PlayStation®2 EE Library Reference
Release 2.4.3

Device Libraries

Summary Table of Contents

About This Manual

This is the Runtime Library Release 2.4.3 version of the *PlayStation®2 EE Library Reference - Device Libraries* manual.

The purpose of this manual is to define all available PlayStation®2 EE device library structures and functions. The companion *PlayStation®2 EE Library Overview - Device Libraries* describes the structure and purpose of the libraries.

Changes Since Last Release

Chapter 1: Device Control Library

- "Calling Conditions" descriptions have been added to each function.

Chapter 3: Hard Disk Library (for EE)

- In the "Return Value" section of the devctl command HDIOC_STATUS, the description of return value 2 has been changed.

Chapter 5: Memory Card Library

- A description of the sceMcStDateTime structure has been added.
- A description of the sceMcEnd() function has been added.
- In the sceMcTblGetDir() structures, the _Create and _Modify members have been added due to the addition of the sceStDateTime structure.
- In the "Syntax" section of the sceMcWrite() function, an error in the type of the second argument has been corrected.

Chapter 9: Controller Library 2

- "Calling Conditions" descriptions have been added to each function.

Chapter 11: Vibration Library

- "Calling Conditions" descriptions have been added to each function.

Related Documentation

Library specifications for the IOP can be found in the *PlayStation®2 IOP Library Reference* manuals and the *PlayStation®2 IOP Library Overview* manuals.

Note: the Developer Support Web site posts current developments regarding the Libraries and also provides notice of future documentation releases and upgrades.

Typographic Conventions

Certain Typographic Conventions are used throughout this manual to clarify the meaning of the text:

Convention	Meaning
`courier`	Indicates literal program code.
italic	Indicates names of arguments and structure members (in structure/function definitions only).
medium bold	Indicates data types and structure/function names (in structure/function definitions only).
blue	Indicates a hyperlink.

Developer Support

Sony Computer Entertainment America (SCEA)

SCEA developer support is available to licensees in North America only. You may obtain developer support or additional copies of this documentation by contacting the following addresses:

Order Information	Developer Support
In North America:	*In North America:*
Attn: Developer Tools Coordinator Sony Computer Entertainment America 919 East Hillsdale Blvd. Foster City, CA 94404, U.S.A. Tel: (650) 655-8000	E-mail: PS2_Support@playstation.sony.com Web: http://www.devnet.scea.com/ Developer Support Hotline: (650) 655-5566 (Call Monday through Friday, 8 a.m. to 5 p.m., PST/PDT)

Sony Computer Entertainment Europe (SCEE)

SCEE developer support is available to licensees in Europe only. You may obtain developer support or additional copies of this documentation by contacting the following addresses:

Order Information	Developer Support
In Europe:	*In Europe:*
Attn: Production Coordinator Sony Computer Entertainment Europe 30 Golden Square London W1F 9LD, U.K. Tel: +44 (0) 20 7859-5000	E-mail: ps2_support@scee.net Web: https://www.ps2-pro.com/ Developer Support Hotline: +44 (0) 20 7859-5777 (Call Monday through Friday, 9 a.m. to 6 p.m., GMT)

Chapter 1: Device Control Library
Table of Contents

Functions

sceDbcEnd

Terminate device control library

Library	*Introduced*	*Documentation last modified*
libdbc	2.4	January 4, 2002

Syntax

int sceDbcEnd(void)

Calling conditions

Can be called from a thread

Not multithread safe (must be called in an interrupt enabled state)

Description

This function terminates the device control library.

Return value

1:	Processing succeeded
Other than 1:	Processing failed

sceDbcInit

Initialize device control library

Library	*Introduced*	*Documentation last modified*
libdbc	2.4	January 4, 2002

Syntax

int sceDbcInit(void)

Calling conditions

Can be called from a thread

Not multithread safe (must be called in an interrupt enabled state)

Description

This function initializes the device control library.

Before initializing the library, dbcman.irx must be loaded in the IOP.

Return value

1:	Initialization succeeded
Other than 1:	Initialization failed

Chapter 2: CD(DVD)-ROM Library (for EE)
Table of Contents

Structures

sceCdCLOCK

Structure which stores the date and time

Library	*Introduced*	*Documentation last modified*
libcdvd	1.5	January 22, 2001

Structure

typedef struct {

u_char *stat;*	0: normal. Any other: error (e.g. internal battery is dead)
u_char *second;*	Second (BCD value)
u_char *minute;*	Minute (BCD value)
u_char *hour;*	Hour (BCD value)
u_char *pad;*	Padding data produced by alignment
u_char *day;*	Day (BCD value)
u_char *month;*	Month (BCD value)
u_char *year;*	Year (BCD value)

} **sceCdCLOCK;**

Description

Stores the date and time with a BCD value.

See also

sceCdReadClock()

sceCdlFILE

File descriptor (for both CD/DVD)

Library	*Introduced*	*Documentation last modified*
libcdvd	1.1	July 2, 2001

Structure

typedef struct {	
u_int *lsn;*	Logical sector number of file
u_int *size;*	File size (in bytes)
char *name[16];*	Filename
u_char *date[8];*	1st: Seconds 2nd: Minutes 3rd: Hours 4th: Date 5th: Month 6th 7th: Year (4 digits)
u_int *flag;*	Bits 0-7 are the ISO9660 file flag; other bits are reserved
} **sceCdlFILE;**	

Description

Structure representing CD(DVD)-ROM file position and size.

See also

sceCdSearchFile()

sceCdlLOCCD

CD-ROM read location

Library	*Introduced*	*Documentation last modified*
libcdvd	1.1	December 23, 1999

Structure

typedef struct {	
u_char *minute;*	Minutes
u_char *second;*	Seconds
u_char *sector;*	Sector
u_char *track;*	Track number
} **sceCdlLOCCD;**	

Description

Structure representing read position (head position) on the CD-ROM.

Notes

Provided solely to calculate the CD read location using minutes/seconds/sectors.

See also

sceCdIntToPos(), sceCdPosToInt()

sceCdRMode

CD(DVD)-ROM read mode

Library	*Introduced*	*Documentation last modified*
libcdvd	1.1	October 11, 2001

Structure

typedef struct {	
u_char *trycount;*	Read try count (No. of error retries + 1) (0: 256 tries)
u_char *spindlctrl;*	SCECdSpinStm: Recommended stream rotation speed.
	SCECdSpinNom: Starts reading data at maximum rotational velocity and if a read error occurs, the rotational velocity is reduced.
u_char *datapattern;*	SCECdSecS2048: Data size 2048 bytes
	SCECdSecS2328: 2328 bytes
	SCECdSecS2340: 2340 bytes
u_char *pad;*	Padding data produced by alignment
} **sceCdRMode;**	

Description

This structure is used to specify the CD(DVD)-ROM read mode. datapattern for DVD media reads is effective only with SCECdSecS2048.

See also

sceCdRead()

sceCdStmInit

File I/O functions: Stream initialization structure

Library	*Introduced*	*Documentation last modified*
libcdvd	2.4	October 11, 2001

Structure

typedef struct {	
u_int *bufmax***;**	Capacity of stream buffer, in its entirety (in number of 2048-byte sectors)
u_int *bankmax***;**	Number of subdivisions of the stream buffer (i.e. number of ring buffers) For a buffer that has been subdivided into 3 more parts, the desired buffer size is approximately 16 sectors.
u_int *iop_bufaddr***;**	IOP memory address of stream buffer
} sceCdStmInit;	

Description

This structure is used to specify initial values of the stream for the sceDevctl() file I/O functions.

See also

CDIOC_STREAMINIT

Functions

sceCdBreak

Break command

Library	*Introduced*	*Documentation last modified*
libcdvd	1.4	October 11, 2001

Syntax

int sceCdBreak (void)

Calling conditions

Can be called from a thread

Multithread safe (must be called in interrupt-enabled state)

Description

Breaks the executing command (e.g., sceCdPause(), sceCdRead(), sceCdSeek(), sceCdStandby(), sceCdSstatus(), sceCdstop()).

The sceCdSync() function is used to confirm that break processing has ended.

Breaks the processing of each command and calls the callback function, if one is set.

SCECdErABRT will be set for drive error information.

Return value

0 if command issue failed.

1 if command issue succeeded.

sceCdCallback

Define sceCdSync callback function

Library	*Introduced*	*Documentation last modified*
libcdvd	1.3	August 31, 2001

Syntax

int sceCdCallback (	
void *(*func)***(int))**	Address of callback function

Calling conditions

Can be called from a thread

Multithread safe (must be called in interrupt-enabled state)

Description

Sets the callback *func* called when a non-blocking function terminates.

When the callback is set, the function *func* is called when the non-blocking function terminates.

The function *func* is called as a callback thread.

If *func* is set to 0 or the command fails to issue, the callback does not occur.

Moreover, a callback cannot be set when a function that has already caused a callback is executing.

The function code of the cause of the callback is passed to the callback function in the first argument, as shown below.

SCECdFuncRead	sceCdRead() function has terminated.
SCECdFuncSeek	sceCdSeek () function has terminated.
SCECdFuncStandby	sceCdStandby() function has terminated.
SCECdFuncStop	sceCdStop() function has terminated.
SCECdFuncPause	sceCdPause() function has terminated.

Note about callback functions

- Calling a function that generates a callback, such as sceCdRead(), while a callback is executing, is not supported.

Return value

Returns the address of the previously set callback function, or 0 if no callback was set.

sceCdChangeThreadPriority

Change the IOP thread priority of an EE-side request processing module

Library	*Introduced*	*Documentation last modified*
libcdvd	2.0	July 2, 2001

Syntax

int sceCdChangeThreadPriority(	
int *priority***)**	Value of IOP thread priority for EE-side request processing module

Calling conditions

Can be called from a thread

Multithread safe (must be called in interrupt-enabled state)

Description

This function sets the IOP thread priority of an EE-side request processing module.

The default value for IOP thread priority of an EE-side request processing module is 81.

When changing the IOP thread priority, careful consideration must be given to the priorities of other modules. Therefore, the IOP thread priority value should not be changed carelessly.

Return value

If command issue failed, the KernelErrorCode from the IOP is returned.
0 is returned if the command was successfully issued.

sceCdDiskReady

Check drive status

Library	*Introduced*	*Documentation last modified*
libcdvd	1.1	July 2, 2001

Syntax

int sceCdDiskReady (

int *mode)*	Check mode (0: blocking, 1: non-blocking) When mode is set to non-blocking, the operating conditions of other threads must be thoroughly considered.

Calling conditions

Can be called from a thread

Multithread safe (must be called in interrupt-enabled state)

Description

Checks the drive status and determines if a command can be issued. If there is no media in the drive, SCECdNotReady is returned.

If the mode argument is set to blocking, and the drive rotation is not stable, the function waits until the drive rotation is stable, then it returns. In the non-blocking mode, the function returns immediately after the status is checked.

When this function is used for polling in non-blocking mode in a multithreaded environment, a function such as DelayThread() must be used so that there is sufficient room for other threads to operate.

Return value

SCECdComplete	Drive state allows commands to be issued
SCECdNotReady	Drive cannot accept commands

sceCdGetDiskType

Get media format

Library	*Introduced*	*Documentation last modified*
libcdvd	1.1	August 31, 2001

Syntax

int sceCdGetDiskType (void)

Calling conditions

Can be called from a thread

Multithread safe (must be called in interrupt-enabled state)

Description

Gets the media format

Return value

SCECdIllgalMedia	Disc cannot be played
SCECdPS2DVD	Disc is a PlayStation 2 DVD
SCECdPS2CD	Disc is a PlayStation 2 CD
SCECdPS2CDDA	Disc is a PlayStation 2 CD (with CDDA)
SCECdPSCD	Disc is a PlayStation CD
SCECdPSCDDA	Disc is a PlayStation CD (with CDDA)
SCECdDVDV	Disc is DVD Video
SCECdCDDA	Disc is a music CD
SCECdDETCT	Analyzing disc
SCECdNODISC	No disc mounted
SCECdUNKNOWN	Undistinguishable disk

sceCdGetError

Get drive error information

Library	*Introduced*	*Documentation last modified*
libcdvd	1.1	October 11, 2001

Syntax

int sceCdGetError (void)

Calling conditions

Can be called from a thread

Multithread safe (must be called in interrupt-enabled state)

Description

Gets drive error information.

Return value

Table 2-1

Return value	Meaning
SCECdErFAIL	sceCdGetError() function issue failed
SCECdErNO	No error
SCECdErEOM	Outermost track reached during playback
SCECdErTRMOPN	Cover opened during playback
SCECdErREAD	Problem occurred during read
SCECdErCUD	Not appropriate for disc in drive
SCECdErNORDY	Processing command
SCECdErABRT	Abort command received
SCECdErREADCF	Read command issue failed

sceCdGetReadPos

Check the progress of sceCdRead()

Library	*Introduced*	*Documentation last modified*
libcdvd	1.3	July 2, 2001

Syntax

u_int sceCdGetReadPos (void)

Calling conditions

Can be called from an interrupt handler

Can be called from a thread

Multithread safe

Description

A value in 16-sector units (a multiple of 32768) is returned. Because of alignment adjustment, if the read buffer is other than 64-byte aligned, care should be taken when performing the last transfer of the buffer area which will not be 64-byte aligned (on either side of the read buffer).

Return value

Returns the progress of the sceCdRead() function as the size of the data transferred to the buffer.

When sceCdRead() terminates, 0 is returned.

sceCdGetToc

Read TOC

Library	*Introduced*	*Documentation last modified*
libcdvd	1.1	July 2, 2001

Syntax

int sceCdGetToc (

u_char **toc)*	Address returned by location table information (a 1024 byte area is required).

Calling conditions

Can be called from a thread

Multithread safe (must be called in an interrupt-enabled state)

Description

Gets TOC sector information from CD-ROM.

Return value

1 is returned if the command was successfully issued, else 0 is returned.

sceCdInit

Initialize the CD(DVD)-ROM subsystem

Library	*Introduced*	*Documentation last modified*
libcdvd	1.1	July 2, 2001

Syntax

int sceCdInit (

int *init_mode)*	Library initialization mode SCECdINIT: Initialize library and block until commands can be issued. SCECdINoD: Initialize library only SCECdEXIT: Close library

Calling conditions

Can be called from a thread

Not multithread safe (must be called in an interrupt-enabled state)

Description

Initializes the CD(DVD)-ROM subsystem.

Notes

sceCdInit must be used for initialization first even if stdio (e.g., sceRead()) will be used.

After performing initialization with sceCdInit(), be sure to call sceCdMmode() to specify the type of media (CD or DVD).

If this function is used when cdvdman.irx and cdvdfsv.irx have not been replaced within the IOP default module, 2 is returned.

Return value

0: Initialization failed.

1: Initialization was performed normally.

2: Although initialization was performed, the default module was detected on the IOP side.

See also

sceCdMmode()

sceCdInitEeCB

Initialize callback thread

Library	*Introduced*	*Documentation last modified*
libcdvd	1.4	July 2, 2001

Syntax

int sceCdInitEeCB (

int *cb_prio,*	Priority of callback thread The priority of the callback thread must always be set to a value higher than the priority of the calling thread.
void **stack_addr;*	Stack address of callback thread
int *stack_size)*	Stack size of callback thread

Calling conditions

Can be called from a thread

Not multithread safe

Description

Initializes a callback thread. When the callback is used, it is always executed immediately after the sceCdInit() function, etc.

Notes

The stack address must be specified as a multiple of 16, with 16-byte alignment.

Return value

0: The callback was initialized, and only the priority was changed.

1: Initialized callback.

sceCdIntToPos

Get CD-ROM's minutes/seconds/sectors from logical sector

Library	*Introduced*	*Documentation last modified*
libcdvd	1.1	July 2, 2001

Syntax

sceCdlLOCCD *sceCdIntToPos (

int *i,*	Logical sector number
sceCdlLOCCD **p)*	Minutes/seconds/sectors

Calling conditions

Can be called from an interrupt handler

Can be called from a thread

Multithread safe

Description

Calculates minutes/seconds/sectors from logical sector number.

Not meaningful when the media is DVD.

Return value

Returns the address of CdlLOCCD.

sceCdMmode

Specify the media for reading

Library	*Introduced*	*Documentation last modified*
libcdvd	2.0	July 2, 2001

Syntax

int sceCdMmode(

int *media***)**	Read media	
	SCECdCD	Specify CD as the read media.
	SCECdDVD	Specify DVD as the read media.

Calling conditions

Can be called from a thread

Multithread safe (must be called in an interrupt-enabled state)

Description

This function is used to specify the read media for the CD(DVD)-ROM subsystem.

Notes

This function must be used to specify the read media after the sceCdInit() function is called.

Return value

0 is returned if command issue failed. 1 is returned if the command was successfully issued.

See also

sceCdInit()

sceCdPause

Pause CD(DVD)-ROM head

Library	*Introduced*	*Documentation last modified*
libcdvd	1.2	July 2, 2001

Syntax

int sceCdPause (void)

Calling conditions

Can be called from a thread

Multithread safe (must be called in an interrupt-enabled state)

Description

The head is put in a pause state at its current position on the CD(DVD)-ROM.

Notes

Since the function is a non-blocking function, the actual pausing of the head must be detected with sceCdSync().

Return value

1 is returned if the command was successfully issued, else 0 is returned.

See also

sceCdSync()

sceCdPOffCallback

Define PlayStation 2 power-off callback function

Library	*Introduced*	*Documentation last modified*
libcdvd	2.2.2	August 31, 2001

Syntax

int sceCdPOffCallback (	
void (**func***)(void *)**	Address of the callback function
void ****addr*)**	Address of the callback argument

Calling conditions

Can be called from a thread

Multithread safe (must be called in interrupt-enabled state)

Description

For compatibility with the hard disk drive (EXPANSION BAY type), in order to use the hard disk this function must be used to perform hard disk power-off processing. It is only for an EXPANSION BAY type hard disk drive.

The function sets the callback *func* that is to be called when the power-off operation is performed.

When a callback is set, the function *func* is called when the power-off operation is performed.

The function *func* is called by the interrupt handler.

If 0 is specified for *func*, no callback will occur.

When cdvdfsv.irx (cdvd_ee_driver) has been unloaded, use the standard I/O function scePowerOffHandler().

Return value

Address of the callback function set previously. 0 is returned if the callback has not been set.

sceCdPosToInt

Get CD-ROM's logical sector number from minutes/seconds/sectors

Library	*Introduced*	*Documentation last modified*
libcdvd	1.1	July 2, 2001

Syntax

int sceCdPosToInt (	
sceCdlLOCCD **p)*	Minutes/seconds/sectors

Calling conditions

Can be called from an interrupt handler

Can be called from a thread

Multithread safe

Description

Calculates logical sector number from minutes/seconds/sectors value.

Not meaningful when the media is DVD.

Return value

Logical sector number

sceCdPowerOff

PlayStation 2 power OFF

Library	*Introduced*	*Documentation last modified*
libcdvd	2.2.2	August 31, 2001

Syntax

int sceCdPowerOff (

int **stat*)	Status

Calling conditions

Can be called from a thread

Multithread safe (must be called in interrupt-enabled state)

Description

This function issues a PlayStation 2 PowerOff request.

This function must be used in power-off processing when a hard disk drive or HDD Ethernet (smap.irx) is used.

For details about power-off processing, refer to the CD(DVD)-ROM library, PlayStation File System (pfs) and network (inet) overviews.

Notes

When calling this function, make sure it is executed after an interrupt is detected by sceCdPOffCallback() and the hard disk drive is powered off.

<Sample power-off processing function calling sequence when a hard disk drive is used>

```
printf("power off request has come.\n");
/* close all files */
sceDevctl("pfs:", PDIOC_CLOSEALL, NULL, 0, NULL, 0);
/* dev9 power off, need to power off PS2 */
while(sceDevctl("hdd:", HDIOC_DEV9OFF, NULL, 0, NULL, 0) < 0);
/* PS2 power off */
while(!sceCdPowerOff(&stat) || stat);
while(1);
```

Notes:

- With a hard disk drive (EXPANSION BAY type), if the RESET button on the system unit is pressed between the time hard disk power-off processing is performed and PlayStation 2 system unit power-off processing is performed, the PlayStation 2 system unit will be reset.
- When cdvdfsv.irx (cdvd_ee_driver) has been unloaded, use the devctl command CDIOC_POWEROFF.

Return value

0 if command issued failed

stat return value	bit7: 1 Command error

sceCdRead

Read data

Library	*Introduced*	*Documentation last modified*
libcdvd	1.1	July 2, 2001

Syntax

int sceCdRead (

u_int *lsn,*	Logical sector number at which to begin reading
u_int *sectors,*	Number of sectors to read
void **buf,*	Read buffer
sceCdRMode **mode***)**	Read mode

Calling conditions

Can be called from a thread

Multithread safe (must be called in an interrupt-enabled state)

Description

A seek is performed to the starting read position indicated by lsn.

The number of sectors of data specified by the sectors argument is read from StartPoint on the CD(DVD)-ROM and placed in the memory specified by buf. The head is then put in the pause state.

Notes

CD-DA and DVD-video data cannot be read.

Since this is a non-blocking function, the actual completion of the data transfer must be detected using sceCdSync().

Note on using this function

- When data is transferred to the EE, sometimes the library will adjust the alignment of the buffer address. Therefore, in the interest of speed, it is best to use 64-byte alignment as much as possible.

Return value

1 is returned if the command was successfully issued, otherwise 0 is returned.

See also

sceCdSync()

sceCdReadChain

Batch read data

Library	*Introduced*	*Documentation last modified*
libcdvd	1.6	July 2, 2001

Syntax

int sceCdReadChain (

u_int **tag,*	Pointer to the read parameter storage data sequence. The data sequence structure is as follows. tag= { lsn, sectors, buf, lsn, sectors, buf, : : : Max. 64 sequences 0xffffffff, 0xffffffff, 0xffffffff }; 0xffffffff is placed at the end of the data sequence. u_int lsn: Logical sector number where reading starts u_int sectors: Number of sectors to read u_int buf: The following kind of value indicating the read buffer position bit31..bit2: High-order 30 bits of the read buffer address bit0: 0: EE memory 1: IOP memory
sceCdRMode **mode)*	Read mode

Calling conditions

Can be called from a thread

Multithread safe (must be called in an interrupt-enabled state)

Description

Reads at most 64 sets of data in a batch according to the contents of the read parameter storage data sequence. CD-DA data and DVD-video data cannot be read.

When EE memory is specified as the read buffer, the addresses must adhere to 64-byte alignment.

Since this is a non-blocking function, the sceCdSync() function must be used to detect the end of the actual transfer.

Return Value

If command issue failed, 0 is returned. If it succeeded, 1 is returned.

See Also

sceCdSync()

sceCdReadClock

Get date and time

Library	*Introduced*	*Documentation last modified*
libcdvd	1.5	July 2, 2001

Syntax

int sceCdReadClock (

sceCdCLOCK **rtc)*	Address of structure where date and time are stored

Calling conditions

Can be called from a thread

Multithread safe (must be called in an interrupt-enabled state)

Description

Gets the date and time from the PlayStation 2's built-in real-time clock.

Notes

For this function to use a controller which performs drive-related processing, an interval of 300(msec) must be cleared when calling it continuously.

Also, the following values are returned in the stat member of the rtc time storage structure.

bit 0:	Clock stop detected
bit 1:	Clock battery monitoring voltage problem
bit 7:	Command error

Return value

1 is returned if the command was successfully issued, otherwise 0 is returned.

sceCdReadIOPm

Read data to IOP memory

Library	*Introduced*	*Documentation last modified*
libcdvd	1.4	July 2, 2001

Syntax

int sceCdReadIOPm (

u_int *lsn,*	Logical sector number at which to begin reading
u_int *sectors,*	Number of sectors to read
void **buf,*	Read buffer (IOP memory)
sceCdRMode **mode)*	Read mode

Calling conditions

Can be called from a thread

Multithread safe (must be called in an interrupt-enabled state)

Description

This function seeks to the read start position indicated by lsn, reads the specified number of sectors from the CD(DVD)-ROM's StartPoint, and fills IOP memory starting at buf. Then the head is placed in pause state.

Notes

It is not permitted to read the CD-DA data's DVD-video data.

This is a non-blocking function, so it is necessary to detect the actual end of transfer using sceCdSync().

Return value

0 if command issue failed.

1 if command issue succeeded.

See also

sceCdSync()

sceCdSearchFile

Get position and size from filename

Library	*Introduced*	*Documentation last modified*
libcdvd	1.1	July 2, 2001

Syntax

int sceCdSearchFile (	
sceCdlFILE **fp,*	Pointer to CD(DVD)-ROM file structure
const char **name***)**	Filename

Calling conditions

Can be called from a thread

Multithread safe (must be called in an interrupt-enabled state)

Description

Determines absolute position LSN (logical sector number) and size from a filename on the CD(DVD)-ROM. The result is stored in **fp*.

Notes

Filenames must be specified fully using absolute paths.

Position information for files in the same directory as the specified file is cached in memory.

Return value

0: No file was found.

1: File structure pointer was successfully obtained.

sceCdSeek

Move CD(DVD)-ROM head

Library	*Introduced*	*Documentation last modified*
libcdvd	1.1	July 2, 2001

Syntax

int sceCdSeek (	
u_int *lsn***)**	Target logical sector number

Calling conditions

Can be called from a thread

Multithread safe (must be called in an interrupt-enabled state)

Description

Seeks CD(DVD)-ROM head to target position and puts head in PAUSE state.

Notes

Since this is a non-blocking function, sceCdSync() must be used to determine completion of the head seek.

Return value

1 is returned if the command was successfully issued, otherwise 0 is returned.

See also

sceCdSync()

sceCdSetEEReadMode

Set mode for reading data

Library	*Introduced*	*Documentation last modified*
libcdvd	2.3	July 2, 2001

Syntax

u_int sceCdSetEEReadMode (	
u_int *mode***);**	Read mode specification Initial value is zero. If multiple values are specified, the logical OR is taken. SCECdNoCheckReady Do not confirm that the drive is ready when a command is issued. SCECdNoWriteBackDCache Do not perform WriteBackDCache to EE memory when a command is issued.

Calling conditions

Can be called from a thread

Multithread safe (must be called in interrupt-enabled state)

Description

This function changes the method by which read commands are issued from the EE to the IOP according to the value of the *mode* argument.

Although the execution speed of a read operation can be increased by leaving out processing, the result of the operation will be indeterminate if there are collisions between commands sent to the drive, and if the cache in EE memory becomes incoherent.

Notes

Use this function with great care.

Return value

The previous setting is returned.

See also

sceCdRead(), sceCdReadChain(), and sceCdReadIOPm

sceCdStandby

Start rotation of the media

Library	*Introduced*	*Documentation last modified*
libcdvd	1.1	July 2, 2001

Syntax

int sceCdStandby(void)

Calling conditions

Can be called from a thread

Multithread safe (must be called in an interrupt-enabled state)

Description

Spins up the CD(DVD)-ROM media and puts the head in PAUSE state at the innermost track.

Notes

Since this is a non-blocking function, sceCdSync() must be used to determine when the actual operation is completed.

Return value

1 is returned if the command was successfully issued, otherwise 0 is returned.

See also

sceCdSync()

sceCdStatus

Get drive status

Library	*Introduced*	*Documentation last modified*
libcdvd	1.2	July 2, 2001

Syntax

int sceCdStatus(void)

Calling conditions

Can be called from a thread

Multithread safe (must be called in an interrupt-enabled state)

Description

Returns current status of drive.

Return value

A -1 is returned if the command was not successfully issued. If the command was successfully issued, the status is returned according to the list below.

Table 2-2

Return value	Meaning
SCECdStatShellOpen	Tray is OPEN
SCECdStatStop	Stopped
SCECdStatSpin	Spindle is spinning
SCECdStatRead	Reading
SCECdStatPause	Paused (unreferenced)
SCECdStatSeek	Seeking
SCECdStatEmg	Abnormal termination

sceCdStInit

Initialize stream

Library	*Introduced*	*Documentation last modified*
libcdvd	1.4	July 2, 2001

Syntax

int sceCdStInit(

u_int *bufmax,*	Capacity of entire stream buffer (specified using number of sectors, in 2048-byte units)
u_int *bankmax,*	Number of stream buffer partitions (number of ring buffers) A buffer with three or more partitions should have a capacity of approximately 16 sectors.
u_int *iop_bufaddr***)**	IOP memory address of stream buffer

Calling conditions

Can be called from a thread

Not multithread safe (must be called in an interrupt-enabled state)

Description

Initializes stream and registers the stream buffer (creates ring buffer).

Notes

Use functions such as sceSifAllocIopHeap() to obtain IOP-side memory (e.g., stream buffer) from the EE.

CD-DA data and DVD-video data cannot be read.

Return value

0 if command issue failed.

1 if command issue succeeded.

See also

sceSifAllocIopHeap(), sceCdStRead(), sceCdStSeek(), sceCdStSeekF(), sceCdStStart(), sceCdStStat(), sceCdStStop(), sceCdStPause(), sceCdStResume()

sceCdStop

Stop rotation of the media

Library	*Introduced*	*Documentation last modified*
libcdvd	1.1	July 2, 2001

Syntax

int sceCdStop(void)

Calling conditions

Can be called from a thread

Multithread safe (must be called in an interrupt-enabled state)

Description

Stops rotation of the CD(DVD)-ROM media.

Notes

Since the function is a non-blocking function, sceCdSync()must be used to determine when the actual operation is finished.

Return value

1 is returned if the command was successfully issued, otherwise 0 is returned.

See also

sceCdSync()

sceCdStPause

Pause stream

Library	*Introduced*	*Documentation last modified*
libcdvd	1.6	July 2, 2001

Syntax

int sceCdStPause(void)

Calling conditions

Can be called from a thread

Not multithread safe (must be called in an interrupt-enabled state)

Description

Pauses the reading of stream data while maintaining the contents of the stream buffer.

Notes

Use sceCdStResume() to restart the reading of stream data.

Return Value

If command issue failed, 0 is returned. If it succeeded, 1 is returned.

See Also

sceCdStInit(), sceCdRead(), sceCdStSeek(), sceCdStSeekF(), sceCdStStart(), sceCdStStat(), sceCdStStop(), sceCdStResume()

sceCdStRead

Read stream data

Library	*Introduced*	*Documentation last modified*
libcdvd	1.4	July 2, 2001

Syntax

int sceCdStRead(

u_int *sectors,*	Number of sectors of data to read from stream buffer
u_int **buf,*	Data read address (should always be 64-byte aligned)
u_int *mode,*	Data read mode STMNBLK: Returns only data currently in stream buffer. STMBLK: Block reads are performed until the specified number of sectors of data are read or an error occurs.
u_int **err)*	Error code storage address Error code is the same as that obtained using sceCdGetError().

Calling conditions

Can be called from a thread

Not multithread safe (must be called in an interrupt-enabled state)

Description

Reads data from the stream buffer.

CD-DA data and DVD-video data cannot be read.

Return value

Returns the number of sectors read (2048-byte units).

See also

sceCdStInit(), sceCdStSeek(), sceCdStStart(), sceCdStStat(), sceCdStSeekF(), sceCdStStop(), sceCdStPause(), sceCdStResume()

sceCdStResume

Restart stream

Library	*Introduced*	*Documentation last modified*
libcdvd	1.6	July 2, 2001

Syntax

int sceCdStResume(void)

Calling conditions

Can be called from a thread

Not multithread safe (must be called in an interrupt-enabled state)

Description

Restarts the reading of stream data (cancels a pause due to the sceCdStPause() function).

Return value

If command issue failed, 0 is returned. If it succeeded, 1 is returned.

See Also

sceCdStInit(), sceCdStRead(), sceCdStSeek(), sceCdStSeekF(), sceCdStStart(), sceCdStStat(), sceCdStStop(), sceCdStPause()

sceCdStSeek

Change stream position

Library	*Introduced*	*Documentation last modified*
libcdvd	1.4	July 2, 2001

Syntax

int sceCdStSeek(

u_int *lsn)*	Changed stream position (specified according to logical sector number)

Calling conditions

Can be called from a thread

Not multithread safe (must be called in an interrupt-enabled state)

Description

Destroys contents of stream buffer and changes the current stream position.

Return value

0 if command issue failed. 1 if command issue succeeded.

See also

sceCdStInit(), sceCdStRead(), sceCdStSeekF(), sceCdStStart(), sceCdStStat(), sceCdStStop(), sceCdStPause(), sceCdStResume()

sceCdStSeekF

Change stream position (high-speed version)

Library	*Introduced*	*Documentation last modified*
libcdvd	2.1	July 2, 2001

Syntax

int sceCdStSeekF(

u_int *lsn)*	Changed stream position (specified according to logical sector number)

Calling conditions

Can be called from a thread

Not multithread safe (must be called in an interrupt-enabled state)

Description

This function discards the stream buffer contents and changes the current stream position.

This entire function has improved performance over sceCdStSeek().

Return value

0 is returned if command issue failed. 1 is returned if it was successful.

See also

sceCdStInit(), sceCdStRead(), sceCdStSeek(), sceCdStStart(), sceCdStStat(), sceCdStStop(), sceCdStPause(), sceCdStResume()

sceCdStStart

Start streaming

Library	*Introduced*	*Documentation last modified*
libcdvd	1.4	October 11, 2001

Syntax

int sceCdStStart(

u_int *lsn,*	Stream start position (specified using logical sector number)
sceCdRMode **mode*)	Read mode

Calling conditions

Can be called from a thread

Not multithread safe (must be called in an interrupt-enabled state)

Description

Starts reading from the specified stream start position into the stream buffer.

After streaming starts, data is read from the CD(DVD) into the streaming buffer recurrently in the background. This means that functions like the file control functions and sceCDRead() cannot be used to read from the CD(DVD)-ROM until streaming has been stopped with sceCdStStop().

The only value that can be specified for datapattern *mode* is SCECdSecS2048.

CD-DA data and DVD-video data cannot be read.

Return value

0 if command issue failed. 1 if command issue succeeded.

See also

sceCdStInit(), sceCdStRead(), sceCdStSeek(), sceCdStSeekF(), sceCdStStat(), sceCdStStop(), sceCdStPause(), sceCdStResume()

sceCdStStat

Get stream data read status

Library	*Introduced*	*Documentation last modified*
libcdvd	1.4	July 2, 2001

Syntax

int sceCdStStat (void)

Calling conditions

Can be called from a thread

Not multithread safe (must be called in an interrupt-enabled state)

Description

Gets current read status of stream data.

Return value

0 if command issue failed. On success, returns the number of sectors of data that have been accumulated in the stream (in 2048-byte units).

See also

sceCdStInit(), sceCdStRead(), sceCdStSeek(), sceCdStSeekF(), sceCdStStart(), sceCdStStop(), sceCdStPause(), sceCdStResume()

sceCdStStop

Stop streaming

Library	*Introduced*	*Documentation last modified*
libcdvd	1.4	July 2, 2001

Syntax

int sceCdStStop (void)

Calling conditions

Can be called from a thread

Not multithread safe (must be called in an interrupt-enabled state)

Description

Stops streaming.

Return value

0 if command issue failed. 1 if command issue succeeded.

See also

sceCdStInit(), sceCdStRead(), sceCdStSeek(), sceCdStSeekF(), sceCdStStart(), sceCdStStat(), sceCdStPause(), sceCdStResume()

sceCdSync

Wait for command completion

Library	*Introduced*	*Documentation last modified*
libcdvd	1.1	July 2, 2001

Syntax

int sceCdSync (

int *mode)*	0x00: Wait for completion of command (blocking) 0x01: Check current status and return immediately (non-blocking). When using this mode, the operating conditions of other threads must be thoroughly considered.

Calling conditions

The blocking type cannot be called in interrupt-disabled state.

Can be called from a thread

Multithread safe (must be called in an interrupt-enabled state)

Description

When *mode* is set to 0x00, this function waits for the command being executed to complete and returns 0.

When *mode* is set to 0x01, this function checks the execution state of the command and returns either 0 or 1.

When this function is used for polling in non-blocking mode in a multithreaded environment, a function such as DelayThread() must be used so that there is sufficient room for other threads to operate.

Return value

0: Completed, 1: Not completed

See also

sceCdRead(), sceCdSeek(), sceCdStop(), sceCdStandby(), sceCdGetToc()

sceCdTrayReq

Open and close the tray

Library	*Introduced*	*Documentation last modified*
libcdvd	1.3	July 2, 2001

Syntax

int sceCdTrayReq(

int *mode,*	Tray control mode SCECdTrayOpen: Open tray SCECdTrayClose: Close tray SCECdTrayCheck: Get tray state change
u_int **traycnt***)**	Address for returning whether or not there was a tray state change 0: Tray was not opened. 1: Tray was opened.

Calling conditions

Can be called from a thread

Multithread safe (must be called in an interrupt-enabled state)

Description

This function opens or closes the tray of the CD(DVD)-ROM drive according to the specified mode.

When *mode* is SCECdTrayCheck, the mode for getting the tray state change is set, and information indicating whether or not the tray was opened since the previous time this command was called in this mode is returned in **traycnt*.

Notes

Use sceCdDiskReady() to determine whether or not commands can be received after a disk has been inserted.

Return value

0 if command issue failed. 1 if command issue succeeded.

File Control Functions

File control functions

Library	*Introduced*	*Documentation last modified*
libcdvd	2.4	October 11, 2001

Syntax

The following file control functions are supported.

#include <sifdev.h>	Refer to the documentation on the standard I/O functions for the arguments.
sceClose()	
sceDclose()	
sceDevctl()	
sceDopen()	
sceDread()	
sceIoctl2()	
sceLseek()	
sceOpen()	Additional arguments for the sceOpen() function:
sceRead()	

sceOpen() additional arguments:

filename cdrom0: + filename (ISO9660 Level 1)

flags Access mode. Specify either of the following constants.

SCE_RDONLY	Open only for reading
SCE_CdSTREAM	Open only for reading a stream

Description

File-based I/O functions are supported.

Precautions when the file is opened with SCE_CdSTREAM for reading a stream:

1. The size argument of the sceRead() function must be specified as a multiple of 2048.
2. The CDIOC_GETERROR command must be used to obtain the read error.
3. After the file is opened, data is recursively read from the CD(DVD)-ROM to the streaming buffer in the background. Therefore, the file control functions and functions such as sceCdRead() cannot be used to read from the CD(DVD)-ROM until streaming is terminated using the sceClose() function.

devctl Commands

CDIOC_BREAK

Interrupt command

Library	*Introduced*	*Documentation last modified*
libcdvd	2.4	October 11, 2001

Arguments

arg	Reserved. Set to NULL.
arglen	Reserved. Set to 0.
bufp	Reserved. Set to NULL.
buflen	Reserved. Set to 0.

Description

This command interrupts a currently executing command (such as sceRead(), CDIOC_STANDBY, CDIOC_STOP, CDIOC_PAUSE, sceCdPause(), sceCdRead(), sceCdSeek(), sceCdStandby(), sceCdSstatus(), or sceCdstop()).

When a command is interrupted, a callback function is called if one was previously set.

SCECdErABRT will be set for drive error information.

Return value

If processing succeeds, 0 is returned.

On error, the product of errno and -1 is returned.

CDIOC_DISKRDY

Check drive state

Library	*Introduced*	*Documentation last modified*
libcdvd	2.3.4	August 31, 2001

Arguments

arg	Check mode (0: Blocking, 1: Non-blocking) storage address
arglen	sizeof(int)
bufp	Drive state storage address
buflen	sizeof(int)

Description

The CDIOC_DISKRDY command checks the following drive states to determine whether or not a command can be issued.

SCECdComplete is the drive state that allows commands to be issued, and SCECdNotReady is the state in which the drive cannot accept commands. The state becomes SCECdNotReady when there is no media in the drive.

For the blocking case, if the drive rotation is unstable, the function will wait until the rotation becomes stable before returning. For the non-blocking case, the function will return immediately after checking the status.

When this function is used for non-blocking polling in a multithread environment, the DelayThread() (or equivalent) function must be used to make sure there is room for other threads to run.

Return value

If processing succeeds, 0 is returned.

If an error occurs, the product of errno and -1 is returned.

CDIOC_GETDISKTYP

Get media format

Library	*Introduced*	*Documentation last modified*
libcdvd	2.3.4	August 31, 2001

Arguments

arg	Reserved. Specify NULL.
arglen	Reserved. Specify 0.
bufp	Media type storage address
buflen	sizeof(int)

Description

This command obtains one of the following media formats.

SCECdIllgalMedia	Play-prohibited disc
SCECdPS2DVD	Disc is a PlayStation 2 DVD
SCECdPS2CD	Disc is a PlayStation 2 CD
SCECdPS2CDDA	Disc is a PlayStation 2 CD (with CDDA)
SCECdPSCD	Disc is a PlayStation CD
SCECdPSCDDA	Disc is a PlayStation CD (with CDDA)
SCECdDVDV	Disc is a DVD Video
SCECdCDDA	Disc is a music CD
SCECdDETCT	Format detection in progress
SCECdNODISC	No disc has been inserted
SCECdUNKNOWN	Unknown disc format

Return value

If processing succeeds, 0 is returned.

If an error occurs, the product of errno and -1 is returned.

CDIOC_GETERROR

Get drive error information

Library	*Introduced*	*Documentation last modified*
libcdvd	2.3.4	August 31, 2001

Arguments

arg	Reserved. Specify NULL.
arglen	Reserved. Specify 0.
bufp	Error information storage address
buflen	sizeof(int)

Description

This command obtains one of the following kinds of drive error information.

SCECdErFAIL	Processing for issuing sceCdGetError() function failed
SCECdErNO	No error occurred
SCECdErEOM	Reached outermost periphery during play
SCECdErTRMOPN	Drive was opened during play
SCECdErREAD	Problem occurred while reading
SCECdErCUD	Improper disc in drive
SCECdErNORDY	Command is being processed
SCECdErABRT	Command aborted
SCECdErREADCF	Read command issue failed

Return value

If processing succeeds, 0 is returned.

If an error occurs, the product of errno and -1 is returned.

CDIOC_GETTOC

Read TOC

Library	*Introduced*	*Documentation last modified*
libcdvd	2.3.4	August 31, 2001

Arguments

arg	Reserved. Specify NULL.
arglen	Reserved. Specify 0.
bufp	TOC storage address. 1024-byte area is required.
buflen	1024

Description

This command gets the TOC sector information of the CD-ROM.

Return value

If processing succeeds, 0 is returned.

If an error occurs, the product of errno and -1 is returned.

CDIOC_MMODE

Specify read media

Library	*Introduced*	*Documentation last modified*
libcdvd	2.3.4	August 31, 2001

Arguments

arg	Read media storage address
arglen	sizeof(int)
bufp	Reserved. Specify NULL.
buflen	Reserved. Specify 0.

Description

This command specifies one of the following read media types for the CD(DVD-ROM) subsystem.

SCECdCD	Specifies CD as the read media.
SCECdDVD	Specifies DVD as the read media.

Return value

If processing succeeds, 0 is returned.

If an error occurs, the product of errno and -1 is returned.

CDIOC_PAUSE

Pause CD(DVD)-ROM head

Library	*Introduced*	*Documentation last modified*
libcdvd	2.3.4	August 31, 2001

Arguments

arg	Reserved. Specify NULL.
arglen	Reserved. Specify 0.
bufp	Reserved. Specify NULL.
buflen	Reserved. Specify 0.

Description

This command pauses the CD(DVD)-ROM head at its current location.

This call will block until processing ends.

Return value

If processing succeeds, 0 is returned.

If an error occurs, the product of errno and -1 is returned.

CDIOC_POWEROFF

Power off PlayStation 2

Library	*Introduced*	*Documentation last modified*
libcdvd	2.3.4	August 31, 2001

Arguments

arg	Reserved. Specify NULL.
arglen	Reserved. Specify 0.
bufp	Status storage address
buflen	sizeof(int)

Description

This command issues a request to power off the PlayStation 2. For details, see the sceCdPowerOff() function reference.

Return value

If processing succeeds, 0 is returned.

If an error occurs, the product of errno and -1 is returned.

CDIOC_READCLOCK

Get date and time

Library	*Introduced*	*Documentation last modified*
libcdvd	2.3.4	August 31, 2001

Arguments

arg	Reserved. Specify NULL.
arglen	Reserved. Specify 0.
bufp	Address of date/time storage structure sceCdClock for storing the date and time
buflen	sizeof(sceCdClock)

Description

This command gets the date and time. See the description of sceCdReadClock().

Return value

If processing succeeds, 0 is returned.

If an error occurs, the product of errno and -1 is returned.

CDIOC_SPINNOM

Set adaptive speed control for the standard I/O media spin rate

Library	*Introduced*	*Documentation last modified*
libcdvd	2.3.4	August 31, 2001

Arguments

arg	Reserved. Specify NULL.
arglen	Reserved. Specify 0.
bufp	Reserved. Specify NULL.
buflen	Reserved. Specify 0.

Description

This command sets adaptive speed control for the standard I/O media spin rate. This causes data reading to begin at the highest spin rate, and when a read error occurs, it lowers the spin rate until reading can be performed properly.

The initial value for the standard I/O spin rate is adaptive speed control.

Return value

If processing succeeds, 0 is returned.

If an error occurs, the product of errno and -1 is returned.

CDIOC_STANDBY

Start media rotation

Library	*Introduced*	*Documentation last modified*
libcdvd	2.3.4	August 31, 2001

Arguments

arg	Reserved. Specify NULL.
arglen	Reserved. Specify 0.
bufp	Reserved. Specify NULL.
buflen	Reserved. Specify 0.

Description

This command causes the CD(DVD)-ROM media to rotate, positions the head at the innermost circumference, and sets pause state.

This call blocks until processing ends.

Return value

If processing succeeds, 0 is returned.

If an error occurs, the product of errno and -1 is returned.

CDIOC_STATUS

Get drive state

Library	*Introduced*	*Documentation last modified*
libcdvd	2.3.4	August 31, 2001

Arguments

arg	Reserved. Specify NULL.
arglen	Reserved. Specify 0.
bufp	Drive status storage address
buflen	sizeof(int)

Description

This command returns one of the following as the current drive state.

SCECdStatShellOpen	Tray is open
SCECdStatStop	Stop state
SCECdStatSpin	Spindle is rotating
SCECdStatRead	Read operation is executing (cannot be referenced)
SCECdStatPause	Pause state (cannot be referenced)
SCECdStatSeek	Seeking
SCECdStatEmg	Emergency stop

Return value

If processing succeeds, 0 is returned.

If an error occurs, the product of errno and -1 is returned.

CDIOC_STOP

Stop media rotation

Library	*Introduced*	*Documentation last modified*
libcdvd	2.3.4	August 31, 2001

Arguments

arg	Reserved. Specify NULL.
arglen	Reserved. Specify 0.
bufp	Reserved. Specify NULL.
buflen	Reserved. Specify 0.

Description

This command stops the rotation of the CD(DVD)-ROM media.

This call blocks until processing ends.

Return value

If processing succeeds, 0 is returned.

If an error occurs, the product of errno and -1 is returned.

CDIOC_STREAMINIT

Initialize streamer for file I/O functions

Library	*Introduced*	*Documentation last modified*
libcdvd	2.4	October 11, 2001

Arguments

arg	Reserved. Set to NULL.
arglen	Reserved. Set to 0.
bufp	Address of sceCdStmInit, the file I/O function stream initialization structure
buflen	sizeof(sceCdStmInit)

Description

This command initializes the streamer for file I/O functions and registers the stream buffer (creates a ring buffer).

Return value

On error, the product of errno and -1 is returned.

CDIOC_TRAYREQ

Open/close tray

Library	*Introduced*	*Documentation last modified*
libcdvd	2.3.4	August 31, 2001

Arguments

arg	Tray control mode
arglen	sizeof(int)
bufp	Address where tray state change, if present, is returned
buflen	sizeof(u_int)

Description

This command opens or closes the CD(DVD)-ROM drive tray according to the tray control mode specification.

If SCECdTrayCheck was specified for the tray control mode, the mode will become tray state change acquisition mode, and whether or not the tray was opened since the last time this command was called in tray state change acquisition mode is returned in the tray state change address.

Return value

If processing succeeds, 0 is returned.

If an error occurs, the product of errno and -1 is returned.

CDIOC_TRYCNT

Set media read retry count for standard I/O

Library	*Introduced*	*Documentation last modified*
libcdvd	2.3.4	August 31, 2001

Arguments

arg	Read retry count storage address (0 <= Retry count <= 255; 0: 256 times)
arglen	sizeof(u_char)
bufp	Reserved. Specify NULL.
buflen	Reserved. Specify 0.

Description

This command sets the media read retry count for standard I/O. The initial value is set to 16 times.

Return value

If processing succeeds, 0 is returned.

If an error occurs, the product of errno and -1 is returned.

ioctl2 Commands

CDIOSTREAMSTAT

Get stream data read status

Library	*Introduced*	*Documentation last modified*
libcdvd	2.4	October 11, 2001

Arguments

arg	Reserved. Set to NULL.
arglen	Size of arg.
bufp	Reserved. Set to NULL.
buflen	Size of bufp.

Description

This command gets the current stream data read status.

Return value

On error, the product of errno and -1 is returned.

If processing succeeds, the amount of data already accumulated in the streamer is returned as the number of sectors (2048-byte units).

CIOCSTREMPAUSE

Pause stream

Library	*Introduced*	*Documentation last modified*
libcdvd	2.4	October 11, 2001

Arguments

arg	Reserved. Set to NULL.
arglen	Size of arg.
bufp	Reserved. Set to NULL.
buflen	Size of bufp.

Description

This command pauses the reading of stream data while maintaining the contents of the stream buffer.

Notes

Stream data reading can be resumed with CIOCSTREMRESUME.

Return value

If processing succeeds, 0 is returned.

On error, the product of errno and -1 is returned.

CIOCSTREMRESUME

Resume stream

Library	*Introduced*	*Documentation last modified*
libcdvd	2.4	October 11, 2001

Arguments

arg	Reserved. Set to NULL.
arglen	Size of arg.
bufp	Reserved. Set to NULL.
buflen	Size of bufp.

Description

This function resumes the reading of stream data (cancels a pause set by CIOCSTREMPAUSE).

Notes

To obtain IOP memory such as the stream buffer from the EE, use a function such as sceSifAllocIopHeap().

CD-DA data and DVD-video data cannot be read.

Return value

If processing succeeds, 0 is returned.

On error, the product of errno and -1 is returned.

Chapter 3: Hard Disk Library (for EE)
Table of Contents

Structures

sce_dirent

Partition table entry

Library	*Introduced*	*Documentation last modified*
hdd	2.2.2	April 16, 2001

Structure

struct sce_dirent {	
struct sce_stat *d_stat;*	Partition status
char *d_name***[256];**	Partition ID
void **d_private* **};**	Reserved

Description

This structure stores an entry of the partition table.

See also

sceDread()

sce_stat

Partition status

Library	*Introduced*	*Documentation last modified*
hdd	2.2.2	April 16, 2001

Structure

struct sce_stat {	
unsigned int *st_mode;*	Filesystem type of the partition
unsigned int *st_attr;*	bit 0 Sub-partition
unsigned int *st_size;*	Number of sectors in the partition
unsigned char *st_ctime***[8];**	Creation time of the partition
unsigned char *st_atime***[8];**	byte 0 reserved
unsigned char *st_mtime***[8];**	byte 1 Seconds byte 2 Minutes byte 3 Hours byte 4 Day byte 5 Month byte 6-7 Year (4 digits)
unsigned int *st_hisize;*	
unsigned int *st_private***[6] };**	word 0 For the main partition, represents the number of sub-partitions. For a sub-partition, represents the sub-partition number starting from 1.

Description

This structure stores partition status.

See also

struct sce_dirent

Functions

sceClose

Close main partition

Library	*Introduced*	*Documentation last modified*
hdd	2.2.2	April 16, 2001

Syntax

#include <sifdev.h>	
int sceClose(	
int *fd***)**	Previously open file descriptor

Calling conditions

Can be called from a thread

Multithread safe (must be called in interrupt-enabled state)

Description

Closes the opened partition and frees the file descriptor.

Return value

0 if successful.

-1 times errno if an error occurred.

EBADF	*fd* is not a valid open descriptor.

sceDclose

Close partition table

Library	*Introduced*	*Documentation last modified*
hdd	2.2.2	April 16, 2001

Syntax

#include <sifdev.h>

int sceDclose(

int *fd***)**	File descriptor

Calling conditions

Can be called from a thread

Multithread safe (must be called in interrupt-enabled state)

Description

Closes the opened partition table and frees the file descriptor.

Return value

0 if successful.

-1 times errno if an error occurred.

EBADF	*fd* is not a valid open descriptor.

sceDevctl

Special operations for a device

Library	*Introduced*	*Documentation last modified*
hdd	2.2.2	April 16, 2001

Syntax

#include <sifdev.h>

int sceDevctl(

const char **name,*	Device name (hdd0:, hdd1:).
int *cmd,*	Operation command. Any of the following constants can be specified. HDIOC_MAXSECTOR HDIOC_TOTALSECTOR HDIOC_IDLE HDIOC_FLUSH HDIOC_SWAPTMP HDIOC_DEV9OFF HDIOC_STATUS HDIOC_FORMATVER
void **arg,*	Command arguments. Depends on *cmd*.
unsigned int *arglen,*	Size of *arg*.
void **bufp,*	Arguments received from the command. Depends on *cmd*.
unsigned int *buflen***)**	Size of *bufp*.

Calling conditions

Can be called from a thread

Multithread safe (must be called in interrupt-enabled state)

Description

Performs special operations for a device. For details regarding each of the commands, refer to the "devctl command list".

Return value

If successful, returns a command-dependent value.

If an error occured, returns -1 times errno.

The errors that are common to each of the commands are as follows.

EMFILE	Reached the maximum number of descriptors that can be opened.
ENODEV	Specified device does not exist.

sceDopen

Open partition table

Library	*Introduced*	*Documentation last modified*
hdd	2.2.2	April 16, 2001

Syntax

#include <sifdev.h>

int sceDopen(

const char **name*)**	Device name (hdd0:, hdd1:)

Calling conditions

Can be called from a thread

Multithread safe (must be called in interrupt-enabled state)

Description

Opens a partition table. For obtaining information about all the partitions present on the disk, the partition table is viewed as a simulated directory.

Return value

Returns file descriptor on normal completion (value > 0).

Returns -1 times errno if an error occurred.

EMFILE	Reached the maximum number of descriptors that can be opened.
ENODEV	Specified device does not exist.

sceDread

Read partition table

Library	*Introduced*	*Documentation last modified*
hdd	2.2.2	April 16, 2001

Syntax

#include <sifdev.h>

int sceDread(

int *fd,*	File descriptor
struct sce_dirent **buf*)	Address of the buffer that stores the data that was read.

Calling conditions

Can be called from a thread

Multithread safe (must be called in interrupt-enabled state)

Description

The next entry from the partition entry stream indicated by *fd* is copied to the sce_dirent structure *buf*. Returns 0 when reaches the end of entries.

Return value

Returns the length of the partition ID string on success. Returns 0 when the end of entries is reached.

Returns -1 times errno if an error occurred.

EBADF	*fd* is not a valid open descriptor.
EIO	I/O error.
ENOMEM	Not enough free memory.
ENOTDIR	*fd* is not a descriptor of the partition table.

sceFormat

Format hard disk drive

Library	*Introduced*	*Documentation last modified*
hdd	2.2.2	April 16, 2001

Syntax

#include <sifdev.h>	
int sceFormat(	
const char **devname,*	Device name (hdd0:, hdd1:)
const char **blockdevname,*	Reserved. Specify NULL.
void **arg,*	Reserved. Specify NULL.
int *arglen***)**	Size of *arg*.

Calling conditions

Can be called from a thread

Multithread safe (must be called in interrupt-enabled state)

Description

Formats the hard disk drive with the specified unit number. The required partition is created in advance by the system.

Notes

For use only during title development and should not be incorporated within a title. Care should be taken when using this command as this operation initializes the information of all partitions on the disk.

Return value

0 if successful. -1 times errno if an error occurred.

EIO	I/O error.
EMFILE	Reached the maximum number of descriptors that can be opened.
ENODEV	Specified device does not exist.
ENXIO	Disk for the specified unit number does not exist.

sceGetstat

Get partition status

Library	*Introduced*	*Documentation last modified*
hdd	2.2.2	April 16, 2001

Syntax

#include <sifdev.h>

int sceGetstat(

const char **name,*	Partition identifier string. If a password has been set then minimally, the read-only password must be specified.
struct sce_stat **buf*)	buffer for storing the status.

Calling conditions

Can be called from a thread

Multithread safe (must be called in interrupt-enabled state)

Description

Copies partition information to the sce_dirent structure buf.

Return value

Returns zero on success.

-1 times errno if an error occurred.

EACCES	No access rights.
EINVAL	Incorrect arguments were specified.
EIO	I/O error.
EMFILE	Reached the maximum number of descriptors that can be opened.
ENODEV	Specified device does not exist.
ENOMEM	Not enough free memory.

sceIoctl2

Special operations for a partition

Library	*Introduced*	*Documentation last modified*
hdd	2.2.2	April 16, 2001

Syntax

#include <sifdev.h>

int sceIoctl2(

int *fd,*	Target file descriptor
int *cmd,*	Operation command. Any of the following constants can be specified. HIOCADDSUB HIOCDELSUB HIOCNSUB HIOCFLUSH
void **arg,*	Command arguments. Depends on *cmd*.
unsigned int *arglen,*	Size of *arg*.
void **bufp,*	Arguments received from the command. Depends on *cmd*.
unsigned int *buflen*)	Size of the *bufp*.

Calling conditions

Can be called from a thread

Multithread safe (must be called in interrupt-enabled state)

Description

Performs special operations on a partition. For details regarding each *cmd*, refer to the "Ioctl2 command table".

Return value

Returns a command-dependent value if successful.

-1 times errno if an error occurred.

The errors that are common to each of the commands are as follows.

EBADF	*fd* is not a valid open descriptor.
EINVAL	Incorrect arguments specified.

sceLseek

Move extended attribute area file pointer of partition

Library	*Introduced*	*Documentation last modified*
hdd	2.2.2	April 16, 2001

Syntax

#include <sifdev.h>

int sceLseek(

int *fd,*	File descriptor of partition for which the pointer will be moved
int *offset,*	Distance to move pointer (multiple of 512 bytes)
int *whence***)**	Reference position of *offset* in the extended attribute area of the partition. Any of the following constants can be specified.

SCE_SEEK_SET	Starting position
SCE_SEEK_CUR	Current position
SCE_SEEK_END	End

Calling conditions

Can be called from a thread

Multithread safe (must be called in interrupt-enabled state)

Description

The offset of file descriptor *fd* is changed to the position specified by the *offset* argument, and according to *whence*. The offset cannot be set to a position exceeding the end of the extended attribute area of the partition.

Return value

On success, returns the updated value of the file pointer.

On error, returns -1 times errno.

EBADF	*fd* is not a valid open descriptor.
EINVAL	The specified size is not a multiple of 512. *whence* is an incorrect value or an *offset* beyond the EOF was specified.
EIO	I/O error.

sceOpen

Create, open main partition

Library	*Introduced*	*Documentation last modified*
hdd	2.2.2	April 16, 2001

Syntax

#include <sifdev.h>	
int sceOpen(	
const char **name,*	Partition identifier string.
int *flags***)**	Access mode. Any of the following constants can be specified. A logical OR is performed if more than one is specified. SCE_RDONLY Open as read-only. SCE_RDWR Open as read/write. SCE_CREAT Create a new partition if one does not exist.

Calling conditions

Can be called from a thread

Multithread safe (must be called in interrupt-enabled state)

Description

Creates, opens the main partition. Assigns a file descriptor to the file that has been opened. Certain partitions cannot be opened simultaneously. The partition identifier string consists of device name + unit number + ':'+ a string made from a sequence of the following strings separated by commas.

Partition ID	This is a unique ID for the entire disk and is formally issued by SCE, however, any character string can be used as long as it is unique during the creation stage and is no more than 32 characters long.
Full password	The password required for read/write access. The password can be up to 8 characters long.
Read-only password	The password for read-only access. The password can be up to 8 characters long.
Partition size	Character string which specifies the size of the partition. The valid characters strings are shown below: 128M, 256M, 512M, 1G, 2G, 4G, 8G, 16G, 32G
Filesystem name	At present, only "PFS" is valid.

All of these need to be specified for creation, except for the passwords. To open a partition, specify up to the required password.

Example 1: Creation with password specifications

```
sceOpen("hdd0:BISLPS-XXXXX,fpwd,rpwd,128M,PFS",SCE_CREAT|SCE_RDWR);
```

Example 2: Creation without password specifications

```
sceOpen("hdd0:BISLPS-XXXXX,,,128M,PFS", SCE_CREAT|SCE_RDWR);
```

Example 3: Open with a password

```
sceOpen("hdd0:BISLPS-XXXXX,fpwd", SCE_RDWR);
```

Example 4: Open without a password

```
sceOpen("hdd0:BISLPS-XXXXX", SCE_RDWR);
```

Example 5: Open with a read-only password

```
sceOpen("hdd0:BISLPS-XXXXX,,rpwd", SCE_RDONLY);
```

Notes

If an opened partition is not closed before the filesystem driver performs a format or mount of the partition, then an EBUSY error is returned.

Return value

Returns the file descriptor on normal completion (value > 0).

-1 times errno if an error occurred.

EACCES	No access rights.
EBUSY	The specified partition is already open.
EINVAL	Incorrect arguments were specified.
EIO	I/O error.
EMFILE	Reached maximum number of descriptors that can be opened.
ENODEV	Specified device does not exist.
ENOENT	Specified partition does not exist.
ENOMEM	Not enough free memory.
ENOSPC	No free space.

sceRead

Read from the extended attribute area of a partition

Library	*Introduced*	*Documentation last modified*
hdd	2.2.2	April 16, 2001

Syntax

#include <sifdev.h>

int sceRead(

int *fd,*	File descriptor of the read target
void **buf,*	Address of the buffer that will store the read data
int *count***)**	Read data size (multiple of 512 bytes)

Calling conditions

Can be called from a thread

Multithread safe (must be called in interrupt-enabled state)

Description

Reads a maximum of *count* bytes from the extended attribute area of the partition that was previously opened, into the buffer starting from the address specified by *buf*. *count* must be a multiple of 512. Specifying any other value than this results in an error.

Also, the buffer must be 64-byte aligned on the EE.

Notes

SPR cannot be specified to *buf.*

Return value

On success, the number of bytes read are returned. The file position is advanced by this amount only. A return value of 0 means end of file. If an error occurred, -1 times errno is returned.

EBADF	*fd* is not a valid open descriptor.
EINVAL	The specified size is not a multiple of 512.
EIO	I/O error.

sceRemove

Delete partition

Library	*Introduced*	*Documentation last modified*
hdd	2.2.2	April 16, 2001

Syntax

#include <sifdev.h>	
int sceRemove(	
const char **name*)	Partition identifier string. If a password is specified, specifications are required up to the full password.

Calling conditions

Can be called from a thread

Multithread safe (must be called in interrupt-enabled state)

Description

Deletes the specified partition. All sub-partitions that were added are also deleted.

Return value

Returns zero on success.

-1 times errno if an error occurred.

EACCES	No access rights.
EBUSY	The specified partition is already open.
EINVAL	Incorrect arguments were specified.
EIO	I/O error.
EMFILE	Reached maximum number of descriptors that can be opened.
ENODEV	Specified device does not exist.
ENOMEM	Not enough free memory.

sceWrite

Write to the extended attribute area of partition

Library	*Introduced*	*Documentation last modified*
hdd	2.2.2	April 16, 2001

Syntax

#include <sifdev.h>	
int sceWrite(	
int *fd,*	File descriptor of the write target.
const void **buf,*	Address of the buffer that stores the write data
int *count***)**	Write data size (multiple of 512 bytes)

Calling conditions

Can be called from a thread

Multithread safe (must be called in interrupt-enabled state)

Description

Writes a maximum of *count* bytes from the buffer indicated by *buf* into the extended attribute area of the partition referenced by the file descriptor *fd*. *count* must be a multiple of 512. Specifying any other value than this results in an error.

The buffer must be 64-byte aligned.

Notes

SPR cannot be specified to *buf*.

Return value

On success, returns the number of bytes written. The file position is advance by this amount only.

If an error occurred, -1 times errno is returned.

EACCES	No write permission.
EBADF	*fd* is not a valid open descriptor.
EINVAL	The specified size is not a multiple of 512.
EIO	I/O error.

devctl Commands

HDIOC_DEV9OFF

Power OFF device

Library	*Introduced*	*Documentation last modified*
hdd	2.2.2	April 16, 2001

Arguments

arg	Reserved. Specify NULL.
arglen	Size of *arg*.
bufp	Reserved. Specify NULL.
buflen	Size of *bufp*.

Description

Powers off the entire dev9 device to which the hard disk drive is connected.

This processing should be performed before powering off the main unit.

Note: When this processing is performed, other devices connected to dev9 (the network adapter) are also powered off.

Return value

0 if successful.

HDIOC_FLUSH

Flush the disk cache

Library	*Introduced*	*Documentation last modified*
hdd	2.2.2	April 16, 2001

Syntax

arg	Reserved. Specify NULL.
arglen	Size of *arg*.
bufp	Reserved. Specify NULL.
buflen	Size of *bufp*.

Description

Flushes the cache on the disk. Usually, the application is not required to perform this operation.

Return value

0 if successful.

-1 times errno if an error occurred.

EIO	I/O error.

HDIOC_FORMATVER

Get partition system version

Library	*Introduced*	*Documentation last modified*
hdd	2.2.2	April 16, 2001

Syntax

arg	Reserved. Specify NULL.
arglen	Size of *arg*.
bufp	Reserved. Specify NULL.
buflen	Size of *bufp*.

Description

Gets the version of the formatted partition system. Usually, the application is not required to verify the version.

Return value

Returns the version of the partition system.

HDIOC_FREESECTOR

Get installable size

Library	*Introduced*	*Documentation last modified*
hdd	2.3.1	July 26, 2001

Syntax

arg	Reserved. Specify NULL.
arglen	Size of *arg*.
bufp	Pointer to an unsigned 32-bit integer, for storing installable size, in sectors.
buflen	Size of *bufp*.

Description

Returns the installable size in *bufp*. The installable size will be equal to the disk's free space as indicated in the browser. All free space over 1 GB is added, and for free space less than 1 GB (512M, 256M and 128M) an addition is performed for up to one area, respectively. If there is more than one free area less than 1 GB, those areas will not be added as such, but as aggregates of smaller areas. However, in this case, any area that has been counted once will not be counted again.

For example, assume there are two areas of 512 MB and one of 128 MB. In this case, the first 512 MB area is simply counted; the second 512 MB area is counted as areas of 256 MB and 128 MB. Upon finding the next 128 MB area, since 128 MB has already been counted once, that area is not counted -it is ignored.

Note: As shown in the example above, the return value for this devctl command does not simply give the amount of free space. Rather, it may return a value that is smaller than the actual partition size that can be created. To find the actual partition size that can be created, the partition list and the capacity of the entire disk would need to be obtained.

Return value

Returns 0 if successful

On error, returns -1 times errno

EIO: input/output error.

HDIOC_IDLE

Set idle mode

Library	*Introduced*	*Documentation last modified*
hdd	2.2.2	April 16, 2001

Syntax

arg	Pointer to an 8-bit variable that stores the setting value.
arglen	Size of *arg*.
bufp	Reserved. Specify NULL.
buflen	Size of *bufp*.

Description

Sets the amount of time after which the hard disk drive will transition to idle mode.

The default time for the hdd module to transition to idle mode is 21 minutes and 15 seconds. The settable values are shown below.

0x00	timeout disable
0x01 - 0xf0	(value * 5) s
0xf1 - 0xfb	((value - 240) * 30) min
0xfc	21 min
0xfd	Period between 8 and 12 hours
0xfe	Reserved
0xff	21 min 15 s

Example:

```
u_char standbytimer = 0xff;
devctl("hdd0:", HDIOC_IDLE, &standbytimer, sizeof(char), NULL, 0);
```

Return value

0 if successful.

-1 times errno if an error occurred.

EIO	I/O error.

HDIOC_MAXSECTOR

Get maximum size of partition that can be created

Library	*Introduced*	*Documentation last modified*
hdd	2.2.2	April 16, 2001

Syntax

arg	Reserved. Specify NULL.
arglen	Size of *arg*.
bufp	Reserved. Specify NULL.
buflen	Size of *bufp*.

Description

Gets the maximum size of a partition that can be created (in units of sectors).

Return value

Returns a value that is a power of 2 (2 ^ n) as an unsigned 32-bit integer.

HDIOC_SMARTSTAT

Check for hard disk drive failure

Library	*Introduced*	*Documentation last modified*
hdd	2.3	July 26, 2001

Syntax

arg	Reserved. Specify NULL.
arglen	Size of *arg*.
bufp	Reserved. Specify NULL.
buflen	Size of *bufp*.

Description

Checks for the presence of a failure using the hard disk drive SMART function.

Return value

Returns 0 if there is no failure and 1 if there is a failure.

In case of an error, -EIO is returned.

HDIOC_STATUS

Get hard disk drive status

Library	*Introduced*	*Documentation last modified*
hdd	2.2.2	January 4, 2002

Syntax

arg	Reserved. Specify NULL.
arglen	Size of *arg.*
bufp	Reserved. Specify NULL.
buflen	Size of *bufp*.

Description

Gets hard disk drive status.

Return value

Returns the following status:

3: Hard disk drive not connected.
2: (Reserved)
1: Not formatted.
0: Normal

HDIOC_SWAPTMP

Exchange partition information with _tmp

Library	*Introduced*	*Documentation last modified*
hdd	2.2.2	April 16, 2001

Syntax

arg	Partition identifer string.
arglen	Size of *arg.*
bufp	Reserved. Specify NULL.
buflen	Size of *bufp.*

Description

Exchanges partition information with the _tmp partition.

Creates a new partition and copies the contents of the existing partition, then deletes the copy source. Can be used as a substitute for defrag, etc. Processing is similar to that of the filesystem rename(), etc., however, this command uses _tmp instead of creating a partition with a new partition ID.

Return value

0 if successful.

-1 times errno if an error occurred.

EACCES	No access rights.
EINVAL	Incorrect arguments were specified.
EIO	I/O error.
ENOMEM	Not enough free memory.

HDIOC_TOTALSECTOR

Get total number of sectors on the disk

Library	*Introduced*	*Documentation last modified*
hdd	2.2.2	April 16, 2001

Syntax

arg	Reserved. Specify NULL.
arglen	Size of *arg.*
bufp	Reserved. Specify NULL.
buflen	Size of *bufp.*

Description

Gets total number of sectors on the disk.

Return value

Returns an unsigned 32-bit integer.

ioctl2 Commands

HIOCADDSUB

Add sub-partition

Library	*Introduced*	*Documentation last modified*
hdd	2.2.2	April 16, 2001

Syntax

arg	Pointer to the partition size string.
arglen	Size of *arg*.
bufp	Reserved. Specify NULL.
buflen	Size of *bufp*.

Description

Adds a sub-partition.

Example:

```
char chsize[] = "128M";
ioctl2(fd, HIOCADDSUB, chsize, strlen(chsize)+1, NULL, 0);
```

Return value

0 if successful.

-1 times errno if an error occurred.

EACCES	No access rights.
EFBIG	Already reached number of sub-partitions that can be added.
EIO	I/O error.
ENOMEM	Not enough free memory.
ENOSPC	No free space.

HIOCDELSUB

Delete sub-partition

Library	*Introduced*	*Documentation last modified*
hdd	2.2.2	April 16, 2001

Syntax

arg	Reserved. Specify NULL.
arglen	Size of *arg.*
bufp	Reserved. Specify NULL.
buflen	Size of *bufp.*

Description

Deletes a sub-partition. The sub-partition that was added last is deleted. If a filesystem has already been created in this partition and if a sub-partition is deleted without first reducing the size of the filesystem, then the filesystem will be destroyed. Usually, the application is not required to perform this operation directly.

Return value

Returns 0 on success.

-1 times errno if an error occurred.

EACCES	No access rights.
EIO	I/O error.
ENOENT	Partition to be deleted does not exist.
ENOMEM	Not enough free memory.

HIOCFLUSH

Flush the disk cache

Library	*Introduced*	*Documentation last modified*
hdd	2.2.2	April 16, 2001

Syntax

arg	Reserved. Specify NULL.
arglen	Size of *arg*.
bufp	Reserved. Specify NULL.
buflen	Size of *bufp*.

Description

Flushes the cache on the disk. Usually, the application is not required to perform this operation as the disk cache is flushed appropriately by the filesystem.

Return value

0 if successful.

HIOCNSUB

Get number of sub-partitions

Library	*Introduced*	*Documentation last modified*
hdd	2.2.2	April 16, 2001

Syntax

arg	Reserved. Specify NULL.
arglen	Size of *arg*.
bufp	Reserved. Specify NULL.
buflen	Size of *bufp*.

Description

Gets number of sub-partitions that were added.

Return value

Returns number of sub-partitions on success.

Chapter 4: PlayStation File System (for EE)
Table of Contents

Structures

sce_dirent

Directory entry

Library	*Introduced*	*Documentation last modified*
pfs	2.2.2	April 16, 2001

Structure

struct sce_dirent {	
struct sce_stat *d_stat;*	File status
char *d_name***[256];**	Filename
void **d_private***};**	Reserved

Description

This structure stores a directory entry.

See also

sceDread()

sce_stat

File status

Library	*Introduced*	*Documentation last modified*
pfs	2.2.2	April 16, 2001

Structure

struct sce_stat {

unsigned int *st_mode;*	File mode
	bit 0 Execute permission (other)
	bit 1 Write permission (other)
	bit 2 Read permission (other)
	bit 3 Execute permission (group)
	bit 4 Write permission (group)
	bit 5 Read permission (group)
	bit 6 Execute permission (user)
	bit 7 Write permission (user)
	bit 8 Read permission (user)
	bit 9 Reserved
	bit10 Reserved
	bit11 Reserved
	bit12-15 File type
	1 Directory
	2 Normal file
	4 Symbolic link
unsigned int *st_attr;*	Flag compatible with memory card mode
unsigned int *st_size;*	File size (64 bit)
unsigned char *st_ctime***[8];**	Creation time
unsigned char *st_atime***[8];**	This field is updated at the same time as last access time and last update time.
unsigned char *st_mtime***[8];**	Last update time
	byte0 Reserved
	byte1 Seconds
	byte2 Minutes
	byte3 Hours
	byte4 Day
	byte5 Month
	byte6-7 (4 digits)
unsigned int *st_hisize;*	
unsigned int *st_private***[6]};**	word0 uid
	word1 gid
	word2 Number of zones used by the file

Description

This structure stores file status.

See also

struct sce_dirent

sceGetstat()

Functions

sceChdir

Change current directory

Library	*Introduced*	*Documentation last modified*
pfs	2.2.2	April 16, 2001

Syntax

#include <sifdev.h>

int sceChdir(

const char **name***)**	File path name

Calling conditions

Can be called from a thread

Multithread safe (must be called in interrupt-enabled state)

Description

Changes current directory.

Return value

Returns zero on success. If an error occurred, returns -1 times errno.

EACCES	No access permission.
EIO	I/O error.
ELOOP	Too many symbolic links encountered when resolving the path name.
EMFILE	Reached the maximum number of descriptors that can be opened.
ENAMETOOLONG	File path name is too long.
ENODEV	Specified device does not exist.
ENOENT	Specified file does not exist.
ENOMEM	Not enough free memory.
ENOTDIR	*name* is not a directory.

sceChstat

Change status of file/directory

Library	*Introduced*	*Documentation last modified*
pfs	2.2.2	April 16, 2001

Syntax

#include <sifdev.h>

int sceChstat(

const char **name,*	File path name (including device name + ':')
struct sce_stat **buf,*	Buffer for storing the status
unsigned int *cbit***)**	Macro specifying the field to be changed. Any of the following constants can be specified. SCE_CST_MODE SCE_CST_ATTR SCE_CST_SIZE SCE_CST_CT SCE_CST_AT SCE_CST_MT SCE_CST_PRVT

Calling conditions

Can be called from a thread

Multithread safe (must be called in interrupt-enabled state)

Description

Changes the status of the specified file/directory. The members of the sce_stat structure that can be changed by this function are: bits except for the file type of the file mode and each time, bits except for the subdirectory bit of the memory card compatibility flag and the close completion flag.

Return value

Returns zero on success. If an error occurred, returns -1 times errno.

EACCES	No access permission.
EIO	I/O error.
ELOOP	Too many symbolic links encountered when resolving the path name.
EMFILE	Reached the maximum number of descriptors that can be opened.
ENAMETOOLONG	File path name is too long.
ENODEV	Specified device does not exist.
ENOENT	Specified file does not exist.
ENOMEM	Not enough free memory.
EROFS	Write access was requested for a file from a read-only filesystem.

sceClose

Close file

Library	*Introduced*	*Documentation last modified*
pfs	2.2.2	April 16, 2001

Syntax

#include <sifdev.h>

int sceClose(

int *fd***)**	Previously opened file descriptor

Calling conditions

Can be called from a thread

Multithread safe (must be called in interrupt-enabled state)

Description

Closes an open file and frees the file descriptor.

Return value

Returns 0 on success. On error, returns -1 times errno.

EBADF	*fd* is not a valid open descriptor.

sceDclose

Close directory

Library	*Introduced*	*Documentation last modified*
pfs	2.2.2	April 16, 2001

Syntax

#include <sifdev.h>

int sceDclose(

int *fd***)**	File descriptor

Calling conditions

Can be called from a thread

Multithread safe (must be called in interrupt-enabled state)

Description

Closes an open directory and frees the file descriptor.

Return value

Returns zero on success. On error returns -1 times errno.

EBADF	*fd* is not a valid open descriptor.

sceDevctl

Special operations for a device

Library	*Introduced*	*Documentation last modified*
pfs	2.2.2	April 16, 2001

Syntax

#include <sifdev.h>

int sceDevctl(

const char **name,*	Filesystem device name
int *cmd,*	Operation command. Any of the following constants can be specified. PDIOC_ZONESZ PDIOC_ZONEFREE PDIOC_CLOSEALL
void **arg,*	Command arguments. Depends upon *cmd*.
unsigned int *arglen,*	Size of *arg*
void **bufp,*	Arguments received from command. Depends upon *cmd*.
unsigned int *buflen***)**	Size of *bufp*

Calling conditions

Can be called from a thread

Multithread safe (must be called in interrupt-enabled state)

Description

Performs special operations for a device. For details regarding each of the commands, refer to the "devctl command list."

Return value

If successful, returns a command-dependent value.

If an error occured, returns -1 times errno.

The errors that are common to each of the commands are as follows.

EINVAL	A non-existent *cmd* was specified.
EMFILE	Reached the maximum number of descriptors that can be opened.
ENODEV	Specified device does not exist.

sceDopen

Open a directory

Library	*Introduced*	*Documentation last modified*
pfs	2.2.2	April 16, 2001

Syntax

#include <sifdev.h>

int sceDopen(

const char **name***)**	Directory path name(including device name + ':')

Calling conditions

Can be called from a thread

Multithread safe (must be called in interrupt-enabled state)

Description

Opens a directory. Assigns a file descriptor to the open directory. Directory path name is "pfs" + unit number + ':' + character string.

Return value

Returns file descriptor on normal completion (value > 0). Returns -1 times errno if an error occurred.

EACCES	No access permission.
EIO	I/O error.
ELOOP	Too many symbolic links encountered when resolving the path name.
EMFILE	Reached maximum number of descriptors that can be opened.
ENAMETOOLONG	File path name is too long.
ENODEV	Specified device does not exist.
ENOENT	Specified directory not found.
ENOMEM	Not enough free memory.
ENOTDIR	Specified file is not a directory.

sceDread

Read directory entry

Library	*Introduced*	*Documentation last modified*
pfs	2.2.2	April 16, 2001

Syntax

#include <sifdev.h>

int sceDread(

int *fd,*	File descriptor
struct sce_dirent ******buf***)**	Address of the buffer that stores the data that was read.

Calling conditions

Can be called from a thread

Multithread safe (must be called in interrupt-enabled state)

Description

The next entry from the directory entry stream indicated by *fd* is copied to the sce_dirent structure *buf*. Returns 0 when the end of entries is reached.

Return value

Returns the length of the filename on success. Returns 0 when the end of entries is reached.

Returns -1 times errno if an error occurred.

EBADF	*fd* is not a valid open descriptor.
EIO	I/O error.
ENOMEM	Not enough free memory.
ENOTDIR	*fd* is not a descriptor for a directory.

sceFormat

Format filesystem

Library	*Introduced*	*Documentation last modified*
pfs	2.2.2	July 2, 2001

Syntax

#include <sifdev.h>

int sceFormat(

const char **devname,*	Filesystem device name (pfs:)
const char **blockdevname,*	Block device name of partition created in advance. (Example: 'hdd0:BISLPS-XXXXXX,fpwd")
void **arg,*	Pointer to zone size variable and fragment option.
int *arglen***)**	Size of *arg*.

Calling conditions

Can be called from a thread

Multithread safe (must be called in interrupt-enabled state)

Description

Builds a new filesystem. The specified zone size must be a power of 2 (2 ^ n) and in the range 2K - 128K. Efficiency will be improved if the zone size is set to a smaller value if most of the files to be created are small, and to a larger value if most of the files to be created are large.

Example:

```
int zonesz = 8192;
sceFormat("pfs:", "hdd0:BISLPS-XXXXXX,fpwd", &zonesz, sizeof(int));
```

In addition, formatting can also be performed if fragmentation was intentionally done for verification purposes during development.

```
int arg[3];
arg[0] = 8192;              // zone size
arg[1] = 0x00002d66;        // -f
arg[2] = 0x01030f0f;        // fragment bit pattern
sceFormat("pfs:", "hdd0:test", &arg, sizeof(arg));
```

Each bit of the bit pattern corresponds to a zone. For example, if 0x0f0f0f0f is specified, formatting will be performed with a repeated pattern in which four zones are used and four zones are empty.

Notes

Note that when this operation is performed, a filesystem previously created on the specified partition will be initialized.

Return value

On success, returns 0. On error, returns -1 times errno.

EACCES	No access permission.
EBUSY	Specified partition is already open.
EINVAL	An invalid argument was specified.
EIO	I/O error.

EMFILE	Reached the maximum number of descriptors that can be opened.
ENODEV	Specified device does not exist.
ENOENT	Specified partition does not exist.
ENOMEM	Not enough free memory.
ENXIO	Not a supported device.

sceGetstat

Get file/directory status

Library	*Introduced*	*Documentation last modified*
pfs	2.2.2	April 16, 2001

Syntax

#include <sifdev.h>

int sceGetstat(

const char **name,*	File path name (including device name + ':')
struct sce_stat **buf***)**	Buffer for storing the status

Calling conditions

Can be called from a thread

Multithread safe (must be called in interrupt-enabled state)

Description

Copies file information to the sce_dirent structure *buf*. The file path name is "pfs" + unit number + ':' + character string.

Return value

On success, returns zero. On error, returns -1 times errno.

EACCES	No access permission.
EIO	I/O error.
ELOOP	Too many symbolic links encountered when resolving the path name.
EMFILE	Reached the maximum number of descriptors that can be opened.
ENAMETOOLONG	File path name is too long.
ENODEV	Specified device does not exist.
ENOENT	Specified file does not exist.
ENOMEM	Not enough free memory.

sceIoctl2

Special operations for a file

Library	*Introduced*	*Documentation last modified*
pfs	2.2.2	April 16, 2001

Syntax

#include <sifdev.h>

int sceIoctl2(

int *fd,*	Target file descriptor
int *cmd,*	Operation command. Any of the following constants can be specified. PIOCALLOC PIOCFREE PIOCATTRADD PIOCATTRDEL PIOCATTRLOOKUP PIOCATTRREAD
void **arg,*	Command arguments. Depends on *cmd*.
unsigned int *arglen,*	Size of *arg.*
void **bufp,*	Arguments received from the command. Depends on *cmd*.
unsigned int *buflen***)**	Size of the *bufp*.

Calling conditions

Can be called from a thread

Multithread safe (must be called in interrupt-enabled state)

Description

Performs special operations on a partition. For details regarding each of the commands, refer to the "ioctl2 command list."

Return value

Returns a command-dependent value if successful.

-1 times errno if an error occurred.

The errors that are common to each of the commands are as follows.

EBADF	*fd* is not a valid open descriptor.
EINVAL	Specified command not found.

sceLseek

Move file pointer

Library	*Introduced*	*Documentation last modified*
pfs	2.2.2	April 16, 2001

Syntax

#include <sifdev.h>

int sceLseek(

int *fd,*	File descriptor
long *offset,*	Distance to move pointer
int *whence***)**	Reference position of *offset* in the extended attribute area of the partition. Any of the following constants can be specified. SEEK_SET Starting position SEEK_CUR Current position SEEK_END End

Calling conditions

Can be called from a thread

Multithread safe (must be called in interrupt-enabled state)

Description

The offset of file descriptor *fd* is changed to the position specified by the *offset* argument, and according to *whence*. The offset cannot be set to a position exceeding the end of file.

Return value

On success, returns the new setting of the file pointer.

On error, returns -1 times errno.

EBADF	*fd* is not a valid open descriptor.
EINVAL	whence is an incorrect value or an offset beyond the EOF was specified.
EIO	I/O error.
EISDIR	The request was made for a directory.

sceLseek64

Move file pointer (64-bit compatible)

Library	*Introduced*	*Documentation last modified*
pfs	2.2.2	April 16, 2001

Syntax

#include <sifdev.h>

int sceLseek64(

int *fd,*	File descriptor
long *offset,*	Distance to move pointer
int *whence***)**	Reference position of offset in the extended attribute area of the partition. Any of the following constants can be specified:

SEEK_SET	Starting position
SEEK_CUR	Current position
SEEK_END	End

Calling conditions

Can be called from a thread

Multithread safe (must be called in interrupt-enabled state)

Description

The offset of file descriptor *fd* is changed to the position specified by the offset argument, and according to *whence*. The offset cannot be set to a position exceeding the end of file. This function supports a 64-bit file size.

Notes

When using this function, the –fno-strict-aliasing option must be specified at compiling.

Return value

On success, returns the new setting of the file pointer.

On error, returns -1 times errno.

EBADF	*fd* is not a valid open descriptor.
EINVAL	whence is an incorrect value or an offset beyond the EOF was specified.
EIO	I/O error.
EISDIR	The request was made for a directory.

sceMkdir

Create directory

Library	*Introduced*	*Documentation last modified*
pfs	2.2.2	April 16, 2001

Syntax

#include <sifdev.h>

int sceMkdir(

const char **name,*	Directory path name (including device name + ':')
int *mode***)**	File mode

bit 0	Execute permission (other)
bit 1	Write permission (other)
bit 2	Read permission (other)
bit 3	Execute permission (group)
bit 4	Write permission (group)
bit 5	Read permission (group)
bit 6	Execute permission (user)
bit 7	Write permission (user)
bit 8	Read permission (user)
bit 9	Reserved
bit10	Reserved
bit11	Reserved

Macros for each mode are provided in sifdev.h. However, using octal codes such as 0777, 0755, etc. is also an easy way to specify the mode.

Calling conditions

Can be called from a thread

Multithread safe (must be called in interrupt-enabled state)

Description

Creates a directory. The path name is "pfs" + unit number + ':' + character string.

Notes

If the *mode* is not properly specified when the directory is created, it might not be possible to access the directory. In the current library, umask, uid and gid cannot be changed. The value of umask is fixed at 0002 and the values of uid and gid are fixed at 0xffff.

Return value

Returns zero on success. On error returns -1 times errno.

EACCES	No access permission.
EEXIST	File already exists.
EIO	I/O error.
ELOOP	Too many symbolic links encountered when resolving the path name.
EMFILE	Reached the maximum number of descriptors that can be opened.

ENAMETOOLONG	File path name is too long.
ENODEV	Specified device does not exist.
ENOENT	Directory not found in the specified path.
ENOMEM	Not enough free memory.
ENOSPC	No free space.
EROFS	Write access was requested for a file from a read-only filesystem.

sceMount

Mount device

Library	*Introduced*	*Documentation last modified*
pfs	2.2.2	April 16, 2001

Syntax

#include <sifdev.h>

int sceMount(

const char **fsname,*	Character string which specifies filesystem device name and unit number after mounting.
const char **devname,*	Character string which identifies the required device that will be used to open the block device to be mounted.
int *flag,*	Mount flag. Any of the following constants can be specified. For multiple specifications, take the logical OR. SCE_MT_RDWR Mount as read/write enabled. SCE_MT_RDONLY Mount as read-only. SCE_MT_ROBUST Mount in ROBUST node. SCE_MT_ERRCHECK Set an error if there is anything abnormal in the filesystem when mounting.
void **arg,*	Reserved
int *arglen***)**	Size of *arg*

Calling conditions

Can be called from a thread

Multithread safe (must be called in interrupt-enabled state)

Description

Mounts the block device specified by *devname* on the filesystem logical device specified by *fsname*.

devname usually specifies a string that identifies a previously created partition. If the mount is done read-only, providing only a read-only password is acceptable, but to allow mounting for read/write, a full password is required.

When SCE_MT_ROBUST is specified, filesystem information will always be updated. When any operation that causes a change to the filesystem (such as sceMkdir(), sceWrite()) is performed, it is immediately synchronized with the disk. Furthermore, updating of the close completion flag for memory card compatibility is only performed in ROBUST mode.

When SCE_MT_ERRCHECK is specified and an abnormality occurs in the filesystem, an EIO error will be returned. When an abnormality is seen in the filesystem, a prompt filesystem check is recommended. Even with an abnormality in the filesystem, the trouble-free portion of the filesystem is still readable, provided that the filesystem is not updated. However, writing should not be performed because it may worsen the problem.

Examples:

```
sceMount("pfs0:", "hdd0:tst1,fpwd1", SCE_MT_RDWR, NULL, 0);
sceMount("pfs1:", "hdd0:tst2,fpwd2", SCE_MT_RDWR|SCE_MT_ROBUST, NULL, 0);
sceMount("pfs2:", "hdd0:tst3,,rpwd3", SCE_MT_RDONLY, NULL, 0);
```

Return value

Returns zero on success. On error returns -1 times errno.

EACCES	No access permission.
EBUSY	The specified partition is already open.
EINVAL	An invalid argument was specified.
EIO	I/O error.
EMFILE	Reached the maximum number of descriptors that can be opened.
ENODEV	Specified device does not exist.
ENOENT	Specified partition not found.
ENOMEM	Not enough free memory.
ENXIO	Not a supported device.

sceOpen

Create, open file

Library	*Introduced*	*Documentation last modified*
pfs	2.2.2	April 16, 2001

Syntax

#include <sifdev.h>

int sceOpen(

const char **name,*	File path name (including device name + ':')
int *flags,*	Access mode. Any of the following constants can be specified. For multiple specifications, take the logical OR. SCE_RDONLY Open as read only SCE_WRONLY Open as write only SCE_RDWR Open for read/write SCE_APPEND Always perform writes at the end of file SCE_CREAT Create a new file if the file does not exist SCE_TRUNC Discard previous file contents SCE_EXCL When specified with SCE_CREAT, if a file exists with the same name, an error will occur
unsigned short *mode***)**	File mode bit 0 Execute permission (other) bit 1 Write permission (other) bit 2 Read permission (other) bit 3 Execute permission (group) bit 4 Write permission (group) bit 5 Read permission (group) bit 6 Execute permission (user) bit 7 Write permission (user) bit 8 Read permission (user) bit 9 Reserved bit10 Reserved bit11 Reserved Macros for each mode are provided in sifdev.h. However, using octal codes such as 0777, 0755, etc. is also an easy way to specify the mode.

Calling conditions

Can be called from a thread

Multithread safe (must be called in interrupt-enabled state)

Description

Creates, opens a file. Assigns a file descriptor to the file that was opened. The file path name is "pfs" + unit number + ':' + character string.

Example:

```
sceOpen("pfs0:/foo", SCE_CREAT | SCE_RDWR, SCE_STM_RWXUGO);
```

Notes

If the *mode* was not properly specified when the file was created, it may not be possible to open the file. In the current library, umask, uid and gid cannot be changed. The value of umask is fixed at 0002 and the values of uid and gid are fixed at 0xffff.

Return value

Returns the file descriptor on normal completion (value > 0).

-1 times errno if an error occurred.

EACCES	No access permission.
EEXIST	Both SCE_CREAT and SCE_EXCL were specified and the file already exists.
EINVAL	An invalid argument was specified.
EIO	I/O error.
EISDIR	The file is a directory.
ELOOP	Too many symbolic links encountered when resolving the path name.
EMFILE	Reached the maximum number of descriptors that can be opened.
ENAMETOOLONG	File path name is too long.
ENODEV	Specified device does not exist.
ENOENT	Specified file does not exist.
ENOMEM	Not enough free memory.
ENOSPC	No free space.
EROFS	Write access was requested for a file from a read-only filesystem.

sceRead

Read file

Library	*Introduced*	*Documentation last modified*
pfs	2.2.2	April 16, 2001

Syntax

#include <sifdev.h>

int sceRead(

int *fd,*	File descriptor of the read target
void **buf,*	Address of the buffer that will store the read data
int *count*)	Read data size

Calling conditions

Can be called from a thread

Multithread safe (must be called in interrupt-enabled state)

Description

Reads a maximum of *count* bytes from the file that was previously opened, into the buffer starting from the address specified by *buf*.

Performance is lowered if the buffer is not 64-byte aligned, so 64-byte alignment is recommended. Performance will improve if count is a multiple of 512. It is recommended that a multiple of 512 be used as much as possible. Even if reading in 512-byte units is not possible, reads should be performed in at least 4-byte units (64-byte units if possible). To the degree that transfers are performed once in large units, performance will improve even more.

If an EIO | 0x10000 error occurs, either overwrite the file or delete it completely without performing a filesystem check.

Notes

SPR cannot be specified to *buf*.

Return value

On success, returns the number of bytes read. The file position is advanced by this amount only. A return value of 0 means end of file. If an error occurred, -1 times errno is returned.

EBADF	*fd* is not a valid open descriptor.
EINVAL	An invalid argument was specified.
EIO	I/O error.
EIO | 0x10000	Bad sector was found while reading the file contents.
ENOMEM	Not enough free memory.

sceReadlink

Read symbolic link value

Library	*Introduced*	*Documentation last modified*
pfs	2.2.2	April 16, 2001

Syntax

#include <sifdev.h>

int sceReadlink(

const char **path,*	File path name
char **buf,*	Buffer for writing contents
unsigned int *bufsiz***)**	Size of *buf (*up to 1023 bytes)

Calling conditions

Can be called from a thread

Multithread safe (must be called in interrupt-enabled state)

Description

Stores the contents of the symbolic link specified by path into *buf*. *bufsiz* specifies the size of *buf*. sceReadlink does not add null (NUL) characters to *buf*. If the buffer is too small to store the entire contents, the contents are truncated to fit into *bufsiz* bytes.

Return value

On success, returns the number of characters stored in the buffer. On error, returns -1 times errno.

EACCES	No access permission.
EEXIST	newname already exists.
EINVAL	Invalid argument was specified, or not a symbolic link.
EIO	I/O error.
ELOOP	Too many symbolic links encountered when resolving the path name.
EMFILE	Reached the maximum number of descriptors that can be opened.
ENAMETOOLONG	File path name is too long.
ENODEV	Specified device does not exist.
ENOENT	Specified file does not exist.
ENOMEM	Not enough free memory.

sceRemove

Delete file

Library	*Introduced*	*Documentation last modified*
pfs	2.2.2	April 16, 2001

Syntax

#include <sifdev.h>

int sceRemove(

const char ****name*)**	File path name (including device name + ':')

Calling conditions

Can be called from a thread

Multithread safe (must be called in interrupt-enabled state)

Description

Deletes the specified file. The file path name is "pfs" + unit number + ':' + character string.

Return value

Returns zero on success. On error returns -1 times errno.

EACCES	No access permission.
EBUSY	The file is open.
EIO	I/O error.
EISDIR	The file is a directory.
ELOOP	Too many symbolic links encountered when resolving the path name.
ENAMETOOLONG	File path name is too long
EMFILE	Reached the maximum number of descriptors that can be opened.
ENODEV	Specified device does not exist.
ENOENT	Specified file does not exist.
ENOMEM	Not enough free memory.
EROFS	Write access was requested for a file from a read-only filesystem.

sceRename

Change file/directory name

Library	*Introduced*	*Documentation last modified*
pfs	2.2.2	April 16, 2001

Syntax

#include <sifdev.h>	
int sceRename(	
const char **oldname,*	Name of file/directory before change
const char **newname*)	Name of file/directory after change

Calling conditions

Can be called from a thread

Multithread safe (must be called in interrupt-enabled state)

Description

Renames file. If required, performs movement between directories. If *newname* already exists, it is automatically replaced if the following conditions are met.

- *oldname* is a file and *newname* is also a file.
- *oldname* is a directory and *newname* is also a directory.
- *newname* is a directory and it is empty.
- *newname* is not open.

If *newname* exists, it is guaranteed that the original *newname* will remain unchanged even if the operation fails for any reason.

Return value

Returns zero on success. On error returns -1 times errno.

EACCES	No access permission.
EBUSY	The file is open or it is a working directory.
EINVAL	"." or ".." was specified, or newname includes part of the path of oldname. In other words, tried to change a directory into its own subdirectory.
EIO	I/O error.
EISDIR	oldname is a file and newname is a directory.
ELOOP	Too many symbolic links encountered when resolving the path name.
EMFILE	Reached the maximum number of descriptors that can be opened.
ENAMETOOLONG	File path name is too long.
ENODEV	Specified device does not exist.
ENOENT	Specified file does not exist.
ENOMEM	Not enough free memory.
ENOSPC	No free space.
ENOTDIR	oldname is directory but newname is not a directory.
ENOTVACANT	newname is a directory but the directory is not empty.
EROFS	Write access was requested for a file from a read-only filesystem.

sceRmdir

Delete directory

Library	*Introduced*	*Documentation last modified*
pfs	2.2.2	April 16, 2001

Syntax

#include <sifdev.h>

int sceRmdir(

const char **name*)	Directory path name (including device name + ':')

Calling conditions

Can be called from a thread

Multithread safe (must be called in interrupt-enabled state)

Description

Deletes the specified directory. The directory to be deleted must be empty. Directory path name is "pfs" + unit number + ':' + character string.

Return value

Returns zero on success. On error, returns -1 times errno.

EACCES	No access permission.
EBUSY	Directory is open or it is a working directory.
EIO	I/O error.
ELOOP	Too many symbolic links encountered when resolving the path name.
EMFILE	Reached the maximum number of descriptors that can be opened.
ENAMETOOLONG	File path name is too long.
ENODEV	Specified device does not exist.
ENOENT	Specified directory not found.
ENOMEM	Not enough free memory.
ENOTDIR	Specified file is not a directory.
ENOTVACANT	Directory is not empty.
EROFS	Write access was requested for a file from a read-only filesystem.

sceSymlink

Create symbolic link

Library	*Introduced*	*Documentation last modified*
pfs	2.2.2	April 16, 2001

Syntax

#include <sifdev.h>

int sceSymlink(

const char **oldname,*	Original filename
const char **newname*)	New filename

Calling conditions

Can be called from a thread

Multithread safe (must be called in interrupt-enabled state)

Description

Creates a symbolic link named *newname* to *oldname*. The symbolic link is interpreted during execution when locating files or directories, by following its contents and replacing them. A symbolic link might specify an existing file, or a file which does not exist. "..." may be included in the path. If *newname* already exists, it will not be replaced.

Return value

Returns zero on success. On error, returns -1 times errno.

EACCES	No access permission.
EEXIST	*newname* already exists.
EINVAL	Invalid argument was specified.
EIO	I/O error.
ELOOP	Too many symbolic links encountered when resolving the path name.
EMFILE	Reached the maximum number of descriptors that can be opened.
ENAMETOOLONG	File path name is too long.
ENODEV	Specified device does not exist.
ENOENT	Specified file does not exist.
ENOMEM	Not enough free memory.
ENOSPC	No free space.
EROFS	Write access was requested for a file from a read-only filesystem.

sceSync

Synchronize buffer cache and disk

Library	*Introduced*	*Documentation last modified*
pfs	2.2.2	April 16, 2001

Syntax

#include <sifdev.h>

int sceSync(

const char **name,*	Device name
int *flag*)	Flag (Reserved, not used)

Calling conditions

Can be called from a thread

Multithread safe (must be called in interrupt-enabled state)

Description

To avoid reading and writing to a slow disk, the filesystem keeps data in memory. This function flushes the contents of the filesystem's buffer cache in this memory to the disk. Flushing also includes the cache on the disk as well.

Return value

Returns zero on success. On error returns -1 times errno.

EIO	I/O error.
ENODEV	Specified device does not exist.

sceUmount

Unmount filesystem

Library	*Introduced*	*Documentation last modified*
pfs	2.2.2	April 16, 2001

Syntax

#include <sifdev.h>

int sceUmount(

const char **fsname*)	Character string specifying filesystem device name and unit number during mounting.

Calling conditions

Can be called from a thread

Multithread safe (must be called in interrupt-enabled state)

Description

Unmounts the filesystem. The contents of the buffer cache in memory are flushed.

Return value

Returns zero on success. On error returns -1 times errno.

EBUSY	File is open.
EMFILE	Reached the maximum number of descriptors that can be opened.
ENODEV	Specified device does not exist.

sceWrite

Write to file

Library	*Introduced*	*Documentation last modified*
pfs	2.2.2	April 16, 2001

Syntax

#include <sifdev.h>

int sceWrite(

int *fd,*	File descriptor of the file to be written.
const void **buf,*	Address that stores the data to be written.
int *count***)**	Size of the data to be written.

Calling conditions

Can be called from a thread

Multithread safe (must be called in interrupt-enabled state)

Description

Writes a maximum of count bytes from the buffer indicated by buf into the file referenced by the file descriptor *fd*. Performance is lowered if the buffer is not 64-byte aligned, so 64-byte alignment is recommended. Performance will improve if count is a multiple of 512. It is recommended that a multiple of 512 be used as much as possible. Even if reading in 512-byte units is not possible, reads should be performed in at least 4-byte units (64-byte units if possible). To the degree that transfers are performed once in large units, performance will improve even more.

If an EIO|0x10000 error occurs, delete the file without performing a filesystem check.

Notes

SPR cannot be specified to *buf*.

Return value

On success, returns the number of bytes written. The file position is advanced by this amount only.

If an error occurred, -1 times errno is returned.

EBADF	*fd* is not a valid open descriptor.
EINVAL	Invalid argument was specified.
EIO	I/O error.
EIO |0x10000	Bad sector was found while writing the file contents.
ENOMEM	Not enough free memory.
ENOSPC	No free space.

devctl Commands

PDIOC_CLOSEALL

Close all files

Library	*Introduced*	*Documentation last modified*
pfs	2.2.2	April 16, 2001

Arguments

arg	Reserved. Specify NULL.
arglen	Size of *arg.*
bufp	Reserved. Specify NULL.
buflen	Size of *bufp.*

Description

Closes all files on all mounted filesystems. However, file descriptors do not get freed, so use this function only when powering off, etc.

Return value

Returns 0.

PDIOC_CLRFSCKSTAT

Clear FSCK status

Library	*Introduced*	*Documentation last modified*
pfs	2.3	July 2, 2001

Arguments

arg	Reserved. Specify NULL.
arglen	Size of *arg.*
bufp	Reserved. Specify NULL.
buflen	Size of *bufp.*

Description

This command clears the state of the filesystem that was updated by fsck.

Return value

0 if processing succeeds.

PDIOC_GETFSCKSTAT

Check FSCK status

Library	*Introduced*	*Documentation last modified*
pfs	2.3	July 2, 2001

Arguments

arg	Reserved. Specify NULL.
arglen	Size of *arg.*
bufp	Reserved. Specify NULL.
buflen	Size of *bufp.*

Description

This command returns 1 only when a problem of some kind was found in the filesystem and the filesystem state was updated. Once this state occurs, it is held until cleared with PDIOC_CLRFSCKSTAT.

Return value

If fsck had previously corrected a problem in the filesystem (i.e. the filesystem state was updated), 1 is returned.

Otherwise, 0 is returned.

PDIOC_ZONEFREE

Get free zones

Library	*Introduced*	*Documentation last modified*
pfs	2.2.2	April 16, 2001

Arguments

arg	Reserved. Specify NULL.
arglen	Size of *arg.*
bufp	Reserved. Specify NULL.
buflen	Size of *bufp.*

Description

Gets the number of available free zones.

Return value

The number of free zones is returned as an unsigned 32 bit integer.

PDIOC_ZONESZ

Get zone size

Library	*Introduced*	*Documentation last modified*
pfs	2.2.2	April 16, 2001

Arguments

arg	Reserved. Specify NULL.
arglen	Size of *arg.*
bufp	Reserved. Specify NULL.
buflen	Size of *bufp.*

Description

Gets the zone size.

Return value

Returns the zone size.

ioctl2 Commands

PIOCALLOC

Allocate area

Library	*Introduced*	*Documentation last modified*
pfs	2.2.2	April 16, 2001

Arguments

arg	Number of allocated zones.
arglen	Size of *arg.*
bufp	Reserved. Specify NULL.
buflen	Size of *bufp.*

Description

Allocates an area that can be used by a file. In pfs, the speed of allocating an area for a new file is not very high. When the file size is approximately known and a large amount of data is to be continuously written to a file, write performance will be improved if the area is allocated in advance before writing.

Example:

```
u_int size = 1024*1024;
sceIoctl2(fd, PIOCALLOC, &size, sizeof(int), NULL, 0);
```

Return value

Returns zero on success. On error returns -1 times errno.

EACCES	No access permission.
EINVAL	Invalid argument was specified.
EIO	I/O error.
ENOMEM	Not enough free memory.
ENOSPC	No free space.

PIOCATTRADD

Add extended file attribute entry

Library	*Introduced*	*Documentation last modified*
pfs	2.2.2	April 16, 2001

Arguments

arg	Buffer that stores a 256 byte key and a 256 byte value.
arglen	Size of *arg.*
bufp	Reserved. Specify NULL.
buflen	Size of *bufp.*

Description

Adds an entry to the extended file attribute area.

Example:

```
struct {
        char key[0x100];
        char value[0x100];
} attr;
strcpy(key, "application");
strcpy(value, "x-compressed");
sceIoctl2(fd, PIOCATTRADD, &attr, 0x100*2, NULL, 0);
```

Return value

Returns zero on success. On error returns -1 times errno.

EACCES	No access permission.
EEXIST	Specified key already exists.
EINVAL	Invalid argument was specified.
EIO	I/O error.
ENOMEM	Not enough free memory.
ENOSPC	No free space.

PIOCATTRDEL

Delete extended file attribute entry

Library	*Introduced*	*Documentation last modified*
pfs	2.2.2	April 16, 2001

Arguments

arg	Buffer that stores the key string.
arglen	Size of *arg.*
bufp	Reserved. Specify NULL.
buflen	Size of *bufp.*

Description

Deletes an entry from the extended file attribute area.

Example:

```
char key[] = "application";
sceIoctl2(fd, PIOCATTRDEL, key, strlen(key)+1, NULL, 0);
```

Return value

Returns zero on success. On error returns -1 times errno.

EACCES	No access permission.
EIO	I/O error.
ENOENT	Entry not found.
ENOMEM	Not enough free memory.

PIOCATTRLOOKUP

Lookup extended file attribute entry

Library	*Introduced*	*Documentation last modified*
pfs	2.2.2	April 16, 2001

Arguments

arg	Buffer that stores the key string.
arglen	Size of *arg.*
bufp	Buffer that will store the value.
buflen	Size of *bufp.*

Description

Searches for the specified key from the extended file attribute area and stores the value in *bufp*.

Example:

```
char key[] = "application";
char value[0x100];
sceIoctl2(fd, PIOCATTRLOOKUP, key, strlen(key)+1, value, 0x100);
```

Return value

Returns zero on success.

On error, returns -1 times errno.

EIO	I/O error.
ENOENT	Entry not found.
ENOMEM	Not enough free memory.

PIOCATTRREAD

Read extended file attribute entry

Library	*Introduced*	*Documentation last modified*
pfs	2.2.2	April 16, 2001

Arguments

arg	Reserved. Specify NULL.
arglen	Size of *arg.*
bufp	512 byte buffer that will store the key and value
buflen	Size of *bufp.*

Description

Copies the next entry from the attribute entry stream into *bufp*. Returns 0 when reaches the end of entries.

Example:

```
struct {
        char key[0x100];
        char value[0x100];
} attr;
while ((r = sceIoctl2(fd, PIOCATTRREAD, NULL, 0, &attr, 0)) > 0)
        printf("%s/%s\n", attr.key, attr.value);
```

Return value

On success, returns length of key. Returns zero if reaches the end of entries.

On error, returns -1 times errno.

EIO	I/O error.
ENOMEM	Not enough free memory.

PIOCFREE

Free area

Library	*Introduced*	*Documentation last modified*
pfs	2.2.2	April 16, 2001

Arguments

arg	Reserved. Specify NULL.
arglen	Size of *arg.*
bufp	Reserved. Specify NULL.
buflen	Size of *bufp.*

Description

Frees areas not being used by files.

Return value

Returns zero on success. On error returns -1 times errno.

EACCES	No access permission.
EINVAL	Invalid argument was specified.
EIO	I/O error.
ENOMEM	Not enough free memory.
ENOSPC	No free space.

Chapter 5: Memory Card Library
Table of Contents

Structures

sceMcIconSys

Structure for generating icon.sys

Library	*Introduced*	*Documentation last modified*
libmc	1.4	July 24, 2000

Structure

typedef struct {	
unsigned char *Head[4];*	'P','S','2','D'
unsigned short *Reserv1;*	Reserved area, must be filled entirely with 00.
unsigned short *OffsLF;*	Line break position in title name
unsigned *Reserv2;*	Reserved area, must be filled entirely with 00.
unsigned *TransRate;*	Background transparency
_iconVu0IVECTOR *BgColor[4];*	Background color (4 points)
_iconVu0FVECTOR *LightDir[3];*	Light source direction (3 light sources)
_iconVu0FVECTOR *LightColor[3];*	Light source color (3 light sources)
_iconVu0FVECTOR *Ambient;*	Ambient light
unsigned char *TitleName[68];*	Title name
unsigned char *FnameView[64];*	List icon file name
unsigned char *FnameCopy[64];*	Copy icon file name
unsigned char *FnameDel[64];*	Deletion icon file name
unsigned char *Reserve3[512];*	Reserved area, must be filled entirely with 00.
} sceMcIconSys;	

Note: _iconVu0IVECTOR structure is the same as that of sceVu0IVECTOR in libvu0.h, without an alignment declaration. _iconVu0FVECTOR structure is the same as that of sceVu0FVECTOR in libvu0.h, without an alignment declaration.

Description

This structure generates icon.sys.

sceMcStDateTime

Date/time structure

Library	*Introduced*	*Documentation last modified*
libmc	2.4.3	January 4, 2002

Structure

typedef struct {	
unsigned char *Resv2;*	Reserved area
unsigned char *Sec;*	Second
unsigned char *Min;*	Minute
unsigned char *Hour;*	Hour
unsigned char *Day;*	Day
unsigned char *Month;*	Month
unsigned short *Year;*	Year
} sceMcStDateTime;	

Description

This structure manages the date/time information.

sceMcTblGetDir

Structure for storing file list results

Library	*Introduced*	*Documentation last modified*
libmc	1.1	January 4, 2002

Structure

typedef struct {	
sceStDateTime_*Create*;	Entry creation date/time
sceStDateTime_*Modify*;	Entry modification date/time
unsigned *FileSizeByte;*	File size (bytes). For a directory entry: 0
unsigned short *AttrFile;*	File attribute
unsigned short Reserve1;	
unsigned Reserve2;	
unsigned *PdaAplNo;*	Application number to be passed to sceMcxSetInfo() when a PDA application is executed (valid only in a PDA file)
unsigned char *EntryName*[32];	Entry name
} sceMcTblGetDir __attribute__((aligned (64)));	

Description

This structure stores file list results.

Functions

sceMcChangeThreadPriority

Change IOP module (mcserv.irx) thread priority

Library	*Introduced*	*Documentation last modified*
libmc	2.0	July 2, 2001

Syntax

int sceMcChangeThreadPriority(	
int *level***)**	Thread priority

Calling conditions

Can be called from a thread

Multithread safe

Description

Changes the thread priority of mcserv.irx, the IOP module of the PDA library. Possible priority settings are values in the range of USER_HIGHEST_PRIORITY - USER_LOWEST_PRIORITY (inclusive), as defined in thread.h. The initial thread priority value for mcserv.irx is 104.

Table 5-1: Return value in result of sceMcSync()

(This macro is defined in kerror.h)

Value	Macro	Result
0	KE_OK	Success
-403	KE_ILLEGAL_PRIORITY	Thread priority value exceeds valid range

Notes

The thread priority can also be set using sceSifLoadModule() when mcserv.irx is loaded. For example, if mcserv.irx is loaded as shown below, the initial thread priority will be 100. The thread priority string should be expressed as a decimal value.

```
unsigned char *param = "thpri=100";
sceSifLoadModule( "host0:/usr/local/sce/iop/modules/mcserv.irx", strlen(param)+1, param);
```

Return value

Table 5-2

Value	Macro	Result
0		Processing succeeded (the process was registered)
-1 to -99		The process could not be registered due to an error.
-100	sceMcErrUnbind	sceMcInit() was not executed.
-200	sceMcErrSemapho	The process could not be registered because another process was executing.

sceMcChdir

Change current directory/get current directory

Library	*Introduced*	*Documentation last modified*
libmc	1.1	October 11, 2001

Syntax

int sceMcChdir(	
int *port,*	Port number
int *slot,*	Slot number
const char **path,*	New subdirectory path
char **pwd***)**	Pointer to buffer for storing current directory. If this is unnecessary, 0 is sent.

Calling conditions

Can be called from a thread

Multithread safe

Description

Changes the current directory.

The new subdirectory can be specified by using either an absolute pathname or relative pathname from the current directory.

In a directory name, "." represents the current directory, and ".." represents the parent directory of the current directory.

Table 5-3: Value returned in result of sceMcSync()

Value	Macro	Result
0	sceMcResSucceed	Success
-2	sceMcResNoFormat	Memory card was unformatted
-4	sceMcResNoEntry	Specified pathname did not exist
-5	sceMcResDeniedPermit	Memory card could not be accessed (a 128 KB memory card was inserted)
-10 or less	-11 to -19, -40 to -49, -50 to -59, or -70 to -79 may be returned	Memory Card could not be detected

Note: References to a "memory card" in this document refer to the PS2 memory card.

Notes

If the result of sceMcSync() is a negative number, the current directory of the port that had been manipulated is forcibly changed to root (/).

Return value

Table 5-4

Value	Macro	Result
0		Processing succeeded (the process was registered)
-1 to -99		The process could not be registered due to an error.
-100	sceMcErrUnbind	sceMcInit() was not executed.
-200	sceMcErrSemapho	The process could not be registered because another process was executing.
-210	sceMcErrNullStr	The process could not be registered because a NULL pointer or a zero-byte length string was passed to the pathname

sceMcClose

Close file

Library	*Introduced*	*Documentation last modified*
libmc	1.1	October 11, 2001

Syntax

int sceMcClose(	
int *fd***)**	File handler number of file to be closed

Calling conditions

Can be called from a thread

Multithread safe

Description

Closes a file. If data to be written to the memory card is not updated and remains in the file cache (without sceMcFlush being executed), this function writes it on to the memory card.

Table 5-5: Value returned in result of sceMcSync()

Value	Macro	Result
0	sceMcResSucceed	Success
-2	sceMcResNoFormat	Memory card was unformatted
-4	sceMcResNoEntry	File handler has not been opened
-5	sceMcResDeniedPermit	Memory card could not be accessed (a 128 KB memory card was inserted)
-10 or less	-11 to -19, -40 to -49, -50 to -59, or -70 to -79 may be returned	Memory Card could not be detected

Return value

Table 5-6

Value	Macro	Result
0		Processing succeeded (the process was registered)
-1 to -99		The process could not be registered due to an error.
-100	sceMcErrUnbind	sceMcInit() was not executed.
-200	sceMcErrSemapho	The process could not be registered because another process was executing.

sceMcDelete

Delete file

Library	*Introduced*	*Documentation last modified*
libmc	1.1	October 11, 2001

Syntax

int sceMcDelete(

int *port,*	Port number
int *slot,*	Slot number
const char **name*)	Name of file to be deleted. Either an absolute pathname or relative pathname can be specified.

Calling conditions

Can be called from a thread

Multithread safe

Description

Deletes a file on the Memory Card.

Table 5-7: Value returned in result of sceMcSync()

Value	Macro	Result
0	sceMcResSucceed	Success
-2	sceMcResNoFormat	Memory card was unformatted
-4	sceMcResNoEntry	An attempt was made to delete a non-existent file name or a deleted file
-5	sceMcResDeniedPermit	An attempt was made to delete a file that is in use. Or, the file that was to be deleted does not have a writeable attribute. Or, because a 128 KB memory card was inserted, it could not be accessed.
-6	sceMcResNotEmpty	Entries remain in the subdirectory
-10 or less	-11 to -19, -40 to -49, -50 to -59, or -70 to -79 may be returned	Memory Card could not be detected

Return value

Table 5-8

Value	Macro	Result
0		Processing succeeded (the process was registered)
-1 to -99		The process could not be registered due to an error.
-100	sceMcErrUnbind	sceMcInit() was not executed.
-200	sceMcErrSemapho	The process could not be registered because another process was executing.
-210	sceMcErrNullStr	The process could not be registered because a NULL pointer or a zero-byte length string was passed to the pathname

sceMcEnd

Memory card environment termination processing

Library	*Introduced*	*Documentation last modified*
libmc	2.4.3	January 4, 2002

Syntax

int sceMcEnd(void)

Calling conditions

Can be called from a thread

Multithread safe

Description

Performs termination processing such as releasing resources used by the memory card library.

Return value

Always returns 1.

sceMcFlush

Immediately write file cache

Library	*Introduced*	*Documentation last modified*
libmc	1.1	October 11, 2001

Syntax

int sceMcFlush(	
int *fd***)**	File handler number of file for which file cache is to be written

Calling conditions

Can be called from a thread

Multithread safe

Description

Writes data that remains in the file cache after a file is written, onto the Memory Card.

Table 5-9: Value returned in result of sceMcSync():

Value	Macro	Result
0	sceMcResSucceed	Success
-2	sceMcResNoFormat	Memory card was unformatted
-4	sceMcResNoEntry	File handler has not been opened
-5	sceMcResDeniedPermit	Memory card could not be accessed (a 128 KB memory card was inserted)
-10 or less	-11 to -19, -40 to -49, -50 to -59, or -70 to -79 may be returned	Memory Card could not be detected

Notes

If the result of sceMcSync() is a negative number, the file that was about to be manipulated is forcibly closed.

Return value

Table 5-10

Value	Macro	Result
0		Processing succeeded (the process was registered)
-1 to -99		The process could not be registered due to an error.
-100	sceMcErrUnbind	sceMcInit() was not executed.
-200	sceMcErrSemapho	The process could not be registered because another process was executing.

sceMcFormat

Logically format Memory Card

Library	*Introduced*	*Documentation last modified*
libmc	1.1	October 11, 2001

Syntax

int sceMcFormat(

int *port,*	Port number
int *slot***)**	Slot number

Calling conditions

Can be called from a thread

Multithread safe

Description

Logically formats a Memory Card. A Memory Card must be logically formatted in order to perform file access operations.

Table 5-11: Value returned in result of sceMcSync()

Value	Macro	Result
0	sceMcResSucceed	Success
-5	sceMcResDeniedPermit	Memory card could not be accessed (a 128 KB memory card was inserted)
-10 or less	-11 to -19, -40 to -49, -50 to -59, or -70 to -79 may be returned	Could not be formatted (Memory Card could not be detected)

Return value

Table 5-12

Value	Macro	Result
0		Processing succeeded (the process was registered)
-1 to -99		The process could not be registered due to an error.
-100	sceMcErrUnbind	sceMcInit() was not executed.
-200	sceMcErrSemapho	The process could not be registered because another process was executing.

sceMcGetDir

Get Memory Card file list

Library	*Introduced*	*Documentation last modified*
libmc	1.1	October 11, 2001

Syntax

int sceMcGetDir(	
int *port,*	Port number
int *slot,*	Slot number
const char **name,*	Either an absolute pathname or a relative pathname from the current directory can be used. Specify the pathname of the file for which you want to get the file list. A wildcard character can be assigned in the entry name specification to get a list of only the entry names that are hits.
unsigned *mode,*	0: First call (specify this for checking a port or a slot that is different from the last time sceMcGetDir() was executed) Non-0: Requests the file list following the portion that was obtained by the first call
int *maxent,*	Maximum number of file entries that should be written to table by one call However, if a negative value is specified, no result is written to *table*, and only the total number of entries that match *name* is returned in result of sceMcSync().
sceMcTblGetDir **table)*	Buffer where the file list that was obtained is written. The buffer should be located at 64-byte alignment.

Calling conditions

Can be called from a thread

Multithread safe

Description

Gets a list of files on the Memory Card.

Table 5-13: Value returned in result of sceMcSync()

Value	Macro	Result
0 or more		Number of file entries that were obtained (success). When a value less than *maxent* is returned, it means that there were no file entries beyond that number.
-2	sceMcResNoFormat	Memory card was unformatted
-4	sceMcResNoEntry	Non-existent path was specified

Value	Macro	Result
-5	sceMcResDeniedPermit	An attempt was made to reference a directory that has no execution attribute or readable attribute. Or, because a 128 KB memory card was inserted, it could not be accessed.
-10 or less	-11 to -19, -40 to -49, -50 to -59, or -70 to -79 may be returned	Memory Card could not be detected

Data contents are stored in a table as follows.

Table 5-14

	+7	+6	+5	+4	+3	+2	+1	+0	
+00h	Year		Month	Day	Hour	Minute	Second	--	Creation date/time
+08h	Year		Month	Day	Hour	Minute	Second	--	Modification date/time
+10h	----		Attribute		File size				
+18	------								
+20	File name/Directory name								
+28									
+30									
+38									

Table 5-15: Valid bits in file attribute

libmc.h:		
Readable	sceMcFileAttrReadable	0x0001
Writable	sceMcFileAttrWriteable	0x0002
Executable	sceMcFileAttrExecutable	0x0004
Copy prohibited	sceMcFileAttrDupProhibit	0x0008
Subdirectory	sceMcFileAttrSubdir	0x0020
File write completed	sceMcFileAttrClosed	0x0080
PDA application (1st generation PDA download)	sceMcFileAttrPDAExec	0x0800
PlayStation format data	sceMcFileAttrPS1	0x1000

(Definitions of SCE_STM_*** are in sifdev.h)

Method of using wildcard characters:

- If "player01.score.0304" is sent as *name*, only the entry "player01.score.0304" matches.
- If "?" is entered for part of *name*, a match occurs for any single character corresponding to the ? portion. For example, if "player??.score.????" is sent, "player01.score.0304", "player02.score.1203", and "player03.score." are all matches. If ? is in the middle of a character string and no character appears there, the entry will not match. However, if ? is at the end of a character string, then an entry that ends just before it will also match.

- If "*" is entered for part of *name*, a match occurs for any character string of any length corresponding to the "*" portion.For example, if *name* is "*", all entries match, including even entries that begin with ".".
- "*" can be entered more than once in a character string. For example, if "*score*" is sent, "player01.score.0304", "1-score1203", and "score" all match.
- "*" and "?" can also be combined.

Notes

sceMcFileAttrClosed is an attribute that can be used to confirm the completion of a write operation.

This attribute indicates that operations on a file have completed and that data written has been reflected onto the memory card. This would occur after sceMcWrite() has been executed, and before sceMcClose() or sceMcFlush(), provided that the Memory Card has not been removed.

There is no change to the attribute if sceMcWrite() isn't executed between an sceMcOpen() and sceMcClose().

Conversely, if a file does not have this attribute set, data may not be completely reflected onto the card if the card is removed or an error occurs while the file is being written.

This attribute can be used with Release 1.4.6 or later. Since a file that was written using a library earlier than Release 1.4.6 will not have this attribute, you cannot check this attribute to confirm that a write operation has completed.

For the current Release 2.2, in the browser of the domestic version of the actual PlayStation 2 and that of the DTL-H10000, the sceMcFileAttrClosed attribute (Closed flag) does not get copied while copying a file. This will become part of the browser specification.

The following events can be inferred from the Closed flag:

Flag 1: Writing completed normally.

Flag 0: 'possible' error during writing.

Therefore, during data check processing if the application determines that 'an error occurred because the Closed flag is 0', then processing should not be considered correct.

The Closed flag is only provided for purposes of support.

(We recommend that to decide whether the final contents of the file are correct, a separate mechanism such as a checksum should be implemented on the application side.)

Return value

Table 5-16

Value	Macro	Result
0		Processing succeeded (the process was registered)
-1 to -99		The process could not be registered due to an error.
-100	sceMcErrUnbind	sceMcInit() was not executed.
-200	sceMcErrSemapho	The process could not be registered because another process was executing.
-210	sceMcErrNullStr	The process could not be registered because a NULL pointer or a zero-byte length string was passed to the pathname

sceMcGetEntSpace

Check free space in entry information storage area

Library	*Introduced*	*Documentation last modified*
libmc	1.5	October 11, 2001

Syntax

int sceMcGetEntSpace(	
int *port,*	Port number
int *slot,*	Slot number
const char **path***)**	Path name to be checked

Calling conditions

Can be called from a thread

Multithread safe

Description

Table 5-17: Value returned in result of sceMcSync()

Value	Macro	Result
0 or more		Number of empty entries (success).
-2	sceMcResNoFormat	Memory card was unformatted.
-5	sceMcResDeniedPermit	An attempt was made to reference a directory that has no execution attribute or readable attribute. Or, because a 128 KB memory card was inserted, it could not be accessed.
-10 or less	-11 to -19, -40 to -49, -50 to -59, or -70 to -79 may be returned	Memory Card could not be detected.

When entries are to be created in a given directory, this function checks the number of entries that can be created without taking up new memory card capacity. Use this function when you want to calculate the memory card capacity to be taken up by creating entries.

Entry information takes up one cluster (1 KB) of memory card storage per two entries.

If an existing directory contains an odd number of entries, the 512 bytes in the last half of the last cluster will be empty, and one entry information storage area will be free.

If the number of entries is even and no entry has been deleted, exactly the required amount of space for entry information storage areas will have been allocated. Therefore, the number of free entry information storage areas will be zero, and one cluster will be required to generate a new entry.

Once an entry information storage area is allocated, its size will not be reduced until the directory in which that entry information is being managed is deleted. Therefore, if five entries are eliminated in a directory that now has zero empty entry information storage areas, the number of empty storage areas will increase to five at that time.

For example, if c new entries are created in a directory that has f empty entry information storage areas, memory card storage that will be taken up by the entry information will be ((c-f)/2) rounded up to the nearest integer.

Return Value

Table 5-18

Value	Macro	Result
0		Processing succeeded (the process was registered)
-1 to -99		The process could not be registered due to an error.
-100	sceMcErrUnbind	sceMcInit() was not executed.
-200	sceMcErrSemapho	The process could not be registered because another process was executing.
-210	sceMcErrNullStr	The process could not be registered because a NULL pointer or a zero-byte length string was passed to the pathname

sceMcGetInfo

Examine Memory Card state

Library	*Introduced*	*Documentation last modified*
libmc	1.1	October 11, 2001

Syntax

int sceMcGetInfo(

int *port,*	Port number
int *slot,*	Slot number
int **type,*	Pointer for writing memory card type
int **free,*	Pointer for writing the number of free clusters
int **format***)**	Pointer for writing whether or not the card is formatted (*format==1: Formatted; 0: Unformatted)

Calling conditions

Can be called from a thread

Multithread safe

Description

Checks whether or not a Memory Card is connected.

Table 5-19: Value returned in result of sceMcSync()

Value	Macro	Result
0	sceMcResSucceed	The same Memory Card has been connected continuously since the previous time sceMcGetInfo() was called.
-1	sceMcResChangedCard	Switched to formatted Memory Card.
-2	sceMcResNoFormat	Switched to unformatted Memory card.
-5	sceMcResDeniedPermit	Memory card could not be accessed (a 128 KB memory card was inserted)
-10 or less	-11 to -19, -40 to -49, -50 to -59, or -70 to -79 may be returned	Memory Card could not be detected

Table 5-20: Value returned in *type*

Value	Macro	Result
0	sceMcTypeNoCard	None of these is connected
1	sceMcTypePS1	128 KB memory card
2	sceMcTypePS2	PS2 memory card
3	sceMcTypePDA	PocketStation

- Value returned in *free*

Number of free clusters. If no Memory Card is connected, 0 is returned.

Notes

If the information corresponding to any of the arguments *type*, *free*, or *format* is not required, setting the argument to 0 will reduce the processing time.

Return value

Table 5-21

Value	Macro	Result
0		Processing succeeded (the process was registered)
-1 to -99		The process could not be registered due to an error.
-100	sceMcErrUnbind	sceMcInit() was not executed.
-200	sceMcErrSemapho	The process could not be registered because another process was executing.
-210	sceMcErrNullStr	The process could not be registered because a NULL pointer or a zero-byte length string was passed to the pathname

sceMcGetSlotMax

Get number of slots

Library	*Introduced*	*Documentation last modified*
libmc	2.0	July 2, 2001

Syntax

int sceMcGetSlotMax(	
int *port***)**	Port number

Calling conditions

Can be called from a thread

Multithread safe

Description

sceMcGetSlotMax() itself is a synchronous function. However, it uses the same IOP calls as other libmc functions so it cannot get the slot count while other asynchronous functions are running.

For functions that require a slot number, a value from 0 to (slot count - 1) can be provided. Slots A - D of the SCPH-10090 multitap correspond to slot numbers 0 - 3, respectively.

Return value

Table 5-22

Value	Macro	Result
0		The number of the usable slots in the port.
-1 to -99		The number of slots could not be determined due to an error.
-100	sceMcErrUnbind	sceMcInit() was not executed.
-200	sceMcErrSemapho	The number of slots could not be determined because another process was executing.

sceMcInit

Initialize Memory Card environment

Library	*Introduced*	*Documentation last modified*
libmc	1.1	March 26, 2001

Syntax

int sceMcInit(void)

Calling conditions

Can be called from a thread

Multithread safe

Description

Initializes internal variables that are used in the Memory Card library.

Return value

Value	Macro	Result
0	sceMcIniSucceed	Completed
-101	sceMcIniErrKernel	Initialization failed
-120	sceMcIniOldMcserv	Version of mcserv.irx is old
-121	sceMcIniOldMcman	Version of mcman.irx is old

sceMcMkdir

Create subdirectory

Library	*Introduced*	*Documentation last modified*
libmc	1.1	October 11, 2001

Syntax

int sceMcMkdir(

int *port,*	Port number
int *slot,*	Slot number
const char **name)*	Name of subdirectory to be newly created

Calling conditions

Can be called from a thread

Multithread safe

Description

Creates a subdirectory.

The subdirectory can be specified by using either an absolute pathname or a relative pathname from the current directory. If an absolute pathname is specified, the subdirectories above the subdirectory to be created must already exist.

Table 5-23: Value returned in result of sceMcSync()

Value	Macro	Result
0	sceMcResSucceed	Success
-2	sceMcResNoFormat	Memory card was unformatted
-3	sceMcResFullDevice	Directory could not be created due to insufficient Memory Card capacity.
-4	sceMcResNoEntry	An entry having the same name already exists.
-5	sceMcResDeniedPermit	Memory card could not be accessed (a 128 KB memory card was inserted)
-10 or less	-11 to -19, -40 to -49, -50 to -59, or -70 to -79 may be returned	Memory Card could not be detected

Return value

Table 5-24

Value	Macro	Result
0		Processing succeeded (the process was registered)
-1 to -99		The process could not be registered due to an error.
-100	sceMcErrUnbind	sceMcInit() was not executed.
-200	sceMcErrSemapho	The process could not be registered because another process was executing.
-210	sceMcErrNullStr	The process could not be registered because a NULL pointer or a zero-byte length string was passed to the pathname

sceMcOpen

Open file

Library	*Introduced*	*Documentation last modified*
libmc	1.1	October 11, 2001

Syntax

int sceMcOpen(	
int *port,*	Port number (0/1)
int *slot,*	Slot number
const char **name,*	File name
int *mode***)**	Readable: 1, Writable: 2, Newly created: 0x200 (The value passed is the logical OR of the required mode properties.) These are defined as follows in sifdev.h: SCE_RDONLY 0x0001 SCE_WRONLY 0x0002 SCE_RDWR 0x0003 SCE_CREAT 0x0200

Calling conditions

Can be called from a thread

Multithread safe

Description

Opens a file and returns the file descriptor. Up to three files can be open at the same time for all ports.

name can include a path, but without the path, *name* must be less than 31 characters.

Table 5-25: Value returned in result of sceMcSync()

Value	Macro	Result
0 or more		File descriptor (success)
-2	sceMcResNoFormat	Memory card was unformatted
-3	sceMcResFullDevice	File could not be opened due to insufficient free space.
-4	sceMcResNoEntry	File name is invalid. Or, file did not exist even though *mode* was not Create mode.
-5	sceMcResDeniedPermit	Since file has already been opened in writeable mode, it could not be opened again in writeable mode. Or, file could not be opened because there was no readable or writeable attribute. Or, because a 128 KB memory card was inserted, it could not be accessed.
-7	sceMcResUpLimitHandle	File could not be opened because it would exceed the number of simultaneously open files.

Value	Macro	Result
-10 or less	-11 to -19, -40 to -49, -50 to -59, or -70 to -79 may be returned	Memory Card could not be detected

Return value

Table 5-26

Value	Macro	Result
0		Processing succeeded (the process was registered)
-1 to -99		The process could not be registered due to an error.
-100	sceMcErrUnbind	sceMcInit() was not executed.
-200	sceMcErrSemapho	The process could not be registered because another process was executing.
-210	sceMcErrNullStr	The process could not be registered because a NULL pointer or a zero-byte length string was passed to the pathname

sceMcRead

Read file

Library	*Introduced*	*Documentation last modified*
libmc	1.1	October 11, 2001

Syntax

int sceMcRead(	
int *fd,*	File handler number
void **buff,*	Pointer to buffer for writing data that was read
int *size***)**	Size to be read (unit: bytes)

Calling conditions

Can be called from a thread

Multithread safe

Description

Reads data from a file on the Memory Card.

Table 5-27: Value returned in result of sceMcSync()

Value	Macro	Result
0 or more		Number of bytes that were actually read (success).
-2	sceMcResNoFormat	Memory card was unformatted.
-3	sceMcResFullDevice	File is damaged and could not be read.
-4	sceMcResNoEntry	File handler has not been opened.
-5	sceMcResDeniedPermit	File handler could not be opened in read mode. Or, because a 128 KB memory card was inserted, it could not be accessed.
-10 or less	-11 to -19, -40 to -49, -50 to -59, or -70 to -79 may be returned	Memory Card could not be detected

Notes

If the result of sceMcSync() is a negative number, the file that was about to be manipulated is forcibly closed.

Return value

Table 5-28

Value	Macro	Result
0		Processing succeeded (the process was registered)
-1 to -99		The process could not be registered due to an error.
-100	sceMcErrUnbind	sceMcInit() was not executed.
-200	sceMcErrSemapho	The process could not be registered because another process was executing.

sceMcRename

Rename file or directory

Library	*Introduced*	*Documentation last modified*
libmc	1.5	October 11, 2001

Syntax

int sceMcRename(	
int *port,*	Port number
int *slot,*	Slot number
const char **org,*	File or directory to be renamed
const char **new***)**	New name

Calling conditions

Can be called from a thread

Multithread safe

Description

Changes the filename or directory name.

Table 5-29: Value returned in result of sceMcSync()

Value	Macro	Result
0	sceMcResSucceed	Success.
-2	sceMcResNoFormat	Memory card was unformatted.
-4	sceMcResNoEntry	File or directory having old name could not be found. Or, entry having the same name already exists.
-5	sceMcResDeniedPermit	Memory card could not be accessed (a 128 KB memory card was inserted)
-10 or less	-11 to -19, -40 to -49, -50 to -59, or -70 to -79 may be returned	Memory Card could not be detected

Notes

This function is used for changing a name. It cannot move a file to another directory like the Linux mv command. Also, when specifying the name after it has been changed, use only the new entry name; do not include the path.

Return Value

Table 5-30

Value	Macro	Result
0		Processing succeeded (the process was registered)
-1 to -99		The process could not be registered due to an error.
-100	sceMcErrUnbind	sceMcInit() was not executed.
-200	sceMcErrSemapho	The process could not be registered because another process was executing.
-210	sceMcErrNullStr	The process could not be registered because a NULL pointer or a zero-byte length string was passed to the pathname

sceMcSeek

Move file pointer

Library	*Introduced*	*Documentation last modified*
libmc	1.1	October 11, 2001

Syntax

int sceMcSeek(

int *fd,*	File handler number
int *offset,*	Offset value from starting point indicated by *mode*
int *mode***)**	Starting point of file pointer to be updated 0: Beginning of file 1: Current file pointer 2: End of file + 1 (file size)

Calling conditions

Can be called from a thread

Multithread safe

Description

Moves the file pointer of a file that is open.

Table 5-31: Value returned in result of sceMcSync()

Value	Macro	Result
0 or more		File pointer after it has been moved (success).
-2	sceMcResNoFormat	Memory card was unformatted.
-4	sceMcResNoEntry	File handler has not been opened.
-5	sceMcResDeniedPermit	Memory card could not be accessed (a 128 KB memory card was inserted)
-10 or less	-11 to -19, -40 to -49, -50 to -59, or -70 to -79 may be returned	Memory Card could not be detected

Notes

If the result of sceMcSync() is a negative number, the file that was about to be manipulated is forcibly closed.

Return value

Table 5-32

Value	Macro	Result
0		Processing succeeded (the process was registered)
-1 to -99		The process could not be registered due to an error.
-100	sceMcErrUnbind	sceMcInit() was not executed.
-200	sceMcErrSemapho	The process could not be registered because another process was executing.

sceMcSetFileInfo

Update file information

Library	*Introduced*	*Documentation last modified*
libmc	1.1	October 11, 2001

Syntax

int sceMcSetFileInfo(	
int *port,*	Port number
int *slot,*	Slot number
const char **name,*	File name
const char **info*	File information to be written
unsigned *valid***)**	Set 1 for bits corresponding to information you want to set.
	Creation date/time: sceMcFileInfoCreate 0x01
	Modification date/time: sceMcFileInfoModify 0x02
	File attribute: sceMcFileInfoAttr 0x04

Calling conditions

Can be called from a thread

Multithread safe

Description

Overwrites the system information of a file.

Data contents are stored in table as follows (the same format as the first half of sceMcTblGetDir).

Table 5-33

	+7	+6	+5	+4	+3	+2	+1	+0	
+00h	Year		Month	Day	Hour	Minute	Second	--	Creation date/time
+08h	Year		Month	Day	Hour	Minute	Second	--	Modification date/time
+10h	----		Attribute		----				

Table 5-34: Valid bits in file attribute:

libmc.h:		
Readable	sceMcFileAttrReadable	0x0001
Writable	sceMcFileAttrWriteable	0x0002
Executable	sceMcFileAttrExecutable	0x0004
Copy prohibited	sceMcFileAttrDupProhibit	0x0008
PDA application (1st generation PDA download)	sceMcFileAttrPDAExec	0x0800
PlayStation format data	sceMcFileAttrPS1	0x1000

(Definitions of SCE_STM_*** are in sifdev.h)

Table 5-35:Value returned in result of sceMcSync()

Value	Macro	Result
0	sceMcResSucceed	Success
-2	sceMcResNoFormat	Memory card was unformatted.
-4	sceMcResNoEntry	Specified entry did not exist.
-5	sceMcResDeniedPermit	Memory card could not be accessed (a 128 KB memory card was inserted)
-10 or less	-11 to -19, -40 to -49, -50 to -59, or -70 to -79 may be returned	Memory Card could not be detected

Return value

Table 5-36

Value	Macro	Result
0		Processing succeeded (the process was registered)
-1 to -99		The process could not be registered due to an error.
-100	sceMcErrUnbind	sceMcInit() was not executed.
-200	sceMcErrSemapho	The process could not be registered because another process was executing.

sceMcSync

Await completion of registered process

Library	*Introduced*	*Documentation last modified*
libmc	1.1	March 26, 2001

Syntax

int sceMcSync(

int *mode,*	0: Wait for end of registered asynchronous function 1: Check status of asynchronous function and return immediately
int **cmd,*	Pointer to variable for storing function number of registered asynchronous function
int **result***)**	Pointer to variable for storing execution result of asynchronous function

Calling conditions

Can be called from a thread

Multithread safe

Description

Checks for the end of an asynchronous function.

Function numbers are defined in libmc.h.

sceMcFuncNoCardInfo:	1
sceMcFuncNoOpen:	2
sceMcFuncNoClose:	3
sceMcFuncNoSeek:	4
sceMcFuncNoRead:	5
sceMcFuncNoWrite:	6
sceMcFuncNoFlush:	10
sceMcFuncNoMkdir:	11
sceMcFuncNoChDir:	12
sceMcFuncNoGetDir:	13
sceMcFuncNoFileInfo:	14
sceMcFuncNoDelete:	15
sceMcFuncNoFormat:	16
sceMcFuncNoUnformat:	17
sceMcFuncNoEntSpace:	18
sceMcFuncNoRename:	19
sceMcFuncChgPrior:	20

Return value

Value	Macro	Result
0	sceMcExecRun	Asynchronous function is being executed
1	sceMcExecFinish	Asynchronous function terminated
-1	sceMcExecIdle	No function registered

sceMcUnformat

Erase Memory Card formatting

Library	*Introduced*	*Documentation last modified*
libmc	1.1	October 11, 2001

Syntax

int sceMcUnformat(

int *port,*	Port number
int *slot*)	Slot number

Calling conditions

Can be called from a thread

Multithread safe

Description

Unformats a Memory Card.

This function is used for debugging.

Table 5-37: Value returned in result of sceMcSync()

Value	Macro	Result
0	sceMcResSucceed	Success
-5	sceMcResDeniedPermit	Memory card could not be accessed (a 128 KB memory card was inserted)
-10 or less	-11 to -19, -40 to -49, -50 to -59, or -70 to -79 may be returned	Could not be unformatted (Memory Card could not be detected)

Notes

Although unformatting a formatted Memory Card takes about three seconds to complete, unformatting an already unformatted Memory Card takes about 20 seconds to complete.

Return value

Table 5-38

Value	Macro	Result
0		Processing succeeded (the process was registered)
-1 to -99		The process could not be registered due to an error.
-100	sceMcErrUnbind	sceMcInit() was not executed.
-200	sceMcErrSemapho	The process could not be registered because another process was executing.

sceMcWrite

Write to file

Library	*Introduced*	*Documentation last modified*
libmc	1.1	January 4, 2002

Syntax

int sceMcWrite(

int *fd,*	File handler number
const void **buff,*	Pointer to buffer where data to be written is stored
int *size***)**	Size to be written (unit:bytes)

Calling conditions

Can be called from a thread

Multithread safe

Description

Writes data to a file on the Memory Card. As long as the write data does not exceed a size that can be maintained in the file cache, it only remains in the file cache and is not reflected on the Memory Card. If you want to immediately reflect it on the Memory Card, call sceMcFlush(). (The file cache size is 24K bytes.)

Table 5-39: Value returned in result of sceMcSync()

Value	Macro	Result
0 or more		Number of bytes that were actually written (success).
-2	sceMcResNoFormat	Memory card was unformatted.
-3	sceMcResFullDevice	Data could not be written due to insufficient free space.
-4	sceMcResNoEntry	File handler has not been opened.
-5	sceMcResDeniedPermit	File handler could not be opened in write mode. Or, because a 128 KB memory card was inserted, it could not be accessed.
-8	sceMcResFailReplace	Attempted to use the subrogation area when writing failed, but writing to the subrogation area failed repeatedly and ultimately data could not be written.
-10 or less	-11 to -19, -40 to -49, -50 to -59, or -70 to -79 may be returned	Memory Card could not be detected

Notes

If the result of sceMcSync() is a negative number, the file that was about to be manipulated is forcibly closed. If writing failed, there is no guarantee that the data that had been written is reflected on the Memory Card.

Return value

Table 5-40

Value	Macro	Result
0		Processing succeeded (the process was registered)
-1 to -99		The process could not be registered due to an error.
-100	sceMcErrUnbind	sceMcInit() was not executed.
-200	sceMcErrSemapho	The process could not be registered because another process was executing.

Chapter 6: PDA Library
Table of Contents

Structures

sceMcxTblInfo

Structure for passing PDA information

Library	*Introduced*	*Documentation last modified*
libmcx	2.0	October 6, 2000

Structure

typedef struct {	
short *AppliNo;*	PDA application number to execute or being executed
short *Reserve1;*	Reserved area. Must be filled with 00.
int *AplArg;*	Arguments to pass to PDA application
unsigned char *Speaker;*	Disable speaker
unsigned char *Infrared;*	Disable infrared communications/remote control transmission
unsigned char *Flash;*	Inhibit PDA application from writing to Flash memory
unsigned char *Led;*	Disable LED
struct {	
unsigned char *Week,;*	RTC (real time clock) day of the week
unsigned char *Sec*	RTC seconds
unsigned char *Min*	RTC minutes
unsigned char *Hour*	RTC hours
unsigned char *Day*	RTC day
unsigned char *Month;*	RTC month
unsigned short *Year***;**	RTC year
} *_Rtc;*	
unsigned *Serial;*	
} sceMcxTblInfo;	

Table 6-1

#define	Member
#define PWeek	_Rtc.Week
#define PSec	_Rtc.Sec
#define PMin	_Rtc.Min
#define PHour	_Rtc.Hour
#define PDay	_Rtc.Day
#define PMonth	_Rtc.Month
#define PYear	_Rtc.Year

Description

This structure is used to pass data when PDA information is obtained withsceMcxGetInfo() or when PDA information is set up with sceMcxSetInfo().

sceMcxTblUifs

Structure for passing user interface status

Library	*Introduced*	*Documentation last modified*
libmcx	2.0	October 6, 2000

Structure

```
typedef struct {
    unsigned char AMin, AHour;          Alarm time (minutes)
                                        Alarm time (hours)
    unsigned Alarm:1;                   Alarm ON/OFF
    unsigned KeyLock:1;                 Key lock ON/OFF
    unsigned Volume:2;                  Speaker volume
    unsigned AreaCode:3;                PocketStation area code (read-only)
    unsigned RtcSet:1;                  Authenticity of real-time clock (whether it is set or not)
    unsigned char Reserve1;             Reserved area. Must be filled with 00.
    unsigned short Font;                Font address (read-only)
    short Reserve2;                     Reserved area. Must be filled with 00.
} sceMcxTblUifs;
```

Description

This data structure is used to pass information when user interface status is obtained with sceMcxGetUifs() or user interface status is set with sceMcxGetUifs().

Functions

sceMcxChangeThreadPriority

Set IOP module (mcxserv.irx) thread priority

Library	*Introduced*	*Documentation last modified*
libmcx	2.0	March 26, 2001

Syntax

int sceMcxChangeThreadPriority(	
int *level)*	Thread priority

Calling conditions

Can be called from a thread

Not multithread safe

Description

Changes the thread priority of mcxserv.irx, the IOP module of the PDA library. Possible priority settings are values in the range of USER_HIGHEST_PRIORITY - USER_LOWEST_PRIORITY (inclusive), as defined in thread.h. The initial thread priority value for the mcxserv.irx module is 104.

Return value in result of sceMcxSync():

Table 6-2

Value	Macro	Result
0	KE_OK	Successful
-403	KE_ILLEGAL_PRIORITY	Thread priority value exceeds valid range

Notes

The thread priority can also be set using sceSifLoadModule() when mcxserv.irx is loaded. For example, if mcxserv.irx is loaded as shown below, the initial thread priority will be 100. The thread priority string should be expressed as a decimal value.

```
unsigned char *param = "thpri=100";
sceSifLoadModule( "host0:/usr/local/sce/iop/modules/mcxserv.irx", strlen(param)+1, param);
```

Return value

0:	Operation was registered
1 or greater:	The function number (sceMcxFuncNo.....) of the function being executed is returned. The operation could not be registered because another process was running.
-1 or less:	The operation could not be registered due to an error.

sceMcxGetInfo

Get PDA information

Library	*Introduced*	*Documentation last modified*
libmcx	2.0	March 26, 2001

Syntax

int sceMcxGetInfo(	
int *port,*	Port number
int *slot,*	Slot number
sceMcxTblInfo **info)*	Pointer to buffer for storing PDA information

Calling conditions

Can be called from a thread

Not multithread safe

Description

This function allows the following values stored in the PDA to be accessed:

- Application number of PDA application being executed

  The starting block of the application being executed on the PDA is returned. The block referred to here is in file management units (8 KB) in the flash memory area of the PlayStation memory card. If the launcher application is executing, 0 is returned. If another application is executing, a value of 1 - 15 is returned.
- Disabled state of [Speaker output/infrared transmission/PDA application flash write/LED]

  If a feature is disabled, a 1 is returned. If enabled, a 0 is returned.
- Real-time clock

  The real-time clock in the PDA is read and the year/month/day/minutes/seconds/day of the week is returned.
- Serial number

  The upper 8 bits represent a single ASCII character, and the lower 24 bits represent an 8-digit decimal number. The two are combined to form a serial number such as A00000001.

Return value in result of sceMcxSync():

Table 6-3

Value	Macro	Result
0	sceMcxResSucceed	Successful
-12	sceMcxResNoDevice	PocketStation was not detected

Return value

0:	Operation was registered
1 or greater:	The function number (sceMcxFuncNo.....) of the function being executed is returned. The operation could not be registered because another process was running.
-1 or less:	The operation could not be registered due to an error.

sceMcxGetMem

Read PDA memory

Library	*Introduced*	*Documentation last modified*
libmcx	2.0	March 26, 2001

Syntax

int sceMcxGetMem(

int *port,*	Port number
int *slot,*	Slot number
void **buff,*	Pointer to buffer for reading memory contents
unsigned *addr,*	Starting address of memory to be read
unsigned *size)*	Number of bytes to read (128 bytes maximum)

Calling conditions

Can be called from a thread

Not multithread safe

Description

The specified number of bytes of PDA memory beginning at the specified address is read and stored in the buffer. A PDA bus error will be generated if the address is not in the ranges shown below or if an attempt is made to access 0x2****** addresses and virtual flash memory has not been set up.

Readable ranges:

0x0******, 0x2******, 0x4******, 0x6******, 0x8******
0xA******, 0xB******, 0xC******, 0xD****** (****** can be any 6 hexadecimal digits)

Return value in result of sceMcxSync():

Table 6-4

Value	Macro	Result
0	sceMcxResSucceed	Successful
-12	sceMcxResNoDevice	PocketStation was not detected

Return value

0:	Operation was registered
1 or greater:	The function number (sceMcxFuncNo.....) of the function being executed is returned. The operation could not be registered because another process was running.
-1 or less:	The operation could not be registered due to an error.

sceMcxGetUifs

Get user interface status

Library	*Introduced*	*Documentation last modified*
libmcx	2.0	March 26, 2001

Syntax

int sceMcxGetUifs(

int *port,*	Port number
int *slot,*	Slot number
sceMcxTblUifs **uifs)*	Pointer to buffer for reading user interface status

Calling conditions

Can be called from a thread

Not multithread safe

Description

This function reads the user interface status from the PDA.

The members of the sceMcxTblUifs structure used to pass the user interface status are shown below.

- alarm time (hours): 0 - 23
- alarm time(minutes): 0 - 59
- alarm (ON/OFF): 0 OFF, 1 ON
- keylock: 0 unlock, 1 lock
- speaker volume: 0 high, 1 low, 2 off
- area code: 0 Japan, 1: North America, 2: Europe
- RTC set: 0 data invalid (RTC has not been set after reset), 1 data valid (RTC has been set after reset)
- Font data base address: relative address from 0x4000000

Return value in result of sceMcxSync():

Table 6-5

Value	Macro	Result
0	sceMcxResSucceed	Successful
-12	sceMcxResNoDevice	PocketStation was not detected

Return value

0:	Operation was registered
1 or greater:	The function number (sceMcxFuncNo.....) of the function being executed is returned. The operation could not be registered because another process was running.
-1 or less:	The operation could not be registered due to an error.

sceMcxHideTrans

Disable display during data transfer

Library	*Introduced*	*Documentation last modified*
libmcx	2.0	March 26, 2001

Syntax

int sceMcxHideTrans(	
int *port,*	Port number
int *slot)*	Slot number

Calling conditions

Can be called from a thread

Not multithread safe

Description

This function hides the data transfer display on the LCD screen provided by sceMcxShowTrans().

When this function is called, the PDA kernel generates a "file transfer control callback" for the running PDA application. The data transfer display, displayed on the PDA's LCD screen in the subroutine registered in the "Start/End setting time for display of file transfers from PlayStation" in the PDA's "User callback settings (swi 1)", is stopped. (For information on operations required by PDA applications, refer to the "PDA Kernel Specification".)

Return value in result of sceMcxSync():

Table 6-6

Value	Macro	Result
0	sceMcxResSucceed	Successful
-12	sceMcxResNoDevice	PocketStation was not detected

Return value

0:	Operation was registered
1 or greater:	The function number (sceMcxFuncNo.....) of the function being executed is returned. The operation could not be registered because another process was running.
-1 or less:	The operation could not be registered due to an error.

sceMcxInit

Initialize PDA library environment

Library	*Introduced*	*Documentation last modified*
libmcx	2.0	March 26, 2001

Syntax

int sceMcxInit(*void***)**

Calling conditions

Can be called from a thread

Not multithread safe

Description

This function initializes the internal variables used in the PDA library.

Return value

Table 6-7

Value	Macro	Result
0	sceMcxIniSucceed	Successful
-101	sceMcxIniErrKernel	Initialization failed
-120	sceMcxIniOldMcxserv	mcxserv.irx version is old
-121	sceMcxIniOldMcxman	mcxman.irx version is old

sceMcxReadDev

Read PDA device entry

Library	*Introduced*	*Documentation last modified*
libmcx	2.0	March 26, 2001

Syntax

int sceMcxReadDev(

int *port,*	Port number
int *slot,*	Slot number
int *devno,*	Device entry number
const void **para,*	Pointer to buffer for storing constant parameters
unsigned *parasize,*	Byte length of constant parameters (fixed by device entry)
void **cont,*	Pointer to buffer for storing variable data
unsigned *contsize)*	Byte length of variable data

Calling conditions

Can be called from a thread

Not multithread safe

Description

This function reads from reserved devices and user-defined devices on the PDA. A constant parameter is passed to a device, and the resulting variable data is read. Some devices do not have constant parameters.

To call a user-defined device, create a subroutine as described in the "Kernel Services Overview: Communication with the PlayStation: Device Entry Callbacks" section of the "PDA Kernel Specification". The device must be registered in the "device entry table" in the memory card file header.

The following three reserved devices are available (macros are defined in libmcx.h):

Table 6-8

Device name	Dev no.	Macro
Real-time clock	0	sceMcxDevRtc
PDA memory	1	sceMcxDevMem
User interface status	2	sceMcxDevUIFS

Return value in result of sceMcxSync():

Table 6-9

Value	Macro	Result
0	sceMcxResSucceed	Successful
-12	sceMcxResNoDevice	PocketStation was not detected

Return value

0:	Operation was registered
1 or greater:	The function number (sceMcxFuncNo.....) of the function being executed is returned. The operation could not be registered because another process was running.
-1 or less:	The operation could not be registered due to an error.

sceMcxSetInfo

Update PDA information

Library	*Introduced*	*Documentation last modified*
libmcx	2.0	March 26, 2001

Syntax

int sceMcxSetInfo(

int *port,*	Port number
int *slot,*	Slot number
const sceMcxTblInfo **info,*	Pointer to buffer for storing update contents
unsigned *part)*	Parameters to be updated

6 types of parameters can be updated.

The following are the bits corresponding to the parameters to be updated (definitions in libmcx.h)

sceMcxBitAppli	PDA application execution
sceMcxBitSpeaker	Speaker disabled state
sceMcxBitInfrared	Infrared communications/remote control disabled state
sceMcxBitFlash	PDA application flash write disabled state
sceMcxBitLed	LED disabled state
sceMcxBitDate	Current time of real-time clock

Calling conditions

Can be called from a thread

Not multithread safe

Description

This function updates PDA information.

- PDA application execution

  A PDA application to be executed is specified for the AppliNo member using the starting block number stored in the PDA. The values set up in AplArg are passed to the application as arguments. The block referred to here is in file management units (8 KB) in the flash memory area of the PlayStation memory card. To run the launcher application, specify 0. To run another application, specify a value of 1 - 15.

  To determine the starting block number, first use sceMcxGetDir() to check the PDA application file information. As a result, the value stored in table ->PdaAplNo is the application number.

  When a PDA application is to be run, the PlayStation sends an end application request to the currently running PDA application. This request is sent as "bit 11: end PDA application" in the result for "Get PDA status (swi 6)" (this flag must be monitored periodically).

  For information about what operations a PDA application needs to perform when it exits, refer to the "PDA Kernel Specification".

  When the parameter is updated, an end request will be sent to the currently running PDA application. A running application may refuse to exit though, so sceMcxGetInfo() should be used to confirm that the PDA application has switched. (However, when there is no communication between the PDA application and the PlayStation 2, the application number of the currently running application cannot be determined).

- [Speaker output/infrared transmission/LED] disabled state 1 to disable a feature, 0 to enable.

  These controls are provided due to the limited current capacity that is available to the front-panel terminals on the PlayStation 2. All features are disabled by default when the PDA is first plugged into the PlayStation 2.

  The table below shows the current consumption of the different modules.

  The maximum current that can be supplied by the PlayStation 2 is 160 mA total for two ports, so adjustments should be made to prevent exceeding this value(particularly when using multitaps).

Table 6-10

Module name	Current consumption
CPU chip	10mA
IR module transmission	70mA
Speaker	20mA
LED	10mA

"Get PDA status (swi 6)" should be used to check the usage restriction state of these three features when using them in a PDA application.

(For information about what operations PDA applications need to perform, please refer to the "PDA Kernel Specification").

- PDA application flash write disabled state

  1 to disable PDA flash memory writes by a PDA application. 0 to enable.

  If a PDA application is writing to flash memory and communication with the PlayStation 2 takes place, problems may develop in the PDA's internal processing and there may be access conflicts. This setting is provided so that writing to flash may be disabled because communication between a PlayStation 2 program and the PDA may fail.

  However, this enable/disable setting is reported to the PDA application only as the result of "Get PDA status (swi 6)". Thus, if a PDA application is to write to flash memory, it must use "Get PDA status (swi 6)" to see that writing is possible. (For information about what operations PDA applications are required to perform, please refer to the "PDA Kernel Specification".)

  The default setting is that PDA applications are disabled from writing to flash memory. In other words, priority is given to communication with the PlayStation 2.

  If flash memory writes are enabled for PDA applications, the PDA may stop communications with the PlayStation 2 to write to flash. Thus, the result from sceMcxSync() may repeatedly give "-12: PocketStation was not detected".

- Real-time clock

  Sets the PDA's internal real-time clock (year, month, day, hour, minute, second, day of the week).

  The day of the week is automatically calculated.

Return value in result of sceMcxSync():

Table 6-11

Value	Macro	Result
0	sceMcxResSucceed	Successful
-12	sceMcxResNoDevice	PocketStation was not detected

Return value

0:	Operation was registered
1 or greater:	The function number (sceMcxFuncNo.....) of the function being executed is returned. The operation could not be registered because another process was running.
-1 or less:	The operation could not be registered due to an error.

sceMcxSetLed

LED on/off

Library	*Introduced*	*Documentation last modified*
libmcx	2.0	March 26, 2001

Syntax

int sceMcxSetLed(

int *port,*	Port number
int *slot,*	Slot number
int *mode)*	0:off, other than 0: on

Calling conditions

Can be called from a thread

Not multithread safe

Description

This function turns the LED on or off.

It would also be possible to enter LED settings or check the LED state using sceMcxGetMem() or sceMcxSetMem() to directly access PIO0, PIO1.

Return value in result of sceMcxSync():

Table 6-12

Value	Macro	Result
0	sceMcxResSucceed	Successful
-12	sceMcxResNoDevice	PocketStation was not detected

Return value

0:	Operation was registered
1 or greater:	The function number (sceMcxFuncNo.....) of the function being executed is returned. The operation could not be registered because another process was running.
-1 or less:	The operation could not be registered due to an error.

sceMcxSetMem

Write to PDA memory

Library	*Introduced*	*Documentation last modified*
libmcx	2.0	March 26, 2001

Syntax

int sceMcxSetMem(

int *port,*	Port number
int *slot,*	Slot number
const void **buff,*	Pointer to buffer containing data to be written to PDA memory
unsigned *addr,*	Base address of PDA memory to be written
unsigned *size)*	Number of bytes to write (128 bytes maximum)

Calling conditions

Can be called from a thread

Not multithread safe

Description

This function writes the specified number of bytes to the specified address.

A PDA bus error will be generated if the write address is not in one of the ranges listed below or if 0x2****** is accessed and virtual flash memory has not been set up.

Writable regions:

0x0******, 0x6******, 0xA******

0xB******, 0xC******, 0xD****** (****** can be any 6 hexadecimal digits)

Return value in result of sceMcxSync():

Table 6-13

Value	Macro	Result
0	sceMcxResSucceed	Successful
-12	sceMcxResNoDevice	PocketStation was not detected

Return value

0:	Operation was registered
1 or greater:	The function number (sceMcxFuncNo.....) of the function being executed is returned. The operation could not be registered because another process was running.
-1 or less:	The operation could not be registered due to an error.

sceMcxSetUifs

Update user interface status

Library	*Introduced*	*Documentation last modified*
libmcx	2.0	March 26, 2001

Syntax

int sceMcxSetUifs(

int *port,*	Port number
int *slot,*	Slot number
const sceMcxTblUifs **uifs)*	Pointer to buffer containing user interface status contents to be updated

Calling conditions

Can be called from a thread

Not multithread safe

Description

This function updates the PDA's user interface status.

The following items from the sceMcxTblUifs structure used to pass the user interface status can be updated. All others are read-only.

- alarm time (hours): 0 - 23
- alarm time(minutes): 0 - 59
- alarm (ON/OFF): 0 OFF, 1 ON
- keylock: 0 unlock, 1 lock
- speaker volume: 0 high, 1 low, 2 off
- RTC setting: 0 data invalid, 1 data valid

Return value in result of sceMcxSync():

Table 6-14

Value	Macro	Result
0	sceMcxResSucceed	Successful
-12	sceMcxResNoDevice	PocketStation was not detected

Return value

0:	Operation was registered
1 or greater:	The function number (sceMcxFuncNo.....) of the function being executed is returned. The operation could not be registered because another process was running.
-1 or less:	The operation could not be registered due to an error.

sceMcxShowTrans

Begin data transfer display

Library	*Introduced*	*Documentation last modified*
libmcx	2.0	March 26, 2001

Syntax

int sceMcxShowTrans(

int *port,*	Port number
int *slot,*	Slot number
int *dir,*	Transfer direction (0: PDA -> PlayStation, non-zero: PlayStation -> PDA)
int *timeout)*	Timeout time to stop display if no request to end transfer display is received (in seconds for the launcher application)

Calling conditions

Can be called from a thread

Not multithread safe

Description

When saving a PDA application file, this function is called before opening the file to avoid alternate sector processing (to save the PDA program to contiguous memory).

sceMcFormat() from libmc performs alternate sector initialization, so sceMcxShowTrans() should not be called when formatting the memory card (initialization of alternate sectors will be prevented and formatting will fail).

When this function is called, a "file transfer control callback" is generated by the PDA kernel for the running PDA application. The data transfer display is displayed on the PDA's LCD screen in the subroutine registered in the "Start/Stop time setting for display of file transfers from PlayStation" in the PDA's "User callback settings (swi 1)". The timeout setting is used to allow the PDA application itself to stop the transfer display in cases such as when the PlayStation 2 is accidentally reset. In normal operations, the transfer display is cleared by calling sceMcxHideTrans() after the file transfer is completed.

(For details on the operations that PDA applications need to perform, please refer to the "PDA Kernel Specification".)

Return value in result of sceMcxSync():

Table 6-15

Value	Macro	Result
0	sceMcxResSucceed	Successful
-12	sceMcxResNoDevice	PocketStation was not detected

Return value

0:	Operation was registered
1 or greater:	The function number (sceMcxFuncNo.....) of the function being executed is returned. The operation could not be registered because another process was running.
-1 or less:	The operation could not be registered due to an error.

sceMcxSync

Wait for completion of registered operation

Library	*Introduced*	*Documentation last modified*
libmcx	2.0	March 26, 2001

Syntax

int sceMcxSync(

int *mode,*	0: wait for registered asynchronous function to complete 1: check state of asynchronous function and return immediately
int **cmd,*	Pointer to variable storing the function number of registered asynchronous function
int **result)*	Pointer to variable for storing execution results of asynchronous function

Calling conditions

Can be called from a thread

Not multithread safe

Description

This function checks for completion of an asynchronous function.

Function numbers are defined in libmcx.h.

sceMcxFuncGetInfo	1
sceMcxFuncSetInfo	2
sceMcxFuncGetMem	3
sceMcxFuncSetMem	4
sceMcxFuncShowTrans	5
sceMcxFuncHideTrans	6
sceMcxFuncReadDev	7
sceMcxFuncWriteDev	8
sceMcxFuncGetUIFS	9
sceMcxFuncSetUIFS	10
sceMcxFuncSetLED	11
sceMcxFuncChgPrior	12

Return value

Table 6-16

Value	Macro	Result
0	sceMcxExecRun	Asynchronous function running
1	sceMcxExecFin	Asynchronous function finished
-1	sceMcxExecNone	Not registered

sceMcxWriteDev

Write to PDA device entry

Library	*Introduced*	*Documentation last modified*
libmcx	2.0	March 26, 2001

Syntax

int sceMcxWriteDev(

int *port,*	Port number
int *slot,*	Slot number
int *devno,*	Device entry number
const void **para,*	Pointer to buffer for storing constant parameters
unsigned *parasize,*	Byte length of constant parameters (fixed by device entry)
const void **cont,*	Pointer to buffer for variable data
unsigned *contsize)*	Byte length of variable data (128 bytes maximum)

Calling conditions

Can be called from a thread

Not multithread safe

Description

This function writes to reserved devices and user-defined devices on the PDA. A constant parameter is passed to the device and variable data requested by the constant parameter is written. Some devices do not have constant parameters.

To call a user-defined device, create a subroutine as described in the "Kernel Services Overview: Communication with the PlayStation: Device Entry Callbacks" section of the "PDA Kernel Specification". The device must be registered in the "device entry table" in the memory card file header.

The following three reserved devices are available (macros are defined in libmcx.h).

Table 6-17

Device name	Dev no.	Macro
Real-time clock	0	sceMcxDevRtc
PDA memory	1	sceMcxDevMem
User interface status	2	sceMcxDevUIFS

Return value in result of sceMcxSync():

Table 6-18

Value	Macro	Result
0	sceMcxResSucceed	Successful
-12	sceMcxResNoDevice	PocketStation was not detected

Return value

0:	Operation was registered
1 or greater:	The function number (sceMcxFuncNo.....) of the function being executed is returned. The operation could not be registered because another process was running.
-1 or less:	The operation could not be registered due to an error.

Chapter 7: Multitap Library
Table of Contents

Functions

sceMtapChangeThreadPriority

Change IOP thread priority

Library	*Introduced*	*Documentation last modified*
libmtap	2.0	March 26, 2001

Syntax

int sceMtapChangeThreadPriority(

int *priority_high,*	Priority of main thread
int *priority_low);*	Priority of SIF interface thread

Calling conditions

Can be called from a thread

Not multithread safe (must be called in an interrupt-enabled state)

Description

This function changes the thread priority of the mtapman.irx IOP module. Both of the following must be specified:

priority_high	main thread
priority_low	SIF interface thread

The main thread is executed once per frame. The SIF interface thread is executed when any of the following are executed.

sceMtapPortOpen()

sceMtapPortClose()

sceMtapGetConnection()

sceMtapChangeThreadPriority()

The thread priority can also be specified when mtapman.irx is loaded. This function cannot be run unless sceMtapInit() has already executed.

Return value

1	Successfully changed
Other than 1	Failed

sceMtapGetConnection

Get multitap connection status

Library	*Introduced*	*Documentation last modified*
libmtap	1.4	March 26, 2001

Syntax

int sceMtapGetConnection(	
int *port***)**	Port number

Calling conditions

Can be called from a thread

Not multithread safe (must be called in an interrupt-enabled state)

Description

Gets information about whether or not the multitap is connected to an opened port. This function uses a RPC to ask the IOP for the latestinformation.

For an unopened port, "No multitap" is returned, regardless of whether the multitap is connected.

Return value

1: Multitap exists

Other than 1: No multitap

sceMtapInit

Initialize multitap library

Library	*Introduced*	*Documentation last modified*
libmtap	1.4	March 26, 2001

Syntax

int sceMtapInit(void)

Calling conditions

Can be called from a thread

Not multithread safe (must be called in an interrupt-enabled state)

Description

Initializes the multitap library.

Return value

1: Success

Other than 1: Failure

sceMtapPortClose

Close a port for the target multitap

Library	*Introduced*	*Documentation last modified*
libmtap	1.4	March 26, 2001

Syntax

int sceMtapPortClose(	
int *port***)**	Port number Controller port: 0 or 1 Memory card slot: 2 or 3

Calling conditions

Can be called from a thread

Not multithread safe (must be called in an interrupt-enabled state)

Description

Closes a port that was opened as a multitap connection destination.

Physically, a single multitap is connected to controller port 0 and memory card slot 2 (or controller port 1 and memory card slot 3) but internally these operate as independent multitaps.

Thus, the ports opened with sceMtapPortOpen() should be closed separately.

Return value

1: Request was accepted

Other than 1: Request was not accepted

See also

sceMtapPortOpen()

sceMtapPortOpen

Open a port for the target multitap

Library	*Introduced*	*Documentation last modified*
libmtap	1.4	March 26, 2001

Syntax

int sceMtapPortOpen(

int *port***)**	Port number Controller port: 0 or 1 Memory card slot: 2 or 3

Calling conditions

Can be called from a thread

Not multithread safe (must be called in an interrupt-enabled state)

Description

Specifies a port that is to be monitored as a multitap connection destination.

The IOP driver only recognizes ports that were specified by this function as multitap connection destinations, and routinely polls them.

Physically, a single multitap is connected to controller port 0 and memory card slot 2 (or controller port 1 and memory card slot 3) but internally these operate as independent multitaps.

Thus, when a multitap is used with both the controller and PS2 memory card ports, they should both be opened. If they are not, only slot A will be enabled.

Return value

1: Request was accepted

Other than 1: Request was not accepted

See also

sceMtapPortClose()

Chapter 8: Controller Library
Table of Contents

Functions

scePadEnd

Terminate controller library

Library	*Introduced*	*Documentation last modified*
libpad	1.2	March 26, 2001

Syntax

int scePadEnd(void)

Calling conditions

Can be called from a thread

Not multithread safe (must be called in an interrupt-enabled state)

Description

Terminates the controller library.

Terminates all threads that had been running in the IOP and releases resources.

Return value

1: Success

Other than 1: Failure

See also

scePadInit()

scePadEnterPressMode

Set controller to pressure-sensitive mode

Library	*Introduced*	*Documentation last modified*
libpad	1.2	March 26, 2001

Syntax

int scePadEnterPressMode(

int *port,*	Controller port number
int *slot***)**	Slot number (fixed at 0, except when using a multitap)

Calling conditions

Valid only when scePadGetState() is scePadStateStable.

Can be called from a thread

Not multithread safe (must be called in an interrupt-enabled state)

Description

Sets the controller of the specified port to pressure-sensitive mode. Actually, to switch the controller to pressure-sensitive mode requires several frames, and the switch is performed asynchronously.

Determine when processing ends either by using scePadGetState() to monitor the controller connection state or by using scePadGetReqState() to get the execution result of the request.

When the controller enters pressure-sensitive mode, the controller ID changes to 0x79.

Return value

1: Request was accepted

Other than 1: Request was not accepted

See also

scePadExitPressMode()

scePadExitPressMode

Exit from pressure-sensitive mode

Library	*Introduced*	*Documentation last modified*
libpad	1.2	March 26, 2001

Syntax

int scePadExitPressMode(

int *port,*	Controller port number
int *slot***)**	Slot number (fixed at 0, except when using a multitap)

Calling conditions

Valid only when scePadGetState() is scePadStateStable.

Can be called from a thread

Not multithread safe (must be called in an interrupt-enabled state)

Description

Causes the controller of the specified port, which is already in pressure-sensitive mode, to exit from pressure-sensitive mode.

Actually, several frames are required for the controller to exit from pressure-sensitive mode, and the mode switch is performed asynchronously.

Determine when processing ends either by using scePadGetState() to monitor the controller connection state or by using scePadGetReqState() to get the execution result of the request.

When the controller exits from pressure-sensitive mode, the controller ID returns to 0x73.

Return value

1: Request was accepted

Other than 1: Request was not accepted

See also

scePadEnterPressMode()

scePadGetReqState

Get execution result of asynchronous function

Library	*Introduced*	*Documentation last modified*
libpad	1.2	March 26, 2001

Syntax

int scePadGetReqState (

int *port,*	Controller port number
int *slot***)**	Slot number (fixed at 0, except when using a multitap)

Calling conditions

Can be called from a thread

Not multithread safe

Description

This function is used after an asynchronous function was executed to check whether or not processing was successful. An asynchronous function is one that terminates immediately but requires several frames to communicate with the controller on the IOP. The following functions are asynchronous functions:

scePadSetMainMode()
scePadSetActAlign()
scePadEnterPressMode()
scePadExitPressMode()

Return value

Table 8-1

Return Value	Description
scePadReqStateBusy	Executing
scePadReqStateFaild	Processing failed for some reason
scePadReqStateComplete	The function terminated normally

See also

scePadSetMainMode(), scePadSetActAlign(), scePadEnterPressMode(), scePadExitPressMode()

scePadGetSlotMax

Get maximum number of controller port slots

Library	*Introduced*	*Documentation last modified*
libpad	1.4	March 26, 2001

Syntax

int scePadGetSlotMax (

int *port)*	Controller port number (0 or 1)

Calling conditions

Can be called from a thread

Not multithread safe

Description

scePadGetSlotMax checks the maximum number of slots on the multi tap connected to the port which was opened using sceMtapPortOpen().

When the multi tap is not connected, 1 is returned. If the multitap is connected, but the port was not opened using sceMtapPortOpen(), 1 is returned.

Return value

Returns the maximum number of slots for the multitaps connected to the specified controller ports.

scePadGetState

Get controller connection state

Library	*Introduced*	*Documentation last modified*
libpad	1.1	March 26, 2001

Syntax

int scePadGetState(

int *port,*	Controller port number
int *slot***)**	Slot number (fixed at 0, except when using a multitap)

Calling conditions

Can be called from a thread

Not multithread safe

Description

Gets the connection state of the controller on the opened port.

When a port is opened, the IOP's padman.irx module independently monitors the connection and gets information for the connected controller.

Also, multiple communications are requried to change the controller state, and an interval of several frames is required until this is completed.

During this interval, the button state cannot be obtained, and application requests cannot be received. An application can use scePadGetState() to check the state of padman.irx processing. If the scePadGetState() return value is scePadStateStable or scePadStateFindCTP1, button information can be obtained from the controller.

Return value

Table 8-2: Controller connection state

Return Value	Meaning
scePadStateDiscon	Controller is not connected
scePadStateFindPad	Controller was not found (processing continuing)
scePadStateFindCTP1	Detected the CTP 1.0 controller
scePadStateExecCmd	Communicating with controller
scePadStateStable	Detected the CTP 2.0 controller
scePadStateError	Communication error detected

Among the states listed above, scePadStateFindCTP1 and scePadStateStable are the only ones in which a request from an application can be accepted or the button state can be obtained.

scePadInfoAct

Get actuator information

Library	*Introduced*	*Documentation last modified*
libpad	1.1	March 26, 2001

Syntax

int scePadInfoAct(

int *port,*	Controller port number
int *slot,*	Slot number (fixed at 0, except when using a multitap)
int *actno,*	Actuator number 0 to total number of actuators-1; or -1 to obtain the total number of actuators
int *term***)**	Term (return value reference (see table below) ignored if *actno* = -1)

Calling conditions

Valid only when scePadGetState() is scePadStateStable

Can be called from a thread

Not multithread safe

Description

Gets detailed information regarding actuators on the controller. When actno = -1, it also obtains the total number of actuators. This function is valid only when scePadGetState() = scePadStateStable.

Return value

The relationship between the *term* argument and the return value is as follows:

Table 8-3

term	Return value
InfoActFunc	Function number (1: continuous rotation vibration)
InfoActSub	Subfunction number (1: low-speed rotation, 2: high-speed rotation)
InfoActSize	Parameter data length (0: 1 bit (on/off only), 1 or more: number of bytes)
InfoActCurr	Current consumption capacity (10mA units)

In addition, if actno = -1, the return value contains the total number of actuators. In this case, the term argument is ignored. When the controller is not in READY state, or some error has occurred, 0 is returned.

scePadInfoComb

Get information about combination of actuators that can operate simultaneously

Library	*Introduced*	*Documentation last modified*
libpad	1.1	March 26, 2001

Syntax

int scePadInfoComb(

int *port,*	Controller port number
int *slot,*	Slot number (fixed at 0, except when using a multitap)
int *listno,*	List number of combination list 0 to total number of combination lists-1 or -1 to obtain the total number of combination lists
int *offs***)**	Offset within combination list 0 to total number of actuators in list-1 or -1 to obtain the total number of actuators in the list

Calling conditions

Valid only when scePadGetState() is scePadStateStable

Can be called from a thread

Not multithread safe

Description

Gets the combinations of actuators that are operable simultaneously. The number is limited by a number of factors such as the physical location of the actuators.

Return value

Table 8-4

listno	*offs*	Return value
-1	X	Total number of combination lists (n)
0 to (n-1)	-1	Total number of actuators in list (m)
0 to (n-1)	(0 to m-1)	Actuator number stored in offset position *offs* of list having 0 list number *listno*

When the controller is not in READY state, or some error has occurred, 0 is returned.

scePadInfoMode

Get information related to the controller mode

Library	*Introduced*	*Documentation last modified*
libpad	1.1	March 26, 2001

Syntax

int scePadInfoMode(

int *port,*	Controller port number
int *slot,*	Slot number (fixed at 0, except when using a multitap)
int *term,*	Item to be checked
int *offs)*	Offset in controller mode ID table that contains the controller mode ID to be checked

Calling conditions

Valid only when scePadGetState() is scePadStateStable or scePadStateFindCTP1

Can be called from a thread

Not multithread safe

Description

Allows the currently operating controller mode ID to be checked, controllers that are compatible or incompatible with the vibration function to be identified, and determines the controller mode ID of controllers with the vibration function. (The SCPH-1150 controller does not have a vibration function and is handled as an exception.)

InfoModeCurID will also work for PSCTP1.0 controllers. All other terms will be valid only for the PSCTP2.0 controller.

The permutations of mode IDs in the controller mode ID table depend on the controller type. The controller mode ID table for the DUALSHOCK 2 is shown below.

Table 8-5

Offset	Controller Mode ID
0	4
1	7

Return value

The return value for the various values of *term* is as follows:

Table 8-6

term	*offs*	Return value
InfoModeCurID	-1	Currently operating controller mode ID Valid no. of digits: 4 bits (same as the value of the button information's terminal type)
InfoModeCurExID	x	Mode ID of the currently operating controller for controllers with a vibration function Valid no. of digits: 16 bits (0 for SCPH-1150 or vibration function incompatibility)
InfoModeCurExOffs	x	Offset in table which contains the currently operating controller mode ID (0 for SCPH-1150 or vibration function incompatibility)
InfoModeIdTable	-1	Total number of controller mode IDs (n)
InfoModeIdTable	0~n-1	Controller mode ID stored at offset specified by offs, in the controller mode ID table (0 for SCPH-1150 or vibration function incompatibility)

When the controller is not in READY state, or some error has occurred, 0 is returned.

scePadInfoPressMode

Determine whether the connected controller supports pressure-sensitive functions

Library	*Introduced*	*Documentation last modified*
libpad	1.2	March 26, 2001

Syntax

int scePadInfoPressMode(

int *port,*	Controller port number
int *slot)*	Slot number (fixed at 0, except when using a multitap)

Calling conditions

Valid only when scePadGetState() is scePadStateStable

Can be called from a thread

Not multithread safe

Description

Determines whether the controller connected to the specified port supports pressure-sensitive functions.

Return value

1: Pressure-sensitive functions supported

Other than 1: Pressure-sensitive functions not supported

scePadInit

Initialize libpad controller library

Library	*Introduced*	*Documentation last modified*
libpad	1.1	March 26, 2001

Syntax

int scePadInit(	
int *mode***)**	Initialization mode (Currently fixed at 0.)

Calling conditions

Can be called from a thread

Not multithread safe (must be called in an interrupt-enabled state)

Description

Initializes the libpad controller library.

Return value

1: Terminated normally

Other than 1: Initialization failure

See also

scePadEnd()

scePadPortClose

Stop communication with the controller

Library	*Introduced*	*Documentation last modified*
libpad	1.2	March 26, 2001

Syntax

int scePadPortClose(	
int *port,*	Controller port number
int *slot)*	Slot number (fixed at 0, except when using a multitap)

Calling conditions

Can be called from a thread

Not multithread safe (must be called in an interrupt-enabled state)

Description

Closes the port that was opened with scePadPortOpen().

Subsequently, communication with the closed port is terminated.

Return value

1: Success

Other than 1: Failure

See also

scePadPortOpen()

scePadPortOpen

Begin communication with the controller

Library	*Introduced*	*Documentation last modified*
libpad	1.1	March 26, 2001

Syntax

int scePadPortOpen (

int *port,*	Controller port number (0 or 1)
int *slot,*	Slot number (fixed at 0, except when using a multi tap)
u_long128* *data***)**	Work buffer, store at 64 byte alignment (The required size is defined by the constant PadDmaBufferMax.)

Calling conditions

Can be called from a thread

Not multithread safe (must be called in an interrupt-enabled state)

Description

Opens the specified controller port.

When the controller port is opened, padman.irx monitors the controller connection. After the connection is made, the controller information is automatically obtained.

Button information can also be obtained. Once the controller port is open, a latency of several frames is needed before the button information becomes available. Monitor scePadGetState() to determine whether or not button information is available.

Return value

1: Request received.

Other than 1: Request not received.

See also

scePadPortClose()

scePadRead

Get button information

Library	*Introduced*	*Documentation last modified*
libpad	1.1	March 26, 2001

Syntax

int scePadRead(

int *port,*	Controller port number (0 or 1)
int *slot,*	Slot number (fixed at 0, except when using a multitap)
unsigned char* *data***)**	Pointer to buffer in which button information is stored. Buffer must be 32 bytes.

Calling conditions

Valid only when scePadGetState() is scePadStateStable or scePadStateFindCTP1

Can be called from a thread

Not multithread safe

Description

Gets the latest button information sent to the EE. The button information is sent only for the opened port. The contents of the buffer are shown below:

Table 8-7: Controller (digital) Data Array

Offset(bytes)	Contents
0	Successful communication: 0, otherwise: 0xff
1	High-order 4 bits: 0x4
	Low-order 4 bits: Data length/2
2,3	Digital button state (1: released, 0: pushed)

Table 8-8: DUALSHOCK Data Array

Offset (bytes)	Contents
0	Successful communication: 0, otherwise:0xff
1	High-order 4 bits: 0x7
	Low-order 4 bits: Data length/2
2,3	Digital button state (1: released, 0: pushed)
4	Analog stick right (X direction)
5	Analog stick right (Y direction)
6	Analog stick left (X direction)
7	Analog stick left (Y direction)

Table 8-9: Analog Joystick Data Array

Offset (bytes)	Contents
0	Successful communication: 0, otherwise: 0xff
1	High-order 4 bits: 0x5 Low-order 4 bits: Data length/2
2,3	Digital button state (1: released, 0: pushed)
4	Analog stick right (X direction)
5	Analog stick right (Y direction)
6	Analog stick left (X direction)
7	Analog stick left (Y direction)

Table 8-10: NeGcon Data Array

Offset (bytes)	Contents
0	Successful communication: 0, otherwise: 0xff
1	High-order 4 bits: 0x2
	Low-order 4 bits: Data length/2
2,3	Digital button state (1: released, 0: pushed)
4	Rotary part's analog data
5	I button analog data
6	II button analog data
7	L button analog data

Table 8-11: Namco Gun Controller (SLPH-00034) Data array

Offset (bytes)	Contents
0	Successful communication: 0, otherwise: 0xff
1	High-order 4 bits: 0x6 Low-order 4 bits:data length/2
2,3	Digital button state (1: released, 0: pushed)
4	Position X direction High-order byte
5	Position X direction Low-order byte
6	Position Y direction High-order byte
7	Position Y direction Low-order byte

Table 8-12: DUALSHOCK 2 Data array (in pressure sensitive mode)

Offset(bytes)	Contents
0	Successful communication: 0, otherwise: 0xff
1	High-order 4 bits: 0x7 Low-order 4 bits: Data length/2
2,3	Digital button state (1: released, 0: pushed)
4	Analog stick right (X direction)
5	Analog stick right (Y direction)
6	Analog stick left (X direction)
7	Analog stick left (Y direction)
8	Pressure sensitivity information(→)
9	Pressure sensitivity information(←)
10	Pressure sensitivity information(↑)
11	Pressure sensitivity information(↓)
12	Pressure sensitivity information(Δ)
13	Pressure sensitivity information(○)
14	Pressure sensitivity information(X)
15	Pressure sensitivity information(□)
16	Pressure sensitivity information(L1)
17	Pressure sensitivity information(R1)
18	Pressure sensitivity information(L2)
19	Pressure sensitivity information(R2)

Table 8-13: Button State Bit Assignments (Offset 2)

Bit offset	7	6	5	4	3	2	1	0
Controller (digital)	←	↓	→	↑	ST			SEL
DUALSHOCK	←	↓	→	↑	ST	R3	L3	SEL
Analog joystick	←	↓	→	↑	ST			SEL
NeGcon	←	↓	→	↑	ST			
Namco Gun Controller	←	↓	→	↑				

Table 8-14: Button State Bit Assignments (Offset 3)

Bit offset	7	6	5	4	3	2	1	0
Controller (digital)	□	X	○	Δ	R1	L1	R2	L2
DUALSHOCK	□	X	○	Δ	R1	L1	R2	L2
Analog joystick	□	X	○	Δ	R1	L1	R2	L2
NeGcon			A	B	R			
Namco Gun Controller		B	TRG					

Return value

0: Failed to get information.

Other than 0: Length of obtained data (currently fixed at 32)

scePadReqIntToStr

Get character string corresponding to execution result of asynchronous function (for debugging)

Library	*Introduced*	*Documentation last modified*
libpad	1.2	March 26, 2001

Syntax

void scePadReqIntToStr (

int *state,*	Execution result code
char* *str)*	Pointer to buffer used for storing character string (required size is max. 16 bytes)

Calling conditions

Can be called from a thread

Not multithread safe

Description

Converts the execution/result code of an asynchronous function obtained using scePadGetReqState() to a character string.

Return value

None

scePadSetActAlign

Send details of actuator parameters to the controller

Library	*Introduced*	*Documentation last modified*
libpad	1.1	March 26, 2001

Syntax

int scePadSetActAlign(	
int *port,*	Controller port number
int *slot,*	Slot number (fixed at 0, except when using a multitap)
const unsigned char* *data***)**	Sent details of actuator parameters (6 bytes)

Calling conditions

Valid only when scePadGetState() is scePadStateStable

Can be called from a thread

Not multithread safe (must be called in an interrupt-enabled state)

Description

Notifies the controller of the locations where the actuator parameters are stored in the send buffer, by writing the actuator numbers in a 6-byte array.

In the following example, send buffer offset 0 is used as actuator no. 0, offset 1 is used as actuator no. 1, and other data are not used. (The actuator numbers are stored in valid locations, and unused locations are set to 0xff.)

Table 8-15

Offset:	0	1	2	3	4	5
Data contents:	0x00	0x01	0xFF	0xFF	0xFF	0xFF

Since communication with the controller is carried out when this function is executed, other requests cannot be received for several frames.

Completion of processing can be monitored using scePadGetState() or by checking the scePadGetReqState() result.

Details of the actuator parameters set by this function become ineffective if the controller is disconnected or the controller mode is changed.

Return value

1: Request received.

Other than 1: Request not received.

scePadSetActDirect

Send actuator parameters to the controller

Library	*Introduced*	*Documentation last modified*
libpad	1.1	March 26, 2001

Syntax

int scePadSetActDirect (

int *port,*	Controller port number
int *slot,*	Slot number (fixed at 0, except when using a multitap)
const unsigned char* *data***)**	Starting address of transmit data 6 bytes of transmit data should be provided

Calling conditions

Valid only when scePadGetState() is scePadStateStable

Can be called from a thread

Not multithread safe (must be called in an interrupt-enabled state)

Description

Sends the transmit data to the IOP for operating an actuator. The transmitted data is sent to the controller during the next VBlank interrupt.

In addition to specifying the send buffer with scePadSetActDirect(), controlling the actuator requires using scePadSetActAlign() to notify the controller regarding which offsets in the send buffer should be treated as actuator parameters.

The data length accepted by the actuator can be determined with scePadInfoAct(). With the DUALSHOCK 2, the data lengths of actuator number 0 and 1 are 1 bit and 1 byte, respectively.

Thus, if scePadSetActAlign() is used with a parameter for actuator number 0 at offset 0 of the send buffer and a parameter for actuator number 1 at offset 1, sending a 0 or 1 at offset 0 and sending values of 0-255 at offset 1 will allow the actuators to be controlled.

An actuator will be stopped when its parameter is 0 and will run faster for larger values.

Table 8-16: DUALSHOCK 2 actuator settings

small motor	0=stop, 1=run
large motor	0-255 larger values give higher speed

Return value

1: Successful transmission

Other than 1: Failure

scePadSetMainMode

Change controller mode / lock changeover switch

Library	*Introduced*	*Documentation last modified*
libpad	1.1	March 26, 2001

Syntax

int scePadSetMainMode(

int *port,*	Controller port number
int *slot,*	Slot number (fixed at 0, except when using a multitap)
int *offs,*	Offset in controller mode ID table containing the switched controller mode
int *lock* **)**	Lock/Unlock analog switch 0, 1: keep current lock/unlock status 2: Unlock 3: Lock

Calling conditions

Valid only when scePadGetState() is scePadStateStable

Can be called from a thread

Not multithread safe (must be called in an interrupt-enabled state)

Description

Switches controller modes, and switches between the locked and unlocked states of the mode switch button on the controller's main unit.

When this function is executed, other requests cannot be received immediately, and controller button information will not be available for several frames. The completion of processing should be checked using scePadGetState() or the result of scePadGetReqState().

In addition, when the controller mode is switched, previously set actuator settings become invalid.

Return value

1: Request received

Other than 1: Request not received.

scePadSetWarrningLevel

Suppress warning messages

Library	*Introduced*	*Documentation last modified*
libpad	2.2	March 26, 2001

Syntax

int scePadSetWarrningLevel (

int *level***)**	0: Suppress warning messages Other than 0: Cancels the suppression of warning messages (initial value)

Calling conditions

Can be called from a thread

Not multithread safe (must be called in interrupt-enabled state)

Description

Suppresses the warning messages output by libpad and padman on the console of dsedb, dsidb etc.

Because the suppression gets cancelled when scePadInit() is executed, execute this function after scePadInit().

Return value

Always 1

scePadStateIntToStr

Get character string corresponding to controller state (for debugging)

Library	*Introduced*	*Documentation last modified*
libpad	1.2	March 26, 2001

Syntax

void scePadStateIntToStr (

int *state,*	State code
char* *str***)**	Pointer to buffer used for storing the string (required size is max. 16 bytes)

Calling conditions

Can be called from a thread

Not multithread safe

Description

Converts the controller state code obtained using scePadGetState() to a character string.

Return value

None

Chapter 9: Controller Library 2
Table of Contents

Structures

scePad2SocketParam

Parameters used when creating a virtual socket

Library	*Introduced*	*Documentation last modified*
libpad2	2.4	October 1, 2001

Structure

typedef struct {	
unsigned int *option***;**	Socket option
int *port***;**	Port number
int *slot***;**	Slot number
int *number***;**	Driver number
unsigned char *name***[SCE_PAD2_MAX_DEVICE_NAME];**	Device name
} scePad2SocketParam;	

Description

This is a structure for setting parameters that will be used when scePad2CreateSocket() is executed.

Setting these parameters enables the socket that is created to be linked to a specific device driver.

The *option* member is specified as the logical OR of the following options.

Table 9-1

Macro Name	Function
SCE_PAD2_SPECIFIC_PORT	Only a device driver that controls a device on a specific port will be used as the link target
SCE_PAD2_SPECIFIC_DRIVER_NUMBER	Only a device driver having a specific number will be used as the link target
SCE_PAD2_SPECIFIC_DEVICE_NAME	Only a device driver that controls a device with a specific name will be used as the link target

When SCE_PAD2_SPECIFIC_PORT is specified for *option*, *port* and *slot* should be set to the specified port. Currently, the following values can be set for *port*.

Table 9-2

Macro Name	Location	slot Specification
SCE_PAD2_PORT_1C	Front controller port 1	Can be specified
SCE_PAD2_PORT_2C	Front controller port 2	Can be specified
SCE_PAD2_PORT_USB	USB port	Cannot be specified (must be 0)

slot is valid only if a controller port is specified. If a multitap is used, the multitap offset can be specified for *slot*. However, since multitaps are not currently supported, specifying a value other than zero is meaningless.

Currently, operation is undefined for the driver number and controller name options. Members that are associated with an unspecified option are not referenced. In addition, if the *option* member isn't specified, all device controller drivers will be link targets.

Functions

scePad2CreateSocket

Create virtual socket

Library	*Introduced*	*Documentation last modified*
libpad2	2.4	January 4, 2002

Syntax

int scePad2CreateSocket(	
scePad2SocketParam* *socket***,**	Starting address of scePad2SocketParam structure
u_long128* *addr***)**	Starting address of work buffer (required size is defined by SCE_PAD2_DMA_BUFFER_MAX)

Calling conditions

Can be called from a thread

Not multi-thread safe (must be called in an interrupt-enabled state)

Description

This function creates a virtual socket. To assign conditions to the virtual socket, set parameters in the scePad2SocketParam structure. If no conditions are to be assigned, NULL should be specified for the *socket* argument.

To create a virtual socket, a work buffer must be prepared to acquire the data that is DMA transferred from the device driver. The buffer, which has a size of 256 bytes, must be aligned on a 64-byte boundary.

Return value

>=0:	Socket number
<0:	Create processing failed

See also

scePad2SocketParam

scePad2DeleteSocket

Delete virtual socket

Library	*Introduced*	*Documentation last modified*
libpad2	2.4	January 4, 2002

Syntax

int scePad2DeleteSocket(

int *socket_number***)**	Socket number

Calling conditions

Can be called from a thread

Not multi-thread safe (must be called in an interrupt-enabled state)

Description

This function deletes the specified virtual socket.

Return value

1:	Processing succeeded
Other than 1:	Processing failed

scePad2End
Controller library termination processing

Library	*Introduced*	*Documentation last modified*
libpad2	2.4	January 4, 2002

Syntax
int scePad2End(void)

Calling conditions
Can be called from a thread

Not multi-thread safe (must be called in an interrupt-enabled state)

Description
This function terminates the controller library.

Return value

1:	Normal termination
Other than 1:	Termination processing failed

scePad2GetButtonInfo

Get button value

Library	*Introduced*	*Documentation last modified*
libpad2	2.4	January 4, 2002

Syntax

int scePad2GetButtonInfo(

int *socket_number*,	Socket number
unsigned char* *data*,	Starting address of acquired button data
int *button*)	Button type

Calling conditions

Can be called from a thread

Not multi-thread safe

Description

This function uses button information that was acquired with the scePad2Read() function to get the value of the button specified by the *button* argument. The macro name for each button is shown below.

Table 9-3

No.	Button	Macro Name
0	SELECT	SCE_PAD2_SELECT
1	L3	SCE_PAD2_L3
2	R3	SCE_PAD2_R3
3	START	SCE_PAD2_START
4	Up (direction keys)	SCE_PAD2_UP
5	Right	SCE_PAD2_RIGHT
6	Down	SCE_PAD2_DOWN
7	Left	SCE_PAD2_LEFT
8	L2	SCE_PAD2_L2
9	R2	SCE_PAD2_R2
10	L1	SCE_PAD2_L1
11	R1	SCE_PAD2_R1
12	Triangle	SCE_PAD2_TRIANGLE
13	Circle	SCE_PAD2_CIRCLE
14	Cross	SCE_PAD2_CROSS
15	Square	SCE_PAD2_SQUARE
16	Analog stick right (X-direction)	SCE_PAD2_STICK_RX
17	Analog stick right (Y-direction)	SCE_PAD2_STICK_RY
18	Analog stick left (X-direction)	SCE_PAD2_STICK_LX
19	Analog stick left (Y-direction)	SCE_PAD2_STICK_LY

No.	Button	Macro Name
20	Pressure-sensitive information (Right)	SCE_PAD2_RIGHT
21	Pressure-sensitive information (Left)	SCE_PAD2_LEFT
22	Pressure-sensitive information (Up)	SCE_PAD2_UP
23	Pressure-sensitive information (Down)	SCE_PAD2_DOWN
24	Pressure-sensitive information (Triangle)	SCE_PAD2_TRIANGLE
25	Pressure-sensitive information (Circle)	SCE_PAD2_CIRCLE
26	Pressure-sensitive information (Cross)	SCE_PAD2_CROSS
27	Pressure-sensitive information (Square)	SCE_PAD2_SQUARE
28	Pressure-sensitive information (L1)	SCE_PAD2_L1
29	Pressure-sensitive information (R1)	SCE_PAD2_R1
30	Pressure-sensitive information (L2)	SCE_PAD2_L2
31	Pressure-sensitive information (R2)	SCE_PAD2_R2

Return value

>=0:	Value of specified button
<0:	Get processing failed

See also

scePad2Read()

scePad2GetButtonProfile

Get button profile

Library	*Introduced*	*Documentation last modified*
libpad2	2.4	January 4, 2002

Syntax

int scePad2GetButtonProfile (

int *socket_number*,	Socket number
unsigned char* *profile***)**	Starting address of profile to be acquired

Calling conditions

Can be called from a thread

Not multi-thread safe (must be called in an interrupt-enabled state)

Description

This function gets the button profile of the controller that is linked to the socket.

Return value

>=0:	Size of acquired profile
<0:	Processing failed

scePad2GetState

Get controller state

Library	*Introduced*	*Documentation last modified*
libpad2	2.4	January 4, 2002

Syntax

int scePad2GetState(

int *socket_number***)**	Socket number

Calling conditions

Can be called from a thread

Not multi-thread safe (must be called in an interrupt-enabled state)

Description

This function gets the state of the linked controller. The state numbers are shown below.

Button information can be acquired when the state is scePad2StateStable.

Table 9-4

State No.	State Name	Remarks
0	scePad2StateNoLink	Not linked to a socket
1	scePad2StateStable	Can communicate with controller
2	scePad2StateExecCmd	Command is being executed
3	scePad2StateError	Communication error

Return value

>=0:	Controller state number
<0:	Get processing failed

scePad2Init

Initialize controller library

Library	*Introduced*	*Documentation last modified*
libpad2	2.4	January 4, 2002

Syntax

int scePad2Init(int *mode* **)**	Initialization mode (currently, this is always 0)

Calling conditions

Can be called from a thread

Not multi-thread safe

Description

This function initializes the controller library.

Return value

1:	Initialization succeeded
Other than 1:	Initialization failed

scePad2Read

Get button information

Library	*Introduced*	*Documentation last modified*
libpad2	2.4	January 4, 2002

Syntax

int scePad2Read(	
int *socket_number***,**	Socket number
unsigned char* *data***)**	Starting address of data to be obtained

Calling conditions

Can be called from a thread

Not multi-thread safe (must be called in an interrupt-enabled state)

Description

This function gets controller button information.

Return value

>=0:	Size of acquired data
<0:	Get processing failed

scePad2StateIntToStr

Get string corresponding to controller state (for debugging)

Library	*Introduced*	*Documentation last modified*
libpad2	2.4	January 4, 2002

Syntax

void scePad2StateIntToStr(

int *state*,	State number
unsigned char* *str*)	Pointer to string storage buffer (maximum size will be at most 16 bytes)

Calling conditions

Can be called from a thread

Not multi-thread safe

Description

This function converts the controller state number that was obtained with the scePad2GetState() function to a character string.

Return value

None

Chapter 10: USB Keyboard Library
Table of Contents

Structures

USBKBDATA_t

Keyboard data

Library	*Introduced*	*Last Modified*
libusbkb	2.2	October 11, 2001

Structure

```
#define USBKB_MAX_KEYCODES 62
typedef struct {
    u_int led;                                  LED lighting state (see table below for details)
    u_int mkey;                                 Modifier key state (see table below for details)
    int len;                                    Number of key codes (0 means no data)
    u_short keycode[USBKB_MAX_KEYCODES];        Pointer to key codes
} USBKBDATA_t;
```

Table 10-1: ***led*** **(LED lighting state)**

Refer to the following macros for the meaning of each bit

```
#define USBKB_LED_NUM_LOCK      (1<<0)  /* 0:OFF 1:ON */
#define USBKB_LED_CAPS_LOCK     (1<<1)  /* 0:OFF 1:ON */
#define USBKB_LED_SCROLL_LOCK   (1<<2)  /* 0:OFF 1:ON */
#define USBKB_LED_COMPOSE       (1<<3)  /* 0:OFF 1:ON */
#define USBKB_LED_KANA          (1<<4)  /* 0:OFF 1:ON */
```

(*) For a Macintosh keyboard, the ALT and WIN keys correspond to the OPTION and APPLE keys.

Table 10-2: ***mkey*** **(Modifier key status)**

Refer to the following macros for the meaning of each bit

```
#define USBKB_MKEY_L_CTRL       (1<<0)  /* 0:Release 1:Push */
#define USBKB_MKEY_L_SHIFT      (1<<1)  /* 0:Release 1:Push */
#define USBKB_MKEY_L_ALT        (1<<2)  /* 0:Release 1:Push */
#define USBKB_MKEY_L_WIN        (1<<3)  /* 0:Release 1:Push */
#define USBKB_MKEY_R_CTRL       (1<<4)  /* 0:Release 1:Push */
#define USBKB_MKEY_R_SHIFT      (1<<5)  /* 0:Release 1:Push */
#define USBKB_MKEY_R_ALT        (1<<6)  /* 0:Release 1:Push */
#define USBKB_MKEY_R_WIN        (1<<7)  /* 0:Release 1:Push */
```

(*) For a Macintosh keyboard, the ALT and WIN keys correspond to the OPTION and APPLE keys.

Description

This is a structure for obtaining data (key codes) from the keyboard using sceUsbKbRead().

The valid key codes are from *keycode*[0] to *keycode*[len-1].

A key code is obtained using the original key code of the USB specification or one that is a converted ASCII code, based on the original key code. The type of key code is selected with sceUsbKbSetCodeType().

See also

sceUsbKbRead(), sceUsbKbSetCodeType()

USBKBINFO_t

Keyboard connection information

Library	*Introduced*	*Last Modified*
libusbkb	2.2	October 11, 2001

Structure

#define MAX_STATUS 127	
typedef struct {	
int *max_connect;*	Maximum number of connections *m*
int *now_connect;*	Current number of connections
u_char *status* **[USBKB_MAX_STATUS];**	Pointer to connection state information
} USBKBINFO_t;	

Description

This is a structure for obtaining the keyboard connection state using sceUsbKbGetInfo(). The connection state information is returned as an array, as shown below.

Table 10-3

Index	Contents
status [0]	Keyboard No. 0 connection information 0: Not connected 1: Connected
:	:
status [m-1]	Keyboard No. (m-1) connection information 0: Not connected 1: Connected
status [m]	Undefined
:	:
status **[USBKB_MAX_STATUS-1]**	Undefined

See also

sceUsbKbGetInfo()

Functions

sceUsbKbClearRbuf

Clear ring buffer

Library	*Introduced*	*Last Modified*
libusbkb	2.3.4	August 31, 2001

Syntax

#include <libusbkb.h>

int sceUsbKbClearRbuf (	
u_int *no***)**	Keyboard No.

Calling conditions

Can be called from a thread

Not multithread safe (must be called in interrupt-enabled state)

Description

usbkb.irx has a ring buffer for storing key data.

This function clears that buffer.

If sceUsbKbRead() is not called for long periods of time, key data will accumulate in the ring buffer of usbkb.irx.

sceUsbKbClearRbuf() is used to clear this accumulated ring buffer data.

Return value

USBKB_OK	Normal termination
USBKB_E_PAR1	Illegal specification of "no" argument
USBKB_E_SIF	SIF error

See also

sceUsbKbRead ()

sceUsbKbCnvRawCode

Convert raw key code

Library	*Introduced*	*Last Modified*
libusbkb	2.3	July 2, 2001

Syntax

#include <libusbkb.h>

int sceUsbKbCnvRawCode (

int *arrange,*	Key arrangement	
	ARRANGEMENT_101	101/104 keyboard
	ARRANGEMENT_106	106/109 keyboard
	ARRANGEMENT_106_KANA	106/109 keyboard (kana state)
u_int *mkey,*	Same as mkey member of USBKBDATA_t structure	
u_int *led,*	Same as led member of USBKBDATA_t structure	
u_short *rawcode***)**	Raw code to be converted	

Calling conditions

Can be called from an interrupt handler

Can be called from a thread

Multithread safe (must be called in interrupt-enabled state)

Description

This function converts a raw key code (USB device key code) based on various kinds of information such as key arrangement, Modifier key, and LED state.

It is used when CODETYPE_RAW is specified in sceUsbKbSetCodeType().

This function is needed to use an FEP.

When an FEP is used, a shortcut such as Ctrl+U may need to be entered while inputting Japanese. In this case, it may be more convenient to use the raw code.

Depending on the FEP state, you can use this function to convert the raw key code and pass the converted result to the FEP.

Return value

Converted key code

See also

USBKBDATA_t, sceUsbKbSetCodeType ()

sceUsbKbEnd

End keyboard library

Library	*Introduced*	*Last Modified*
libusbkb	2.3	July 2, 2001

Syntax

#include <libusbkb.h>

int sceUsbKbEnd(void)

Calling conditions

Can be called from a thread

Not multithread safe (must be called in interrupt-enabled state)

Description

This function ends the keyboard library.

It frees the interrupt handler and the semaphores and memory that were allocated by sceUsbKbInit().

Return value

USBKB_OK	Normal termination
USBKB_NG	Freeing of resources failed

See also

sceUsbKbInit()

sceUsbKbGetInfo

Get keyboard connection information (asynchronous)

Library	*Introduced*	*Last Modified*
libusbkb	2.2	March 23, 2001

Syntax

#include <libusbkb.h>
int sceUsbKbGetInfo(

USBKEYBDINFO_t **info***)**	Keyboard connection information

Calling conditions

Can be called from a thread.

Not multithread safe (must be called in interrupt-enabled state).

Description

This function gets keyboard connection information.

This function is executed asynchronously, and the contents of **info* are undefined when control returns. **info* should be read after the end of execution is detected with sceUsbKbSync(). Whether or not the keyboard connection information could be obtained is indicated by the value returned in the result argument of sceUsbKbSync(), as follows:

[result of sceUsbKbSync()]
USBKB_OK (Normal termination)
USBKB_NG (Abnormal termination)

Return value

USBKB_OK (Normal termination)

USBKB_NG (Abnormal termination)

See also

USBKEYBDINFO_t, USBsceUsbKbSync()

sceUsbKbGetLocation

Get keyboard connection location (asynchronous)

Library	*Introduced*	*Last Modified*
libusbkb	2.2	March 23, 2001

Syntax

#include <libusbkb.h>

int sceUsbKbGetLocation(	
int *no,*	Keyboard No.
u_char **location*)	Keyboard connection location information

Calling conditions

Can be called from a thread.

Not multithread safe (must be called in interrupt-enabled state).

Description

This function gets information about the location on the USB bus where the keyboard specified by the *no* argument is connected. The location information is returned in **location* as follows.

Table 10-4

location[0]	Port No. of host (RootHub) (0 if no keyboard is connected)
location[1]	Port No. of first stage HUB (0 if no keyboard is connected)
location[2]	Port No. of second stage HUB (0 if no keyboard is connected)
location[3]	Port No. of third stage HUB (0 if no keyboard is connected)
location[4]	Port No. of fourth stage HUB (0 if no keyboard is connected)
location[5]	Port No. of fifth stage HUB (0 if no keyboard is connected)
location[6]	Always 0

Since this is an asynchronous function, its completion must be detected using sceUsbKbSync(). Whether or not the location information could be obtained is indicated by the value returned in the result argument of sceUsbKbSync(), as follows:

[result of sceUsbKbSync()]
USBKB_OK (Normal termination)
USBKB_NG (Abnormal termination)

Return value

USBKB_OK	Normal termination
USBKB_E_PAR1	Invalid specification for *no*
USBKB_E_SIF	SIF error

See also

sceUsbKbSync()

sceUsbKbInit

Initialize library

Library	*Introduced*	*Last Modified*
libusbkb	2.2	July 2, 2001

Syntax

#include <libusbkb.h>	
int sceUsbKbInit(	
int **max_connect*)	Maximum number of connections (same meaning as *max_connect* of USBKBINFO_t)

Calling conditions

Can be called from a thread

Not multithread safe (must be called in interrupt-enabled state)

Description

This function obtains RPC information, allocates resources and initializes settings.

The settings after this function is called are as follows:

LED state:	Maintain current state
LED lighting mode:	AUTO1 mode (see sceUsbKbSetLEDMode())
Key repeat:	OFF
Key code:	ASCII
Key arrangement:	106-key keyboard
Read mode:	Character input mode

Notes

1. When the library is initialized, the following resources are allocated.
   - One semaphore is required.
   - One V-BLANK end interrupt handler is registered.
   - (54 bytes x Maximum number of possible connections) of memory are allocated.
     Use sceUsbKbEnd() to free the above resources.
2. Do not call sceUsbKbInit() twice consecutively.
   If you want to call sceUsbKbInit() again, first call sceUsbKbEnd() to free resources.

Return value

USBKB_OK	(Normal termination)
USBKB_NG	(Abnormal termination)

See also

sceUsbKbEnd()

sceUsbKbRead

Read keyboard data (asynchronous)

Library	*Introduced*	*Last Modified*
libusbkb	2.2	August 31, 2001

Syntax

#include <libusbkb.h>	
int sceUsbKbRead(	
u_int *no,*	Keyboard No.
USBKEYBDDATA_t **data***)**	Pointer to key data structure

Calling conditions

Can be called from a thread.

Not multithread safe (must be called in interrupt-enabled state).

Description

This function gets data that is stored in the ring buffer of usbkb.irx for the keyboard specified by the *no* argument. If there is no data, 0 is stored in *data->len*.

Since this is an asynchronous function, its completion must be detected with sceUsbKbSync(). Whether or not the keyboard data could be obtained is indicated by the value returned in the *result* argument of sceUsbKbSync() as follows:

[result of sceUsbKbSync()]
USBKB_OK (Normal termination)
USBKB_NG (Abnormal termination: e.g. When a disconnected keyboard was accessed)

Use sceUsbKbClearRbuf() to clear the usbkb.irx ring buffer.

Return value

USBKB_OK (Normal termination)

USBKB_E_PAR1 (Invalid specification for *no*)

USBKB_E_SIF (SIF error)

See also

USBKEYBDDATA_t, sceUsbKbSync(), sceUsbKbReadMode(), sceUsbKbClearRbuf()

sceUsbKbSetArrangement

Set key arrangement

Library	*Introduced*	*Last Modified*
libusbkb	2.2	July 2, 2001

Syntax

#include <libusbkb.h>

int sceUsbKbSetArrangement(

int *no,*	Keyboard No.
int *arrange*)	Key arrangement ARRANGEMENT_101 101- or 104-key keyboard ARRANGEMENT_106 106- or 109-key keyboard ARRANGEMENT_106_KANA 106- or 109-key keyboard (kana state)

Calling conditions

Can be called from a thread.

Not multithread safe.

Description

This function switches the key arrangement data of the keyboard specified by the *no* argument.

Since the key arrangement cannot be automatically determined in the USB specification, this function should be used to switch between the 101- or 106-key keyboard (as specified by the user).

Return value

USBKB_OK	Normal termination
USBKB_E_PAR1	Invalid specification for *no*
USBKB_E_PAR2	Invalid specification of *arrange*

sceUsbKbSetCodeType

Set key code format

Library	*Introduced*	*Last Modified*
libusbkb	2.2	March 23, 2001

Syntax

#include <libusbkb.h>

int sceUsbKbSetCodeType(

int *no,*	Keyboard No.	
int *type***)**	Code type setting	
	CODETYPE_RAW	USB device code as is
	CODETYPE_ASCII	Converted to ASCII code

Calling conditions

Can be called from a thread.

Not multithread safe.

Description

This function sets the type of key code that is stored in the keycode member of the USBKEYBDDATA_t structure for the keyboard specified by the *no* argument.

If CODETYPE_RAW is specified for the *type* argument, the key code that is returned by the USB device will be stored as is. If CODETYPE_ASCII is specified, the code that has been converted to ASCII with the states of the Shift key and CAPSLOCK-LED taken into account, will be stored. However, key codes for keys on the numeric key pad and those for non-ASCII characters are handled differently.

Return value

USBKB_OK	Normal termination
USBKB_E_PAR1	Invalid specification for *no*
USBKB_E_PAR2	Invalid specification of *type*

sceUsbKbSetLEDMode

Set LED lighting mode (asynchronous)

Library	*Introduced*	*Last Modified*
libusbkb	2.2	July 2, 2001

Syntax

#include <libusbkb.h>

int sceUsbKbSetLEDMode(

int *no,*	Keyboard No.
int *mode***)**	LED lighting control mode

Calling conditions

Can be called from a thread

Not multithread safe (must be called in interrupt-enabled state)

Description

This function sets whether or not the illumination and extinguishing of the LEDs on the keyboard specified by the *no* argument are to be automatically controlled.

Table 10-5

mode	Control mode
LED_MODE_AUTO1	Automatically control the NumLock, CapsLock, ScrollLock LEDs
(*) Default setting	(Light CAPSLOCK-LED by pressing CAPSLOCK key)
LED_MODE_AUTO2	Automatically control the NumLock, CapsLock, ScrollLock LEDs (Light CAPSLOCK-LED by pressing Shift+CAPSLOCK key)
LED_MODE_MANUAL	Manually control all LEDs from EE application

For LED_MODE_AUTO1 and LED_MODE_AUTO2, the IOP module usbkb.irx controls the LEDs. For LED_MODE_MANUAL, the LEDs must be controlled on the EE. For normal use, LED_MODE_AUTO1 or LED_MODE_AUTO2 should be selected.

Since this is an asynchronous function, its completion must be detected with sceUsbKbSync(). Whether or not the LED lighting mode could be set is indicated by the value returned in the result argument of sceUsbKbSync() as follows:

[result of sceUsbKbSync()]
USBKB_OK (Normal termination)
USBKB_NG (Abnormal termination)

Return value

USBKB_OK	Normal termination
USBKB_E_PAR1	Invalid specification for *no*
USBKB_E_PAR2	Parameter 2 error
USBKB_E_SIF	SIF error

See also

sceUsbKbSync()

sceUsbKbSetLEDStatus

Control LED lighting (asynchronous)

Library	*Introduced*	*Last Modified*
libusbkb	2.2	March 23, 2001

Syntax

#include <libusbkb.h>

int sceUsbKbSetLEDStatus(

int *no,*	Keyboard No.
u_char *led***)**	LED state to be set

Calling conditions

Can be called from a thread.

Not multithread safe (must be called in interrupt-enabled state).

Description

This function changes the state of the LEDs on the keyboard specified by the *no* argument. It is used for LED control when you want to forcibly light up the LEDs when software is started up or when manual control is selected by sceUsbKbSetLEDMode(). For details about values to be specified for the *led* argument, see the *led* member of the USBKEYBDDATA_t structure.

Since this is an asynchronous function, its completion must be detected with sceUsbKbSync(). Whether or not the state could be set is indicated by the value returned in the *result* argument of sceUsbKbSync() as follows:

[result of sceUsbKbSync()]
SBKB_OK (Normal termination)
SBKB_NG (Abnormal termination)

Return value

USBKB_OK	Normal termination
USBKB_E_PAR1	Invalid specification for *no*
USBKB_E_SIF	SIF error

See also

USBKEYBDDATA_t, sceUsbKbSync()

sceUsbKbSetReadMode

Set keyboard data read mode

Library	*Introduced*	*Last Modified*
libusbkb	2.3	July 2, 2001

Syntax

#include <libusbkb.h>

int sceUsbKbSetReadMode(

u_int *no,*	Keyboard No.	
int *rmode*)	Mode for reading key data	
	USBKB_RMODE_INPUTCHAR	Character input mode
	USBKB_RMODE_PACKET	Packet mode

Calling conditions

Can be called from a thread

Not multithread safe (must be called in interrupt-enabled state)

Description

This function sets the method that sceUsbKbRead() uses for storing the key code in the keycode member of USBKEYBDDATA_t, for the USB keyboard specified by the *no* argument.

If *rmode* is set to USBKB_RMODE_INPUTCHAR, the mode is suitable for character input. When this mode is used, character input can easily be implemented if the key code that was entered in the keycode member can be used as is.

If *rmode* is set to USBKB_RMODE_PACKET, the mode enables the keyboard to be handled as a keypad.

Normally, a data packet that is obtained from a USB keyboard includes all key codes that were pressed simultaneously. In this mode, all key codes that were pressed simultaneously are stored in the keycode member.

However, in this mode, key repeat can no longer be used.

Return value

USBKB_OK	Normal termination
USBKB_E_PAR1	Specification of *no* is illegal
USBKB_E_PAR2	Specification of *rmode* is illegal

See also

USBKEYBDDATA_t, sceUsbKbRead()

sceUsbKbSetRepeat

Set key repeat mode

Library	*Introduced*	*Last Modified*
libusbkb	2.2	March 23, 2001

Syntax

#include <libusbkb.h>

int sceUsbKbSetRepeat(

int *no,*	Keyboard No
int *sta_time,*	Repeat starting time (VSync units: 0 means no key repeat)
int *interval*)	Repeat interval (VSync units: 0 means no key repeat)

Calling conditions

Can be called from a thread.

Not multithread safe.

Description

This function sets the key repeat mode of the keyboard specified by the *no* argument.

Once the time specified by the *sta_time* argument elapses after a given key is pressed, the key code will be repeatedly generated at the interval specified by the *interval* argument as long as that key continues to be held down. If 0 is specified for either the *sta_time* or *interval* arguments, key repeat will not function.

Return value

USBKB_OK (Normal termination)

USBKB_E_PAR1 (Invalid specification for *no*)

USBKB_E_PAR2 (Invalid specification of *sta_time*)

sceUsbKbSync

Wait for completion of asynchronous function processing

Library	*Introduced*	*Last Modified*
libusbkb	2.2	July 2, 2001

Syntax

#include <libusbkb.h>	
int sceUsbKbSync(	
int *mode,*	USBKB_WAIT: Blocking USBKB_NO_WAIT: Non-blocking
int **result***)**	Pointer where asynchronous function result is to be stored

Calling conditions

Can be called from a thread.

Not multithread safe (must be called in interrupt-enabled state).

Description

This function waits for the completion of the execution of an asynchronous function such as sceUsbKbRead().

If USBKB_WAIT is specified for the *mode* argument, this function uses a semaphore to wait for the completion of an executing asynchronous function and returns when execution has completed. Consequently, other threads cannot run while execution is waiting to complete. If USBKB_NO_WAIT is specified, this function checks the execution state of the asynchronous function and returns immediately.

sceUsbKbSync() should be used on a one-to-one basis with asynchronous functions.

Return value

USBKB_DONE	Completed
USBKB_EXEC	Executing
USBKB_E_PARA1	Invalid specification for mode

Chapter 11: Vibration Library
Table of Contents

Functions

sceVibGetProfile

Get vibration profile

Library	*Introduced*	*Documentation last modified*
libvib	2.4	January 4, 2002

Syntax

int sceVibGetProfile (	
int *socket_number*,	Socket number
unsigned *char* profile*)	Starting address of profile to be acquired

Calling conditions

Can be called from a thread

Not multithread safe (must be called in an interrupt-enabled state)

Description

This function gets the vibration profile of the controller that is linked to the socket.

The current vibration profile is shown below. Actuators that exist in the controller will have their corresponding bits set to 1. If they do not exist, they will be set to 0.

Table 11-1

Byte	Bit	Feature	Size (bits)
0	0	Small motor	1
	1	Large motor	8
	2	(Subsequent	
	3	bits are	
	4	undefined)	
	5		
	6		
	7		

Return value

>=0:	Size of acquired profile
<0:	Processing failed

sceVibSetActParam

Set parameters for actuators

Library	*Introduced*	*Documentation last modified*
libvib	2.4	January 4, 2002

Syntax

int sceVibSetActParam (

int *socket_number***,**	Socket number
int *profile_size***,**	Profile size to be sent
unsigned char* *profile***,**	Starting address of profile to be sent
int *data_size***,**	Send data size
unsigned char* *data***)**	Starting address of send data

Calling conditions

Can be called from a thread

Not multithread safe (must be called in an interrupt-enabled state)

Description

This function sets parameters for the actuators.

The bits for actuators whose parameters are to be changed in the profile to be sent are each set to 1, and the parameters of those actuators are stored so that they are pre-packed in the send data. After the data is sent, the device driver sets the parameters of each actuator from the send profile and send data.

(Sample send data) Setting a value for the large motor

Table 11-2: Profile Data

Byte	0
Bit	7 to 0
Bit Pattern	00000010

Figure 11-1: Vibration Data

Byte	Bit							
	7	6	5	4	3	2	1	0
0	Large motor set value							

(Sample send data 2) Simultaneously setting values for the large and small motors

Table 11-3: Profile Data

Byte	0
Bit	7 to 0
Bit Pattern	00000011

Figure 11-2: Vibration Data

Byte	Bit							
	7	6	5	4	3	2	1	0
0	Large motor set value Lower 7 bits							Small motor set value
1								Large motor set value Lower 1 bit

Return value

>=0:	Parameter setting succeeded
<0	Parameter setting failed