Sega Dreamcast

Dreamcast Shinobi Library Specification & Dreamcast Shinobi Sound Library

Sega Dreamcast

Table of Contents

1. System Functions

sbInitSystem

Initializes the Shinobi library.

Format

```
#include <shinobi.h>
extern void sbInitSystem( Int mode, Int frame, Int Count );
```

Parameters

`mode`	Screen mode (resolution) Specifies the screen resolution.
`frame`	Frame buffer mode Sets the frame buffer color mode.
`count`	Frame count Specifies the number of frames with a value in units of 1/60th of a second.

Return Value

None

Description

This function initializes the hardware and readies the library for use.

In addition, this function also calls njInitSystem, and sets the mode for the specified screen resolution.

This function sets the 2D clipping area to the same size as the screen, and sets Z clipping to a range form -1.0 to -60000.0.

This function sets the 3D screen projection surface distance to 500, and sets the aspect for both X and Y to 1.0.

This function also sets the color mode to `NJD_COLOR_MODE_NORMAL`.

This function sets the frame count with a value given in units of 1/60th of a second.

For example, a setting of "2" results in a frame change every 1/30th of a second.

Frame changes are performed by using the `njWaitVSync` function.

The screen modes that can be set are listed below.

Variable name	Screen mode
NJD_RESOLUTION_VGA	VGA 60Hz
NJD_RESOLUTION_320x240_NTSCNI	NTSC, non-interlaced, 60Hz
NJD_RESOLUTION_320x240_NTSCI	NTSC, interlaced, 30Hz
NJD_RESOLUTION_640x240_NTSCNI	NTSC, non-interlaced, 60Hz
NJD_RESOLUTION_640x240_NTSCI	NTSC, interlaced, 30Hz
NJD_RESOLUTION_320x480_NTSCNI	NTSC, non-interlaced, 60Hz
NJD_RESOLUTION_320x480_NTSCI	NTSC, interlaced, 30Hz
NJD_RESOLUTION_640x480_NTSCNI_FF	NTSC, flicker-free, 60Hz
NJD_RESOLUTION_640x480_NTSCNI	NTSC, non-interlaced, 60Hz
NJD_RESOLUTION_640x480_NTSCI	NTSC, interlaced, 30Hz
NJD_RESOLUTION_320x240_PALNI	PAL, non-interlaced, 50Hz
NJD_RESOLUTION_320x240_PALI	PAL, interlaced, 25Hz
NJD_RESOLUTION_640x240_PALNI	PAL, non-interlaced, 50Hz
NJD_RESOLUTION_640x240_PALI	PAL, interlaced, 25Hz
NJD_RESOLUTION_320x480_PALNI	PAL, non-interlaced, 50Hz
NJD_RESOLUTION_320x480_PALI	PAL, interlaced, 25Hz
NJD_RESOLUTION_640x480_PALNI_FF	PAL, flicker-free, 50Hz
NJD_RESOLUTION_640x480_PALNI	PAL, non-interlaced, 50Hz
NJD_RESOLUTION_640x480_PALI	PAL, interlaced, 25Hz
NJD_RESOLUTION_VGA_ANTI	VGA, anti-aliasing, 60Hz
NJD_RESOLUTION_320x240_NTSCNI_ANTI	NTSC, anti-aliasing, non-interlaced, 60Hz
NJD_RESOLUTION_320x240_NTSCI_ANTI	NTSC, anti-aliasing, interlaced, 30Hz
NJD_RESOLUTION_640x240_NTSCNI_ANTI	NTSC, anti-aliasing, non-interlaced, 60Hz
NJD_RESOLUTION_640x240_NTSCI_ANTI	NTSC, anti-aliasing, interlaced, 30Hz
NJD_RESOLUTION_320x480_NTSCNI_ANTI	NTSC, anti-aliasing, non-interlaced, 60Hz
NJD_RESOLUTION_320x480_NTSCI_ANTI	NTSC, anti-aliasing, interlaced, 30Hz

NJD_RESOLUTION_640x480_NTSCNI_FF_ANTI	NTSC, anti-aliasing, flicker-free, 60Hz
NJD_RESOLUTION_640x480_NTSCNI_ANTI	NTSC, anti-aliasing, non-interlaced, 60Hz
NJD_RESOLUTION_640x480_NTSCI_ANTI	NTSC, anti-aliasing, interlaced, 30Hz
NJD_RESOLUTION_320x240_PALNI_ANTI	PAL, anti-aliasing, non-interlaced, 50Hz
NJD_RESOLUTION_320x240_PALI_ANTI	PAL, anti-aliasing, interlaced, 25Hz
NJD_RESOLUTION_640x240_PALNI_ANTI	PAL, anti-aliasing, non-interlaced, 50Hz
NJD_RESOLUTION_640x240_PALI_ANTI	PAL, anti-aliasing, interlaced, 25Hz
NJD_RESOLUTION_320x480_PALNI_ANTI	PAL, anti-aliasing, non-interlaced, 50Hz
NJD_RESOLUTION_320x480_PALI_ANTI	PAL, anti-aliasing, interlaced, 25Hz
NJD_RESOLUTION_640x480_PALNI_FF_ANTI	PAL, anti-aliasing, flicker-free, 50Hz
NJD_RESOLUTION_640x480_PALNI_ANTI	PAL, anti-aliasing, non-interlaced, 50Hz
NJD_RESOLUTION_640x480_PALI_ANTI	PAL, anti-aliasing, interlaced, 25Hz

The frame buffer modes are listed below.

```
NJD_FRAMEBUFFER_MODE_RGB565
NJD_FRAMEBUFFER_MODE_RGB555
NJD_FRAMEBUFFER_MODE_ARGB1555
NJD_FRAMEBUFFER_MODE_RGB888
NJD_FRAMEBUFFER_MODE_ARGB8888
```

Example

```
sbInitSystem( NJD_RESOLUTION_VGA, NJD_FRAMEBUFFER_MODE_RGB565, 1 );
```

Initializes the Shinobi library, sets the screen resolution to VGA (640 x 480), one frame, 1/60th second.

Note

This function must be called at the beginning of a program.

Some screen modes may not be permitted with some hardware configurations.

Processing that was previously performed using njInitSystem should be replaced with this function.

Because the parameters are completely identical to njInitSystem, all that is necessary is to replace the function literal only.

sbExitSystem

Exits the Shinobi system.

Format

```
#include <shinobi.h>
extern void sbExitSystem( void )
```

Parameters

None

Return Value

None

Description

This function exits the system.
Execute this function in place of `njExitSystem`.

syHwFinish

Performs the hardware exit processing.

Format

```
void syHwFinish( void )
```

Parameters

None

Return Value

None

Description

Releases SH CPU resources and performs the hardware exit processing.

syHwInit

Initializes the hardware.

Format

```
void syHwInit( void )
```

Parameters

None

Return Value

None

Description

Mainly initializes the hardware.
This function must be executed before initializing the graphic library.
The following items are initialized with this function.

- CPU initialization
- G1-Bus initialization
- Interrupt controller initialization
- Cache initialization
- Timer initialization

syHwInit2 Initializes the interrupt controller.

Format

```
void syHwInit2( void )
```

Parameters

None

Return Value

None

Description

- Initializes the interrupt controller.
- Initializes the required library after the graphic library is initialized.

2. File System Functions

gdFsCalcSctSize

Calculates the number of sectors on the basis of the number of bytes.

Format

```
gdFsCalcSctSize( bytes )
```

Parameters

`bytes`	File byte size

Return Value

Sector size

Description

Calculates the number of sectors on the basis of the number of bytes.

Examples

```
GDFS gf;
Sint32 flen;
Sint32 fslen;

gf = gdFsOpen("TEST.BIN", NULL);
gdFsGetFileSize(gdfs, &flen);
fslen = gdFsCalcSctSize(flen);
```

Note

This function does not practically access the disc drive.

It is recommended to check header file which defines this function.

gdFsChangeDir

Changes the current directory.

Format

```
Sint32 gdFsChangeDir( dirname )
const char *dirname
```

Parameters

dirname	Directory name

Return Value

GDD_ERR_OK	Normal termination
GDD_ERR_NOTFOUND	File cannot be found
GDD_ERR_NOTDIR	Directory cannot be used
GDD_ERR_DIROVER	Exceeded storable entries
GDD_ERR_BUSY	In process
GDD_ERR_TOUT	Timed out
GDD_ERR_NOERR	No error
GDD_ERR_RECOVER	Error recovered
GDD_ERR_NOTREADY	Drive not ready
GDD_ERR_MEDIA	Media error
GDD_ERR_HWARE	Hardware error
GDD_ERR_UNITATTENT	Detection of media exchange
GDD_ERR_TRAYOPEND	Disc door is open
GDD_ERR_CHECKBUSY	Media check in process

Description

Changes the current directory.

Example

```
* Change the current directory to the DATA directory */
gdFsChangeDir("DATA");
```

Note

No distinction is made between upper- and lower-case characters.

This function practically accesses the disc drive.

gdFsCheckEof

Detects the end of a file.

Format

```
Sint32 gdFsCheckEof( gdfs, iseof )
GDFS gdfs
Bool *iseof
```

Parameters

`gdfs`	File handle
`iseof`	Pointer that returns results

Return Value

`GDD_ERR_OK`	Normal termination
`GDD_ERR_ILLHNDL`	Incorrect handle used

Description

Checks whether the end of the file (EOF) has been reached.

Results are returned in the argument `iseof`.

Value	Meaning
`0`	Not the end of file
Other than `0`	End of file

Example

```
GDFS gf;
Bool flag;
Sint32 ret;

          gf = gdFsOpen("TEST.BIN", NULL);

/* After read processing */

ret = gdFsCheckEof(gf, &flag);
if (ret == GDD_ERR_OK && flag == TRUE) {
          gdFsClose(gf);
          return;
}
```

Note

This function does not practically access the disc drive.

Reference

`GDD_ERR`	File error

gdFsClose

Closes a file.

Format

```
void gdFsClose( gdfs )
GDFS gdfs
```

Parameters

`gdfs`	File handle

Output

None

Return Value

None

Description

Closes a file.

Example

```
GDFS gf;

gf = gdFsOpen("TEST.BIN", NULL);
gdFsClose(gf);
```

gdFsCreateDirhn

Generates a directory handle.

Format

```
GDFS_DIRREC gdFsCreateDirhn( dirbuf, max_dirent )
void *dirbuf
Sint32 max_dirent
```

Parameters

`dirbuf`	Directory buffer
`max_dirent`	Number of directory entries (3 and up)

Return Value

Directory record handle

Description

Generates the directory handle.

Number of directory entry includes itself and parent directory. In other words, number of directory entry will be no less than 3.

Example

```
Uint32 dirbuf[gdFsGetDirrecSize(64)];
GDFS_DIRREC g_dir;
g_dir = gdFsCreateDirhn(dirbuf, 64);
```

gdFsDaGetInfo

Gets the DA playback information.

Format

```
Sint32 gdFsDaGetInfo( dainfo )
GDFS_DAINFO *dainfo
```

Parameters

dainfo	Pointer for where DA playback information is put

Return Value

GDD_ERR_OK	Normal termination
GDD_ERR_BUSY	In process

Description

Gets the DA playback information.

Note

Get the system handle and monitor the status. The information is stored only when the status is GDD_STAT_COMPLETE.

This function practically accesses the disc drive.

Reference

GDD_ERR	File error

gdFsDaPause

Pauses DA playback.

Format

```
Sint32 gdFsDaPause( void )
```

Parameters

None

Return Value

GDD_ERR_OK	Normal termination
GDD_ERR_BUSY	In process

Description

Pauses DA playback.

Example

```
GDFS sys = gdFsGetSysHn();
gdFsDaPause();

/* After a while */

stat = gdFsGetStat(sys);
if (stat == GDD_STAT_COMPLETE) {
          /* Pause is finished */
} else if (stat == GDD_STAT_ERR) {
          /* Error */
}
```

Note

To check whether pause is finished, use the system handle.

This function practically accesses the disc drive.

Reference

GDD_ERR	File error
gdFsDaRelease()	Releases DA playback from the paused state

gdFsDaPlay

Starts DA playback (sector specified).

Format

```
Sint32 gdFsDaPlay( st_track, end_track, reptime )
Sint32 st_track
Sint32 end_track
Sint32 reptime
```

Parameters

st_track	Starting track number
end_track	Ending track number
reptime	Repetitions (0 to 14: Number of repetitions; 15: Infinite loop)

Return Value

GDD_ERR_OK	Normal termination
GDD_ERR_BUSY	In process

Description

Specifies the track and starts DA playback.

Example

```
GDFS sys = gdFsGetSysHn();
gdFsDaPlay(4, 4, 0);

          /* After a while */

stat = gdFsGetStat(sys);
if (stat == GDD_STAT_COMPLETE) {
          /* Playback is finished */
} else if (stat == GDD_STAT_ERR) {
          /* Error */
}
```

Note

Single-density track cannot be played back.

To check whether playback is finished, use the system handle.

This function practically accesses the disc drive.

Reference

GDD_ERR	File error

gdFsDaPlaySct

Starts DA playback (sector specified).

Format

```
Sint32 gdFsDaPlaySct( st_sct, end_sct, reptime )
Sint32 st_sct
Sint32 end_sct
Sint32 reptime
```

Parameters

st_sct	Starting sector
end_sct	Ending sector
reptime	Repetitions (0 to 14: Number of repetitions; 15: Infinite loop)

Return Value

GDD_ERR_OK	Normal termination
GDD_ERR_BUSY	In process

Description

Starts DA playback at a specified sector.

Example

```
GDFS sys = gdFsGetSysHn();
gdFsDaPlaySct(0xC000, 0xD000, 2);

          /* After a while */

stat = gdFsGetStat(sys);
if (stat == GDD_STAT_COMPLETE) {
          /* Playback is finished */
} else if (stat == GDD_STAT_ERR) {
          /* Error */
}
```

gdFsDaRelease

Releases DA playback from the paused state.

Format

```
Sint32 gdFsDaRelease( reptime )
Sint32 reptime
```

Parameters

reptime	Number of repetitions

Return Value

GDD_ERR_OK	Normal termination
GDD_ERR_BUSY	In process

Description

Releases DA playback from the paused state.

Example

```
GDFS sys = gdFsGetSysHn();
gdFsDaRelease(0);

/* After a while */

stat = gdFsGetStat(sys);
if (stat == GDD_STAT_COMPLETE) {
          /* Release pause is finished */
} else if (stat == GDD_STAT_ERR) {
          /* Error */
}
```

Note

Parameter reptime is currently invalid.

To chech whether release pause is finished, use the system handle.

This function practically accesses the disc drive.

Reference

GDD_ERR	File error
gdFsDaPause()	Pauses DA playback

gdFsDaStop

Stops DA playback.

Format

```
Sint32 gdFsDaStop( void )
```

Parameters

None

Return Value

GDD_ERR_OK	Normal termination
GDD_ERR_BUSY	In process

Description

Stops DA playback.

Example

```
gdFsDaStop();
GDFS sys = gdFsGetSysHn();
gdFsDaStop();

/* After a while */

stat = gdFsGetStat(sys);
if (stat == GDD_STAT_COMPLETE) {
          /* Stop is finished */
} else if (stat == GDD_STAT_ERR) {
          /* Error */
}
```

Note

To check whether stop is finished, use the system handle.

This function practically accesses the disc drive.

gdFsEntryErrFunc

Registers error generation callback function for a specified handle.

Format

```
void gdFsEntryErrFunc( gdfs, func, obj )
GDFS gdfs
GDFS_ERRFUNC func
void *obj
```

Parameters

gdfs	File handle
func	Pointer to a allback function
obj	First parameter that is passed to the callback function

Return Value

None

Description

Registers an error generation callback function for a specified handle.

Example

```
void err_callback(void *obj, Sint32 errcode)
{
          /* Processing */
}

gdFsEntryErrFunc(gf, err_callback, (void *)0x1234);
```

Note

This function does not practically access the disc drive.

gdFsEntryErrFuncAll

Registers an error generation callback function for all handles.

Format

```
void gdFsEntryErrFuncAll( erfunc, obj )
GDFS_ERRFUNC erfunc
void *obj
```

Parameters

`erfunc`	Poiner to callback function
`obj`	Pointer to first parameter which is passed to the callback function

Return Value

None

Description

Registers an error generation callback function for all handles.

Example

```
void err_callback(void *obj, Sint32 errcode)
{
          /* Processing */
}

gdFsEntryErrFuncAll(err_callback, (void *) 0x1234);
```

Note

This function does not practically access the disc drive.

gdFsEntryRdEndFunc

Registers the callback function for when a read is completed.

Format

```
void gdFsEntryRdEndFunc( gdfs, func, obj )
GDFS gdfs
GDFS_FUNC func
void *obj
```

Parameters

`gdfs`	File handle
`func`	Pointer to a allback function
`obj`	Pointer to first parameter which is passed to the callback function

Return Value

None

Description

Registers the callback function for when a read is completed.

Example

```
void rdend_callback(void *obj)
{
          /* Processing */
}

gdFsEntryRdEndFunc(gf, rdend_callback, (void *) 0x1234);
```

Note

This function does not practically access the disc drive.

gdFsEntryTrEndFunc

Registers the callback function for when a transfer ends.

Format

```
void gdFsEntryTrEndFunc( gdfs, func, obj )
GDFS gdfs
GDFS_FUNC func
void *obj
```

Parameters

`gdfs`	File handle
`func`	Pointer to callback function
`obj`	Pointer to first parameter which is passed to the callback function

Return Value

None

Description

Registers the callback function for when a transfer is completed.

Example

```
void trend_callback(void *obj)
{
          /* Processing */
}

gdFsEntryTrEndFunc(gf, trend_callback, (void *) 0x1234);
```

Note

This function does not practically access the disc drive.

gdFsFinish

Ends use of the GD file system .

Format

```
void gdFsFinish( void )
```

Parameters

None

Return Value

None

Description

Ends use of the GD file system.

Example

```
Uint32 gdfswork[gdFsGetWorkSize(8)/4];
Uint32 gdfscurdir[gdFsGetDirrecSize(64)/4];
gdFsInit(8, gdfswork, 64, gdfscurdir);
gdFsFinish();
```

Note

This function does not practically access the disc drive.

gdFsGetDirInfo

Gets the file information.

Format

```
Sint32 gdFsGetDirInfo( name, dirinfo )
const char *name
GDFS_DIRINFO *dirinfo
```

Parameters

name	File name/directory name
dirinfo	Pointer to structure where file information is to be stored

Return Value

GDD_ERR_OK	Normal termination
GDD_ERR_NOTFOUND	File cannot be found

Description

Gets file information.

Example

```
GDFS_DIRINFO dinfo;
gdFsGetDirInfo("ABC.DAT", &dinfo);
```

Note

This function does not practically access the disc drive.

Reference

GDD_ERR	File error

gdFsGetDirrecSize

Gets the byte count for the directory buffer.

Format

```
gdFsGetDirrecSize(max_dirent)
```

Parameters

`max_dirent`	Maximum number of entries in the directory

Return Value

Number of bytes required for the directory buffer

Description

Determines the number of bytes required for the directory buffer.

Examples

```
Uint32 gdfscurdirgdFsGetDirrecSize(64)/4;
```

Note

It is recommended to check header file which defines this function.
This function does not practically access the disc drive.

gdFsGetDrvStat

Gets the drive status.

Format

```
Sint32 gdFsGetDrvStat(void)
```

Parameters

None

Return Value

Drive status

Description

This function gets the drive status.

The following values are returned as drive status

Name	Function
GDD_DRVSTAT_BUSY	Busy
GDD_DRVSTAT_PAUSE	Paused
GDD_DRVSTAT_STANDBY	Standby
GDD_DRVSTAT_SEEK	Seek in progress
GDD_DRVSTAT_PLAY	Playback in progress
GDD_DRVSTAT_SCAN	Scan playback in progress
GDD_DRVSTAT_OPEN	Tray is open
GDD_DRVSTAT_NODISC	No disc
GDD_DRVSTAT_RETRY	Retry
GDD_DRVSTAT_ERROR	Error

Example

```
Sint32 dstat;
```

dstat = **gdFsGetDrvStat**();

Note

This function does not practically access the disc drive.

gdFsGetErrStat

Gets the handle error status.

Format

```
Sint32 gdFsGetErrStat( gdfs )
GDFS gdfs
```

Parameters

gdfs	File handle

Return Value

GDD_ERR_NOERR	No error
GDD_ERR_RECOVER	Error recovered
GDD_ERR_NOTREADY	Drive not ready
GDD_ERR_MEDIA	Media error
GDD_ERR_HWARE	Hardware error
GDD_ERR_ILLREQ	Illegal request issued
GDD_ERR_UNITATTENT	Detection of media exchange
GDD_ERR_PROTECT	Protected
GDD_ERR_ABORT	Aborted
GDD_ERR_NOREADABLE	Unreadable
GDD_ERR_TRAYOPEND	Tray is open
GDD_ERR_CHECKBUSY	Media check in process

Description

Gets the handle error status.

Example

```
GDFS gf;
Uint32 buf[32*2048/4];
Sint32 stat;
Sint32 err;

gf = gdFsOpen("TEST.BIN", NULL);
gdFsReqRd32(gf, 32, buf);

while ((stat = gdFsGetStat(gf)) == GDD_STAT_READ);
if (stat == GDD_STAT_ERR) {
          err = gdFsGetErrStat(gf);
          /* Error processing */
}
gdFsClose(gf);
```

Note

This function does not practically access the disc drive.

Reference

GDD_ERR	File error

gdFsGetFileFad

Gets a file's FAD.

Format

```
Bool gdFsGetFileFad( gdfs, fad )
GDFS gdfs
Sint32 *fad
```

Parameters

gdfs	Output
fad	Frame address (physical sector)

Return Value

TRUE	Succeeded
FALSE	Failed (wrong handle, etc.)

Description

Gets a file's FAD..

Example

```
Sint32 fad;
Sint32 gf;

gf = gdFsOpen("TEST.BIN", NULL);
gdFsGetFileFad(gf, &fad);
```

Note

This function does not practically access the disc drive.

gdFsGetFileSctSize

Gets file's sector size.

Format

```
Bool gdFsGetFileSctSize( gdfs, fsctsize )
GDFS gdfs
Sint32 *fsctsize
```

Parameters

`gdfs`	File handle
`fsctsize`	File size (sector)

Return Value

`TRUE`	Succeeded
`FALSE`	Failed (wrong handle, etc.

Description

Gets a file's sector size.

Example

```
GDFS gf;
Sint32 fslen;

gf = gdFsOpen("TEST.BIN", NULL);
gdFsGetFileSctSize(gdfs, &fslen);
```

Note

This function does not practically access the disc drive

gdFsGetFileSize

Gets file size information.

Format

```
Bool gdFsGetFileSize( gdfs, fsize )
GDFS gdfs
Sint32 *fsize
```

Parameters

gdfs	File handle
fsize	Pointer that returns results

Return Value

TRUE	Succeeded
FALSE	Failed (wrong handle, etc.)

Description

This function gets file size information.

Example

```
GDFS gf;
Sint32 flen;

gf = gdFsOpen("TEST.BIN", NULL);
gdFsGetFileSize(gdfs, &flen);
```

Note

This function does not practically access the disc drive

gdFsGetNumRd

Gets the number of bytes that were read .

Format

```
Sint32 gdFsGetNumRd( gdfs )
GDFS gdfs
```

Parameters

gdfs	File handle

Return Value

0 or greater:	Number of bytes that were read

Description

This function gets the number of bytes that were read.

Example

```
GDFS gf;
Uint32 buf[32*2048/4];

gf = gdFsOpen("TEST.BIN", NULL);
gdFsReqRd32(gf, 32, buf);

while (gdFsGetStat(gf) != GDD_STAT_COMPLETE) {
          readnum = gdFsGetNumRd(gdfs);
}

gdFsClose(gf);
```

Note

This function does not practically access the disc drive.

gdFsGetStat

Gets the handle status.

Format

```
Sint32 gdFsGetStat( gdfs )
GDFS gdfs
```

Parameters

gdfs	File handle

Return Value

GDD_STAT	Handle status

Description

This function gets the handle status.

The following values are returned as handle status.

Definition	Meaning
GDD_STAT_IDLE	Idle
GDD_STAT_COMPLETE	Operation complete
GDD_STAT_READ	Read in progress
GDD_STAT_ERR	Error occurred
GDD_STAT_FATAL	Fatal error occurred

Example

```
GDFS gf;
Uint32 buf[32*2048/4];

gf = gdFsOpen("TEST.BIN", NULL);
gdFsReqRd32(gf, 32, buf);
while (gdFsGetStat(gf) == GDD_STAT_READ);
gdFsClose(gf);
```

Note

This function does not practically access the disc drive.

gdFsGetSysHn

Gets the system handle.

Format

```
GDFS gdFsGetSysHn(void)
```

Parameters

None

Return Value

System handle

Description

Gets the system handle.

This function is used to check whether the system handle is currently being processed.

Example

```
GDFS sys;
sys = gdFsGetSysHn();
stat = gdFsGetStat(sys);
```

Note

This function does not practically access the disc drive.

gdFsGetToc

Gets the TOC.

Format

```
Sint32 gdFsGetToc( type, buf )
Sint32 type
void *buf
```

Parameters

type	TOC type
buf	Buffer pointer that reads TOC (408 bytes are required).

Return Value

GDD_ERR_OK	Normal termination
GDD_ERR_TOUT	Timed out
GDD_ERR_ILLHNDL	Incorrect handle used
GDD_ERR_BUSY	In process
GDD_ERR_RECOVER	Error recovered
GDD_ERR_NOTREADY	Drive not ready
GDD_ERR_MEDIA	Media error
GDD_ERR_HWARE	Hardware error
GDD_ERR_ILLREQ	Illegal request issued
GDD_ERR_UNITATTENT	Detection of media exchange
GDD_ERR_PROTECT	Protected
GDD_ERR_ABORT	Aborted
GDD_ERR_NOREADABLE	Unreadable
GDD_ERR_TRAYOPEND	Tray is open
GDD_ERR_CHECKBUSY	Media check in process

Description

Gets a TOC.

TOC type is specified in parameter type.

Only 1 (high density area) can be specified on current version. 408 bytes are necessary in the buffer.

Example

```
gdFsGetToc(1, &buf);
```

Note

```
This function practically accesses the disc drive.
It is not possible to get TOC information from single-density area.
```

Reference

GDD_ERR	File error

gdFsGetTransStat

Gets the transfer status.

Format

```
Sint32 gdFsGetTransStat( gdfs )
GDFS gdfs
```

Parameters

gdfs	File handle

Return Value

GDD_ERR_ILLHNDL	Incorrect handle used

Description

Gets the transfer status.

The following values are returned as transfer status.

Definition	Meaning
GDD_FS_TRANS_READY	Transfer ready
GDD_FS_TRANS_BUSY	Transfer in progress
GDD_FS_TRANS_COMPLETE	Transfer complete
GDD_FS_TRANS_ERROR	Transfer error

Note

This function does not practically access the disc drive.

Reference

GDD_ERR	File error

gdFsGetWorkHn

Gets the handle that is being processed.

Format

```
GDFS gdFsGetWorkHn( void )
```

Parameters

None

Return Value

Handle that is currently being processed

Description

Gets the handle that is being processed.

If no handle is being processed, `NULL` is returned.

Example

```
GDFS gf;
gf = gdFsGetWorkHn();
```

Note

This function does not practically access the disc drive.

gdFsGetWorkSize

Determines the number of bytes for the library work area.

Format

```
gdFsGetWorkSize( max_open )
```

Parameters

max_open	Number of files that can be open at one time

Return Value

Number of bytes required for the library work area

Description

This macro determines the number of bytes required for the library work area.

Examples

```
Uint32 gdfsworkgdFsGetWorkSize(8)/4;
```

Note

It is recommended to check header file which defines this function.
This function does not practically access the disc drive.

gdFsInit

Initializes the GD file system.

Format

```
Sint32 gdFsInit( max_open, gdfs_work, max_dirent,
dirbuf )
Sint32 max_open
void *gdfs_work
Sint32 max_dirent
void *dirbuf
```

Parameters

`max_open`	Number of files that can be open simultaneously
`gdfs_work`	Work area pointer (provided from user area)
`max_dirent`	Number of entries in current directory
`dirbuf`	Current directory buffer (provided from user area)

Return Value

`GDD_ERR_OK`	Initialization completed
`GDD_ERR_32ALIGN`	"gdfs_work" does not coincide with a 32-byte boundary
`GDD_ERR_RESET`	Drive reset failed
`GDD_ERR_TRAYOPEND`	GD tray is open
`GDD_ERR_DISC`	Disc is unusable
`GDD_ERR_MOUNT`	Mount failed
`GDD_ERR_DIROVER`	Too many entries in the root directory

Description

Initializes the GD file system.

Parameter `gdfs_work` must be aligned with a 32-byte boundary. (dirbuf with a 4-byte boundary). Once initialization has been completed, re-initialization is not possible even if `gdFsInit()` function is called, unless `gdFsFinish()` function has been called beforehand.

The main processing that is performed by `gdFsInit()` is as follows:

- Device driver initialization
- Work area initialization
- Device initialization
- Mount processing

Example

```
Uint32 gdfsworkgdFsGetWorkSize(8)/4;
Uint32 gdfscurdirgdFsGetDirrecSize(64)/4;
gdFsInit(8, gdfswork, 64, gdfscurdir);
```

Note

This function practically accesses the disc drive.

Reference

`gdFsFinish()`	Ends use of the GD file system
`GDD_ERR`	File error

gdFsLoadDir

Loads a directory record.

Format

```
Sint32 gdFsLoadDir( dirname, gf_dirrec )
const char *dirname
GDFS_DIRREC gf_dirrec
```

Parameters

dirname	Directory name
gf_dirrec	Directory record handle (NULL: To the current directory)

Return Value

GDD_ERR_OK	Normal termination
GDD_ERR_NOTFOUND	File cannot be found
GDD_ERR_NOTDIR	Directory cannot be used
GDD_ERR_DIROVER	Exceeded storable entries
GDD_ERR_BUSY	In process
GDD_ERR_TOUT	Timed out
GDD_ERR_NOERR	No error
GDD_ERR_RECOVER	Error recovered
GDD_ERR_NOTREADY	Drive not ready
GDD_ERR_MEDIA	Media error
GDD_ERR_HWARE	Hardware error
GDD_ERR_UNITATTENT	Detection of media exchange
GDD_ERR_TRAYOPEND	Disc door is open
GDD_ERR_CHECKBUSY	Media check in process

Description

Loads a directory record.

If parameter gf_dirrec is specified as NULL, the current directory is changed to a loaded directory record.

Example

```
/* Sample 1 */
/* Load MOVIE directory into g_dir */
Uint32 dirbufgdFsGetDirrecSize(64);
GDFS_DIRREC g_dir;
g_dir = gdFsCreateDirhn(dirbuf, 64);
gdFsLoadDir("MOVIE", g_dir);
/* Sample 2 */
/* Change the current directory to the DATA directory */
gdFsLoadDir("DATA", NULL);
```

Note

No distinction is made between upper- and lower-case characters.

This function practically accesses the disc drive.

Reference

GDD_ERR	File error

gdFsMovePickup

Moves the pickupto the next position.

Format

```
Sint32 gdFsMovePickup( gdfs )
GDFS gdfs
```

Parameters

gdfs	File handle

Return Value

GDD_ERR_OK	Normal termination
GDD_ERR_TOUT	Timed out
GDD_ERR_ILLHNDL	Incorrect handle used
GDD_ERR_BUSY	In process
GDD_ERR_RECOVER	Error recovered
GDD_ERR_NOTREADY	Drive not ready
GDD_ERR_MEDIA	Media error
GDD_ERR_HWARE	Hardware error
GDD_ERR_ILLREQ	Illegal request issued
GDD_ERR_UNITATTENT	Detection of media exchange
GDD_ERR_PROTECT	Protected
GDD_ERR_ABORT	Aborted
GDD_ERR_NOREADABLE	Unreadable
GDD_ERR_TRAYOPEND	Tray is open
GDD_ERR_CHECKBUSY	Media check in process

Description

This function moves the pickup to the next reading position.

Example

```
gdFsMovePickup(gf);
```

Note

This function practically accesses the disc drive.

Reference

GDD_ERR	File error

gdFsOpen

Opens a file.

Format

```
GDFS gdFsOpen( fname, gf_dirrec )
const char *fname
GDFS_DIRREC gf_dirrec
```

Parameters

`fname`	File name
`gf_dirrec`	Directory record handle where the file name will be searched for.

Return Value

`NULL`	Failed to open
`Any other value:`	File handle

Description

This function opens a file.
If **`gf_dirrec`** is specified as `NULL`, file will be searched for in the current directory.

Example

```
/* Sample 1 */
GDFS gf;
gf = gdFsOpen("A.BIN", NULL);

/* Sample 2 */
GDFS gf;
Uint32 dirbuf[gdFsGetDirSize(64)];
GDFS_DIRREC g_dir;
g_dir = gdFsCreateDirhn(dirbuf, 64);
gdFsLoadDir("MOVIE", g_dir);
gf = gdFsOpen("SMP.MOV", g_dir);
```

Note

This function does not practically access the disc drive.

gdFsOpenRange

Opens a file in specified sectors.

Format

```
GDFS gdFsOpenRange( stsct, nsct )
Sint32 stsct
Sint32 endsct
```

Parameters

`stsct`	Starting sector
`nsct`	Sector size

Return Value

`NULL`	Failed to open
`Any other value:`	File handle

Description

Opens a file in specified sectors.

Example

```
GDFS gf;

gf = gdFsOpenRange(0xB555, 0x10);
gdFsClose(gf);
```

Note

The single-density area cannot be accessed.

This function does not practically access the disc drive.

gdFsRead

Reads a file.

Format

```
Sint32 gdFsRead( gdfs, nsct, buf )
GDFS gdfs
Sint32 nsct
void *buf
```

Parameters

gdfs	File handle
nsct	Number of sectors to be read
buf	Pointer to storage buffer

Return Value

GDD_ERR_OK	Normal termination
GDD_ERR_TOUT	Timed out
GDD_ERR_ILLHNDL	Incorrect handle used
GDD_ERR_RECOVER	Error recovered
GDD_ERR_NOTREADY	Drive not ready
GDD_ERR_MEDIA	Media error
GDD_ERR_HWARE	Hardware error
GDD_ERR_UNITATTENT	Detection of media exchange
GDD_ERR_PROTECT	Protected
GDD_ERR_ABORT	Aborted
GDD_ERR_NOREADABLE	Unreadable
GDD_ERR_TRAYOPEND	Disc door is open
GDD_ERR_CHECKBUSY	Media check in process

Description

Reads a file.

Control does not return from the function until the read is completed. (Return on completion)

Example

```
GDFS gf;
Sint32 buf32*2048/4;

gf = gdFsOpen("TEST.BIN", NULL);
gdFsRead(gf, 32, buf);
gdFsClose(gf);
```

Note

This function practically accesses the disc drive.

Reference

GDD_ERR	File error

gdFsReinit

Remounts the GD file system.

Format

```
Sint32 gdFsReinit( void )
```

Parameters

None

Return Value

GDD_ERR_OK	Initialization completed
GDD_ERR_32ALIGN	gdfs_work does not coincide with a 32-byte boundary
GDD_ERR_RESET	Drive reset failed
GDD_ERR_TRAYOPEND	GD tray is open
GDD_ERR_DISC	Disc is unusable
GDD_ERR_MOUNT	Mount failed
GDD_ERR_DIROVER	Too many entries in the root directory

Description

Remounts the GD file system, from device initialization to mount processing. Use this function when changing media, etc.

Note

This function practically accesses the disc drive.

Reference

GDD_ERR	File error

gdFsReqDrvStat

Changes the drive status .

Format

```
Sint32 gdFsReqDrvStat( void )
```

Parameters

None

Return Value

GDD_ERR_OK	Normal termination
GDD_ERR_BUSY	In process

Description

Changes the drive status.

Example

```
Sint32 dstat;
GDFS sys;

gdFsReqDrvStat();
sys = gdFsGetSysHn();

/* After a while */

if (gdFsGetStat(sys) != GDD_STAT_BUSY) {
          /* Command execution is finished */
          dstat = gdFsGetDrvStat();
} else {
          /* Command execution is not finished */
}
```

Note

This command issues a dummy command to the drive and let it update the drive status. After the command is issued, control is returned immediately. However, it takes at least several ms for the command execution to finish.

While the command is running, the drive status cannot be read. To check whether the command execution is finished, use the system handle.

Reference

GDD_ERR	File error

gdFsReqGdRd

Issues a read-ahead request to the GD buffer.

Format

```
Sint32 gdFsReqGdRd( gdfs, nsct )
GDFS gdfs
Sint32 nsct
```

Parameters

gdfs	File handle
nsct	Number of sectors read

Return Value

`Positive value(including 0)`	Number of sectors actually requested
`Negative value`	Error code

Description

Issues a read-ahead request to the GD buffer.

Example

```
gdFsReqGdRd(gf, 128);
```

Note

This function practically accesses the disc drive.

gdFsReqRd32

Requests to read the error code file.

Format

```
Sint32 gdFsReqRd32( gdfs, nsct, buf )
GDFS gdfs
Sint32 nsct
void *buf
```

Parameters

gdfs	File handle
nsct	Number of sectors to be read
buf	Pointer to storage buffer

Return Value

Positive value	Number of sectors actually requested
GDD_ERR_ILLHNDL	Incorrect handle used
GDD_ERR_32ALIGN	Not aligned with 32-byte boundary

Description

Requests to read the error code file.

Control returns from this function once the read request is made. A separate check must be made to determine if the read has actually been completed or not. (Immediate return).

Example

```
GDFS gf;
Uint32 buf32*2048/4;

gf = gdFsOpen("TEST.BIN", NULL);
gdFsReqRd32(gf, 32, buf);
while (gdFsGetStat(gf) == GDD_STAT_READ);
gdFsClose(gf);
```

Note

A 32-byte boundary is necessary for buf. Also, only one request can be accepted at any one time.

This function practically accesses the disc drive.

Reference

GDD_ERR	File error

gdFsSeek

Searches for a file.

Format

```
Sint32 gdFsSeek( gdfs, sctno, type )
GDFS gdfs
Sint32 sctno
Sint32 type
```

Parameters

gdfs	File handle
sctno	Sector number
type	Basic position

Return Value

GDD_ERR_OK	Normal termination
GDD_ERR_ILLHNDL	Incorrect handle used
GDD_ERR_OFS	Illegally specified position
GDD_ERR_SEEK	Illegal seek specification

Description

Moves the file reading pointer. Basic position type is specified as follows.

Definition	Meaning
GDD_SEEK_SET	File start
GDD_SEEK_CUR	Current file reading position
GDD_SEEK_END	File end

Example

```
GDFS gf;

gf = gdFsOpen("TEST.BIN", NULL);

/* Search in the 5th sector of the starting address */
gdFsSeek(gf, 5, SEEK_SET);
```

Note

This function does not practically access the disc drive.

Reference

GDD_ERR	File error

gdFsSetDir

Sets the current directory.

Format

```
Sint32 gdFsSetDir( gf_dirrec )
GDFS_DIRREC gf_dirrec
```

Parameters

gf_dirrec	Directory record handle

Return Value

GDD_ERR_OK	Normal termination
GDD_ERR_DIROVER	Exceeded storable entries

Description

Sets the current directory.

Example

```
/* Load MOVIE directory into g_dir */
Uint32 dirbuf[gdFsGetDirrecSize(64)];
GDFS_DIRREC g_dir;
g_dir = gdFsCreateDirhn(dirbuf, 64);
gdFsLoadDir("MOVIE", g_dir);
gdFsSetDir(g_dir);
```

Note

This function does not practically access the disc drive.

Reference

GDD_ERR	File error

gdFsStopRd

Aborts a request .

Format

```
Sint32 gdFsStopRd( gdfs )
GDFS gdfs
```

Parameters

gdfs	File handle

Return Value

GDD_ERR_OK	Normal termination
GDD_ERR_TOUT	Timed out
GDD_ERR_ILLHNDL	Incorrect handle used
GDD_ERR_BUSY	In process
GDD_ERR_RECOVER	Error recovered
GDD_ERR_NOTREADY	Drive not ready
GDD_ERR_MEDIA	Media error
GDD_ERR_HWARE	Hardware error
GDD_ERR_ILLREQ	Illegal request issued
GDD_ERR_UNITATTENT	Detection of media exchange
GDD_ERR_PROTECT	Protected
GDD_ERR_ABORT	Aborted
GDD_ERR_NOREADABLE	Unreadable
GDD_ERR_TRAYOPEND	Tray is open
GDD_ERR_CHECKBUSY	Media check in process

Description

Aborts a request.

Example

```
gdFsStopRd(gf);
```

Note

This function does not practically access the disc drive.

Reference

GDD_ERR	File error

gdFsTell

Gets the position of the file reading pointer.

Format

```
Sint32 gdFsTell( gdfs )
GDFS gdfs
```

Parameters

```
gdfs      File handle
```

Return Value

Reading position

Description

Gets the position of the file reading pointer in sector units.

Example

```
Sint32 pos;
GDFS gf;

gf = gdFsOpen("TEST.BIN", NULL);

pos = gdFsTell(gf);
```

Note

This function does not practically access the disc drive.

gdFsTrans32

Specifies transfer from the GD buffer.

Format

```
Sint32 gdFsTrans32( gdfs, nbytes, buf )
GDFS gdfs
Sint32 nbytes
void *buf
```

Parameters

gdfs	File handle
nbytes	Number of bytes to be transferred (32-byte units)
buf	Transfer address

Return Value

GDD_ERR_OK	Normal termination
GDD_ERR_ILLHNDL	Incorrect handle used
GDD_ERR_32ALIGN	Not aligned with 32-byte boundary
GDD_ERR_SIZE	Not 32-byte unit
GDD_ERR_SIZEOVER	Request size is too large
GDD_ERR_NOTREAD	No read
GDD_ERR_NOTREADY	Drive not ready
GDD_ERR_SIZEOVER	Request size is too large

Description

Specifies a transfer from the GD buffer.

Transfer byte count is specified with a 32-byte boundary.

Example

```
while ((stat = gdFsGetStat(gf)) == GDD_STAT_READ) {
          if (gdFsGetTransStat(gf) == GDD_TRANS_READY)
              gdFsTrans32(gf, 2048, buf);
}
```

Note

This function does not practically access the disc drive.

Reference

GDD_ERR	File error

3. Memory Management Functions

syCalloc

Allocates the memory.

Format

```
void *syCalloc( nobj, size )
Uint32 nobj
Uint32 size
```

Parameters

`nobj`	Number of blocks
`size`	Requested size (bytes)

Return Value

Pointer to allocated area

Description

Allocates memory block for the size (bytes) indated by parameter size and block numbers indicated by parameter `nobj`.

If `NULL` is returned, it indicates the failure of memory allocation.

This function allocates memory in alignment with 32-byte boundaries and that area is zero-cleared.

Note

This function is the same as the standard C function, `calloc()`.

Reference

`syMalloc()`	Allocates the memory
`syRealloc()`	Changes size of the allocated memory

syFree

Releases the memory.

Format

```
Void syFree( ap )
Void *ap
```

Parameters

`ap`	Pointer to area to be freed

Return Value

None

Description

Frees the memory area.

Performs the same function as standard C function `free()`.

Note

It is recommended to check header file which defines this function.

Reference

`syMalloc()`	Allocates the memory

syMalloc

Allocates the memory.

Format

```
Void *syMalloc( size )
Uint32 size
```

Parameters

`size`	Requested size (bytes)

Return Value

Pointer to allocated area

Description

Allocates the memory.

This function is the same as the standard C function malloc(size).

If the return value is `NULL`, securing memory has failed.

This function always allocates memory in alignment with 32-byte boundaries. Defining the data size in alignment with 32-byte boundaries permits efficient allocation of memory.

Note

It is recommended to check header file which defines this function.

Reference

`syMallocInit()`	Initializes the Malloc system
`syMallocStat()`	Gets the memory use status

syMallocFinish — Exits the Malloc system.

Format

```
Void syMallocFinish( Void )
```

Parameters

None

Return Value

None

Description

Exits the Malloc system. When this occurs, the heap that was allocated by `syMallocInit()` function is released.

Reference

`syMallocInit()`	Initializes the Malloc system

syMallocInit

Initializes the Malloc system.

Format

```
Void syMallocInit( heap, size )
Void *heap
Uint32 size
```

Parameters

`heap`	Starting address of the area to be managed
`size`	Size of memory to be managed

Return Value

None

Description

Specifies the area that is to be managed and initializes the library.

The memory management functions manage memory directly under the Shinobi library, which isn't intermediated by the OS. The default values required are as follows:

Heap area starting address: End of section B

Heap size: T o the end of memory

Note

This function regards the space starting from section B as the memory allocation space, and initializes the contents of memory. If there is a user section subsequent to section B, that section will be initialized. When allocating user sections, always allocate them in front of section B.

syMallocStat

Gets the memory use status.

Format

```
Void syMallocStat( free, size )
Uint32 *free
Uint32 *size
```

Parameters

`free`	Pointer that returns the remainder of usable heap area
`size`	Pointer that returns the maximum size possible to secure at one time

Return Value

None

Description

Gets the total unused area in the heap that was allocated by `syMallocInit()` function, and the maximum size that could be obtained with a single `syMalloc()` function.

Reference

`syMallocInit()`	Initializes the Malloc system
`syMalloc()`	Allocates the memory

syRealloc

Changes size of the allocated memory.

Format

```
void *syRealloc( op, nbytes )
void *op
Uint32 nbytes
```

Parameters

op	Starting address of heap to be changed
nbytes	Requested size (bytes)

Return Value

Pointer to allocated memory

Description

Changes memory size allocated by `syMalloc()`, `syCalloc()`, `syRealloc()` functions to the size (byte) indicated by nbytes.

This function allocates (if available) specifed memory size nbytes starting from address op.

In case there is not enough free memory from address op, `nbytes` size of memory is allocated at different area. op is released after the contents are copied to this new area.

This function allocates memory in alignment with 32-byte boundaries.

If `NULL` is returned, it indicates the failure of memory allocation.

Note

This function is the same as the standard C function, `realloc()`.

Reference

`syMalloc()`	Allocates the memory
`syCalloc()`	Allocates the memory

4. Cache Functions

syCacheICI

Invalidates all entries in the instruction cache.

Format

```
Void syCacheICI( Void )
```

Parameters

None

Return Value

None

Description

Invalidates all entries in the instruction cache.

Note

It is recommended to check header file which defines this function.

syCacheInit

Initializes the cache functions.

Format

```
Void syCacheInit( form )
SY_CACHE_FORM form
```

Parameters

`form`	Cache type

Return Value

None

Description

Initializes the cache functions.

The argument form can be specified as follows:

Note

Definition	Meaning
SYD_CACHE_FORM_IC_INDEX	Sets instruction cache to index mode
SYD_CACHE_FORM_OC_INDEX	Sets operand cache to index mode
SYD_CACHE_FORM_IC_ENABLE	Enables instruction cache
SYD_CACHE_FORM_OC_ENABLE	Enables operand cache
SYD_CACHE_FORM_OC_RAM	Sets operand cache to RAM mode
SYD_CACHE_FORM_P1_CBP1	Sets area to copyback mode
SYD_CACHE_FORM_P0_WTP0	Sets area to writeback mode

It is recommended to check header file which defines this function.

This function is called from within the `syHwInit()` function.

Applications are prohibited from overwriting the values that were set by this function.

Upon initialization, the settings are as follows:

- `SYD_CACHE_FORM_IC_ENABLE |`
- `SYD_CACHE_FORM_OC_ENABLE |`
- `SYD_CACHE_FORM_P1_CB`

RAM mode for the operand cache is prohibited for the Sega library.

Reference

`syHwInit()`	Initializes the hardware

syCacheMOVCAL

Executes MOVCAL on all cache blocks that include the specified area.

Format

```
Void syCacheMOVCAL( address, size )
Void *address
Uint32 size
```

Parameters

`address`	Starting address
`size`	Size

Return Value

None

Description

Executes `MOVCAL` on all cache blocks that include the specified area. To "allocate a cache block" means to write data only in the cache when there is a cache miss, without performing a block read.)

Note

If this function is executed when the start and end of the specified area do not define an area in units of 32 bytes, adjacent data will be damaged.

syCacheOCBI

Invalidates all cache blocks that include the specified area.

Format

```
Void syCacheOCBI( address, size )
Void *address
Uint32 size
```

Parameters

`address`	Starting address
`size`	Size

Return Value

None

Description

Invalidates all cache blocks that include the specified area. (To “invalidate” means to discard the contents of the cache.)

Note

If the address for P2 area (range from 0xa0000000 to 0xbfffffff) is specified, this area is replaced with the P0 area and is invalidated.

syCacheOCBP Purges all cache blocks that include the specified area.

Format

```
Void syCacheOCBP( address, size )
Void *address
Uint32 size
```

Parameters

`address`	Starting address
`size`	Size

Return Value

None

Description

Purges all cache blocks that include the specified area. (“To purge” means to discard the contents of the cache after a writeback.)

Note

If the address of P2 area (range from 0xa0000000 to 0xbfffffff) is specified, this area is replaced with the P0 area and is purged. In other words, even in the case of the noncacheable P2 area, the change is still made in the actual memory.

syCacheOCI Invalidates all entries in the operand cache.

Format

```
Void syCacheOCI( Void )
```

Parameters

None

Return Value

None

Description

Invalidates all entries in the operand cache.

Note

It is recommended to check header file which defines this function.

syCacheOCWB

Writes back all cache blocks that include the specified area.

Format

```
Void syCacheOCWB( address, size )
Void *address
Uint32 size
```

Parameters

`address`	Starting address
`size`	Size

Return Value

None

Description

Writes back all cache blocks that include the specified area. (“To write back” means to write the contents of the cache back to physical memory.)

Note

If the address of P2 area (range from 0xa0000000 to 0xbfffffff) is specified, this area is replaced with the P0 area and is written back. In other words, even in the case of the non-cacheable P2 area, the change is still made in the actual memory.

syCachePREF

Pre-fetch for the specified area.

Format

```
Void syCachePREF( address, size )
Void *address
Uint32 size
```

Parameters

address	Starting adrress
size	Size

Return Value

None

Description

Performs pre-fetch for the specified area.

5. Get Peripheral Data Function

pdExecPeripheralServer

Creates peripheral data for a frame.

Format

```
void pdExecPeripheralServer( void )
```

Parameters

None

Return Value

None

Description

Creates peripheral data for a frame.

Note

This function is called within the `njWaitVSync()` function. This is because this function is run in synchronization with frames. (In an application that runs at 2 Int, data is gotten once every 2 Int, so the button edge is gotten correctly.)

Use this function only when it is desired to get peripheral data according to timing other than the number of synchronized frames specified by `sbInitSystem()` function. In this case, it is not possible to guarantee the contents of the data in the press and release members of the `PDS_PERIPHERAL` structure that is gotten.

Reference

`njWaitVSync()`	Waits for a vertical sync interrupt
`njInitSystem()`	Initializes the graphics system

pdExitPeripheral

Termination processing of control port.

Format

```
void pdExitPeripheral( void )
```

Parameters

None

Return Value

None

Description

Ends peripheral library for control port.

pdGetPeripheral

Gets the controller button status.

Format

```
const PDS_PERIPHERAL *pdGetPeripheral( port )
Uint32 port
```

Parameters

port	Port number

Return Value

Pointer of structure that shows the controller button status

Description

Gets the status of buttons of peripherals connected to the control port.

Results are returned as the PDS_PERIPHERAL structure pointer in the return value.

The port numbers that are specified in port are listed below.

Definition	Meaning
PDD_PORT_A0	Control port A
PDD_PORT_B0	Control port B
PDD_PORT_C0	Control port C
PDD_PORT_D0	Control port D

Example

```
const PDS_PERIPHERAL *per;
per = pdGetPeripheral(PDD_PORT_A0);
if(per > press & PDD_DGT_ST) {
       /* Start botton was pressed */
           :
           :
}
```

Note

Please be noted returns cannot be lvalue as it has const attributes. (For example, it cannot be assigned.) Returns can be only referenced.

pdGetPeripheralDirect

Gets peripheral data (for low latency).

Format

```
const PDS_PERIPHERAL* pdGetPeripheralDirect( port,
buf, rawdata, rawinfo )
Uint32 port
PDS_PERIPHERAL *buf
void *rawdata
void *rawinfo
```

Parameters

`port`	Port number
`buf`	Pointer to structure to hold peripheral data obtained
`rawdata`	Must always be `NULL`
`rawinfo`	Must always be `NULL`

Return Value

Address specified by argument `buf`

Description

Gets peripheral data and stores it in the specified buffer.

The port number specified by the argument port has the following values.

Definition	Meaning
PDD_PORT_A0	Control port A
PDD_PORT_B0	Control port B
PDD_PORT_C0	Control port C
PDD_PORT_D0	Control port D

This function is used within a user interrupt handler which has been registered using the `pdSetIntFunction()` function.

Example

```
PDS_PERIPHERAL PadData;
pdGetPeripheralDirect(PDD_PORT_A0, &PadData, NULL, NULL);
```

Note

Always pass the same buffer to this function for any particular controller. This function creates the press and release members of the `PDS_PERIPHERAL` structure by reference to the member old in the same structure.

In other words it depends on the previous value. If different buffers are passed the press and release members will not be obtained correctly. Note that this operates entirely independently of the normal function for getting peripheral data, `pdGetPeripheral()`, and therefore does not affect the `pdGetPeripheral()` function.

This function can also be called at any time, not necessarily within an interrupt function. If VM file access, liquid crystal display, vibration device vibration, etc. is carried out, because of the long hardware processing time the maximum latency reduction effect cannot be obtained.

Reference

`pdGetPeripheral()`	Gets the controller button status
`pdSetIntFunction()`	Registers an interrupt handling function for a control port

pdGetPeripheralError

Gets controller data errors.

Format

```
Sint32 pdGetPeripheralError( port )
Uint32 port
```

Parameters

`port`	Port number

Return Value

`0`	No errors
`Negative value`	Error value

Description

Gets an error code related to the acquisition of controller data.

The port numbers that are specified in port are listed below.

Definition	Meaning
PDD_PORT_A0	Control port A
PDD_PORT_B0	Control port B
PDD_PORT_C0	Control port C
PDD_PORT_D0	Control port D

Returned error codes are as follows.

Error code	Content
PDD_ERR_OK	No error
PDD_ERR_RETRY	Button data cannot be gotten normally
PDD_ERR_GENERIC	Generic error

Example

```
Sint32 err;
err = pdGetPeripheralError(PDD_PORT_B0);
```

Note

If an error is generated, there is no recover procedure in the library. Use the appropriate process from the application.

pdGetPeripheralInfo

Gets data that is inherent to a peripheral.

Format

```
const PDS_PERIPHERALINFO *pdGetPeripheralInfo(
port )
Uint32 port
```

Parameters

port	Port number

Return Value

None

Description

Gets data that is inherent to a peripheral.

Port numbers specified in port are as follows.

Definition	Meaning
PDD_PORT_A0	Control port A peripheral device
PDD_PORT_A1	Expansion socket 1 of control port A
PDD_PORT_A2	Expansion socket 2 of control port A
PDD_PORT_B0	Control port B peripheral device
PDD_PORT_B1	Expansion socket 1 of control port B
PDD_PORT_B2	Expansion socket 2 of control port B
PDD_PORT_C0	Control port C peripheral device
PDD_PORT_C1	Expansion socket 1 of control port C
PDD_PORT_C2	Expansion socket 2 of control port C
PDD_PORT_D0	Control port D peripheral device
PDD_PORT_D1	Expansion socket 1 of control port D
PDD_PORT_D2	Expansion socket 2 of control port D

Example

```
const PDS_PERIPHERALINFO *info;
info = pdGetPeripheralInfo(PDD_PORT_A1);
if (info->type & PDD_DEVTYPE_LCD) {
       /* This peripheral has an LCD */
              :
}
if (info->area_code & PDD_DEVAREA_USA) {
       /* This peripheral is intended for North America */
              :
}
/* Display the product name and license */
njPrintC(NJM_LOCATION(0, 0), info->product_name);
njPrintC(NJM_LOCATION(0, 1), info->license);
```

Note

Because these members are added to the PDS_PERIPHERAL structure, even if this function is not used, the same operation can be done using the pdGetPeripheral() function.

Reference

pdGetPeripheral()	Gets the controller button status

pdInitPeripheral

Initializes control port.

Format

```
void pdInitPeripheral( plogic, recvbuf, sendbuf )
Sint32 plogic
void *recvbuf
void *sendbuf
```

Parameters

`plogic`	Logic mode
`recvbuf`	Receive buffer of control port
`sendbuf`	Send buffer of control port

Return Value

None

Description

Initializes library which uses controller group peripherals.

Parameter plogic which indicates logic mode can be specified whether digital button information of PDS structure is positive or negative logic.

Definition	Meaning
PDD_PLOGIC_ACTIVE	Positive logic
PDD_PLOGIC_NEGATIVE	Negative logic

48KB is needed by each buffer.

Example

```
user_init()
{
        pdInitPeripheral( PDD_PLOGIC_ACTIVE, recvbuf, sendbuf );
}
```

Note

The operation is not guaranteed if peripheral-related function is called before this function.

Reference

`PDS_PERIPHERAL()`	Structure where control pad status information is stored
`pdGetPeripheral()`	Gets the controller button status

pdInitPeripheralEx

Extension initialization of control port.

Format

```
void pdInitPeripheralEx( plogic, pertbl, recvbuf,
sendbuf, num )
Sint32 plogic
Sint32 *pertbl
void **recvbuf
void *sendbuf
Sint32 num
```

Parameters

`plogic`	Logic mode
`pertbl`	Peripherals in use table
`recvbuf`	Control port receive buffer address table
`sendbuf`	Control port send buffer address
`num`	Number of peripherals

Return Value

None

Description

Carries out extension initialization of the library using the controller peripheral.

The difference between this function and the `pdInitPeripheral()` function that regularly carries out initialization in that this function can reduce the buffer size and the time needed to access the port by restricting initialization to the port being used.

Selecting positive or negative logic for digital button information of the `PDS_PERIPHERAL` structure can be done in the argument plogic which specifies the logic mode.

Definition	Meaning
`PDD_PLOGIC_ACTIVE`	positive logic
`PDD_PLOGIC_NEGATIVE`	negative logic

Be sure to secure the necessary size for the receive and send buffers and set parameters so they don't contradict.

Example

```
/* Receive Buffer */
Uint8 gMapleRecvBuf[12][PDM_WORK_SIZE(1) + 32];
/* Send Buffer */
Uint8 gMapleSendBuf[PDM_WORK_SIZE(12) + 32];
const Sint32 gMapleDevs[] = {
       PDD_PORT_A0, PDD_PORT_A1, PDD_PORT_A2,
       PDD_PORT_B0, PDD_PORT_B1, PDD_PORT_B2,
       PDD_PORT_C0, PDD_PORT_C1, PDD_PORT_C2,
       PDD_PORT_D0, PDD_PORT_D1, PDD_PORT_D2,
};
const void* gMapleRecvBufTbl[] = {
       gMapleRecvBuf[0],/* A0 work area 2KB */
       gMapleRecvBuf[1],/* A1 work area 2KB */
       gMapleRecvBuf[2],/* A2 work area 2KB */
       gMapleRecvBuf[3],/* B0 work area 2KB */
       gMapleRecvBuf[4],/* B1 work area 2KB */
       gMapleRecvBuf[5],/* B2 work area 2KB */
       gMapleRecvBuf[6],/* C0 work area 2KB */
       gMapleRecvBuf[7],/* C1 work area 2KB */
       gMapleRecvBuf[8],/* C2 work area 2KB */
       gMapleRecvBuf[9],/* D0 work area 2KB */
       gMapleRecvBuf[10],/* D1 work area 2KB */
       gMapleRecvBuf[11],/* D2 work area 2KB */
};
pdInitPeripheralEx( PDD_PLOGIC_ACTIVE, gMapleDevs,
       gMapleRecvBufTbl, gMapleSendBuf,
       sizeof(gMapleDevs) / sizeof(gMapleDevs[0]) );
```

Reference

`pdInitPeripheral()`	Initializes control port
`pdExitPeripheral()`	Termination processing of control port

pdKbdExit

Ends keyboard library.

Format

```
void pdKbdExit( void )
```

Parameters

None

Return Value

None

Description

Ends keyboard library.

Reference

`pdKbdInit()`	Initializes keyboard library

pdKbdGetData

Gets keyboard data.

Format

```
const PDS_KEYBOARD *pdKbdGetData( port )
Uint32 port
```

Parameters

port	Control port number

Return Value

Pointer to structure where keyboard data is stored

Description

Gets information from the keyboard peripheral.

Port numbers to be specified in the argument port are as follow.

Definition	Meaning
PDD_PORT_A0	Control port A
PDD_PORT_B0	Control port B
PDD_PORT_C0	Control port C
PDD_PORT_D0	Control port D

A return value of NULL indicates the keyboard peripheral is not connected.

Example

```
Sint32 njUserMain(void)
{
PDS_KEYBOARD* kbd;
Sint32 i;
       /* Get port A keyboard data */
       kbd = pdKbdGetData(PDD_PORT_A0);
       /* If keyboard is connected */
       /* Display key code */
       if (kbd != NULL) {
          for (i = 0; i < 6; i++) {
             njPrintH(NJM_LOCATION(i * 4, 10), kbd->key[i]);
          }
       }
       return NJD_USER_CONTINUE;
}
```

Note

This function is for returning multiple acquisitions of key codes that are pressed at the same time.

For key input carried out by the application, data is gotten every 1 frame (1 INT), and key up, key down, key repeat processing must be carried out looking at the difference from the previous time.

Reference

`pdKbdInit()`	Initializes keyboard library
`pdKbdExit()`	Ends keyboard library
`pdKbdGetInfo()`	Gets hardware information of the keyboard

pdKbdGetInfo

Gets hardware information of the keyboard.

Format

```
const PDS_KEYBOARDINFO *pdKbdGetInfo( port )
Uint32 port
```

Parameters

por	Control port number

Return Value

Hardware information of the keyboard

Description

Gets hardware information (language, type, etc.) of the keyboard.
Port numbers to be specified in the argument port are as follow.

Definition	Meaning
PDD_PORT_A0	Control port A
PDD_PORT_B0	Control port B
PDD_PORT_C0	Control port C
PDD_PORT_D0	Control port D

Reference

pdKbdInit()	Initializes keyboard library
pdKbdExit()	Ends keyboard library

pdKbdInit

Initializes keyboard library.

Format

```
void pdKbdInit( void )
```

Parameters

None

Return Value

None

Description

Initializes keyboard library.

Reference

`pdKbdExit()`	Ends keyboard library

pdSetIntFunction

Registers an interrupt handling function for a control port.

Format

```
void pdSetIntFunction( func )
void *func
```

Parameters

`func`	User interrupt function address

Return Value

None

Description

Registers a user handling function for interrupts from a control port.

If the argument func is set to NULL, any existing registered function is canceled.

Example

```
void UserFunc(void)
{
       :
}
pdSetIntFunction(UserFunc);
```

Note

With the normal function for getting peripheral data, `pdGetPeripheal()`, from the actual data being received from the controller until it is reflected in the application there is a latency (delay) of 1V to 2V.

In normal applications there is absolutely no need to be aware of this delay, but applications running games over communications links, etc., are unable to ignore it because they require information from the controller as soon as possible for communications handling.

By using this function in combination with the `pdGetPeripheralDirect()` function, peripheral data can be obtained with the minimum delay at the time of the hardware interrupt occurring. Even during a user interrupt handling function, data obtained with the normal function for getting peripheral data, `pdGetPeripheral()`, is from the previous frame, and is not the minimum latency data. In the user interrupt handling function, use the `pdGetPeripheralDirect()` function.

Reference

`pdGetPeripheralDirect()`	Gets peripheral data (for low latency)

pdTmrAlarm

Sounds peripheral alarm.

Format

```
Sint32 pdTmrAlarm( port, data )
int32 port
Uint8 *data
```

Parameters

`port`	Port number
`data`	Alarm data(4 bytes)

Return Value

`PDD_TMRERR_OK`	"Get time" request success
`PDD_TMRERR_NO_TIMER`	No device with a timer is connected
`PDD_TMRERR_BUSY`	Device busy

Description

Sounds the alarm in a device, such as visual memory, that has a timer.

Of the four bytes of alarm data, the upper two bytes are for buzzer 1 and the lower two bytes are for buzzer 2. To prevent both buzzers from sounding, write `00h` to both of the two bytes. Port numbers specified in port are as follows.

Definition	Meaning
PDD_PORT_A1	Expansion socket 1 of control port A
PDD_PORT_A2	Expansion socket 2 of control port A
PDD_PORT_B1	Expansion socket 1 of control port B
PDD_PORT_B2	Expansion socket 2 of control port B
PDD_PORT_C1	Expansion socket 1 of control port C
PDD_PORT_C2	Expansion socket 2 of control port C
PDD_PORT_D1	Expansion socket 1 of control port D
PDD_PORT_D2	Expansion socket 2 of control port D

Example

```
Uint8  data[4];
Uint32 ret;
data[0] = 0xc0;
data[1] = 0x80;
data[2] = 0x00;
data[3] = 0x00;
ret = pdTmrAlarm(PDD_PORT_A1, data);
if (ret != PDD_TMRERR_OK) {
       /* Did not sound */
}
```

Note

Because only buzzer 1 is currently supported, do not use buzzer 2.

pdTmrGetTime

Gets time of timer device.

Format

```
Sint32 pdTmrGetTime( port, time, func, param )
Uint32 port
PDS_TIME *time
PD_TIMERCALLBACK func
Uint32 param
```

Parameters

port	Port number
time	Time data to get
func	Callback function called at completion of time aquisition
param	Parameter to be passed to callback function

Return Value

PDD_TMRERR_OK	Request of getting time success
PDD_TMRERR_NO_TIMER	Device with timer is not connected
PDD_TMRERR_BUSY	Device is busy

Description

Gets time from timer device in a peripheral such as Visual Memory.

Callback occurs regardless of return value.

Port numbers specified in port are as follows.

Definition	Meaning
PDD_PORT_A1	Expansion socket 1 of control port A
PDD_PORT_A2	Expansion socket 2 of control port A
PDD_PORT_B1	Expansion socket 1 of control port B
PDD_PORT_B2	Expansion socket 2 of control port B
PDD_PORT_C1	Expansion socket 1 of control port C
PDD_PORT_C2	Expansion socket 2 of control port C
PDD_PORT_D1	Expansion socket 1 of control port D PDD_PORT_D2 Expansion socket 2 of control port D

Example

```
PDS_TIME time;
ret = pdTmrGetTime( PDD_PORT_A1, &time, callback, 0 );
if (ret != PDD_TMRERR_OK) {
       /* Could not be set */
}
/* Callback function */
void callback(Sint32 stat, Uint32 param)
{
       switch (stat) {
       case PDD_TMRERR_OK:
       /* Setting completed */
       break;
       case PDD_TMRERR_NO_TIMER:
       /* Timer device being disconnected */
       break;
       case PDD_TMRERR_BUSY:
       /* Busy status */
       break;
       }
}
```

Note

Days of the week data is not set in Visual Memory. Therefore, days of the week data of `PDS_TIME` structure always shows Monday.

Reference

`pdTmrSetTime()`	Sets time of timer device

pdTmrIsReady

Checks connection of timer device.

Format

```
Sint32 pdTmrIsReady( port )
Uint32 port
```

Parameters

port	Port number

Return Value

TRUE	Connected
FALSE	Disconnected

Description

Checks whether a peripheral with timer device, such as Visual Memory, is connected or not.
Port numbers specified in port are as follows.

Definition	Meaning
PDD_PORT_A1	Expansion socket 1 of control port A
PDD_PORT_A2	Expansion socket 2 of control port A
PDD_PORT_B1	Expansion socket 1 of control port B
PDD_PORT_B2	Expansion socket 2 of control port B
PDD_PORT_C1	Expansion socket 1 of control port C
PDD_PORT_C2	Expansion socket 2 of control port C
PDD_PORT_D1	Expansion socket 1 of control port D
PDD_PORT_D2	Expansion socket 2 of control port D

Example

```
if (!pdTmrIsReady(PDD_PORT_A1)){
      return;
}
```

pdTmrSetTime

Sets time of timer device.

Format

```
Sint32 pdTmrSetTime( port, time, func, param )
Uint32 port
const PDS_TIME *time
PD_TIMERCALLBACK func
Uint32 param
```

Parameters

port	Port number
time	Time structure
func	Callback function called at completion of time aquisition
param	Parameter to be passed to callback function

Return Value

PDD_TMRERR_OK	Request of getting time success
PDD_TMRERR_NO_TIMER	Device with timer is not connected
PDD_TMRERR_BUSY	Device is busy

Description

Sets time of timer device in a peripheral such as Visual Memory.

Callback occurs regardless of return value.

Port numbers specified in port are as follows.

Definition	Meaning
PDD_PORT_A1	Expansion socket 1 of control port A
PDD_PORT_A2	Expansion socket 2 of control port A
PDD_PORT_B1	Expansion socket 1 of control port B
PDD_PORT_B2	Expansion socket 2 of control port B
PDD_PORT_C1	Expansion socket 1 of control port C
PDD_PORT_C2	Expansion socket 2 of control port C
PDD_PORT_D1	Expansion socket 1 of control port D
PDD_PORT_D2	Expansion socket 2 of control port D

Example

```
PDS_TIME time;
time.year = 1998;
time.month = 12;
time.day = 31;
time.hour = 23;
time.minute = 59;
time.second = 59;
time.dayofweek = dayofweek(1998, 12, 31);
ret = pdTmrSetTime(PDD_PORT_A1, &time, callback, 0);
if (ret != PDD_TMRERR_OK) {
       /* Could not be set */
}
       :
/* Callback function */
void callback(Sint32 stat, Uint32 param)
{
       switch (stat) {
          case PDD_TMRERR_OK:
             /* Setting completed */
             break;
          case PDD_TMRERR_NO_TIMER:
             /* Timer device being disconnected */
             break;
          case PDD_TMRERR_INVALID:
             /* Wrong setting value */
             break;
          case PDD_TMRERR_BUSY:
             /* Busy status */
             break;
       }
}
```

Note

Days of the week data is not set in Visual Memory.

Reference

`pdTmrGetTime()` Gets time of timer device

6. LCD Functions

pdLcdGetDirection

Detects the LCD device orientation.

Format

```
Sint32 pdLcdGetDirection( port )
Uint32 port
```

Parameters

`port`	Port number

Return Value

Orientation of connection

Description

Detects the orientation of how the LCD device is connected to the controller peripheral. This information is stored in each peripheral. The following values are specified for port.

Definition	Meaning
PDD_PORT_A1	Expansion socket 1 of control port A
PDD_PORT_A2	Expansion socket 2 of control port A
PDD_PORT_A3	Expansion socket 3 of control port A
PDD_PORT_A4	Expansion socket 4 of control port A
PDD_PORT_A5	Expansion socket 5 of control port A
PDD_PORT_B1	Expansion socket 1 of control port B
PDD_PORT_B2	Expansion socket 2 of control port B
PDD_PORT_B3	Expansion socket 3 of control port B
PDD_PORT_B4	Expansion socket 4 of control port B

Definition	Meaning
PDD_PORT_B5	Expansion socket 5 of control port B
PDD_PORT_C1	Expansion socket 1 of control port C
PDD_PORT_C2	Expansion socket 2 of control port C
PDD_PORT_C3	Expansion socket 3 of control port C
PDD_PORT_C4	Expansion socket 4 of control port C
PDD_PORT_C5	Expansion socket 5 of control port C
PDD_PORT_D1	Expansion socket 1 of control port D
PDD_PORT_D2	Expansion socket 2 of control port D
PDD_PORT_D3	Expansion socket 3 of control port D
PDD_PORT_D4	Expansion socket 4 of control port D
PDD_PORT_D5	Expansion socket 5 of control port D

The following values are returned as orientation of connection.

Definition	Meaning
PDD_LCD_DIRECTION_NORMAL	Normal
PDD_LCD_DIRECTION_FLIP	Upside down
PDD_LCD_DIRECTION_LEFT	Rotated 90 degrees to the left
PDD_LCD_DIRECTION_RIGHT	Rotated 90 degrees to the right

Example

```
Sint32 dir;
Uint32 flag;
dir = pdLcdGetDirection(PDD_PORT_A1);
switch (dir) {
case PDD_LCD_DIRECTION_NORMAL:                  /* Normal */
       flag = PDD_LCD_FLAG_NOFLIP;
       break;
case PDD_LCD_DIRECTION_FLIP:                    /* Because the LCD is upside down, */
       flag = PDD_LCD_FLAG_HVFLIP;              /* the data is sent upside down */
       break;
default:                                        /*If the LCD is oriented sideways, */
       flag = PDD_LCD_FLAG_NOFLIP;              /* the data is sent */
       return;                                  /* as is */
}
pdVmsLcdWrite(PDD_PORT_A1, cgdata, flag);
```

Note

When you insert visual memory in the expansion socket of Dreamcast controller, the display is flipped.

pdVmsLcdIsReady

Gets connection status of LCD device.

Format

```
Sint32 pdVmsLcdIsReady( port )
Uint32 port
```

Parameters

port	Port number

Return Values

TRUE	Connected
FALSE	Not connected

Description

Checks whether or not LCD device is connected to the specified expansion socket.
The following values are specified for port.

Definition	Meaning
PDD_PORT_A1	Expansion socket 1 of control port A
PDD_PORT_A2	Expansion socket 2 of control port A
PDD_PORT_A3	Expansion socket 3 of control port A
PDD_PORT_A4	Expansion socket 4 of control port A
PDD_PORT_A5	Expansion socket 5 of control port A
PDD_PORT_B1	Expansion socket 1 of control port B
PDD_PORT_B2	Expansion socket 2 of control port B
PDD_PORT_B3	Expansion socket 3 of control port B
PDD_PORT_B4	Expansion socket 4 of control port B
PDD_PORT_B5	Expansion socket 5 of control port B
PDD_PORT_C1	Expansion socket 1 of control port C
PDD_PORT_C2	Expansion socket 2 of control port C
PDD_PORT_C3	Expansion socket 3 of control port C
PDD_PORT_C4	Expansion socket 4 of control port C
PDD_PORT_C5	Expansion socket 5 of control port C
PDD_PORT_D1	Expansion socket 1 of control port D
PDD_PORT_D2	Expansion socket 2 of control port D
PDD_PORT_D3	Expansion socket 3 of control port D
PDD_PORT_D4	Expansion socket 4 of control port D
PDD_PORT_D5	Expansion socket 5 of control port D

Example

```
if (!pdVmsLcdIsReady(PDD_PORT_A1))
        return;
pdVmsLcdWrite(PDD_PORT_A1, cgdata, PDD_LCD_FLAG_HVFLIP);
```

pdVmsLcdWrite

Displays LCD device.

Format

```
Sint32 pdVmsLcdWrite( port, data, flag )
Uint32 port
void *data
Uint32 flag
```

Parameters

`port`	Port number
`data`	Pixel data address
`flag`	Flag

Return Values

`PDD_LCDERR_OK`	Display succeeded
`PDD_LCDERR_NO_LCD`	LCD device is not connected
`PDD_LCDERR_BUSY`	LCD device is busy

Description

Displays bitmap graphics on the LCD device.
The following values are specified for port.

Definition	Meaning
PDD_PORT_A1	Expansion socket 1 of control port A
PDD_PORT_A2	Expansion socket 2 of control port A
PDD_PORT_A3	Expansion socket 3 of control port A
PDD_PORT_A4	Expansion socket 4 of control port A
PDD_PORT_A5	Expansion socket 5 of control port A
PDD_PORT_B1	Expansion socket 1 of control port B
PDD_PORT_B2	Expansion socket 2 of control port B
PDD_PORT_B3	Expansion socket 3 of control port B
PDD_PORT_B4	Expansion socket 4 of control port B
PDD_PORT_B5	Expansion socket 5 of control port B
PDD_PORT_C1	Expansion socket 1 of control port C
PDD_PORT_C2	Expansion socket 2 of control port C
PDD_PORT_C3	Expansion socket 3 of control port C
PDD_PORT_C4	Expansion socket 4 of control port C
PDD_PORT_C5	Expansion socket 5 of control port C
PDD_PORT_D1	Expansion socket 1 of control port D

Definition	Meaning
PDD_PORT_D2	Expansion socket 2 of control port D
PDD_PORT_D3	Expansion socket 3 of control port D
PDD_PORT_D4	Expansion socket 4 of control port D
PDD_PORT_D5	Expansion socket 5 of control port D

Specification of LCD device is as follows.

- Resolution: H 48xV 32 pixels
- Intensity: Intensity 2 (monochrome)

The pixel data format for the argument `data` is expressed by 8 bits (1 byte) per pixel for easier handling by the software. The intensity is expressed in the lower 4 bits, but in this library only bit 3 is referred.

- For pixels that are white (LCD base color), store data (0x00, 0x07, etc.) as 0 in bit 3.
- For pixels that are black, store data (0x0f, 0x08, etc.) as 1 in bit 3.

For a full LCD screen, pixel data has 48x32=1536 bytes.

Pixel data is stored continuously from left to right, top to bottom. The following is a memory image example of pixel data.

0x8C100000	(0, 0)	(1, 0)	(2, 0)	...	(47, 0)
0x8C100030	(0, 1)	(1, 1)	(2, 1)	...	(47, 1)
	:	:	:		:
0x8C1005D0	(0, 31)	(1, 31)	(2, 31)	...	(47, 31)

For a software recording, the following method can be used.

```
const Uint8 cgdata[48 * 32] =
{
0x00, 0x00, 0x0f, 0x0f, ...
...
};
```

The following values are specified for flag.

Definition	Meaning
PDD_LCD_FLAG_NOFLIP	Displays data as is
PDD_LCD_FLAG_HFLIP	Displays data with a horizontal flip
PDD_LCD_FLAG_VFLIP	Displays data with a vertical flip
PDD_LCD_FLAG_HVFLIP	Displays data with a horizontal and vertical flip

Example

```
const Uint8 cgdata[48 * 32] = {
       0x00, 0xff, ...
:
};
ret = pdVmsLcdWrite(PDD_PORT_A1, cgdata, PDD_LCD_HVFLIP);
if (ret != PDD_LCDERR_OK) {
       /* Display not possible */
}
```

pdVmsLcdWrite1

Displys LCD device (low level).

Format

```
Sint32 pdVmsLcdWrite1( port, data )
Uint32 port
void *data
```

Parameters

port	Port number
data	Address of pixel data

Return Value

PDD_LCDERR_OK	Succeeded
PDD_LCDERR_NO_LCD	LCD device is not connected
PDD_LCDERR_BUSY	LCD device is busy

Description

Displays graphics directly to the LCD device.

This function sends and displays data of 1 bit 1 pixel on the LCD device as is.

The following values are specified for port.

Definition	Meaning
PDD_PORT_A1	Expansion socket 1 of control port A
PDD_PORT_A2	Expansion socket 2 of control port A
PDD_PORT_A3	Expansion socket 3 of control port A
PDD_PORT_A4	Expansion socket 4 of control port A
PDD_PORT_A5	Expansion socket 5 of control port A
PDD_PORT_B1	Expansion socket 1 of control port B
PDD_PORT_B2	Expansion socket 2 of control port B
PDD_PORT_B3	Expansion socket 3 of control port B
PDD_PORT_B4	Expansion socket 4 of control port B
PDD_PORT_B5	Expansion socket 5 of control port B
PDD_PORT_C1	Expansion socket 1 of control port C
PDD_PORT_C2	Expansion socket 2 of control port C
PDD_PORT_C3	Expansion socket 3 of control port C
PDD_PORT_C4	Expansion socket 4 of control port C
PDD_PORT_C5	Expansion socket 5 of control port C
PDD_PORT_D1	Expansion socket 1 of control port D
PDD_PORT_D2	Expansion socket 2 of control port D

Definition	Meaning
PDD_PORT_D3	Expansion socket 3 of control port D
PDD_PORT_D4	Expansion socket 4 of control port D
PDD_PORT_D5	Expansion socket 5 of control port D

This function differs from `pdVmsLcdWrite()` fuction on th following points.

The performance is quicker due to no conversion of data format. Data amount can be lessened to one-eighth. As data does not need to be converted, there is no reverse function of data corresponding to the orientation of LCD device. In case reverse function is needed, it needs to be done on the application.

Example

```
ret = pdVmsLcdWrite1(PDD_PORT_A1, &lcddata);
       if (ret != PDD_LCDERR_OK) {
              :
           /* Display not possible */
              :
       }
```

Reference

`pdVmsLcdWrite()` Displays LCD device

7. Timer Functions

syTmrCountToMicro

Converts a counter value into a microseconds value.

Format

```
Uint32 syTmrCountToMicro( count )
Uint32 count
```

Parameters

`count`	Counter value to be converted

Return Value

Microseconds value

Description

Converts a counter value into a microseconds value which can be get by `syTmrGetCount()` or `syTmrDiffCount()` function.

Note

Decimals are truncated.

Reference

`syTmrGetCount()`	Gets the free running timer value
`syTmrDiffCount()`	Gets the difference of two counter values

syTmrDiffCount

Gets the difference of two counter values.

Format

```
Uint32 syTmrDiffCount( count1, count2 )
Uint32 count1
Uint32 count2
```

Parameters

`count1`	Counter value that was gotten first
`count2`	Counter value that was gotten second

Return Value

Difference between the two counter values

Description

Calculates the difference between two counter values. Specify the counter value, not microseconds, in the parameter.

Reference

`syTmrGetCount()`	Gets the free running timer value
`syTmrCountToMicro()`	Converts a counter value into a microseconds value

syTmrGenCancelInt

Cancels interrupts by a general-purpose timer.

Format

```
void syTmrGenCancelInt( void )
```

Parameters

None

Return Value

None

Description

Disables the interrupt callback that was set by `syTmrGenSetInt()`

syTmrGenCancelInt

Cancels interrupts by a general-purpose timer.

Format

```
void syTmrGenCancelInt( void )
```

Parameters

None

Return Value

None

Description

Disables the interrupt callback that was set by `syTmrGenSetInt()`

syTmrGenCountToMicro

Converts general-purpose counter values to microseconds.

Format

```
Uint32 syTmrGenCountToMicro( count, clock )
Uint32 count
SY_TMR_CLOCK clock
```

Parameters

`count`	Counter value to be converted
`clock`	Counter clock

Return Value

microseconds

Description

Converts general-purpose counter values to microseconds.

The following values can be set in argument clock.

Value	Meaning	Time equivalent to 1 count	Upper limit of counter
SYD_TMR_CLOCK_P4	Phi/4	0.08 microseconds	Approx. 5.7 min.
SYD_TMR_CLOCK_P16	Phi/16	0.32 microseconds	Approx. 22.9 min.
SYD_TMR_CLOCK_P64	Phi/64	1.28 microseconds	Approx. 91.6 min.
SYD_TMR_CLOCK_P1024	Phi/1024	20.48 microseconds	Approx. 24 hrs. 26 min.

Note

Decimals are truncated.

syTmrGenDiffCount

Calculates the difference between general-purpose interrupt counters.

Format

```
Uint32 syTmrGenDiffCount( count1, count2 )
Uint32 count1
Uint32 count2
```

Parameters

`count1`	Counter value that was gotten first
`count2`	Counter value that was gotten second

Return Value

Counter value

Description

Gets the difference between two timer counter values. Specify the counter value, not microseconds, in the parameter.

Note

Decimals are truncated.

syTmrGenGetCount Gets general-purpose timer interrupt counter.

Format

```
Uint32 syTmrGenGetCount( void )
```

Parameters

None

Return Value

Counter value

Description

Gets the general-purpose timer counter value. Counter clock is 1.28 microseconds per 1.28 counts.

Reference

`syTmrCountToMicro()`	Converts a counter value into a microseconds value

syTmrGenMicroToCount

Converts microseconds value to general counter value.

Format

```
Uint32 syTmrGenMicroToCount( micro, clock )
Uint32 micro
SY_TMR_CLOCK clock
```

Parameters

`micro`	Microseconds value to be converted
`clock`	Counter clock

Return Value

Counter value

Description

Converts microseconds value to general counter value.

The following values is specified in argument clock.

Value	Meaning	Time equivalent to 1 count	Upper limit of counter
SYD_TMR_CLOCK_P4	Phi/4	0.08 microseconds	Approx. 5.7 min.
SYD_TMR_CLOCK_P16	Phi/16	0.32 microseconds	Approx. 22.9 min.
SYD_TMR_CLOCK_P64	Phi/64	1.28 microseconds	Approx. 91.6 min.
SYD_TMR_CLOCK_P1024	Phi/1024	20.48 microseconds	Approx. 24 hrs. 26 min.

Note

Decimals are truncated.

Reference

`syTmrGenCountToMicro()`	Converts general-purpose counter values to microseconds

syTmrGenSetClock

Sets the base clock for the general-purpose timer.

Format

```
void syTmrGenSetClock( clock )
SY_TMR_CLOCK clock
```

Parameters

`clock` Counter clock

Return Value

None

Description

Sets the base clock for the general-purpose timer.

The base clock is given as a fraction of the CPU's peripheral module clock(50MHz); four settings are possible. The following values are specified for argument clock.

Value	Meaning	Time equivalent to 1 count	Upper limit of counter
SYD_TMR_CLOCK_P4	Phi/4	0.08 microseconds	Approx. 5.7 min.
SYD_TMR_CLOCK_P16	Phi/16	0.32 microseconds	Approx. 22.9 min.
SYD_TMR_CLOCK_P64	Phi/64	1.28 microseconds	Approx. 91.6 min.
SYD_TMR_CLOCK_P1024	Phi/1024	20.48 microseconds	Approx. 24 hrs. 26 min.

Because the counter value is 32 bits, the time needed in order to get a unique value depends on the counter clock that is set. The default is "divided by 64", so that 100000000h x 1.28 micro=5497.558 seconds(=approximately 91 minutes and 38 seconds).

syTmrGenSetCount

Sets the general-purpose timer interrupt counter.

Format

```
void syTmrGenSetCount( count )
Uint32 count
```

Parameters

`count`	Value to be set in the counter

Return Value

None

Description

Sets the general-purpose timer counter value.

syTmrGenSetInt

Authorizes interrupts by a general-purpose timer.

Format

```
void syTmrGenSetInt( func, param )
void (*func)(void *)
void *param
```

Parameters

`func`	Function that is called when the counter value reaches "0"
`param`	Pointer to parameter handed to function

Return Value

None

Description

The general-purpose timer is a subtraction-type timer that is designed to generate a timer interrupt once the timer count value reaches "0." This function registers the function that is called back when this timer interrupt is generated. Once the registered function is called, the function registration becomes invalid. Therefore, in order to consecutively call timer interrupts, it is necessary to register the callback function each time with this function. If `NULL` is specified in the parameter, wait until the counter reaches 0 to exit this function.

Example

```
/* Callback function example */
void IntFunc(void* param)
{
       /* Save the parameters */
       IntCount = param;

       /* Continue interrupt generation */
       syTmrGenSetInt(IntFunc, param+1);

       /* Set the initial value in the counter*/
       syTmrGenSetCount(0-10000);

       /* Restart the timer */
       syTmrGenStart();
}
```

Note

If this function is called when an interrupt callback function has already been registered, only the function that was registered last is valid. If the timer is stopped and this function is called with `NULL` specified for the callback function, start up the timer within this function.

syTmrGenStart

Starts the general-purpose timer.

Format

```
void syTmrGenStart( void )
```

Parameters

None

Return Value

None

Description

Starts the general-purpose timer.

syTmrGenStop

Stops the general-purpose timer.

Format

```
void syTmrGenStop( void )
```

Parameters

None

Return Value

None

Description

Stops the general-purpose timer.

syTmrGetCount

Gets the free running timer value.

Format

```
Uint32 syTmrGetCount( void )
```

Parameters

None

Return Value

`TMU0`	Counter value

Description

Gets the `TMU0` counter value.

Example

```
main()
{
       Uint32 count1, count2, count, micro;
       /* Start measurement */
       count1 = syTmrGetCount();
       /* Execute function that is the target */
       TestFunc();
       /* End measurement */
       count2 = syTmrGetCount();
       /* Get difference in counter values */
       count = syTmrDiffCount(count1, count2);
       /* Convert counter value to microseconds */
       micro = syTmrCountToMicro(count);
       while (1);
}
```

Note

Because timer 0 is a down counter, this function returns the value (0xffffffff-counter value).

syTmrMicroToCount

Converts a microseconds value to a counter value.

Format

```
Uint32 syTmrMicroToCount( micro )
Uint32 micro
```

Parameters

`micro`	Microseconds value to be converted

Return Value

Counter value

Description

Converts a microseconds value to a counter value.

Note

Decimals are truncated.

8. Real-time Clock Functions

syRtcCompareDate

Compares date and time.

Format

```
Sint32 syRtcCompareDate( date1, date2 )
const SYS_RTC_DATE *date1
const SYS_RTC_DATE *date2
```

Parameters

`date1`	Pointer for date and time 1 to be compared
`date2`	Pointer for date and time 2 to be compared

Return Value

`-1`	"date1" is smaller
`0`	equal
`+1`	"date1" is larger

Description

Compares two dates and times. The larger value is the more recent.

syRtcCountToDate

Converts a count into a date and time value.

Format

```
void syRtcCountToDate( count, date )
const Uint32 count
SYS_RTC_DATE *date
```

Parameters

`count`	Count to be converted
`date`	Pointer to area where the converted date and time is stored

Return Value

None

Description

Converts a count(one count=one second) into a date and time value.

syRtcDateToCount

Converts a date and time value into a count.

Format

```
void syRtcDateToCount( date, count )
const SYS_RTC_DATE *date
Uint32 *count
```

Parameters

`date`	Pointer for the date and time to be converted
`count`	Pointer for storing count value after conversion

Return Value

None

Description

Converts a date and time value into a count. One count is equal to one second.

Note

The dayofweek and ageofmoon members in the parameter *date (`SYS_RTC_DATE` structure) are not referenced.

syRtcExecServer

Updates internal information concerning the date and time (server function).

Format

```
void syRtcExecServer( void )
```

Parameters

None

Return Value

None

Description

Updates internal information concerning the date and time.

Note

Because this function is automatically called during V-`BlankIn`, an application never explicitly calls this function.

When the server function is stopped, the value that is gotten by the `syRtcGetDate()` function is no longer updated. The `syRtcSetDate()` function is executed even while the server function is stopped, but afterwards, even if the `syRtcGetDate()` function is executed, the new time is not returned.

syRtcFinish — Exits the real-time clock functions.

Format

```
void syRtcFinish( void )
```

Parameters

None

Return Value

None

Description

Exits the real-time clock functions and releases the work area.

syRtcGetDate

Gets the date and time.

Format

```
Sint32 syRtcGetDate( date )
SYS_RTC_DATE *date
```

Parameters

`date`	Pointer to current date and time

Return Value

`SYD_RTC_STAT_ACTIVE`	Accesses the real-time clock
`SYD_RTC_STAT_PASSIVE`	Doesn't access the real-time clock

Description

Gets the date and time from the real-time clock.

syRtcGetStat

Gets the server function status.

Format

```
Sint32 syRtcGetStat( void )
```

Parameters

None

Return Value

SYD_RTC_STAT_ACTIVE	Accesses the real-time clock
SYD_RTC_STAT_PASSIVE	Doesn't access the real-time clock

Description

Gets the server function activity status.

Note

By default, the RTC server functions are running (SYD_RTC_STAT_ACTIVE).

syRtcInit

Initializes the real-time clock functions.

Format

```
Sint32 syRtcInit( void )
```

Parameters

None

Return Value

SYD_RTC_ERR_OK	Normal termination
SYD_RTC_ERR_FATAL	Abnormal termination

Description

Initializes the RTC functions and allocates a work area.

Real-time clock functions accesses, gets date and time information from, and sets the real-time clock AICA on the G2 bus.

AICA's real-time clock has a 32-bit counter with 1 second counting as 1 count.

syRtcSetDate

Sets the date and time.

Format

```
Sint32 syRtcSetDate( date )
const SYS_RTC_DATE *date
```

Parameters

`date`	Pointer for setting the date and time

Return Value

None

Description

Sets the date and time.

The dayofweek and ageofmoon members in the structure are not referenced. The AICA internal registers are updated even when the server functions are stopped.

Note

Except when the player sets the time from the main menu when Dreamcast boots up, don't set the time from the application except in special circumstances.

syRtcSetServerMode

Sets server release operation.

Format

```
void syRtcSetServerMode( mode )
const Sint32 mode
```

Parameters

mode	Server function operation

Return Value

None

Description

Sets the server function activity status.

The operation that sets the server function in mode can specify the following statuses.

Definition	Meaning
SYD_RTC_STAT_ACTIVE	Accesses the real-time clock (default)
SYD_RTC_STAT_PASSIVE	Doesn't access the real-time clock

9. Backup Functions (Memory Card)

buAnalyzeBackupFileImage

Analyzes the file image in memory.

Format

```
Sint32 buAnalyzeBackupFileImage( hdr, buf )
BUS_BACKUPFILEHEADER *hdr
const void *bufInputs
```

Parameters

hdr	Address to store memory card file header image
buf	Address where the file to be analyzed was loaded (4 byte boundary)

Return Value

BUD_ERR_OK	Normal termination
BUD_ERR_BUPFILE_ILLEGAL	File is not in the correct format.
BUD_ERR_BUPFILE_CRC	CRC error

Description

This function analyzes the data in memory and creates the BUS_BACKUPFILEHEADER structure.

Example

```
Sint32 ret, nblock;
extern Uint8 buf[];
BUS_BACKUPFILEHEADER hdr;
ret = buLoadFile(BUD_DRIVE_A1, "SAVEDATA_001", buf, 0);

if (ret != BUD_ERR_OK)
       return NG;

while (1) {
```

```
        if (buStat(BUD_DRIVE_A1) == BUD_STAT_READY) break;
}

ret = buAnalyzeBackupFileImage(&hdr, buf);
switch (ret) {
        case BUD_ERR_OK:
                return OK;
        default:
                return NG;
}
```

Note

The address that is specified for `buf` must be on a four-byte boundary.

buCalcBackupFileSize

Calculates file size of image.

Format

```
Sint32 buCalcBackupFileSize( inum, vtype, size )
Uint32 inum
Uint32 vtype
Uint32 size
```

Parameters

inum	Number of icons
vtype	Visual type size Number of bytes of data to be saved for applications

Return Value

Positive value	Number of blocks used in the memory card file
BUD_ERR_INVALID_PARAM	Invalid parameter

Description

Calculates the size of a memory card file format image.

Example

```
/* Game name (sort item) */
extern Uint8 game_name[];

/* Icon palette data */
extern Uint16 icon_palette[];

/* Icon pixel data */
extern Uint8 icon_data[];

/* Visual data */
extern Uint8 visual_data[];

/* Application save data */
extern Uint8 save_data[];
Sint32 nblock1, nblock2;
void* buf;

/* Memory card file header */
BUS_BACKUPFILEHEADER hdr;
memset(&hdr, 0, sizeof(hdr));

/* Visual memory comment setting */
strcpy(hdr.vms_comment, "VMS_COMMENT");

/* BootROM comment setting */
```

```
strcpy(hdr.btr_comment, "BOOT ROM Comment");

/* Game name (sort item) setting */
memcpy(hdr.game_name,   game_name, 16);

/* Icon palette data setting */
hdr.icon_palette = icon_palette;

/* Icon pixel data setting */
hdr.icon_data    = icon_data;

/* Number of icons setting */
hdr.icon_num     = 1;

/* Icon speed setting */
hdr.icon_speed   = 1;

/* Visual comment setting */
hdr.visual_data  = visual_data;

/* Visual type setting */
hdr.visual_type  = BUD_VISUALTYPE_C;

/* Application save data */
hdr.save_data    = save_data;

/* Application save data size */
hdr.save_size    = 0x400;
nblock = buCalcBackupFileSize(hdr.icon_num,
       hdr.visual_type, hdr.save_size);

/* Allocates (the number of blocks to be used x 512 bytes) */
buf = syMalloc(nblock * 512);
if (buf == NULL) {
       /* Memory cannot be allocated */
       return NG;
}
nblock = buMakeBackupFileImage(buf, &hdr);
if (nblock < 0) {
       /* Structure includes an invalid parameter */
       return NG;
}
```

buDefragDisk

Optimizes a memory card.

Format

```
Sint32 buDefragDisk( drive, work )
Sint32 drive
void *work
```

Parameters

drive	Expansion socket number
work	Work buffer (512 bytes)

Return Value

BUD_ERR_OK	Processing request was received
BUD_ERR_BUSY	Busy; request could not be accepted

Description

Optimizes the memory of a memory card in order to permit allocation of an area where an executable file can be written.

The following values can be designated for drive that specifies expansion sockets.

Definition	Meaning
BUD_DRIVE_A1	Expansion socket 1 of control port A
BUD_DRIVE_A2	Expansion socket 2 of control port A
BUD_DRIVE_B1	Expansion socket 1 of control port B
BUD_DRIVE_B2	Expansion socket 2 of control port B
BUD_DRIVE_C1	Expansion socket 1 of control port C
BUD_DRIVE_C2	Expansion socket 2 of control port C
BUD_DRIVE_D1	Expansion socket 1 of control port D
BUD_DRIVE_D2	Expansion socket 2 of control port D

Set aside 512 bytes as workspace for the argument work.

Use this function when you have enough free space to save an executable file, but cannot reserve a block of free space large enough to save the file.

Example

```
Sint32 ret;
Uint32 DefragWork[512 / sizeof(Uint32)];
ret = buDefragDisk(BUD_DRIVE_A1, DefragWork);
if (ret != BUD_ERR_OK)
       return NG;
while (buStat(BUD_DRIVE_A1) == BUD_STAT_BUSY) {
}
if (buGetLastError(BUD_DRIVE_A1) != BUD_ERR_OK)
       return NG;
return OK;
```

Note

Do not change the size of the work buffer (as indicated by `work`) while optimization processing is in progress.

In addition, `work` must specify a 512-byte buffer that begins on a four-byte boundary.

If the memory card is removed while this function is being executed, in most cases all data will be lost. In application program, prohibition of “inserting or removing the memory card while this function is being executed” needs to be notified in any ways.

buDeleteFile

Deletes a file.

Format

```
Sint32 buDeleteFile( drive, fname )
Sint32 drive
const char *fname
```

Parameters

drive	Expansion socket number
fname	File name

Return Value

BUD_ERR_OK	Processing request was accepted
BUD_ERR_BUSY	Request could not be accepted because processing was in progress

Description

Erases a file on a memory card inserted in a specified expansion socket.

The following values can be designated for the argument drive that specifies expansion sockets.

Definition	Meaning
BUD_DRIVE_A1	Expansion socket 1 of control port A
BUD_DRIVE_A2	Expansion socket 2 of control port A
BUD_DRIVE_B1	Expansion socket 1 of control port B
BUD_DRIVE_B2	Expansion socket 2 of control port B
BUD_DRIVE_C1	Expansion socket 1 of control port C
BUD_DRIVE_C2	Expansion socket 2 of control port C
BUD_DRIVE_D1	Expansion socket 1 of control port D
BUD_DRIVE_D2	Expansion socket 2 of control port D

Example

```
Sint32 ret;
ret = buDeleteFile(BUD_DRIVE_A1, "SAVEDATA");
if (ret == BUD_ERR_OK) {
       /* Delete request was successful */
} else {
       /* Delete request failed (BUSY) */
}
```

Note

The following are the completion statuses for this function that are obtained by `buStat()`:

Status	Meaning
BUD_ERR_OK	Normal termination
BUD_ERR_NO_DISK	Memory card not found
BUD_ERR_UNFORMAT	Disk is not formatted
BUD_ERR_FILE_NOT_FOUND	File not found

A deletion generally requires 5 Int. (This time requirement may vary, depending on the status of other drives, etc.)

buExit

Exits the memory card file system.

Format

```
Sint32 buExit( void )
```

Parameters

None

Return Value

BUD_ERR_OK	Normal termination
BUD_ERR_BUSY	A TYPE_B function is being processed

Description

Exits the memory card file system.

Example

```
do {
} while(buExit() != BUD_ERR_OK);
```

Note

It is not possible to exit from the backup file system if some type of processing is in progress, so BUD_ERR_BUSY is returned.

Reference

buInit()	Initializes the memory card file system

buFindExecFile

Gets the name of an executable file.

Format

```
Sint32 buFindExecFile( drive, fname )
Sint32 drive
char *fname
```

Parameters

drive	Expansion socket number
fname	Pointer to get file name

Return Value

BUD_ERR_OK	Normal termination
BUD_ERR_FILE_NOT_FOUND	Executable file not found
BUD_ERR_UNFORMAT	Disk is not formatted
BUD_ERR_NO_DISK	Memory card not found
BUD_ERR_BUSY	A TYPE_B function was being processed

Description

Gets the name of an executable file.

The following values can be designated for the argument drive that specifies expansion sockets.

Definition	Meaning
BUD_DRIVE_A1	Expansion socket 1 of control port A
BUD_DRIVE_A2	Expansion socket 2 of control port A
BUD_DRIVE_B1	Expansion socket 1 of control port B
BUD_DRIVE_B2	Expansion socket 2 of control port B
BUD_DRIVE_C1	Expansion socket 1 of control port C
BUD_DRIVE_C2	Expansion socket 2 of control port C
BUD_DRIVE_D1	Expansion socket 1 of control port D
BUD_DRIVE_D2	Expansion socket 2 of control port D

Example

A minimum of 13 bytes is required for the area where the file name is to be stored. If the file does not exist, then fname[0] = '\0' results. This function is completely independent of the `buFindFirstFile()` and `buFindNextFile()` functions, and can be used without affecting the operation of either of those functions.

```
Sint32 ret;
char fname[16];
/* Get name of executable file */
ret = buFindExecFile(BUD_DRIVE_A1, fname);
switch (ret) {
        case BUD_ERR_OK:
                /* Executable file found.
                The file name is stored in fname.
                */
                printf("Executable file: %s\n", fname);
                break;
        case BUD_ERR_FILE_NOT_FOUND:
                printf("Executable file not found. \n");
                break;
        default:
                printf("Error \n");
                break;
}
```

buFindFirstFile

Gets the name of the first file.

Format

```
Sint32 buFindFirstFile( drive, fname )
Sint32 drive
char *fname
```

Parameters

drive	Expansion socket number
fname	Address that gets file name

Return Value

BUD_ERR_OK	Normal termination
BUD_ERR_FILE_NOT_FOUND	File not found
BUD_ERR_UNFORMAT	Disk is not formatted
BUD_ERR_NO_DISK	No memory card found
BUD_ERR_BUSY	A TYPE_B function is being processed

Description

Gets the name of the first file on a memory card inserted in a specified expansion socket.

The following values can be designated for the argument drive that specifies expansion sockets.

Definition	Meaning
BUD_DRIVE_A1	Expansion socket 1 of control port A
BUD_DRIVE_A2	Expansion socket 2 of control port A
BUD_DRIVE_B1	Expansion socket 1 of control port B
BUD_DRIVE_B2	Expansion socket 2 of control port B
BUD_DRIVE_C1	Expansion socket 1 of control port C
BUD_DRIVE_C2	Expansion socket 2 of control port C
BUD_DRIVE_D1	Expansion socket 1 of control port D
BUD_DRIVE_D2	Expansion socket 2 of control port D

If the file does not exist, then fname[0] = '\0' results.

Normally, this function is used in combination with `buFindNextFile()` function to get the names of all files in a drive.

Example

```
Sint32 ret, files, blocks, totel;
char fname[16];
BUS_FILEINFO info;
files = blocks = total = 0;

/* Get name of first file */
ret = buFindFirstFile(drive, fname);
if (ret < 0) {
        if (ret == BUD_ERR_FILE_NOT_FOUND)
                goto end;
        else
                goto err;
}
if (buGetFileInfo(drive, fname, &info) < 0)
        goto err;
blocks = info.blocks;
total += blocks;
files++;
do {
        printf("%12s %10d bytes(%3d blocks)\n",
                        files, blocks * 512, blocks);
        /* Get name of second and subsequent files */
        ret = buFindNextFile(drive, fname);
        if (ret < 0) {
                if (ret == BUD_ERR_FILE_NOT_FOUND)
                        goto end;
                else
                        goto err;
        }
        if (buGetFileInfo(drive, fname, &info) < 0)
                goto err;
        blocks = info.blocks;
        total += blocks;
        files++;
} while (files < FILE_MAX);

end:
return OK;

err:
return NG;
```

Note

A minimum of 13 bytes is required for the area where the file name is to be stored.

buFindNextFile

Gets the name of the next file.

Format

```
Sint32 buFindNextFile( drive, fname )
Sint32 drive
char *fname
```

Parameters

drive	Extended socket number
fname	Address that gets file name

Return Value

BUD_ERR_OK	Normal termination
BUD_ERR_FILE_NOT_FOUND	File not found
BUD_ERR_UNFORMAT	Disk is not formatted
BUD_ERR_NO_DISK	No memory card found
BUD_ERR_BUSY	A TYPE_B function is being processed

Description

Gets the name of the next file previously acquired from a memory card inserted in a specified expansion socket.

The following values can be designated for the argument drive that specifies expansion sockets.

Definition	Meaning
BUD_DRIVE_A1	Extended socket 1 of control port A
BUD_DRIVE_A2	Extended socket 2 of control port A
BUD_DRIVE_B1	Extended socket 1 of control port B
BUD_DRIVE_B2	Extended socket 2 of control port B
BUD_DRIVE_C1	Extended socket 1 of control port C
BUD_DRIVE_C2	Extended socket 2 of control port C
BUD_DRIVE_D1	Extended socket 1 of control port D
BUD_DRIVE_D2	Extended socket 2 of control port D

Normally, this function is used in combination with `buFindFirstFile()` function to get the names of all files in a drive.

Example

```
Sint32 ret, files, blocks, totel;
char fname[16];
BUS_FILEINFO info;
files = blocks = total = 0;

/* Get name of first file */
ret = buFindFirstFile(drive, fname);
if (ret < 0) {
        if (ret == BUD_ERR_FILE_NOT_FOUND)
                goto end;
        else
                goto err;
}
if (buGetFileInfo(drive, fname, &info) < 0)
        goto err;
blocks = info.blocks;
total += blocks;
files++;
do {
        printf("%12s %10d bytes(%3d blocks)\n",
                files, blocks * 512, blocks);
        /* Get name of second and subsequent files */
        ret = buFindNextFile(drive, fname);
        if (ret < 0) {
                if (ret == BUD_ERR_FILE_NOT_FOUND)
                        goto end;
                else
                        goto err;
        }
        if (buGetFileInfo(drive, fname, &info) < 0)
                goto err;
        blocks = info.blocks;
        total += blocks;
        files++;
} while (files < FILE_MAX);

end:
return OK;

err:
return NG;
```

Note

A minimum of 13 bytes is required for the area where the file name is to be stored. If the file does not exist, then fname[0] = '\0' results.

buFormatDisk

Formats a memory card.

Format

```
Sint32 buFormatDisk( drive, volume, icon, time, flag )
Sint32 drive
const Uint8 *volume
Sint32 icon
const BUS_TIME *time
Uint32 flag
```

Parameters

drive	Expansion socket number
volume	Volume data
icon	Icon number(0 to 123)
time	Time stamp
flag	Format flag

Return Value

BUD_ERR_OK	Processing request was accepted
BUD_ERR_BUSY	Request could not be accepted because processing was in progress

Description

Formats a memory card. The following values can be designated for the argument drive that specifies expansion sockets.

Definition	Meaning
BUD_DRIVE_A1	Expansion socket 1 of control port A
BUD_DRIVE_A2	Expansion socket 2 of control port A
BUD_DRIVE_B1	Expansion socket 1 of control port B
BUD_DRIVE_B2	Expansion socket 2 of control port B
BUD_DRIVE_C1	Expansion socket 1 of control port C
BUD_DRIVE_C2	Expansion socket 2 of control port C
BUD_DRIVE_D1	Expansion socket 1 of control port D
BUD_DRIVE_D2	Expansion socket 2 of control port D

Volume data volume specifies pointer to the area where body color information is stored. Body color information is in area of 32 bytes and is as follows.

When body color information is specified:

+0	+1	+2	+3	+4	+5	...	+31
01H	BLUE	GREEN	RED	ALPHA	00H	...	00H

Store '\x01' in volume [0] without exception. Any other values are not accepted. Blue, green and red components needs to be specified in range of 0 to 255 in each BLUE, GREEN and RED.

ALPHA value needs to be between 128 and 255. Value of 127 or lower is not acceptable (because screen of BootROM gets hard to watch).

Not to specify body color, all components needs to be set to 00H. If there is no body color information, file control screen will be defalt color, white.

The followings can be specified in flag.

Definition	Meainig
TRUE	Full format
FALSE	Quick format

Example

```
Sint32 ret;
BUS_TIME time;
SYS_RTC_DATE rtc;
Uint8 volume[32];
Sint32 icon_no;
syRtcGetDate( &rtc );

time = ( BUS_TIME )rtc; /* Set time stamp */
icon_no = 0; /* Set icon number */

/* Set body color to blue */
memset(volume, 0, sizeof(volume));
volume[0] = 0x01;  /* Body color information found */
volume[1] = 0xbf;  /* B */
volume[2] = 0x00;  /* G */
volume[3] = 0x00;  /* R */
volume[4] = 0xff;  /* A */
ret = buFormatDisk(BD_DRIVE_A1, volume, icon_no, &time, TRUE);
while (buStat(BUD_DRIVE_A1) == BUD_STAT_BUSY) {
}
if (buGetLastError(BUD_DRIVE_A1) != BUD_ERR_OK)
        return NG;
return OK;
```

Note

The following are the completion statuses for this function that are obtained by `buStat()`:

Status	Meaning
BUD_ERR_OK	Normal termination
BUD_ERR_NO_DISK	Memory card not found

buGetDiskFree

Gets free space on the memory card.

Format

```
Sint32 buGetDiskFree( drive, type )
Sint32 drive
Sint32 type
```

Parameters

drive	Expansion socket number
type	File type

Return Value

`0 or positive value`	Number of free blocks
`BUD_ERR_UNFORMAT`	Not formatted
`BUD_ERR_NO_DISK`	Memory card not found
`BUD_ERR_BUSY`	A `TYPE_B` function is being processed

Description

Returns a block of free space on a memory card inserted in an expansion socket.
The following values can be designated for drive that specifies operand expansion sockets.

Definition	Meaning
`BUD_DRIVE_A1`	Expansion socket 1 of control port A
`BUD_DRIVE_A2`	Expansion socket 2 of control port A
`BUD_DRIVE_B1`	Expansion socket 1 of control port B
`BUD_DRIVE_B2`	Expansion socket 2 of control port B
`BUD_DRIVE_C1`	Expansion socket 1 of control port C
`BUD_DRIVE_C2`	Expansion socket 2 of control port C
`BUD_DRIVE_D1`	Expansion socket 1 of control port D
`BUD_DRIVE_D2`	Expansion socket 2 of control port D

Type of file to be looked up free space of is specified in argument type.

Definition	Meaning
`BUD_FILETYPE_NORMAL`	Normal file free space
`BUD_FILETYPE_EXECUTABLE`	Executable file free space

Normally, file free space includes executable file free space.

Example

```
Sint32 free;
free = buGetDiskFree(BUD_DRIVE_A1, BUD_FILETYPE_NORMAL);
if (free < 0)
        return NG;
```

buGetDiskInfo

Gets memory card information.

Format

```
Sint32 buGetDiskInfo( drive, info )
Sint32 drive
BUS_DISKINFO *info
```

Parameters

drive	Expansion socket number
info	Address of memory card information structure

Return Value

BUD_ERR_OK	Normal termination

Description

Gets information from the memory card for a specified expansion socket.

The following values can be designated for drive that specifies operand expansion sockets.

Definition	Meaning
BUD_DRIVE_A1	Expansion socket 1 of control port A
BUD_DRIVE_A2	Expansion socket 2 of control port A
BUD_DRIVE_B1	Expansion socket 1 of control port B
BUD_DRIVE_B2	Expansion socket 2 of control port B
BUD_DRIVE_C1	Expansion socket 1 of control port C
BUD_DRIVE_C2	Expansion socket 2 of control port C
BUD_DRIVE_D1	Expansion socket 1 of control port D
BUD_DRIVE_D2	Expansion socket 2 of control port D

Example

```
BUS_DISKINFO info;
Sint32 ret;
ret = buGetDiskInfo(BUD_DRIVE_A1, &info);
if (ret < 0)
        return NG;
```

buGetFileInfo

Gets file information.

Format

```
Sint32 buGetFileInfo( drive, fname, info )
Sint32 drive
const char *fname
BUS_FILEINFO *info
```

Parameters

drive	Expansion socket number
fname	File name
info	Address of file information structure

Return Value

BUD_ERR_OK	File found
BUD_ERR_FILE_NOT_FOUND	File not found
BUD_ERR_UNFORMAT	Disk is not formatted
BUD_ERR_NO_DISK	Memory card not found
BUD_ERR_BUSY	A TYPE_B function is being processed

Description

Gets information for a file from a memory card inserted in a specified expansion socket.

The following values can be designated for the argument drive that specifies expansion sockets.

Definition	Meaning
BUD_DRIVE_A1	Expansion socket 1 of control port A
BUD_DRIVE_A2	Expansion socket 2 of control port A
BUD_DRIVE_B1	Expansion socket 1 of control port B
BUD_DRIVE_B2	Expansion socket 2 of control port B
BUD_DRIVE_C1	Expansion socket 1 of control port C
BUD_DRIVE_C2	Expansion socket 2 of control port C
BUD_DRIVE_D1	Expansion socket 1 of control port D
BUD_DRIVE_D2	Expansion socket 2 of control port D

Example

```
BUS_FILEINFO info;
buGetFileInfo( BUD_DRIVE_A1, "SAVEDATA_001", &info );
```

buGetFileSize

Gets file block count.

Format

```
Sint32 buGetFileSize( drive, fname )
Sint32 drive
const char *fname
```

Parameters

drive	Expansion socket number
fname	File name

Return Value

BUD_ERR_OK	File found
BUD_ERR_FILE_NOT_FOUND	File not found
BUD_ERR_UNFORMAT	Disk is not formatted
BUD_ERR_NO_DISK	Memory card not found
BUD_ERR_BUSY	A type_B function is being processed

Description

Gets the size of a file with a block count from a memory card inserted in a specified expansion socket. The following values can be designated for the argument drive that specifies expansion sockets.

Definition	Meaning
BUD_DRIVE_A1	Expansion socket 1 of control port A
BUD_DRIVE_A2	Expansion socket 2 of control port A
BUD_DRIVE_B1	Expansion socket 1 of control port B
BUD_DRIVE_B2	Expansion socket 2 of control port B
BUD_DRIVE_C1	Expansion socket 1 of control port C
BUD_DRIVE_C2	Expansion socket 2 of control port C
BUD_DRIVE_D1	Expansion socket 1 of control port D
BUD_DRIVE_D2	Expansion socket 2 of control port D

Example

```
Sint32 size = buGetFileSize(BUD_DRIVE_A1, "SAVEDATA_001");
```

buGetLastError

Returns the last error that was generated.

Format

```
Sint32 buGetLastError( drive )
Sint32 drive
```

Parameters

drive	Expansion socket number

Return Value

BUD_ERR_OK	No error
BUD_ERR_BUSY	Executing command
BUD_ERR_INVALID_PARAM	Invalid function parameter
BUD_ERR_ILLEGAL_DISK	Last block failed
BUD_ERR_UNKNOWN_DISK	Memory card is unknown
BUD_ERR_NO_DISK	Memory card is not found
BUD_ERR_UNFORMAT	Disk is not formatted
BUD_ERR_DISK_FULL	Memory card is full
BUD_ERR_FILE_NOT_FOUND	File not found
BUD_ERR_FILE_EXIST	File already exists
BUD_ERR_CANNOT_OPEN	File cannot be opened
BUD_ERR_CANNOT_CREATE	Cannot create execute file
BUD_ERR_EXECFILE_EXIST	Execute file already exists
BUD_ERR_CANNOT_DELETE	Cannot delete file
BUD_ERR_ACCESS_DENIED	File access denied
BUD_ERR_VERIFY	Verify error
BUD_ERR_WRITE_ERROR	Write error
BUD_ERR_FILE_BROKEN	File is corrupted
BUD_ERR_BUPFILE_CRC	Backup file CRC error
BUD_ERR_BUPFILE_ILLEGAL	Backup file illegal

BUD_ERR_GENERIC Generic error

Description

Finds the last error generated while accessing the expansion socket.

The following values can be designated for the argument drive that specifies expansion sockets.

Definition	Meaning
BUD_DRIVE_A1	Expansion socket 1 of control port A
BUD_DRIVE_A2	Expansion socket 2 of control port A
BUD_DRIVE_B1	Expansion socket 1 of control port B
BUD_DRIVE_B2	Expansion socket 2 of control port B
BUD_DRIVE_C1	Expansion socket 1 of control port C
BUD_DRIVE_C2	Expansion socket 2 of control port C
BUD_DRIVE_D1	Expansion socket 1 of control port D
BUD_DRIVE_D2	Expansion socket 2 of control port D

Example

```
if (buGetLastError(BUD_DRIVE_A1) == BUD_ERR_OK) {
        /* Save successful */
} else {
        /* Error was generated */
}
```

buInit

Initializes the memory card file system.

Format

```
Sint32 buInit( capa, drive, work, func )
Snt32 capa
Sint32 drive
void *work
BU_INITCALLBACK func
```

Parameters

capa	Maximum capacity of media
drive	Expansion socket number
work	Work buffer
func	Initialization complete callback

Return Value

BUD_ERR_OK	Normal termination
BUD_ERR_INVALID_PARAM	Invalid parameter

Description

Initializes the memory card file system.

The following values can be designated for the argument capa which shows the maximum capacity of media.

BUD_CAPACITY_128KB
BUD_CAPACITY_256KB
BUD_CAPACITY_512KB
BUD_CAPACITY_1MB

Designate the following values in the argument drive that shows the operand expansion sockets.

Definition	Meaning
BUD_USE_DRIVE_ALL	All expansion sockets
BUD_USE_DRIVE_A1	Expansion socket 1 of control port A
BUD_USE_DRIVE_A2	Expansion socket 2 of control port A
BUD_USE_DRIVE_B1	Expansion socket 1 of control port B
BUD_USE_DRIVE_B2	Expansion socket 2 of control port B
BUD_USE_DRIVE_C1	Expansion socket 1 of control port C
BUD_USE_DRIVE_C2	Expansion socket 2 of control port C
BUD_USE_DRIVE_D1	Expansion socket 1 of control port D
BUD_USE_DRIVE_D2	Expansion socket 2 of control port D

When NULL is specified in the argument work that marks the work buffer,the setting for the argument capa is ignored and the work buffer for the memory card can be dynamically managed.

When initialization is completed successfully, the specified function is called back. The callback function is called before returning from this function. In this case, after the BUD_OP_CONNECT callback is detected, the application allocates a buffer that is suited for the size of the memory card, and then mount processing is performed by the buMountDisk() function. The backup RAM can then be used in the normal manner afterwards.

This method makes it possible to handle large size memory cards without allocating a large buffer size beforehand.

Example

```
/* Allocating two buffers for memory cards of up to 128K */
Uint32 work[BUM_WORK_SIZE(BUD_CAPACITY_128KB, 2) / sizeof(Uint32)];
void init_callback(void)
{
        :
}
/* Corresponds to expansion sockets 1 and 2 of control port A */
buInit(BUD_CAPACITY_128KB, BUD_USE_DRIVE_A1
        | BUD_USE_DRIVE_A2, work, init_callback);
```

Note

The necessary amount of work space differs greatly depending on the maximum capacity and the count of the memory card being used.

Do not specify an unnecessary memory card in an expansion socket for an application. Not specifying unnecessary memory cards reduces the size of the work space and permits faster library operation. The following table lists guidelines for work space:

Maximum capacity	Workspace per 1 socket	Workspace by 8 slots
128KB	8KB	64KB
256KB	14KB	112KB
512KB	28KB	220KB
1MB	55KB	440KB

Specify files used in expansion sockets as BUD_USE_DRIVE_XXX. Do not specify BUD_DRIVE_XXX constant by mistake.

Reference

buExit()	Exits the memory card file system
buMountDisk()	Mounts a memory card

buIsExistFile

Gets existing of files.

Format

```
Sint32 buIsExistFile( drive, fname )
Sint32 drive
const char *fname
```

Parameters

drive	Expansion socket number
fname	File name

Return Value

BUD_ERR_OK	File found
BUD_ERR_FILE_NOT_FOUND	File not found
BUD_ERR_UNFORMAT	Disk is not formatted
BUD_ERR_NO_DISK	Memory card is not found
BUD_ERR_BUSY	A type_B function is being processed

Description

Finds whether a file exists on a memory card inserted in an expansion socket.

The following values can be designated for the argument drive that specifies expansion sockets.

Definition	Meaning
BUD_DRIVE_A1	Expansion socket 1 of control port A
BUD_DRIVE_A2	Expansion socket 2 of control port A
BUD_DRIVE_B1	Expansion socket 1 of control port B
BUD_DRIVE_B2	Expansion socket 2 of control port B
BUD_DRIVE_C1	Expansion socket 1 of control port C
BUD_DRIVE_C2	Expansion socket 2 of control port C
BUD_DRIVE_D1	Expansion socket 1 of control port D
BUD_DRIVE_D2	Expansion socket 2 of control port D

Example

```
Sint32 ret;
ret = buIsExistFile(BUD_DRIVE_A1, "SAVEDATA_001");
switch (ret) {
      case BUD_ERR_OK:
              /* File found */
              break;
      case BUD_ERR_FILE_NOT_FOUND:
              /* File not found */
              break;
      defalut:
              /* Other error */
              break;
}
```

buIsFormat

Returns whether or not a memory card has been formatted.

Format

```
Sint32 buIsFormat( drive )
Sint32 drive
```

Parameters

drive	Expansion socket number

Return Value

BUD_ERR_OK	Disk is formatted

Description

Returns whether or not a memory card inserted in an expansion socket has been formatted.
The following values can be designated for the argument drive that specifies operand expansion sockets.

Definition	Meaning
BUD_DRIVE_A1	Expansion socket 1 of control port A
BUD_DRIVE_A2	Expansion socket 2 of control port A
BUD_DRIVE_B1	Expansion socket 1 of control port B
BUD_DRIVE_B2	Expansion socket 2 of control port B
BUD_DRIVE_C1	Expansion socket 1 of control port C
BUD_DRIVE_C2	Expansion socket 2 of control port C
BUD_DRIVE_D1	Expansion socket 1 of control port D
BUD_DRIVE_D2	Expansion socket 2 of control port D

Example

```
switch (buIsFormat(BUD_DRIVE_A1)) {
        case BUD_ERR_OK:
                /* Formatted */
                break;
        case BUD_ERR_NO_DISK:
                /* Memory card is not connected */
                break;
        case BUD_ERR_BUSY:
                /* Busy - memory card cound not be checked */
                break;
}
```

buIsReady

Returns the connection status of a memory card.

Format

```
Sint32 buIsReady( drive )
Sint32 drive
```

Parameters

drive	Expansion socket number

Return Value

TRUE	Memory card is connected and ready for access
FALSE	Memory card is not connected

Description

Returns the connection status of a memory card for a specified expansion socket.

The following values can be designated for the argument drive that specifies operand expansion sockets.

Definition	Meaning
BUD_DRIVE_A1	Expansion socket 1 of control port A
BUD_DRIVE_A2	Expansion socket 2 of control port A
BUD_DRIVE_B1	Expansion socket 1 of control port B
BUD_DRIVE_B2	Expansion socket 2 of control port B
BUD_DRIVE_C1	Expansion socket 1 of control port C
BUD_DRIVE_C2	Expansion socket 2 of control port C
BUD_DRIVE_D1	Expansion socket 1 of control port D
BUD_DRIVE_D2	Expansion socket 2 of control port D

Example

```
if (buIsReady(BUD_DRIVE_A1)) {
        /* Memory card is connected */
}
```

buLoadFile

Loads a file.

Format

```
Sint32 buLoadFile( drive, fname, buf, nblock )
Sint32 drive
const char *fname
void *buf
Uint32 nblock
```

Parameters

drive	Expansion socket number
fname	File name
buf	Load start address (4 byte boundary)
nblock	Number of blocks to be loaded

Return Value

buStat()	Processing request was accepted
buStat()	Request could not be accepted because processing was in progress

Description

Loads a file from a memory card inserted in a specified expansion socket.

The following values can be designated for the argument drive that specifies expansion sockets.

Definition	Meaning
BUD_DRIVE_A1	Expansion socket 1 of control port A
BUD_DRIVE_A2	Expansion socket 2 of control port A
BUD_DRIVE_B1	Expansion socket 1 of control port B
BUD_DRIVE_B2	Expansion socket 2 of control port B
BUD_DRIVE_C1	Expansion socket 1 of control port C
BUD_DRIVE_C2	Expansion socket 2 of control port C
BUD_DRIVE_D1	Expansion socket 1 of control port D
BUD_DRIVE_D2	Expansion socket 2 of control port D

Parameter buf is the buffer that is read (however, because of immediate return, it is undefined after the function finishes).

In the case of a normal application, this function loads an entire file. If "0" is specified for the number of blocks to be loaded, the entire file is loaded.

Example

```
Sint32 ret;
Sint32 blocks;

/* Load buffer */
extern char SaveData[];

/* Get file size */
blocks = buGetFileSize(BUD_DRIVE_A1, "SAVEDATA");

if (blocks <= 0)
        return NG;
ret = buLoadFile(BUD_DRIVE_A1, "SAVEDATA", SaveData, blocks);
if (ret == BUD_ERR_OK) {
        /* Load request was successful */
} else {
        /* Load request failed (busy) */
}
```

Note

The following are the completion statuses for this function that are obtained by buStat():

Status	Meaning
BUD_ERR_OK	Normal termination
BUD_ERR_NO_DISK	Memory card not found
BUD_ERR_UNFORMAT	Disk is not formatted
BUD_ERR_FILE_NOT_FOUND	File not found
BUD_ERR_CANNOT_OPEN	Cannot open file
BUD_ERR_FILE_BROKEN	File is damaged

A load generally requires (number of blocks being loaded x 1 Int). (This time requirement may vary, depending on the status of other drives, etc.)

In addition, the address that is specified for `buf` must be on a four-byte boundary.

Reference

`buStat()`	Checks whether processing is completed
`buGetLastError()`	Returns the last error that was generated

buLoadFileEx

Loads a file (specifies initial block).

Format

```
Sint32 buLoadFileEx(drive, fname, buf, start, nblock)
Sint32 drive
const char *fname
void *buf
Uint32 start
Uint32 nblock
```

Parameters

drive	Expansion socket number
fname	File name
buf	Write start address (4 byte boundary)
start	Load start block
nblock	Number of blocks to be loaded

Return Value

BUD_ERR_OK	Processing request was accepted
BUD_ERR_BUSY	Processing request was not accepted because processing was in progess

Description

Loads a file. File can be read partially.

The following values can be designated for the argument drive which specifies expansion sockets.

Definition	Meaning
BUD_DRIVE_A1	Expansion socket 1 of control port A
BUD_DRIVE_A2	Expansion socket 2 of control port A
BUD_DRIVE_B1	Expansion socket 1 of control port B
BUD_DRIVE_B2	Expansion socket 2 of control port B
BUD_DRIVE_C1	Expansion socket 1 of control port C
BUD_DRIVE_C2	Expansion socket 2 of control port C
BUD_DRIVE_D1	Expansion socket 1 of control port D
BUD_DRIVE_D2	Expansion socket 2 of control port D

If "0" is specified for the load start block, the file is loaded from the beginning. If "0" is specified for the number of blocks to be loaded, the entire file is loaded.

Example

```
Sint32 ret;
Sint32 blocks;
Sint32 start_block;

/* Load buffer */
extern char LoadBuffer[];

/* Partial file load */
start_block = 0;
blocks = 1;
ret = buLoadFileEx(BUD_DRIVE_A1, "SAVEDATA_001",
        LoadBuffer, start_block, blocks);
if (ret != BUD_ERR_OK)
        return NG;
while (buStat(BUD_DRIVE_A1) == BUD_STAT_BUSY) {
        :
}
if (buGetLastError(BUD_DRIVE_A1) != BUD_ERR_OK){
        return NG;
} else {
        return OK;
}
```

Note

The following are the completion statuses for this function that are obtained by `buStat()`:

Status	Meaning
BUD_ERR_OK	Normal termination
BUD_ERR_NO_DISK	Memory card not found
BUD_ERR_UNFORMAT	Disk is not formatted
BUD_ERR_FILE_NOT_FOUND	File not found
BUD_ERR_CANNOT_OPEN	Cannot open file
BUD_ERR_FILE_BROKEN	File is damaged

A load generally requires (number of blocks being loaded x 1 Int). (This time requirement may vary, depending on the status of other drives, etc.)

In addition, the address that is specified for buf must be on a four-byte boundary.

buMakeBackupFileImage

Gets file image.

Format

```
Sint32 buMakeBackupFileImage( buf, hdr )
void *buf
const BUS_BACKUPFILEHEADER *hdr
```

Parameters

buf	Address where memory card file format image is to be created (4-byte boundary)
hdr	Memory card file header address

Return Value

Positive value	Number of blocks used in the memory card file
BUD_ERR_INVALID_PARAM	Invalid header

Description

Creates a memory card file format image in memory.

Example

```
/* Game name (sort item) */
extern Uint8 game_name[];
/* Icon palette data */
extern Uint16 icon_palette[];
/* Icon pixel data */
extern Uint8 icon_data[];
/* Visual data */
extern Uint8 visual_data[];
/* Application save data */
extern Uint8 save_data[];
Sint32 nblock1, nblock2;
void* buf;
/* Memory card file header */
BUS_BACKUPFILEHEADER hdr;
memset(&hdr, 0, sizeof(hdr));
/* VM comment setting */
strcpy(hdr.vms_comment, "VMS_COMMENT");
/* BOOT ROM comment setting */
strcpy(hdr.btr_comment, "BOOT ROM full-character comment");
/* Game name (sort item) setting */
memcpy(hdr.game_name,   game_name, 16);
/* Icon palette data setting */
hdr.icon_palette = icon_palette;
/* Icon pixel data setting */
hdr.icon_data    = icon_data;
/* Number of icons setting */
```

```
hdr.icon_num     = 1;
/* Icon speed setting */
hdr.icon_speed   = 1;
/* Visual comment setting */
hdr.visual_data  = visual_data;
/* Visual type setting */
hdr.visual_type  = BUD_VISUALTYPE_C;
/* Application save data */
hdr.save_data    = save_data;
/* Application save data size */
hdr.save_size    = 0x400;
nblock = buCalcBackupFileSize(hdr.icon_num,
        hdr.visual_type, hdr.save_size);
/* Allocates (the number of blocks to be used x 512 bytes) */
buf = syMalloc(nblock * 512);
if (buf == NULL) {
        /* Memory cannot be allocated */
        return NG;
}
nblock = buMakeBackupFileImage(buf, &hdr);
if (nblock < 0) {
        /* Structure includes an invalid parameter */
        return NG;
}
```

buMountDisk

Mounts a memory card.

Format

```
Sint32 buMountDisk( drive, addr, size )
Sint32 drive
void *addr
Sint32 size
```

Parameters

drive	Expansion socket number
addr	Work address (4 byte boundary)
size	Work size

Return Value

BUD_ERR_INVALID_PARAM	Buffer is not sufficient for the specified memory card
BUD_ERR_OK	Normal termination

Description

Mounts a memory card loaded in a specified expansion socket.

The following values can be designated for the argument drive which specifies operand expansion sockets.

Definition	Meaning
BUD_DRIVE_A1	Expansion socket 1 of control port A
BUD_DRIVE_A2	Expansion socket 2 of control port A
BUD_DRIVE_B1	Expansion socket 1 of control port B
BUD_DRIVE_B2	Expansion socket 2 of control port B
BUD_DRIVE_C1	Expansion socket 1 of control port C
BUD_DRIVE_C2	Expansion socket 2 of control port C
BUD_DRIVE_D1	Expansion socket 1 of control port D
BUD_DRIVE_D2	Expansion socket 2 of control port D

The addr that is specified must be on a four-byte boundary. The drive is not mounted if NULL is specified for addr or if the buffer is too small.

Note

If the the work buffer parameter for the function bulnit() is not set to NULL when the memory system is initialized, the memory card cannot be mounted, even with this function.

Reference

buInit()	Initializes the memory card file system

buSaveExecFile

Saves an executable file.

Format

```
Sint32 buSaveExecFile( drive, fname, buf, nblock, time, flag )
Sint32 drive
const char *fname
const void *buf
Uint32 nblock
BUS_TIME *time
Uint32 flag
```

Parameters

drive	Expansion socket number
fname	File name
buf	Read address
nblock	Number of blocks to be read
time	Time stamp
flag	flag

Return Value

BUD_ERR_OK	Processing request was received
BUD_ERR_BUSY	Busy; request could not be accepted

Description

Saves an executable file.

The following values can be designated for the argument drive which specifies expansion sockets.

Definition	Meaning
BUD_DRIVE_A1	Expansion socket 1 of control port A
BUD_DRIVE_A2	Expansion socket 2 of control port A
BUD_DRIVE_B1	Expansion socket 1 of control port B
BUD_DRIVE_B2	Expansion socket 2 of control port B
BUD_DRIVE_C1	Expansion socket 1 of control port C
BUD_DRIVE_C2	Expansion socket 2 of control port C
BUD_DRIVE_D1	Expansion socket 1 of control port D
BUD_DRIVE_D2	Expansion socket 2 of control port D

In a normal application, this function saves an entire file.

If an executable file exists or if a file with the same name already exists, an error results. The followings can be designated for the argument flag.

Definition	Meaning
BUD_FLAG_VERIFY	Verify enabled
BUD_FLAG_COPY(n)	Copy flag ("n" is the value of the copy flag)

Example

```
Sint32 ret;
Sint32 blocks, flag;
BUS_TIME time;
SYS_RTC_DATE rtc;

/* Data to be saved(512*10 bytes) */
extern char SaveData[];

/* File size is 10 blocks */
blocks = 10;

/* Verify enabled | Set copy flag to FFH */
flag = BUD_FLAG_VERIFY | BUD_FLAG_COPY(0xff);
syRtcGetDate( &rtc );

/* Time stamp setting */
time = ( BUS_TIME )rtc;
ret = buSaveExecFile(BUD_DRIVE_A1, "SAVEDATA",
        SaveData, blocks, &time, flag);
if (ret != BUD_ERR_OK)
        return NG;
while (buStat(BUD_DRIVE_A1) == BUD_STAT_BUSY) {
        :
}
if (buGetLastError(BUD_DRIVE_A1) != BUD_ERR_OK){
        return NG;
} else {
        return OK;
}
```

Note

Even if a verify error occurs, the file is still saved.

The following are the completion statuses for this function that are obtained by `buStat()`:

tatus	Meaning
`BUD_ERR_OK`	Normal termination
`BUD_ERR_UNFORMAT`	Disk is not formatted
`BUD_ERR_NO_DISK`	Memory card is not found
`BUD_ERR_DISK_FULL`	Executable file already exists
`BUD_ERR_FILE_EXIST`	File with the same name already exists

A save requires (number of blocks being saved x 5 Int). (This time requirement may vary, depending on the status of other drives, etc.)

In addition, the address that is specified for `buf` must be on a four-byte boundary.

buSaveFile

Saves a file.

Format

```
Sint32 buSaveFile( drive, fname, buf, nblock, time, flag )
Sint32 drive
const char *fname
cnst void *buf
Uint32 nblock
BUS_TIME *time
Uint32 flag
```

Parameters

drive	Expansion socket number
fname	File name
buf	Read address
nblock	Number of blocks to be read
time	Time stamp
flag	Flag

Return Value

BUD_ERR_OK	Processing request was received
BUD_ERR_BUSY	Busy; request could not be accepted

Description

Saves a file on the memory card inserted in a specified expansion socket.

The following values can be designated for drive that specifies expansion sockets.

Definition	Meaning
BUD_DRIVE_A1	Expansion socket 1 of control port A
BUD_DRIVE_A2	Expansion socket 2 of control port A
BUD_DRIVE_B1	Expansion socket 1 of control port B
BUD_DRIVE_B2	Expansion socket 2 of control port B
BUD_DRIVE_C1	Expansion socket 1 of control port C
BUD_DRIVE_C2	Expansion socket 2 of control port C
BUD_DRIVE_D1	Expansion socket 1 of control port D
BUD_DRIVE_D2	Expansion socket 2 of control port D

In a normal application, this function saves an entire file.

If a file with the same name already exists, the old file is deleted. However, if the file with the same name was an executable file, an error results.

In addition, the address that is specified for *buf must be on a four-byte boundary.

Specify file attributes in flag.

Definition	Meaning
BUD_FLAG_VERIFY	Verify enabled
BUD_FLAG_COPY(n)	Copy flag ("n" is the value of the copy flag)

Example

```
Sint32 ret;
Sint32 blocks, flag;
BUS_TIME time;
SYS_RTC_DATE rtc;

/* Data to be saved (512*10 bytes) */
extern char SaveData[];
       :
/* File size is 10 blocks */
blocks = 10;

/* Verify enabled | Set copy flag to 00H */
flag = BUD_FLAG_VERIFY | BUD_FLAG_COPY(0x00);
syRtcGetDate( &rtc );

/* Time stamp setting */
time = ( BUS_TIME )rtc;
ret = buSaveFile(BUD_DRIVE_A1, "SAVEDATA",
       SaveData, blocks, &time, flag);
if (ret == BUD_ERR_OK) {
       /* Save request was successful */
} else {
       /* Save request failed (busy) */
}
       :
```

Note

Even if a verify error occurs, the file is still saved.

The following are the completion statuses for this function that are obtained by `buStat()`:

Status	Meaning
BUD_ERR_OK	Normal termination
BUD_ERR_NO_DISK	Memory card not found
BUD_ERR_UNFORMAT	Memory card not formatted
BUD_ERR_DISK_FULL	Memory card full
BUD_ERR_EXECFILE_EXIST	Executable file with same name exists
BUD_ERR_CANNOT_CREATE	File cannot be created
BUD_ERR_VERIFY	Verify error

A save requires (number of blocks being saved x 5 Int). (Depending on the status of other expansion sockets, the time required to save a file varies.)

buSetCompleteCallback

Specifies the callback function.

Format

```
void buSetCompleteCallback( func )
BU_COMPLETECALLBACK func
```

Parameters

func	Complete callback function address

Return Value

None

Description

Specifies the callback function for when processing is complete.

Example

```
Sint32 complete_callback(Sint32 drive, Sint32 op, Sint32 status, Uint32 param)
{
       return BUD_CBRET_OK;
}
Sint32 progress_callback(Sint32 drive, Sint32 op, Sint32 count, Uint32 max)
{
       return BUD_CBRET_OK;

void init_callback(void)
{
       buSetCompleteCallback(complete_callback);
       buSetProgressCallback(progress_callback);
}
```

buSetFileAttr

Changes file attributes.

Format

```
Sint32 buSetFileAttr( drive, fname, header, copyflag )
Sint32 drive
const char *fname
Uint16 header
Uint8 copyflag
```

Parameters

drive	Expansion socket number
fname	File name
header	Header offset (ignored)
copyflag	Copy flag (00H to FFH)

Return Value

BUD_ERR_OK	Processing request was accepted
BUD_ERR_BUSY	Request could not be accepted because processing was in progress

Description

Changes the attributes of a specified file in a memory card inserted in a specified expansion socket. The following values can be designated for drive that specifies expansion sockets.

Definition	Meaning
BUD_DRIVE_A1	Expansion socket 1 of control port A
BUD_DRIVE_A2	Expansion socket 2 of control port A
BUD_DRIVE_B1	Expansion socket 1 of control port B
BUD_DRIVE_B2	Expansion socket 2 of control port B
BUD_DRIVE_C1	Expansion socket 1 of control port C
BUD_DRIVE_C2	Expansion socket 2 of control port C
BUD_DRIVE_D1	Expansion socket 1 of control port D
BUD_DRIVE_D2	Expansion socket 2 of control port D

Example

```
/* Set "copying prohibited" flag (set copy flag to "FFH") */
buSetFileAttr(BUD_DRIVE_A1, "SAVEFILE_001", 0, 0xff);
if (ret != BUD_ERR_OK) return NG;
while (1) {
       if (buStat(BUD_DRIVE_A1) == BUD_STAT_READY) break;
}
if (buGetLastError(BUD_DRIVE_A1) != BUD_ERR_OK)
       return NG;
return OK;
```

Note

Use this function to change the copy flag for a file that already exists. The header offset is ignored. Specify "0".

buSetProgressCallback

Specifies the progress callback function.

Format

```
void buSetProgressCallback( func )
BU_PROGRESSCALLBACK func
```

Parameters

`func`	Progress callback function address

Return Value

None

Description

Specifies the progress callback function for that is called while processing is in progress.

Note

Execute this function during the callback function after initialization is complete. The progress callback function is executed in the following situations:

- When an immediate return-type function was called:

The progress callback function is called each time the writing of one block is completed. Because reading one block in a memory card requires approximately 1 Int and a write requires approximately 5 Int, that is how often the callback function is called.

- When a memory card has been inserted:

If a memory card is inserted, the system area, FAT, directory entry and other management areas are read. There are a total of 15 blocks of management area that need to be read; reading all of them requires 16 Int.

buStat

Checks whether processing is completed.

Format

```
Sint32 buStat( drive )
Sint32 drive
```

Parameters

drive	Expansion socket number

Return Value

BUD_STAT_READY	Processing has been completed
BUD_STAT_BUSY	A type_B function is being processed

Description

Checks whether processing has been completed on a memory card inserted in a specified expansion socket. The following values can be designated for the argument drive that specifies expansion sockets.

Definition	Meaning
BUD_DRIVE_A1	Expansion socket 1 of control port A
BUD_DRIVE_A2	Expansion socket 2 of control port A
BUD_DRIVE_B1	Expansion socket 1 of control port B
BUD_DRIVE_B2	Expansion socket 2 of control port B
BUD_DRIVE_C1	Expansion socket 1 of control port C
BUD_DRIVE_C2	Expansion socket 2 of control port C
BUD_DRIVE_D1	Expansion socket 1 of control port D
BUD_DRIVE_D2	Expansion socket 2 of control port D

Example

```
if (buStat(BUD_DRIVE_A1) == BUD_STAT_BUSY) {
       :
/* Processing still in progress */
       :
}
```

buUnmount

Unmounts a memory card.

Format

```
Sint32 buUnmount( drive )
Sint32 drive
```

Parameters

drive	Expansion socket number

Return Value

BUD_ERR_OK	Normal termination

Description

Unmounts a memory card inserted in a specified expansion socket.

The following values can be designated for drive that specifies operand expansion sockets.

Definition	Meaning
BUD_DRIVE_A1	Expansion socket 1 of control port A
BUD_DRIVE_A2	Expansion socket 2 of control port A
BUD_DRIVE_B1	Expansion socket 1 of control port B
BUD_DRIVE_B2	Expansion socket 2 of control port B
BUD_DRIVE_C1	Expansion socket 1 of control port C
BUD_DRIVE_C2	Expansion socket 2 of control port C
BUD_DRIVE_D1	Expansion socket 1 of control port D
BUD_DRIVE_D2	Expansion socket 2 of control port D

Note

When this function is called, BUD_OP_UNMOUNT callback is generated.

The BUD_OP_UNMOUNT callback is also generated if the memory card is removed.

When the BUD_OP_UNMOUNT callback is generated, the work buffer that was being used by that drive is released.

10. Get Video Cable Type Function

syCblCheckCable

Gets the video cable type.

Format

```
SYE_CBL syCblCheck( void )
```

Parameters

None

Return Value

Type of video output

Description

Checks type of video output from Dreamcast unit and connected video cable.
The return values have the following meaning:

Value	Meaning
SYE_CBL_NTSC	NTSC
SYE_CBL_VGA	VGA
SYE_CBL_PAL	PAL

Example

```
cable = syCblCheck();
switch( cable ) {
          case SYE_CBL_VGA :      /* We are running in VGA */
          njInitSystem(NJD_RESOLUTION_VGA, NJD_FRAMEBUFFER_MODE_RGB565, 1);
          break;
          default:
       case SYE_CBL_NTSC :
          njInitSystem(NJD_RESOLUTION_640x480_NTSCNI, NJD_FRAMEBUFFER_MODE_RGB565, 1);
          break;
               case SYE_CBL_PAL :
          njInitSystem(NJD_RESOLUTION_640x544_PALNI, NJD_FRAMEBUFFER_MODE_RGB565, 1);
          break;
}
```

Remarks

`syCblCheckCable()` function does not need to be used on creation of the regular application. Please use this function instead.

Reference

`syCblCheckCable()`	Checks type of video cable

syCblCheckBroadcast

Gets broadcast system.

Format

```
SYE_CBL_BROADCAST syCblCheckBroadcast( void )
```

Parameters

None

Return Value

SYE_CBL_BROADCAST_NTSC	NTSC(Japan, US)
SYE_CBL_BROADCAST_PAL	PAL(Europe)
SYE_CBL_BORADCAST_PALM	PAL-M(Brazil)
SYE_CBL_BROADCAST_PALN	PAL-N(Argentina)

DESCRIPTION

Gets the broadcast system.

NOTE

The broadcast system is set when Dreamcast hardware is factory-issued.

syCblCheckCable

Checks type of video cable.

Format

```
SYE_CBL_CABLE syCblCheckCable( void )
```

Parameters

None

Return Value

Cable type

Description

Checks the type of cable connected to the AV terminal.

Return Value	Meaning
SYE_CBL_CABLE_VGA	VGA cable
SYE_CBL_CABLE_VBS	VBS/S, RF output
SYE_CBL_CABLE_RGB	RGB cable (Multi21P)
SYE_CBL_CABLE_RESERVED	SEGA RESERVED

Note

It is possible, when starting up an application, to first determine which screen mode will be in use, and then initialize the system.

In Dev.Box, this function returns the DIP switch contents that set cable emulation. This function does not need to be used on creation of the regular application. Please use syCblCheck() function instead.

Reference

`syCblCheck()`	Checks type of video output (simple check)

11. Boot ROM Font Functions

syBtFntChkSmph

Checks the font semaphore.

Format

```
Sint32 syBtFntChkSmph( void )
```

Parameters

None

Return Value

`SYD_BT_FNT_SMPH_SUCCESS`	Access to BootROM possible
`SYD_BT_FNT_SMPH_FAILURE`	Access to BootROM impossible

Description

Checks the font semaphore and, if the semaphore is not already set, sets the semaphore and returns the "success" result.

If the font semaphore has already been set by another process (GD driver), this function returns the "failure" result.

Note

If font data is gotten without acquiring the semaphore, an undefined value may be returned.

syBtFntClrSmph

Clears the font semaphore.

Format

```
oid syBtFntClrSmph( void )
```

Parameters

None

Return Value

None

Description

This function clears the font semaphore.

syBtFntGet

Gets the font address.

Format

```
void *syBtFntGet( num )
Uint32 num
```

Parameters

num	Control number

Return Value

Starting address for the font data in the boot ROM

Description

Gets the address where font data is stored in the boot ROM.

Starting address for font data and the necessary fonts with the following constants must be requested with this function.

- Offset to the section starting data

Definition	Meaning
SYD_BT_FNT_ASCII_24_OFS	ASCII 8-bit code first half
SYD_BT_FNT_WEST_24_OFS	ISO 8859-1 8-bit code second half for overseas
SYD_BT_FNT_JISX_24_OFS	JIS X 0201 8-bit code second half for Japan
SYD_BT_FNT_KANA_24_OFS	Shift JIS part1 Full character Japanese kana
SYD_BT_FNT_LVL1_24_OFS	Shift JIS part2 Japanese kanji first standard part
SYD_BT_FNT_LVL2_24_OFS	Shift JIS part3 Japanese kanji second standard part
SYD_BT_FNT_GAIJ_24_OFS	GAIJI Special characters
SYD_BT_FNT_VMSICON_OFS	VM default label icons

• Section data size

Definition	Meaning
SYD_BT_FNT_ASCII_24_SIZE	ASCII 8-bit code first half
SYD_BT_FNT_WEST_24_SIZE	ISO 8859-1 8-bit code second half for overseas
SYD_BT_FNT_JISX_24_SIZE	JIS X 0201 8-bit code second half for Japan
SYD_BT_FNT_KANA_24_SIZE	Shift JIS part1 Full character Japanese kana
SYD_BT_FNT_LVL1_24_SIZE	Shift JIS part2 Japanese kanji first standard part
SYD_BT_FNT_LVL2_24_SIZE	Shift JIS part3 Japanese kanji second standard part
SYD_BT_FNT_GAIJ_24_SIZE	GAIJISpecial characters
SYD_BT_FNT_VMSICON_SIZE	VM default label icons

• Representative character code for each section (2 byte Japanese is Shift JIS code):

Definition	Meaning
SYD_BT_FNT_CODE_ASCII_MIN	First ASCII code
SYD_BT_FNT_CODE_ASCII_MAX	Last ASCII code
SYD_BT_FNT_CODE_8BIT_MIN	First 8-bit code
SYD_BT_FNT_CODE_8BIT_MAX	Last 8-bit code
SYD_BT_FNT_CODE_KANA_MIN	First 2 byte code
SYD_BT_FNT_CODE_KANA_MAX	Last Japanese kana code
SYD_BT_FNT_CODE_LVL1_MIN	First kanji first standard code
SYD_BT_FNT_CODE_LVL1_MAX	Last kanji first standard code
SYD_BT_FNT_CODE_LVL2_MIN	First kanji second standard code
SYD_BT_FNT_CODE_LVL2_MAX	Last kanji second standard code
SYD_BT_FNT_GAIJ_MIN	First special character code
SYD_BT_FNT_GAIJ_MAX	Last special character code
SYD_BT_FNT_ICON_MIN	First visual memory icon code
SYD_BT_FNT_ICON_MAX	Last visual memory icon code

Example

```
#define SMD_WRK_ADDR  (0x0c020000)
main()
{
       void *smFntAddr, *smTheFntAddr;
       void *smWrkAddr;
       Uint8 smTheCode;

       smFntAddr = syBtFntGet(SYD_BT_FNTINI);
       smWrkAddr = (void *)SMD_WRK_ADDR;
       smTheCode = '?';

       /* evaluate the font address in Boot ROM */
       smTheFntAddr = (void *)((Uint8)smFntAddr
          + SYD_BT_FNT_WORK_ASCII_24_OFS
          + (smTheCode - SYD_BT_FNT_CODE_ASCII_MIN)
          * SYD_BT_FNT_24_SIZE_HAN );

       /* font semaphore check */
       if( syBtFntChkSmph() == SYD_BT_FNT_SMPH_SUCCESS ) {
          /* copy 1 letter to work RAM from Boot ROM */
          memcpy( smTheFntAddr, smWrkAddr, SYD_BT_FNT_24_SIZE_HAN );
          /* font semaphore clear */
          syBtFntClrSmph();
       }
}
```

Note

The current value of SYD_BT_FNTINI must be specified for the control number num.

If font data is gotten without acquiring the semaphore, an undefined value may be returned.

The purpose of this function is to get the address of the font data, not to load the font data into the application. The application must be designed to copy the font data on its own.

syBtFntGetAddr

Gets font address.

Format

```
Uint32 syBtFntGetAddr( set, code )
SYE_BT_FNT_FONTSET set
Sint16 code
```

Parameters

`set`	Font set
`code`	Shift JIS code. In the case of 8-byte codes, add 0x00 to the upper 8 bytes.

Return Value

Offset (in number of bytes) from the font data starting address

Description

Calculates the offset from the font data starting address from the Shift JIS code in number of bytes.
The argument set that specifies the font set can have the following values.

Definition	Meaning
SYE_BT_FNT_FONTSET_WESTERN_24	ISO 8859-1
SYE_BT_FNT_FONTSET_JP_JIS_24	JIS X 0201
SYE_BT_FNT_FONTSET_JP_KANA_24	Shift JIS hiragana
SYE_BT_FNT_FONTSET_JP_LVL1_24	Shift JIS first standard
SYE_BT_FNT_FONTSET_JP_LVL2_24	Shift JIS second standard
SYE_BT_FNT_FONTSET_JP_GAIJ_24	SEGA special characters
SYE_BT_FNT_FONTSET_VMSICON	Visual memory default label icons

8-bit codes can be specified by adding 0x00 to the upper 8-bit for the argument code (16-bit).

syBtFntGetInfo

Gets size information for target font data from Shift JIS codes.

Format

```
void syBtFntGetInfo( set, code, info )
SYE_BT_FNT_FONTSET set
Sint16 code
SYS_BT_FNT_INFO *info
```

Parameters

`set`	Font set
`code`	Shift JIS code. In the case of 8-byte codes, add 0x00 to the upper 8 bytes.
`info`	Pointer that stores font data size information

Return Value

None

Description

Gets the number of pixels in the vertical and horizontal directions of the font data.
The argument set that specifies the font set can have the following values.

Definition	Meaning
SYE_BT_FNT_FONTSET_WESTERN_24	ISO 8859-1
SYE_BT_FNT_FONTSET_JP_JIS_24	JIS X 0201
SYE_BT_FNT_FONTSET_JP_KANA_24	Shift JIS hiragana
SYE_BT_FNT_FONTSET_JP_LVL1_24	Shift JIS first standard
SYE_BT_FNT_FONTSET_JP_LVL2_24	Shift JIS second standard
SYE_BT_FNT_FONTSET_JP_GAIJ_24	SEGA secial characters
SYE_BT_FNT_FONTSET_VMSICON	Visual memory default label icons

8-bit codes can be specified by adding 0x00 to the upper 8-bit for the argument code (16-bit).

syBtFntSjis2Jis

Converts Shift JIS codes to JIS codes.

Format

```
Sint16 syBtFntSjis2Jis( sjis )
Sint16 sjis
```

Parameters

`sjis`	Shift JIS code

Return Value

JIS code

Description

This function converts a Shift JIS code into a JIS code.

12. Configuration Functions

syCfgExit

Exits the configuration functions.

Format

```
Sint32 syCfgExit( Void )
```

Parameters

None

Return Value

Error code

Description

Performs the configuration function exit processing.

The following values are returned as error codes.

Definition	Meaning
SYD_CFG_OK	Normal termination
SYD_CFG_NG	Abnormal termination (not currently used)

Note

`syCfgExit()` function needs to be executed after `syCfgInit()` function to run `syCfgInit()` function again.

syCfgGetIndividualID

Gets individual ID.

Format

```
Sint32 syCfgGetIndividualID( pIID )
Sint8 *pIID
```

Parameters

pIID	Pointer to area where individual ID is stored

Return Value

SYD_CFG_IID_OK	Individual ID acquisition OK
SYD_CFG_IID_CHKERR	Check sum error
SYD_CFG_IID_NG	Individual ID acquisition NG

Description

Gets the "Individual ID" set for the Dreamcast main unit.

Each Dreamcast unit has an inherent individual ID. The area size to store the individual ID is defined by SYD_CFG_IID_SIZE (bytes).

The gotten individual ID is stored even if a check sum error is displayed in the return value.

Note

You can overwrite the individual ID on Dev.Box, the tool to develop Dreamcast.

If an individual ID is not recorded when the unit is delivered, the stored individual ID is filled with 0xff and a check sum error is returned.

It is not necessary to call the initialization and exit functions syCfgInit/syCfgExit for this function.

syCfgGetLanguage

Gets language setting.

Format

```
Sint32 syCfgGetLanguage( nMode )
Sint32 *nMode
```

Parameters

nMode	Pointer to where data is stored

Return Value

SYD_CFG_OK	Normal termination
SYD_CFG_NG	Abnormal termination

Description

Gets language setting which is set from "Main Menu" at start up of Dreamcast when disc is not set. The following values are set in parameter nMode which gets language setting.

Definition	Meaning
SYD_CFG_JAPANESE	Japanese
SYD_CFG_ENGLISH	English
SYD_CFG_GERMAN	German
SYD_CFG_FRENCH	French
SYD_CFG_SPANISH	Spanish
SYD_CFG_ITALIAN	Italian

When "abnormal termination" is returned, value of nMode is SYD_CFG_JAPANESE.

syCfgGetSoundMode

Gets the sound setting (stereo/monaural).

Format

```
Sint32 syCfgGetSoundMode( pnMode )
Sint32 *pnMode
```

Parameters

pnMode	Return status pointer

Return Value

Error code

Description

Gets the sound setting (stereo/monaural).
Setting information is returned in address indicated by pnMode.

Definition	Meaning
SYD_CFG_STEREO	Stereo setting
SYD_CFG_MONO	Monaural setting

The following values are returned as error codes.

Definition	Meaning
SYD_CFG_OK	Normal termination
SYD_CFG_NG	Abnormal termination

Reference

syCfgInit()	Initializes the configuration functions

syCfgInit

Initializes the configuration functions.

Description

```
Sint32 syCfgInit( pBuf )
void *pBuf
```

Parameters

pBuf	Buffer pointer

Return Value

Error code

Description

Initializes the configuration functions.

The parameter specifies the buffer address (16K byte) used in the library.

The following values are returned as error code.

Definition	Meaning
SYD_CFG_OK	Normal termination
SYD_CFG_NG	Abnormal termination

Note

"Abnormal termination" results if an invalid parameter (NULL) was specified.

Reference

syCfgExit()	Performs the configuration function exit processing

syCfgSetSoundMode Updates the sound setting (stereo/monaural).

Format

```
Sint32 syCfgSetSoundMode( nMode )
Sint32 nMode
```

Parameters

nMode	Setting value

Return Value

Error code

Description

Updates the sound setting information (stereo/monaural).

The followings can be set in setting information nMode.

Definition	Meaning
SYD_CFG_STEREO	Stereo setting
SYD_CFG_MONO	Monaural setting

The following values are returned as error codes.

Definition	Meaning
SYD_CFG_OK	Normal termination
SYD_CFG_NG	Abnormal termination

Note

Settings for this function are saved in the hardware's internal flash memory, but when settings are saved, only the setting information is updated. Changing the sound status in the actual application must be carried out in a different way. “Abnormal termination” results if an invalid parameter was specified, or if the sound setting has not been initialized.

Reference

syCfgInit()	Initializes the configuration functions

13. Boot ROM Service Functions

syBtCheckDisc

Checks for disk replacement.

Format

```
Sint32 syBtCheckDisc( Void )
```

Parameters

None

Return value

`0:`	Dreamcast disk
`Positive value:`	Other than Dreamcast disk
`Negative value:`	Checking

Decription

Checks the type of disc inserted in the disc tray.

Remarks

This function operates only in boot ROM version 0.972 and later.

In boot ROM versions earlier than version 0.972, this function always returns a `0`, and normal internal operation will be disrupted.

This function should not be called more than once per 1Vsync.

syBtExit

Displays the main menu of the BootROM.

Format

```
Void syBtExit( Void )
```

Parameters

None

Return Value

None

Function

Forcibly displays the main menu screen of the BootROM.

Note

Be sure to call this function after `sbExitSystem()`.

syBtGetBootSystemID

Gets the system ID information on the startup disk.

Format

```
Sint32 syBtGetBootSystemID( pData )
SYS_BT_SYSTEMID *pData
```

Parameters

`pData`	Pointer to the structure where the system ID information is stored

Return Value

Error code

Description

Gets the system ID information on the startup disc.

Confirm `SYS_BT_SYSTEMID` structure entity in the application for the parameter `pData`, and transfer the pointer.

The followings are returned as error codes.

Definition	Meaning
`SYD_BT_OK`	Normal termination
`SYD_BT_NG`	Abnormal termination

The followings can be the case if abnormal termination is returned…

1) A non-numeric character was found in a location in the media information where a digit from "0" to "9" was expected.
2) The `NULL` value was passed for the address.

syBtGetCurrentSystemID

Gets the system ID information on the disk.

Format

```
Sint32 syBtGetCurrentSystemID( pData )
SYS_BT_SYSTEMID *pData
```

Parameters

pData	Address of the structure where the system ID information is stored

Return Value

Error code

Description

Gets the system ID information on the disc which is being recognized currently.

Confirm SYS_BT_SYSTEMID structure entity in the application for the argument pData, and transfer the pointer.

The followings are returned as error codes.

Definition	Meaning
SYD_BT_OK	Normal termination
SYD_BT_NG	Abnormal termination

The followings can be the case if abnormal termination is returned.

1) A non-numeric character was found in a location in the media information where a digit from "0" to "9" was expected.
2) The NULL value was passed for the address.

Example

```
SYS_BT_SYSTEMID     systemidBoot ;
SYS_BT_SYSTEMID     systemidCurrent ;
int                 nRet1,nRet2;
int                 y ;
nRet1 = syBtGetBootSystemID( &systemidBoot ) ;
nRet2 = syBtGetCurrentSystemID( &systemidCurrent ) ;
y = 2 ;
njPrintC( NJM_LOCATION( 4,y++),"BOOT DISC INFO" ) ;
njPrintC( NJM_LOCATION( 4,y), "SYSTEM ID ADDRESS") ;
njPrintH( NJM_LOCATION(25,y++),(Uint32)pAddr1,8) ;
y++ ;
njPrintC( NJM_LOCATION( 4,y),"ID GET RETURN CODE") ;
njPrintH( NJM_LOCATION(25,y++),(Uint32)nRet1, 8) ;
njPrintC( NJM_LOCATION( 4,y),"NO." ) ;
njPrintD( NJM_LOCATION(10,y++), systemidBoot.nNo ,8 ) ;
njPrintC( NJM_LOCATION( 4,y),"All." ) ;
njPrintD( NJM_LOCATION(10,y++), systemidBoot.nAll,8 ) ;
njPrintC( NJM_LOCATION( 4,y),"PID." ) ;
njPrintC( NJM_LOCATION(10,y++), systemidBoot.szProduct ) ;
y = 10 ;
njPrintC( NJM_LOCATION( 4,y++),"NEW DISC INFO" ) ;
njPrintC( NJM_LOCATION( 4,y),"SYSTEM ID ADDRESS") ;
njPrintH( NJM_LOCATION(25,y++),(Uint32)pAddr2,8) ;
y++ ;
njPrintC( NJM_LOCATION( 4,y),"ID GET RETURN CODE") ;
njPrintH( NJM_LOCATION(25,y++),(Uint32)nRet2, 8) ;
njPrintC( NJM_LOCATION( 4,y),"NO." ) ;
njPrintD( NJM_LOCATION(10,y++), systemidCurrent.nNo ,8 ) ;
njPrintC( NJM_LOCATION( 4,y),"All." ) ;
njPrintD( NJM_LOCATION(10,y++), systemidCurrent.nAll,8 ) ;
njPrintC( NJM_LOCATION( 4,y),"PID." ) ;
njPrintC( NJM_LOCATION(10,y++), systemidCurrent.szProduct ) ;
```

Note

This function performs only when it is decided as the Dreamcast disk by the disk check function, `syBtCheckDisc()`, and the disk door doesn't open. If you convert the disc and occur the failure of disk check recognition, we cannot guarantee this activities.

Reference

`syBtCheckDisc()`	Distinguishes disc types

14. Gun Device Functions

pdGunEnter

Sets port to gun mode.

Format

```
void pdGunEnter( portbit )
Uint32 portbit
```

Parameters

`portbit`	Port to be set to gun mode

Return Value

None

Description

Sets controller library to gun mode.

Example

```
pdGunEnter( PDD_GUNMODE_A | PDD_GUNMODE_B );
```

Reference

`pdGunLeave()`	Ends gun mode setting of the port

pdGunGetLatchedPort

Gets port which has detected gun coordinate.

Format

```
Sint32 pdGunGetLatchedPort( void )
```

Parameters

None

Return Value

`Other than -1`	Control port number which has detected gun coordinate
`-1`	Gun coordinate not detected

Description

Returns the control port number which has detected gun coordinates.

Detects of coordinates are performed when trigger (A button) is pulled or when pseudo trigger is pulled by **`pdGunSetTrigger`**() function.

Control port numbers returned are as follows.

Definition	Meaning
PDD_PORT_A0	Port A
PDD_PORT_B0	Port B
PDD_PORT_C0	Port C
PDD_PORT_D0	Port D

Example

```
Sint32 gunno, x, y;
gunno = pdGunGetLatchedPort(void);
if (gunno < 0)
       return 0;
pdGunGetPosition(&x, &y);
```

pdGunGetPosition

Gets gun coordinates.

Format

```
void pdGunGetPosition( x, y )
Sint32 *x
Sint32 *y
```

Parameters

x	Pointer to variable to get gun coordinates (x)
y	Pointer to variable to get gun coordinates (y)

Return Value

None

Description

Gets gun coordinates.

Detects of coordinates are performed when trigger (A button) is pulled or when pseudo trigger is pulled by `pdGunSetTrigger()` function.

Example

```
Please refer to example of pdGunGetLatchedPort() function.
```

Reference

`pdGunGetLatchedPort()`	Gets port which has detected gun coordinate

pdGunLeave

Ends gun mode setting of the port.

Format

```
void pdGunLeave( void )
```

Parameters

None

Return Value

None

Description

Sets controller library to normal mode.

Example

```
pdGunLeave();
```

Reference

`pdGunEnter()`	Sets port to gun mode

pdGunSetCallback

Registers callback function to set trigger.

Format

```
void pdGunSetCallback( func )
void (*func)(void)
```

Parameters

func	Address of callback function

Return Value

None

Description

Registers callback function to set trigger.

When FALSE is returned by callback function, detects of coordinates are not performed even trigger (A button) is pulled. Set trigger by pdGunSetTrigger() function as necessity requires.

When TRUE is returned by callback funciton, coordinates are detected normally by trigger.

Example

```
static Sint32 callback(void)
{
       static Uint32 count = 0;
       if ((count & 3) == 0) {
          pdGunSetTrigger(PDD_PORT_A0);
       }
       count++;
       return TRUE;
}
```

Reference

pdGunSetTrigger()	Creates pseudo trigger

pdGunSetFlashColor

Specifies color of CRT flash.

Format

```
void pdGunSetFlashColor( color )
Uint32 color
```

Parameters

color	Color of CRT flash

Return Value

None

Description

Sets CRT flash color of when trigger (A button) is pulled.

When 0 is specified, CRT flash is not performed. If it is designed to have game display relatively bright, detect of coordinate is performed up to certain extent even without being flashed.

There are cases disable to detect coordinates at all or accurately in case of display being dark (brightness is low) when coordinates are detected.

Initial value is white (0x00c0c0c0). The way of setting colors in this function is as same as setting the border colors in graphic library.

Example

```
pdGunSetFlashColor(0x00c0c0c0);
```

pdGunSetTrigger

Creates pseudo trigger.

Format

```
void pdGunSetTrigger( port )
Uint32 port
```

Parameters

`port`	Control port number

Return Value

None

Description

Pulls pseudo trigger and detects coordinates.

This function needs to be executed within callback function to set trigger, or otherwise it is invalid. Make sure to register callback function by `pdGunSetCallback()` function and call

`pdGunSetTrigger()` function within.

This function enables machine gun shooting on software.

The followings are control port numbers which can be specified in parameter port.

Definition	Meaning
PDD_PORT_A0	Port A
PDD_PORT_B0	Port B
PDD_PORT_C0	Port C
PDD_PORT_D0	Port D

Example

Please refer to example of `pdGunSetCallback()` function.

Reference

`pdGunSetCallback()`	Registers callback function to set trigger

15. Wave Sampling Functions

wsBufCreate

Creates WSBUF handle.

Format

```
WSBUF wsBufCreate( pno, para )
Sint32 pno
WSS_BUF_PRM para
```

Parameters

pno	Port number (0 to 23)
para	Parameter for WSBUF handle

Return Value

WSBUF handle

Description

Creates WSBUF handle.

WSBUF handle is a handle to control the capturing of sound data. It holds the information for each Sound Input device.

Parameter pno indicating port number takes the following values.

Definition	Meaning
WSD_DEV_A1	Expansion socket 1 of control port A
WSD_DEV_A2	Expansion socket 2 of control port A
WSD_DEV_A3	Expansion socket 3 of control port A
WSD_DEV_A4	Expansion socket 4 of control port A
WSD_DEV_A5	Expansion socket 5 of control port A
WSD_DEV_B1	Expansion socket 1 of control port B
WSD_DEV_B2	Expansion socket 2 of control port B
WSD_DEV_B3	Expansion socket 3 of control port B
WSD_DEV_B4	Expansion socket 4 of control port B
WSD_DEV_B5	Expansion socket 5 of control port B
WSD_DEV_C1	Expansion socket 1 of control port C
WSD_DEV_C2	Expansion socket 2 of control port C
WSD_DEV_C3	Expansion socket 3 of control port C
WSD_DEV_C4	Expansion socket 4 of control port C
WSD_DEV_C5	Expansion socket 5 of control port C
WSD_DEV_D1	Expansion socket 1 of control port D
WSD_DEV_D2	Expansion socket 2 of control port D
WSD_DEV_D3	Expansion socket 3 of control port D
WSD_DEV_D4	Expansion socket 4 of control port D
WSD_DEV_D5	Expansion socket 5 of control port D

wsBufDestroy

Destroys WSBUF handle.

Format

```
void wsBufDestroy( wsbuf )
WSBUF wsbuf
```

Parameters

`wsbuf`	WSBUF handle

Return Value

None

Description

Destroys WSBUF handle.

wsBufExecServer

Server function.

Format

```
void wsBufExecServer( void )
```

Parameters

None

Return Value

None

Description

Writes sampling data into ring buffer through Sound Input device and updates internal status of the library.

Note

This function is registered in interrupt handler, "DMA termination interrupt (Maple DMA)", by `wsInit()`. Library termination function, `wsFinish()`, cancels registration made by this function. Users do not need to execute this function.

wsBufGetAmpGain

Gets AMP gain.

Format

```
Sint32 wsBufGetAmpGain( wsbuf )
WSBUF wsbuf
```

Parameters

wsbuf	WSBUF handle

Return Value

AMP gain

Description

Gets AMP gain information from Sound Input device of specified WSBUF handle.

Maximum volume of AMP gain is +16 and minimum volume is -15. (Unit: dB)

Note

With sample Sound Input device, maximum volume of AMP gain is +8 and minimum is -7.

Maximum and minimum volume is defined by `WSD_BUF_GAIN_MAX` and `WSD_BUF_GAIN_MIN` respectively.

wsBufGetBitPerSmpl

Gets bits per sample information.

Format

```
Sint32 wsBufGetBitPerSmpl( wsbuf )
WSBUF wsbuf
```

Parameters

wsbuf	WSBUF handle

Return Value

Bits per sample information

Description

Gets bps (number of bits per sample) information of Sound Input device from specified WSBUF handle.

The followings are defined as return value.

Value	Meaning
WSD_BUF_BPS_16BIT	16 bit linear
WSD_BUF_BPS_8BIT	8bit mu-law codec

When 16 bit linear, sound is sampled as 16 bit and the effective bit number is upper 14 bit.

When 8 bit mu-law codec, data that has been coded is stored in the ring buffer. Coding is necessary for deciding sound data.

wsBufGetErr

Gets an error code of WSB module.

Format

```
Sint32 wsBufGetErr( wsbuf )
WSBUF wsbuf
```

Parameters

wsbuf	WSBUF handle

Return Value

WSD_ERR_OK	Normal termination
WSD_ERR_FATAL	Error

Description

Gets an error code of WSB (Wave Sampling Buffer) module.

wsBufGetNumSmpl

Gets number of samples in ring buffer.

Format

```
Sint32 wsBufGetNumSmpl( wsbuf )
WSBUF wsbuf
```

Parameters

wsbuf	WSBUF handle

Return Value

Number of samples in ring buffer

Description

Gets number of samples (amount of data) stored in ring buffer.

Note

When the ring buffer is filled up with sampling data, number of samples returned will be the same.

wsBufGetSBFOV

Gets overflow bit of Sound Input device buffer.

Format

```
Sint32 wsBufGetSBFOV( wsbuf )
WSBUF wsbuf
```

Parameters

wsbuf	WSBUF handle

Return Value

TRUE	Overflow has occurred
FALSE	Overflow has not occurred

Description

Gets memory overflow status of Sound Input device.

Note

This function returns different information from the one returned by `wsStmIsOverflow()`.

During recording, Sound Input device gets sampling data in buffer of Sound Input device at intervals of 1/60 sec. When interrupt process is prohibitted for a long time, `wsBufExecServer()` is not called and this function returns `TRUE`. In 11kHz 16bit PCM, Sound Input device gets number of 0xba or 0xb8 samples. Maximum recordable number of samples in buffer of Sound Input device is 0xf0.

wsBufGetSfreq

Gets sampling frequency information.

Format

```
Sint32 wsBufGetSfreq( wsbuf )
WSBUF wsbuf
```

Parameters

wsbuf	WSBUF handle

Return Value

WSD_BUF_SFREQ_11KHZ	11025Hz sampling
WSD_BUF_SFREQ_8KHZ	8000Hz sampling

Description

Gets sampling frequency information of Sound Input device from WSBUF handle.

wsBufGetStat

Gets status of WSB module.

Format

```
Sint32 wsBufGetStat( wsbuf )
WSBUF wsbuf
```

Parameters

wsbuf	WSBUF handle

Return Value

WSD_BUF_STAT_DISCNCT	Sound Input device disconnected
WSD_BUF_STAT_STOP	Stop sampling into ring buffer
WSD_BUF_STAT_REC	Sampling into ring buffer
WSD_BUF_STAT_ERR	Error
WSD_STAT_RETRY	Retry in error
WSD_STAT_RESTART	Restarting by CPU of Sound Input device

Description

Gets status of WSB (Wave Sampling Buffer) module.

wsBufGetWrPos

Gets writing position in ring buffer.

Format

```
WSS_POS wsBufGetWrPos( wsbuf )
WSBUF wsbuf
```

Parameters

`wsbuf`	WSBUF handle

Return Value

Index of sample

Description

Gets position where Sound Input device is writing currently in ring buffer.
You may think the writing position as time from executing `wsBufStart()` function.

Note

Writing position is incremented by unit of samples from start of sampling sound data to stop of it.

wsBufSetAmpGain

Sets AMP gain of Sound Input device.

Format

```
Sint32 wsBufSetAmpGain( wsbuf, gain )
WSBUF wsbuf
Sint32 gain
```

Parameters

wsbuf	WSBUF handle
gain	AMP gain

Return Value

WSD_ERR_OK	Normal termination
WSD_ERR_FATAL	Error

Description

Sets AMP gain to Sound Input device through a specified WSBUF handle.

Maximum volume of AMP gain is +16 and minimum volume gain is -15. (Unit : dB)

Note

With sample Sound Input device, maximum volume of AMP gain is +8 and minimum is -7. Maximum and minimum volume is defined by WSD_BUF_GAIN_MAX and WSD_BUF_GAIN_MIN respectively.

wsBufStart

Starts sampling sound data.

Format

```
Sint32 wsBufStart( wsbuf )
WSBUF wsbuf
```

Parameters

wsbuf	WSBUF handle

Return Value

WSD_ERR_OK	Normal termination
WSD_ERR_FATAL	Error

Description

Starts sampling sound data into a ring buffer and counting a writing position in a ring buffer.

wsBufStop

Stops sampling sound data.

Format

```
Sint32 wsBufStop( wsbuf )
WSBUF wsbuf
```

Parameters

wsbuf	WSBUF handle

Return Value

WSD_ERR_OK	Normal termination
WSD_ERR_FATAL	Error

Description

Stops sampling sound data into ring buffer, and clears writing position of ring buffer.

wsFinish

Terminates library.

Format

```
void wsFinish( void )
```

Return Value

None

Description

Terminates wave sampling library and releases work area.

Note

This function cancels registration in interrupt handler made by `wsBufExecServer()`.

Reference

`wsInit()`	Initializes wave sampling library
`wsBufExecServer()`	Server function

wsInit

Initializes wave sampling library.

Format

```
Sint32 wsInit( void )
```

Parameters

None

Return Value

WSD_ERR_OK	Normal termination
WSD_ERR_FATAL	Error

Description

Initializes wave sampling library.

Note

WSBUF handles can be created up to 4 and WSSTM handles up to 16.

This function registers `wsBufExecServer()` in interrupt handler. `wsBufExecServer()` function does not need to be called by users as interrupt function during V-Blank period or for other use.

Reference

`wsFinish()`	Terminates library

wsStmAddRdPos

Advances a seek position.

Format

```
WSS_POS wsStmAddRdPos( wsstm, nsmpl )
WSSTM wsstm
Sint32 nsmpl
```

Parameters

`wsstm`	WSSTM handle
`nsmpl`	Number of samples to advance seek position

Return Value

Seek position (Unit: sample)

Description

```
Advances a seek position for specified number of samples.
EXAMPLE
WSSTM gWsstm1;
Sint16 buffer1, buffer2;
/* Get samples for user function 1 */
wsStmCopyPcm( gWsstm1, (void *)buffer1, 512  )
userFunction1( buffer1  )
/* Get samples for user function 2 */
wsStmCopyPdm( gWsstm1, (void *)buffer2, 256  )
userFunction2( buffer2 )
/* Advance a seek position */
wsStmAddRdPos( gWsstm1, 256  )
```

Note

In WSSTM module, it does not change the seek position in ring buffer after getting sampling data from it. You should use this function to change the seek position.

wsStmClearOverflow

Clears overflow flag in a handle.

Format

```
void wsStmClearOverflow( wsstm )
WSSTM wsstm
```

Parameters

`wsstm`	WSSTM handle

Return Value

None

Description

Clears an overflow flag in a WSSTM handle.

wsStmCopyPcm

Fetches sampling data.

Format

```
Sint32 wsStmCopyPcm( wsstm, buf, bufsize )
WSSTM wsstm
void *buf
Sint32 bufsize
```

Parameters

`wsstm`	WSSTM handle
`buf`	Pointer to buffer where sampling data is taken
`bufsize`	Buffer size (Unit: sample)

Return Value

Size of sample fetched

Description

Fetches wave data from ring buffer.

Note

This function does not change the reading position of ring buffer. Use `wsStmAddRdPos()` function to advance the reading position.

For more details, please refer to example of `wsStmAddRdPos()` function.

Reference

`wsStmAddRdPos()`	Advances a seek position
`wsStmCopySample()`	Gets sampling data from the ring buffer

wsStmCopySample

Gets sampling data from the ring buffer.

Format

```
Sint32 wsStmCopySample( wsstm, buf, nsample )
WSSTM wsstm
void *buf
Sint32 nsample
```

Parameters

`wsstm`	WSSTM handle
`buf`	Pointer to buffer where sampling data is transferred to
`nsample`	Number of samples

Return Value

Number of samples obtained

Description

Gets sampling data from the ring buffer.

NOote

This function does not change a seek position of ring buffer. `wsStmAddRdPos()` function is used to advance the seek position. Please refer to the example of `wsStmAddRdPos()` function for more details.

If you work Sound Input Peripheral with 8-bit sampling, this function is as same as `wsStmCopyPcm()` function.

Reference

`wsStmAddRdPos()`	Advances a seek position
`wsStmCopyPcm()`	Fetches sampling data

wsStmCreate

Creates WSSTM handle.

Format

```
WSSTM wsStmCreate( wsbuf )
WSBUF wsbuf
```

Parameters

wsbuf	WSBUF handle

Return Value

WSSTM handle

Description

Creates WSSTM handle from WSBUF handle.

wsStmDestroy

Destroys WSSTM handle.

Format

```
void wsStmDestroy( wsstm )
WSSTM wsstm
```

Parameters

`wsstm`	WSSTM handle

Return Value

None

Description

Destroys WSSTM handle.

wsStmGetNumSmpl

Gets number of samples in ring buffer.

Format

```
Sint32 wsStmGetNumSmpl( wsstm )
WSSTM wsstm
```

Parameters

`wsstm`	WSSTM handle

Return Value

Number of samples

Description

Gets number of samples (amount of data) stored in ring buffer.

Note

When the ring buffer is filled up with sampling data, number of samples returned will be the same.

wsStmGetRdPos

Gets a writing position.

Format

```
WSS_POS wsStmGetRdPos( wsstm )
WSSTM wsstm
```

Parameters

`wsstm`	WSSTM handle

Return Value

Writing position (Unit: sample)

Description

Get a writing position.

wsStmGetWsbuf

Gets WSBUF handle from WSSTM handle.

Format

```
WSBUF wsStmGetWsbuf( wsstm )
WSSTM wsstm
```

Parameters

`wsstm`	WSSTM handle

Return Value

WSBUF handle

Description

Gets WSBUF handle from WSSTM handle.

wsStmIsOverflow

Gets overflow information.

Format

```
Bool wsStmIsOverflow( wsstm )
WSSTM wsstm
```

Parameters

`wsstm`	WSSTM handle

Return Value

`TRUE`	Overflow has occurred
`FALSE`	Overflow has not occurred

Description

Gets overflow information.

Overflow occurs when writing position passes a reading position.

wsStmSeekRdPos

Changes a seek position.

Format

```
WSS_POS wsStmSeekRdPos( wsstm, pos, mode )
WSSTM wsstm
WSS_POS pos
Sint32 mode
```

Parameters

`wsstm`	WSSTM handle
`pos`	Seek postion (Unit: sample)
`mode`	Seek mode

Return Value

Seek position (Unit: sample)

Description

Seeks reading position to the specified place in the ring buffer.

Following values are set in seek mode, `mode`.

Definition	Meaning
`WSD_STM_SEEK_SET`	Oldest data position in ring buffer
`WSD_STM_SEEK_CUR`	Current data position in ring buffer
`WSD_STM_SEEK_END`	Latest data position in ring buffer

Note

This function can change a seek position from the oldest position to the latest position in a ring buffer. Usually, `wsStmAddRdPos()` is used in place of this function.

16. Data Type Functions

BUS_BACKUPFILEHEADER

Structure where file image information is stored.

Format

```
typedef struct {
       char vms_comment[18];
       char btr_comment[34];
       Uint8 game_name[16];
       void* icon_palette;
       void* icon_data;
       Uint16 icon_num;
       Uint16 icon_speed;
       void* visual_data;
       Uint16 visual_type;
       Uint16 reserved;
       void* save_data;
       Uint32 save_size;
} BUS_BACKUPFILEHEADER;
```

Return Value

`vms_comment[18]`	VM comment
`btr_comment[34]`	BootROM comment
`game_name[16]`	Game name (sort item)
`*icon_palette`	Icon palette address
`*icon_data`	Icon data address
`icon_num`	Icon number
`icon_speed`	Icon animation speed
`*visual_data`	Visual data address
`visual_type`	Visual type

reserved	Reserved
*save_data	Application save data address
save_size	Application save data size

Description

Structure where memory card file image creation and analysis information is stored.

Reference

buAnalyzeBackupFileImage()	Creates a file header image
buMakeBackupFileImage()	Gets file image

BUS_DISKINFO Structure where memory card information is stored.

Defînçition

```
typedef struct {
       Sint8 volume[32];
       Uint16 total_blocks;
       Uint16 total_user_blocks;
       Uint16 free_blocks;
       Uint16 free_user_blocks
       Uint16 total_exe_blocks;
       Uint16 free_exe_blocks
       Uint16 block_size;
       Uint16 icon_no;
       BUS_TIME time;
} BUS_DISKINFO;
```

Return Value

`volume`	Volume data
`total_blocks`	Number of total blocks
`total_user_blocks`	Number of total user blocks (maximum file number)
`free_blocks`	Number of free blocks
`free_user_blocks`	Number of free user blocks
`total_exe_blocks`	Number of total blocks for all executable files
`free_exe_blocks`	Number of free blocks for executable files
`block_size`	Block size (512 fixed)
`icon_no`	Icon number (0-255)
`time`	Formatted time

Description

Structure where memory card information is stored.

Reference

`buGetDiskInfo()`	Gets memory card information

BUS_FILEINFO

Structure where file information is stored

Format

```
typedef struct {
      Uint32 filesize;
      Uint16 blocks;
      Uint8 type;
      Uint8 copyflag;
      Uint16 headerofs;
      BUS_TIME time;
} BUS_FILEINFO;
```

Return Value

`filesize`	File size (byte count)
`blocks`	Used block number
`type`	File type
`copyflag`	Copy flag (FFH=copy impossible)
`headerofs`	Header offset (0 fixed)
`time`	Time type

Description

Structure where file information is stored.
Values set for type, which expresses the file type, are as follows:

Definition	Meaning
BUD_FILETYPE_NORMAL	Normal file
BUD_FILETYPE_EXECUTABLE	Executable file

Reference

`buGetFileInfo()`	Gets file information

BUS_TIME

Structure that stores the time stamp of a file.

Format

```
typedef struct {
       Uint16 year;
       Uint8 month;
       Uint8 day;
       Uint8 hour;
       Uint8 minute;
       Uint8 second;
       Uint8 dayofweek;
} BUS_TIME;
```

Return Value

`year`	Year (1998 or later)
`month`	Month (1 to 12)
`day`	Date (1 to 31)
`hour`	Hour (0 to 23)
`minute`	Minute (0 to 59)
`second`	Second (0 to 59)
`dayofweek`	Days of the week (Sunday=0 to Saturday=6)

Description

Structure that stores the time stamp of a file recorded in the memory card.

Reference

`buFormatDisk()`	Formats a memory card
`buSaveExecFile()`	Saves an executable file
`buSaveFile()`	Saves a file

GDD_ERR

File error.

Description

GDD_ERR_OK	Normal termination
GDD_ERR_INIT	Library initialization failure
GDD_ERR_RESET	Drive initialization failure
GDD_ERR_LIBOV	Overlay initialization failure
GDD_ERR_MOUNT	Mount unsuccessful
GDD_ERR_DISC	Incorrect disk used
GDD_ERR_DIRREC	Incorrect directory code handle used
GDD_ERR_CANTOPEN	File cannot be opened
GDD_ERR_NOTFOUND	File cannot be found
GDD_ERR_NOHNDL	Unused handle not found
GDD_ERR_ILLHNDL	Incorrect handle used
GDD_ERR_NOTDIR	Directory cannot be used
GDD_ERR_DIROVER	Exceeded storable entries
GDD_ERR_CHECKBUSY	In process
GDD_ERR_32ALIGN	Not aligned with 32-byte boundary
GDD_ERR_SIZE	Not 32-byte unit
GDD_ERR_SEEK	Illegal seek specification
GDD_ERR_OFS	Illegally specified position
GDD_ERR_ILLTMODE	Abnormal transfer mode
GDD_ERR_READ	Read failure
GDD_ERR_NOTREAD	No read
GDD_ERR_TOUT	Timed out
GDD_ERR_EOF	File end reached
GDD_ERR_TRAYOPEND	Disc door is open
GDD_ERR_SIZEOVER	Request size is too large
GDD_ERR_FATAL	Fatal error
GDD_ERR_UNDEF	Undefined error
GDD_ERR_NOERR	No error
GDD_ERR_RECOVER	Error recovered
GDD_ERR_NOTREADY	Drive not ready
GDD_ERR_MEDIA	Media error
GDD_ERR_HWARE	Hardware error

GDD_ERR_OK	Normal termination
GDD_ERR_ILLREQ	Illegal request issued
GDD_ERR_UNITATTENT	Detection of media exchange
GDD_ERR_PROTECT	Protected
GDD_ERR_ABORT	Aborted
GDD_ERR_NOREADABLE	Unreadable
GDD_ERR_CHECKBUSY	Media check in process

GDFS_DAINFO

Structure where DA file information is stored.

Format

```
typedef struct GDS_FS_DAINFO {
      Sint32 track;
      Sint32 min;
      Sint32 sec;
      Sint32 frame;
      Sint32 fad;
} GDFS_DAINFO;
```

Return Value

`track`	Track number
`min`	Progress time (minutes)
`sec`	Progress time (seconds)
`frame`	Progress time (frames)
`fad`	Frame address

Description

Structure where DA file information is stored.

Reference

`gdFsDaGetInfo()`	Gets the DA playback information

GDFS_DIRINFO Structure where file directory information is stored.

Format

```
typedef struct GDS_FS_DIRINFO {
      Sint32 fad;
      Sint32 fsize;
      Uint8  flag;
      Uint8  pad[3];
} GDFS_DIRINFO;
```

Return Value

`fad`	Starting address FAD
`fsize`	File size
`flag`	File flag
`pad[3]`	Reserved

Description

Structure where file directory information is stored.

Bits of the member flags have the following meaning.

Bit	Meaning
Bit0	Existence
Bit1	Directory
Bit2	Associated
Bit3	Record
Bit4	Protection
Bit5	****
Bit6	****
Bit7	Multi Extent

Reference

`gdFsGetDirInfo()`	Gets file information

PDS_KEYBOARD

Keyboard status.

Format

```
typedef struct {
      Uint8 ctrl;
      Uint8 led;
      Uint8 key[6];
      PDS_KEYBOARDINFO *info;
} PDS_KEYBOARD;
```

Return Value

`ctrl`	Special key status
`led`	LED light on/off status
`key[6]`	Key data
`info`	Pointer to keyboard information structure

Description

Structure that stores input data from the keyboard.

Which key is currently being pressed is stored in the member `ctrl`. When a key is pressed, the corresponding bit is 1; when it is not pressed, the corresponding bit is 0.

Definition	Meaning
PDD_KEY_CTRL_RGUI/S2	S2
PDD_KEY_CTRL_RALT	Right-ALT
PDD_KEY_CTRL_RSHIFT	Right-Shift
PDD_KEY_CTRL_RCTRL	Right-Ctrl
PDD_KEY_CTRL_LGUI/S1	S1
PDD_KEY_CTRL_LALT	Left-Alt
PDD_KEY_CTRL_LSHIFT	Left-Shift
PDD_KEY_CTRL_LCTRL	Left-Ctrl

Which LED is currently being on is stored in the member `led`. When LED is on, the corresponding bit is 1; when it is off, the corresponding bit is 0.

Definition	Meaning
PDD_LED_SHIFT	Shift
PDD_LED_POWER	Power
PDD_LED_KANA	Kana (Japanese character)
PDD_LED_SCRLOCK	Scroll Lock
PDD_LED_CAPLOCK	Caps Lock
PDD_LED_NUMLOCK	Num Lock

The pressed key code is stored in the member key. Key codes are stored in order from key [0] to key [5] up to the 6th pressed key.

If fewer than 6 keys are pressed, the remaining members are stored as 0x00. Detection cannot be done of 7 or more simultaneously pressed keys. If 7 or more keys are pressed at the same time, 0x01 is stored in key [0].

Also, key code detection of less than 6 keys may fail by key combination pressed simultaneously. If this happens, 0x01 is stored in key [0].

Code	Code (hexidecimal)	English 104 Keyboard (US)	English 105 Keyboard (UK	Japanese 92 Keyboard
0	00h	No operation	No operation	No operation
1	01h	Roll-over error	Roll-over error	Roll-over error
2	02h	POST Fail	POST Fail	POST Fail
3	03h	Undefined error	Undefined error	Undefined error
4	04h	“a”, “A”	“a”, “A”	“a”, “A”
5	05h	“b”, “B”	“b”, “B”	“b”, “B”
6	06h	“c”, “C”	“c”, “C”	“c”, “C”
7	07h	“d”, “D”	“d”, “D”	“d”, “D”
8	08h	“e”, “E”	“e”, “E”	“e”, “E”
9	09h	“f”, “F”	“f”, “F”	“f”, “F”
10	0Ah	“g”, “G”	“g”, “G”	“g”, “G”
11	0Bh	“h”, “H”	“h”, “H”	“h”, “H”
12	0Ch	“i”, “I”	“i”, “I”	“i”, “I”
13	0Dh	“j”, “J”	“j”, “J”	“j”, “J”
14	0Eh	“k”, “K”	“k”, “K”	“k”, “K”
15	0Fh	“l”, “L”	“l”, “L”	“l”, “L”
16	10h	“m”, “M”	“m”, “M”	“m”, “M”
17	11h	“n”, “N”	“n”, “N”	“n”, “N”
18	12h	“o”, “O”	“o”, “O”	“o”, “O”

Code	Code (hexidecimal)	English 104 Keyboard (US)	English 105 Keyboard (UK	Japanese 92 Keyboard
19	13h	“p”, “P”	“p”, “P”	“p”, “P”
20	14h	“q”, “Q”	“q”, “Q”	“q”, “Q”
21	15h	“r”, “R”	“r”, “R”	“r”, “R”
22	16h	“s”, “S”	“s”, “S”	“s”, “S”
23	17h	“t”, “T”	“t”, “T”	“t”, “T”
24	18h	“u”, “U”	“u”, “U”	“u”, “U”
25	19h	“v”, “V”	“v”, “V”	“v”, “V”
26	1Ah	“w”, “W”	“w”, “W”	“w”, “W”
27	1Bh	“x”, “X”	“x”, “X”	“x”, “X”
28	1Ch	“y”, “Y”	“y”, “Y”	“y”, “Y”
29	1Dh	“z”, “Z”	“z”, “Z”	“z”, “Z”
30	1Eh	“1”, “!”	“1”, “!”	“1”, “!”
31	1Fh	“2”, “@”	“2”, ““”	“2”, ““”
32	20h	“3”, “#”	“3”, “£”	“3”, “#”
33	21h	“4”, “$”	“4”, “$”	“4”, “$”
34	22h	“5”, “%”	“5”, “%”	“5”, “%”
35	23h	“6”, “^”	“6”, “^”	“6”, “&”
36	24h	“7”, “&”	“7”, “&”	“7”, “‘”
37	25h	“8”, “*”	“8”, “*”	“8”, “(”
38	26h	“9”, “(”	“9”, “(”	“9”, “)”
39	27h	“0”, “)”	“0”, “)”	“0”, “~”
40	28h	“ENTER”	“↵”	“ENTER”
41	29h	“ESC”	“ESC”	“ESC”
42	2Ah	“←” (Backspace)	“←” (Backspace)	“Back Space”
43	2Bh	“TAB”	“TAB”	“TAB”
44	2Ch	Spacebar	Spacebar	Spacebar
45	2Dh	“–“, “_“	“–“, “_“	“_“, “=“
46	2Eh	“=“, “+“	“=“, “+“	“^“, “¯“
47	2Fh	“[“, “{“	“[“, “{“	“@“, “`“
48	30h	“]“, “}“	“]“, “}“	“[“, “{“
49	31h	“\“, “|“	Not Used	Not Used
50	32h	Not Used	“#“, “~“	“]“, “}“
51	33h	“;“, “:“	“;“, “:“	“;“, “+“
52	34h	““, ““	““, “@“	“:“, “*“

Code	Code (hexidecimal)	English 104 Keyboard (US)	English 105 Keyboard (UK	Japanese 92 Keyboard
53	35h	“`“,“¬“	“`“,“~“	“半角/全角“
54	36h	“.“,“<“	“.“,“<“	“.“,“<“
55	37h	“.“,“>“	“.“,“>“	“.“,“>“
56	38h	“/“,“?“	“/“,“?“	“/“,“?“
57	39h	“Caps Lock”	“Caps Lock”	“Caps Lock”
58	3Ah	“F1”	“F1”	“F1”
59	3Bh	“F2”	“F2”	“F2”
60	3Ch	“F3”	“F3”	“F3”
61	3Dh	“F4”	“F4”	“F4”
62	3Eh	“F5”	“F5”	“F5”
63	3Fh	“F6”	“F6”	“F6”
64	40h	“F7”	“F7”	“F7”
65	41h	“F8”	“F8”	“F8”
66	42h	“F9”	“F9”	“F9”
67	43h	“F10”	“F10”	“F10”
68	44h	“F11”	“F11”	“F11”
69	45h	“F12”	“F12”	“F12”
70	46h	“Print Screen”	“Print Screen”	“Print Screen”
71	47h	“Scroll Lock”	“Scroll Lock”	“Scroll Lock”
72	48h	“Pause”	“Pause”	“Pause”
73	49h	“Insert”	“Insert”	“Insert”
74	4Ah	“Home”	“Home”	“Home”
75	4Bh	“Page Up”	“Page Up”	“Page Up”
76	4Ch	“Delete”	“Delete”	“Delete”
77	4Dh	“End”	“End”	“End”
78	4Eh	“Page Down”	“Page Down”	“Page Down”
79	4Fh	“→”	“→”	“→”
80	50h	“←”	“←”	“←”
81	51h	“↓”	“↓”	“↓”
82	52h	“↑”	“↑”	“↑”
83	53h	[“Num Lock”]	[“Num Lock”]	“Not Used”
84	54h	[“/”]	[“/”]	“Not Used”
85	55h	[“*”]	[“*”]	“Not Used”
86	56h	[“–”]	[“–”]	“Not Used”

Code	Code (hexidecimal)	English 104 Keyboard (US)	English 105 Keyboard (UK	Japanese 92 Keyboard
87	57h	[“+”]	[“+”]	“Not Used”
88	58h	[“Enter”]	[“Enter”]	“Not Used”
89	59h	[“1”,“End”]	[“1”,“End”]	“Not Used”
90	5Ah	[“2”,“↓”]	[“2”,“↓”]	“Not Used”
91	5Bh	[“3”,“Pg Dn”]	[“3”,“Pg Dn”]	“Not Used”
92	5Ch	[“4”,“←”]	[“4”,“←”]	“Not Used”
93	5Dh	[“5”]	[“5”]	“Not Used”
94	5Eh	[“6”,“→”]	[“6”,“→”]	“Not Used”
95	5Fh	[“7”,“Home”]	[“7”,“Home”]	“Not Used”
96	60h	[“8”,“↑”]	[“8”,“↑”]	“Not Used”
97	61h	[“9”,“Pg Up”]	[“9”,“Pg Up”]	“Not Used”
98	62h	[“0”,“Ins”]	[“0”,“Ins”]	“Not Used”
99	63h	[“.”,“Del”]	[“.”,“Del”]	“Not Used”
100	64h	“Not Used”	“\”,“|”	“Not Used”
101	65h	“S3”	“S3”	“S3”
135	87h	“Not Used”	“Not Used”	“\”,“_”
136	88h	“Not Used”	“Not Used”	“カタカナ”,“ひらがな”
137	89h	“Not Used”	“Not Used”	“¥”,“|”
138	8Ah	“Not Used”	“Not Used”	“変換”
139	8Bh	“Not Used”	“Not Used”	“無変換”

Note: Key tops in brackets ([]) indicate key tops for ten-key pad.

Note

Keyboard with LED is not planned on release at present.

Reference

`pdKbdGetData()` Gets keyboard data

PDS_KEYBOARDINFO

Keyboard hardware information.

Format

```
typedef struct {
       Uint8 lang;
       Uint8 type;
       Uint8 led;
       Uint8 led_ctrl;
} PDS_KEYBOARDINFO;
```

Return Value

`lang`	Keyboard language
`type`	Keyboard type
`led`	LED presence/absence
`led_ctrl`	Whether keyboard controls LEDs

Description

This structure holds the keyboard hardware information.

The member lang holds the keyboard language, coded as follows:

Value	Meaning
PDD_KBDLANG_JP	Japan
PDD_KBDLANG_US	USA
PDD_KBDLANG_UK	UK
PDD_KBDLANG_GERMANY	Germany
PDD_KBDLANG_FRANCE	France
PDD_KBDLANG_ITALY	Italy
PDD_KBDLANG_SPAIN	Spain
PDD_KBDLANG_SWEDEN	Sweden
PDD_KBDLANG_SWITZER	Switzerland
PDD_KBDLANG_NETHER	Netherlands
PDD_KBDLANG_PORTUGAL	Portugal
PDD_KBDLANG_LATIN	Latin America
PDD_KBDLANG_CANFRENCH	Canadian French
PDD_KBDLANG_RUSSIA	Russia
PDD_KBDLANG_CHINA	China
PDD_KBDLANG_KOREA	Korea

The member type holds the keyboard type, coded as follows:

Value	Meaning
PDD_KBDTYPE_89	89 keys
PDD_KBDTYPE_92	92 keys
PDD_KBDTYPE_101	101 keys
PDD_KBDTYPE_102	102 keys
PDD_KBDTYPE_104	104 keys
PDD_KBDTYPE_105	105 keys
PDD_KBDTYPE_106	106 keys
PDD_KBDTYPE_109	109 keys
PDD_KBDTYPE_87	87 keys
PDD_KBDTYPE_88	88 keys

The member led includes bits indicating whether or not the keyboard has particular LEDs. A bit set to 1 indicates the present of an LED, and a bit set to 0 indicates the LED is absent. The bits are coded as follows:

Value	Meaning
PDD_LED_SHIFT	Shift
PDD_LED_POWER	Power
PDD_LED_KANA	Kana
PDD_LED_SCRLOCK	Scroll Lock
PDD_LED_CAPLOCK	Caps Lock
PDD_LED_NUMLOCK	Num Lock

The member `led_ctrl` indicates whether LEDs on the keyboard are switched on and off by the keyboard itself or should be controlled by the host (Dreamcast). The current library does not support control of LEDs. The control value is coded as follows:

Value	Meaning
PDD_KBDCTRL_HOST	Controlled by host (Dreamcast)
PDD_KBDCTRL_KEYBOARD	Controlled by keyboard

Note

Since all European language keyboards are identified as UK105 keyboards, the software cannot distinguish them. Applications supporting European language keyboards must take note of this.

Reference

`pdKbdGetInfo()` Gets hardware information of the keyboard

PDS_PERIPHERAL

Structure where control pad status information is stored.

Format

```
typedef struct {
       Uint32 id;
       Uint32 support;
       Uint32 on;
       Uint32 off;
       Uint32 press;
       Uint32 release;
       Uint16 r;
       Uint16 l;
       Sint16 x1;
       Sint16 y1;
       Sint16 x2;
       Sint16 y2;
       Sint8* name;
       void* extend;
       Uint32 old;
       PDS_PERIPHERALINFO *info;
} PDS_PERIPHERAL;
```

Return Value

`id`	Device ID
`support`	Button, lever support status
`on`	Digital button status
`off`	Digital button status (reverse)
`press`	Digital button down edge status
`release`	Digital button up edge status
`r`	R trigger value (0 to 255)
`l`	L trigger value (0 to 255)
`x1`	X-value (-128 to 127) of analog directional key 1
`y1`	Y-value (-128 to 127) of analog directional key 1
`x2`	X-value (-128 to 127) of analog directional key 2
`y2`	Y-value (-128 to 127) of analog directional key 2
`name`	Device name
`extend`	Extended data address (unused)
`old`	Reserved
`info`	Pointer to peripheral information

Description

Structure where control pad status information is stored.

Member `support` has the following bit assignments. Bits are stored as 1 if the front has buttons and levers, and 0 if the front has no buttons or levers.

Constant for bit assignment	Button or Lever
PDD_DEV_SUPPORT_KU	Directional button A up
PDD_DEV_SUPPORT_KD	Directional button A down
PDD_DEV_SUPPORT_KL	Directional button A left
PDD_DEV_SUPPORT_KR	Directional button A right
PDD_DEV_SUPPORT_KUB	Directional button B up
PDD_DEV_SUPPORT_KDB	Directional button B down
PDD_DEV_SUPPORT_KLB	Directional button B left
PDD_DEV_SUPPORT_KRB	Directional button B right
PDD_DEV_SUPPORT_ST	Start button
PDD_DEV_SUPPORT_TA	A button
PDD_DEV_SUPPORT_TB	B button
PDD_DEV_SUPPORT_TC	C button
PDD_DEV_SUPPORT_TD	D button
PDD_DEV_SUPPORT_TX	X button
PDD_DEV_SUPPORT_TY	Y button
PDD_DEV_SUPPORT_TZ	Z button
PDD_DEV_SUPPORT_AR	R trigger
PDD_DEV_SUPPORT_AL	L trigger
PDD_DEV_SUPPORT_AX1	Analog directional key 1x
PDD_DEV_SUPPORT_AY1	Analog directional key 1y
PDD_DEV_SUPPORT_AX2	Analog directional key 2x
PDD_DEV_SUPPORT_AY2	Analog directional key 2y

For the four members of `on`, `off`, `press` and `release`, digital information is stored and used as needed. Buttons are allocated in the bits of each member and have the property of 1 if pressed, 0 if not pressed (positive logic). Furthermore, the logic status is set to negative logic when the initialization function `pdInitPeripheral()` is called.

Bit assignments	Digital button
PDD_DGT_KU	Directional button A up
PDD_DGT_KD	Directional button A down
PDD_DGT_KL	Directional button A left
PDD_DGT_KR	Directional button A right
PDD_DGT_KUB	Directional button B up
PDD_DGT_KDB	Directional button B down
PDD_DGT_KLB	Directional button B left
PDD_DGT_KRB	Directional button B right
PDD_DGT_ST	Start button
PDD_DGT_TA	A button
PDD_DGT_TB	B button
PDD_DGT_TC	C button
PDD_DGT_TD	D button
PDD_DGT_TX	X button
PDD_DGT_TY	Y button
PDD_DGT_TZ	Z button
PDD_DGT_TL	L button
PDD_DGT_TR	R button

The corresponding bit for member on becomes 1 when the button is pressed (button down). When the button is not pressed, the bit is 0.

Member off is the reverse bit of member on. The corresponding bit for member off is 1 when the button is not pressed (button up). When member press status changes from the button not being pressed to a pressed status (button down edge), the corresponding bit is 1. Otherwise the bit is 0.

When member release status changes from the the button being pressed to the button not being pressed (button up edge), the corresponding bit is 1. Otherwise, the bit is 0.

If negative logic is set, all bits for button, on, off, press, release are inversely stored.

Digital LR button information is imitated by the software from the LR trigger information. Note that if there is no LR trigger device, the bit does not change. (There is no device that physically has digital LR buttons).

Analog data is stored in members r, l, x1, y1, x2, and y2. These center position values are 0. The display ID of device type and device name are stored in a gotten structure.

In the Maple Bus specs, the device ID is not oriented for handling by the application because the device ID that can be acquired has a complicated format. Acquiring the device ID is simplified in the Shinobi library and is defined as follows.

Device	DeviceID (id)	Device name (name)
Standard control pad	PDD_DEV_CONTROLER	The brand name for the device

Note

Because controller devices always have the direction key A, start button, A button, B button, it is possible to maintain compatibility by combining applications using only these buttons.

Reference

`pdGetPeripheral()`	Gets the controller button status
`pdGetPeripheralDirect()`	Gets peripheral data (for low latency)

PDS_PERIPHERALINFO

Structure where inherent peripheral information is stored.

Format

```
typedef struct {
       Uint32 type;
       Uint32 reserved[3];
       Uint8 is_root;
       Uint8 area_code;
       Uint8 connector_dir[2];
       Sint8 product_name[32];
       Sint8 license[64];
       Uint16 stdby_pow;
       Uint16 max_pow;
} PDS_PERIPHERALINFO;
```

Return Value

`type`	Peripheral type
`reserved[3]`	Reserved
`is_root`	Root device flag
`area_code`	Area code
`connector_dir[2]`	Extended connector oriented
`product_name[32]`	Product name
`license[64]`	License string
`stdby_pow`	Standby power consumption
`max_pow`	Maximum power consumption

Description

Structure where inherent peripheral information is stored.

The following values are defined in the member type that returns peripheral types.

Definition	Meaning
PDD_DEVTYPE_CONTROLLER	Controller device
PDD_DEVTYPE_STORAGE	Storage device (memory card)
PDD_DEVTYPE_LCD	LCD device
PDD_DEVTYPE_TIMER	Timer device
PDD_DEVTYPE_SOUNDINPUT	Sound input peripheral
PDD_DEVTYPE_KEYBOARD	Keyboard peripheral
PDD_DEVTYPE_LIGHTGUN	Gun peripheral
PDD_DEVTYPE_VIBRATION	Vibration peripheral

Each is expressed in bits. One peripheral can have more than one type. For example, "Visual Memory" returns "memory card", "LCD", and "Timer". The following values are defined in the member `area_code` which indicates a peripheral's area code.

Definition	Meaning
PDD_DEVAREA_USA	North American Region
PDD_DEVAREA_JAPAN	Japanese Region
PDD_DEVAREA_ASIA	Asian Region
PDD_DEVAREA_EUROPE	European Region

Each is expressed in bits, and peripherals can be supported for more than one area. Following values are defined for the expansion socket directions. Following values are defined for the member connector_dir[2] which shows the direction of the peripheral expansion socket.

Definition	Meaning
PDD_CONDIR_TOPSIDE	Up
PDD_CONDIR_BOTTOMSIDE	Down
PDD_CONDIR_LEFTSIDE	Left
PDD_CONDIR_RIGHTSIDE	Right

These values express the direction an expansion socket is facing when the peripheral is a root device (a peripheral directly connected to a port).

The character string of the name registered by each peripheral is returned in the member `product_name[32]` which shows the product name of a peripheral.

The manufacturer's name is returned in the member `license[64]` which shows character strings of licenses.

The member `stdby_pow`, which shows standby current, and `max_pow`, which shows maximum current, are in units of 0.1mA (milliamperes). Maximum current to the control port is 1000mA.

Reference

`pdGetPeripheralInfo()` Gets data that is inherent to a peripheral

PDS_TIME

Structure where time set by timer device is stored.

Format

```
typedef struct {
       Uint16 year;
       Uint8 month;
       Uint8 day;
       Uint8 hour;
       Uint8 minute;
       Uint8 second;
       Uint8 dayofweek;
} PDS_TIME;
```

Return Value

`year`	Year
`month`	Month
`day`	Date
`hour`	Hour
`minute`	Minute
`second`	Second
`dayofweek`	Days of the week (Sunday=0, Monday=1, ..., Saturday=6)

Description

Structure where time set by timer device is stored.

Note

Member dayofweek needs to be set valid value when setting the time for timer device. When getting the time, calculate day of the week from year, month and day without referring to the stored data. This is to correspond to future timer device.

Reference

`pdTmrGetTime()`	Gets time of timer device
`pdTmrSetTime()`	Sets time of timer device

PDS_VIBINFO

Structure where vibrating peripheral information is stored.

Format

```
typedef struct {
       Uint8 units;
       Uint8 se_units;
} PDS_VIBINFO;
```

Return Value

units	Number of units
se_units	Possible units of synchronous vibration setting

Description

Structure where vibration peripheral information is stored.

Reference

pdVibGetInfo()	Gets vibration peripheral information

PDS_VIBPARAM Structure where vibration parameters are stored.

Format

```
typedef struct {
       Uint8 unit;
       Uint8 flag;
       Sint8 power;
       Uint8 freq;
       Uint8 inc;
       Uint8 reserved[3];
} PDS_VIBPARAM;
```

Return Value

`unit`	Unit number that sends the vibration parameter
`flag`	Vibration flag
`power`	Vibration intensity
`freq`	Vibration frequency
`inc`	Vibration incline cycle
`reserved`	Reserved

Description

Structure that stores the vibration parameters of the vibration peripheral.

Reference

`pdVibMxStart()`	Starts vibration for vibration peripheral
`pdVibStart()`	Starts vibration in vibration peripheral

PDS_VIBUNITINFO

Structure where vibration peripheral unit information is stored.

Format

```
typedef struct {
       Uint8 position;
       Uint8 axis;
       Uint8 pow_enable;
       Uint8 cont_enable;
       Uint8 dir_enable;
       Uint8 wave_enable;
       Uint8 min_freq;
       Uint8 max_freq;
} PDS_VIBUNITINFO;
```

Return Value

`position`	Vibration peripheral unit position
`axis`	Vibration axis
`pow_enable`	Intensity variable
`cont_enable`	Continuous vibration enable
`dir_enable`	Direction specification enable
`wave_enable`	Optional waveform setting enable
`min_freq`	Minimum vibration frequency
`max_freq`	Maximum vibration frequency

Description

Structure where unit information for the vibration peripheral is stored.

SYS_BT_FNT_INFO

Structure where font information is stored.

Format

```
typedef struct {
       Uint8  width;
       Uint8  height;
} SYS_BT_FNT_INFO;
```

Return Value

`width`	Font width (pixel)
`height`	Font height (pixel)

Description

Structure where font information is stored.

Reference

`syBtFntGetInfo()`	Gets font size

SYS_BT_SYSTEMID Structure where product information is stored.

Format

```
struct tag_SYS_BT_SYSTEMID{
       Sint32 nNo ;
       Sint32 nAll ;
       Sint8  szProduct[16];
} ;
typedef struct tag_SYS_BT_SYSTEMID SYS_BT_SYSTEMID;
```

Return Value

`nNo`	The value of which disk includes system ID media information
`nAll`	The value of which program disk includes system ID media information
`szProduct`	Product code up to 10 characters (the 11th byte = szProduct[10] is NULL)

Description

Structure where the product code is stored.

Reference

`syBtGetBootSystemID()`	Gets the system ID information on the startup disc
`syBtGetCurrentSystemID()`	Gets the system ID information on the disc

WSS_POS

Index of sample.

Format

```
typedef struct _WSS_POS{
       Uint32 high;
       Uint32 low;
} WSS_POS;
```

Return Value

`high`	63bit to 32bit
`low`	31bit to 0bit

Description

`WSS_POS` shows an index of sample for Sound Input device. Writing position is incremented by unit of samples from start of sampling sound data to stop of it.

Reference

`wsBufGetWrPos()`	Gets writing position in ring buffer
`wsStmAddRdPos()`	Advances a seek position
`wsStmGetRdPos()`	Gets a writing position
`wsStmSeekRdPos()`	Changes a seek position

SYS_RTC_DATE

Structure where date information is stored.

Format

```
typedef struct {
      Uint16 year;
      Uint8  month;
      Uint8  day;
      Uint8  hour;
      Uint8  minute;
      Uint8  second;
      Uint8  dayofweek;
      Uint8  ageofmoon;
} SYS_RTC_DATE;
```

Return Value

year	Year (1950 to 2085)
month	Month (1 to 12)
day	Date (1 to 31)
hour	Hour (0 to 23)
minute	Minute (0 to 59)
second	Second (0 to 59)
dayofweek	Days of the week (Sunday=0 to Saturday=6)
ageofmoon	Moon phase (0 to 29; new moon=0, full moon=15)

Description

Structure that manages date information.

Reference

syRtcCompareDate()	Compares date and time
syRtcCountToDate()	Converts a count into a date and time value
syRtcDateToCount()	Converts a date and time value into a count
syRtcGetDate()	Gets the date and time
syRtcSetDate()	Sets the date and time

WSS_BUF_PRM

Parameter for WSBUF handle.

Format

```
typedef struct _WSS_BUF_PRM{
       Sint32 sfreq;
       Sint32 bps;
       Sint32 blksize;
       Sint32 nblk;
       Sint32 gain;
       void *bufptr;
} WSS_BUF_PRM;
```

Return Value

`sfreq`	Sampling frequency
`bps`	Bits per a sample
`gain`	AMP gain
`blksize`	Number of samples in a block size
`nblk`	Number of block in a ring buffer
`bufptr`	Address of a ring buffer

Description

The structure that sets parameters when creating the WSBUF handle. Parameter for creating Sound Input device control and ring buffer.

The following values are defined in the member sfreq which expresses the sampling frequency.

Value	Definition
WSD_BUF_SFREQ_11KHZ	11025Hz sampling
WSD_BUF_SFREQ_8KHZ	8000Hz sampling

The following values are defined in the member bps which expresses the bit number per 1 sample.

Value	Definition
WSD_BUF_BPS_16BIT	16 bit linear
WSD_BUF_BPS_8BIT	8bit mu-law codec

When 16 bit linear, sound is sampled as 16 bit and the effective bit number is upper 14 bit. When 8 bit mu-law codec, data that has been coded is stored in the ring buffer. Coding is necessary for deciding sound data.

The values `WSD_BUF_GAIN_MAX` to `WSD_BUF_GAIN_MIN` can be specified in the member `gain` that expresses AMP gain.

Reference

`wsBufCreate()`	Creates WSBUF handle

17. PAL Support for European Software

European software requires the use of PAL screen modes, to ensure compatibility with 100% of televisions in Europe.

The traditional side effect of this is that games tend to run 17.5% slower than intended and often feature ugly borders at the top and bottom of the screen. This is due to the greater number of lines on a 50 Hz display.

Extended PAL Modes

There are now EXTENDED PAL modes available with the latest versions of Sega Libraries.

These modes effectively stretch the display to fit a number of increased vertical resolutions, thus correcting the aspect ratio and eliminating the borders. The only penalty; a slight increase in frame buffer size, and therefore a slight drop in the amount of Video RAM available for textures.

Details of how to use these modes are included. Due to the way these modes work, there is no need to adjust clipping regions or aspect rations within a 3D engine to compensate for the extra lines.

We see little reason why an Extended PAL mode should not be the default mode for European software, due to the ease with which it can be implemented.

The following chart shows the additional Frame buffer memory requirement for each available aspect ratio.

Ratio	Height	Add Memory
1.033	495	40K
1.066	511	80K
1.1	527	120K
1.133	543	160K
1.166	559	200K
(Based on a non-interlaced 640*480 16-Bit display)		

Using Extended PAL modes.

Here's how to use the Extended PAL modes

The first this you need to do is set up the callback function. The callback is triggered by `kmSetDisplayMode()`, so you must register it before calling this function. This function is called in `sbInitSystem()`. The example below shows this in use with the `sbinit.c` file.

If you are using your own version of `sbInitSystem()` then you can insert this code into that function - there is no need to call `kmSetDisplayMode()` twice.

```
/*-------------------------------------------------------------*/
/*  Function Name   : PALExtCallbackFunc                       */
/*  Inputs          : pCallbackArguments                       */
/*  Outputs         : None                                     */
/*  Returns         : None                                     */
/*  Globals Used    :                                          */
/*  Description     : PAL mode callback (set the height ratio)*/
/*-------------------------------------------------------------*/
VOID STATIC             PALExtCallbackFunc(PVOID pCallbackArguments)
{
PKMPALEXTINFO           pInfo;

       pInfo = (PKMPALEXTINFO)pCallbackArguments;
       pInfo->nPALExtMode = KM_PALEXT_HEIGHT_RATIO_1_133;           // or
whatever
       return;
}


void main(void)
{
    SYE_CBL_CABLE cable;

    cable = syCblCheck();
    switch (cable)
    {
       case      SYE_CBL_VGA:
                 sbInitSystem((int)KM_DSPMODE_VGA, (int)KM_DSPBPP_RGB565, 1);
                 break;
       case      SYE_CBL_NTSC:
                 sbInitSystem((int)KM_DSPMODE_NTSCNI640x480,
(int)KM_DSPBPP_RGB565, 1);
                 break;
       case      SYE_CBL_PAL:
                 sbInitSystem((int)KM_DSPMODE_PALNI640x480EXT,
(int)KM_DSPBPP_RGB565, 1);

                 /*                              Change the PAL height ratio */
                 kmSetPALEXTCallback(PALExtCallbackFunc,NULL);
                 kmSetDisplayMode (KM_DSPMODE_PALNI640x480EXT,
KM_DSPBPP_RGB565, TRUE, FALSE);
                 break;
    }
```

60 Hz mode

Also possible, and very much encouraged, is an optional 60 Hz video mode in European software. One of the reasons we allow this, is just because it's possible on Dreamcast and we thought it would be a valuable feature, especially amongst the "hardcore" gamers. Another reason was due to the VGA option Dreamcast offers.

Even on a PAL Dreamcast, the use of VGA results in 60 Hz gameplay. So potentially having to support correct gameplay timing and speed at both 50 and 60 Hz is something that may well be necessary. That being the case, it's only a matter of including the relevant menus to enable 60Hz play on a TV too.

The Software Standards contain information on the procedures to follow when implementing 60Hz video modes on European Software. The procedure designed for use upon game loading is soon going to be the method required for use in the option menus too. So please use this method under the option DISPLAY MODE.

To change to 60 Hz on PAL software using the Kamui Low Level API, you simply need to call `kmSetDisplayMode()` again, with the equivalent NTSC screen mode (Keeping the Frame buffer type, and other parameters the same). Calling `kmActivateFrameBuffer()` after the display mode change is necessary when using Kamui 1.

Note: When the initial display mode is an EXTENDED PAL mode, switching to a 640*480 NTSC mode still works fine, as the frame buffer requirement is less than for the extended PAL display. As the frame buffers, native buffers and textures are left intact during mode switches.

---

18. Dreamcast SCART Solution

There have been reports of problems running some game code using the SCART (RGB) connector. What happens is that the game crashes after the Sega licence screen, or exits prematurely. This is due to a programming error at initialization, along with a slightly misleading API.

First of all here's the bad code example:

```
cable          =            syCblCheckCable();
switch( cable )
{
case SYE_CBL_CABLE_VGA :
      /* We are running in VGA */
      njInitSystem(NJD_RESOLUTION_VGA, NJD_FRAMEBUFFER_MODE_RGB565, 1);
      break;
case SYE_CBL_CABLE_NTSC :
case SYE_CBL_CABLE_PAL :
      broadcast = syCblCheckBroadcast();
      if (broadcast==SYE_CBL_BROADCAST_NTSC)
            njInitSystem(NJD_RESOLUTION_640x480_NTSCNI,
NJD_FRAMEBUFFER_MODE_RGB565, 1);
      else
            njInitSystem(NJD_RESOLUTION_640x544_PALNI,
NJD_FRAMEBUFFER_MODE_RGB565, 1);
      break;
}
```

This can fail because of the specification of the enumation SYE_CBL_CABLE_XXXX, which is:

```
typedef enum SYE_CBL_CABLE_TAG {
      SYE_CBL_CABLE_VGA = 0,
      SYE_CBL_CABLE_RESERVED = 1,
      SYE_CBL_CABLE_NTSC_RGB = 2,
      SYE_CBL_CABLE_PAL_RGB = 2,
      SYE_CBL_CABLE_RGB = 2,
      SYE_CBL_CABLE_NTSC = 3,
      SYE_CBL_CABLE_PAL = 3,
      SYE_CBL_CABLE_VBS = 3,
```

```
#if 1 /* 1.10 */
        SYE_CBL_CABLE_MAX
#endif
} SYE_CBL_CABLE;
```

If the cable code does not check for SYE_CBL_CABLE_NTSC_RGB, SYE_CBL_CABLE_PAL_RGB or SYE_CBL_CABLE_RGB it can drop through the check without initializing Shinobi. This will obviously have disastrous results if the program is run with RGB cable.

The fix is very easy. You have two options :

1) Modify your code to check for this instance.

```
cable  =                    syCblCheckCable();
switch( cable )
{
case SYE_CBL_CABLE_VGA :
   /* We are running in VGA */
   njInitSystem(NJD_RESOLUTION_VGA, NJD_FRAMEBUFFER_MODE_RGB565, 1);
   break;
case SYE_CBL_CABLE_NTSC :
case SYE_CBL_CABLE_PAL :
case SYE_CBL_CABLE_NTSC_RGB:
case SYE_CBL_CABLE_PAL_RGB:
case SYE_CBL_CABLE_RGB:
   broadcast = syCblCheckBroadcast();
   if (broadcast==SYE_CBL_BROADCAST_NTSC)
      njInitSystem(NJD_RESOLUTION_640x480_NTSCNI, NJD_FRAMEBUFFER_MODE_RGB565,
1);
   else
      njInitSystem(NJD_RESOLUTION_640x544_PALNI, NJD_FRAMEBUFFER_MODE_RGB565, 1);
   break;
default:
   syBtExit();           // I don't recognise this type
}
```

2) Alternative the newer versions of Shinobi library have a much cleaner interface for checking the cable and broadcast type by using the new function.

```
SYE_CBL syCblCheck( void );.

#if 1 /* 1.10 */
typedef enum SYE_CBL_TAG {
        SYE_CBL_NTSC = 0,
        SYE_CBL_VGA = 1,
        SYE_CBL_PAL = 2,
        SYE_CBL_MAX
} SYE_CBL;
#endif
```

For example:

```
        cable  =                                    syCblCable();
        switch( cable )
```

```
{
case SYE_CBL_VGA :
   /* We are running in VGA */
   njInitSystem(NJD_RESOLUTION_VGA, NJD_FRAMEBUFFER_MODE_RGB565, 1);
   break;
case SYE_CBL_NTSC :
   njInitSystem(NJD_RESOLUTION_640x480_NTSCNI, NJD_FRAMEBUFFER_MODE_RGB565, 1);
   break;
case SYE_CBL_PAL :
   njInitSystem(NJD_RESOLUTION_640x544_PALNI, NJD_FRAMEBUFFER_MODE_RGB565, 1);
   break;
default:
   syBtExit();                          // I don't recognise this type
}
```

Sega Dreamcast™

Shinobi Sound Library

1. Data Utility Functionss

sdBankDownload

Transfers a bank.

Format

```
SDE_ERR sdBankDownload( handle, bank_type, bank_num )
SDMEMBLK handle
const SDE_DATA_TYPE bank_type
const Sint8 bank_num
```

Parameters

`handle`	Bank memory block handle
`bank_type`	Bank type
`bank_num`	Destination bank number

Return Value

`SDE_ERR_NOTHING`	No error
`SDE_ERR_NO_INIT`	Sound Library has not been initialized
`SDE_ERR_HANDLE_NULL`	Handle is `NULL`
`SDE_ERR_HANDLE_ILLEGAL_VALUE`	An illegal address is specified for the handle
`SDE_ERR_DATA_ILLEGAL_TYPE`	Illegal bank type

Description

Transfers a bank from main memory to sound memory.

The following values can be specified for the argument `bank_type` which determines the bank type.

Definition	Meaning	
SDE_DATA_TYPE_MIDI_SEQ_BANK	MIDI	Sequence Bank
SDE_DATA_TYPE_MIDI_PRG_BANK	MIDI	Program Bank
SDE_DATA_TYPE_SHOT_BANK	One Shot Bank	
SDE_DATA_TYPE_FX_OUT_BANK	FX Output Bank	
SDE_DATA_TYPE_FX_PRG_BANK	FX Program Bank	

Before using this function, a Multi Unit transfer to sound memory, and memory mapping must be carried out.

The Multi Unit transfer can be achieved using the `sdMultiUnitDownload()` or `sdMultiUnitDownloadFromFile()` function.

If the error type is `SDE_ERR_BANK_ILLEGAL_TYPE` (illegal bank type), an attempt may have been made to transfer a bank containing a Bank ID not recognized by the library. If it is necessary to determine whether the transfer has actually taken place, set the memory block transfer mode to `SDD_MEMBLK_SYNC_FUNC`.

Example

```
Void BankLoad( char *file_name, SDE_DATA_TYPE bank_type, Sint8 bank_num)
{
    SDMEMBLK memblk = NULL;
    GDFS gdfs = NULL;
    Void *bank_ptr;
    Sint32 bank_sz;

    /*
     * Load Bank File into memory
     */
    gdfs = gdFsOpen( SMPD_PSTM_R_SRC_FILE_NAME, NULL);

    /* Get Bank File size */
    gdFsGetFileSize( gdfs, &bank_sz);

    /* Get buffer to read */
    /* Allocate enough amount of memory to support loading from GD */
    bank_ptr = syMalloc( gSrcWaveSz + GDD_FS_SCTSIZE);

    /* Load Bank File */
    gdFsRead( gdfs, gdFsCalcSctSize( bank_sz), bank_ptr);

    gdFsClose( gdfs);

    /*
     * Transfer sound memory
```

```
     */
    sdMemBlkCreate( &memblk);
    sdMemBlkSetPrm( memblk, bank_ptr, bank_sz, SDD_MEMBLK_SYNC_FUNC, NULL);
    sdBankDownload( memblk, bank_type, bank_num);
    sdMemBlkDestroy( memblk);
}
```

Reference

`sdBankDownloadFromFile()`	Reads in and transfers bank file
`sdMultiUnitDownload()`	Transfers a multi-unit file
`sdMultiUnitDownloadFromFile()`	Reads and transfers a multi-unit file

sdBankDownloadFromFile

Reads in and transfers bank file.

Format

```
SDE_ERR sdBankDownloadFromFile( file_name, bank_type, bank_num, work_memblk )
char *file_name
const SDE_DATA_TYPE bank_type
const Sint8 bank_num
SDMEMBLK memblk
```

Parameters

file_name	Bank file name
bank_type	Bank type
bank_num	Destination bank number
work_memblk	Work memory block handle

Return Value

SDE_ERR_NOTHING	No error
SDE_ERR_NO_INIT	Sound Library has not been initialized
SDE_ERR_HANDLE_NULL	Handle is NULL
SDE_ERR_HANDLE_ILLEGAL_VALUE	An illegal address is specified for the handle
SDE_ERR_BANK_ILLEGAL_TYPE	Illegal bank type

Description

Reads a bank file from GD-ROM into main memory, and transfers it to sound memory.

The following values can be specified for the argument bank_type which determines the bank type.

Definition	Meaning
SDE_DATA_TYPE_MIDI_SEQ_BANK	MIDI Sequence Bank
SDE_DATA_TYPE_MIDI_PRG_BANK	MIDI Program Bank
SDE_DATA_TYPE_SHOT_BANK	One Shot Bank
SDE_DATA_TYPE_FX_OUT_BANK	FX Output Bank
SDE_DATA_TYPE_FX_PRG_BANK	FX Program Bank

Before using this function, a Multi Unit transfer to sound memory, and memory mapping must be carried out.

The Multi Unit transfer can be achieved using the `sdMultiUnitDownload()` or `sdMultiUnitDownloadFromFile()` function. If the error type is `SDE_ERR_BANK_ILLEGAL_TYPE` (illegal bank type), an attempt may have been made to transfer a bank containing a Bank ID not recognized by the library. A work buffer of at least 800H bytes is required, and may be allocated in 800H units.

Example

```
#defineSMPD_BUF_SZ (0x2000)
#defineSMPD_BANK_TYPE ( SDD_DATA_TYPE_MIDI_SEQ_BANK)
#defineSMPD_BANK_NUM (0x00)

Void *buf_ptr;
SDMEMBLK memblk;

/*
 *     Allocate buffer to read and set memory block handle
 */
buf_ptr = syMalloc( SMPD_BUF_SZ);
/* Get memory block handle */
sdMemBlkCreate( &memblk);
/* Set buffer in memory block  */
sdMemBlkSetPrm( memblk, buf_ptr, SMPD_BUF_SZ, SDD_MEMBLK_SYNC_FUNC, NULL);

/*
 *     Read Bank File
 */
/* Load file */
sdBankDownloadFromFile( "sample.msb", SMPD_BANK_TYPE, SMPD_BANK_NUM, memblk);
/* Destoy memory block handle */
sdMemBlkDestroy( memblk);
/* Destroy work buffer  */
syFree( buf_ptr);
```

Note

Do not use at the same time as the Middleware Library functions.

Reference

`sdMultiUnitDownload()`	Transfers a multi-unit file
`sdMultiUnitDownloadFromFile()`	Reads and transfers a multi-unit file

sdMultiUnitDownload

Transfers a multi-unit file.

Format

```
SDE_ERR sdMultiUnitDownload( handle )
SDMEMBLK handle
```

Parameters

handle	Memory block handle

Return Value

SDE_ERR_NOTHING	No error
SDE_ERR_NO_INIT	Sound Library has not been initialized
SDE_ERR_HANDLE_NULL	Handle is NULL
SDE_ERR_HANDLE_ILLEGAL_VALUE	An illegal address is specified for the handle
SDE_ERR_DATA_ILLEGAL_TYPE	Illegal bank type

Description

Transfers a multi-unit from main memory to sound memory.

Note

If the error type is SDE_ERR_DATA_ILLEGAL_TYPE (illegal data type), a buffer may have been specified which has a file not of multi-unit type read in, or an attempt may have been made to download a multi-unit containing a bank ID or data ID not recognized by the library.

sdMultiUnitDownloadFromFile

Reads and transfers a multi-unit file.

Format

```
SDE_ERR sdMultiUnitDownloadFromFile( file_name, work_memblk )

char *file_name
SDMEMBLK memblk
```

Parameters

file_name	Multi-unit file name
work_memblk	Work memory block handle

Return Value

SDE_ERR_NOTHING	No error
SDE_ERR_NO_INIT	Sound Library has not been initialized
SDE_ERR_HANDLE_NULL	Handle is NULL
SDE_ERR_HANDLE_ILLEGAL_VALUE	An illegal address is specified for the handle
SDE_ERR_DATA_ILLEGAL_TYPE	Data type is illegal (Data (file) may be corrupted)

Description

Reads a multi-unit file from GD-ROM to main memory, and transfers it to sound memory.

A work buffer of at least 800H bytes is required, and may be allocated in 800H units.

Example

```
Void    *buf_ptr;
SDMEMBLK        memblk;
buf_ptr = syMalloc( 0x800);
sdMemBlkCreate( &memblk);
sdMemBlkSetPrm( memblk, buf_ptr, 0x800, SDD_MEMBLK_SYNC_FUNC, NULL);
sdMultiUnitDownloadFromFile( "sample.mlt", memblk );
sdMemBlkDestroy( memblk);
syFree( buf_ptr);
```

Note

In the current version (Ver. 2.0), this function only supports return after completion. Do not use at the same time as the Middleware Library functions.

2. Global Control Functions

sdQsndSetPos

Sets Q Sound position.

Format

```
SDE_ERR sdQsndSetPos( pos )
const Sint8 *pos
```

Parameters

pos	Pointer to array indicating Q position

Return Value

SDE_ERR_NOTHING	No error
SDE_ERR_NO_INIT	Sound Library has not been initialized
SDE_ERR_HOST_CMD_BUF_NO_ENOUGH	Host command buffer overflow
SDE_ERR_PRM_OVER_RANGE	Out-of-range parameter value

Description

Sets Q Sound position.

The argument pos indicating the position is a pointer to an array of 8 bytes, which are 8-bit signed integer values; the array index corresponds to the Q Sound module number. The array size is determined by the Q Sound type.

Type	Size (range)
Q Sound4	4 (0...3)
Q Sound8	8(0...7)

The array values are relative values from the created sound data values, and the range is from 00H (right edge) to 40H (center) to 7FH (right edge).

sdSndClearFxPrg

Clears FX program.

Format

```
SDE_ERR sdSndClearFxPrg( Void )
```

Parameters

None

Return Value

SDE_ERR_NOTHING	No error
SDE_ERR_NO_INIT	Sound Library has not been initialized
SDE_ERR_HOST_CMD_BUF_NO_ENOUGH	Host command buffer overflow

Description

Clears the FX program currently being used.

Note

This function puts a heavy load on the sound driver.

For approximately 20 ms after executing this function, the sound driver is unable to carry out any other processing.

Execute this function when there is no sound output. Calling it while sound is being output may result in noise.

sdSndGetFxOut

Gets current FX output data number.

Format

```
SDE_ERR sdSndGetFxOut( cur_fx_out_num )
Sint8 *cur_fx_out_num
```

Parameters

`cur_fx_out_num`	Pointer to storage for FX output data number

Return Value

`SDE_ERR_NOTHING`	No error
`SDE_ERR_NO_INIT`	Sound Library has not been initialized

Description

Gets the current FX output data number.

Example

```
Sint8 cur_fx_out;
sdSndSetFxPrg( &cur_fx_out);
```

sdSndGetFxPrg

Gets current FX program data number.

Format

```
SDE_ERR sdSndGetFxPrg( cur_fx_prg_num )
Sint8 *cur_fx_prg_num
```

Parameters

`cur_fx_prg_num`	Pointer to current FX program data number

Return Value

`SDE_ERR_NOTHING`	No error
`SDE_ERR_NO_INIT`	Sound Library has not been initialized

Description

Gets the current FX program data number.

Example

```
Sint8 cur_fx_prg;
sdSndSetFxPrg( &cur_fx_prg);
```

sdSndSetFxOut

Sets FX output data.

Format

```
SDE_ERR sdSndSetFxOut( new_fx_out_num )
Sint8 const new_fx_out_num
```

Parameters

new_fx_out_num	FX output data number

Return Value

SDE_ERR_NOTHING	No error
SDE_ERR_NO_INIT	Sound Library has not been initialized
SDE_ERR_HOST_CMD_BUF_NO_ENOUGH	Host command buffer overflow
SDE_ERR_PRM_OVER_RANGE	Out-of-range parameter value

Description

Specifies FX output data number to use.

sdSndSetFxPrg

Sets FX program data.

Format

```
SDE_ERR sdSndSetFxPrg( new_fx_prg_num, new_fx_out_num )
const Sint8 new_fx_prg_num
const Sint8 new_fx_out_num
```

Parameters

new_fx_prg_num	FX program data number
new_fx_out_num	FX output data number

Return Value

SDE_ERR_NOTHING	No error
SDE_ERR_NO_INIT	Sound Library has not been initialized
SDE_ERR_HOST_CMD_BUF_NO_ENOUGH	Host command buffer overflow
SDE_ERR_PRM_OVER_RANGE	Out-of-range parameter value

Description

Specifies the FX program data to be used and the FX output data.

The range of values, both specifying the FX program data number and FX output data number, is from 00H to 7FH. However, it is not possible to exceed the number of data values set in the sound data.

Example

```
sdSndSetFxPrg( 0, 0);
```

Note

This function puts a heavy load on the sound driver. For approximately 20 ms after executing this function, the sound driver is unable to carry out any other processing.

Execute this function when there is no sound output. Calling it while sound is being output may result in noise.

sdSndSetMasterVol

Sets the master volume.

Format

```
SDE_ERR sdSndSetMasterVol( master_vol )
const Sint8 master_vol
```

Parameters

master_vol	Master volume value

Return Value

SDE_ERR_NOTHING	No error
SDE_ERR_NO_INIT	Sound Library has not been initialized
SDE_ERR_PRM_OVER_RANGE	Host command buffer overflow
SDE_ERR_HOST_CMD_BUF_NO_ENOUGH	Out-of-range parameter value

Description

Sets the master volume value. The setting range is from 0 to 15. The default value is 15.

Note

This affects the playback volume of all sound.

sdSndSetPanMode

Sets the pan mode.

Format

```
SDE_ERR sdSndSetPanMode( pan_mode )
const SDE_PAN_MODE pan_mode
```

Parameters

`pan_mode` Pan mode

Return Value

`SDE_ERR_NOTHING`	No error
`SDE_ERR_NO_INIT`	Sound Library has not been initialized
`SDE_ERR_PRM_OVER_RANGE`	Host command buffer overflow
`SDE_ERR_HOST_CMD_BUF_NO_ENOUGH`	Out-of-range parameter value

Description

Sets the pan mode. The values of the argument pan_mode are as follows.

Definition	Meaning
SDE_PAN_MODE_MONO	Monoral sound
SDE_PAN_MODE_STEREO	Stereo sound

Example

```
sdSndSetPanMode( SDE_SPACE_MONO);
sdSndSetPanMode( SDE_SPACE_STEREO);
```

Note

Note that the playback balance may change between monaural and stereo.

sdSndStopAll

Stops all sound data playback.

Format

```
SDE_ERR sdSndStopAll( Void )
```

Parameters

None

Return Value

SDE_ERR_NOTHING	No error
SDE_ERR_NO_INIT	Sound Library has not been initialized
SDE_ERR_HOST_CMD_BUF_NO_ENOUGH	Host command buffer overflow

Description

Stops the playback of all sound currently being played.

Note

When playback is stopped using this function, it cannot be resumed with the `sdMidiContinue()` function.

3. Memory Block Transfer Functions

sdSndStopAll

Stops all sound data playback.

Format

```
SDE_ERR sdSndStopAll( Void )
```

Parameters

None

Return Value

SDE_ERR_NOTHING	No error
SDE_ERR_NO_INIT	Sound Library has not been initialized
SDE_ERR_HOST_CMD_BUF_NO_ENOUGH	Host command buffer overflow

Description

Stops the playback of all sound currently being played.

Note

When playback is stopped using this function, it cannot be resumed with the `sdMidiContinue()` function.

sdMemBlkDestroy

Destroys a Memory Block handle.

Format

SDE_ERR sdMemBlkDestroy(handle)
SDMEMBLK handle

Parameters

`handle` Memory Block handle

Return Value

`SDE_ERR_NOTHING` No error
`SDE_ERR_NO_INIT` Sound Library has not been initialized
`SDE_ERR_HANDLE_NULL` Handle is NULL
`SDE_ERR_HANDLE_ILLEGAL_VALUE` An illegal address was specified for the handle

Description

Destroys a Memory Block handle.

sdMemBlkGetStat

Gets the status of a Memory Block.

Format

```
SDE_ERR sdMemBlkGetStat( handle, stat )
SDMEMBLK handle
SDE_MEMBLK_STAT *stat
```

Parameters

`handle`	Memory Block handle
`stat`	Pointer to storage for status information

Return Value

`SDE_ERR_NOTHING`	No error
`SDE_ERR_NO_INIT`	Sound Library has not been initialized
`SDE_ERR_HANDLE_NULL`	Handle is `NULL`
`SDE_ERR_HANDLE_ILLEGAL_VALUE`	An illegal address was specified for the handle

Description

Checks the status of the Memory Block handle.

The argument stat is set to one of the following values:

Definition	Meaning
`SDE_MEMBLK_STAT_TRANSFER_REMAIN`	Not yet transferred
`SDE_MEMBLK_STAT_TRANSFER_FINISHED`	Transferred
`SDE_MEMBLK_STAT_TRANSFER_PROGRESS`	In transfer

Note

Even if the `sdMemBlkTransfer()` is used to specify a transfer, when the status is `SDE_MEMBLK_STAT_TRANSFER_REMAIN` the memory block may still be in the queue.

sdMemBlkSetPrm Sets the parameters for a Memory Block handle.

Format

```
SDE_ERR sdMemBlkSetPrm( handle, ptr, blk_sz, cb_func,
cb_1st_arg )
SDMEMBLK handle
const Void *ptr
const Sint32 blk_sz
const SD_MEMBLK_CALLBACK_FUNC cb_func
const Void *cb_1st_arg
```

Parameters

handle	Memory Block handle
ptr	Pointer to first Memory Block to transfer
blk_sz	Block size
cb_func	Callback function
cb_1st_arg	First argument to be passed to the callback function

Return Value

SDE_ERR_NOTHING	No error
SDE_ERR_NO_INIT	Sound Library has not been initialized
SDE_ERR_HANDLE_NULL	Handle is NULL
SDE_ERR_HANDLE_ILLEGAL_VALUE	An illegal address is specified for the handle

Description

Sets the parameters for a Memory Block handle.

The following values can be specified for the argument cb_func, which registers the callback function.

Definition	Meaning
SDD_MEMBLK_NO_FUNC	Do not register the callback function.
SDD_MEMBLK_SYNC_FUNC	Use the callback function for synchronizing the internal library.
Other ...	The function address specified here is registered.

Simply specifying SDD_MEMBLK_SYNC_FUNC establishes a synchronous mode. If the first argument to the callback function, cb_1st_arg, is not required, set this to NULL.

sdMemBlkSetTransferMode

Sets the transfer mode for Memory Blocks.

Format

```
SDE_ERR sdMemBlkSetTransferMode( transfer_mode )
SDE_MEMBLK_TRANSFER_MODE transfer_mode
```

Parameters

`transfer_mode`	Transfer mode

Return Value

`SDE_ERR_NOTHING`	No error
`SDE_ERR_NO_INIT`	Sound Library has not been initialized

Description

Sets the transfer method for Memory Blocks.

The following values can be specified for the parameter transfer_mode.

Definition	Meaning
SDE_MEMBLK_TRANSFER_MODE_CPU	CPU transfer
SDE_MEMBLK_TRANSFER_MODE_DMA	DMA transfer

Note

From Sound Libray Ver. 2.0, the default is DMA transfer. In Ver. 1.0 the default was CPU transfer, and therefore if the `sdMemBlkSetPrm()` function specifies a particular memory block callback function, it is important to note that the return timing from the callback function is different. (In CPU mode the return is after function completion; in DMA mode return is immediate.)

Reference

`sdMemBlkSetPrm()`	Sets the parameters for a Memory Block handle

4. Memory Control Functions

sdSndMemGetBankStat

Gets the bank status of Sound Memory.

Format

```
SDE_ERR sdSndMemGetBankStat( bank_type, bank_num, adr, sz )

const SDE_DATA_TYPE bank_type
const Sint8 bank_num
Sint32 *adr
Sint32 *sz
```

Parameters

bank_type	Bank type
bank_num	Bank number
adr	Pointer to get the starting address
sz	Pointer to get the size of bank

Return Value

SDE_ERR_NOTHING	No error
SDE_ERR_NO_INIT	Sound Library has not been initialized
SDE_ERR_DATA_ILLEGAL_TYPE	Data type is illegal (data (file) may be corrupted)

Description

Gets the bank status of Sound Memory.

The argument `bank_type` which specifies the bank type to download can have the following values.

Definition	Meaning
SDE_DATA_TYPE_MIDI_DRUM_BANK	MIDI Drum Bank
SDE_DATA_TYPE_MIDI_SEQ_BANK	MIDI Sequence Bank
SDE_DATA_TYPE_MIDI_PRG_BANK	MIDI Program Bank
SDE_DATA_TYPE_SHOT_BANK	One Shot Bank
SDE_DATA_TYPE_PSTM_RING_BUF	PCM Stream Ring Buffer
SDE_DATA_TYPE_FX_OUT_BANK	FX Output Bank
SDE_DATA_TYPE_FX_PRG_BANK	FX Program Bank
SDE_DATA_TYPE_FX_PRG_WRK	FX Program Work

Example

```
Sint32 adr;
Sint32 sz;
sdSndMemGetBankStat( SDE_DATA_TYPE_PSTM_RING_BUF, 0, &adr, &sz);
```

5. PCM Stream Module Control

sdGddaGetStat

Gets the GD-DA port status.

Format

```
SDE_ERR sdGddaGetStat( stat )
SDS_GDDA_STAT *stat
```

Parameters

`stat`	Pointer to storage for status information

Return Value

`SDE_ERR_NOTHING`	No error
`SDE_ERR_NO_INIT`	Sound Library has not been initialized

Description

Gets GD-DA port status.

sdGddaResetPrm

Resets GD-DA port parameters.

Format

```
SDE_ERR sdGddaResetPrm( Void )
```

Parameters

None

Return Value

SDE_ERR_NOTHING	No error
SDE_ERR_NO_INIT	Sound Library has not been initialized
SDE_ERR_HOST_CMD_BUF_NO_ENOUGH	Host command buffer overflow

Description

Resets GD-DA port parameter levels.

sdGddaSetPan

Sets the GD-DA port panpot levels.

Format

```
SDE_ERR sdGddaSetPan( left_pan, right_pan )
const Sint8 left_pan
const Sint8 right_pan
```

Parameters

`left_pan`	Panpot value for left channel
`right_pan`	Panpot value for right channel

Return Value

`SDE_ERR_NOTHING`	No error
`SDE_ERR_NO_INIT`	Sound Library has not been initialized
`SDE_ERR_HOST_CMD_BUF_NO_ENOUGH`	Host command buffer overflow

Description

Sets GD-DA port panpot levels.

The value range that can be set for each parameter is: 0080H (left) to 0000H (center) to 007FH (right).

sdGddaSetVol

Sets the GD-DA port volume.

Format

```
SDE_ERR sdGddaSetVol( left_vol, right_vol )
const Sint8 left_vol
const Sint8 right_vol
```

Parameters

`left_vol`	Left side volume level
`right_vol`	Right side volume level

Return Value

`SDE_ERR_NOTHING`	No error
`SDE_ERR_NO_INIT`	Sound Library has not been initialized
`SDE_ERR_HOST_CMD_BUF_NO_ENOUGH`	Host command buffer overflow

Description

Sets the GD-DA port volume level.

6. GD-DA Module Control

sdMidiClosePort

Releases MIDI port access permission.

Format

```
SDE_ERR sdMidiClosePort( handle )
SDMIDI handle
```

Parameters

handle	MIDI port handle

Return Value

SDE_ERR_NOTHING	No error
SDE_ERR_NO_INIT	Sound Library has not been initialized
SDE_ERR_HANDLE_NULL	Handle is NULL

Description

Releases the specified MIDI port access permission.

Example

```
SDMIDI midi_handle;
sdMidiOpenPort( &midi_handle);
sdMidiClosePort( midi_handle);
```

Note

Frequently getting and releasing access permission puts a heavy load on the CPU. As far as possible, pass the handle around. For example, if the handle is obtained when initializing a stage and released when ending the stage, this will keep the CPU load to an acceptable level.

sdMidiContinue

Resumes playback of a paused MIDI sequence.

Format

```
SDE_ERR sdMidiContinue( handle )
SDMIDI handle
```

Parameters

handle	MIDI port handle

Return Value

SDE_ERR_NOTHING	No error
SDE_ERR_NO_INIT	Sound Library has not been initialized
SDE_ERR_HANDLE_NULL	Handle is NULL
SDE_ERR_HOST_CMD_BUF_NO_ENOUGH	Host command buffer overflow

Description

Resumes playback of the MIDI sequence which has been paused on the specified MIDI port.

Example

```
SDMIDI midi_handle;
/* Get access permission to MIDI port */
sdMidiOpenPort( &midi_handle);
/* Play MIDI sequence from bank 00H, data value 00H, with priority 00H  */
sdMidiPlay( midi_handle, 0, 0, 0);
/* Pause MIDI sequence */
sdMidiPause( midi_handle);
/* Resume paused MIDI sequence */
sdMidiResume( midi_handle);
/* Stop MIDI sequence playback */
sdMidiStop( midi_handle);
/* Release MIDI port access permission */
sdMidiClosePort( midi_handle);
```

Note

Use the `sdMidiPause()` function to pause the MIDI sequence playback.

Reference

sdMidiPause()	Pauses MIDI sequence playback

sdMidiGetCurAdr Gets the current play address of a MIDI sequence.

Format

```
SDE_ERR sdMidiGetCurAdr( handle, cur_adr )
SDMIDI handle
Sint32 *cur_adr
```

Parameters

handle	MIDI port handle
cur_adr	Playback address

Return Value

SDE_ERR_NOTHING	No error
SDE_ERR_NO_INIT	Sound Library has not been initialized
SDE_ERR_HANDLE_NULL	Handle is NULL

Description

Gets the current playback address of a MIDI sequence.

sdMidiGetStat

Gets MIDI port status.

Format

```
SDE_ERR sdMidiGetStat( handle, stat )
SDMIDI handle
SDS_MIDI_STAT *stat
```

Parameters

`handle`	MIDI port handle
`stat`	Pointer to storage for MIDI port status

Return Value

`SDE_ERR_NOTHING`	No error
`SDE_ERR_NO_INIT`	Sound Library has not been initialized
`SDE_ERR_HANDLE_NULL`	Handle is `NULL`

Description

Gets the status of the specified MIDI port.

sdMidiGetTotalBeatTime

Gets the current playing beat time of a MIDI sequence.

Format

```
SDE_ERR sdMidiGetTotalBeatTime( handle, total_beat_time )
SDMIDI handle
Uint32 *total_beat_time
```

Parameters

`handle`	MIDI port handle
`total_beat_time`	Pointer to storage for current playing beat number

Return Value

`SDE_ERR_NOTHING`	No error
`SDE_ERR_NO_INIT`	Sound Library has not been initialized
`SDE_ERR_HANDLE_NULL`	Handle is `NULL`

Description

Gets the current playing beat time of the MIDI sequence being played from the specified MIDI port. One beat is calculated as a quarter note.

Note

The beat count returns to zero when the MIDI sequence data is played.

sdMidiOpenPort

Gets MIDI port access permission.

Format

```
SDE_ERR sdMidiOpenPort( handle )
SDMIDI *handle
```

Parameters

handle	Pointer to storage for the port handle

Return Value

SDE_ERR_NOTHING	No error
SDE_ERR_NO_INIT	Sound Library has not been initialized
SDE_ERR_HANDLE_NO_ENOUGH	

Description

Gets MIDI port access permission.

Note

Frequently getting and releasing access permission puts a heavy load on the CPU. As far as possible, pass the handle around. For example, if the handle is obtained when initializing a stage and released when ending the stage, this will keep the CPU load to an acceptable level.

sdMidiPause

Pauses MIDI sequence playback.

Format

```
SDE_ERR sdMidiPause( handle )
SDMIDI handle
```

Parameters

handle	MIDI port handle

Return Value

SDE_ERR_NOTHING	No error
SDE_ERR_NO_INIT	Sound Library has not been initialized
SDE_ERR_HANDLE_NULL	Handle is NULL
SDE_ERR_HOST_CMD_BUF_NO_ENOUGH	Host command buffer overflow

Description

Pauses playback of the MIDI sequence from the specified MIDI port.

sdMidiPlay

Plays a MIDI sequence.

Format

```
SDE_ERR sdMidiPlay( handle, bank_num, list_num, priority )
SDMIDI handle
const Sint8 bank_num
const Sint8 list_num
const Sint8 priority
```

Parameters

`handle`	MIDI port handle
`bank_num`	MIDI sequence bank number
`list_num`	MIDI sequence data number
`priority`	Playback priority

Return Value

`SDE_ERR_NOTHING`	No error
`SDE_ERR_NO_INIT`	Sound Library has not been initialized
`SDE_ERR_HANDLE_NULL`	Handle is `NULL`
`SDE_ERR_PRM_OVER_RANGE`	Out-of-range parameter value
`SDE_ERR_HOST_CMD_BUF_NO_ENOUGH`	Host command buffer overflow

Description

Plays the MIDI sequence data from the specified MIDI port.

The argument `bank_num` specifying the MIDI sequence bank number is in the range 00H to 1FH.

The argument `list_num` specifying the MIDI sequence data value number is in the range 0000H to 007FH.

The range of `priority`, which specifies the priority level, is 0000H (always take priority) to 0001H (lowest) to 000FH (highest).

sdMidiResetAllPrm

Resets parameters of all MIDI ports.

Format

```
SDE_ERR sdMidiResetAllPrm( Void )
```

Parameters

None

Return Value

SDE_ERR_NOTHING	No error
SDE_ERR_NO_INIT	Sound Library has not been initialized
SDE_ERR_HOST_CMD_BUF_NO_ENOUGH	Host command buffer overflow

Description

Resets the level parameters for all MIDI ports.

Note

All MIDI port settings other than playback speed are reset to the center value (0).

sdMidiResetPrm

Resets a MIDI port.

Format

```
SDE_ERR sdMidiResetPrm( handle )
SDMIDI handle
```

Parameters

handle	MIDI port handle

Return Value

SDE_ERR_NOTHING	No error
SDE_ERR_NO_INIT	Sound Library has not been initialized
SDE_ERR_HANDLE_NULL	Handle is NULL
SDE_ERR_HOST_CMD_BUF_NO_ENOUGH	Host command buffer overflow

Description

Resets the parameters for the specified MIDI port.

Note

All MIDI port settings other than playback speed are reset to the center value (0).

sdMidiSendMes

Sends a MIDI message to a MIDI port.

Format

```
SDE_ERR sdMidiSendMes( handle, midi_mes_ptr, priority )
SDMIDI handle
const SDS_MIDI_MES *midi_mes_ptr
const Sint8 priority
```

Parameters

`handle`	MIDI port handle
`midi_mes_ptr`	Pointer to MIDI message
`priority`	Priority level

Return Value

`SDE_ERR_NOTHING`	No error
`SDE_ERR_NO_INIT`	Sound Library has not been initialized
`SDE_ERR_HANDLE_NULL`	Handle is `NULL`
`SDE_ERR_HOST_CMD_BUF_NO_ENOUGH`	Host command buffer overflow

Description

Sends a MIDI message to the specified MIDI port.

The range of priority, which marks the priority level of MIDI messages, is 0000H (always take priority), to 0001H (lowest) to 000FH (highest).

Example

```
Void NoteOn( SDMIDI handle, Sint8 note_num, Sin8 velocity)
{
    SDS_MIDI_MES midi_mes;

    sdMidiSetMes( handle, &midi_mes, note_num, velocity);
    sdMidiSendMes( handle, &midi_mes);
}
```

Note

Please use the same MIDI Handle as the one used for `sdMidiSet()` function to set MIDI Event.

Reference

`sdMidiSetMes()`	Creates a MIDI message to send to a MIDI port

sdMidiSetDrctLev

Sets the direct level of a MIDI port.

Format

```
SDE_ERR sdMidiSetDrctLev( handle, drct_lev )
SDMIDI handle
const Sint8 drct_lev
```

Parameters

`handle`	MIDI port handle
`drct_lev`	Direct level

Return Value

`SDE_ERR_NOTHING`	No error
`SDE_ERR_NO_INIT`	Sound Library has not been initialized
`SDE_ERR_HANDLE_NULL`	Handle is `NULL`
`SDE_ERR_PRM_OVER_RANGE`	Out-of-range parameter value
`SDE_ERR_HOST_CMD_BUF_NO_ENOUGH`	Host command buffer overflow

Description

Sets the direct level (the sound level other than for effects) of the specified MIDI port.

The value range of `drct_lev`, which marks direct levels, is -007FH (minimum) to 0000H (standard) to 007FH (maximum).

The argument `drct_lev` is a relative value with respect to the direct level of the created data. In other words, to return the data to its original direct level, set `drct_lev` to 0.

sdMidiSetFxLev

Sets the FX level of a MIDI port.

Format

```
SDE_ERR sdMidiSetFxLev( handle, fx_lev )
SDMIDI handle
const Sint8 fx_lev
```

Parameters

handle	MIDI port handle
fx_lev	FX level

Return Value

SDE_ERR_NOTHING	No error
SDE_ERR_NO_INIT	Sound Library has not been initialized
SDE_ERR_HANDLE_NULL	Handle is NULL
SDE_ERR_PRM_OVER_RANGE	Out-of-range parameter value
SDE_ERR_HOST_CMD_BUF_NO_ENOUGH	Host command buffer overflow

Description

Sets the FX level of the specified MIDI port.

The value range of fx_lev, which specifies FX levels, is -007FH (minimum) to 0000H (standard) to 007FH (maximum).

The argument fx_lev is a relative level with respect to the FX level of the created data.

In other words, to return the data to its original FX level, set fx_lev to 0.

sdMidiSetMes

Creates a MIDI message to send to a MIDI port.

Format

```
SDE_ERR sdMidiSetMes( handle, midi_mes_ptr, midi_mes, ... )
SDMIDI handle
SDS_MIDI_MES *midi_mes_ptr
const Uint8 midi_mes
...
```

Parameters

`handle`	MIDI port handle
`midi_mes_ptr`	Pointer to storage for MIDI message
`midi_mes`	MIDI message
`...`	

Return Value

`SDE_ERR_NOTHING`	No error
`SDE_ERR_NO_INIT`	Sound Library has not been initialized
`SDE_ERR_HANDLE_NULL`	Handle is `NULL`

Description

Creates a MIDI message to send to a MIDI port.

The variable length parameter allows a sequence of any number of message items to be written as required for the MIDI message. MIDI message items that can be set are shown below.

Definition	Meaning
`SDD_MIDI_NOTE_OFF`	MIDI note off
`SDD_MIDI_NOTE_ON`	MIDI note on
`SDD_MIDI_MONO_PRES`	MIDI monopressure
`SDD_MIDI_PRG_CHG`	MIDI program change
`SDD_MIDI_CTL_CHG`	MIDI control change
`SDD_MIDI_POLY_PRES`	MIDI polypressure
`SDD_MIDI_PITCH_BEND`	MIDI pitch bend

The following macros can be used for MIDI control changes.

Definition	Meaning
SDD_MIDI_CTL_CHG_LSB	Offset the LSB of the MIDI control number
SDD_MIDI_CTL_CHG_MSB	Offset the MSB of the MIDI control number
SDD_MIDI_CTL_CHG_BANK	Bank change
SDD_MIDI_CTL_CHG_MODULATION	Modulation
SDD_MIDI_CTL_CHG_VOL	Volume
SDD_MIDI_CTL_CHG_PAN	Panpot
SDD_MIDI_CTL_CHG_EXPRESSION	Expression

In the SEGA library environment, SDD_MIDI_CTL_CHG_BANK must be ORed with SDD_MIDI_CTL_CHG_MSB. This function simply creates the MIDI message. To send the message use the sdMidiSendMes() function.

Example

```
SDMIDI midi_handle;
SDS_MIDI_MES midi_mes;
Sint8 midi_ch;
midi_ch = 0;
sdMidiOpenPort( &midi_handle);
/* Set MIDI Program Bank Number */
sdMidiSetMes( midi_handle, &midi_mes, SDD_MIDI_MES_CTL_CHG | midi_ch,
SDD_MIDI_CTL_CHG_MSB | SDD_MIDI_CTL_CHG_BANK, 0);
sdMidiSendMes( midi_handle, &midi_mes, 0);
/* Set MIDI Volume */
sdMidiSetMes( midi_handle, &midi_mes, SDD_MIDI_MES_CTL_CHG | midi_ch,
SDD_MIDI_CTL_CHG_VOL, 127);
sdMidiSendMes( midi_handle, &midi_mes, 0);
/* Set MIDI Panpot */
sdMidiSetMes( midi_handle, &midi_mes, SDD_MIDI_MES_CTL_CHG | midi_ch,
SDD_MIDI_CTL_CHG_PAN, 64);
sdMidiSendMes( midi_handle, &midi_mes, 0);
/* Set MIDI Program Data Number */
sdMidiSetMes( midi_handle, &midi_mes, SDD_MIDI_MES_PRG_CHG | midi_ch, 0x00);
sdMidiSendMes( midi_handle, &midi_mes, 0);
/* Set MIDI Note On */
sdMidiSetMes( midi_handle, &midi_mes, SDD_MIDI_MES_NOTE_ON | midi_ch, 60, 127);
sdMidiSendMes( midi_handle, &midi_mes, 0);
```

Reference

sdMidiSendMes()	Sends a MIDI message to a MIDI port

sdMidiSetPan

Sets the panpot of a MIDI port.

Format

```
SDE_ERR sdMidiSetPan( handle, pan, fade_time )
SDMIDI handle
const Sint8 pan
const Sint32 fade_time
```

Parameters

handle	MIDI port handle
pan	Panpot value
fade_time	Time to reach target panpot value

Return Value

SDE_ERRNOTHING	No error
SDE_ERR_NO_INIT	Sound Library has not been initialized
SDE_ERR_HANDLE_NULL	Handle is NULL
SDE_ERR_PRM_OVER_RANGE	Out-of-range parameter value
SDE_ERR_HOST_CMD_BUF_NO_ENOUGH	Host command buffer overflow

Description

Sets the playback panpot for the specified MIDI port.

The value range of pan, which indicates the panpot setting, is -007FH (left) to 0000H (center) to 007FH (right).

The argument pan is a relative value with respect to the panpot balance of the created data. In other words, to return the data to its original panpot setting, set pan to 0.

The units of fade_time, which indicates the time to reach play speed, are in milliseconds from 0000H (fastest) to 7FFFH (slowest).

During the interval until this playback speed is reached, the function `sdSysServer()` must be called at each VSync.

sdMidiSetPitch

Sets the playback pitch of a MIDI port.

Format

```
SDE_ERR sdMidiSetPitch( handle, pitch, fade_time )
SDMIDI handle
const Sint16 pitch
const Sint32 fade_time
```

Parameters

handle	MIDI port handle
pitch	Playback pitch
fade_time	Time to reach target playback pitch

Return Value

SDE_ERR_NOTHING	No error
SDE_ERR_NO_INIT	Sound Library has not been initialized
SDE_ERR_HANDLE_NULL	Handle is NULL
SDE_ERR_PRM_OVER_RANGE	Out-of-range parameter value
SDE_ERR_HOST_CMD_BUF_NO_ENOUGH	Host command buffer overflow

Description

Sets the playback pitch of the specified MIDI port.

The argument pitch which indicates the playback pitch is 0100H for a semitone up, -0100H for a semitone down.

The argument pitch is a relative value with respect to the pitch setting of the created data. In other words, to return the data to its original pitch setting, set pitch to 0.

The units of fade_time, which indicates the time to reach the pitch setting, are in milliseconds from 0000H (fastest) to 7FFFH (slowest).

During the interval until this pitch is reached, the function sdSysServer() must be called at each VSync.

sdMidiSetSpeed

Sets the playback speed of a MIDI port.

Format

```
SDE_ERR sdMidiSetSpeed( handle, speed, fade_time )
SDMIDI handle
const Sint16 speed
const Sint32 fade_time
```

Parameters

handle	MIDI port handle
speed	Play speed
fade_time	Time to reach target play speed

Return Value

SDE_ERR_NOTHING	No error
SDE_ERR_NO_INIT	Sound Library has not been initialized
SDE_ERR_HANDLE_NULL	Handle is NULL
SDE_ERR_PRM_OVER_RANGE	Out-of-range parameter value
SDE_ERR_HOST_CMD_BUF_NO_ENOUGH	Host command buffer overflow

Description

Sets the sequence playback speed for the specified MIDI port.

The parameter speed which marks the target play speed is calculated as follows:

`speed = m x 1000H - 1000H`

where multiplier m = 1/64 to 3. Valid calculation results must be between -FC0H (min) and 1FFFH (max).

The argument speed is a relative value with respect to the playback speed setting of the created data. In other words, to return the data to its original speed setting, set speed to 0.

The units of fade_time, which indicates the time to reach the playback speed, are in milliseconds from 0000H (fastest) to 7FFFH (slowest). During the interval until this playback speed is reached, the function `sdSysServer()` must be called at each VSync.

Note

This does not affect the MIDI message sending API function `sdMidiSendMes()`.

Reference

sdMidiSendMes()	Sends a MIDI message to a MIDI port

sdMidiSetVol

Sets the volume of a MIDI port.

Format

```
SDE_ERR sdMidiSetVol( handle, vol, fade_time )
SDMIDI handle
const Sint8 vol
const Sint32 fade_time
```

Parameters

handle	MIDI port handle
vol	Volume
fade_time	Time to reach target volume

Return Value

SDE_ERR_NOTHING	No error
SDE_ERR_NO_INIT	Sound Library has not been initialized
SDE_ERR_HANDLE_NULL	Handle is NULL
SDE_ERR_PRM_OVER_RANGE	Out-of-range parameter value
SDE_ERR_HOST_CMD_BUF_NO_ENOUGH	Host command buffer overflow

Description

Sets the playback volume for the specified MIDI port.

The playback volume setting is in the range 007FH (minimum) to 0000H (standard) to 007FH (maximum).

The argument vol is a relative value with respect to the playback volume setting of the created data. In other words, to return the data to its original volume setting, set vol to 0.

The units of fade_time, which indicates the time to reach the volume setting, are in milliseconds from 0000H (fastest) to 7FFFH (slowest). During the interval until this volume is reached, the function `sdSysServer()` must be called at each VSync.

Note

Very low volume levels may be inaudible. Also, setting a higher level may not increase the volume (if, for example, the level of the original data was already set high).

sdMidiStop

Stops playback on a MIDI port.

Format

```
SDE_ERR sdMidiStop( handle )
SDMIDI handle
```

Parameters

`handle`	MIDI port handle

Return Value

`SDE_ERR_NOTHING`	No error
`SDE_ERR_NO_INIT`	Sound Library has not been initialized
`SDE_ERR_HANDLE_NULL`	Handle is `NULL`
`SDE_ERR_HOST_CMD_BUF_NO_ENOUGH`	Host command buffer overflow

Description

Stops MIDI sequence playback on the specified port.

sdMidiStopAll

Stops playback of all MIDI sequence data.

Format

```
SDE_ERR sdMidiStopAll( Void )
```

Parameters

None

Return Value

SDE_ERR_NOTHING	No error
SDE_ERR_NO_INIT	Sound Library has not been initialized
SDE_ERR_HOST_CMD_BUF_NO_ENOUGH	Host command buffer overflow

Description

Stops playback of the MIDI sequences of all MIDI ports.

7. MIDI Module Control Functions

sdPstmClosePort

Releases PCM Stream port access permission.

Format

```
SDE_ERR sdPstmClosePort( handle )
SDPSTM handle
```

Parameters

handle	PCM Stream port handle

Return Value

SDE_ERR_NOTHING	No error
SDE_ERR_NO_INIT	Sound Library has not been initialized
SDE_ERR_HANDLE_NULL	Handle is NULL

Description

Releases the access permission of the specified PCM Stream port .

Note

Frequently getting and releasing access permission puts a heavy load on the CPU. As far as possible, pass the handle around. For example, if the handle is obtained when initializing a stage and released when ending the stage, this will keep the CPU load to an acceptable level.

sdPstmGetCurAdr — Gets the current play address of a PCM Stream.

Format

```
SDE_ERR sdPstmGetCurAdr( handle, target_slot, cur_adr )
SDPSTM handle
const Sint8 target_slot
Sint32 *cur_adr
```

Parameters

handle	PCM Stream port handle
target_slot	Slot number
cur_adr	Pointer to receive current address

Return Value

SDE_ERR_NOTHING	No error
SDE_ERR_NO_INIT	Sound Library has not been initialized
SDE_ERR_HANDLE_NULL	Handle is NULL
SDE_ERR_HOST_CMD_BUF_NO_ENOUGH	Host command buffer overflow

Description

Gets the current playback address of the PCM stream currently playing on the specified PCM stream port and slot.

sdPstmGetStat

Gets PCM Stream port status.

Format

```
SDE_ERR sdPstmGetStat( handle, targer_slot, stat )
SDPSTM handle
const Sint8 targer_slot
SDS_PSTM_STAT *stat
```

Parameters

handle	PCM Stream port handle
targer_slot	Slot number
stat	Pointer to storage for the status of PCM Stream port

Return Value

SDE_ERR_NOTHING	No error
SDE_ERR_NO_INIT	Sound Library has not been initialized
SDE_ERR_HANDLE_NULL	Handle is NULL

Description

Gets the status of the specified PCM stream port and slot number.

sdPstmGetTotalSmpFrame

Gets total PCM Stream play sample frames.

Format

```
SDE_ERR sdPstmGetTotalSmpFrame( handle, target_ch,
total_smp_frame )
SDPSTM handle
const Sint8 target_ch
Uint32 *total_smp_frame
```

Parameters

`handle`	PCM Stream port handle
`target_ch`	Slot number
`total_smp_frame`	Pointer to storage for total playing sample frame count

Return Value

`SDE_ERR_NOTHING`	No error
`SDE_ERR_NO_INIT`	Sound Library has not been initialized
`SDE_ERR_HANDLE_NULL`	Handle is `NULL`
`SDE_ERR_HOST_CMD_BUF_NO_ENOUGH`	Host command buffer overflow

Description

Gets a count of the total PCM stream play sample frames included in the currently playing PCM stream. The count of sample frames is reset each time playback starts.

sdPstmIsTransferWaveData

Checks if okay to transfer a sample frame to a PCM Stream port.

Format

```
SDE_ERR sdPstmIsTransferWaveData( handle, target_slot,
data_sz, flg )
SDPSTM handle
const Sint8 target_slot
const Sint32 data_sz
Bool *flg
```

Parameters

`handle`	Object handle
`target_slot`	Slot number
`data_sz`	Size of sample frame to transfer
`flg`	Pointer to storage for test result

Return Value

`SDE_ERR_NOTHING`	No error
`SDE_ERR_NO_INIT`	Sound Library has not been initialized
`SDE_ERR_HANDLE_NULL`	Handle is `NULL`
`SDE_ERR_PRM_OVER_RANGE`	Out-of-range parameter value

Description

Checks if okay to transfer a waveform to the specified PCM stream port and slot.

The units of `data_size` which specifies the sample frame size are bytes.

Argument `flg` is a pointer to storage for test result; true if transfer possible, or false if not.

Note

Do not transfer a sample frame until the result returned in the location pointed to be `flg` is true. If PCM stream playback has not been started, a sample frame size larger than 7000H cannot be specified for the argument `data_size`.

sdPstmOnMemPlay

Plays on-memory PCM Stream.

Format

```
SDE_ERR sdPstmOnMemPlay( handle, pcm_type, pcm_freq,
transfer_sz, loop_flg )
SDPSTM handle
const SDE_PCM_TYPE pcm_type
const Uint16 pcm_freq
const Sint32 transfer_sz
const Bool loop_flg
```

Parameters

`handle`	PCM Stream port handle
`pcm_type`	PCM stream type
`pcm_freq`	Sampling frequency
`transfer_sz`	Size to be transferred to ring buffer
`loop_flag`	Repeat playback flag

Return Value

`SDE_ERR_NOTHING`	No error
`SDE_ERR_NO_INIT`	Sound Library has not been initialized
`SDE_ERR_HANDLE_NULL`	Handle is `NULL`
`SDE_ERR_PRM_OVER_RANGE`	Out-of-range parameter value

Description

Plays on-memory PCM stream sample frames.

The argument `pcm_type` which specifies the PCM stream type has the following values.

Definition	Meaning
`SDE_PCM_TYPE_4BIT_ADPCM`	4-bit ADPCM compressed format
`SDE_PCM_TYPE_8BIT_LINEAR`	8-bit PCM uncompressed format
`SDE_PCM_TYPE_16BIT_LINEAR`	16-bit PCM uncompressed format

The argument `smp_frequency` specifying the sampling frequency indicates the sampling frequency used for playback. The argument `transfer_sz` specifying the size to be transferred to the ring buffer must be an integral divisor of the size of the ring buffer actually used.

For example, if the ring buffer size is 4000H, this is 4000H divided by some integer n, for example 1000H, or 2000H. The size must also correspond to 32-byte boundaries.

The repeated playback flag `loop_flag` contains true for repeated playback or false for single-play. Specifying repeated playback treats the specified memory block as an endless loop.

Example

```
/*
 * Monaural on-memory PCM stream sample
 */
/* Base parameters */
#define BASE_VOL        (0x7F)
#define BASE_PAN        (0x40)
#define BASE_FX_CH      (0x00)
#define BASE_FX_LEV     (0x00)
#define BASE_DRCT_LEV   (0x7F)
#define BASE_FILTER_LEV (SDD_PSTM_PORT_SLOT_FILTER_THROUGH)
#define BASE_FILER_Q    (0x00)
/* Playback waveform size */
#define WAVE_SZ         (0x10000)
/* Size to transfer to ring buffer */
#define TRANSFER_SZ     (0x1000)
/* Loop or not */
#define LOOP_FLG                (true)
SDPSTM  pstm_handle;
SDMEMBLK        memblk;
Void    *src_wave;      /* Source waveform value must be divisible by transfer_sz */
src_wave = syMalloc( WAVE_SZ);  /* Address of wavefore for actual playback */
sdMemBlkCreate( &memblk);
sdMemBlkSetPrm( memblk, src_wave, WAVE_SZ, SDD_MEMBLK_NO_FUNC, NULL);
/* Get PCM stream port access permission, using PCM stream ring buffer index 0 */
sdPstmOpenPort( &pstm_handle, 1, 0);
/* Set base parameters */
sdPstmSetBasePrm( pstm_handle, SDD_PSTM_PORT_SLOT_ALL,
                BASE_VOL, BASE_PAN, BASE_FX_CH, BASE_FX_LEV,
                BASE_DRCT_LEV, BASE_FILTER_LEV, BASE_FILTER_Q);
/* Set source memory block for on-memory playback */
sdPstmOnMemSetWaveData( pstm_handle, 0, memblk);
/* On-memory playback */
sdPstmOnMemPlay( pstm_handle, SDE_PCM_TYPE_16BIT_LINEAR,
                44100, TRANSFER_SZ, LOOP_FLG);
/* Game processing */
main_loop();
/* Stop playback */
sdPstmStop( pstm_handle);
/* Termination processing */
sdMemBlkDestroy( memblk);
sdPstmClosePort( pstm_handle);
syMalloc( src_wave);
```

Note

Set the sample frame memory block size to be an integral multiple of `transfer_sz`.

‘‘Reference

`sdPstmOnMemSetWaveData()`	Sets on-memory PCM Stream

dPstmOnMemSetWaveData

Sets on-memory PCM Stream.

Format

```
SDE_ERR sdPstmOnMemSetWaveData( handle, target_slot,
src_handle )
SDPSTM handle
const Sint8 target_slot
SDMEMBLK src_handle
```

Parameters

handle	PCM stream port handle
target_slot	Slot number
src_handle	Memory block handle

Return Value

SDE_ERR_NOTHING	No error
SDE_ERR_NO_INIT	Sound Library has not been initialized
SDE_ERR_HANDLE_NULL	Handle is NULL
SDE_ERR_PRM_OVER_RANGE	Out-of-range parameter value

Description

Sets on-memory PCM stream playback.

Example

```
/*
 * Monaural on-memory PCM stream sample
 */
/* Base parameters */
#define BASE_VOL        (0x7F)
#define BASE_PAN        (0x40)
#define BASE_FX_CH      (0x00)
#define BASE_FX_LEV     (0x00)
#define BASE_DRCT_LEV   (0x7F)
#define BASE_FILTER_LEV (SDD_PSTM_PORT_SLOT_FILTER_THROUGH)
#define BASE_FILER_Q    (0x00)
/* Playback waveform size */
#define WAVE_SZ         (0x10000)
/* Size to transfer to ring buffer */
#define TRANSFER_SZ     (0x1000)
/* Loop or not */
#define LOOP_FLG                (true)
SDPSTM  pstm_handle;
SDMEMBLK        memblk;
Void    *src_wave;      /* Source waveform value must be divisible by transfer_sz */
src_wave = syMalloc( WAVE_SZ);  /* Address of wavefore for actual playback */
sdMemBlkCreate( &memblk);
```

```
sdMemBlkSetPrm( memblk, src_wave, WAVE_SZ, SDD_MEMBLK_NO_FUNC, NULL);
/* Get PCM stream port access permission, using PCM stream ring buffer index 0 */
sdPstmOpenPort( &pstm_handle, 1, 0);
/* Set base parameters */
sdPstmSetBasePrm( pstm_handle, SDD_PSTM_PORT_SLOT_ALL,
                BASE_VOL, BASE_PAN, BASE_FX_CH, BASE_FX_LEV,
                BASE_DRCT_LEV, BASE_FILTER_LEV, BASE_FILTER_Q);
/* Set source memory block for on-memory playback */
sdPstmOnMemSetWaveData( pstm_handle, 0, memblk);
/* On-memory playback */
sdPstmOnMemPlay( pstm_handle, SDE_PCM_TYPE_16BIT_LINEAR,
                44100, TRANSFER_SZ, LOOP_FLG);
/* Game processing */
main_loop();
/* Stop playback */
sdPstmStop( pstm_handle);
/* Termination processing */
sdMemBlkDestroy( memblk);
sdPstmClosePort( pstm_handle);
syMalloc( src_wave);
```

sdPstmOpenPort

Gets PCM Stream port access permission.

Format

```
SDE_ERR sdPstmOpenPort( handle, allocate_slot_num,
use_ring_buf_num, ... )
SDPSTM *handle
const Sint8 allocate_slot_num
Sint8 use_ring_buf_num
```

Parameters

`handle`	Pointer to storage for port number
`allocate_slot_num`	Number of slots to allocate
`use_ring_buf_num`	Ring buffer number to use

Return Value

`SDE_ERR_NOTHING`	No error
`SDE_ERR_NO_INIT`	Sound Library has not been initialized
`SDE_ERR_HANDLE_NULL`	Handle is `NULL`
`SDE_ERR_PRM_OVER_RANGE`	Out-of-range parameter value
`SDE_ERR_HANDLE_NO_ENOUGH`	

Description

Gets PCM Stream port access permission.

The range of the argument `allocate_slot_num` which specifies the number of slots to allocate is from 1 to the macro-defined constant `SDD_PSTM_PORT_SLOT_NUM`.

The function has a variable number of parameters: specify the required ring buffer numbers in argument `use_ring_buf_num` and subsequent arguments.

Example

```
SDPSTM pstm_handle;
/*  case 1  */
sdPstmOpen( pstm_handle, 1, 0);  /*  Allocate one slot and use PCM stream ring buffer 0 */
/*  case 2  */
sdPstmOpen( pstm_handle, 2, 0, 1);  /*  Allocate two slots and use PCM stream ring buffers
0 and 1 */
```

Note

Allocating more than one slot within a single port reduces the number of slots that can be allocated simultaneously. The PCM stream ring buffer number or numbers to be used should be determined when creating the data. Frequently getting and releasing access permission puts a heavy load on the CPU. As far as possible, pass the handle around. For example, if the handle is obtained when initializing a stage and released when ending the stage, this will keep the CPU load to an acceptable level.

sdPstmPlay

Plays a PCM Stream.

Format

```
SDE_ERR sdPstmPlay( handle, pcm_type, pcm_frequency )
SDPSTM handle
const SDE_PCM_TYPE pcm_type
const Uint16 pcm_frequency
```

Parameters

`handle`	PCM Stream port handle
`pcm_type`	PCM Stream type
`pcm_frequency`	Sampling frequency

Return Value

`SDE_ERR_NOTHING`	No error
`SDE_ERR_NO_INIT`	Sound Library has not been initialized
`SDE_ERR_HANDLE_NULL`	Handle is `NULL`
`SDE_ERR_HOST_CMD_BUF_NO_ENOUGH`	Host command buffer overflow

Description

Plays a PCM stream.

The argument `pcm_type` which specifies the PCM encoding format stream type has the following values.

Definition	Meaning
SDE_PCM_TYPE_4BIT_ADPCM	4-bit ADPCM compressed format
SDE_PCM_TYPE_8BIT_LINEAR	8-bit PCM uncompressed format
SDE_PCM_TYPE_16BIT_LINEAR	16-bit PCM uncompressed format

The argument `smp_frequency` specifies the sampling frequency for playback (unit Hz). The upper limit is 48,000. For example, for CD quality, specify 44,100.

Example

```
/*  4BIT ADPCM format Play */
sdPstmPlay( pstm_handle, SDE_PCM_TYPE_4BIT_ADPCM, 44100);
/*  8BIT Linear format Play */
sdPstmPlay( pstm_handle, SDE_PCM_TYPE_8BIT_LINEAR, 44100);
/*  16BIT Linear format Play */
sdPstmPlay( pstm_handle, SDE_PCM_TYPE_16BIT_LINEAR, 44100);
```

Note

The method of specifying the argument `pcm_frequency` for the sampling frequency has changed from Sound Library Ver. 2.0. Instead of specifying a defined constant, the sampling frequency is specified directly.

sdPstmResetAllPrm

Resets parameters of all PCM Stream ports.

Format

```
SDE_ERR sdPstmResetAllPrm( Void )
```

Parameters

None

Return Value

SDE_ERR_NOTHING	No error
SDE_ERR_NO_INIT	Sound Library has not been initialized
SDE_ERR_HOST_CMD_BUF_NO_ENOUGH	Host command buffer overflow

Description

Resets parameters of all PCM Stream ports.

sdPstmResetPrm

Resets a PCM Stream port.

Format

```
SDE_ERR sdPstmResetPrm( handle, target_slot )
SDPSTM handle
const Sint8 target_slot
```

Parameters

`handle`	PCM stream port handle
`target_slot`	Slot number

Return Value

`SDE_ERR_NOTHING`	No error
`SDE_ERR_NO_INIT`	Sound Library has not been initialized
`SDE_ERR_HANDLE_NULL`	Handle is `NULL`
`SDE_ERR_HOST_CMD_BUF_NO_ENOUGH`	Host command buffer overflow

Description

Resets the parameters for the specified PCM stream port and slot number.

sdPstmSetBasePrm Sets base parameters for a PCM Stream port.

Format

```
SDE_ERR sdPstmSetBasePrm( handle, target_slot, base_vol,
base_pan, fx_in_ch, base_fx_lev, base_drct_lev,
base_filter_lev, base_q_lev )
SDPSTM handle
const Sint8 target_slot
const Sint8 base_vol
const Sint8 base_pan
const Sint8 fx_in_ch
const Sint8 base_fx_lev
const Sint8 base_drct_lev
const Sint16 base_filter_lev
const Sint8 base_filter_q
```

Parameters

handle	PCM Stream port handle
target_slot	Slot number
base_vol	Base volume
base_pan	Base panpot setting
fx_in_ch	FX input channel number
base_fx_lev	Base FX input level
base_drct_lev	Base direct level
base_filter_lev	Base filter level
base_q_lev	Base Q level

Return Value

SDE_ERR_NOTHING	No error
SDE_ERR_NO_INIT	Sound Library has not been initialized
SDE_ERR_HANDLE_NULL	Handle is NULL

Description

Sets the base parameters for a PCM stream port.

The argument target_slot specifies the slot number for the settings. Specifying SDD_PSTM_PORT_SLOT_ALL for this argument makes the settings for all slots of the port.

The argument base_vol sets the base volume. The range of values is from 0000H (minimum) to 007FH (maximum). The final volume is determined by calculation from the value of this argument and the argument vol specified to the sdPstmSetVol() function.

The argument base_pan sets the base panpot value. The range of values is from 0000H (left end) to 0040H (center) to 007FH(right end). The final panpot setting is determined by calculation from the value of this argument and the argument pan specified to the sdPstmSetPan() function.

The argument fx_in_ch sets the FX input channel number. The range of FX input channel numbers is from 0000H to 000FH.

The argument `base_fx_lev` sets the base FX level for sending to FX. The final FX level is determined by calculation from the value of this argument and the argument `fx_lev` specified to the `sdPstmSetFxLev()` function.

The argument `base_drct_lev` sets the base direct level. The range of values is from 0000H (minimum) to 007FH (maximum). The final direct level is determined by calculation from the value of this argument and the argument `drct_lev` specified to the `sdPstmSetDrctLev()` function.

The argument `base_filter_lev` sets the base filter level. The range of values is from 0000H (minimum) to 007FH (maximum). This cannot be changed after playback. A change in the setting after playback takes effect from the next playback. In other words, this setting must be made before playback. If this argument is set to `SDD_PSTM_PORT_SLOT_FILTER_THROUGH`, then no filtering is carried out.

The argument `base_q_lev` sets the filter base Q level. The values are from 0000H (minimum) to 007FH (maximum). This cannot be changed after playback. A change in the setting after playback takes effect from the next playback. In other words, this setting must be made before playback.

If the argument `base_filter_lev` is set to `SDD_PSTM_PORT_SLOT_FILTER_THROUGH`, then this argument is ignored. The settings of parameters by this function takes effect with the execution of the `sdPstmPlay()` function.

Example

```
/* Case 1(Stereo*1) */
sdPstmOpen( &pcm_handle, 2, 0, 1);
sdPstmSetBasePrm( pcm_handle, 0, 0x7F, 0x00, 0, 0x00, 0x7F,
        SDD_PSTM_PORT_SLOT_FILTER_THROUGH, 0);
sdPstmSetBasePrm( pcm_handle, 1, 0x7F, 0x7F, 0, 0x00, 0x7F,
        SDD_PSTM_PORT_SLOT_FILTER_THROUGH, 0);
/* Case 2 (Monaural*4) */
sdPstmOpen( &pcm_handle, 4, 0, 1, 2, 3);
sdPstmSetBasePrm( pcm_handle, SDD_PSTM_PORT_SLOT_ALL,
        0x7F, 0x00, 0, 0x00, 0x7F,
        SDD_PSTM_PORT_SLOT_FILTER_THROUGH, 0);
```

Note

Determine the FX input channel number from the connected FX at the time of sound data creation. To stop filtering, specify the argument `base_filter_lev` as `SDD_PSTM_PORT_SLOT_FILTER_THROUGH`.

Reference

`sdPstmSetVol()`	Sets the playback volume of a PCM Stream port
`sdPstmSetPan()`	Sets the panpot setting of a PCM Stream port
`sdPstmSetPitch()`	Sets the playback pitch of a PCM Stream port
`sdPstmSetFxCh()`	Sets the FX input channel number for a PCM Stream port
`sdPstmSetFxLev()`	Sets the FX input level for a PCM Stream port
`sdPstmSetDrctLev()`	Sets the direct level of a PCM Stream port

sdPstmSetDrctLev

Sets the direct level of a PCM Stream port.

Format

```
SDE_ERR sdPstmSetDrctLev( handle, target_slot, drct_lev )
SDPSTM handle
const Sint8 target_slot
const Sint8 drct_lev
```

Parameters

`handle`	PCM Stream port handle
`target_slot`	Slot number
`drct_lev`	Direct level

Return Value

`SDE_ERR_NOTHING`	No error
`SDE_ERR_NO_INIT`	Sound Library has not been initialized
`SDE_ERR_HANDLE_NULL`	Handle is `NULL`
`SDE_ERR_HOST_CMD_BUF_NO_ENOUGH`	Host command buffer overflow

Description

Sets the direct level (the sound level other than for effects) of the specified PCM stream port.

The range of values for `drct_lev`, which specifies direct levels, is -007FH (minimum) to 0000H (standard) to 007FH (maximum).

sdPstmSetFxCh

Sets the FX input channel number for a PCM Stream port.

Format

```
SDE_ERR sdPstmSetFxCh( handle, target_slot, fx_in_ch,
base_fx_lev )
SDPSTM handle
const Sint8 target_slot
const Sint8 fx_in_ch
const Sint8 base_fx_lev
```

Parameters

`handle`	PCM Stream port handle
`target_slot`	Slot number
`dsp_in_ch`	FX input channel number
`base_fx_lev`	Base input level

Return Value

`SDE_ERR_NOTHING`	No error
`SDE_ERR_NO_INIT`	Sound Library has not been initialized
`SDE_ERR_HANDLE_NULL`	Handle is `NULL`
`SDE_ERR_PRM_OVER_RANGE`	Out-of-range parameter value
`SDE_ERR_HOST_CMD_BUF_NO_ENOUGH`	Host command buffer overflow

Description

Sets the FX input channel number for a PCM stream port.

The FX input channel can be specified in the range 00H to 0FH.

The argument target_slot specifies the slot number for the setting. Specifying `SDD_PSTM_PORT_SLOT_ALL` for this argument makes the settings for all slots of the port.

The range of `base_fx_lev`, which specifies base input levels, is from 0000H (minimum) to 007FH (maximum). Parameters `dsp_in_ch` and `base_fx_lev` are absolute settings.

The final FX level is determined by calculation from the value of the `base_fx_lev` argument and the argument `fx_lev` specified to the `sdPstmSetFxLev()` function.

sdPstmSetFxLev

Sets the FX input level for a PCM Stream port.

Format

```
SDE_ERR sdPstmSetFxLev( handle, target_slot, fx_lev )
SDPSTM handle
const Sint8 target_slot
const Sint8 fx_lev
```

Parameters

handle	PCM stream port handle
target_slot	Slot number
fx_lev	FX input level

Return Value

SDE_ERR_NOTHING	No error
SDE_ERR_NO_INIT	Sound Library has not been initialized
SDE_ERR_HANDLE_NULL	Handle is NULL
SDE_ERR_PRM_OVER_RANGE	Out-of-range parameter value
SDE_ERR_HOST_CMD_BUF_NO_ENOUGH	Host command buffer overflow

Description

Sets the FX input level for a PCM Stream port.

The argument target_slot specifies the slot number for the setting. Specifying SDD_PSTM_PORT_SLOT_ALL for this argument makes the settings for all slots of the port.

The range of values for fx_lev, which specifies FX levels, is -0080H (minimum) to 0000H (standard) to 007FH (maximum).

sdPstmSetPan

Sets the panpot setting of a PCM Stream port.

Format

```
SDE_ERR sdPstmSetPan( handle, target_slot, pan, fade_time )
SDPSTM handle
const Sint8 target_slot
const Sint8 pan
const Sint32 fade_time
```

Parameters

handle	PCM stream port handle
target_slot	Slot number
pan	Panpot value
fade_time	Time to reach target panpot value

Return Value

SDE_ERR_NOTHING	No error
SDE_ERR_NO_INIT	Sound Library has not been initialized
SDE_ERR_HANDLE_NULL	Handle is NULL
SDE_ERR_HOST_CMD_BUF_NO_ENOUGH	Host command buffer overflow

Description

Sets the panpot setting for each channel of the specified PCM stream port.

The value range of pan, which specifies the target panpot value, is -007FH (left) to 0000H (center) to 007FH (right).

The units of fade_time, which indicates the time to reach the panpot setting, are in milliseconds from 0000H (fastest) to 7FFFH (slowest). During the interval until this playback speed is reached, the function sdSysServer() must be called at each VSync.

sdPstmSetPitch

Sets the playback pitch of a PCM Stream port.

Format

```
SDE_ERR sdPstmSetPitch( handle, target_slot, pitch, fade_time
)
SDPSTM handle
const Sint8 target_slot
const Sint16 pitch
const Sint32 fade_time
```

Parameters

`handle`	PCM Stream port handle
`target_slot`	Slot number
`pitch`	Target play pitch
`fade_time`	Time to reach target pitch value

Return Value

`SDE_ERR_NOTHING`	No error
`SDE_ERR_NO_INIT`	Sound Library has not been initialized
`SDE_ERR_HANDLE_NULL`	Handle is `NULL`
`SDE_ERR_PRM_OVER_RANGE`	Out-of-range parameter value
`SDE_ERR_HOST_CMD_BUF_NO_ENOUGH`	Host command buffer overflow

Description

Sets the playback pitch of the specified PCM Stream port.

The argument pitch, which specifies the playback pitch, has the value 0100H for one semitone higher and -0100H for one semitone lower. The upper limit which can be specified corresponds to just below the sampling frequency of 88200 Hz in the case of the ADPCM format. For playback at 44100 Hz, the pitch can be up to 0BFFH. At 22100 Hz the setting can be twice this.

For linear format, an even higher pitch can be set. There is close to no lower limit, but use 23FFH as a guideline. The units of fade_time, which indicates the time to reach the target playback pitch, are in milliseconds from 0000H (fastest) to 7FFFH (slowest).

During the interval until this playback pitch is reached, the function `sdSysServer()` must be called at each VSync.

Note

As a side-effect of raising the playback pitch, the playback speed increases; and as a side-effect of lowering the playback pitch, the playback speed decreases.

sdPstmSetVol

Sets the playback volume of a PCM Stream port.

Format

```
SDE_ERR sdPstmSetVol( handle, target_slot, vol, fade_time )

SDPSTM handle
const Sint8 target_slot
const Sint8 vol
const Sint32 fade_time
```

Parameters

handle	PCM Stream port handle
target_slot	Slot number
vol	Volume
fade_time	Time to reach target volume level

Return Value

SDE_ERR_NOTHING	No error
SDE_ERR_NO_INIT	Sound Library has not been initialized
SDE_ERR_HANDLE_NULL	Handle is NULL
SDE_ERR_HOST_CMD_BUF_NO_ENOUGH	Host command buffer overflow

Description

Sets the playback volume for a slot of the specified PCM stream port.

The argument target_slot specifies the slot number for the setting. Specifying SDD_PSTM_PORT_SLOT_ALL for this argument makes the settings for all slots of the port.

The range for vol, which specifies volume, is -0080H (minimum) to 0000H (standard) to 007FH (maximum). The units of fade_time, which indicates the time to reach the target volume, are in milliseconds from 0000H (fastest) to 7FFFH (slowest). During the interval until this playback pitch is reached, the function sdSysServer() must be called at each VSync.

Note

Very low volume levels may be inaudible. Also, setting a higher level may not increase the volume (if, for example, the level of the original data was already set high).

sdPstmStop

Stops PCM Stream playback.

Format

```
SDE_ERR sdPstmStop( handle )
SDPSTM handle
```

Parameters

`handle`	PCM stream port handle

Return Value

`SDE_ERR_NOTHING`	No error
`SDE_ERR_NO_INIT`	Sound Library has not been initialized
`SDE_ERR_HANDLE_NULL`	Handle is `NULL`
`SDE_ERR_HOST_CMD_BUF_NO_ENOUGH`	Host command buffer overflow

Description

Stops playback of the PCM stream on the specified port.

Note

The method of continuous playback is different from other sound sources. After using this function to stop playback, always use the `sdPstmPlay()` function to start playback.

sdPstmStopAll

Stops playback on all PCM Streams.

Format

```
SDE_ERR sdPstmStopAll( Void )
```

Parameters

None

Return Value

SDE_ERR_NOTHING	No error
SDE_ERR_NO_INIT	Sound Library has not been initialized
SDE_ERR_HOST_CMD_BUF_NO_ENOUGH	Host command buffer overflow

Description

Stops PCM stream playback on all ports.

sdPstmTransferWaveData

Transfers sample data to a PCM Stream port.

Format

```
SDE_ERR sdPstmTransferWaveData( handle, target_ch,
memblk_handle )
SDPSTM handle
const Sint8 target_ch
SDMEMBLK memblk_handle
```

Parameters

`handle`	PCM stream port handle
`target_ch`	Channel number
`memblk_handle`	Memory block handle

Return Value

`SDE_ERR_NOTHING`	No error
`SDE_ERR_HOST_CMD_BUF_NO_ENOUGH`	Host command buffer overflow
`SDE_ERR_NO_INIT`	Sound Library has not been initialized
`SDE_ERR_HANDLE_NULL`	Handle is `NULL`

Description

Transfers a sample data to a PCM Stream port.

Note

Check if it is okay to transfer sample data before actually transferring. Execute the `sdPstmlsTransferWaveData()` function to see if it possible to transfer the sample data.

If PCM stream playback has not been started, a sample data size larger than 7000H cannot be specified. The maximum size of ring buffer which can actually be used depends on the PCM encoding format.

Format	Defined constant	Maximum value
16-bit uncompressed format	SDE_PCM_TYPE_16BIT_LINEAR	001F000H bytes
8-bit uncompressed format	SDE_PCM_TYPE_8BIT_LINEAR	000F000H bytes
4-bit ADPCM compressed format	SDE_PCM_TYPE_4BIT_ADPCM	0007000H bytes

Reference

`sdPstmIsTransferWaveData()`	Checks if okay to transfer a sample frame to a PCM Stream port

8. One Shot Module Control Functions

sdShotClosePort

Releases one-shot port access permission.

Format

```
SDE_ERR sdShotClosePort( handle ) SDSHOT handle
```

Parameters

handle	One-shot port handle

Return Value

SDE_ERR_NOTHING	No error
SDE_ERR_NO_INIT	Sound Library has not been initialized
SDE_ERR_HANDLE_NULL	Handle is NULL

Description

Releases access permission for the specified one-shot port.

Example

```
SDShot SHOT_handle;
/* Get access permission for one-shot port */
sdShotOpenPort( &SHOT_handle);
/* Play one-shot bank number 00H from data value 00H with priority 00H */
sdShotPlay( SHOT_handle, 0, 0, 0);
/* Pause one-shot playback */
sdShotPause( SHOT_handle);
/* Resume one-shot playback */
sdShotResume( SHOT_handle);
/* Stop one-shot playback */
sdShotStop( SHOT_handle);
/* Release one-shot port access permission */
sdShotClosePort( SHOT_handle);
```

Note

Frequently getting and releasing access permission puts a heavy load on the CPU. As far as possible, pass the handle around. For example, if the handle is obtained when initializing a stage and released when ending the stage, this will keep the CPU load to an acceptable level.

sdShotGetCurAdr

Gets the current play address of a one-shot sample frame.

Format

```
SDE_ERR sdShotGetCurAdr( handle, cur_adr )
SDSHOT handle
Sint32 *cur_adr
```

Parameters

`handle`	One-shot port handle
`cur_adr`	Pointer to storage for the address of the sample frame

Return Value

`SDE_ERR_NOTHING`	No error
`SDE_ERR_NO_INIT`	Sound Library has not been initialized
`SDE_ERR_HANDLE_NULL`	Handle is `NULL`

Description

Gets the current play address of a one-shot sample frame.

sdShotGetStat

Gets one-shot port status.

Format

```
SDE_ERR sdShotGetStat( handle, stat )
SDSHOT handle
SDS_SHOT_STAT *stat
```

Parameters

`handle`	One-shot port handle
`stat`	Pointer to storage for one-shot port status

Return Value

`SDE_ERR_NOTHING`	No error
`SDE_ERR_NO_INIT`	Sound Library has not been initialized
`SDE_ERR_HANDLE_NULL`	Handle is `NULL`

Description

Gets the status of the specified one-shot port.

sdShotGetTotalSmpFrame

Gets total count of one-shot play sample frames.

Format

```
SDE_ERR sdShotGetTotalSmpFrame( handle, total_smp_frame )
SDSHOT handle
Uint32 *total_smp_frame
```

Parameters

`handle`	One-shot port handle
`total_smp_frame`	Pointer to storage for total sample frame count

Return Value

`SDE_ERR_NOTHING`	No error
`SDE_ERR_NO_INIT`	Sound Library has not been initialized
`SDE_ERR_HANDLE_NULL`	Handle is `NULL`

Description

Gets the total count of one-shot play sample frames being played on the specified

sdShotOpenPort

Gets one-shot port access permission.

Format

```
SDE_ERR sdShotOpenPort( handle )
SDSHOT *handle
```

Parameters

handle	Pointer to storage for one-shot port handle

Return Value

SDE_ERR_NOTHING	No error
SDE_ERR_NO_INIT	Sound Library has not been initialized
SDE_ERR_HANDLE_NO_ENOUGH	Unable to get handle

Description

Gets one-shot port access permission.

Note

Frequently getting and releasing access permission puts a heavy load on the CPU. As far as possible, pass the handle around. For example, if the handle is obtained when initializing a stage and released when ending the stage, this will keep the CPU load to an acceptable level.

sdShotPlay

Plays one-shot data.

Format

```
SDE_ERR sdShotPlay( handle, bank_num, list_num, priority )
SDSHOT handle
const Sint8 bank_num
const Sint8 list_num
const Sint8 priority
```

Parameters

handle	One-shot port handle
bank_num	One-shot bank number
list_num	One-shot data value number
priority	Priority level

Return Value

SDE_ERR_NOTHING	No error
SDE_ERR_NO_INIT	Sound Library has not been initialized
SDE_ERR_HANDLE_NULL	Handle is NULL
SDE_ERR_PRM_OVER_RANGE	Out-of-range parameter value
SDE_ERR_HOST_CMD_BUF_NO_ENOUGH	Host command buffer overflow

Description

Plays the one-shot sample stream from the specified port, bank, and data value number.

The argument bank_num specifying the playback one-shot bank number is in the range 0000H to 007FH.

The argument list_num which specifies the playback one-shot number (the data value number) is in the range 0000H to 007FH.

The range of priority, which specifies the priority level, is 00000H (always take priority) to 0001H (lowest) to 000FH (highest)

sdShotResetAllPrm

Resets parameters of all one-shot ports.

Format

```
SDE_ERR sdShotResetAllPrm( Void )
```

Parameters

None

Return Value

SDE_ERR_NOTHING	No error
SDE_ERR_NO_INIT	Sound Library has not been initialized
SDE_ERR_HANDLE_NULL	Handle is NULL
SDE_ERR_HOST_CMD_BUF_NO_ENOUGH	Host command buffer overflow

Description

Resets the parameters for all one-shot ports.

Note

All values are reset to their central, zero values.

sdShotResetPrm

Resets a one-shot port.

Format

```
SDE_ERR sdShotResetPrm( handle )
SDSHOT handle
```

Parameters

`handle`	One-shot port handle

Return Value

`SDE_ERR_NOTHING`	No error
`SDE_ERR_NO_INIT`	Sound Library has not been initialized
`SDE_ERR_HANDLE_NULL`	Handle is `NULL`
`SDE_ERR_HOST_CMD_BUF_NO_ENOUGH`	Host command buffer overflow

Description

Resets the one-shot parameters for a specified port. When the parameters are reset, the values are those set in the sound data.

Note

All values are reset to their central, zero values.

sdShotSetDrctLev

Sets the direct level of a one-shot port.

Format

```
SDE_ERR sdShotSetDrctLev( handle, drct_lev )
SDSHOT handle
const Sint8 drct_lev
```

Parameters

handle	One-shot port handle
drct_lev	Direct level to set

Return Value

SDE_ERR_NOTHING	No error
SDE_ERR_NO_INIT	Sound Library has not been initialized
SDE_ERR_HANDLE_NULL	Handle is NULL
SDE_ERR_HOST_CMD_BUF_NO_ENOUGH	Host command buffer overflow

Description

Sets the direct level (for other than sound effects) of the specified one-shot port.

The range of drct_lev, which specifies the direct level, is -007FH (minimum) to 0000H (standard) to 007FH (maximum).

sdShotSetFxCh Sets the FX input channel number for a one-shot port.

Format

```
SDE_ERR sdShotSetFxCh( handle, fx_in_ch, base_fx_lev )
SDSHOT handle
const Sint8 fx_in_ch
const Sint8 base_fx_lev
```

Parameters

`handle`	One-shot port handle
`vol`	FX input channel number
`fade_time`	Base FX input level

Return Value

`SDE_ERR_NOTHING`	No error
`SDE_ERR_NO_INIT`	Sound Library has not been initialized
`SDE_ERR_HANDLE_NULL`	Handle is `NULL`
`SDE_ERR_PRM_OVER_RANGE`	Out-of-range parameter value
`SDE_ERR_HOST_CMD_BUF_NO_ENOUGH`	Host command buffer overflow

Description

Sets the FX input channel number for a one-shot port. The FX input channels which can be set are from 00H to 0FH.

The range of `base_fx_lev`, which specifies the base FX input level, is from 0000H (fastest) to 7FFFH (slowest).

The arguments `fx_in_ch` and `base_fx_lev` are absolute values.

sdShotSetFxLev

Sets the FX input level for a one-shot port.

Format

```
SDE_ERR sdShotSetFxLev( handle, fx_lev )
SDSHOT handle
const Sint8 fx_lev
```

Parameters

handle	One-shot port handle
fx_lev	FX input level

Return Value

SDE_ERR_NOTHING	No error
SDE_ERR_NO_INIT	Sound Library has not been initialized
SDE_ERR_HANDLE_NULL	Handle is NULL
SDE_ERR_PRM_OVER_RANGE	Out-of-range parameter value
SDE_ERR_HOST_CMD_BUF_NO_ENOUGH	Host command buffer overflow

Description

Sets the FX input level for the specified one-shot port.

The range of fx_lev, which specifies the FX level, is -007FH (minimum) to 0000H (standard) to 007FH (maximum).

The FX input is at the level specified by the parameters in the one-shot data or by calculation from the value base_fx_lev specified to the sdShotSetFxCh() function and fx_lev.

sdShotSetPan

Sets the panpot value for a one-shot port.

Format

```
SDE_ERR sdShotSetPan( handle, pan, fade_time )
SDSHOT handle
const Sint8 pan
const Sint32 fade_time
```

Parameters

`handle`	One-shot port handle
`pan`	Panpot value
`fade_time`	Time to reach target panpot value

Return Value

`SDE_ERR_NOTHING`	No error
`SDE_ERR_NO_INIT`	Sound Library has not been initialized
`SDE_ERR_HANDLE_NULL`	Handle is `NULL`
`SDE_ERR_HOST_CMD_BUF_NO_ENOUGH`	Host command buffer overflow

Description

Sets the panpot value for the specified one-shot port.

The range of `pan`, the panpot setting, is -0080H (left) to 0000H (center) to 007FH (right).

The units of `fade_time`, which indicates the time to reach the target panpot setting, are in milliseconds from 0000H (fastest) to 7FFFH (slowest). During the interval until this playback pitch is reached, the function `sdSysServer()` must be called at each VSync.

sdShotSetPitch

Sets the pitch of a one-shot port.

Format

```
SDE_ERR sdShotSetPitch( handle, pitch, fade_time )
SDSHOT handle
const Sint16 pitch
const Sint32 fade_time
```

Parameters

`handle` One-shot port handle

`pitch`	Play pitch
`fade_time`	Time to reach target playback pitch

Return Value

`SDE_ERR_NOTHING`	No error
`SDE_ERR_NO_INIT`	Sound Library has not been initialized
`SDE_ERR_HANDLE_NULL`	Handle is `NULL`
`SDE_ERR_HOST_CMD_BUF_NO_ENOUGH`	Host command buffer overflow

Description

Sets the playback pitch of the specified one-shot port.

The argument `pitch` which indicates the playback pitch is 0100H for a semitone up, -0100H for a semitone down. The upper limit which can be specified corresponds to just below the sampling frequency of 88200 Hz in the case of the ADPCM format. For playback at 44100 Hz, the pitch can be up to 0BFFH. At 22100 Hz the setting can be twice this.

For linear format, an even higher pitch can be set. There is close to no lower limit, but use 23FFH as a guideline.

The units of `fade_time`, which indicates the time to reach the pitch setting, are in milliseconds from 0000H (fastest) to 7FFFH (slowest). During the interval until this pitch is reached, the function `sdSysServer()` must be called at each VSync.

Note

As a side-effect of raising the playback pitch, the playback speed increases; and as a side-effect of lowering the playback pitch, the playback speed decreases.

sdShotSetPlayTime

Sets playback time for one-shot playback.

Format

```
SDE_ERR sdShotSetPlayTime( handle, scale_type, time_value )

SDSHOT handle
const SDE_SHOT_PLAY_TIME scale_type
Sint32 time_value
```

Parameters

`handle`	One-shot port handle
`scale_type`	Type of time setting
`time_value`	Playback time

Return Value

`SDE_ERR_NOTHING`	No error
`SDE_ERR_NO_INIT`	Sound Library has not been initialized
`SDE_ERR_HANDLE_NULL`	Handle is `NULL`
`SDE_ERR_PRM_OVER_RANGE`	Out-of-range parameter value
`SDE_ERR_HOST_CMD_BUF_NO_ENOUGH`	Host command buffer overflow

Description

Sets the one-shot stream playback time for the specified port.

The playback time is indicated by the argument `time_value`, and the type of time specification by the argument `scale_type`.

The values for the argument `scale_type` are as follows.

Definition	Meaning
`SDE_SHOT_PLAY_TIME_RELATIVE`	Relative value (time to be directly added or subtracted from the parameter in the one-shot data)
`SDE_SHOT_PLAY_TIME_RATIO`	Ratio (time as a proportion of the parameter in the one-shot data)
`SDE_SHOT_PLAY_TIME_ABSOLUTE`	Absolute value (units of milliseconds)

Note

The sound driver operates with 4-ms timing, and therefore a setting of less than 4 ms is meaningless. There may be a discrepancy of several milliseconds in the actual play time.

sdShotSetVol

Sets the volume of a one-shot port.

Format

```
SDE_ERR sdShotSetVol( handle, vol, fade_time )
SDSHOT handle
const Sint8 vol
const Sint32 fade_time
```

Parameters

handle	One-shot port handle
vol	Volume
fade_time	Time to reach target volume

Return Value

SDE_ERR_NOTHING	No error
SDE_ERR_NO_INIT	Sound Library has not been initialized
SDE_ERR_HANDLE_NULL	Handle is NULL
SDE_ERR_HOST_CMD_BUF_NO_ENOUGH	Host command buffer overflow

Description

Sets the One-Shot volume of a specified port.

The range of vol, which specifies volume, is -007FH (minimum) to 0000H (standard) to 007FH (maximum).

The units of fade_time, which indicates the time to reach the target volume, are in milliseconds from 0000H (fastest) to 7FFFH (slowest). During the interval until this playback pitch is reached, the function `sdSysServer()` must be called at each VSync.

Note

Very low volume levels may be inaudible. Also, setting a higher level may not increase the volume (if, for example, the level of the original data was already set high).

sdShotStop

Stops one-shot playback.

Format

```
SDE_ERR sdShotStop( handle )
SDSHOT handle
```

Parameters

handle	One-shot port handle

Return Value

SDE_ERR_NOTHING	No error
SDE_ERR_NO_INIT	Sound Library has not been initialized
SDE_ERR_HOST_CMD_BUF_NO_ENOUGH	Host command buffer overflow

Description

Stops one-shot playback on the specified port.

sdShotStopAll

Stops all one-shot playback.

Format

```
SDE_ERR sdShotStopAll( Void )
```

Parameters

None

Return Value

SDE_ERR_NOTHING	No error
SDE_ERR_NO_INIT	Sound Library has not been initialized
SDE_ERR_HOST_CMD_BUF_NO_ENOUGH	Host command buffer overflow

Description

Stops playback on all one-shot ports.

9. System Functions

sdDrvCheckExecute

Checks execution of Sound Driver.

Format

```
SDE_ERR sdDrvCheckExecute( Void)
```

Parameters

None

Return Value

SDE_ERR_NOTHING	No error
SDE_ERR_NO_INIT	Sound Library has not been initialized
SDE_ERR_SND_DRV_PROBLEM	Sound driver has a problem

Description

Checks execution of the Sound Driver.

If the sound library finds a fault in the Sound Driver it returns an error.

sdDrvDownloadFromFile

Reads in and initializes Sound Driver.

Format

```
SDE_ERR sdDrvDownloadFromFile( file_name, work_memblk )
const char *file_name
SDMEMBLK work_memblk
```

Parameters

`file_name`	File name
`work_memblk`	Buffer memory block

Return Value

`SDE_ERR_NOTHING`	No error

Description

Reads in the sound driver with the specified file name from GD-ROM, transfers it to sound memory, and carries out initialization.

The argument `work_memblk` requires a minimum size of 800H bytes.

Example

```
#define WORK_SZ (0x800)
SDMEMBLK        memblk
Void    *buf;
buf = syMalloc( WORK_SZ);
sdLibInit( NULL, 0, 0);
sdMemBlkCreate( &memblk);
sdMemBlkSetPrm( memblk, buf, WORK_SZ, SDD_MEMBLK_NO_FUNC, NULL);
sdDrvDownloadFromFile( "manatee.drv", memblk);
sdMemBlkDestroy( memblk);
syFree( buf);
```

Note

This function must be called before the `sdLibInit()` function which initializes the sound driver.

Reference

`sdLibInit()`	Initializes Sound Library

sdDrvGetErr

Gets Sound Driver error information.

Format

```
SDE_ERR sdDrvGetErr( err )
Uint32 *err
```

Parameters

err	Pointer to storage for the Sound Driver error information

Return Value

SDE_ERR_NOTHING	No error
SDE_ERR_NO_INIT	Sound Library has not been initialized
SDE_ERR_PTR_NULL	Address is NULL

Description

Gets Sound Driver error information.

sdDrvGetExecuteCounter

Gets Sound Driver execution counter.

Format

```
SDE_ERR sdDrvGetExecuteCounter( count_num )
Uint32 *count_num
```

Parameters

`count_num`	Pointer to variable to hold the Sound Driver count

Return Value

`SDE_ERR_NOTHING`	No error
`SDE_ERR_NO_INIT`	Sound Library has not been initialized
`SDE_ERR_PTR_NULL`	Address is `NULL`

Description

Gets the counter value maintained by the Sound Driver using an interrupt routine.

Note

This function can be used to determine whether the Sound Driver is operating or not.

sdDrvGetVer

Gets Sound Driver version.

Format

```
SDE_ERR sdDrvGetVer( drv_ver )
SDS_VER *drv_ver
```

Parameters

drv_ver	Pointer to storage for version information

Return Value

SDE_ERR_NOTHING	No error
SDE_ERR_NO_INIT	Sound Library has not been initialized

Description

Gets Sound Driver version.

Note

The Sound Driver version can only be obtained after the initialization functions `sdDrvInit()` and `sdLibInit()` have been executed.

sdDrvInit

Initializes Sound Driver.

Format

```
SDE_ERR sdDrvInit( handle )
SDMEMBLK handle
```

Parameters

handle	Handle of memory block holding the Sound Driver

Return Value

SDE_ERR_NOTHING	No error

Description

Initializes the Sound Driver.

Note

If a callback function has been registered with the sdMemBlkSetPrm() function for setting memory block information, it can be called after the Sound Driver has been downloaded and initialized.

The function sdLibInit() must be executed first to initialize the Sound Library.

Reference

sdLibInit()	Initializes Sound Library

sdLibGetVer

Gets Sound Library version.

Format

```
SDE_ERR sdLibGetVer( lib_ver )
SDS_VER *lib_ver
```

Parameters

lib_ver	Pointer to storage for version information

Return Value

SDE_ERR_NOTHING	No error

Description

Gets Sound Library version.

sdLibInit

Initializes Sound Library.

Format

```
SDE_ERR sdLibInit( wrk_ptr, mem_blk_handle_max,
second_host_cmd_max )
Void *wrk_ptr
Sint32 mem_blk_handle_max
Sint32 second_host_cmd_max
```

Parameters

`wrk_ptr`	Pointer to the work buffer used by the Sound Library
`mem_blk_handle_max`	Maximum number of usable memory block handles
`second_host_cmd_max`	Maximum number of usable secondary command buffers

Return Value

`SDE_ERR_NOTHING`	No error

Description

`sdLibInit` must be executed first to initialize the Sound Library.

Note

If the Sound Library is going to be used, this function must be executed first. The Sound Driver is initialized by `sdDrvInit()`.

Reference

`sdDrvInit()`	Initializes Sound Driver

sdSysFinish

Closes the Sound Driver.

Format

```
SDE_ERR sdSysFinish( Void)
```

Parameters

None

Return Value

SDE_ERR_NOTHING	No error
SDE_ERR_NO_INIT	Sound Library has not been initialized

Description

Closes the Sound Driver.

sdSysFlushHostCmd

Sends a host command.

Format

```
SDE_ERR sdSysFlushHostCmd( Void )
```

Parameters

None

Return Value

SDE_ERR_NOTHING	No error
SDE_ERR_NO_INIT	Sound Library has not been initialized
SDE_ERR_SND_DRV_BUSY	Sound Driver cannot accept the host command

Description

Sends a communication packet (host command) to the ARM7 Sound Driver, buffered by the Sound Library.

Note

Sends the amount of data which can be received by the Sound Driver. Some time is required for the ARM7 Sound Driver to process the data, so the response is delayed slightly (not more than a few milliseconds).

sdSysServer

Server function to register a VSync interrupt.

Format

```
SDE_ERR sdSysServer( Void )
```

Parameters

None

Return Value

SDE_ERR_NOTHING	No error
SDE_ERR_NO_INIT	Sound Library has not been initialized

Description

A server function which is activated on each VSynch.

It carries out the actual communication with the ARM7. It is only possible to transmit a host command from the library to the Sound Driver is this function call is used or the `sdSysFlushHostCmd()` function is called.

Note

Depending on the timing of the execution of the `sdSysFlushHostCmd()` function, the Sound Driver may be parsing commands, and therefore the `sdSysServer()` function may return `SDE_ERR_SND_DRV_BUSY`.

sdSysSetSlotMax

Sets the maximum number of sounds produced from each sound source.

Format

```
SDE_ERR sdSysSetSlotMax( midi_slot_max, shot_slot_max,
pstm_slot_max )
const Sint32 midi_slot_max
const Sint32 shot_slot_max
const Sint32 pstm_slot_max
```

Parameters

midi_slot_max	Maximum sounds from MIDI port
shot_slot_max	Maximum sounds from one-shot port
pstm_slot_max	Maximum sounds from PCM stream port

Return Value

SDE_ERR_NOTHING	No error
SDE_ERR_NO_INIT	Sound Library has not been initialized
SDE_ERR_PRM_OVER_RANGE	Out-of-range parameter value

Description

Sets the maximum number of sounds produced from each sound source.

The standard maximum settings are 48 sounds for a MIDI source, 8 sounds for a one-shot source, and 8 sounds for a PCM stream source.

Note

Regardless of the setting method for a MIDI sound source, the maximum number of handles which can be obtained is eight. The maximum number of handles which can be obtained for a one-shot source is the same. The number of handles which can be obtained for a PCM stream is increased depending on the number of slots used for each port.

10. Structure Functions

SDS_GDDA_STAT

Data types defining GD-DA status.

Format

```
struct SDS_GDDA_STAT
{
       Sint16  rsv;
       Sint8   m_LeftVol;
       Sint8   m_RightVol;
       Sint8   m_LeftPan;
       Sint8   m_RightPan;
       Uint32  m_Err;
       Uint32  m_Flg;
};
typedef struct SDS_GDDA_STAT    SDS_GDDA_STAT;
```

Parameters

`rev`	Reserved
`m_LeftVol`	Left channel volume of current GD-DA
`m_RightVol`	Right channel volume of current GD-DA
`m_LeftPan`	Left channel panpot of current GD-DA
`m_RightPan`	Right channel panpot of current GD-DA
`m_Err`	Error status
`m_Flg`	Various flags

Description

GD-DA status information data types.

Reference

`sdGddaGetStat()`	Gets the GD-DA port status

SDS_MIDI_MES

Sound MIDI message data type.

Format

```
struct SDS_MIDI_MES
{
       Uint8 m_Member[ 0x04];
};
typedef struct SDS_MIDI_MES SDS_MIDI_MES;
```

Parameters

`m_Member`	MIDI Message

Description

MIDI message data type.

Reference

`sdMidiSendMes()`	Sends a MIDI message to a MIDI port
`sdMidiSetMes()`	Creates a MIDI message to send to a MIDI port

SDS_MIDI_STAT

Data types defining MIDI port status.

Format

```
struct SDS_MIDI_STAT
{
       Sint16 rev;
       Sint8 m_Vol;
       Sint8 m_Pan;
       Sint8 m_FxLev;
       Sint8 m_DrctLev;
       Sint16 m_Pitch;
       Sint16 m_Speed;
       Uint32 m_TotalBeatTime;
       Sint32 m_CurAdr;
       Uint32 m_Err;
       Uint32 m_Flg;
};
typedef struct SDS_MIDI_STAT SDS_MIDI_STAT;
```

Parameters

rev	Reserved area
m_Vol	Current port volume
m_Pan	Current port panpot setting
m_FxLev	Current port FX level
m_DrctLev	Current port direct level
m_Pitch	Current port pitch setting
m_Speed	Current port speed setting
m_TotalBeatTime	Total play beat time for sound data playing from current port
m_CurAdr	Current pointer to sound playing from current port
m_Err	Various errors
m_Flg	Various flags

Description

MIDI port status information data types.

The following bits are included in member `m_Err`.

Definition	Meaning
SDD_PORT_ERR_PRIORITY	When the priority of sound data intended to be played is lower than the priority of the currently playing sound data.
SDD_PORT_ERR_REQUEST_NUM	When a sound data play request is received, but cannot be executed (for example, when a bank number or list number that doesn't exist is specified).
SDD_PORT_ERR_DATA_ID_ERR	When the data ID and bank ID of sound data to be played is incorrect.
SDD_PORT_ERR_VER_ERR	When the sound data version is incorrect (too old or too new).
SDD_PORT_ERR_MIDI_BUF_NO_ENOUGH	When the buffer overflows sending a message to the MIDI sound source from the MIDI sequencer or sound library.
SDD_PORT_ERR_MIDI_SEQ_BUF_NO_ENOUGH	When the stack overflows when the MIDI sequencer attempts to send a message to the MIDI sound source, but it cannot be sent because of buffer flow.
SDD_PORT_ERR_SLOT_NO_ENOUGH	When the buffer overflows sending a message to the MIDI sound source from the MIDI sequencer or sound library.
SDD_PORT_FLG_PLAY	Sound data is playing.
SDD_PORT_FLG_PAUSE	Sound data playback is paused.
SDD_PORT_FLG_CHG_VOL	The volume is being set. (During a fade)
SDD_PORT_FLG_CHG_SPEED	The speed is being set. (During a fade)
SDD_PORT_FLG_CHG_PITCH	The pitch is being set. (During a fade)
SDD_PORT_FLG_CHG_PAN	The panpot setting is being changed. (During a fade)
SDD_PORT_FLG_TROUBLE	When an error is generated.

Reference

`sdMidiGetStat()`	Gets MIDI port status

SDS_PSTM_STAT

Data types defining the channel status for a PCM Stream port.

Format

```
struct SDS_PSTM_STAT
{
       Sint16 rev;
       Sint8 m_Vol;
       Sint8 m_Pan;
       Sint8 m_FxLev;
       Sint8 m_DrctLev;
       Sint16 m_Pitch;
       Sint16 m_Speed;
       Uint32 m_TotalSmpFrame;
       Sint32 m_CurAdr;
       Uint32 m_Err;
       Uint32 m_Flg;
};
typedef struct SDS_PSTM_STAT SDS_PSTM_STAT;
```

Parameters

`rev`	Reserved area
`m_Vol`	Current port volume
`m_Pan`	Current port panpot setting
`m_FxLev`	Current port FX level
`m_DrctLev`	Current port direct level
`m_Pitch`	Current port pitch setting
`m_Speed`	Current port speed setting
`m_TotalSmpFrame`	Total play sample frames for sound data playing from current port
`m_CurAdr`	Current pointer to sound playing from current port
`m_Err`	Various errors
`m_Flg`	Various flags

Description

Data types of PCM Stream status information.

The following bits are included in member `m_Err`.

Definition	Meaning
SDD_PORT_ERR_PRIORITY	When the priority of sound data to be played is lower than the priority of the currently playing sound data.
SDD_PORT_ERR_REQUEST_NUM	When a sound data play request is received, but cannot be executed (for example, when a bank number or list number that doesn't exist is specified).
SDD_PORT_ERR_DATA_ID_ERR	When the data ID and bank ID of sound data to be played is incorrect.
SDD_PORT_ERR_VER_ERR	When the sound data version is incorrect (too old or too new).
SDD_PORT_ERR_MIDI_BUF_NO_ENOUGH	When the buffer overflows sending a message to the MIDI sound source from the MIDI sequencer or sound library.
SDD_PORT_ERR_MIDI_SEQ_BUF_NO_ENOUGH	When the stack overflows when the MIDI sequencer attempts to send a message to the MIDI sound source, but it cannot be sent because of buffer overflow.
SDD_PORT_ERR_SLOT_NO_ENOUGH	When the buffer overflows sending a message to the MIDI sound source from the MIDI sequencer or sound library.
SDD_PORT_FLG_PLAY	Sound data is playing.
SDD_PORT_FLG_PAUSE	Sound data playback is paused.
SDD_PORT_FLG_CHG_VOL	The volume is being set. (During a fade)
SDD_PORT_FLG_CHG_SPEED	The speed is being set. (Duringa fade)
SDD_PORT_FLG_CHG_PITCH	The pitch is being set. (During a fade)
SDD_PORT_FLG_CHG_PAN	The panpot setting is being changed. (During a fade)
SDD_PORT_FLG_TROUBLE	When an error is generated.

Reference

`sdPstmGetStat()` Gets PCM Stream port status

SDS_SHOT_STAT

Data types defining one-shot port status.

Format

```
struct SDS_SHOT_STAT
{
       Sint16 rev;
       Sint8 m_Vol;
       Sint8 m_Pan;
       Sint8 m_FxLev;
       Sint8 m_DrctLev;
       Sint16 m_Pitch;
       Sint16 m_Speed;
       Uint32 m_TotalSmpFrame;
       Sint32 m_CurAdr;
       Uint32 m_Err;
       Uint32 m_Flg;
};
typedef struct SDS_SHOT_STAT SDS_SHOT_STAT;
```

Parameters

res	Reserved area
m_Vol	Current port volume
m_Pan	Current port panpot setting
m_FxLev	Current port FX level
m_DrctLev	Current port direct level
m_Pitch	Current port pitch setting
m_Speed	Current port speed setting
m_TotalSmpFrame	Total play sample frames for sound data playing from current port
m_CurAdr	Current pointer to sound playing from current port
m_Err	Various errors
m_Flg	Various flags

Description

One-shot port status information data types.

The following bits are included in member `m_Err`.

Definition	Meaning
SDD_PORT_ERR_PRIORITY	When the priority of sound data intended to be played is lower than the priority of the currently playing sound data.
SDD_PORT_ERR_REQUEST_NUM	When sound data play request is received, but cannot be executed (for example, when a bank number or list number that doesn't exist is specified).
SDD_PORT_ERR_DATA_ID_ERR	When the data ID and bank ID of sound data to be played is incorrect. SDD_PORT_ERR_VER_ERR When the sound data version is incorrect (too old or too new).
SDD_PORT_ERR_SLOT_NO_ENOUGH	When the buffer overflows sending a message to the MIDI sound source from the MIDI sequencer or sound library.
SDD_PORT_FLG_PLAY	Sound data is playing.
SDD_PORT_FLG_CHG_VOL	The volume is being set. (During a fade)
SDD_PORT_FLG_CHG_SPEED	The speed is being set. (During a fade)
SDD_PORT_FLG_CHG_PITCH	The pitch is being set. (During a fade)
SDD_PORT_FLG_CHG_PAN	The panpot setting is being changed. (During a fade)
SDD_PORT_FLG_TROUBLE	When an error is generated.

Reference

`sdShotGetStat()` Gets one-shot port status

SDS_VER

Data types defining versions.

Format

```
struct SDS_VER
{
       Uint8 m_Major;
       Uint8 m_Minor;
       Uint8 m_Debug;
       Uint8 rsv;
};
typedef struct SDS_VER SDS_VER;
```

Parameters

m_MajorVer	Major version number
m_MinorVer	Minor version number
m_DebugVer	Debug version number
rev	Reserved

Description

Version number data types.

These are used mainly to get the driver or library versions. The major version number is incremented when a function is significantly changed. The minor version number is incremented when a slight change is made to the specification or processing speed. The debug version number is incremented each time there is a debugging change.

Reference

sdDrvGetVer()	Gets Sound Driver version

11. Other Data Type Functions

SDD

Constant macros.

Description

The following macro constants are defined in the Sound Library.

Definition	Meaning
SDD_LIB_VER_MAJOR	Sound Library major version number
SDD_LIB_VER_MINOR	Sound Library minor version number
SDD_LIB_VER_DEBUG	Sound Library debug version number
SDD_LIB_VER_STRING	Sound Library version number as a character string
SDD_LIB_VER1_NAME_AVAILABLE	Make names from versions older than Sound Library Ver. 2.0 available
SDD_HOST_CMD_2ND_BUF_NUM	Maximum buffer size for secondary host commands
SDD_MEMBLK_HANDLE_NUM	Maximum number of memory block handles allocated
SDD_MEMBLK_QUEUE_NUM	Maximum size of queue for memory block transfer requests
SDD_MIDI_PORT_NUM	Maximum number of MIDI ports available
SDD_PSTM_PORT_NUM	Maximum number of PCM Stream ports available
SDD_SHOT_PORT_NUM	Maximum number of One Shot ports available
SDD_MIDI_SLOT_NUM	Maximum number of slots available for all MIDI ports
SDD_PSTM_SLOT_NUM	Maximum number of slots available for all PCM Streams
SDD_SHOT_SLOT_NUM	Maximum number of slots available for all One Shot ports
SDD_PSTM_PORT_SLOT_NUM	Number of slots available for a single PCM Stream port
SDD_PSTM_PORT_SLOT_ALL	Total PCM Stream port slots
SDD_PSTM_PORT_SLOT_LEFT	Conventional left slot of PCM Stream port

Definition	Meaning
SDD_PSTM_PORT_SLOT_RIGHT	Conventional right slot of PCM Stream port
SDD_MIDI_CH_NUM	Number of MIDI channels
SDD_MIDI_CH_MIN	Minimum value of MIDI channel
SDD_MIDI_CH_MAX	Maximum value of MIDI channel
SDD_MIDI_NOTE_NUM	Number of MIDI notes
SDD_MIDI_NOTE_MIN	Minimum value of MIDI note
SDD_MIDI_NOTE_MAX	Maximum value of MIDI note
SDD_MIDI_MES_NOTE_OFF	MIDI note off message
SDD_MIDI_MES_NOTE_ON	MIDI on-off message
SDD_MIDI_MES_MONO_PRES	MIDI mono-pressure message
SDD_MIDI_MES_CTL_CHG	MIDI control change message
SDD_MIDI_MES_PRG_CHG	MIDI program change message
SDD_MIDI_MES_POLY_PRES	MIDI poly-pressure message
SDD_MIDI_MES_PITCH_BEND	MIDI pitch bend
SDD_MIDI_CTL_CHG_LSB	MIDI control change LSB offset
SDD_MIDI_CTL_CHG_MSB	MIDI control change MSB offset
SDD_MIDI_CTL_CHG_BANK	MIDI bank change
SDD_MIDI_CTL_CHG_MODULATION	MIDI modulation
SDD_MIDI_CTL_CHG_VOL	MIDI volume
SDD_MIDI_CTL_CHG_PAN	MIDI pan-pot
SDD_MIDI_CTL_CHG_EXPRESSION	MIDI expression
SDD_PORT_ERR_PRIORITY	Repeated attempt to play back material at a lower priority than is already played back by the port
SDD_PORT_ERR_REQUEST_NUM	Bank requested for playback or data corresponding to Bank does not exist
SDD_PORT_ERR_MIDI_BUF_NO_ENOUGH	Buffer for event transfer from MIDI Sequencer to MIDI sound source overflowed
SDD_PORT_ERR_MIDI_SEQ_BUF_NO_ENOUGH	MIDI Event buffer within MIDI Sequencer overflowed
SDD_PORT_ERR_SLOT_NO_ENOUGH	Insufficient slots within sound source
SDD_PORT_ERR_BANK_NOT_FOUND	Bank to be played back not found
SDD_PORT_ERR_BANK_ILLEGAL_ID	Bank ID is illegal
SDD_PORT_ERR_BANK_ILLEGAL_SZ	Bank size is illegal
SDD_PORT_ERR_BANK_ILLEGAL_VER	Bank version is illegal
SDD_DRV_ERR_BANK_NO_DOWNLOAD	Bank is not downloaded
SDD_DRV_ERR_BANK_ILLEGAL_ID	Bank discovered with illegal Bank ID
SDD_DRV_ERR_BANK_ILLEGAL_END_ID	Bank discovered with illegal Bank end ID

Definition	Meaning
SDD_DRV_ERR_BANK_ILLEGAL_VER	Bank version is illegal
SDD_DRV_ERR_BANK_ILLEGAL_NUM	Bank number is illegal

SDE_DATA_TYPE

Sound data types.

Description

The following data types are handled by the Sound Library.

Definition	Meaning
SDE_DATA_TYPE_SND_DRV	Sound Driver
SDE_DATA_TYPE_MULTI_UNIT	Multi Unit
SDE_DATA_TYPE_MIDI_DRUM_BANK	MIDI Drum Bank
SDE_DATA_TYPE_MIDI_SEQ_BANK	MIDI Sequence Bank
SDE_DATA_TYPE_MIDI_PRG_BANK	MIDI Program Bank
SDE_DATA_TYPE_SHOT_BANK	One Shot Bank
SDE_DATA_TYPE_PSTM_RING_BUF	PCM Stream Ring Buffer
SDE_DATA_TYPE_FX_OUT_BANK	FX Output Bank
SDE_DATA_TYPE_FX_PRG_BANK	FX Program Bank
SDE_DATA_TYPE_FX_PRG_WRK	FX Program Work

Note

The relation between the enumeration values and the file extensions is as follows.

Definition	File extension
SDE_DATA_TYPE_SND_DRV	.drv
SDE_DATA_TYPE_MULTI_UNIT	.mlt
SDE_DATA_TYPE_MIDI_SEQ_BANK	.msb
SDE_DATA_TYPE_MIDI_PRG_BANK	.mpb
SDE_DATA_TYPE_SHOT_BANK	.osb
SDE_DATA_TYPE_FX_OUT_BANK	.fob
SDE_DATA_TYPE_FX_PRG_BANK	.fpb

SDE_ERR

Sound Library errors.

Description

The error status of each Sound function is explained below.

Definition	Meaning
SDE_ERR_ALREADY_INIT	Already initialized
SDE_ERR_BANK_NOTHING	Bank does not exist
SDE_ERR_BANK_ILLEGAL_TYPE	Bank type is illegal
SDE_ERR_BANK_ILLEGAL_VER	Bank version is illegal
SDE_ERR_DATA_ILLEGAL_NUM	Data item number is illegal
SDE_ERR_DATA_ILLEGAL_TYPE	Data type is illegal (Data (file) may be corrupted)
SDE_ERR_DATA_ILLEGAL_VER	Data version is illegal
SDE_ERR_DATA_NOTHING	Data does not exist
SDE_ERR_GD_FS_LIB_ERR	Error occurred in GF-FS library access within the Sound Library
SDE_ERR_HANDLE_ILLEGAL_VALUE	Handle points to an illegalm address
SDE_ERR_HANDLE_NO_ENOUGH	Handle cannot be obtained because of insufficient buffer
SDE_ERR_HANDLE_NULL	Handle is NULL
SDE_ERR_HARD_WARE	Hardware error
SDE_ERR_HOST_CMD_BUF_NO_ENOUGH	Insufficient buffer for host command
SDE_ERR_MAIN_MEM_ADR_ERR	Address error in main memory
SDE_ERR_MEMBLK_ALREADY_TRANSFER	Attempt to transfer memory block which is being transferred (or already queued)
SDE_ERR_MEMBLK_QUEUE_NO_ENOUGH	Insufficient memory block transfer queue
SDE_ERR_MODULE_SLOT_NO_ENOUGH	Insufficient sound source slots
SDE_ERR_NO_INIT	Sound Library is not initialized
SDE_ERR_NOTHING	No error
SDE_ERR_PRM_ILLEGAL_VALUE	Parameter value is illegal
SDE_ERR_PRM_OVER_RANGE	Out-of-range prameter value
SDE_ERR_PTR_ILLEGAL_VALUE	Pointer value is illegal
SDE_ERR_PTR_NULL	Address is NULL
SDE_ERR_SND_DRV_BUSY	Sound Driver has rejected host command
SDE_ERR_SND_DRV_ILLEGAL_VER	Sound Driver version is wrong
SDE_ERR_SND_DRV_PROBLEM	Sound Driver operation problrem
SDE_ERR_SND_MEM_ADR_ERR	Address error in sound memory
SDE_ERR_UNKNOW_NUM	Unknown error

Reference

`sdDrvCheckExecute()`	Checks execution of Sound Driver
`sdDrvDownloadFromFile()`	Reads in and initializes Sound Driver
`sdDrvGetErr()`	Gets Sound Driver error information
`sdDrvGetExecuteCounter()`	Gets Sound Driver execution counter
`sdDrvGetVer()`	Gets Sound Driver version
`sdDrvInit()`	Initializes Sound Driver
`sdLibGetVer()`	Gets Sound Library version
`sdLibInit()`	Initializes Sound Library
`sdSysFinish()`	Closes the Sound Driver
`sdSysFlushHostCmd()`	Sends a host command
`sdSysServer()`	Server function to register a VSync interrupt
`sdSysSetSlotMax()`	Sets the maximum number of sounds produced from each sound source
`sdShotClosePort()`	Releases one-shot port access permission
`sdShotGetCurAdr()`	Gets the current play address of a one-shot sample frame
`sdShotGetStat()`	Gets one-shot port status
`sdShotGetTotalSmpFrame()`	Gets total count of one-shot play sample frames
`sdShotOpenPort()`	Gets one-shot port access permission
`sdShotPlay()`	Plays one-shot data
`sdShotResetAllPrm()`	Resets parameters of all one-shot ports
`sdShotResetPrm()`	Resets a one-shot port
`sdShotSetDrctLev()`	Sets the direct level of a one-shot port
`sdShotSetFxCh()`	Sets the FX input channel number for a one-shot port
`sdShotSetFxLev()`	Sets the FX input level for a one-shot port
`sdShotSetPan()`	Sets the panpot value for a one-shot port
`sdShotSetPitch()`	Sets the pitch of a one-shot port
`sdShotSetPlayTime()`	Sets playback time for one-shot playback
`sdShotSetVol()`	Sets the volume of a one-shot port
`sdShotStop()`	Stops one-shot playback
`sdShotStopAll()`	Stops all one-shot playback
`sdPstmClosePort()`	Releases PCM Stream port access permission
`sdPstmGetCurAdr()`	Gets the current play address of a PCM Stream
`sdPstmGetStat()`	Gets PCM Stream port status
`sdPstmGetTotalSmpFrame()`	Gets total PCM Stream play sample frames
`sdPstmIsTransferWaveData()`	Checks if okay to transfer a sample frame to a PCM Stream port
`sdPstmOnMemPlay()`	Plays on-memory PCM Stream
`sdPstmOnMemSetWaveData()`	Sets on-memory PCM Stream
`sdPstmOpenPort()`	Gets PCM Stream port access permission
`sdPstmPlay()`	Plays a PCM Stream

`sdPstmResetAllPrm()`	Resets parameters of all PCM Stream ports
`sdPstmResetPrm()`	Resets a PCM Stream port
`sdPstmSetBasePrm()`	Sets base parameters for a PCM Stream port
`sdPstmSetDrctLev()`	Sets the direct level of a PCM Stream port
`sdPstmSetFxCh()`	Sets the FX input channel number for a PCM Stream port
`sdPstmSetFxLev()`	Sets the FX input level for a PCM Stream port
`sdPstmSetPan()`	Sets the panpot setting of a PCM Stream port
`sdPstmSetPitch()`	Sets the playback pitch of a PCM Stream port
`sdPstmSetVol()`	Sets the playback volume of a PCM Stream port
`sdPstmStop()`	Stops PCM Stream playback
`sdPstmStopAll()`	Stops playback on all PCM Streams
`sdPstmTransferWaveData()`	Transfers sample data to a PCM Stream port
`sdMidiClosePort()`	Releases MIDI port access permission
`sdMidiContinue()`	Resumes playback of a paused MIDI sequence
`sdMidiGetCurAdr()`	Gets the current play address of a MIDI sequence
`sdMidiGetStat()`	Gets MIDI port status
`sdMidiGetTotalBeatTime()`	Gets the current playing beat time of a MIDI sequence
`sdMidiOpenPort()`	Gets MIDI port access permission
`sdMidiPause()`	Pauses MIDI sequence playback
`sdMidiPlay()`	Plays a MIDI sequence
`sdMidiResetAllPrm()`	Resets parameters of all MIDI ports
`sdMidiResetPrm()`	Resets a MIDI port
`sdMidiSendMes()`	Sends a MIDI message to a MIDI port
`sdMidiSetDrctLev()`	Sets the direct level of a MIDI port
`sdMidiSetFxLev()`	Sets the FX level of a MIDI port
`sdMidiStop()`	Stops playback on a MIDI port
`sdMidiSetMes()`	Creates a MIDI message to send to a MIDI port
`sdMidiSetPan()`	Sets the panpot of a MIDI port
`sdMidiSetPitch()`	Sets the playback pitch of a MIDI port
`sdMidiSetSpeed()`	Sets the playback speed of a MIDI port
`sdMidiSetVol()`	Sets the volume of a MIDI port
`sdMidiStopAll()`	Stops playback of all MIDI sequence data
`sdGddaGetStat()`	Gets the GD-DA port status
`sdGddaResetPrm()`	Resets GD-DA port parameters
`sdGddaSetPan()`	Sets the GD-DA port panpot levels
`sdGddaSetVol()`	Sets the GD-DA port volume
`sdSndMemGetBankStat()`	Gets the bank status of Sound Memory
`sdMemBlkCreate()`	Gets a Memory Block handle
`sdMemBlkDestroy()`	Destroys a Memory Block handle
`sdMemBlkGetStat()`	Gets the status of a Memory Block

`sdMemBlkSetPrm()`	Sets the parameters for a Memory Block handle
`sdMemBlkSetTransferMode()`	Sets the transfer mode for Memory Blocks
`sdQsndSetPos()`	Sets Q Sound position
`sdSndClearFxPrg()`	Clears FX program
`sdSndGetFxOut()`	Gets current FX output data number
`sdSndGetFxPrg()`	Gets current FX program data number
`sdSndSetFxOut()`	Sets FX output data
`sdSndSetFxPrg()`	Sets FX program data
`sdSndSetMasterVol()`	Sets the master volume
`sdSndSetPanMode()`	Sets the pan mode
`sdSndStopAll()`	Stops all sound data playback
`sdBankDownload()`	Transfers a bank
`sdBankDownloadFromFile()`	Reads in and transfers bank file
`sdMultiUnitDownload()`	Transfers a multi-unit file
`sdMultiUnitDownloadFromFile()`	Reads and transfers a multi-unit file

SDE_MEMBLK_TRANSFER_MODE Memory Block transfer modes

Description

Shows the memory block transfer mode. The setting is made with the `sdMemBlkSetTransferMode()` function.

Definition	Meaning
SDE_MEMBLK_TRANSFER_MODE_CPU	CPU transfer
SDE_MEMBLK_TRANSFER_MODE_DMA	DMA transfer

Note

DMA transfer provides an immediate-return function. During the transfer requests from the `sdMemBlkTransfer()` function are queued.

If carrying out continuous transfers, be careful not to request duplicate transfers. A duplicate transfer request means that during a transfer an attempt is made to transfer with the same memory block handle. If there is a possibility of this occurring, obtain multiple handles for the memory block transfer. The `sdMemBlkGetStat()` function can be used to check whether a memory block is being transferred.

SDE_MIDI_GM_MODE

GM mode for MIDI port.

Description

Used to set/release GM sound generator mode for MIDI port.

Definition	Meaning
SDE_MIDI_GM_MODE_ON	Valid
SDE_MIDI_GM_MODE_OFF	Invalid

SDE_PAN_MODE

Panpot mode.

Description

Enables or disables the panpot specified for all sound data and functions.

The panpot modes are defined as follows.

Definition	Meaning
SDE_PAN_MODE_MONO	Invalid
SDE_PAN_MODE_STEREO	Valid
SDE_PAN_MODE_DISABLE	Invalid (same asSDE_PAN_MODE_MONO)
SDE_PAN_MODE_ENABLE	Valid (same as SDE_PAN_MODE_STEREO)

Note

Even with the panpot disabled, the volume balance may not give the expected value. This is because the volume balance is affected by the original sound data panpot setting and the values of various functions that make panpot settings. For example, the output volume will be greater when the panpot setting is to the right or left than when it is in the center.

Reference

`sdSndSetPanMode()` Sets the pan mode

SDE_PCM_TYPE

PCM encoding format.

Description

Used to specify the PCM encoding format.

Definition	Meaning
SDE_PCM_TYPE_4BIT_ADPCM	4-bit ADPCM compressed format
SDE_PCM_TYPE_8BIT_LINEAR	8-bit PCM uncompressed format
SDE_PCM_TYPE_16BIT_LINEAR	16-bit PCM uncompressed format

Example

```
SDMIDI midi_handle;
sdMidiOpenPort( &midi_handle);
sdMidiSetGmMode( midi_handle, SDE_PCM_TYPE_ON);
```

Reference

sdPstmOnMemPlay()	Plays on-memory PCM Stream
sdPstmPlay()	Plays a PCM Stream

SDE_SHOT_PLAY_TIME

Method of specifying one-shot playback time.

Description

Indicates the method of specifying a one-shot playback time.

Note

Definition	Meaning
SDE_SHOT_PLAY_TIME_RELATIVE	Relative value
SDE_SHOT_PLAY_TIME_RATIO	Ratio
SDE_SHOT_PLAY_TIME_ABSOLUTE	Absolute value

If a value other than SDE_SHOT_PLAY_TIME_ABSOLUTE is pecified, the playback time is affected by the playback time parameter within the one-shot data.

When SDE_SHOT_PLAY_TIME_ABSOLUTE is specified, the effect is applied ignoring the playback time parameter within the one-shot data.

Reference

sdShotSetPlayTime()	Sets playback time for one-shot playback