size	`sizeof(uint32_t)`
Value	Byte size of maximum payload of RUDP segment (default: 1346 bytes)

Option Name	`SCE_RUDP_OPTION_SNDBUF`
Description	Send buffer size
Area Type	`uint32_t`
Area Size	`sizeof(uint32_t)`
Value	Byte size of send buffer (default: 65536 bytes)

Option Name	`SCE_RUDP_OPTION_RCVBUF`
Description	Receive buffer size
Area Type	`uint32_t`
Area Size	`sizeof(uint32_t)`
Value	Byte size of receive buffer (default: 65536 bytes)

Option Name	`SCE_RUDP_OPTION_NODELAY`
Description	Message aggregation for context
Area Type	`int`
Area Size	`sizeof(int)`
Value	0 Aggregate messages (default) 1 No message aggregation
Notes	Setting this option to 1 is the equivalent of setting a `SCE_RUDP_MSG_LATENCY_CRITICAL` flag to each outgoing message. Though there will be no delay due to message aggregation, it is possible for network bandwidth efficiency to deteriorate.

Option Name	`SCE_RUDP_OPTION_DELIVERY_CRITICAL`
Description	Delivery Critical (DC) option
Area Type	`int`
Area Size	`sizeof(int)`
Value	0 DC is Off 1 DC is On (default)

Option Name	`SCE_RUDP_OPTION_ORDER_CRITICAL`
Description	Order Critical (OC) option
Area Type	`int`
Area Size	`sizeof(int)`
Value	0 OC is Off 1 OC is On (default)

Option Name	`SCE_RUDP_OPTION_NONBLOCK`
Description	Nonblocking mode
Area Type	`int`
Area Size	`sizeof(int)`
Value	0 Blocking mode 1 Nonblocking mode (default)
Notes	This option indicates whether the functions `sceRudpInitiate()`, `sceRudpActivate()`, `sceRudpRead()`, `sceRudpWrite()`, and `sceRudpFlush()` are executed as blocking or nonblocking functions.

Option Name	`SCE_RUDP_OPTION_STREAM`
Description	Transport type
Area Type	`int`
Area Size	`sizeof(int)`
Value	0 DGRAM (default) 1 STREAM

Option Name	`SCE_RUDP_OPTION_CONNECTION_TIMEOUT`
Description	Connection timeout
Area Type	`uint32_t`
Area Size	`sizeof(uint32_t)`
Value	Time (in milliseconds) before `sceRudpInitiate()` or `sceRudpActivate()` times out (default: 60 seconds)

Option Name	`SCE_RUDP_OPTION_CLOSE_WAIT_TIMEOUT`
Description	Close-wait timeout
Area Type	`uint32_t`
Area Size	`sizeof(uint32_t)`
Value	Time (in milliseconds) for a close-wait timeout (default: 0)

Option Name	`SCE_RUDP_OPTION_AGGREGATION_TIMEOUT`
Description	Aggregation timeout
Area Type	`uint32_t`
Area Size	`sizeof(uint32_t)`
Value	Time (in milliseconds) for an aggregation timeout (default: 30)
Notes	This option causes a delay of the specified time when sending data, for message aggregation. If `SCE_RUDP_OPTION_NODELAY` is set to 1, messages will not be aggregated. (This option will be ignored.)

Option Name	`SCE_RUDP_OPTION_LAST_ERROR`
Description	Last error
Area Type	`int`
Area Size	`sizeof(int)`
Value	Error code of most recent error (`SCE_RUDP_ERROR_XXX`)
Notes	After a WRITE event is detected, the result of a connection operation can be obtained by using the polling function for detecting connection events. When the result is read, the internal value is automatically cleared to 0.

Option Name	`SCE_RUDP_OPTION_READ_TIMEOUT`
Description	Read timeout
Area Type	`uint32_t`
Area Size	`sizeof(uint32_t)`
Value	Time (in milliseconds) before a read times out (default: 0)
Notes	This is the maximum blocking time by `sceRudpRead()` in blocking mode. If 0 is specified, the operation will not time out. In nonblocking mode, this value is meaningless.

Option Name	`SCE_RUDP_OPTION_WRITE_TIMEOUT`
Description	Write timeout
Area Type	`uint32_t`
Area Size	`sizeof(uint32_t)`
Value	Time (in milliseconds) before a write times out (default: 0)
Notes	This is the maximum blocking time by `sceRudpWrite()` in blocking mode. If 0 is specified, the operation will not time out. In nonblocking mode, this value is meaningless.

Option Name	`SCE_RUDP_OPTION_FLUSH_TIMEOUT`
Description	Flush timeout
Area Type	`uint32_t`
Area Size	`sizeof(uint32_t)`
Value	Time (in milliseconds) before a flush times out (default: 0)
Notes	This is the maximum blocking time by `sceRudpFlush()` in blocking mode. If 0 is specified, the operation will not time out. In nonblocking mode, this value is meaningless.

Option Name	`SCE_RUDP_OPTION_KEEP_ALIVE_INTERVAL`
Description	Keep-alive interval
Area Type	`uint32_t`
Area Size	`sizeof(uint32_t)`
Value	Time (in milliseconds) after the last send/receive before starting the transmission of keep-alive packets (default: 0, no keep-alive)
Notes	While 0 is set, the keep-alive feature will be disabled.

Option Name	`SCE_RUDP_OPTION_KEEP_ALIVE_TIMEOUT`
Description	Keep-alive timeout
Area Type	`uint32_t`
Area Size	`sizeof(uint32_t)`
Value	Time (in milliseconds) after starting the transmission of a keep-alive packet and no response is received (the connection is considered closed) (default: 0 – no timeout)
Notes	A keep-alive packet will continue to be resent until a timeout occurs or a response is received and another keep-alive is sent. The interval for the first resend is the larger of two values: the actual RTT measured by librudp or 1 second. Each interval for a subsequent resend will be twice as long as the previous interval.

See Also

`sceRudpSetOption()`

Context Options

Constants representing context options

Definition

Value	(Number)	Description
`SCE_RUDP_OPTION_MAX_PAYLOAD`	1	Maximum payload size of RUDP segment
`SCE_RUDP_OPTION_SNDBUF`	2	Send buffer size
`SCE_RUDP_OPTION_RCVBUF`	3	Receive buffer size
`SCE_RUDP_OPTION_NODELAY`	4	Message aggregation
`SCE_RUDP_OPTION_DELIVERY_CRITICAL`	5	Delivery Critical (DC) option
`SCE_RUDP_OPTION_ORDER_CRITICAL`	6	Order Critical (OC) option
`SCE_RUDP_OPTION_NONBLOCK`	7	Nonblocking mode
`SCE_RUDP_OPTION_STREAM`	8	Transport type (DGRAM/STREAM)
`SCE_RUDP_OPTION_CONNECTION_TIMEOUT`	9	Connection timeout
`SCE_RUDP_OPTION_CLOSE_WAIT_TIMEOUT`	10	Close-wait timeout
`SCE_RUDP_OPTION_AGGREGATION_TIMEOUT`	11	Aggregation timeout
`SCE_RUDP_OPTION_LAST_ERROR`	14	Last error
`SCE_RUDP_OPTION_READ_TIMEOUT`	15	Read timeout
`SCE_RUDP_OPTION_WRITE_TIMEOUT`	16	Write timeout
`SCE_RUDP_OPTION_FLUSH_TIMEOUT`	17	Flush timeout
`SCE_RUDP_OPTION_KEEP_ALIVE_INTERVAL`	18	Keep-alive interval
`SCE_RUDP_OPTION_KEEP_ALIVE_TIMEOUT`	19	Keep-alive timeout

Description

These constants represent the options used in `sceRudpSetOption()` and `sceRudpGetOption()`.

States and Statistics APIs

sceRudpGetContextStatus

Get status and statistics of context

Definition

```
#include <rudp.h>
int sceRudpGetContextStatus(
        int ctxId,
        SceRudpContextStatus *status,
        size_t statusSize
)
```

Calling Conditions

Multithread safe.

Arguments

ctxId	Context ID
status	Pointer to area to store information of the context
statusSize	Size of *status*

Return Values

Returns `SCE_RUDP_SUCCESS(0)` for normal termination.

Returns a negative value for errors. The main error codes are shown below. Note, however, that the application must not malfunction even if other error codes are returned.

Value	Description
`SCE_RUDP_ERROR_NOT_INITIALIZED`	Not initialized. `sceRudpInit()` has not been called, or `sceRudpEnd()` has already been called. Check the calling order.
`SCE_RUDP_ERROR_INVALID_ARGUMENT`	A negative value was set to *ctxId*, or NULL was specified to *status*, or *statusSize* was too large

Description

This function obtains the status and statistics of a context.

See Also

`SceRudpContextStatus`

SceRudpContextStatus

Context status and statistics

Definition

```
#include <rudp.h>
typedef struct SceRudpContextStatus {
        uint32_t state;
        int parentId;
        uint32_t children;
        uint32_t lostPackets;
        uint32_t sentPackets;
        uint32_t rcvdPackets;
        uint64_t sentBytes;
        uint64_t rcvdBytes;
        uint32_t retransmissions;
        uint32_t rtt;
} SceRudpContextStatus;
```

Members

state	Context status
parentId	Unused (the value is undefined and must not be accessed)
children	Unused (the value is undefined and must not be accessed)
lostPackets	Number of packets lost
sentPackets	Number of packets sent
rcvdPackets	Number of packets received
sentBytes	Number of bytes sent
rcvdBytes	Number of bytes received
retransmissions	Number of retransmissions
rtt	RTT (round-trip time) (microseconds)

One of the following values will be set to *state*.

Value	(Number)	Description
`SCE_RUDP_STATE_IDLE`	0	IDLE state
`SCE_RUDP_STATE_CLOSED`	1	CLOSED state
`SCE_RUDP_STATE_SYN_SENT`	2	SYN_SENT state
`SCE_RUDP_STATE_SYN_RCVD`	3	SYN_RCVD state
`SCE_RUDP_STATE_ESTABLISHED`	4	ESTABLISHED state
`SCE_RUDP_STATE_CLOSE_WAIT`	5	CLOSE_WAIT state

Description

This structure is used to store the information of a context.

See Also

`sceRudpGetContextStatus()`

sceRudpGetStatus

Get status and statistics of librudp

Definition

```
#include <rudp.h>
int sceRudpGetStatus(
        SceRudpStatus *status,
        size_t statusSize
)
```

Calling Conditions

Multithread safe.

Arguments

status	Pointer to area to store information of librudp
statusSize	Size of *status*

Return Values

Returns `SCE_RUDP_SUCCESS(0)` for normal termination.

Returns a negative value for errors. The main error codes are shown below. Note, however, that the application must not malfunction even if other error codes are returned.

Value	Description
`SCE_RUDP_ERROR_NOT_INITIALIZED`	Not initialized. `sceRudpInit()` has not been called, or `sceRudpEnd()` has already been called. Check the calling order.
`SCE_RUDP_ERROR_INVALID_ARGUMENT`	NULL was specified to *status*, or *statusSize* was too large

Description

This function obtains the status and statistics of librudp. The statistics are aggregate values from the point that librudp was initialized.

See Also

`SceRudpStatus`

SceRudpStatus

librudp status and statistics

Definition

```
#include <rudp.h>
typedef struct SceRudpStatus {
        uint64_t sentUdpBytes;
        uint64_t rcvdUdpBytes;
        uint32_t sentUdpPackets;
        uint32_t rcvdUdpPackets;
        uint64_t sentUserBytes;
        uint32_t sentUserPackets;
        uint64_t rcvdUserBytes;
        uint32_t rcvdUserPackets;
        uint32_t sentLatencyCriticalPackets;
        uint32_t rcvdLatencyCriticalPackets;
        uint32_t sentSynPackets;
        uint32_t rcvdSynPackets;
        uint32_t sentUsrPackets;
        uint32_t rcvdUsrPackets;
        uint32_t sentPrbPackets;
        uint32_t rcvdPrbPackets;
        uint32_t sentRstPackets;
        uint32_t rcvdRstPackets;
        uint32_t lostPackets;
        uint32_t retransmittedPackets;
        uint32_t reorderedPackets;
        uint32_t currentContexts;
        uint64_t sentQualityLevel1Bytes;
        uint64_t rcvdQualityLevel1Bytes;
        uint32_t sentQualityLevel1Packets;
        uint32_t rcvdQualityLevel1Packets;
        uint64_t sentQualityLevel2Bytes;
        uint64_t rcvdQualityLevel2Bytes;
        uint32_t sentQualityLevel2Packets;
        uint32_t rcvdQualityLevel2Packets;
        uint64_t sentQualityLevel3Bytes;
        uint64_t rcvdQualityLevel3Bytes;
        uint32_t sentQualityLevel3Packets;
        uint32_t rcvdQualityLevel3Packets;
        uint64_t sentQualityLevel4Bytes;
        uint64_t rcvdQualityLevel4Bytes;
        uint32_t sentQualityLevel4Packets;
        uint32_t rcvdQualityLevel4Packets;
        uint32_t allocs;
        uint32_t frees;
        uint32_t memCurrent;
        uint32_t memPeak;
        uint32_t establishedConnections;
        uint32_t failedConnections;
        uint32_t failedConnectionsReset;
        uint32_t failedConnectionsRefused;
        uint32_t failedConnectionsTimeout;
        uint32_t failedConnectionsVersionMismatch;
        uint32_t failedConnectionsTransportTypeMismatch;
        uint32_t failedConnectionsQualityLevelMismatch;
} SceRudpStatus;
```

Members

sentUdpBytes	Number of UDP bytes sent
rcvdUdpBytes	Number of UDP bytes received
sentUdpPackets	Number of UDP packets sent
rcvdUdpPackets	Number of UDP packets received
sentUserBytes	Number of bytes sent in user messages
sentUserPackets	Number of user messages sent
rcvdUserBytes	Number of bytes received in user messages
rcvdUserPackets	Number of user messages received
sentLatencyCriticalPackets	Number of Latency Critical packets sent
rcvdLatencyCriticalPackets	Number of Latency Critical packets received
sentSynPackets	Number of SYN packets sent
rcvdSynPackets	Number of SYN packets received
sentUsrPackets	Number of USR packets sent (USR packets are used in the actual transmission of user data.)
rcvdUsrPackets	Number of USR packets received
sentPrbPackets	Number of PRB packets sent (PRB packets are used for window update probes from the sender when the window size of the receive buffer becomes 0)
rcvdPrbPackets	Number of PRB packets received
sentRstPackets	Number of RST packets sent
rcvdRstPackets	Number of RST packets received
lostPackets	Number of packets lost
retransmittedPackets	Number of packets resent
reorderedPackets	Number of packets reordered (reverse order)
currentContexts	Current number of contexts
sentQualityLevel1Bytes	Number of bytes sent in Quality Level 1 (DC=1/OC=1) packets
rcvdQualityLevel1Bytes	Number of bytes received in Quality Level 1 (DC=1/OC=1) packets
sentQualityLevel1Packets	Number of Quality Level 1 (DC=1/OC=1) packets sent
rcvdQualityLevel1Packets	Number of Quality Level 1 (DC=1/OC=1) packets received
sentQualityLevel2Bytes	Number of bytes sent in Quality Level 2 (DC=1/OC=0) packets
rcvdQualityLevel2Bytes	Number of bytes received in Quality Level 2 (DC=1/OC=0) packets
sentQualityLevel2Packets	Number of Quality Level 2 (DC=1/OC=0) packets sent
rcvdQualityLevel2Packets	Number of Quality Level 2 (DC=1/OC=0) packets received
sentQualityLevel3Bytes	Number of bytes sent in Quality Level 3 (DC=0/OC=1) packets
rcvdQualityLevel3Bytes	Number of bytes received in Quality Level 3 (DC=0/OC=1) packets
sentQualityLevel3Packets	Number of Quality Level 3 (DC=0/OC=1) packets sent
rcvdQualityLevel3Packets	Number of Quality Level 3 (DC=0/OC=1) packets received
sentQualityLevel4Bytes	Number of bytes sent in Quality Level 4 (DC=0/OC=0) packets

rcvdQualityLevel4Bytes	Number of bytes received in Quality Level 4 (DC=0/OC=0) packets
sentQualityLevel4Packets	Number of Quality Level 4 (DC=0/OC=0) packets sent
rcvdQualityLevel4Packets	Number of Quality Level 4 (DC=0/OC=0) packets received
allocs	Number of malloc times
frees	Number of free times
memCurrent	Current size of allocated memory
memPeak	Maximum size of allocated memory
establishedConnections	Number of successful connections
failedConnections	Number of failed connections
failedConnectionsReset	Number of failed connections due to reset
failedConnectionsRefused	Number of failed connections due to refusal
failedConnectionsTimeout	Number of failed connections due to timeout
failedConnectionsVersionMismatch	Number of failed connections due to version mismatch
failedConnectionsTransportTypeMismatch	Number of failed connections due to transport type mismatch
failedConnectionsQualityLevelMismatch	Number of failed connections due to quality level mismatch

Description

This structure is used to store the status and statistics information of librudp.

See Also

`sceRudpGetStatus()`

sceRudpGetLocalInfo

Get local information of context

Definition

```
#include <rudp.h>
int sceRudpGetLocalInfo(
        int ctxId,
        int *soc,
        struct SceNetSockaddr *addr,
        SceNetSocklen_t *addrLen,
        uint16_t *vport,
        uint8_t *muxMode
)
```

Calling Conditions

Multithread safe.

Arguments

ctxId	Context ID
soc	Pointer to area to store the socket ID
addr	Pointer to area to store the local socket address
addrLen	Size of area indicated by *addr*
vport	Pointer to area to store the local virtual port number
muxMode	Pointer to area to store the multiplexing mode

In the current version, `SCE_RUDP_MUXMODE_P2P` is always set to the area indicated by *muxMode*.

Return Values

Returns `SCE_RUDP_SUCCESS(0)` for normal termination.

Returns a negative value for errors. The main error codes are shown below. Note, however, that the application must not malfunction even if other error codes are returned.

Value	Description
`SCE_RUDP_ERROR_NOT_INITIALIZED`	Not initialized. `sceRudpInit()` has not been called, or `sceRudpEnd()` has already been called. Check the calling order.
`SCE_RUDP_ERROR_INVALID_ARGUMENT`	A negative value was set to *ctxId*
`SCE_RUDP_ERROR_INVALID_CONTEXT_ID`	The context ID is invalid. Check the value specified to *ctxId*. Also make sure that the context itself has not already been deleted with `sceRudpTerminate()`.
`SCE_RUDP_ERROR_NOT_BOUND`	Context is not bound using `sceRudpBind()` and local information does not exist

Description

This function obtains the local information of a context.

To *soc*, *addr*, *addrLen*, *vport*, or *muxMode*, set NULL if the corresponding information is not necessary.

See Also

`sceRudpGetRemoteInfo()`,`sceRudpBind()`

sceRudpGetRemoteInfo

Get remote (peer) information of context

Definition

```
#include <rudp.h>
int sceRudpGetRemoteInfo(
        int ctxId,
        struct SceNetSockaddr *addr,
        SceNetSocklen_t *addrLen,
        uint16_t *vport
)
```

Calling Conditions

Multithread safe.

Arguments

ctxId	Context ID
addr	Pointer to area to store the peer's socket address
addrLen	Size of area indicated by *addr*
vport	Pointer to area to store the peer's virtual port number

Return Values

Returns `SCE_RUDP_SUCCESS(0)` for normal termination.

Returns a negative value for errors. The main error codes are shown below. Note, however, that the application must not malfunction even if other error codes are returned.

Value	Description
`SCE_RUDP_ERROR_NOT_INITIALIZED`	Not initialized. `sceRudpInit()` has not been called, or `sceRudpEnd()` has already been called. Check the calling order.
`SCE_RUDP_ERROR_INVALID_ARGUMENT`	A negative value was set to *ctxId*
`SCE_RUDP_ERROR_INVALID_CONTEXT_ID`	The context ID is invalid. Check the value specified to *ctxId*. Also make sure that the context itself has not already been deleted with `sceRudpTerminate()`.
`SCE_RUDP_ERROR_NOT_ACCEPTABLE`	Context does not have an established connection and remote (peer) information does not exist

Description

This function gets the information of the peer in a connection.

To *addr*, *addrLen*, or *vport*, set NULL if the corresponding information is not necessary.

See Also

`sceRudpGetLocalInfo()`

Connection APIs

sceRudpBind

Bind a context

Definition

```
#include <rudp.h>
int sceRudpBind(
        int ctxId,
        int soc,
        uint16_t vport,
        uint8_t muxMode
)
```

Calling Conditions

Multithread safe.

Arguments

ctxId	Context ID
soc	UDPP2P socket ID to use
vport	Local virtual port number to use
muxMode	Multiplexing mode to use (specify `SCE_RUDP_MUXMODE_P2P`)

Return Values

Returns `SCE_RUDP_SUCCESS(0)` for normal termination.

Returns a negative value for errors. The main error codes are shown below. Note, however, that the application must not malfunction even if other error codes are returned.

Value	Description
`SCE_RUDP_ERROR_NOT_INITIALIZED`	Not initialized. `sceRudpInit()` has not been called, or `sceRudpEnd()` has already been called. Check the calling order.
`SCE_RUDP_ERROR_INVALID_ARGUMENT`	A negative value was set to *ctxId* or *soc*
`SCE_RUDP_ERROR_MEMORY`	Could not allocate memory from the heap. This error does not occur during normal operation.
`SCE_RUDP_ERROR_INVALID_CONTEXT_ID`	The context ID is invalid. Check the value specified to *ctxId*. Also make sure that the context itself has not already been deleted with `sceRudpTerminate()`.
`SCE_RUDP_ERROR_ALREADY_BOUND`	Already bound
`SCE_RUDP_ERROR_INVALID_MUXMODE`	An invalid value was specified to *muxMode*, or the specified socket is already being used in a different multiplexing mode for another context. (In the current version, it is not possible for another context to use a different multiplexing mode, since there is only multiplexing mode supported.)
`SCE_RUDP_ERROR_INVALID_VPORT`	0 was specified to *vport*
`SCE_RUDP_ERROR_VPORT_EXHAUSTED`	No more virtual port numbers available

Description

This function associates a context with the socket ID, virtual port, and multiplexing mode that it will use.

To *vport*, specify the local virtual port number to use.

Notes

This function must always be called before `sceRudpInitiate()` or `sceRudpActivate()` is called. The socket ID is used as the identifier of local socket addresses; if an internal network I/O thread is used, librudp will use this socket for data input and output.

Also refer to note in the "Limitation of UDP Sockets and Multiplexing Mode" section in the Chapter 4 "Notes" of the "librudp Overview" document.

See Also

`sceRudpGetLocalInfo()`

sceRudpInitiate

Start connection to peer

Definition

```
#include <rudp.h>
int sceRudpInitiate(
        int ctxId,
        struct SceNetSockaddr const *to,
        SceNetSocklen_t toLen,
        uint16_t vport
)
```

Calling Conditions

Multithread safe.

Arguments

ctxId	Context ID
to	Pointer to area storing the socket address of the connection peer
toLen	Size of *to*
vport	Unused (specify 0)

Return Values

Returns `SCE_RUDP_SUCCESS(0)` for normal termination.

Returns a negative value for errors. The main error codes are shown below. Note, however, that the application must not malfunction even if other error codes are returned.

Value	Description
`SCE_RUDP_ERROR_NOT_INITIALIZED`	Not initialized. `sceRudpInit()` has not been called, or `sceRudpEnd()` has already been called. Check the calling order.
`SCE_RUDP_ERROR_INVALID_ARGUMENT`	A negative value was set to *ctxId*, or NULL was specified to *to*, or *toLen* was too large
`SCE_RUDP_ERROR_MEMORY`	Could not allocate memory from the heap. This error does not occur during normal operation.
`SCE_RUDP_ERROR_NO_EVENT_HANDLER`	A common event handler has not been registered
`SCE_RUDP_ERROR_INVALID_CONTEXT_ID`	The context ID is invalid. Check the value specified to *ctxId*. Also make sure that the context itself has not already been deleted with `sceRudpTerminate()`.
`SCE_RUDP_ERROR_NOT_ACCEPTABLE`	The context is not in IDLE state, or it is in LISTEN state
`SCE_RUDP_ERROR_CANCELLED`	Blocking state was cancelled during blocking mode by `sceRudpTerminate()` or another API
`SCE_RUDP_ERROR_CONN_TIMEOUT`	A timeout occurred before a peer connection could be established in blocking mode
`SCE_RUDP_ERROR_NOT_BOUND`	Not bound with `sceRudpBind()`

Value	Description
`SCE_RUDP_ERROR_ADDR_IN_USE`	Another context exists with the same socket and the same peer (socket address). Check that previously used context IDs have been deleted with `sceRudpTerminate()`.
`SCE_RUDP_ERROR_VPORT_IN_USE`	The virtual port is already being used
`SCE_RUDP_ERROR_IN_PROGRESS`	Already waiting for a connection
`SCE_RUDP_ERROR_ALREADY_ESTABLISHED`	Connection is already established

Description

This function starts a connection with the specified peer. This is also known as an active-open or a simultaneous-open procedure.

To *to*, specify a pointer to the area storing the socket address of the peer, and to *toLen*, specify the size of the area (as obtained with `sizeof(struct sockadddr_in)`, for example).

In the current version, *vport* is not used. Specify 0.

Notes

If blocking mode is specified in the context options, this function will be blocking until the processing completes or times out. Connection timeouts are specified by specifying `SCE_RUDP_OPTION_CONNECTION_TIMEOUT` in `sceRudpSetOption()`. Note that this timeout setting is also used by `sceRudpActivate()`. The default value is 60 seconds. If a timeout occurs, the blocking state will be cancelled, and the function will return `SCE_RUDP_ERROR_CONN_TIMEOUT`.

To check for the completion of processing by using the librudp polling feature, specify nonblocking mode in the context options, call this function, and specify `SCE_RUDP_POLL_EV_WRITE` in `sceRudpPollControl()`. To find out whether a connection was successfully established, check for errors by specifying `SCE_RUDP_OPTION_LAST_ERROR` in `sceRudpGetOption()`, or check if the context status obtained with `sceRudpGetContextStatus()` is `SCE_RUDP_STATE_ESTABLISHED`.

See Also

`sceRudpActivate()`

sceRudpActivate

Start waiting for connection from peer

Definition

```
#include <rudp.h>
int sceRudpActivate(
        int ctxId,
        struct SceNetSockaddr const *to,
        SceNetSocklen_t toLen
)
```

Calling Conditions

Multithread safe.

Arguments

ctxId	Context ID
to	Pointer to area storing the socket address of the connection peer
toLen	Size of *to*

Return Values

Returns `SCE_RUDP_SUCCESS(0)` for normal termination.

Returns a negative value for errors. The main error codes are shown below. Note, however, that the application must not malfunction even if other error codes are returned.

Value	Description
`SCE_RUDP_ERROR_NOT_INITIALIZED`	Not initialized. `sceRudpInit()` has not been called, or `sceRudpEnd()` has already been called. Check the calling order.
`SCE_RUDP_ERROR_INVALID_ARGUMENT`	A negative value was set to *ctxId*, or 0 was specified to *to*, or *toLen* was too large
`SCE_RUDP_ERROR_MEMORY`	Could not allocate memory from the heap. This error does not occur during normal operation.
`SCE_RUDP_ERROR_NO_EVENT_HANDLER`	A common event handler has not been registered
`SCE_RUDP_ERROR_INVALID_CONTEXT_ID`	The context ID is invalid. Check the value specified to *ctxId*. Also make sure that the context itself has not already been deleted with `sceRudpTerminate()`.
`SCE_RUDP_ERROR_NOT_ACCEPTABLE`	The context is not in IDLE state.
`SCE_RUDP_ERROR_CANCELLED`	Blocking state was cancelled during blocking mode by `sceRudpTerminate()` or another API
`SCE_RUDP_ERROR_CONN_TIMEOUT`	A timeout occurred before a peer connection could be established in blocking mode
`SCE_RUDP_ERROR_NOT_BOUND`	Not bound with `sceRudpBind()`
`SCE_RUDP_ERROR_ADDR_IN_USE`	Another context exists with the same socket and the same peer (socket address). Check that previously used context IDs have been deleted with `sceRudpTerminate()`.
`SCE_RUDP_ERROR_VPORT_IN_USE`	The virtual port is already being used
`SCE_RUDP_ERROR_IN_PROGRESS`	Already waiting for a connection

Value	Description
`SCE_RUDP_ERROR_ALREADY_ESTABLISHED`	Connection is already established

Description

This function starts waiting for a connection with the specified peer. This is also known as a passive-open procedure.

To *`to`*, specify a pointer to the area storing the socket address of the peer, and to *`toLen`*, specify the size of the area (as obtained with `sizeof(struct sockadddr_in)`, for example).

Notes

If blocking mode is specified in the context options, the maximum wait time is the connection timeout time. Connection timeouts are specified by specifying `SCE_RUDP_OPTION_CONNECTION_TIMEOUT` in `sceRudpSetOption()`. Note that this timeout setting is also used by `sceRudpInitiate()`. The default value is 60 seconds. If a timeout occurs, the blocking state will be cancelled, and the function will return `SCE_RUDP_ERROR_CONN_TIMEOUT`. In such cases, call `sceRudpTerminate()` to delete the context and carry out proper error handling since it is possible for the peer to have cancelled the connection, or a network error to have occurred.

To check for the completion of processing by using the librudp polling feature, specify nonblocking mode in the context options, call this function, and specify `SCE_RUDP_POLL_EV_WRITE` in `sceRudpPollControl()`. To find out whether a connection was successfully established, check for errors by specifying `SCE_RUDP_OPTION_LAST_ERROR` in `sceRudpGetOption()`, or check if the context status obtained with `sceRudpGetContextStatus()` is `SCE_RUDP_STATE_ESTABLISHED`.

See Also

`sceRudpInitiate()`

sceRudpTerminate

End connection and destroy context ID

Definition

```
#include <rudp.h>
int sceRudpTerminate(
        int ctxId
)
```

Calling Conditions

Multithread safe.

Arguments

ctxId	Context ID

Return Values

Returns `SCE_RUDP_SUCCESS(0)` for normal termination.

Returns a negative value for errors. The main error codes are shown below. Note, however, that the application must not malfunction even if other error codes are returned.

Value	Description
`SCE_RUDP_ERROR_NOT_INITIALIZED`	Not initialized. `sceRudpInit()` has not been called, or `sceRudpEnd()` has already been called. Check the calling order.
`SCE_RUDP_ERROR_INVALID_ARGUMENT`	A negative value was set to *ctxId*
`SCE_RUDP_ERROR_INVALID_CONTEXT_ID`	The context ID is invalid. Check the value specified to *ctxId*. Also make sure that the context itself has not already been deleted with `sceRudpTerminate()`.

Description

This function sends a termination request to the context currently in a connection and destroys the context ID.

All threads in blocking state will be unblocked immediately for the specified context ID, and blocking functions will return `SCE_RUDP_ERROR_CANCELLED`.

See Also

`sceRudpCreateContext()`

Data Transmission APIs

sceRudpRead

Read receive data

Definition

```
#include <rudp.h>
int sceRudpRead(
        int ctxId,
        void *data,
        size_t len,
        uint8_t flags,
        SceRudpReadInfo *info
)
```

Calling Conditions

Multithread safe.

Arguments

ctxId	Context ID
data	Area to store the data read
len	Data size to read
flags	Message flag
info	Area to store to the additional information of the data read

To *flags*, the following message flag can be specified.

Value	(Number)	Description
`SCE_RUDP_MSG_DONTWAIT`	0x01	Call as nonblocking

Return Values

Returns the actual size read (0 and over) for normal termination.

Returns a negative value for errors. The main error codes are shown below. Note, however, that the application must not malfunction even if other error codes are returned.

Value	Description
`SCE_RUDP_ERROR_NOT_INITIALIZED`	Not initialized. `sceRudpInit()` has not been called, or `sceRudpEnd()` has already been called. Check the calling order.
`SCE_RUDP_ERROR_INVALID_ARGUMENT`	A negative value was set to *`ctxId`*, or 0 was specified to *`data`*, or the *`size`* member of *`info`* was too large
`SCE_RUDP_ERROR_MEMORY`	Could not allocate memory from the heap. This error does not occur during normal operation.
`SCE_RUDP_ERROR_INVALID_CONTEXT_ID`	The context ID is invalid. Check the value specified to *`ctxId`*. Also make sure that the context itself has not already been deleted with `sceRudpTerminate()`.
`SCE_RUDP_ERROR_NOT_ACCEPTABLE`	The context is not in ESTABLISHED or CLOSE_WAIT state
`SCE_RUDP_ERROR_CANCELLED`	Blocking state was cancelled during blocking mode by `sceRudpTerminate()` or another API
`SCE_RUDP_ERROR_WOULDBLOCK`	No data exists for reading (in nonblocking mode), or a timeout occurred before receive data could be read (in blocking mode)
`SCE_RUDP_ERROR_BUFFER_TOO_SMALL`	*`len`* is too small
`SCE_RUDP_ERROR_END_OF_DATA`	Peer finished sending data (called `sceRudpTerminate()` already), and there is no more data to receive

Description

This function reads data from the library's internal receive buffer.

To *`data`*, specify a pointer to the area to store the data read, and to *`len`*, specify the data size to read.

To *`info`*, specify a pointer to a `SceRudpReadInfo` type variable. Specify NULL if additional information will not be used. Also, to `SceRudpReadInfo::`*`size`*, always specify the size of the structure.

If blocking mode is specified in the context options, this function will be blocking until the read completes or times out. The return value of the function will be a value of 0 and over if the processing completes, and `SCE_RUDP_ERROR_WOULDBLOCK` if a timeout occurs.

The timeout can be specified and modified in the context options. The default setting is 0 (no timeout).

Notes

If nonblocking mode is specified in the context options, and polling is not executed, the event `SCE_RUDP_CONTEXT_EVENT_READABLE` is notified to the context event handler when the receive data becomes readable.

For polling, specify the `SCE_RUDP_POLL_EV_READ` flag in `sceRudpPollControl()`, and wait for the data to become readable by polling the status with `sceRudpPollWait()`.

See Also

`sceRudpGetSizeReadable()`, `SCE_RUDP_CONTEXT_EVENT_READABLE`, `SCE_RUDP_POLL_EV_READ`

sceRudpWrite

Write send data

Definition

```
#include <rudp.h>
int sceRudpWrite(
        int ctxId,
        void const *data,
        size_t len,
        uint8_t flags
)
```

Calling Conditions

Multithread safe.

Arguments

ctxId	Context ID
data	Send data
len	Size of send data
flags	Message flag

To *flags*, the following message flags can be specified.

Value	(Number)	Description
SCE_RUDP_MSG_DONTWAIT	0x01	Call as nonblocking
SCE_RUDP_MSG_LATENCY_CRITICAL	0x08	Send as an urgent message
SCE_RUDP_MSG_ALIGN_32	0x10	Adjust the header size so the start of the payload is 4-byte-aligned
SCE_RUDP_MSG_ALIGN_64	0x20	Adjust the header size so the start of the payload is 8-byte-aligned
SCE_RUDP_MSG_WITH_TX_TIMESTAMP	0x40	Automatically add timestamp and send during network transmission (Refer to `SceRudpReadInfo`)

Return Values

Returns the actual size written (0 and over) for normal termination.

Returns a negative value for errors. The main error codes are shown below. Note, however, that the application must not malfunction even if other error codes are returned.

Value	Description
`SCE_RUDP_ERROR_NOT_INITIALIZED`	Not initialized. `sceRudpInit()` has not been called, or `sceRudpEnd()` has already been called. Check the calling order.
`SCE_RUDP_ERROR_INVALID_ARGUMENT`	A negative value was set to *`ctxId`*, or NULL was specified to *`data`*
`SCE_RUDP_ERROR_MEMORY`	Could not allocate memory from the heap. This error does not occur during normal operation.
`SCE_RUDP_ERROR_INVALID_CONTEXT_ID`	The context ID is invalid. Check the value specified to *`ctxId`*. Also make sure that the context itself has not already been deleted with `sceRudpTerminate()`.
`SCE_RUDP_ERROR_NOT_ACCEPTABLE`	The context is not in ESTABLISHED state
`SCE_RUDP_ERROR_CANCELLED`	Blocking state was cancelled during blocking mode by `sceRudpTerminate()` or another API
`SCE_RUDP_ERROR_WOULDBLOCK`	No space in the send buffer (in nonblocking mode), or a timeout occurred before send data could be written (in blocking mode)
`SCE_RUDP_ERROR_MSG_TOO_LARGE`	Size of send data is too large

Description

This function writes data to the library's internal send buffer.

To *`data`*, specify a pointer to the buffer where the send data is stored, and to *`len`*, specify the size of the send data (in bytes).

If blocking mode is specified in the context options, this function will be blocking until the write operation of the applicable data size completes or times out. The applicable data size depends on the transport type: for DGRAM, this is the size of the entire data (the byte size specified with *`len`*), and for STREAM, this is at least 1 byte. The return value of the function will be the number of bytes written if the processing completes, and `SCE_RUDP_ERROR_WOULDBLOCK` if a timeout occurs.

The timeout can be specified and modified in the context options. The default setting is 0 (no timeout).

Notes

If there is any part of the data that remains unwritten (in other words, if the return value is `SCE_RUDP_ERROR_WOULDBLOCK` or a positive value smaller than *`len`*), the remaining data must be written by calling this function again.

If nonblocking mode is specified in the context options, and space opens up in the send buffer, the event `SCE_RUDP_CONTEXT_EVENT_WRITABLE` will be notified to the context event handler. Use this event notification to start writing the rest of the data.

It is also possible to specify the `SCE_RUDP_POLL_EV_WRITE` flag in `sceRudpPollControl()`, and wait for the data to become writable by polling the status with `sceRudpPollWait()`.

See Also

`sceRudpGetSizeWritable()`, `SCE_RUDP_CONTEXT_EVENT_WRITABLE`, `SCE_RUDP_POLL_EV_WRITE`

SceRudpReadInfo

Additional information regarding the read data

Definition

```
#include <rudp.h>
typedef struct SceRudpReadInfo {
        uint8_t size;
        uint8_t retransmissionCount;
        uint16_t retransmissionDelay;
        uint8_t retransmissionDelay2;
        uint8_t flags;
        uint16_t sequenceNumber;
        uint32_t timestamp;
} SceRudpReadInfo;
```

Members

size	Size of the structure
retransmissionCount	Number of retransmissions
retransmissionDelay	Retransmission delay
retransmissionDelay2	Most significant byte of retransmission delay
flags	Supplementary information flag
sequenceNumber	Sequence number
timestamp	Transmission timestamp

Description

This structure stores information of the data read with `sceRudpRead()`. It indicates the number of times it was resent before it was received, the total time it took (in milliseconds) for the data to be sent (from the first transmission to the last), the sequence number given on the sending side, and timestamp information (in milliseconds).

The retransmission delay value is obtained as (*(retransmissionDelay2 << 16) + retransmissionDelay*).

The sequence number is a 16-bit wrapped counter given by the sending-side librudp in RUDP segments. It is used to monitor packet loss status in real time.

The transmission timestamp is a 32-bit wrapped timestamp (in milliseconds) given by the sending-side librudp when RUDP segments are sent to the network. It is used to monitor packet delay status in real time. It is only enabled when `SCE_RUDP_MSG_WITH_TX_TIMESTAMP` is set in *flags*.

The following two types are used as supplementary information flags:

Supplementary Information Flag	Description
`SCE_RUDP_MSG_LATENCY_CRITICAL`	LC flag set when sending.
`SCE_RUDP_MSG_WITH_TX_TIMESTAMP`	Transmission timestamp given when sending.

See Also

`sceRudpRead()`

sceRudpGetSizeReadable

Get size of readable data

Definition

```
#include <rudp.h>
SceSSize sceRudpGetSizeReadable(
        int ctxId
)
```

Calling Conditions

Multithread safe.

Arguments

ctxId	Context ID

Return Values

Returns the size that can be read (0 and over) for normal termination.

Returns a negative value for errors. The main error codes are shown below. Note, however, that the application must not malfunction even if other error codes are returned.

Value	Description
`SCE_RUDP_ERROR_NOT_INITIALIZED`	Not initialized. `sceRudpInit()` has not been called, or `sceRudpEnd()` has already been called. Check the calling order.
`SCE_RUDP_ERROR_INVALID_ARGUMENT`	A negative value was set to *ctxId*
`SCE_RUDP_ERROR_INVALID_CONTEXT_ID`	The context ID is invalid. Check the value specified to *ctxId*. Also make sure that the context itself has not already been deleted with `sceRudpTerminate()`.
`SCE_RUDP_ERROR_NOT_ACCEPTABLE`	The context is not in ESTABLISHED or CLOSE_WAIT state
`SCE_RUDP_ERROR_END_OF_DATA`	Peer finished sending data (called `sceRudpTerminate()` already), and there is no more data to receive

Description

This function obtains the size (in bytes) of the data that can be read from the library's internal receive buffer.

See Also

`sceRudpRead()`, `SCE_RUDP_CONTEXT_EVENT_READABLE`, `sceRudpGetNumberOfPacketsToRead()`

sceRudpGetNumberOfPacketsToRead

Get the number of packets with readable data

Definition

```
#include <rudp.h>
unsigned int sceRudpGetNumberOfPacketsToRead(
        int ctxId
)
```

Calling Conditions

Multithread safe.

Arguments

ctxId	Context ID

Return Values

Returns the number of datagrams that can be read (0 and over) for normal termination.

Returns a negative value for errors. The main error codes are shown below. Note, however, that the application must not malfunction even if other error codes are returned.

Value	Description
`SCE_RUDP_ERROR_NOT_INITIALIZED`	Not initialized. `sceRudpInit()` has not been called, or `sceRudpEnd()` has already been called. Check the calling order.
`SCE_RUDP_ERROR_INVALID_ARGUMENT`	A negative value was set to *ctxId*
`SCE_RUDP_ERROR_INVALID_CONTEXT_ID`	The context ID is invalid. Check the value specified to *ctxId*. Also make sure that the context itself has not already been deleted with `sceRudpTerminate()`.

Description

This function obtains the number of available datagrams that can be read from the library's internal receive buffer.

Notes

It is useful calling this function before calling `sceRudpGetSizeReadable()` in order to know how many datagrams are readable from the receive buffer. For each datagram, call `sceRudpGetSizeReadable()` in order to get its size.

See Also

`sceRudpRead(),SCE_RUDP_CONTEXT_EVENT_READABLE, sceRudpGetSizeReadable()`

sceRudpGetSizeWritable

Get size of writable data

Definition

```
#include <rudp.h>
SceSSize sceRudpGetSizeWritable(
        int ctxId
)
```

Calling Conditions

Multithread safe.

Arguments

ctxId	Context ID

Return Values

Returns the size that can be written (0 and over) for normal termination.

Returns a negative value for errors. The main error codes are shown below. Note, however, that the application must not malfunction even if other error codes are returned.

Value	Description
`SCE_RUDP_ERROR_NOT_INITIALIZED`	Not initialized. `sceRudpInit()` has not been called, or `sceRudpEnd()` has already been called. Check the calling order.
`SCE_RUDP_ERROR_INVALID_ARGUMENT`	A negative value was set to *ctxId*
`SCE_RUDP_ERROR_INVALID_CONTEXT_ID`	The context ID is invalid. Check the value specified to *ctxId*. Also make sure that the context itself has not already been deleted with `sceRudpTerminate()`.
`SCE_RUDP_ERROR_NOT_ACCEPTABLE`	The context is not in ESTABLISHED or CLOSE_WAIT state

Description

This function obtains the data size (in bytes) that can be written to the library's internal send buffer.

See Also

`sceRudpWrite()`, `SCE_RUDP_CONTEXT_EVENT_WRITABLE`

sceRudpFlush

Check if sent data arrived at peer's

Definition

```
#include <rudp.h>
int sceRudpFlush(
        int ctxId
)
```

Calling Conditions

Multithread safe.

Arguments

ctxId	Context ID

Return Values

Returns `SCE_RUDP_SUCCESS(0)` for normal termination.

Returns a negative value for errors. The main error codes are shown below. Note, however, that the application must not malfunction even if other error codes are returned.

Value	Description
`SCE_RUDP_ERROR_NOT_INITIALIZED`	Not initialized. `sceRudpInit()` has not been called, or `sceRudpEnd()` has already been called. Check the calling order.
`SCE_RUDP_ERROR_INVALID_ARGUMENT`	A negative value was set to *ctxId*
`SCE_RUDP_ERROR_INVALID_CONTEXT_ID`	The context ID is invalid. Check the value specified to *ctxId*. Also make sure that the context itself has not already been deleted with `sceRudpTerminate()`.
`SCE_RUDP_ERROR_NOT_ACCEPTABLE`	The context is not in ESTABLISHED or CLOSE_WAIT state
`SCE_RUDP_ERROR_IN_PROGRESS`	Still carrying out a previous flush operation (waiting for arrival notification). `sceRudpFlush()` may have been called again before a previous request completed.
`SCE_RUDP_ERROR_CANCELLED`	Blocking state was cancelled during blocking mode by `sceRudpTerminate()` or another API
`SCE_RUDP_ERROR_WOULDBLOCK`	A timeout occurred before arrival notification of sent data (in blocking mode)

Description

This function checks whether all the data written to the library's internal send buffer arrived at the peer's location.

After this function is called, the event `SCE_RUDP_CONTEXT_EVENT_FLUSHED` is notified to the context event handler when the arrival of the entire data has been confirmed. Note that ACK confirmation is not possible when DC=0, so in this case, arrival at the peer's is assumed upon transmission to the network.

See Also

`sceRudpWrite(),SCE_RUDP_CONTEXT_EVENT_FLUSHED,SCE_RUDP_POLL_EV_FLUSH`

Polling APIs

sceRudpPollCreate

Create a polling ID

Definition

```
#include <rudp.h>
int sceRudpPollCreate(
        size_t size
)
```

Calling Conditions

Multithread safe.

Arguments

size	Number of contexts to poll (initial value)

Return Values

Returns the polling ID (0 and over) for normal termination.

Returns a negative value for errors. The main error codes are shown below. Note, however, that the application must not malfunction even if other error codes are returned.

Value	Description
`SCE_RUDP_ERROR_NOT_INITIALIZED`	Not initialized. `sceRudpInit()` has not been called, or `sceRudpEnd()` has already been called. Check the calling order.
`SCE_RUDP_ERROR_INVALID_ARGUMENT`	0 was specified to *size*
`SCE_RUDP_ERROR_MEMORY`	Could not allocate memory from the heap. This error does not occur during normal operation.

Description

This function creates a polling ID for polling context events.

To *size*, set the maximum number of contexts expected to be polled. For example, specify 4 if it is known that there are 4 contexts to be monitored with this polling ID. This value does not dictate a limit and is used to specify the initial size of the memory to be allocated in the library. The internal memory size will automatically be increased when the number of target contexts increases. The actual limit on the number of contexts that can be monitored with one polling ID is 65536.

See Also

`sceRudpPollDestroy()`, `sceRudpPollControl()`, `sceRudpPollWait()`

sceRudpPollDestroy

Destroy a polling ID

Definition

```
#include <rudp.h>
int sceRudpPollDestroy(
        int pollId
)
```

Calling Conditions

Multithread safe.

Arguments

pollId	Polling ID to destroy

Return Values

Returns `SCE_RUDP_SUCCESS(0)` for normal termination.

Returns a negative value for errors. The main error codes are shown below. Note, however, that the application must not malfunction even if other error codes are returned.

Value	Description
`SCE_RUDP_ERROR_NOT_INITIALIZED`	Not initialized. `sceRudpInit()` has not been called, or `sceRudpEnd()` has already been called. Check the calling order.
`SCE_RUDP_ERROR_INVALID_ARGUMENT`	A negative value was set to *pollId*
`SCE_RUDP_ERROR_INVALID_POLL_ID`	An invalid polling ID was specified. Either the ID was already destroyed with `sceRudpPollDestroy()`, or it was not properly created.

Description

This function deletes a polling ID that is no longer needed, and frees its resources.

If the polling ID is destroyed while another thread is waiting with `sceRudpPollWait()`, the thread in blocking state will immediately be unblocked, and `sceRudpPollWait()` will return `SCE_RUDP_ERROR_CANCELLED`.

See Also

`sceRudpPollCreate()`, `sceRudpPollControl()`, `sceRudpPollWait()`

sceRudpPollControl

Modify polling ID settings

Definition

```
#include <rudp.h>
int sceRudpPollControl(
        int pollId,
        int op,
        int ctxId,
        uint16_t events
)
```

Calling Conditions

Multithread safe.

Arguments

pollId	Polling ID
op	Operation
ctxId	Context ID
events	Event flag

To *op*, specify one of the following values.

Value	(Number)	Description
SCE_RUDP_POLL_OP_ADD	1	Add target contexts
SCE_RUDP_POLL_OP_MODIFY	2	Modify target contexts
SCE_RUDP_POLL_OP_REMOVE	3	Delete target contexts

To *events*, specify a logical OR of the following event flags.

Value	(Number)	Description
SCE_RUDP_POLL_EV_READ	0x0001	READ event flag
SCE_RUDP_POLL_EV_WRITE	0x0002	WRITE event flag
SCE_RUDP_POLL_EV_FLUSH	0x0004	FLUSH event flag
SCE_RUDP_POLL_EV_ERROR	0x0008	ERROR event flag

Return Values

Returns `SCE_RUDP_SUCCESS(0)` for normal termination.

Returns a negative value for errors. The main error codes are shown below. Note, however, that the application must not malfunction even if other error codes are returned.

Value	Description
`SCE_RUDP_ERROR_NOT_INITIALIZED`	Not initialized. `sceRudpInit()` has not been called, or `sceRudpEnd()` has already been called. Check the calling order.
`SCE_RUDP_ERROR_INVALID_ARGUMENT`	A negative value was set to *`pollId`*, or an invalid value was specified to *`op`*, or a negative value was set to *`ctxId`*
`SCE_RUDP_ERROR_INVALID_CONTEXT_ID`	The context ID is invalid. Check the value specified to *`ctxId`*. Also make sure that the context itself has not already been deleted with `sceRudpTerminate()`.
`SCE_RUDP_ERROR_INVALID_POLL_ID`	An invalid polling ID was specified. Either the ID was already destroyed with `sceRudpPollDestroy()`, or it was not properly created.
`SCE_RUDP_ERROR_TOO_MANY_CONTEXTS`	Exceeded the maximum number of contexts that can be registered. This error does not occur during normal operation.

Description

This function adds, modifies, and deletes the contexts polled by the specified polling ID.

For additions and modifications, set the event flags to be monitored (all the target flags after addition/modification) to *`events`*. To delete the event flags, specify 0 (since the value of *`events`* will be ignored).

See Also

`sceRudpPollCreate(), sceRudpPollDestroy(), sceRudpPollWait()`

sceRudpPollWait

Wait for events by polling

Definition

```
#include <rudp.h>
int sceRudpPollWait(
        int pollId,
        SceRudpPollEvent *events,
        size_t eventLen,
        SceRudpUsec timeout
)
```

Calling Conditions

Multithread safe.

Arguments

pollId	Polling ID
events	Pointer to array to store any detected events
eventLen	Size of the array specified by *events*
timeout	Wait timeout (microseconds)

Return Values

Returns the number of events detected for normal termination. 0 indicates a timeout.

Returns a negative value for errors. The main error codes are shown below. Note, however, that the application must not malfunction even if other error codes are returned.

Value	Description
SCE_RUDP_ERROR_NOT_INITIALIZED	Not initialized. `sceRudpInit()` has not been called, or `sceRudpEnd()` has already been called. Check the calling order.
SCE_RUDP_ERROR_INVALID_ARGUMENT	A negative value was set to *pollId*, or 0 was specified to *events*, or a negative value was set to *eventLen*
SCE_RUDP_ERROR_INVALID_POLL_ID	An invalid polling ID was specified. Either the ID was already destroyed with `sceRudpPollDestroy()`, or it was not properly created.
SCE_RUDP_ERROR_CANCELLED	Wait was cancelled

Description

This function is used to wait for events of the registered context.

To *events*, specify a pointer to a `SceRudpPollEvent` type array, and to *eventLen*, specify the size of the array.

Examples

```
// Create a polling ID
int pollId;

pollId = sceRudpPollCreate(8);
if ( pollId < 0 ) {
        // Error handling
}

// Add a context
int ret;
ret = sceRudpPollControl(
                    pollId,
                    SCE_RUDP_POLL_OP_ADD,
                    ctxId,
                    SCE_RUDP_POLL_EV_WRITE | SCE_RUDP_POLL_EV_ERROR);
if ( ret < 0 ) {
        // Error handling
}

//  Wait for events
#define NUM_EVENTS    (4)
int numEvents;
SceRudpPollEvent events[NUM_EVENTS];
SceRudpUsec timeout = 1000000ull;

//  Event processing loop
while( good )
{
        numEvents = sceRudpPollWait(pollId, events, NUM_EVENTS, timeout);
        if ( numEvents < 0 ){
              // Error handling
              break;
        }

        if ( numEvents == 0 ){
              // Processing upon timeout
        } else {
              for (int i = 0; i < numEvents; ++i){
                    if (events[i].rtnEvents & SCE_RUDP_POLL_EV_READ){
                          // Call read event routine
                          onReadable(&events[i]);
                    }

                    // Stop polling write events
                    uint16_t newFlags =
                          events[i].reqEvents & ~SCE_RUDP_POLL_EV_WRITE;
                    ret = sceRudpPollControl(
                          pollId,
                          SCE_RUDP_POLL_OP_MODIFY,
                          events[i].ctxId,
                          newFlags);
                    if (ret < 0) {
                          // Error handling
                    }
              }
        }
}
```

Notes

The array specified in *`events`* can be any size; to *`eventLen`*, specify the maximum number of events to be processed at one time by the application. The number of detected events will never be larger than *`eventLen`*. The events that did not fit in the array will be passed to the application the next time `sceRudpPollWait()` is called. This function does not write more than one event with the same context ID to the array.

See Also

`sceRudpPollCreate()`, `sceRudpPollDestroy()`, `sceRudpPollControl()`, `sceRudpPollCancel()`

sceRudpPollCancel

Cancel waiting for events

Definition

```
#include <rudp.h>
int sceRudpPollCancel(
        int pollId
)
```

Calling Conditions

Multithread safe.

Arguments

pollId	Polling ID to cancel

Return Values

Returns `SCE_RUDP_SUCCESS(0)` for normal termination.

Returns a negative value for errors. The main error codes are shown below. Note, however, that the application must not malfunction even if other error codes are returned.

Value	Description
`SCE_RUDP_ERROR_NOT_INITIALIZED`	Not initialized. `sceRudpInit()` has not been called, or `sceRudpEnd()` has already been called. Check the calling order.
`SCE_RUDP_ERROR_INVALID_ARGUMENT`	A negative value was set to *pollId*
`SCE_RUDP_ERROR_INVALID_POLL_ID`	An invalid polling ID was specified. Either the ID was already destroyed with `sceRudpPollDestroy()`, or it was not properly created.

Description

This function is used to cancel the blocking wait started by `sceRudpPollWait()`.

If this function is called (on another thread) while waiting for an event with `sceRudpPollWait()`, the thread that called `sceRudpPollWait()` is released immediately from blocking state, and `sceRudpPollWait()` will return `SCE_RUDP_ERROR_CANCELLED`.

See Also

`sceRudpPollWait()`

SceRudpPollEvent

Information of polling event

Definition

```
#include <rudp.h>
typedef struct SceRudpPollEvent {
        int ctxId;
        uint16_t reqEvents;
        uint16_t rtnEvents;
} SceRudpPollEvent;
```

Members

ctxId	ID of context where the event occurred
reqEvents	Event flag being polled
rtnEvents	Event flag detected

For *reqEvents* and *rtnEvents*, the following flags are used.

Value	(Number)	Description
SCE_RUDP_POLL_EV_READ	0x0001	Detected a READ event
SCE_RUDP_POLL_EV_WRITE	0x0002	Detected a WRITE event
SCE_RUDP_POLL_EV_FLUSH	0x0004	Detected a FLUSH event
SCE_RUDP_POLL_EV_ERROR	0x0008	Detected an ERROR event

Description

This structure is used to store the events that were obtained during polling.

To *reqEvents*, the event flag being monitored is set. This member can be used to modify just a few of the target event flags while processing an event, by setting `SCE_RUDP_POLL_OP_MODIFY` in `sceRudpPollControl()`.

See Also

`sceRudpPollWait()`

Polling Event Flags

Constants representing polling events

Definition

Value	(Number)	Description
`SCE_RUDP_POLL_EV_READ`	0x0001	READ event flag
`SCE_RUDP_POLL_EV_WRITE`	0x0002	WRITE event flag
`SCE_RUDP_POLL_EV_FLUSH`	0x0004	FLUSH event flag
`SCE_RUDP_POLL_EV_ERROR`	0x0008	ERROR event flag

Description

These constants are used to represent events in `sceRudpPollControl()` and `sceRudpPollWait()`.

Event Handling and Network APIs

sceRudpNetReceived

Input received UDP data

Definition

```
#include <rudp.h>
int sceRudpNetReceived(
        int soc,
        uint8_t const *data,
        size_t dataLen,
        struct SceNetSockaddr const *from,
        SceNetSocklen_t fromLen
)
```

Calling Conditions

Multithread safe.

Arguments

soc	Socket ID where data was received
data	Receive data
dataLen	Size of receive data
from	Socket address of peer
fromLen	Size of *from*

Return Values

Returns `SCE_RUDP_SUCCESS(0)` for normal termination.

Returns a negative value for errors. The main error codes are shown below. Note, however, that the application must not malfunction even if other error codes are returned.

Value	Description
`SCE_RUDP_ERROR_NOT_INITIALIZED`	Not initialized. `sceRudpInit()` has not been called, or `sceRudpEnd()` has already been called. Check the calling order.
`SCE_RUDP_ERROR_INVALID_SOCKET`	A negative value was set to *soc*
`SCE_RUDP_ERROR_INVALID_ARGUMENT`	NULL was specified to *data*, or NULL was specified to *from*, or *fromLen* was too large
`SCE_RUDP_ERROR_MEMORY`	Could not allocate memory from the heap. This error does not occur during normal operation.
`SCE_RUDP_ERROR_THREAD_IN_USE`	Internal network I/O thread is already being used
`SCE_RUDP_ERROR_MSG_MALFORMED`	Input data is not an RUDP packet, or it is corrupted

Description

This function inputs the UDP data received by the application to librudp, when an internal network I/O thread is not used.

Notes

Communication processing without the internal network I/O thread involves receiving the event `SCE_RUDP_EVENT_SEND` notified to the common event handler. Refer to the description of `SceRudpEventHandler`.

See Also

`SceRudpEventHandler`

sceRudpProcessEvents

Process library events

Definition

```
#include <rudp.h>
int sceRudpProcessEvents(
        SceRudpUsec timeout
)
```

Calling Conditions

Multithread safe.

Arguments

timeout	Timeout time (in microseconds)

Return Values

Returns `SCE_RUDP_SUCCESS(0)` for normal termination.

Returns a negative value for errors. The main error codes are shown below. Note, however, that the application must not malfunction even if other error codes are returned.

Value	Description
`SCE_RUDP_ERROR_NOT_INITIALIZED`	Not initialized. `sceRudpInit()` has not been called, or `sceRudpEnd()` has already been called. Check the calling order.
`SCE_RUDP_ERROR_MEMORY`	Could not allocate memory from the heap. This error does not occur during normal operation.
`SCE_RUDP_ERROR_THREAD_IN_USE`	Internal network I/O thread is already being used.

Description

This function is used for processing the internal events of the library.

Call this function periodically if the internal network I/O thread will not be used. The recommended interval is 16milliseconds (60Hz).

It is possible to block out a certain interval to wait for events, by setting a value larger than 0ull to *timeout*. If 0ull is specified, the function will return immediately after all pending events are processed.

Notes

If the internal network I/O thread is not used, all callbacks from librudp will be called within this function.

Constants

Auxiliary Constants

Other auxiliary constants

Definition

Value	(Number)	Description
`SCE_RUDP_USEC_INDEFINITE`	0xffffffffffffffffllu	Infinite timeout time

Description

This is an auxiliary constant used in programming.

Return Codes

List of return codes returned by librudp

Definition

Value	(Number)	Description
SCE_RUDP_SUCCESS	0	Normal termination
SCE_RUDP_ERROR_NOT_INITIALIZED	0x80770001	Not initialized
SCE_RUDP_ERROR_ALREADY_INITIALIZED	0x80770002	Already initialized
SCE_RUDP_ERROR_INVALID_CONTEXT_ID	0x80770003	Invalid context ID
SCE_RUDP_ERROR_INVALID_ARGUMENT	0x80770004	Invalid argument
SCE_RUDP_ERROR_INVALID_OPTION	0x80770005	Invalid option name
SCE_RUDP_ERROR_INVALID_MUXMODE	0x80770006	Invalid multiplexing mode
SCE_RUDP_ERROR_MEMORY	0x80770007	Memory allocation failed
SCE_RUDP_ERROR_INTERNAL	0x80770008	Undefined internal error
SCE_RUDP_ERROR_CONN_RESET	0x80770009	Connection was reset
SCE_RUDP_ERROR_CONN_REFUSED	0x8077000a	Connection was refused
SCE_RUDP_ERROR_CONN_TIMEOUT	0x8077000b	Connection timed out
SCE_RUDP_ERROR_CONN_VERSION_MISMATCH	0x8077000c	Version does not match
SCE_RUDP_ERROR_CONN_TRANSPORT_TYPE_MISMATCH	0x8077000d	Transport type does not match
SCE_RUDP_ERROR_CONN_QUALITY_LEVEL_MISMATCH	0x8077000e	Quality level does not match
SCE_RUDP_ERROR_THREAD	0x8077000f	Internal I/O thread error
SCE_RUDP_ERROR_THREAD_IN_USE	0x80770010	Internal I/O thread is currently in use
SCE_RUDP_ERROR_NOT_ACCEPTABLE	0x80770011	Operation is not permitted
SCE_RUDP_ERROR_MSG_TOO_LARGE	0x80770012	Message is too large
SCE_RUDP_ERROR_NOT_BOUND	0x80770013	Not bound
SCE_RUDP_ERROR_CANCELLED	0x80770014	Blocking was cancelled
SCE_RUDP_ERROR_INVALID_VPORT	0x80770015	Invalid virtual port
SCE_RUDP_ERROR_WOULDBLOCK	0x80770016	Currently executing operation
SCE_RUDP_ERROR_VPORT_IN_USE	0x80770017	Virtual port is currently in use
SCE_RUDP_ERROR_VPORT_EXHAUSTED	0x80770018	No more available virtual port numbers
SCE_RUDP_ERROR_INVALID_SOCKET	0x80770019	Invalid socket
SCE_RUDP_ERROR_BUFFER_TOO_SMALL	0x8077001a	Buffer is too small
SCE_RUDP_ERROR_MSG_MALFORMED	0x8077001b	Invalid packet
SCE_RUDP_ERROR_ADDR_IN_USE	0x8077001c	Address is currently in use
SCE_RUDP_ERROR_ALREADY_BOUND	0x8077001d	Already bound
SCE_RUDP_ERROR_ALREADY_EXISTS	0x8077001e	Already exists
SCE_RUDP_ERROR_INVALID_POLL_ID	0x8077001f	Invalid polling ID
SCE_RUDP_ERROR_TOO_MANY_CONTEXTS	0x80770020	Too many contexts
SCE_RUDP_ERROR_IN_PROGRESS	0x80770021	Currently executing operation
SCE_RUDP_ERROR_NO_EVENT_HANDLER	0x80770022	Common event handler has not been registered
SCE_RUDP_ERROR_PAYLOAD_TOO_LARGE	0x80770023	Payload is too large
SCE_RUDP_ERROR_END_OF_DATA	0x80770024	End of receive data
SCE_RUDP_ERROR_ALREADY_ESTABLISHED	0x80770025	Connection is already established
SCE_RUDP_ERROR_KEEP_ALIVE_FAILURE	0x80770026	Connection was closed due to keep-alive timeout