

*Dreamcast
Shinobi Library
Specification
and Dreamcast
Menu Screens*

Table of Contents

1. System Functions	SLS-1
sbExitSystem	Exits the Shinobi system.SLS-2
syHwFinish	Hardware exit processing.SLS-3
syHwInit	Initialization processing that should be performed before Ninja initialization.SLS-4
syHwInit2	Initialization processing that must be performed after Ninja initialization.SLS-5
sbInitSystem	Initializes the Shinobi library.SLS-6
2. File System Functions	SLS-9
gdFsCalcSctSize	Calculates the number of sectors based on the number of bytes. (macro).SLS-10
gdFsChangeDir	Changes the current directory.SLS-11
gdFsCheckEof	Checks whether the end of the file has been reached.SLS-12
gdFsClose	Closes a file.SLS-13
gdFsCreateDirhn	Generates a directory handle.SLS-14
gdFsDaPause	Pauses DA playback.SLS-15
gdFsDaPlay	Plays back a DA track.SLS-16
gdFsDaPlaySct	Plays back a specified DA sector.SLS-17
gdFsDaRelease	Releases DA playback from the paused state.SLS-18
gdFsDaStop	Stops DA playback.SLS-19
gdFsEntryErrFunc	Registers the callback function for when an error occurs.SLS-20
gdFsEntryRdEndFunc	Registers the callback function for when a read ends.SLS-21
gdFsEntryTrEndFunc	Registers the callback function for when a transfer ends.SLS-22
gdFsFinish	Exits the library.SLS-23
gdFsGetDirInfo	Gets the file information.SLS-24
gdFsGetDirrecSize	Gets the directory record buffer size (macro).SLS-25
gdFsGetDrvStat	Gets the drive status.SLS-27
gdFsGetErrStat	Gets the file error status.SLS-29
gdFsGetFileFad	Gets the file FAD.SLS-30
gdFsGetFileSctSize	Gets the file sector size.SLS-31
gdFsGetFileSize	Gets the file size.SLS-32
gdFsGetNumRd	Gets the size of the file that has already been read.SLS-33

gdFsGetStat.....	Gets the file status.	SLS-34
gdFsGetSysHn.....	Gets the system handle.	SLS-35
gdFsGetToc.....	Gets the TOC.	SLS-36
gdFsGetTransStat.....	Gets the transfer status.	SLS-37
gdFsGetWorkHn.....	Gets the handle for the task currently being executed.	SLS-38
gdFsGetWorkSize	Gets the library work size (macro).	SLS-39
gdFsInit.....	Initializes the library.	SLS-40
gdFsLoadDir.....	Loads a directory.	SLS-42
gdFsMovePickup	Moves the pickup.	SLS-43
gdFsOpen.....	Opens a file.	SLS-44
gdFsOpenRange.....	Opens specified sectors in a file.	SLS-45
gdFsRead.....	Reads a file.	SLS-46
gdFsReinit	Reinitializes the library.	SLS-47
gdFsReqDrvStat	Updates the drive status.	SLS-48
gdFsReqGdRd	Issues read-ahead request to the GD buffer.	SLS-49
gdFsReqRd32.....	Reads a file.	SLS-50
gdFsSeek.....	Changes the file reading position.	SLS-51
gdFsSetDir.....	Sets the current directory.	SLS-52
gdFsStopRd.....	Stops the reading of a file.	SLS-53
gdFsTell.....	Gets the next file reading position.	SLS-54
gdFsTrans32.....	Specifies transfer from the GD buffer.	SLS-55

3. Memory Management Function SLS-57

syFree.....	Releases memory. (macro).	SLS-58
syMalloc	Allocates memory.	SLS-59
syMallocFinish	Exits the Malloc system.	SLS-60
syMallocInit	Allocates and declares the memory area to be used by the memory management function.	SLS-61
syMallocStat.....	Gets the free memory size and the maximum contiguous memory size that can be obtained.	SLS-62

4. Cache Functions SLS-63

syCacheICI.....	Invalidates all caches. (Macro.)	SLS-64
syCacheInit	Initializes the functions. (Macro.)	SLS-65
syCacheMOVCA.....	Executes MOVCA.L instruction.	SLS-66
syCachePREF.....	Operand cache pre-fetch.	SLS-67
syCacheOCI_P2.....	Invalidates all entries in the operand cache.	SLS-68
syCacheOCBI.....	Invalidates the operand cache block.	SLS-69
syCacheOCBP.....	Purges the operand cache block.	SLS-70
syCacheOCWB	Writes back to the operand cache block.	SLS-71

5. Get Peripheral Data Function SLS-73

pdExecPeripheralServer	Server functions for getting peripheral data.	SLS-74
pdGetPeripheral.....	Gets peripheral data.	SLS-75
pdGetPeripheralError	Gets the peripheral status.	SLS-76
pdGetPeripheralInfo.....	Gets peripheral information.	SLS-77
pdTmrAlarm	Gets peripheral information.	SLS-79

6. LCD Functions SLS-81

pdLcdGetDirection.....	Gets the LCD orientation.	SLS-82
pdVmdLcdIsReady	Checks whether VMS is connected or not.	SLS-84
pdVmsLcdWrite.....	Displays data on the VMS LCD.	SLS-85

7. Timer Functions SLS-87

syTmrCountToMicro	Converts a count value to μ seconds.	SLS-88
syTmrDiffCount.....	Gets the difference between two timer count values.	SLS-89
syTmrGenCancelInt	Disables timer interrupts in response to the general-purpose timer.	SLS-90
syTmrGetCount	Gets the value of the free-running timer.	SLS-91
syTmrGenCountToMicro	Converts the general-purpose counter value to μ seconds.	SLS-92
syTmrGenDiffCount	Calculates the difference in the general-purpose interrupt count.	SLS-93
syTmrGenGetCount	Gets the general-purpose timer interrupt counter.	SLS-94
syTmrGenSetClock.....	Sets the general-purpose timer interrupt counter.	SLS-95
syTmrGenSetInt	Enables timer interrupts in response to the general-purpose timer.	SLS-96
syTmrGenStart	Starts the general-purpose timer.	SLS-97
syTmrGenStop	Stops the general-purpose timer.	SLS-98
syTmrGenMicroToCount	Converts μ seconds to a counter value.	SLS-99
syTmrGenSetCount	Sets the general-purpose timer interrupt counter.	SLS-100
syTmrMicroToCount	Converts μ seconds to a counter value.	SLS-101

8. Real-time Clock Functions SLS-103

syRtcCompareDate.....	Compares the date and time.	SLS-104
syRtcDateToCount	Converts from date and time to count.	SLS-106
syRtcExecServer.....	Server functions.	SLS-107
syRtcFinish.....	Exits the library.	SLS-108
syRtcGetDate.....	Gets the current date and time.	SLS-109
syRtcGetStat	Gets the server function status.	SLS-111
syRtcInit	Initializes the library.	SLS-112
syRtcSetDate.....	Sets the date and time.	SLS-113
syRtcSetServerMode	Sets the server function operation.	SLS-114

9. Backup Functions SLS-115

buAnalyzeBackupFileImage	Analyzes the file image in memory.	SLS-116
buCalcBackupFileSize.....	Calculates the block size of the backup file image.	SLS-118
buDefragDisk	Allocates consecutive free blocks.	SLS-120
buDeleteFile.....	Deletes a file.	SLS-122
buExit.....	Exits the library.	SLS-124
buFindExecFile.....	Gets the name of an executable file.	SLS-125
buFindFirstFile.....	Gets the name of the first file.	SLS-127
buFindNextFile	Gets the name of the next file.	SLS-129
buFormatDisk.....	Formats backup RAM.	SLS-131
buGetDiskInfo	Gets backup RAM information.	SLS-133
buGetDiskFree	Checks the free space.	SLS-134
buGetFileInfo.....	Gets information on the specified file.	SLS-135
buGetFileSize.....	Checks the size of the specified file.	SLS-136

buGetLastError.....	Gets the last error that was generated.	SLS-137
buInit.....	Initializes the library.	SLS-138
buIsReady	Checks whether backup RAM is connected or not.	SLS-140
buIsFormat.....	Checks whether backup RAM has been formatted or not.	SLS-141
buIsExistFile	Checks whether the specified file exists or not.	SLS-142
buLoadFile	Loads a file.	SLS-143
buMakeBackupFileImage	Creates a backup file format memory image.	SLS-145
buLoadFileEx.....	Loads a file with the load start block and the number of blocks specified.	SLS-147
buMountDisk	Mounts a drive.	SLS-149
buSaveExecFile.....	Saves an executable file.	SLS-150
buSaveFile	Saves a file.	SLS-152
buSetCompleteCallback.....	Registers a function that is called back when processing is completed.	SLS-154
buSetFileAttr.....	Changes the file attributes.	SLS-155
buSetProgressCallback.....	Registers a function that is called back when processing progresses to a certain point.	SLS-156
buStat	Checks whether processing is complete or not.	SLS-157
buUnmount	Unmounts a drive.	SLS-158
syCblCheckCable	Gets the video cable type.	SLS-160

11. Boot ROM Font Functions SLS-163

syBtFntChkSmph	Checks the font semaphore.	SLS-164
syBtFntClrSmph.....	Clears the font semaphore.	SLS-165
syBtFntGetAddr	Gets offset for target font data from Shift JIS codes.	SLS-166
syBtFntGet	Gets starting address of font data in boot ROM.	SLS-167
syBtFntGetInfo	Gets size information for target font data from Shift JIS codes.	SLS-169
syBtFntSjis2Jis.....	Converts Shift JIS codes to JIS codes.	SLS-170

12. Configuration Functions SLS-171

syCfgExit	Exits the configuration functions.	SLS-172
syCfgGetSoundMode	Gets the sound setting.	SLS-173
syCfgInit	Initializes the configuration functions.	SLS-174
syCfgSetSoundMode.....	Updates the sound setting.	SLS-175

13. Boot ROM Service Functions SLS-177

syBtCheckDisc.....	Checks for disk replacement.	SLS-178
syBtExit.....	Returns to the boot ROM GUI screen.	SLS-179

14. Sound System API SLS-181

sdDrvGetExecuteCounter.....	Get Sound Driver execution counter.	SLS-182
sdDrvGetMidiTimingCounter	Get count number of special messages counted by sound driver.	SLS-183
sdDrvInit	Initialize Sound Driver.	SLS-185
sdLibGetVer.....	Get Sound Library version.	SLS-186
sdLibInit	Initialize Sound Library.	SLS-187
sdSysFlushHostCmd	Send Host Command immediately to ARM7 Sound Driver.	SLS-188
sdSysServer	Server function to register V interrupt.	SLS-189

15. Global Sound Control API SLS-191

sdMidiClosePort	Releases MIDI port access permission.	SLS-192
sdMidiContinue	Resumes MIDI sequence play.	SLS-193
sdMidiGetCurAdr	Gets the current play address of a MIDI sequence.	SLS-194
sdMidiGetStat	Gets MIDI port status.	SLS-195
sdMidiGetTotalBeatTime	Gets the total beat time of a MIDI sequence.	SLS-196
sdMidiOpenPort	Gets MIDI port access permission.	SLS-197
sdMidiPause	Pauses MIDI sequence play.	SLS-198
sdMidiPlay	Plays a MIDI sequence.	SLS-199
sdMidiResetAllPrm	Resets all MIDI ports.	SLS-200
sdMidiResetPrm	Resets a MIDI port.	SLS-201
sdMidiSendMes	Sends a MIDI message to a MIDI port.	SLS-202
sdMidiSetDrctLev	Sets the direct level of a MIDI port.	SLS-203
sdMidiSetFxLev	Sets the FX level of a MIDI port.	SLS-204
sdMidiSetMes	Creates a MIDI message to send to a MIDI port.	SLS-205
sdMidiSetPan	Sets the panpot (sound localization) of a MIDI port.	SLS-206
sdMidiSetPitch	Sets the play pitch of a MIDI port.	SLS-207
sdMidiSetSpeed	Sets the play speed of a MIDI port.	SLS-208
sdMidiStop	Stops MIDI sequence play.	SLS-209
sdMidiStopAll	Stops play on all MIDI ports.	SLS-210
sdMidiSetVol	Sets the play volume of a MIDI port.	SLS-211
sdPstmClosePort	Releases PCM Stream port access permission.	SLS-212
sdPstmGetCurAdr	Gets the current play address of a PCM Stream.	SLS-213
sdPstmGetStat	Gets PCM Stream port status.	SLS-214
sdPstmGetTotalSmpFrame	Gets total PCM Stream play sample frames.	SLS-215
sdPstmIsTransferWaveData	Check if okay to transfer a waveform to a PCM Stream port.	SLS-216
sdPstmOpenPort	Gets PCM Stream port access permission.	SLS-217
sdPstmPlay	Plays a PCM Stream.	SLS-218
sdPstmResetAllPrm	Resets all PCM Stream ports.	SLS-219
sdPstmResetPrm	Resets a PCM Stream port.	SLS-220
sdPstmSetBaseVol	Sets the base volume of a PCM Stream port.	SLS-221
sdPstmSetDrctLev	Sets the direct level of a PCM Stream port.	SLS-222
sdPstmSetFxCh	Sets the FX input for a PCM Stream port.	SLS-223
sdPstmSetFxLev	Sets the FX level of a PCM Stream port.	SLS-224
sdPstmSetPan	Sets the panpot of a PCM Stream port.	SLS-225
sdPstmSetPitch	Sets the play pitch of a PCM Stream port.	SLS-226
sdPstmSetSpeed	Sets the play speed of a PCM Stream port.	SLS-227
sdPstmSetVol	Sets the play volume of a PCM Stream port.	SLS-228
sdPstmStop	Stops PCM Stream play.	SLS-229
sdPstmStopAll	Stops play on all PCM Stream ports.	SLS-230
sdPstmTransferWaveData	Transfers a waveform to a PCM Stream port.	SLS-231
sdShotClosePort	Releases One-Shot port access permission.	SLS-232
sdShotGetCurAdr	Gets the current play address of a One-Shot.	SLS-233
sdShotSetStat	Gets One-Shot port status.	SLS-234
sdShotGetTotalSmpFrame	Gets total One-Shot play sample frames.	SLS-235
sdShotOpenPort	Gets One-Shot port access permission.	SLS-236
sdShotPlay	Plays a One-Shot.	SLS-237
sdShotResetAllPrm	Resets all One-Shot ports.	SLS-238
sdShotResetPrm	Resets a One-Shot port.	SLS-239
sdShotSetDrctLev	Sets the direct level of a One-Shot port.	SLS-240
sdShotSetFxCh	Sets the FX output destination for a One-Shot port.	SLS-241
sdShotSetFxLev	Sets the FX level of a One-Shot port.	SLS-242
sdShotSetPan	Sets the panpot of a One-Shot port.	SLS-243

sdShotSetPitch	Sets the play pitch of a One-Shot port.	SLS-244
sdShotSetSpeed	Sets the play speed of a One-Shot port.	SLS-245
sdShotSetVol.....	Sets the play volume of a One-Shot port.	SLS-246
sdShotStop	Stops One-Shot play.	SLS-247
sdShotStopAll.....	Stops play on all One-Shot ports.	SLS-248

16. Sound Data Utility API SLS-249

sdBankDownload	Downloads various sound data from main memory to sound memory. (Macro.)	SLS-250
sdMultiUnitDownload.....	Downloads a Multi-Unit file from main memory to sound memory.	SLS-251

17. Memory Block Transfer API SLS-253

sdMemBlkCreate.....	Creates a Memory Block handle.	SLS-254
sdMemBlkDestroy	Destroys a Memory Block handle.	SLS-255
sdMemBlkGetStat	Checks the status of a Memory Block handle.	SLS-256
sdMemBlkSetPrm	Sets Memory Block handle parameters.	SLS-257
sdMemBlkSetTransferMode.....	Sets transfer mode for a Memory Block handle.	SLS-258

18. Dreamcast Middleware Guide SLS-259

Purpose	SLS-259
Modular Structure	SLS-260

Middleware Library Internal Structure SLS-260

Middleware Usage Methods SLS-261

Initializing the middleware library	SLS-261
Middleware playback handle	SLS-262
How to Use the Middleware Library	SLS-263
Operational states of the middleware playback handle	SLS-263
Playback of MPEG Sofdec	SLS-264
Playback of WAVE	SLS-266
Playback of TrueMotion	SLS-267
Playback of MPEG / Audio	SLS-268

Data Specifications SLS-269

Constants	SLS-269
Data Types	SLS-271

Function Specifications SLS-274

Functions for MPEG Sofdec	SLS-276
Functions for WAVE	SLS-284
Functions for TrueMotion	SLS-285
Functions for MPEG / Audio	SLS-286
Global functions	SLS-287

Appendix SLS-293

Door Status Check	SLS-293
Concurrent playback of MPEG Sofdec and ADX	SLS-294

19. Dreamcast Middleware Outline	SLS-295
Introduction	SLS-295
Middleware Library	SLS-297
Software Configuration	SLS-297
Common Header Files and Library Files	SLS-297
CRI MPEG Sofdec Outline	SLS-298
Features	SLS-298
Software Configuration	SLS-298
Data Creation Tools	SLS-298
Header Files and Library Files	SLS-299
CRI ADX Outline	SLS-299
Features	SLS-299
Software Configuration	SLS-299
Data Creation Tools	SLS-300
Header Files and Library Files	SLS-300
Joint Use With MPEG SOFDEC	SLS-300
Multistream playback with CODEC	SLS-300
WAV Decoder Outline	SLS-301
Features	SLS-301
Software Configuration	SLS-301
Data Creation Tools	SLS-301
Header Files and Library Files	SLS-301
TrueMotion Outline	SLS-302
Features	SLS-302
Software Configuration	SLS-302
Data Creation Tools	SLS-302
Header File and Library File	SLS-302
MPEG/Audio Outline	SLS-303
Features	SLS-303
Software Configuration	SLS-303
Data Creation Tools	SLS-303
Header Files and Library Files	SLS-303
 20. ADX Playback Library Implementation Manual	 SLS-305
Introduction	SLS-305
ADX Playback System	SLS-307
System Configuration	SLS-307
ADX File System	SLS-309
Module Configuration	SLS-310
Application Interface (API)	SLS-311
ADX Data Creation	SLS-312

Using the Library Functions	SLS-314
Library Initialization and Shutdown	SLS-314
ADXT Handle Creation	SLS-314
Playback of ADX Data in Memory (Memory Playback)	SLS-315
Playback of ACX Data in Memory (Memory Index Playback)	SLS-315
Playback of ADX Data on CD (CD Stream Playback)	SLS-316
Playback of AFS Data on CD (CD Index Stream Playback)	SLS-316
Get Playback Status	SLS-317
Table of Library Functions	SLS-319
Description of Library Functions	SLS-320
ADX Library	SLS-339
Sound Data Creation Precautions	SLS-340
Increasing Data Reading Speed	SLS-340
How to Reduce Sound Seek	SLS-342
Concurrent playback with MPEG Sofdec	SLS-343
Joint Use with GDFS/NINJA Library	SLS-344
Sample Programs	SLS-344
A. Gun Controller Appendix	SLS-345
Overview	SLS-345
Details of Coordinate Compensation	SLS-345
Fine Adjustments with Coordinate Compensation	SLS-348
Gun Controller Sample # Continuous Shooting	SLS-348
B. Peripheral Data Acquisition Function Group Gun Controller	SLS-349
Overview	SLS-349
Gun Controller Specifications	SLS-349
Gun Mode	SLS-349
Limitations with Extension Peripherals	SLS-349
Screen Flash	SLS-350
Gun Priority Order	SLS-350
Function Specifications	SLS-351
Function List	SLS-351
Function API	SLS-352
pdGunEnter	Enters gun mode. SLS-352
pdGunLeave	Exits gun mode. SLS-353
pdGunGetLatchedPort	Gets control ports where coordinates were detected. SLS-354
pdGunGetPosition	Gets gun coordinates. SLS-355
pdGunSetFlashColor	Sets screen flash color. SLS-356
pdGunSetCallback	Registers call back for trigger information setting. SLS-357
pdGunSetTrigger	Sets trigger information. SLS-358

Dreamcast Menus

1. Main Menu	EMS-1
The Main Menu	EMS-1
2. Memory Card Selection Screen	EMS-3
Memory Card Selection Screen	EMS-3
3. Save Data	EMS-5
File Menu Screen	EMS-5
4. Game Names and Sorting	EMS-11
Game Names and Character Maps	EMS-11
Examples of Japanese Language Titles	EMS-12
5. Format Settings	EMS-13
Body Color	EMS-13
Formatting Visual Memory	EMS-14
6. CD Playback Screen	EMS-17
CD Playback Screen	EMS-17
Changing the CD/GD Texture	EMS-18
7. Setting Screen	EMS-19
Set Icon from the Settings Screen	EMS-19

1. System Functions

sbExitSystem

Exits the Shinobi system.

Format

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

Parameters

None

Outputs

None

Return value

None

Operation

This function exits the system.

Execute this function in place of njExitSystem.

syHwFinish

Hardware exit processing.

Format

```
#include sg_syhw.h  
void syHwFinish(void)
```

Parameters

None

Outputs

None

Return value

None

Operation

This function releases SH CPU resources and performs the hardware exit processing.

syHwInit

Initialization processing that should be performed before Ninja initialization.

Format

```
#include sg_syhw.h
void syHwInit(void);
```

Parameters

None

Outputs

None

Return value

None

Operation

This function must be executed before Ninja initialization. This function mainly initializes the hardware.

The following initializations are performed by this function:

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

Notes

This function is executed within sbInitSystem.

syHwInit2

Initialization processing that must be performed after Ninja initialization.

Format

```
#include sg_syhw.h  
void syHwInit2(void);
```

Inputs

None

Outputs

None

Return value

None

Function

This function performs library initialization, which must be performed after Ninja/Kamui initialization.

This function also initializes the interrupt controller.

sbInitSystem

Initializes the Shinobi library.

Format

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

Inputs

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.

Outputs

None

Return value

None

Function

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 from -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.

Remarks

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.

2. File System Functions

gdFsCalcSctSize

Calculates the number of sectors based on the number of bytes. (macro).

SYNOPSIS

```
#include <sg_gd.h>
gdFsCalcSctSize(bytes)
```

ARGUMENTS

bytes	File byte size
--------------	----------------

RETURN

Sector size

DESCRIPTION

This macro 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);
```

gdFsChangeDir

Changes the current directory.

Format

```
#include <sg_gd.h>
Sint32 gdFsChangeDir(const char *dirname)
```

Inputs

dirname	Directory name
----------------	----------------

Outputs

None

Return value

Error codes

Function

This function changes the current directory.

Example

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

Notes

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

gdFsCheckEof Checks whether the end of the file has been reached.

Format

```
#include <sg_gd.h>
Sint32 gdFsCheckEof(GDFS gdfs, Bool *iseof)
```

Inputs

gdfs	File handle
-------------	-------------

Outputs

iseof	
0:	Not the end of file
Other than 0:	End of file

Return value

Error codes

Function

This function checks whether the end of the file has been reached.

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;
}
```

Notes

gdFsClose

Closes a file.

Format

```
#include <sg_gd.h>
void gdFsClose(GDFS gdfs);
```

Inputs

gdfs	File handle
-------------	-------------

Outputs

None

Return value

None

Function

This function closes a file.

Example

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

Notes

gdFsCreateDirhn

Generates a directory handle.

Format

```
#include <sg_gd.h>
GDFS_DIRREC gdFsCreateDirhn(void *dirbuf, Sint32 max_dirent)
```

Inputs

dirbuf	Directory buffer
max_dirent	Number of directory entries

Outputs

None

Return value

Directory record handle

Function

This function generates the directory handle.

Example

```
Uint32 dirbufgdFsGetDirrecSize(64);
GDFS_DIRREC g_dir;
g_dir = gdFsCreateDirhn(dirbuf, 64);
```

Notes

gdFsDaPause

Pauses DA playback.

Format

```
#include <sg_gd.h>
Sint32 gdFsDaPause(void)
```

Inputs

None

Outputs

None

Return value

Error codes

Function

This function pauses DA playback.

Example

```
gdFsDaPause( ) ;
```

Notes

gdFsDaPlay

Plays back a DA track.

Format

```
#include <sg_gd.h>
Sint32 gdFsDaPlay(Sint32 st_track, Sint32 end_track, Sint32 reptime)
```

Inputs

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

Outputs

None

Return value

Error codes

Function

This function starts DA playback.

Example

```
gdFsDaPlay(4, 4, 0);
```

Notes

gdFsDaPlaySct

Plays back a specified DA sector.

Format

```
#include <sg_gd.h>
Sint32 gdFsDaPlaySct(Sint32 st_sct, Sint32 end_sct, Sint32 reptime)
```

Inputs

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

Outputs

None

Return value

Error codes

Function

This function starts DA playback at a specified sector.

Example

```
gdFsDaPlaySct(0xC000, 0xD000, 2);
```

Notes

gdFsDaRelease

Releases DA playback from the paused state.

Format

```
#include <sg_gd.h>
Sint32 gdFsDaRelease(Sint32 reptime)
```

Inputs

reptime	Repetitions
----------------	-------------

Outputs

None

Return value

Error codes

Function

This function releases DA playback from the paused state.

Example

```
gdFsDaRelease(0);
```

Notes

"reptime" is currently disabled.

gdFsDaStop

Stops DA playback.

Format

```
#include <sg_gd.h>
Sint32 gdFsDaStop(void)
```

Inputs

None

Outputs

None

Return value

Error codes

Function

This function stops DA playback.

Example

```
gdFsDaStop( );
```

Notes

gdFsEntryErrFunc Registers the callback function for when an error occurs.

Format

```
#include <sg_gd.h>
void gdFsEntryErrFunc(GDFS gdFs, GDFS_ERRFUNC erfunc, void *obj)
```

Inputs

gdFs	File handle
func	Callback function
obj	First parameter that is passed to the callback function

Outputs

None

Return value

None

Function

This function registers the callback function for when an error occurs.

Example

```
void err_callback(void *obj, Sint32 errcode)
{ /* Processing */
}
gdFsEntryErrFunc(gf, err_callback, (void *) 0x1234);
```

Notes

gdFsEntryRdEndFunc

Registers the callback function for when a read ends.

Format

```
#include <sg_gd.h>
void gdFsEntryRdEndFunc(GDFS gdfs, GDFS_FUNC func, void *obj);
```

Inputs

gdfs	File handle
func	Callback function
obj	First parameter that is passed to the callback function

Outputs

None

Return value

None

Function

This function registers the callback function for when a read is completed.

Example

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

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

Notes

gdFsEntryTrEndFunc

Registers the callback function for when a transfer ends.

Format

```
#include <sg_gd.h>
void gdFsEntryTrEndFunc(GDFS gdFs, GDFS_FUNC func, void *obj)
```

Inputs

gdFs	File handle
func	Callback function
obj	First parameter that is passed to the callback function

Outputs

None

Return value

None

Function

This function registers the callback function for when a transfer is completed.

Example

```
void trend_callback(void *obj)
{ /* Processing */
}
gdFsEntryTrEndFunc(gf, trend_callback, (void *) 0x1234);
```

Notes

gdFsFinish

Exits the library.

Format

```
#include <sg_gd.h>
void gdFsFinish(void)
```

Inputs

None

Outputs

None

Return value

None

Function

This function ends use of the GD file system.

Example

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

Notes

gdFsGetDirInfo

Gets the file information.

Format

```
#include <sg_gd.h>
Sint32 gdFsGetDirInfo(const char *name, GDFS_DIRINFO dirinfo)
```

Inputs

name	File name/directory name
-------------	--------------------------

Outputs

dirinfo	File information
----------------	------------------

Return value

Error codes

Function

This function gets file information.

Example

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

Notes

gdFsGetDirrecSize

Gets the directory record buffer size (macro).

SYNOPSIS

```
#include <sg_gd.h>
gdFsGetDirrecSize(max_dirent)
```

ARGUMENTS

max_dirent	Maximum number of entries in the directory
-------------------	--

RETURN

Number of bytes required for the directory buffer

DESCRIPTION

This macro determines the number of bytes required for the directory buffer.

EXAMPLES

```
UInt32 gdfscurdirgdFsGetDirrecSize(64)/4;
gdFsCalcSctSize
```

SYNOPSIS

```
gdFsCalcSctSize(bytes)
```

ARGUMENTS

bytes	File byte size
--------------	----------------

RETURN

Sector size

DESCRIPTION

This macro 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);
```

gdFsGetDrvStat

Gets the drive status.

Format

```
#include <sg_gd.h>

Sint32 gdFsGetDrvStat(void)
```

Inputs

None

Outputs

None

Return value

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

Function

This function gets the drive status.

Example

```
Sint32 dstat;                                dstat = gdFsGetDrvStat();
```

Notes

gdFsReqDrvStat: Function that updates the drive status; Inputs/Outputs: None; Return values: Error codes

gdFsGetErrStat

Gets the file error status.

Format

```
#include <sg_gd.h>
Sint32 gdFsGetErrStat(GDFS gdfs)
```

Inputs

gdfs	File handle
-------------	-------------

Outputs

None

Return value

Error codes

Function

This function gets the handle error status.

Example

```
GDFS gf;
Uint32 buf32*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);
```

Notes

gdFsGetFileFad

Gets the file FAD.

Format

```
#include <sg_gd.h>
Bool gdFsGetFileFad(GDFS gdfs, Sint32 *fad)
```

Inputs

gdfs	File handle
-------------	-------------

Outputs

fad	Frame address (physical sector)
------------	---------------------------------

Return value

Error codes

Function

This function gets a file's FAD.

Example

```
Sint32 fad;
Sint32 gf;
gf = gdFsOpen("TEST.BIN", NULL);
gdFsGetFileFad(gf, &fad);
```

Notes

gdFsGetFileSctSize

Gets the file sector size.

Format

```
#include <sg_gd.h>
Bool gdFsGetFileSctSize(GDFS gdfs, Sint32 *fsctsize)
```

Inputs

gdfs	File handle
-------------	-------------

Outputs

fsctsize	File size (bytes)
-----------------	-------------------

Return value

Error codes

Function

This function gets a file's sector size.

Example

```
GDFS gf;
Sint32 fslen;
gf = gdFsOpen("TEST.BIN", NULL);
gdFsGetFileSctSize(gdfs, &fslen);
```

Notes

gdFsGetFileSize

Gets the file size.

Format

```
#include <sg_gd.h>
Bool gdFsGetFileSize(GDFS gdfs, Sint32 *fsize)
```

Inputs

gdfs	File handle
-------------	-------------

Outputs

fsize	File size (bytes)
--------------	-------------------

Return value

Error codes

Function

This function gets file size information.

Example

```
GDFS gf;
Sint32 flen;
gf = gdFsOpen("TEST.BIN", NULL);
gdFsGetFileSize(gdfs, &flen);
```

Notes

gdFsGetNumRd Gets the size of the file that has already been read.

Format

```
#include <sg_gd.h>
Sint32 gdFsGetNumRd(GDFS gdfs);
```

Inputs

gdfs	File handle
-------------	-------------

Outputs

None

Return value

0 or greater: Number of bytes that were read

Function

This function gets the number of bytes that were read.

Example

```
GDFS gf;
Uint32 buf32*2048/4;
gf = gdFsOpen("TEST.BIN", NULL);
gdFsReqRd32(gf, 32, buf);
while (gdFsGetStat(gf) != GDD_STAT_COMPLETE) {
    readnum = gdFsGetNumRd(gdfs);
}
gdFsClose(gf);
```

Notes

gdFsGetStat

Gets the file status.

Format

```
#include <sg_gd.h>
Sint32 gdFsGetStat(GDFS gdfs)
```

Inputs

gdfs	File handle
-------------	-------------

Outputs

None

Return value

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

Function

This function gets the handle status.

Example

```
GDFS gf;
Uint32 buf32*2048/4;
gf = gdFsOpen("TEST.BIN", NULL);
gdFsReqRd32(gf, 32, buf);
while (gdFsGetStat(gf) == GDD_STAT_READ);
gdFsClose(gf);
```

Notes

gdFsGetSysHn

Gets the system handle.

Format

```
#include <sg_gd.h>
GDFS gdFsGetSysHn(void)
```

Inputs

None

Outputs

None

Return value

System handle

Function

This function 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);
```

Notes

gdFsGetToc

Gets the TOC.

Format

```
#include <sg_gd.h>
Sint32 gdFsGetToc(Sint32 type, void *buf)
```

Inputs

type	TOC type
0:	Single density
1:	High density

Outputs

buf	Buffer for loading the TOC
------------	----------------------------

408 bytes are required.

Return value

Error codes

Function

This function gets a TOC.

Example

```
gdFsGetToc(0, &buf);
```

Notes

gdFsGetTransStat

Gets the transfer status.

Format

```
#include <sg_gd.h>
Sint32 gdFsGetTransStat(GDFS gdfs)
```

Inputs

gdfs	File handle
-------------	-------------

Outputs

None

Return value

Error codes, transfer status

Function

This function gets the transfer status.

Example

```
gdFsGetTransStat(gf);
```

Notes

gdFsGetWorkHn

Gets the handle for the task currently being executed.

Format

```
#include <sg_gd.h>
GDFS gdFsGetWorkHn(void)
```

Inputs

None

Outputs

None

Return value

Handle that is currently being processed. If none, NULL.

Function

This function gets the handle that is being processed.

Example

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

Notes

gdFsGetWorkSize

Gets the library work size (macro).

SYNOPSIS

```
#include <sg_gd.h>
gdFsGetWorkSize(max_open)
```

ARGUMENTS

max_open	Number of files that can be open at one time
-----------------	--

RETURN

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;
```

gdFsInit

Initializes the library.

Format

```
#include <sg_gd.h>
Sint32 gdFsInit(Sint32 max_open, void *gdfs_work, Sint32 max_dirent, void *dirbuf)
```

Inputs

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)

Outputs

None

Return value

The following are the main error codes that are returned by this function.

Function

Return value	Meaning
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_TRAYOPEN	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.

This function initializes the GD file system.

Example

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

Notes

"gdfs_work" must be aligned with a 32-byte boundary. ("dirbuf" only requires alignment with a 4-byte boundary.)

Once initialization has been completed, re-initialization is not possible even if the gdFsInit function is called unless the gdFsFinish function has been called.

The main processing that is performed by gdFsInit is as follows:

- 1) Device driver initialization
- 2) Work area initialization
- 3) Device initialization
- 4) Mount processing

gdFsLoadDir

Loads a directory.

Format

```
#include <sg_gd.h>
Sint32 gdFsLoadDir(const char *dirname, GDFS_DIRREC gf_dirrec)
```

Inputs

dirname	Directory name
----------------	----------------

Outputs

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

Return value

Error codes

Function

This function loads a 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);
```

Notes

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

gdFsMovePickup

Moves the pickup.

Format

```
#include <sg_gd.h>
Sint32 gdFsMovePickup(GDFS gdfs)
```

Inputs

gdfs	File handle
-------------	-------------

Outputs

None

Return value

Error codes

Function

This function moves the pickup to the next reading position.

Example

```
gdFsMovePickup(gf);
```

Notes

gdFsOpen

Opens a file.

Format

```
#include <sg_gd.h>
GDFS gdFsOpen(const char *fname, GDFS_DIRREC gf_dirrec)
```

Inputs

fname	File name
gf_dirrec	Directory record handle where the file name will be searched for
NULL: Searches in the current directory.	

Outputs

None

Return value

NULL	Failed
Any other value:	File handle

Function

This function opens a file.

Example

```
/* Sample 1 */
GDFS gf;
gf = gdFsOpen("A.BIN", NULL);
/* Sample 2 */
GDFS gf;
Uint32 dirbufgdFsGetDirSize(64);
GDFS_DIRREC g_dir;
g_dir = gdFsCreateDirhn(dirbuf, 64);
gdFsLoadDir("MOVIE", g_dir);
gf = gdFsOpen("SMP.MOV", g_dir);
```

Notes

gdFsOpenRange

Opens specified sectors in a file.

Format

```
#include <sg_gd.h>
GDFS gdFsOpenRange(Sint32 stsct, Sint32 nsct)
```

Inputs

stsct	Starting sector
endsct	Ending sector

Outputs

None

Return value

NULL	Failed
Any other value:	File handle

Function

This function opens a file in specified sectors.

Example

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

Notes

gdFsRead

Reads a file.

Format

```
#include <sg_gd.h>
Sint32 gdFsRead(GDFS gdfs, Sint32 nsct, void *buf)
```

Inputs

gdfs	File handle
nsct	Number of sectors to be read

Outputs

buf	Storage buffer
------------	----------------

Return value

Error codes

Function

This function 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);
```

Notes

gdFsReinit

Reinitializes the library.

Format

```
#include <sg_gd.h>
Sint32 gdFsReinit(void)
```

Inputs

None

Outputs

None

Return value

Error codes

Function

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

Example

```
gdFsReinit();
```

Notes

gdFsReqDrvStat

Updates the drive status.

Format

```
#include <sg_gd.h>
Sint32 gdFsReqDrvStat(void)
```

Inputs

None

Outputs

None

Return value

Error codes

Function

This function changes the drive status.

Example

```
Sint32 dstat;
gdFsReqDrvStat();
/* After completion of request operation */
dstat = gdFsGetDrvStat();
```

Notes

gdFsReqGdRd

Issues read-ahead request to the GD buffer.

Format

```
#include <sg_gd.h>
Sint32 gdFsReqGdRd(GDFS gdfs, Sint32 nsct)
```

Inputs

gdfs	File handle
nsct	Number of sectors read

Outputs

None

Return value

Number of sectors actually requested (> 0). A negative value indicates an error code.

Function

This function issues a read-ahead request to the GD buffer.

Example

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

Notes

gdFsReqRd32

Reads a file.

Format

```
#include <sg_gd.h>
Sint32 gdFsReqRd32(GDFS gdfs, Sint32 nsct, void *buf)
```

Inputs

gdfs	File handle
nsct	Number of sectors to be read

Outputs

buf	Storage buffer
------------	----------------

Return value

Number of sectors actually requested (> 0). A negative value is an error code.

Function

This function 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.

Example

```
GDFS gf;
Uint32 buf32*2048/4;
gf = gdFsOpen("TEST.BIN", NULL);
gdFsReqRd32(gf, 32, buf);
while (gdFsGetStat(gf) == GDD_STAT_READ);
gdFsClose(gf);
```

Notes

"buf" must be aligned on a 32-byte boundary.

Furthermore, only one request can be accepted at any one time.

gdFsSeek

Changes the file reading position.

Format

```
#include <sg_gd.h>
Sint32 gdFsSeek(GDFS gdfs, Sint32 sctno, Sint32 type)
```

Inputs

gdfs	File handle
sctno	Sector number
type	

GDD_SEEK_SET	File start
GDD_SEEK_CUR	Current file reading position
GDD_SEEK_END	File end

Outputs

None

Return value

Error codes

Function

This function moves the file reading pointer.

Example

```
GDFS gf;
gf = gdFsOpen("TEST.BIN", NULL);
/* Seek the 6th sector from the start of the file */
gdFsSeek(gf, 5, SEEK_SET);
```

Notes

gdFsSetDir

Sets the current directory.

Format

```
#include <sg_gd.h>
Sint32 gdFsSetDir(GDFS_DIRREC gf_dirrec)
```

Inputs

gf_dirrec	Directory record handle
------------------	-------------------------

Outputs

None

Return value

Error codes

Function

This function sets the current directory.

Example

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

Notes

gdFsStopRd

Stops the reading of a file.

Format

```
#include <sg_gd.h>
Sint32 gdFsStopRd(GDFS gdfs)
```

Inputs

gdfs	File handle
-------------	-------------

Outputs

None

Return value

Error codes

Function

This function aborts a request.

Example

```
gdFsStopRd(gf);
```

Notes

gdFsTell

Gets the next file reading position.

Format

```
#include <sg_gd.h>
Sint32 gdFsTell(GDFS gdfs);
```

Inputs

gdfsFile handle

Outputs

None

Return value

Reading position

Function

This function gets the position of the file reading pointer in sector units.

Example

```
Sint32 pos;
GDFS gf;
gf = gdFsOpen("TEST.BIN", NULL);
pos = gdFsTell(gf);
```

Notes

gdFsTrans32

Specifies transfer from the GD buffer.

Format

```
#include <sg_gd.h>
Sint32 gdFsTrans32(GDFS gdfs, Sint32 nbytes, void *buf)
```

Inputs

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

Outputs

buf	Transfer address
------------	------------------

Return value

Error codes

Function

This function specifies a transfer from the GD buffer.

Example

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

Notes

3. Memory Management Function

syFree

Releases memory. (macro).

Format

```
#include <sg_Malloc.h>
syFree(__ap__) free(__ap__)
```

Inputs

Pointer to area to be freed

Outputs

None

Return value

None

Function

This macro is identical to free(*ap)

syMalloc

Allocates memory.

Format

```
#include <sg_Malloc.h>
Void *syMalloc(Void *heap, Uint32 size);
```

Inputs

size
Requested size (bytes)

Outputs

None

Return value

Pointer to allocated area
If NULL, allocation failed.

Function

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

syMallocFinish

Exits the Malloc system.

Format

```
#include <sg_Malloc.h>
Void syMallocFinish(Void)
```

Inputs

None

Outputs

None

Return value

None

Function

This function exits the Malloc system. When this occurs, the heap that was allocated by syMallocInit is released.

syMallocInit Allocates and declares the memory area to be used by the memory management function.

Format

```
#include <sg_Malloc.h>
Void syMallocInit(Void *heap, Uint32 size);
```

Inputs

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

Outputs

None

Return value

None

Function

This function specifies the area that is to be managed by the syMalloc library, and initializes the library.

This function is executed within the sbInitSystem function.

The default values in the sbInitSystem function are as follows:

Heap area starting address: End of section B

Heap size: To the end of memory

Notes

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 free memory size and the maximum contiguous memory size that can be obtained.

Format

```
#include <sg_Malloc.h>
Void syMallocStat(Uint32 *wholeFreeSize, Uint32 *biggestFreeBlockSize);
```

Inputs

free	Size of heap remaining
size	Maximum size that can be allocated at one time

Outputs

free	Size of heap remaining
size	Maximum size that can be allocated at one time

Return value

None

Function

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

4. Cache Functions

syCacheICI

Invalidates all caches. (Macro.)

Format

```
#include <sg_Cache.h>
syCacheICI()
```

Inputs

None

Outputs

None

Return value

None

Function

This macro invalidates all entries in the instruction cache.

Remarks

None

syCacheInit

Initializes the functions. (Macro.)

Format

```
#include <sg_Cache.h>
Void syCacheInit(SY_CACH_FORM form)
```

Inputs

form

Cache type. Specify by ORing the following types:

SYD_CACHE_IC_INDEX	Sets instruction cache to index mode.
SYD_CACHE_OC_INDEX	Sets operand cache to index mode.
SYD_CACHE_IC_ENABLE	Enables instruction cache.
SYD_CACHE_OC_ENABLE	Enables operand cache.
SYD_CACHE_OC_RAM	Sets operand cache to RAM mode.
SYD_CACHE_P1_CB	Sets P1 area to copyback mode.
SYD_CACHE_P0_WT	Sets P0 area to writeback mode.

Outputs

None

Return value

None

Function

This macro initializes the cache functions.

Remarks

This macro is called from within the syHwInit function. Applications are prohibited from overwriting the values that were set by this macro.

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 Shinobi library.

syCacheMOVCA

Executes MOVCA.L instruction.

Format

```
#include <sg_Cache.h>
Void syCacheMOVCA(Void *address, Uint32 size)
```

Inputs

address	Starting address
size	Size

Outputs

None

Return value

None

Function

This function executes MOVCA.L on all 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.)

Remarks

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.

syCachePREF

Operand cache pre-fetch.

Format

```
#include <sg_Cache.h>
Void syCachePREF(Void *address, Uint32 size)
```

Inputs

address	Starting address
size	Size

Outputs

None

Return value

None

Function

This function executes a pre-fetch for the specified area.

Remarks

None

syCacheOCI_P2

Invalidates all entries in the operand cache.

Format

```
#include <sg_Cache.h>
Void syCacheOCI_P2(Void)
```

Inputs

None

Outputs

None

Return value

None

Function

This function invalidates all entries in the operand cache.

Remarks

None

syCacheOCBI

Invalidates the operand cache block.

Format

```
#include <sg_Cache.h>
Void syCacheOCBI(void *address, Uint32 size)
```

Inputs

address	Starting address
size	Size

Outputs

None

Return value

None

Function

This function invalidates all cache blocks that include the specified area. ("To invalidate" means to discard the contents of the cache.)

Example

Remarks

None

syCacheOCBP

Purges the operand cache block.

Format

```
#include <sg_Cache.h>
Void syCacheOCBP(Void *address, Uint32 size)
```

Inputs

address	Starting address
size	Size

Outputs

None

Return value

None

Function

This function purges all cache blocks that include the specified area. ("To purge" means to discard the contents of the cache after a writeback.)

Remarks

None

syCacheOCWB

Writes back to the operand cache block.

Format

```
#include <sg_Cache.h>
Void syCacheOCWB(Void *address, Uint32 size);
```

Inputs

address	Starting address
size	Size

Outputs

None

Return value

None

Function

This function 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.)

Remarks

None

5. Get Peripheral Data Function

pdExecPeripheralServer Server functions for getting peripheral data.

Format

```
#include <sg_pad.h>
void pdExecPeripheralServer( void )
```

Inputs

None

Outputs

None

Return value

None

Function

This function creates peripheral data for a frame.

Remarks

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. 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.

pdGetPeripheral

Gets peripheral data.

Format

```
#include <sg_pad.h>
const PDS_PERIPHERAL* pdGetPeripheral(Uint32 port)
```

Inputs

port	Port number (PDD_PORT_A0/B0/C0/D0)
-------------	------------------------------------

Outputs

None

Return value

PDS_PERIPHERALAddress of PDS_PERIPHERAL structure

Function

This function gets peripheral data.

Reference

PDS_PERIPHERAL structure

Example

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

pdGetPeripheralError

Gets the peripheral status.

Format

```
#include <sg_pad.h>
Sint32 pdGetPeripheralError(UInt32 port)
```

Inputs

port ...Port number (PDD_PORT_A0/B0/C0/D0)

Outputs

None

Return value

0: No errors

Negative value: Error value (PDD_ERR_XXXX)

Function

This function gets an error code related to the acquisition of controller data.

Example

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


pdGetPeripheralInfo

Gets peripheral information.

Format

```
#include <sg_pad.h>
const PDS_PERIPHERALINFO* pdGetPeripheralInfo(Uint32 port)
```

Inputs

port	Port number (PDD_PORT_A0-A2/B0-B2/C0-C2/D0-D2)
-------------	--

Outputs

None

Return value

Address of PDS_PERIPHERALINFO structure

Function

This function gets data that is inherent to a peripheral.

Reference

PDS_PERIPHERALINFO structure, PDS_PERIPHERAL structure

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);
```

Remarks

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

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

pdTmrAlarm

Gets peripheral information.

Format

```
#include <sg_pdtmr.h>
Sint32 pdTmrAlarm( Uint32 port, Uint8* data )
```

Inputs

port Port number (PDD_PORT_A1/A2/B1/B2/C1/C2/D1/D2)

data Alarm data (4 bytes)

Of the four bytes of alarm data, the upper two bytes are for buzzer 1 and the lower two bytes are for buzzer 2.

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

To prevent both buzzers from sounding, write "00h" to both of the two bytes.

Outputs

None

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.

Function

This function sounds the alarm in a device, such as VMS, that has a timer.

Example

```
Uint8 data4;
Uint32 ret;
data0 = 0xc0;
data1 = 0x80;
data2 = 0x00;
data3 = 0x00;
ret = pdTmrAlarm(PDD_PORT_A1, data);
if (ret != PDD_TMRERR_OK) {
    /* Did not sound */
}
```

Remarks

6. LCD Functions

Reference

The screen is upside down when VMS is connected to the standard Katana controller.

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);
```

pdVmdLcdIsReady

Checks whether VMS is connected or not.

Format

```
#include <sg_lcd.h>
int32 pdVmsLcdIsReady(Uint32 port)
```

Inputs

port	Port number (PDD_PORT_A1/A2/B1/B2/C1/C2/D1/D2)
-------------	--

Outputs

None

Return value

TRUE	VMS is connected
FALSE	VMS is not connected

Function

This function checks whether VMS is connected or not.

Reference

Example

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


pdVmsLcdWrite

Displays data on the VMS LCD.

Format

```
#include <sg_lcd.h>
Sint32 pdVmsLcdWrite(Uint32 port, const void* data, Uint32 flag)
```

Inputs

port	Port number (PDD_PORT_A1/A2/B1/B2/C1/C2/D1/D2)
data	Pixel data address
flag	Flag
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

Outputs

None

Return value

PDD_LCDERR_OK	OK
PDD_LCDERR_NO_LCD	VMS not connected
PDD_LCDERR_BUSY	VMS is busy

Function

This function displays graphics on the VMS LCD.

Reference

Example

```
const Uint8 cgdata48 * 32 = {
0x00, 0xff, ...
...
};
ret = pdVmsLcdWrite(PDD_PORT_A1, cgdata, PDD_LCD_HVFLIP);
if (ret != PDD_LCDERR_OK) {
/* Display not possible */
}
```


7. Timer Functions

syTmrCountToMicro

Converts a count value to μ seconds.

Format

```
#include <sg_sytmr.h>
Uint32 syTmrCountToMicro(Uint32 count)
```

Inputs

count	Counter value to be converted
--------------	-------------------------------

Outputs

None

Return value

Microseconds value

Function

This function converts a counter value into a microseconds value.

Remarks

Decimals are truncated.

syTmrDiffCount Gets the difference between two timer count values.

Format

```
#include <sg_sytmr.h>
Uint32 syTmrDiffCount(Uint32 count1, Uint32 count2)
```

Inputs

count1	Counter value (not μ seconds value) that was gotten first
count2	Counter value (not μ seconds value) that was gotten second

Outputs

None

Return value

Difference between the two counter values

Function

This function calculates the difference between two counter values.

syTmrGenCancelInt Disables timer interrupts in response to the general-purpose timer.

Format

```
#include <sg_sytmr.h>
void syTmrGenCancelInt( void )
```

Inputs

None

Outputs

None

Return value

None

Function

This function disables the interrupt callback that was set by syTmrGenSetInt.

Remarks

syTmrGetCount

Gets the value of the free-running timer.

Format

```
#include <sg_sytmr.h>
Uint32 syTmrGetCount(void)
```

Inputs

None

Outputs

None

Return value

TMU0 counter value

Function

This function gets the TMU0 counter value.

Reference

Example

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

Remarks

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

syTmrGenCountToMicro

Converts the general-purpose counter value to μ seconds.

Format

```
#include <sg_sytmr.h>
Uint32 syTmrGenCountToMicro(Uint32 count, SY_TMR_CLOCK clock)
```

Inputs

count	Counter value to be converted
clock	Counter clock

Outputs

None

Return value

Microseconds value

Function

This function converts a counter value into a microseconds value.

Remarks

Decimals are truncated.

syTmrGenDiffCount

Calculates the difference in the general-purpose interrupt count.

Format

```
#include <sg_sytmr.h>
Uint32 syTmrGenDiffCount(Uint32 count1, Uint32 count2)
```

Inputs

count1	Counter value (not μ seconds value) that was gotten first
count2	Counter value (not μ seconds value) that was gotten second

Outputs

None

Return value

Counter value

Function

This function gets the difference between two timer counter values.

Remarks

Decimals are truncated.

syTmrGenGetCount Gets the general-purpose timer interrupt counter.

Format

```
#include <sg_sytmr.h>
Uint32 syTmrGenGetCount( void )
```

Inputs

None

Outputs

Return value

Counter value

Function

This function gets the general-purpose timer counter value.

Remarks

syTmrGenSetClock Sets the general-purpose timer interrupt counter.

Format

```
#include <sg_sytmr.h>
void syTmrGenSetClock( SYD_TMR_CLOCK clock )
```

Inputs

clock Counter clock

Macro name	Meaning
Time equivalent to 1 count	Upper limit of time that can be counted
SYD_TMR _CLOCK_P4	PØ/4
0.08µs	Approximately 21 minutes
SYD_TMR _CLOCK_P16	PØ/16
0.32µs	Approximately 22 hours and 52 minutes
SYD_TMR _CLOCK_P64	PØ/64
1.28µs	Approximately 91 hours and 38 minutes
SYD_TMR _CLOCK_P1024	PØ/1024
20.48µs	Approximately 17 years

Outputs

None

Return value

None

Function

This function 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. 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 1.28µ = 5497.558 seconds (= approximately 91 hours and 38 minutes).

Remarks

syTmrGenSetInt

Enables timer interrupts in response to the general-purpose timer.

Format

```
#include <sg_sytmr.h>
void syTmrGenSetInt( void (*func)( void * ) )
```

Inputs

func Function that is called when the counter value reaches "0"

If "NULL" is specified, control waits until the counter value reaches "0" and then exits this function.

Outputs

None

Return value

None

Function

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.

Example

Callback function example

```
void IntFunc(void* param)
{
    IntCount = param;           /* Save the parameters */
    syTmrGenSetInt(IntFunc, param+1); /* Continue interrupt generation */
    syTmrGenSetCount(0-10000);    /* Set the initial value in the counter*/
    syTmrGenStart();             /* Restart the timer */
}
```

Remarks

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

```
#include <sg_sytmr.h>
void syTmrGenStart( void )
```

Inputs

None

Outputs

None

Return value

None

Function

This function starts the general-purpose timer.

Remarks

syTmrGenStop

Stops the general-purpose timer.

Format

```
#include <sg_sytmr.h>
void syTmrGenStop( void )
```

Inputs

None

Outputs

None

Return value

None

Function

This function stops the general-purpose timer.

Remarks

syTmrGenMicroToCount

Converts μ seconds to a counter value.

Format

```
#include <sg_sytmr.h>
Uint32 syTmrGenMicroToCount(Uint32 micro, SY_TMR_CLOCK clock)
```

Inputs

micro	Microseconds value to be converted
clock	Timer clock

Outputs

None

Return value

Counter value

Function

This function converts a microseconds value to a counter value.

Remarks

Decimals are truncated.

syTmrGenSetCount Sets the general-purpose timer interrupt counter.

Format

```
#include <sg_sytmr.h>
void syTmrGenSetCount(Uint32 count)
```

Inputs

count	Value to be set in the counter
--------------	--------------------------------

Outputs

None

Return value

None

Function

This function sets the general-purpose timer counter value.

Remarks

syTmrMicroToCount

Converts μ seconds to a counter value.

Format

```
#include <sg_sytmr.h>
Uint32 syTmrMicroToCount(Uint32 micro)
```

Inputs

micro	Microseconds value to be converted
--------------	------------------------------------

Outputs

None

Return value

Counter value

Function

This function converts a microseconds value to a counter value.

Remarks

Decimals are truncated.

8. Real-time Clock Functions

syRtcCompareDate

Compares the date and time.

Format

```
#include<sg_syrtc.h>
Sint32 syRtcCompareDate( const SYS_RTC_DATE *date1, const SYS_RTC_DATE *date2 )
```

Inputs

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

Outputs

None

Return value

-1:	"date1" is smaller
0:	Equal
+1:	"date1" is larger

Function

This function compares two dates and times; the "larger" value is the more recent.

Remarks

None
syRtcCountToDate

Format

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

Inputs

count	Count to be converted
--------------	-----------------------

Outputs

date	Pointer for the converted date and time
-------------	---

Return value

None

Function

This function converts a count (one count = one second) into a date and time value.

Remarks

None

syRtcDateToCount

Converts from date and time to count.

Format

```
#include<sg_syrtc.h>
void syRtcDateToCount( const SYS_RTC_DATE *date, Uint32 *count )
```

Inputs

date	Pointer for the date and time to be converted
-------------	---

Outputs

count	Converted count
--------------	-----------------

Return value

None

Function

This function converts a date and time value into a count. The count is a Uint32 value in which one count is equal to one second.

Remarks

The "dayofweek" and "ageofmoon" members in the input structure are not referenced.

syRtcExecServer

Server functions.

Format

```
#include<sg_syrtc.h>
void syRtcExecServer( void )
```

Inputs

None

Outputs

None

Return value

None

Function

This function updates internal information concerning the date and time.

Remarks

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 syRtcGetDate() 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 library.

Format

```
#include<sg_syrtc.h>
void syRtcFinish( void )
```

Inputs

None

Outputs

None

Return value

None

Function

This function releases the work area and exits the RTC functions.

Remarks

None

syRtcGetDate

Gets the current date and time.

Format

```
#include<sg_syrtc.h>
Sint32 syRtcGetDate( SYS_RTC_DATE *date )
```

Inputs

None

Outputs

date	Current date and time
-------------	-----------------------

Return value

Server function status (SYD_RTC_STAT_***)

Function

This function gets the date and time.

Example

```
void main( void )
{
    SYS_RTC_DATE date, startdate;
    char *dayofweek = {
        "SUNDAY",
        "MONDAY",
        "TUESDAY",
        "WEDNESDAY",
        "THURSDAY",
        "FRIDAY",
        "SATURDAY"
    };
    while( 1 ){
        syRtcGetDate( &date );
        printf( "yyyy/mm/dd hh:mm:ss = %4d/%2d/%2d %2d:%2d:%2d\n",
            date.year, date.month, date.day,
            date.hour, date.minute, date.second );
        printf( "Today is %s\n", dayofweekday.dayofweek );
        switch( date.ageofmoon ) {
        case 0:
            printf( "new moon\n" );
            break;
        case 15:
            printf( "full moon\n" );
            break;
        }
    }
}
```

Remarks

None

syRtcGetStat

Gets the server function status.

Format

```
#include<sg_syrtc.h>
Sint32 syRtcGetStat( void )
```

Inputs

None

Outputs

None

Return value

Server function status (SYD_RTC_STAT_***)

Function

This function gets the server function activity status (SYD_RTC_STAT_**).

Remarks

In the initial state, the RTC server functions are running (SYD_RTC_STAT_ACTIVE).

syRtcInit

Initializes the library.

Format

```
#include<sg_syrtc.h>
Sint32 syRtcInit( void )
```

Inputs

None

Outputs

None

Return value

SYD_RTC_ERR_***

Function

This function allocates a work area and initializes the RTC functions.

Remarks

This function is called within the sbInitSystem function.

syRtcSetDate

Sets the date and time.

Format

```
#include<sg_syrtc.h>
Sint32 syRtcSetDate( const SYS_RTC_DATE *date )
```

Inputs

date	Pointer for setting the date and time
-------------	---------------------------------------

Outputs

None

Return value

Error code (SYD_RTC_ERR_***)

Function

This function sets the date and time.

Remarks

The "dayofweek" and "ageofmoon" members in the input structure are not referenced.

The AICA internal registers are updated even when the server functions are stopped.

syRtcSetServerMode

Sets the server function operation.

Format

```
#include<sg_syrtc.h>
void syRtcSetServerMode( const Sint32 mode )
```

Inputs

mode	Server function operation (SYD_RTC_STAT_***)
-------------	--

Outputs

None

Return value

None

Function

This function sets the server function activity status.

Remarks

In the initial state, the RTC server functions are running (SYD_RTC_STAT_ACTIVE).

9. Backup Functions

buAnalyzeBackupFileImage

Analyzes the file image in memory.

Format

```
#include<sg_bup.h>
Sint32 buAnalyzeBackupFileImage(BUS_BACKUPFILEHEADER *hdr, const void *buf)
```

Inputs

hdr	Backup file header address
buf	Address where the file to be analyzed was loaded (4-byte boundary)

Outputs

None

Return value

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

Function

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;
}
```

Remarks

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

buCalcBackupFileSize

Calculates the block size of the backup file image.

Format

```
#include<sg_bup.h>

Sint32 buCalcBackupFileSize(UInt32 inum, UInt32 vtype, UInt32 size)
```

Inputs

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

Outputs

None

Return value

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

Function

This function calculates the size of a Dreamcast backup file format image.

Example

```
extern UInt8 game_name;      /* Game name (sort item) */
extern UInt16 icon_palette; /* Icon palette data */
extern UInt8 icon_data;     /* Icon pixel data */
extern UInt8 visual_data;   /* Visual data */
extern UInt8 save_data;     /* Application save data */
Sint32 nblock1, nblock2;
void* buf;
BUS_BACKUPFILEHEADER hdr;    /* Backup file header */
memset(&hdr, 0, sizeof(hdr));
strcpy(hdr.vms_comment, "VMS_COMMENT"); /* VMS comment setting */
strcpy(hdr.btr_comment, "Boot ROM double-width character comment"); /* Boot ROM comment
setting */
memcpy(hdr.game_name, game_name, 16); /* Game name (sort item) setting */
hdr.icon_palette = icon_palette; /* Icon palette data setting */
hdr.icon_data = icon_data; /* Icon pixel data setting */
hdr.icon_num = 1; /* Number of icons setting */
hdr.icon_speed = 1; /* Icon speed setting */
hdr.visual_data = visual_data; /* Visual comment setting */
hdr.visual_type = BUD_VISUALTYPE_C; /* Visual type setting */
hdr.save_data = save_data; /* Application save data */
hdr.save_size = 0x400; /* Application save data size */
nblock = buCalcBackupFileSize(hdr.icon_num, hdr.visual_type, hdr.save_size);
buf = syMalloc(nblock * 512); /* Allocates (the number of blocks to be used x 512 bytes) */
if (buf == NULL) {
/* Memory cannot be allocated */
return NG;
}
nblock = buMakeBackupFileImage(buf, &hdr);
if (nblock < 0) {
/* Structure includes an invalid parameter */
return NG;
}
}
```

Remarks

buDefragDisk

Allocates consecutive free blocks.

Format

```
#include<sg_bup.h>
Sint32 buDefragDisk(Sint32 drive, void* work)
```

Inputs

drive	Drive number
work	Work buffer (512 bytes)

Outputs

None

Return value

BUD_ERR_OK	Processing request was received.
BUD_ERR_BUSY	Busy - request could not be accepted.

Function

This function optimizes a disk in order permit allocation of an area where an executable file can be written.

If the size of the executable file that is to be saved is larger ??SMALLER?? than the free space in visual memory, but the consecutive free space is smaller than the size of the executable file that is to be saved, this function can be used in order to make it possible to allocate an area where the executable file can be saved.

Example

```
Sint32 ret;
Uint32 DefragWork512 / 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;
```

Remarks

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 defrag processing is in progress, all data will be lost.

We strongly recommend that the application remind the user to never remove the memory card while defrag processing is in progress.

buDeleteFile

Deletes a file.

Format

```
#include<sg_bup.h>

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

Inputs

drive	Drive number
fname	File name

Outputs

None

Return value

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

Function

This function deletes a file.

Example

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

Remarks

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

BUD_ERR_OK	Normal end
BUD_ERR_NO_DISK	Disk 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 library.

Format

```
#include<sg_bup.h>
Sint32 buExit( void );
```

Inputs

None

Outputs

None

Return value

BUD_ERR_OK

Normal end

BUD_ERR_BUSY

A type B function is being processed.

Function

This function exits the backup file system.

Example

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

Remarks

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.

buFindExecFile

Gets the name of an executable file.

Format

```
#include<sg_bup.h>
Sint32 buFindExecFile( Sint32 drive, char* fname )
```

Inputs

drive
Drive number
fname
File name

Outputs

None

Return value

BUD_ERR_OK	Normal end
BUD_ERR_FILE_NOT_FOUND	Executable file not found.
BUD_ERR_UNFORMAT	Disk is not formatted.
BUD_ERR_NO_DISK	Disk not found.
BUD_ERR_BUSY	A type B function was being processed.

Function

This function gets the name of an executable file.

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 fname0 = "*E0" 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 fname16;
/* 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*En", fname);
    break;
case BUD_ERR_FILE_NOT_FOUND:
    printf("Executable file not found.  \n");
    break;
default:
    printf("Error \n");
    break;
}
```

Remarks

buFindFirstFile

Gets the name of the first file.

Format

```
#include<sg_bup.h>
Sint32 buFindFirstFile( Sint32 drive, char* fname )
```

Inputs

drive	Drive number
fname	Address where file name is to be output

Outputs

fname	File name
-------	-----------

Return value

BUD_ERR_OK	Normal end
BUD_ERR_FILE_NOT_FOUND	File not found.
BUD_ERR_UNFORMAT	Disk is not formatted.
BUD_ERR_NO_DISK	No disk found.
BUD_ERR_BUSY	A type B function is being processed.

Function

This function gets the name of the first file.

If the file does not exist, then fname0 = "*E0" results.

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

Example

```
Sint32 ret, files, blocks, totel;
char fname16;
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;
```

Remarks

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

```
#include<sg_bup.h>
Sint32 buFindNextFile( Sint32 drive, char* fname )
```

Inputs

drive	Drive number
fname	Address where file name is to be output

Outputs

fname	File name
--------------	-----------

Return value

BUD_ERR_OK	Normal end
BUD_ERR_FILE_NOT_FOUND	File not found.
BUD_ERR_UNFORMAT	Disk is not formatted.
BUD_ERR_NO_DISK	No disk found.
BUD_ERR_BUSY	A type B function is being processed.

Function

This function gets the name of the next file.

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

Example

```
Sint32 ret, files, blocks, total;
char fname16;
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;
```

Remarks

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 fname0 = "*E0" results.

buFormatDisk

Formats backup RAM.

Format

```
#include<sg_bup.h>

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

Inputs

drive	Drive number
volume	Volume data
icon	Icon number (0 to 123)
time	Time stamp
flag	Format flag
TRUE	Full format
FALSE	Quick format

Outputs

None

Return value

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

Function

This function formats a disk.

Example

```
Sint32 ret;
BUS_TIME time;
SYS_RTC_DATE rtc;
Uint8 volume32;
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));

volume0 = 0x01; /* Body color information found */
volume1 = 0xbf; /* B */
volume2 = 0x00; /* G */
volume3 = 0x00; /* R */
volume4 = 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;
```

Remarks

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

BUD_ERR_OK	Normal end
BUD_ERR_NO_DISK	Disk not found

buGetDiskInfo

Gets backup RAM information.

Format

```
#include<sg_bup.h>
Sint32 buGetDiskInfo( Sint32 drive, BUS_DISKINFO* info )
```

Inputs

drive	Drive number
info	Address of disk information structure

Outputs

None

Return value

BUD_ERR_OK	Normal end
BUD_ERR_UNFORMAT	Disk is not formatted.
BUD_ERR_NO_DISK	Disk not found.
BUD_ERR_BUSY	A type B function is being processed.

Function

This function gets the disk information for a specified drive.

Example

```
BUS_DISKINFO info;

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

buGetDiskFree

Checks the free space.

Format

```
#include<sg_bup.h>
Sint32 buGetDiskFree( Sint32 drive, Sint32 type )
```

Inputs

drive	Drive number
type	Type of blocks for which number of free blocks is to be checked
BUD_FILETYPE_NORMAL	Normal file free space
BUD_FILETYPE_EXECUTABLE	Executable file free space

Outputs

None

Return value

0 or positive value	Number of free blocks
BUD_ERR_UNFORMAT	Not formatted.
BUD_ERR_NO_DISK	No disk found.
BUD_ERR_BUSY	A type B function is being processed.

Function

This function gets the number of free blocks in the specified drive.

Normal file free space includes executable file free space.

Example

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

buGetFileInfo

Gets information on the specified file.

Format

```
#include<sg_bup.h>
Sint32 buGetFileInfo( Sint32 drive, const char* fname, BUS_FILEINFO* finfo )
```

Inputs

drive	Drive number
fname	File name
finfo	Address of file information structure

Outputs

None

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	Disk not found.
BUD_ERR_BUSY	A type B function is being processed.

Function

This function gets file information.

Example

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

buGetFileSize

Checks the size of the specified file.

Format

```
#include<sg_bup.h>
Sint32 buGetFileSize( Sint32 drive, const char* fname )
```

Inputs

drive	Drive number
fname	File name

Outputs

None

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	Disk not found.
BUD_ERR_BUSY	A type B function is being processed.

Function

This function gets the file size in terms of the number of blocks.

Example

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

buGetLastError

Gets the last error that was generated.

Format

```
#include<sg_bup.h>
Sint32 buGetLastError( Sint32 drive )
```

Inputs

drive	Drive number
--------------	--------------

Outputs

None

Return value

Error code

Function

This function gets the last error that was generated by the specified drive.

Example

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

bulnit

Initializes the library.

Format

```
#include<sg_bup.h>
Sint32 buInit( Sint32 capacity, Sint32 drives, void *work, BU_INITCALLBACK func )
```

Inputs

- capacity

Maximum capacity of media

BUD_CAPACITY_128KB

BUD_CAPACITY_256KB

BUD_CAPACITY_512KB

BUD_CAPACITY_1MB
- drives

Target drive

BUD_USE_DRIVE_ALL	All drives
BUD_USE_DRIVE_A1	Port A, drive 1
BUD_USE_DRIVE_A2	Port A, drive 2
BUD_USE_DRIVE_B1	Port B, drive 1
BUD_USE_DRIVE_B2	Port B, drive 2
BUD_USE_DRIVE_C1	Port C, drive 1
BUD_USE_DRIVE_C2	Port C, drive 2
BUD_USE_DRIVE_D1	Port D, drive 1
BUD_USE_DRIVE_D2	Port D, drive 2

- work

Work buffer
- func

When NULL is specified, the "capa" value is ignored.

Initialization complete callback

Outputs

None

Return value

- BUD_ERR_OK

Normal end
- BUD_ERR_INVALID_PARAM

Invalid parameter

Function

This function initializes the backup file system.

When initialization is completed successfully, the specified function is called back. The callback function is called before returning from

`buInit()`.

In this case, after the `BUD_OP_CONNECT` callback is detected, the application allocates a buffer that is suited for the size of the indicated media, 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 media without allocating a large buffer size beforehand.

Example

```
/* Allocating two buffers for backup RAM of up to 128K */
Uint32 workBUM_WORK_SIZE(BUD_CAPACITY_128KB, 2) / sizeof(Uint32);
void init_callback(void)
{
}
/* Supporting drives 1 and 2 for port A */
buInit(BUD_CAPACITY_128KB,
      BUD_USE_DRIVE_A1 | BUD_USE_DRIVE_A2,
      work, init_callback);
```

Remarks

The work space that is needed varies greatly, depending on the maximum size and the number of drives. Do not specify backup RAM and drives beyond the needs of the application.

Not specifying unnecessary drives reduces the size of the work space and permits faster library operation.

The following table lists guidelines for work space:

Maximum capacity	Work space per drive	Work space for 8 drives
128KB	8KB	64KB
256KB	14KB	112KB
512KB	28KB	220KB
1MB	55KB	440KB

Be sure to specify the `BUD_USE_DRIVE_XXX` constant for the number of drives in use. Do not specify `BUD_DRIVE_XXX` constant by mistake.

If `NULL` is specified for "work", the "capa" setting can be ignored, allowing dynamic management of the backup buffer.

buIsReady Checks whether backup RAM is connected or not.

Format

```
#include<sg_bup.h>
Sint32 buIsReady( Sint32 drive )
```

Inputs

drive	Drive number
--------------	--------------

Outputs

None

Return value

TRUE	Drive is connected and ready for access
FALSE	Drive is not connected

Function

This function returns the connection status of a drive.

Example

```
if (buIsReady(BUD_DRIVE_A1)) {
    /* Backup RAM is connected */
}
```


buIsFormat Checks whether backup RAM has been formatted or not.

Format

```
#include<sg_bup.h>
Sint32 buIsFormat( Sint32 drive )
```

Inputs

drive	Drive number
--------------	--------------

Outputs

None

Return value

BUD_ERR_OK	Drive is formatted.
BUD_ERR_UNFORMAT	Disk is not formatted.
BUD_ERR_NO_DISK	No disk found.
BUD_ERR_BUSY	A type B function is being processed.

Function

This function checks whether the disk in the specified drive has been formatted or not.

Example

```
switch (buIsFormat(BUD_DRIVE_A1)) {
case BUD_ERR_OK:
    /* Formatted */
    break;
case BUD_ERR_NO_DISK:
    /* Backup RAM is not connected */
    break;
case BUD_ERR_BUSY:
    /* Busy - backup RAM could not be checked */
    break;
}
```

buIsExistFile

Checks whether the specified file exists or not.

Format

```
#include<sg_bup.h>
Sint32 buIsExistFile( Sint32 drive, const char* fname )
```

Inputs

drive	Drive number
fname	File name

Outputs

None

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	Disk not found.
BUD_ERR_BUSY	A type B function is being processed.

Function

This function checks whether the specified file exists.

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;
}
```

buLoadFile

Loads a file.

Format

```
#include<sg_bup.h>
Sint32 buLoadFile( Sint32 drive, const char* fname, void* buf, Uint32 nblock )
```

Inputs

drive	Drive number
fname	File name
buf	Load address
nblock	Number of blocks to be loaded

Outputs

buf	Buffer where data was loaded (However, because this function is of the immediate return type, this value becomes undefined after the function terminates.)
------------	--

Return value

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

Function

This function loads a file.

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;
extern char SaveData;          /* Load buffer */
blocks = buGetFileSize(BUD_DRIVE_A1, "SAVEDATA"); /* Get file size */
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) */
}
```

Remarks

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

BUD_ERR_OK	Normal end
BUD_ERR_NO_DISK	
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

Creates a backup file format memory image.

Format

```
#include<sg_bup.h>
Sint32 buMakeBackupFileImage(void* buf, BUS_BACKUPFILEHEADER* hdr)
```

Inputs

buf	Address where backup file format image is to be created (4-byte boundary)
hdr	Backup file header address

Outputs

None

Return value

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

Function

This function creates a Dreamcast backup file format image in memory.

Example

```
extern UInt8 game_name;      /* Game name (sort item) */
extern UInt16 icon_palette; /* Icon palette data */
extern UInt8 icon_data;     /* Icon pixel data */
extern UInt8 visual_data;   /* Visual data */
extern UInt8 save_data;     /* Application save data */
Sint32 nblock1, nblock2;
void* buf;
BUS_BACKUPFILEHEADER hdr;    /* Backup file header */
memset(&hdr, 0, sizeof(hdr));
strcpy(hdr.vms_comment, "VMS_COMMENT"); /* VMS comment setting */
strcpy(hdr.btr_comment, "Boot ROM double-width character comment"); /* Boot ROM comment
setting */
memcpy(hdr.game_name, game_name, 16); /* Game name (sort item) setting */
hdr.icon_palette = icon_palette; /* Icon palette data setting */
hdr.icon_data = icon_data; /* Icon pixel data setting */
hdr.icon_num = 1; /* Number of icons setting */
hdr.icon_speed = 1; /* Icon speed setting */
hdr.visual_data = visual_data; /* Visual comment setting */
hdr.visual_type = BUD_VISUALTYPE_C; /* Visual type setting */
hdr.save_data = save_data; /* Application save data */
hdr.save_size = 0x400; /* Application save data size */
nblock = buCalcBackupFileSize(hdr.icon_num, hdr.visual_type, hdr.save_size);
buf = syMalloc(nblock * 512); /* Allocates (the number of blocks to be used x 512 bytes) */
if (buf == NULL) {
    /* Memory cannot be allocated */
    return NG;
}
nblock = buMakeBackupFileImage(buf, &hdr);
if (nblock < 0) {
    /* Structure includes an invalid parameter */
    return NG;
}
```

Remarks

buLoadFileEx

Loads a file with the load start block and the number of blocks specified.

Format

```
#include<sg_bup.h>
Sint32 buLoadFileEx(Sint32 drive, const char* fname, void* buf, Uint32 start, Uint32
nblock)
```

Inputs

drive	Drive number
fname	File name
buf	Load address
start	Load start address
nblock	Number of blocks to be loaded

Outputs

buf	Data that was loaded
------------	----------------------

Return value

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

Function

This function loads a file.

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;
extern char LoadBuffer; /* Load buffer */
/* 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;
return OK;
```

Remarks

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

BUD_ERR_OK	Normal end
BUD_ERR_NO_DISK	Disk 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.

buMountDisk

Mounts a drive.

Format

```
#include<sg_bup.h>
Sint32 buMountDisk( Sint32 drive, void *addr, Sint32 size );
```

Inputs

drive	Drive number
addr	Work address
size	Work size

Outputs

None

Return value

BUD_INVALID_PARAM	Buffer is not sufficient for the backup RAM that is connected.
BUD_ERR_OK	Normal end

Function

This function mounts a drive.

Example

Remarks

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.

The drive is also not mounted if NULL was specified for the work buffer in buInit and this function was not called.

buSaveExecFile

Saves an executable file.

Format

```
#include<sg_bup.h>
Sint32 buSaveExecFile( Sint32 drive, const char* fname, const void* buf, Uint32 nblock,
BUS_TIME* time, Sint32 flag )
```

Inputs

drive	Drive number
fname	File name
buf	Read address
nblock	Number of blocks to be read
time	Time stamp
flag	Flag
BUD_FLAG_VERIFY	Verify enabled
BUD_FLAG_COPY(n)	Copy flag ("n" is the value of the copy flag)

Outputs

None

Return value

BUD_ERR_OK	Processing request was received.
BUD_ERR_BUSY	Busy - request could not be accepted.

Function

This function saves an executable file.

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.

Example

```
Sint32 ret;
Sint32 blocks, flag;
BUS_TIME time;
SYS_RTC_DATE rtc;
extern char SaveData; /* Data to be saved (512 * 10 bytes) */
blocks = 10; /* File size is 10 blocks */
flag = BUD_FLAG_VERIFY | BUD_FLAG_COPY(0xff); /* Verify enabled/Set copy flag to FFH */
syRtcGetDate( &rtc );
time = ( BUS_TIME )rtc; /* Time stamp setting */
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;
return OK;
```

Remarks

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:

BUD_ERR_OK	Normal end
BUD_ERR_UNFORMAT	Disk is not formatted
BUD_ERR_NO_DISK	Disk 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

```
#include<sg_bup.h>
Sint32 buSaveFile( Sint32 drive, const char* fname, const void* buf, Uint32 nblock, const
BUS_TIME* time, Uint32 flag )
```

Inputs

drive	Drive number
fname	File name
buf	Read address
nblock	Number of blocks to be read
time	Time stamp
flag	Flag
BUD_FLAG_VERIFY	Verify enabled
BUD_FLAG_COPY(n)	Copy flag ("n" is the value of the copy flag)

Outputs

None

Return value

BUD_ERR_OK	Processing request was received.
BUD_ERR_BUSY	Busy - request could not be accepted.

Function

This function saves a file.

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.

Example

```
Sint32 ret;
Sint32 blocks, flag;
BUS_TIME time;
SYS_RTC_DATE rtc;
extern char SaveData; /* Data to be saved (512 * 10 bytes) */
blocks = 10; /* File size is 10 blocks */
flag = BUD_FLAG_VERIFY | BUD_FLAG_COPY(0x00); /* Verify enabled/Set copy flag to 00H */
syRtcGetDate( &rtc );
time = ( BUS_TIME )rtc; /* Time stamp setting */
ret = buSaveFile(BUD_DRIVE_A1, "SAVEDATA", SaveData, blocks, &time, flag);
if (ret == BUD_ERR_OK) {
    /* Save request was successful */
} else { /* Save request failed (busy) */
}
```

Remarks

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

BUD_ERR_OK	Normal end
BUD_ERR_NO_DISK	Disk not found
BUD_ERR_UNFORMAT	Disk is not formatted
BUD_ERR_FILE_NOT_FOUND	File not found
BUD_ERR_CANNOT_OPEN	Cannot open file

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.

buSetCompleteCallback Registers a function that is called back when processing is completed.

Format

```
#include<sg_bup.h>
void buSetCompleteCallback( BU_COMPLETECALLBACK func )
```

Inputs

func	Complete callback function address
-------------	------------------------------------

Outputs

None

Return value

None

Function

This function 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 the file attributes.

Format

```
#include<sg_bup.h>
Sint32 buSetFileAttr( Sint32 drive, const char* fname, Uint16 hdrofs, Uint8 copyflag )
```

Inputs

drive	Drive number
fname	File name
hdrofs	Header offset (ignored)
copyflag	Copy flag (00H to FFH)

Outputs

None

Return value

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

Function

This function changes file attributes.

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;
```

Remarks

Use this function to change the copy flag for a file that already exists.

The header offset is ignored. Specify "0".

buSetProgressCallback Registers a function that is called back when processing progresses to a certain point.

Format

```
#include<sg_bup.h>
void buSetProgressCallback( BU_PROGRESSCALLBACK func )
```

Inputs

func	Progress callback function address
-------------	------------------------------------

Outputs

None

Return value

None

Function

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

Remarks

Call this function during the initialization complete callback function. The progress callback function is called in the following cases:

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 complete or not.

Format

```
#include<sg_bup.h>
Sint32 buStat( Sint32 drive )
```

Inputs

drive	Drive number
--------------	--------------

Outputs

None

Return value

BUD_STAT_READY	Processing has been completed.
BUD_STAT_BUSY	A type B function is being processed.

Function

This function checks whether processing is completed.

Example

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

buUnmount

Unmounts a drive.

Format

```
#include<sg_bup.h>
Sint32 buUnmount( Sint32 drive );
```

Inputs

drive	Drive number
-------	--------------

Outputs

None

Return value

BUD_ERR_OK	Normal end
BUD_ERR_BUSY	The drive is being accessed.

Function

This function unmounts a drive.

Example

Remarks

The 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

```
#include <sg_sycbl.h>
SYE_CBL_CABLE syCblCheckCable( void )
```

Inputs

None

Outputs

None

Return value

Cable type

Function

This function gets the cable type.

Example

```
#include "sg_sychbl.h"
void displayCable( Sint16 x, Sint16 y )
{
    SYE_CBL_CABLE cable;
    cable = syCblCheckCable();
    switch( cable ) {
    case SYE_CBL_CABLE_VGA:
        njPrintC( NJM_LOCATION(x,y), "VGA CABLE" );
        break;
    case SYE_CBL_CABLE_NTSC:
    case SYE_CBL_CABLE_PAL:
        njPrintC( NJM_LOCATION(x,y), "AV CABLE" );
        break;
    case SYE_CBL_CABLE_NTSC_RGB:
    case SYE_CBL_CABLE_PAL_RGB:
        njPrintC( NJM_LOCATION(x,y), "RGB CABLE" );
        break;
    default:
        njPrintC( NJM_LOCATION(x,y), "UNKNOWN CABLE" );
        break;
    }
}
```

Remarks

This function may be called before sbInitSystem without causing any problems.

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

11. Boot ROM Font Functions

syBtFntChkSmph

Checks the font semaphore.

Format

```
#include <sg_btfnt.h>
extern Sint32 syBtFntChkSmph( void )
```

Inputs

None

Outputs

None

Return value

Font semaphore check result SYD_BT_FNT_SMPH_***

Function

This function 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.

Notes

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

syBtFntClrSmph

Clears the font semaphore.

Format

```
#include <sg_btfmt.h>
extern void syBtFntClrSmph( void )
```

Inputs

None

Outputs

None

Return value

None

Function

This function clears the font semaphore.

syBtFntGetAddr Gets offset for target font data from Shift JIS codes.

Format

```
#include <sg_btfont.h>
Uint32 syBtFntGetAddr( SYE_BT_FNT_FONTSET set, Sint16 code )
```

Inputs

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

Outputs

None

Return value

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

Function

This function calculates the offset from the font data starting address from the Shift JIS code. The value that is calculated is the number of bytes.

Notes

None

syBtFntGet

Gets starting address of font data in boot ROM.

Format

```
#include <sg_btfont.h>
extern void *syBtFntGet(Uint32 num)
```

Inputs

num	Control number
------------	----------------

Outputs

None

Return value

Starting address for font data in boot ROM

Function

This function gets the address where font data is stored in boot ROM.

Notes

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

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.

Example of usage

```
#define SMD_WRK_ADDR (0x0c020000)
main()
{
void *smFntAddr, *smTheFntAddr;
void *smWrkAddr;
Uint8 smTheCode;
smFntAddr = syBtFntGet();
smWrkAddr = (void *)SMD_WRK_ADDR;
smTheCode = '?';
/* evaluate the font address in Boot ROM */
smTheFntAddr = (void *)((Uint32)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();
}
}
```

syBtFntGetInfo

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

Format

```
#include <sg_btfont.h>
void syBtFntGetInfo( SYE_BT_FNT_FONTSET set, Sint16 code, SYS_BT_FNT_INFO *info )
```

Inputs

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

Outputs

info	Size information for font data specified by the font set and code
-------------	---

Return value

None

Function

This function gets the number of pixels in the vertical and horizontal directions of the font data.

Notes

None

syBtFntSjis2Jis

Converts Shift JIS codes to JIS codes.

Format

```
#include <sg_btfont.h>
Sint16 syBtFntSjis2Jis( Sint16 sjis )
```

Inputs

sjis	Shift JIS code
-------------	----------------

Outputs

None

Return value

JIS code

Function

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

Notes

None

12. Configuration Functions

syCfgExit

Exits the configuration functions.

Format

```
#include <sg_syscfg.h>
Sint32 syCfgExit( Void )
```

Inputs

None

Outputs

None

Return value

Error code

SYD_CFG_OK	Normal end
SYD_CFG_NG	Abnormal end (not currently used)

Function

This function performs the configuration function exit processing.

Remarks

Under the current specifications, re-entry is not possible. (Once the configuration functions have been exited, they must be initialized again in order to be used.)

syCfgGetSoundMode

Gets the sound setting.

Format

```
#include <sg_sycfg.h>
Sint32 syCfgGetSoundMode( Sint32* nMode ) ;
```

Inputs

None

Outputs

nMode

SYD_CFG_STEREO Stereo setting

SYD_CFG_MONO Monaural setting

Return value

Error code

SYD_CFG_OK Normal end

SYD_CFG_NG Abnormal end

Function

This function gets the sound setting (stereo/monaural).

Example

```
Sint32 mode;
syCfgGetSoundMode( &mode );
if ( mode == SYD_CFG_STEREO ) {
    sdSndSetSpace( SDE_SPACE_STEREO );
} else {
    sdSndSetSpace( SDE_SPACE_MONO );
}
```

Remarks

"Abnormal end" results if an invalid parameter was specified, or if the sound setting has not been initialized.

syCfgInit

Initializes the configuration functions.

Function

```
#include <sg_sycfg.h>
Sint32 syCfgInit( Void* pBuf )
```

Inputs

pBuf

Specifies the address of the (16K) buffer that is used by the library.

Outputs

None

Return value

Error code

SYD_CFG_OK Normal end

SYD_CFG_NG Abnormal end

Function

This function initializes the configuration functions.

Remarks

syCfgSetSoundMode

Updates the sound setting.

Format

```
#include <sg_sycfg.h>
Sint32 syCfgSetSoundMode( Sint32 nMode ) ;
```

Inputs

None

Outputs

nMode
SYD_CFG_STEREO Stereo setting
SYD_CFG_MONO Monaural setting

Return value

Error code
SYD_CFG_OK Normal end
SYD_CFG_NG Abnormal end

Function

This function gets the sound setting (stereo/monaural).

Example

```
Sint32 mode;
syCfgGetSoundMode( &mode );
if ( mode == SYD_CFG_STEREO ) {
    sdSndSetSpace( SDE_SPACE_STEREO );
} else {
    sdSndSetSpace( SDE_SPACE_MONO );
}
```

Remarks

"Abnormal end" results if an invalid parameter was specified, or if the sound setting has not been initialized.

13. Boot ROM Service Functions

syBtCheckDisc

Checks for disk replacement.

Format

```
#include<sg_sybt.h>
extern Sint32 syBtCheckDisc( Void )
```

Inputs

None

Outputs

None

Return value

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

Function

This function gets the type of a disk that has been replaced.

Remarks

syBtExit

Returns to the boot ROM GUI screen.

Format

```
#include<sg_sybt.h>
Sint32 syBtExit( Void )
```

Inputs

None

Outputs

None

Return value

Function

This function forces to return to boot ROM GUI screen.

Be sure to call this function after sbExitSystem.

Remarks

14. Sound System API

sdDrvGetExecuteCounter

Get Sound Driver execution counter.

Prototype

```
#include <sg_sd.h>
EXTERN SDE_ERR sdDrvGetExecuteCounter( Uint32 *count_num);
```

Purpose

Gets the counter value maintained by the sound driver.

Parameter

count_num

...

Pointer to storage for the Sound Driver count.

Return

SDE_ERR_NO_INIT

...

The Sound Library has not been initialized.

Remarks

This function can be used to determine whether the sound driver is working.

Reference

sdDrvGetMidiTimingCounter Get count number of special messages counted by sound driver.

Prototype

```
#include <sg_sd.h>
EXTERN SDE_ERR sdDrvGetMidiTimingCounter( Uint32 *count_num);
```

Purpose

Gets count value counted by sound driver based on special messages of sound data.

Parameter

count_num

...

Pointer of variables to store the Sound Driver count.

Return

SDE_ERR_NO_INIT

...

The Sound Library has not been initialized.

Remarks

This function can be used to use the special timing intended by the sound creator.

Reference

sdDrvGetVer

Prototype

```
#include <sg_sd.h>
EXTERN SDE_ERR sdDrvGetVer( SDS_VER *drv_ver );
```

Purpose

Gets the Sound Driver version.

Parameter

drv_ver

...

Pointer to storage for the Sound Driver version information.

Return

SDE_ERR_NO_INIT

...

The Sound Library has not been initialized.

Remarks

Executes only after sdDrvInit and sdLibInit have been executed to initialize the API.

Reference

sdDrvInit

Initialize Sound Driver.

Prototype

```
#include <sg_sd.h>
EXTERN SDE_ERR sdDrvInit( SDMEMBLK handle);
```

Purpose

Initializes the Sound Driver.

Parameter

handle

...

Memory Block handle for the Sound Driver.

Return

Error code

Remarks

If a callback function has been registered by setting memory block information, it can be called after downloading and initializing the sound driver.

Reference

sdLibInit must be executed first to initialize the sound library.

sdLibGetVer

Get Sound Library version.

Prototype

```
#include <sg_sd.h>
EXTERN SDE_ERR sdLibGetVer( SDS_VER *lib_ver);
```

Purpose

Gets the Sound Library version.

Parameter

lib_ver

...

Pointer to storage for the Sound Library version information.

Return

Error code

Remarks

Reference

sdLibInit

Initialize Sound Library.

Prototype

```
#include <sg_sd.h>
EXTERN SDE_ERR sdLibInit( Void *wrk_ptr, Sint32 mem_blk_handle_max, Sint32
second_host_cmd_max);
```

Purpose

Initializes the Sound Library.

Parameter

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

Error code

Remarks

This function must be executed before using the sound library.

The Sound Driver is initialized by sdDrvInit

Reference

sdDrvInit

sdSysFlushHostCmd

Send Host Command immediately to ARM7 Sound Driver.

Prototype

```
#include <sg_sd.h>
EXTERN SDE_ERR sdSysFlushHostCmd( Void );
```

Purpose

Sends communication packets (Host Command) to the ARM7 sound driver, buffered by the sound library.

Parameter

Return

Remarks

Sends the number that were not yet received by the sound driver.

Reference

Some time is required for the ARM7 sound driver to process the sent data, so the response is delayed slightly (not more than several milliseconds).

sdSysServer

Server function to register V interrupt.

Prototype

```
#include <sg_sd.h>
EXTERN SDE_ERR sdSysServer( Void );
```

Purpose

Server function to act during V synch, to actually communicate with ARM7.

Parameter

Return

Remarks

If a callback function has been registered by setting memory block information, it can be called after initializing the downloaded sound driver/library.

Reference

15. Global Sound Control API

sdMidiClosePort

Releases MIDI port access permission.

Prototype

```
#include <sg_sd.h>
EXTERN SDE_ERR sdMidiClosePort( SDMIDI handle);
```

Purpose

Releases MIDI port access permission.

Parameter

handle	...	MIDI port handle to be released.
--------	-----	----------------------------------

Return

SDE_ERR_NO_INIT	...	The Sound Library has not been initialized.
SDE_ERR_HANDLE_NULL	...	The handle is NULL.

Remarks

Reference

SDMIDI

sdMidiContinue

Resumes MIDI sequence play.

Prototype

```
#include <sg_sd.h>
EXTERN SDE_ERR sdMidiContinue( SDMIDI handle);
```

Purpose

Resumes MIDI sequence play after pause.

Parameter

handle	...	Object port handle
--------	-----	--------------------

Return

SDE_ERR_NO_INIT	...The Sound Library has not been initialized.
SDE_ERR_HANDLE_NULL...	The handle is NULL.
SDE_ERR_HOST_CMD_BUF_NO_ENOUGH...	No more host commands can be accepted.

Remarks

sdMidiPause must be called first for pause/resume operation.

Reference

sdMidiGetCurAdr Gets the current play address of a MIDI sequence.

Prototype

```
#include <sg_sd.h>
EXTERN SDE_ERR sdMidiGetCurAdr( SDMIDI handle, Sint32 *cur_adr);
```

Purpose

Gets the current play address of a MIDI sequence.

Parameters

handle	...	Obtained port handle
cur_adr	...	Current playing address

Return

SDE_ERR_NO_INIT	...	The Sound Library has not been initialized.
SDE_ERR_HANDLE_NULL...		The handle is NULL.

Remarks

Accuracy is only approximate.

Reference

sdMidiGetStat

Gets MIDI port status.

Prototype

```
#include <sg_sd.h>
EXTERN SDE_ERR sdMidiGetStat( SDMIDI handle, SDS_MIDI_STAT *midi_stat);
```

Purpose

Gets MIDI port status.

Parameters

handle	...	Object port handle
midi_stat	...	Pointer to storage for obtained MIDI port status.

Return

SDE_ERR_NO_INIT	...	The Sound Library has not been initialized.
SDE_ERR_HANDLE_NULL...		The handle is NULL.

Remarks

Reference

SDS_MIDI_STAT

sdMidiGetTotalBeatTime Gets the total beat time of a MIDI sequence.

Prototype

```
#include <sg_sd.h>
EXTERN SDE_ERR sdMidiGetTotalBeatTime( SDMIDI handle, Uint32 *cur_beat_time);
```

Purpose

Gets the total beat time of a MIDI sequence.

Parameters

handle	...Object port handle
cur_beat_time ...	Pointer to storage for obtained total playing beat number

Return

SDE_ERR_NO_INIT	...The Sound Library has not been initialized.
SDE_ERR_HANDLE_NULL...	The handle is NULL.

Remarks

Reference

sdMidiOpenPort

Gets MIDI port access permission.

Prototype

```
#include <sg_sd.h>
EXTERN SDE_ERR sdMidiOpenPort( SDMIDI *handle);
```

Purpose

Gets MIDI port access permission.

Parameter

handle	...	Pointer to storage for the port handle for access
--------	-----	---

Return

SDE_ERR_NO_INIT	...	The Sound Library has not been initialized.
-----------------	-----	---

Remarks

Reference

SDMIDI

sdMidiPause

Pauses MIDI sequence play.

Prototype

```
#include <sg_sd.h>
SDE_ERR sdMidiPause( SDMIDI handle);
```

Purpose

Pauses MIDI sequence play.

Parameter

handle	...	Object port handle
--------	-----	--------------------

Return

- | | |
|-----------------------------------|--|
| SDE_ERR_NO_INIT | ...The Sound Library has not been initialized. |
| SDE_ERR_HANDLE_NULL... | The handle is NULL. |
| SDE_ERR_HOST_CMD_BUF_NO_ENOUGH... | No more host commands can be accepted. |

Remarks

Reference

sdMidiPlay

Plays a MIDI sequence.

Prototype

```
#include <sg_sd.h>
EXTERN SDE_ERR sdMidiPlay( SDMIDI handle, Sint8 bank_num, const Sint8 data_num, Sint8
priority);
```

Purpose

Plays a MIDI sequence.

Parameters

handle	...	Object port handle
bank_num	...	MIDI sequence bank number to play (0000H to 007FH)
data_num	...	MIDI sequence number to play (0000H to 007FH)
priority	...	Priority level: 0000H (always take priority), 0001H (lowest) to 000FH (highest)

Return

SDE_ERR_NO_INIT	...	The Sound Library has not been initialized.
SDE_ERR_HANDLE_NULL...		The handle is NULL.
SDE_ERR_PRM_OVER_RANGE...		Parameter out of range.
SDE_ERR_HOST_CMD_BUF_NO_ENOUGH...		No more host commands can be accepted.

Remarks

Reference

SDMIDI

sdMidiResetAllPrm

Resets all MIDI ports.

Prototype

```
#include <sg_sd.h>
EXTERN SDE_ERR sdMidiResetAllPrm( Void);
```

Purpose

Resets the level parameters for all MIDI ports.

Parameter

Return

SDE_ERR_NO_INIT ...The Sound Library has not been initialized.
SDE_ERR_HOST_CMD_BUF_NO_ENOUGH...No more host commands can be accepted.

Remarks

All levels are reset to 0 (center).

Reference

sdMidiResetPrm

Resets a MIDI port.

Prototype

```
#include <sg_sd.h>
EXTERN SDE_ERR sdMidiResetPrm( SDMIDI handle);
```

Purpose

Resets the level parameters for a MIDI port.

Parameter

handle	...Object port handle
--------	-----------------------

Return

SDE_ERR_NO_INIT	...	The Sound Library has not been initialized.
SDE_ERR_HANDLE_NULL...		The handle is NULL.
SDE_ERR_HOST_CMD_BUF_NO_ENOUGH...		No more host commands can be accepted.

Remarks

All levels are reset to 0 (center).

Reference

sdMidiSendMes

Sends a MIDI message to a MIDI port.

Prototype

```
#include <sg_sd.h>
EXTERN SDE_ERR sdMidiSendMes( SDMIDI handle, const SDS_MIDI_MES *midi_mes_ptr, const Sint8
priority);
```

Purpose

Sends a MIDI message to a MIDI port.

Parameters

handle	...	Object port handle
midi_mes_ptr	...	MIDI message to be sent
priority	...	MIDI message priority level: range is 0 (always take priority), or 1 (lowest) to 000FH (highest).

Return

SDE_ERR_NO_INIT	...The Sound Library has not been initialized.
SDE_ERR_HANDLE_NULL...	The handle is NULL.
SDE_ERR_HOST_CMD_BUF_NO_ENOUGH...	No more host commands can be accepted.

Remarks

Reference

sdMidiSetDrctLev

Sets the direct level of a MIDI port.

Prototype

```
#include <sg_sd.h>
EXTERN SDE_ERR sdMidiSetDrctLev( SDMIDI handle, const Sint8 lev);
```

Purpose

Sets the direct (without sound effects) level of a MIDI port.

Parameters

handleObject port handle
lev	...	Direct level to set: range is -0000H (min) to 0000H (mid) to 000FH (max).

Return

SDE_ERR_NO_INIT	...The Sound Library has not been initialized.
SDE_ERR_HANDLE_NULL...	The handle is NULL.
SDE_ERR_PRM_OVER_RANGE...	Parameter out of range.
SDE_ERR_HOST_CMD_BUF_NO_ENOUGH...	No more host commands can be accepted.

Remarks

Reference

sdMidiSetFxLev

Sets the FX level of a MIDI port.

Prototype

```
#include <sg_sd.h>
EXTERN SDE_ERR sdMidiSetFxLev( SDMIDI handle, const Sint8 fx_lev);
```

Purpose

Sets the FX level of a MIDI port.

Parameters

handle	...	Object port handle
fx_lev	...	FX level to set: range is -0000H (min) to 0000H (mid) to 000FH (max).

Return

SDE_ERR_NO_INIT	...The Sound Library has not been initialized.
SDE_ERR_HANDLE_NULL...	The handle is NULL.
SDE_ERR_PRM_OVER_RANGE...	Parameter out of range.
SDE_ERR_HOST_CMD_BUF_NO_ENOUGH...	No more host commands can be accepted.

Remarks

Reference

sdMidiSetMes

Creates a MIDI message to send to a MIDI port.

Prototype

```
#include <sg_sd.h>
EXTERN SDE_ERR sdMidiSetMes( SDMIDI handle, SDS_MIDI_MES *midi_mes_ptr, const Uint8
midi_mes,... );
```

Purpose

Creates a MIDI message to send to a MIDI port.

Parameters

handle	...	Object port handle
midi_mes_ptr	...	Pointer to storage containing created MIDI message.
midi_mes	...	MIDI message

Return

SDE_ERR_NO_INIT	...The Sound Library has not been initialized.
SDE_ERR_HANDLE_NULL...	The handle is NULL.

Remarks

The variable length parameter allows data to be written continuously as required for the MIDI message.

Reference

sdMidiSetPan Sets the panpot (sound localization) of a MIDI port.

Prototype

```
#include <sg_sd.h>
EXTERN SDE_ERR sdMidiSetPan( SDMIDI handle, const Sint8 pan, Sint16 fade_time);
```

Purpose

Sets the panpot (sound localization) of a MIDI port.

Parameters

handle		...Object port handle
pan	...	Target panpot: range is -007FH (left) to 0000H (center) to 007FH (right).
fade_time	...	Time to reach target. Units are milliseconds, from 0000H (fastest) to 7FFFH (slowest).

Return

SDE_ERR_NO_INIT	...The Sound Library has not been initialized.
SDE_ERR_HANDLE_NULL...	The handle is NULL.
SDE_ERR_HOST_CMD_BUF_NO_ENOUGH...	No more host commands can be accepted.

Remarks

Reference

sdMidiSetPitch

Sets the play pitch of a MIDI port.

Prototype

```
#include <sg_sd.h>
EXTERN SDE_ERR sdMidiSetPitch( SDMIDI handle, const Sint16 pitch, const Sint32 fade_time);
```

Purpose

Sets the pitch of a MIDI port.

Parameters

handle		...Object port handle
pitch	...	Target play pitch. Half tone up = 0100H, half tone down = -0100H.
fade_time	...	Time to reach target play speed. Units are milliseconds, from 0000H (fastest) to 7FFFH (slowest).

SDE_ERR_HOST_CMD_BUF_NO_ENOUGH...No more host commands can be accepted.

Return

SDE_ERR_NO_INIT	...The Sound Library has not been initialized.
SDE_ERR_HANDLE_NULL...	The handle is NULL.

Remarks

Reference

sdMidiSetSpeed

Sets the play speed of a MIDI port.

Prototype

```
#include <sg_sd.h>
SDE_ERR sdMidiSetSpeed( SDMIDI handle, const Sint16 speed, cibst Sint32 fade_time);
```

Purpose

Sets the speed of a MIDI port.

Parameters

handle	...	Object port handle
speed	...	Target play speed, calculated as follows: speed = m x 1000H - 1000H where multiplier m = 1/64 to 3. Valid calculation results must be between 100H (min) and 1FFFH (max).
fade_time	...	Time to reach target play speed. Units are milliseconds, from 0000H (fastest) to 7FFFH (slowest).

Return

SDE_ERR_NO_INIT	...The Sound Library has not been initialized.
SDE_ERR_HANDLE_NULL...	The handle is NULL.
SDE_ERR_HOST_CMD_BUF_NO_ENOUGH...	No more host commands can be accepted.

Remarks

MIDI message sending API function sdMidiSendMes is not affected.

Reference

sdMidiStop

Stops MIDI sequence play.

Prototype

```
#include <sg_sd.h>
EXTERN SDE_ERR sdMidiStop( SDMIDI handle);
```

Purpose

Stops MIDI sequence play.

Parameter

handle	...	Object port handle
--------	-----	--------------------

Return

- | | |
|-----------------------------------|--|
| SDE_ERR_NO_INIT | ...The Sound Library has not been initialized. |
| SDE_ERR_HANDLE_NULL... | The handle is NULL. |
| SDE_ERR_HOST_CMD_BUF_NO_ENOUGH... | No more host commands can be accepted. |

Remarks

Reference

sdMidiStopAll

Stops play on all MIDI ports.

Prototype

```
#include <sg_sd.h>
extern SDE_ERR sdMidiStopAll( Void);
```

Purpose

Stops play on all MIDI ports.

Parameter

None

Return

SDE_ERR_NO_INIT ...The Sound Library has not been initialized.
SDE_ERR_HOST_CMD_BUF_NO_ENOUGH...No more host commands can be accepted.

Remarks

Reference

sdMidiSetVol

Sets the play volume of a MIDI port.

Prototype

```
#include <sg_sd.h>
EXTERN SDE_ERR sdMidiSetVol( SDMIDI handle, const Sint8 vol, const Sint32 fade_time);
```

Purpose

Sets the volume of a MIDI port.

Parameters

handleObject port handle
vol	...	Target volume: range is -007FH (min) to 0000H (mid) to 007FH (max).
fade_time	...	Time to reach target volume. Units are milliseconds, from 0000H (fastest) to 7FFFH (slowest).
SDE_ERR_HOST_CMD_BUF_NO_ENOUGH...No more host commands can be accepted.		

Return

SDE_ERR_NO_INIT	...The Sound Library has not been initialized.
SDE_ERR_HANDLE_NULL...	The handle is NULL.
SDE_ERR_HOST_CMD_BUF_NO_ENOUGH...No more host commands can be accepted.	

Remarks

Very low volume levels may be inaudible. Also, setting a higher level may not increase volume (if, for example, the level of the original data was already set high).

Reference

sdPstmClosePort Releases PCM Stream port access permission.

Prototype

```
#include <sg_sd.h>
EXTERN SDE_ERR sdPstmClosePort( SDPSTM handle);
```

Purpose

Releases PCM Stream port access permission.

Parameter

handle	...Port number of PCM Stream port to release
--------	--

Return

SDE_ERR_NO_INIT	...The Sound Library has not been initialized.
SDE_ERR_HANDLE_NULL...	The handle is NULL.

Remarks

Reference

SDPSTM

sdPstmGetCurAdr Gets the current play address of a PCM Stream.

Prototype

```
#include <sg_sd.h>
EXTERN SDE_ERR sdPstmGetCurAdr( SDPSTM handle, const Sint8 target_ch, Sint32
*cur_smp_adr );
```

Purpose

Gets the current play address of a PCM Stream.

Parameters

handleObject handle
target_ch	...	Channel number to get current playing address
cur_smp_adr	...	Waveform address that AICA is currently reading

Return

SDE_ERR_NO_INIT	...The Sound Library has not been initialized.
SDE_ERR_HANDLE_NULL...	The handle is NULL.
SDE_ERR_HOST_CMD_BUF_NO_ENOUGH...	No more host commands can be accepted.

Remarks

Accuracy is only approximate.

Reference

sdPstmGetStat

Gets PCM Stream port status.

Prototype

```
#include <sg_sd.h>
EXTERN SDE_ERR sdPstmGetStat( SDPSTM handle, const Sint8 target_ch, SDS_PSTM_CH_STAT
*pstm_stat);
```

Purpose

Gets PCM Stream port channel status.

Parameters

handle	...	Object handle
target_ch	...	Channel number for which to get current playing address
pstm_stat	...	Pointer to storage for obtained channel status of PCM Stream

Return

SDE_ERR_NO_INIT	...	The Sound Library has not been initialized.
SDE_ERR_HANDLE_NULL...		The handle is NULL.

Remarks

Reference

SDS_PSTM_CH_STAT

sdPstmGetTotalSmpFrame Gets total PCM Stream play sample frames.

Prototype

```
#include <sg_sd.h>
EXTERN SDE_ERR sdPstmGetTotalSmpFrame( SDPSTM handle, const Sint8 target_ch, Uint32
*cur_smp_frame);
```

Purpose

Gets the total sample frames playing on the PCM Stream.

Parameters

handle	...	Object handle
target_ch	...	Channel number for which to get current playing address
cur_smp_frame	...	Pointer to storage for obtained total playing sample frame number

Return

SDE_ERR_NO_INIT	...The Sound Library has not been initialized.
SDE_ERR_HANDLE_NULL...	The handle is NULL.
SDE_ERR_HOST_CMD_BUF_NO_ENOUGH...	No more host commands can be accepted.

Remarks

Accuracy is only approximate.

Reference

sdPstmIsTransferWaveData

Check if okay to transfer a waveform to a PCM Stream port.

Prototype

```
#include <sg_sd.h>
EXTERN SDE_ERR sdPstmIsTransferWaveData( SDPSTM handle, const Sint8 target_ch, Sint32
data_sz, Bool *flg);
```

Purpose

Check if okay to transfer a waveform to a PCM Stream port.

Parameters

handle		...Object handle
target_ch	...	Object channel
data_sz	...	Size of waveform desired to transfer, in bytes
flg	...	Pointer to storage for test result: true if transfer possible, or false if not

Return

SDE_ERR_NO_INIT	...The Sound Library has not been initialized.
SDE_ERR_HANDLE_NULL...	The handle is NULL.
SDE_ERR_PRM_OVER_RANGE...	Parameter out of range.

Remarks

- The waveform cannot be transferred until flg is returned true.
- If PCM Stream play has not been started, a size larger than 07000H cannot be specified.
- Even if this API function is called several times, if PCM Stream play has not been started, the ring buffer is not refreshed, but overwritten instead (that is, the refresh position does not advance from the ring buffer head address).

Reference

sdPstmOpenPort

Gets PCM Stream port access permission.

Prototype

```
#include <sg_sd.h>
extern SDE_ERR sdPstmOpenPort( SDPSTM *handle, const Sint8 use_ch, ...);
```

Purpose

Gets PCM Stream port access permission.

Parameters

handle	...	Pointer to storage for obtained port number
use_ch	...	Channel number to use

Return

SDE_ERR_NO_INIT	...	The Sound Library has not been initialized.
SDE_ERR_HANDLE_NULL...		The handle is NULL.
SDE_ERR_PRM_OVER_RANGE...		Parameter out of range.

Remarks

Specify the required ring buffer number as a variable-length parameter.

Reference

SDPSTM

sdPstmPlay

Plays a PCM Stream.

Prototype

```
#include <sg_sd.h>
EXTERN sdPstmPlay( SDPSTM handle, const SDE_PCM_TYPE pcm_Type, const Uint16 freq, const Sint8 priority);
```

Purpose

Plays a PCM Stream.

Parameters

handle	...	Object handle
pcm_Type	...	PCM type of playing data
freq	...	Standard play rate. Only the following are valid: SDD_PSTM_FREQ_44100 for 44,100 Hz SDD_PSTM_FREQ_22050 for 22,050 Hz SDD_PSTM_FREQ_11025 for 11,025 Hz
priority	...	Priority level: 0000H (always take priority), 0001H (lowest) to 000FH (highest)

Return

SDE_ERR_NO_INIT	...The Sound Library has not been initialized.
SDE_ERR_HANDLE_NULL...	The handle is NULL.
SDE_ERR_HOST_CMD_BUF_NO_ENOUGH...	No more host commands can be accepted.

Remarks

Reference

sdPstmResetAllPrm

Resets all PCM Stream ports.

Prototype

```
#include <sg_sd.h>
EXTERN SDE_ERR sdPstmResetAllPrm( Void);
```

Purpose

Resets the level parameters for all PCM Stream ports.

Parameter

Return

SDE_ERR_NO_INIT ...The Sound Library has not been initialized.
SDE_ERR_HOST_CMD_BUF_NO_ENOUGH...No more host commands can be accepted.

Remarks

Reference

sdPstmResetPrm

Resets a PCM Stream port.

Prototype

```
#include <sg_sd.h>
EXTERN SDE_ERR sdPstmResetPrm( SDPSTM handle);
```

Purpose

Resets the level parameters for a PCM Stream port.

Parameter

handle	...	Object handle
--------	-----	---------------

Return

SDE_ERR_NO_INIT	...The Sound Library has not been initialized.
SDE_ERR_HANDLE_NULL...	The handle is NULL.
SDE_ERR_HOST_CMD_BUF_NO_ENOUGH...	No more host commands can be accepted.

Remarks

Reference

sdPstmSetBaseVol Sets the base volume of a PCM Stream port.

Prototype

```
#include <sg_sd.h>
EXTERN SDE_ERR sdPstmSetBaseVol( SDPSTM handle, const Sint8 base_vol);
```

Purpose

Sets the base volume of channel's PCM Stream port.

Parameters

handle	...	Object handle
base_vol	...	Base volume: from 0000H (min) to 007FH (max)

Return

SDE_ERR_NO_INIT	...The Sound Library has not been initialized.
SDE_ERR_HANDLE_NULL...	The handle is NULL.

Remarks

Actual output volume of a port results from combining this function setting and the relative volume setting of sdPstmSetVol. Therefore, if this functions sets the volume to 007FH, the volume cannot be further increased by specifying a higher level with sdPstmSetVol

Reference

sdPstmSetDrctLev

Sets the direct level of a PCM Stream port.

Prototype

```
#include <sg_sd.h>
EXTERN SDE_ERR sdPstmSetDrctLev( SDPSTM handle, const Sint8 drct_lev);
```

Purpose

Sets the direct (without sound effects) level of a PCM Stream port.

Parameters

handle	...	Object handle
drct_lev	...	Direct level to set: range is -0000H (min) to 0000H (mid) to 000FH (max).

Return

SDE_ERR_NO_INIT	...	The Sound Library has not been initialized.
SDE_ERR_HANDLE_NULL...		The handle is NULL.
SDE_ERR_HOST_CMD_BUF_NO_ENOUGH...		No more host commands can be accepted.

Remarks

Reference

sdPstmSetFxCh

Sets the FX input for a PCM Stream port.

Prototype

```
#include <sg_sd.h>
EXTERN SDE_ERR sdPstmSetFxCh( SDPSTM handle, const Sint8 fix_in_ch, const Sint8
fx_base_lev);
```

Purpose

Sets the FX output destination for a PCM Stream port.

Parameters

handle	...	Object handle
fix_in_ch	...	DSP input channel number to connect
fx_base_lev	...	Base FX level: from 0000H (min) to 000FH (max)

Return

SDE_ERR_NO_INIT	...The Sound Library has not been initialized.
SDE_ERR_HANDLE_NULL...	The handle is NULL.
SDE_ERR_PRM_OVER_RANGE...	Parameter out of range.
SDE_ERR_HOST_CMD_BUF_NO_ENOUGH...	No more host commands can be accepted.

Remarks

The ch and lev parameters of this API function are absolute settings.

Reference

sdPstmSetFxLev Sets the FX level of a PCM Stream port.

Prototype

```
#include <sg_sd.h>
EXTERN SDE_ERR sdPstmSetFxLev( SDPSTM handle, const Sint8 fx_lev);
```

Purpose

Sets the FX level of a PCM Stream port.

Parameters

handle	...	Object handle
fx_lev	...	FX level to set: range is -0000H (min) to 0000H (mid) to 000FH (max).

Return

SDE_ERR_NO_INIT	...The Sound Library has not been initialized.
SDE_ERR_HANDLE_NULL...	The handle is NULL.
SDE_ERR_PRM_OVER_RANGE...	Parameter out of range.
SDE_ERR_HOST_CMD_BUF_NO_ENOUGH...	No more host commands can be accepted.

Remarks

Reference

sdPstmSetPan

Sets the panpot of a PCM Stream port.

Prototype

```
#include <sg_sd.h>
EXTERN SDE_ERR sdPstmSetPan( SDPSTM handle, const Sint8 target_ch, const Sint8 pan, const
Sint16 fade_time);
```

Purpose

Sets the panpot (sound localization) of a PCM Stream port channel.

Parameters

handle		...Object handle
target_ch	...	Object channel
pan	...	Target panpot: range is -007FH (left) to 0000H (center) to 007FH (right).
fade_time	...	Time to reach target. Units are milliseconds, from 0000H (fastest) to 7FFFH (slowest).

Return

SDE_ERR_NO_INIT	...The Sound Library has not been initialized.
SDE_ERR_HANDLE_NULL...	The handle is NULL.
SDE_ERR_HOST_CMD_BUF_NO_ENOUGH...	No more host commands can be accepted.

Remarks

The left channel parameter is invalid for sdPstmSetPan, and is overwritten by this API function.

Reference

sdPstmSetPitch

Sets the play pitch of a PCM Stream port.

Prototype

```
#include <sg_sd.h>
EXTERN SDE_ERR sdPstmSetPitch( SDPSTM handle, const Sint16 pitch, const Sint32 fade_time);
```

Purpose

Sets the play pitch of a PCM Stream port.

Parameters

handle		...Object handle
pitch	...	Target play pitch. Half tone up = 0100H, half tone down = -0100H.
fade_time	...	Time to reach target play speed. Units are milliseconds, from 0000H (fastest) to 7FFFH (slowest).

Return

SDE_ERR_NO_INIT	...The Sound Library has not been initialized.
SDE_ERR_HANDLE_NULL...	The handle is NULL.
SDE_ERR_HOST_CMD_BUF_NO_ENOUGH...	No more host commands can be accepted.

Remarks

Higher/lower pitch results in faster/slower play, respectively.

Reference

sdPstmSetSpeed

Sets the play speed of a PCM Stream port.

Prototype

```
#include <sg_sd.h>
EXTERN SDE_ERR sdPstmSetSpeed( SDPSTM handle, const Sint16 speed, const Sint32 fade_time);
```

Purpose

Sets the play speed of a PCM Stream port.

Parameters

handle	...	Object handle
speed	...	Target play speed: range is -18FFH (slowest) to 0000H (mid) to 18FFH (fastest), with half speed = -0C00H and double speed = 0C00H.

Return

SDE_ERR_NO_INIT	...The Sound Library has not been initialized.
SDE_ERR_HANDLE_NULL...	The handle is NULL.
SDE_ERR_HOST_CMD_BUF_NO_ENOUGH...	No more host commands can be accepted.

Remarks

Faster/slower play results in higher/lower pitch, respectively.

Reference

sdPstmSetVol

Sets the play volume of a PCM Stream port.

Prototype

```
#include <sg_sd.h>
EXTERN SDE_ERR sdPstmSetVol( SDPSTM handle, const Sint8 ch_num, const Sint8 vol, const
Sint32 fade_time);
```

Purpose

Sets the play volume of a PCM Stream port channel.

Parameters

handle	...	Object handle
ch_num	...	Object channel
vol	...	Target volume: range is -007FH (min) to 0000H (mid) to 007FH (max).
fade_time	...	Time to reach target volume. Units are milliseconds, from 0000H (fastest) to 7FFFH (slowest).

Return

SDE_ERR_NO_INIT	...The Sound Library has not been initialized.
SDE_ERR_HANDLE_NULL...	The handle is NULL.
SDE_ERR_HOST_CMD_BUF_NO_ENOUGH...	No more host commands can be accepted.

Remarks

The left channel parameter is invalid for sdPstmSetVol, and is overwritten by this API function.

Reference

sdPstmStop

Stops PCM Stream play.

Prototype

```
#include <sg_sd.h>
EXTERN SDE_ERR sdPstmStop( SDPSTM handle);
```

Purpose

Stops PCM Stream play.

Parameter

handle	...PCM Stream port to stop playing
--------	------------------------------------

Return

SDE_ERR_NO_INIT	...The Sound Library has not been initialized.
SDE_ERR_HANDLE_NULL...	The handle is NULL.
SDE_ERR_HOST_CMD_BUF_NO_ENOUGH...	No more host commands can be accepted.

Remarks

Reference

sdPstmStopAll

Stops play on all PCM Stream ports.

Prototype

```
#include <sg_sd.h>
EXTERN SDE_ERR sdPstmStopAll( Void);
```

Purpose

Stops play on all PCM Stream ports.

Parameter

None

Return

SDE_ERR_NO_INIT ...The Sound Library has not been initialized.
SDE_ERR_HOST_CMD_BUF_NO_ENOUGH...No more host commands can be accepted.

Remarks

Reference

sdPstmTransferWaveData Transfers a waveform to a PCM Stream port.

Prototype

```
#include <sg_sd.h>
EXTERN SDE_ERR sdPstmTransferWaveData( SDPSTM handle, const Sint8 ch, SDS_MEMBLK memblk);
```

Purpose

Transfers a waveform to a PCM Stream port.

Parameters

handle		...Object handle
ch_num	...	Object channel
memblk	...	Memory Block handle
SDE_ERR_HOST_CMD_BUF_NO_ENOUGH...No more host commands can be accepted.		

Return

SDE_ERR_NO_INIT	...	The Sound Library has not been initialized.
SDE_ERR_HANDLE_NULL...		The handle is NULL.

Remarks

Call sdPstmIsTransferWaveData to check whether transfer is possible before transferring. If PCM Stream play has not been started, a size larger than 07000H cannot be specified.

Reference

sdShotClosePort Releases One-Shot port access permission.

Prototype

```
#include <sg_sd.h>
EXTERN SDE_ERR sdShotClosePort( SDSHOT handle);
```

Purpose

Releases One-Shot port access permission.

Parameter

handle	...	Handle of One-Shot port to be released
--------	-----	--

Return

SDE_ERR_NO_INIT	...	The Sound Library has not been initialized.
SDE_ERR_HANDLE_NULL...		The handle is NULL.

Remarks

Reference

sdShotGetCurAdr

Gets the current play address of a One-Shot.

Prototype

```
#include <sg_sd.h>
EXTERN SDE_ERR sdShotGetCurAdr( SDSHOT handle, Sint32 *cur_smp_adr);
```

Purpose

Gets the current play address of a One-Shot.

Parameters

handle	...Object handle
cur_smp_adr ...	Pointer to storage for the currently reading waveform

Return

SDE_ERR_NO_INIT	...The Sound Library has not been initialized.
SDE_ERR_HANDLE_NULL...	The handle is NULL.

Remarks

Accuracy is only approximate.

Reference

sdShotSetStat

Gets One-Shot port status.

Prototype

```
#include <sg_sd.h>
EXTERN SDE_ERR sdShotGetStat( SDSHOT handle, SDS_SHOT_STAT *shot_stat);
```

Purpose

Gets One-Shot port status.

Parameters

handle	...Object handle
shot_stat ...	Pointer to storage of obtained One-Shot port status

Return

SDE_ERR_NO_INIT	...The Sound Library has not been initialized.
SDE_ERR_HANDLE_NULL...	The handle is NULL.

Remarks

Reference

SDS_SHOT_STAT

sdShotGetTotalSmpFrame Gets total One-Shot play sample frames.

Prototype

```
#include <sg_sd.h>
EXTERN SDE_ERR sdShotGetTotalSmpFrame( SDSHOT handle, Uint32 *cur_smp_frame);
```

Purpose

Gets total currently playing One-Shot sample frames.

Parameters

handle	...	Object handle
cur_smp_frame	...	Pointer to storage for obtained total playing sample frame number

Return

SDE_ERR_NO_INIT	...	The Sound Library has not been initialized.
SDE_ERR_HANDLE_NULL...		The handle is NULL.

Remarks

Accuracy is only approximate.

Reference

sdShotOpenPort

Gets One-Shot port access permission.

Prototype

```
#include <sg_sd.h>
EXTERN SDE_ERR sdShotOpenPort( SDSTM *handle);
```

Purpose

Gets One-Shot port access permission.

Parameter

handle	...Obtained port handle
--------	-------------------------

Return

SDE_ERR_NO_INIT	...The Sound Library has not been initialized.
-----------------	--

Remarks

Reference

SDSHOT

sdShotPlay

Plays a One-Shot.

Prototype

```
#include <sg_sd.h>
EXTERN SDE_ERR sdShotPlay( SDSHOT handle, const Sint8 bank_num, const Sint8 data_num, const
Sint8 priority);
```

Purpose

Plays a One-Shot.

Parameters

handle		...Object port handle
bank_num	...	One-Shot bank number to play (0000H to 007FH)
data_num	...	One-Shot number to play (0000H to 007FH)
priority	...	Priority level: 0000H (always take priority), 0001H (lowest) to 000FH (highest)

Return

SDE_ERR_NO_INIT	...The Sound Library has not been initialized.
SDE_ERR_HANDLE_NULL...	The handle is NULL.
SDE_ERR_PRM_OVER_RANGE...	Parameter out of range.
SDE_ERR_HOST_CMD_BUF_NO_ENOUGH...	No more host commands can be accepted.

Remarks

Reference

sdShotResetAllPrm

Resets all One-Shot ports.

Prototype

```
#include <sg_sd.h>
EXTERN SDE_ERR sdShotResetAllPrm( Void);
```

Purpose

Resets the parameters for all One-Shot ports.

Parameter

Return

SDE_ERR_NO_INIT	...The Sound Library has not been initialized.
SDE_ERR_HANDLE_NULL...	The handle is NULL.
SDE_ERR_HOST_CMD_BUF_NO_ENOUGH...	No more host commands can be accepted.

Remarks

All levels are reset to 0 (center).

Reference

sdShotResetPrm

Resets a One-Shot port.

Prototype

```
#include <sg_sd.h>
EXTERN SDE_ERR sdShotResetPrm( SDSHOT handle);
```

Purpose

Resets the level parameters for a One-Shot port.

Parameter

handle	...	Object handle
--------	-----	---------------

Return

SDE_ERR_NO_INIT	...The Sound Library has not been initialized.
SDE_ERR_HANDLE_NULL...	The handle is NULL.
SDE_ERR_HOST_CMD_BUF_NO_ENOUGH...	No more host commands can be accepted.

Remarks

All levels are reset to 0 (center).

Reference

sdShotSetDrctLev

Sets the direct level of a One-Shot port.

Prototype

```
#include <sg_sd.h>
EXTERN SDE_ERR sdShotSetDrctLev( SDSHOT handle, const Sint8 drct_lev);
```

Purpose

Sets the direct (without sound effects) level of a One-Shot port.

Parameters

handle	...	Object handle
drct_lev	...	Direct level to set: range is -0000H (min) to 0000H (mid) to 000FH (max).

Return

SDE_ERR_NO_INIT	...The Sound Library has not been initialized.
SDE_ERR_HANDLE_NULL...	The handle is NULL.
SDE_ERR_HOST_CMD_BUF_NO_ENOUGH...	No more host commands can be accepted.

Remarks

Reference

sdShotSetFxCh Sets the FX output destination for a One-Shot port.

Prototype

```
#include <sg_sd.h>
EXTERN SDE_ERR sdShotSetFxCh( SDSHOT handle, const Sint8 fx_in_ch, const Sint8
fx_base_lev);
```

Purpose

Sets the FX output destination for a One-Shot port.

Parameters

handle	...	Object handle
fx_in_ch	...	DSP input channel number to connect
fx_base_lev	...	Base FX level: from 0000H (min) to 000FH (max)

Return

SDE_ERR_NO_INIT	...The Sound Library has not been initialized.
SDE_ERR_HANDLE_NULL...	The handle is NULL.
SDE_ERR_PRM_OVER_RANGE...	Parameter out of range.
SDE_ERR_HOST_CMD_BUF_NO_ENOUGH...	No more host commands can be accepted.

Remarks

The ch and lev parameters of this API function are absolute settings.

Reference

sdShotSetFxLev

Sets the FX level of a One-Shot port.

Prototype

```
include <sg_sd.h>
EXTERN SDE_ERR sdShotSetFxLev( SDSHOT handle, const Sint8 fx_lev);
```

Purpose

Sets the FX level of a One-Shot port.

Parameters

handle	...	Object handle
fx_lev	...	FX level to set: range is -0000H (min) to 0000H (mid) to 000FH (max).

Return

SDE_ERR_NO_INIT	...The Sound Library has not been initialized.
SDE_ERR_HANDLE_NULL...	The handle is NULL.
SDE_ERR_PRM_OVER_RANGE...	Parameter out of range.
SDE_ERR_HOST_CMD_BUF_NO_ENOUGH...	No more host commands can be accepted.

Remarks

Reference

sdShotSetPan

Sets the panpot of a One-Shot port.

Prototype

```
include <sg_sd.h>
EXTERN SDE_ERR sdShotSetPan( SDSHOT handle, const Sint8 pan, const Sint32 fade_time);
```

Purpose

Sets the panpot (sound localization) of a One-Shot port.

Parameters

handle		...Object handle
pan	...	Target panpot: range is -007FH (left) to 0000H (center) to 007FH (right).
fade_time	...	Time to reach target. Units are milliseconds, from 0000H (fastest) to 7FFFH (slowest).

Return

SDE_ERR_NO_INIT	...The Sound Library has not been initialized.
SDE_ERR_HANDLE_NULL...	The handle is NULL.
SDE_ERR_HOST_CMD_BUF_NO_ENOUGH...	No more host commands can be accepted.

Remarks

Reference

sdShotSetPitch

Sets the play pitch of a One-Shot port.

Prototype

```
include <sg_sd.h>
EXTERN SDE_ERR sdShotSetPitch( SDSHOT handle, const Sint16 pitch, const Sint32 fade_time);
```

Purpose

Sets the play pitch of a One-Shot port.

Parameters

handle		...Object handle
pitch	...	Target play pitch. Half tone up = 0100H, half tone down = -0100H.
fade_time	...	Time to reach target pitch. Units are milliseconds, from 0000H (fastest) to 7FFFH (slowest).

Return

SDE_ERR_NO_INIT	...The Sound Library has not been initialized.
SDE_ERR_HANDLE_NULL...	The handle is NULL.
SDE_ERR_HOST_CMD_BUF_NO_ENOUGH...	No more host commands can be accepted.

Remarks

Higher/lower pitch results in faster/slower play, respectively.

Reference

sdShotSetSpeed

Sets the play speed of a One-Shot port.

Prototype

```
include <sg_sd.h>
EXTERN SDE_ERR sdShotSetSpeed( SDSHOT handle, const Sint16 speed, const Sint32 fade_time);
```

Purpose

Sets the play speed of a One-Shot port.

Parameters

handle	...	Object handle
speed	...	Target play speed: range is -18FFH (slowest) to 0000H (mid) to 18FFH (fastest), with half speed = -0C00H and double speed = 0C00H.
fade_time	...	Time to reach target play speed. Units are milliseconds, from 0000H (fastest) to 7FFFH (slowest).

Return

SDE_ERR_NO_INIT	...The Sound Library has not been initialized.
SDE_ERR_HANDLE_NULL...	The handle is NULL.
SDE_ERR_HOST_CMD_BUF_NO_ENOUGH...	No more host commands can be accepted.

Remarks

Faster/slower play results in higher/lower pitch, respectively.

Reference

sdShotSetVol

Sets the play volume of a One-Shot port.

Prototype

```
include <sg_sd.h>
EXTERN SDE_ERR sdShotSetVol( SDSHOT handle, const Sint8 vol, const Sint32 fade_time);
```

Purpose

Sets the play volume of a One-Shot port.

Parameters

handle	...	Object handle
vol	...	Target volume: range is -007FH (min) to 0000H (mid) to 007FH (max).
fade_time	...	Time to reach target volume. Units are milliseconds, from 0000H (fastest) to 7FFFH (slowest).

Return

SDE_ERR_NO_INIT	...The Sound Library has not been initialized.
SDE_ERR_HANDLE_NULL...	The handle is NULL.
SDE_ERR_HOST_CMD_BUF_NO_ENOUGH...	No more host commands can be accepted.

Remarks

Reference

sdShotStop

Stops One-Shot play.

Prototype

```
include <sg_sd.h>
EXTERN SDE_ERR sdShotStop( SDSHOT handle);
```

Purpose

Stops One-Shot play.

Parameter

handle	...	Object port handle
--------	-----	--------------------

Return

SDE_ERR_NO_INIT	...The Sound Library has not been initialized.
SDE_ERR_HANDLE_NULL...	The handle is NULL.
SDE_ERR_PRM_OVER_RANGE...	Parameter out of range.
SDE_ERR_HOST_CMD_BUF_NO_ENOUGH...	No more host commands can be accepted.

Remarks

Reference

sdShotStopAll

Stops play on all One-Shot ports.

Prototype

```
include <sg_sd.h>
EXTERN SDE_ERR sdShotStopAll( Void);
```

Purpose

Stops play on all One-Shot ports.

Parameter

None

Return

SDE_ERR_NO_INIT ...The Sound Library has not been initialized.
SDE_ERR_HOST_CMD_BUF_NO_ENOUGH...No more host commands can be accepted.

Remarks

Reference

16. Sound Data Utility API

sdBankDownload

Downloads various sound data from main memory to sound memory. (Macro.)

Prototype

```
#include <sg_sd.h>
EXTERN SDE_ERR sdBankDownload( SDMEMBLK handle, const SDE_DATA_TYPE bank_type, const Sint8
bank_num);
```

Purpose

Downloads a bank of sound data from main memory to sound memory.

Parameter

handle	...	Handle of the block information for the bank to download
bank_type	...	Bank type to download
bank_num	...	Bank number of download destination

Return

SDE_ERR_NO_INIT	...	The Sound Library has not been initialized.
SDE_ERR_HANDLE_NULL...		The handle was NULL.
SDE_ERR_HANDLE_ILLEGAL_VALUE...		An illegal address was specified for the handle.
SDE_ERR_DATA_ILLEGAL_TYPE...		Illegal data type: a Multi-Unit was not specified, or an attempt was made to download a Multi-Unit with a data ID not recognized by the library.

Remarks

This function cannot be used until sdMultiUnitDownload has been called at least once to download a Multi-Unit file (which creates the memory mapping).

Reference

sdMultiUnitDownload

Downloads a Multi-Unit file from main memory to sound memory.

Prototype

```
#include <sg_sd.h>
EXTERN SDE_ERR sdMultiUnitDownload( SDMEMBLK handle);
```

Purpose

Downloads a Multi-Unit file from main memory to sound memory.

Parameter

handle	...	Handle of the block information for the data to download.
--------	-----	---

Return

SDE_ERR_NO_INIT	...	The Sound Library has not been initialized.
SDE_ERR_HANDLE_NULL	...	The handle was NULL.
SDE_ERR_HANDLE_ILLEGAL_VALUE ...		An illegal address was specified for the handle.
SDE_ERR_DATA_ILLEGAL_TYPE ...		Illegal data type: a Multi-Unit was not specified, or an attempt was made to download a Multi-Unit with a data ID not recognized by the library.

Remarks

Reference

17. Memory Block Transfer API

sdMemBlkCreate

Creates a Memory Block handle.

Prototype

```
#include <sg_sd.h>
EXTERN SDE_ERR sdMemBlkCreate( SDMEMBLK *handle);
```

Purpose

Creates a Memory Block handle.

Parameter

handle	...	Pointer to storage for handle.
--------	-----	--------------------------------

Return

SDE_ERR_NO_INIT	...	The Sound Library has not been initialized.
SDE_ERR_HANDLE_NO_ENOUGH	...	All handle buffers have been used, so no more handles available.

Remarks

If a handle cannot be created, it is set to NULL.

Reference

SDMEMBLK

sdMemBlkDestroy

Destroys a Memory Block handle.

Prototype

```
#include <sg_sd.h>
EXTERN SDE_ERR sdMemBlkDestroy( SDMEMBLK handle);
```

Purpose

Destroy a Memory Block handle.

Parameter

handle	...	Memory Block handle
--------	-----	---------------------

Return

SDE_ERR_NO_INIT	...	The Sound Library has not been initialized.
SDE_ERR_HANDLE_NULL	...	The handle was NULL.
SDE_ERR_HANDLE_ILLEGAL_VALUE...		An illegal address was specified for the handle.

Remarks

Reference

sdMemBlkGetStat

Checks the status of a Memory Block handle.

Prototype

```
#include <sg_sd.h>
EXTERN SDE_ERR sdMemBlkGetStat( SDMEMBLK handle, SDE_MEMBLK_STAT *stat);
```

Purpose

Checks the status of the Memory Block handle.

Parameter

handle	...	Memory Block handle
stat	...	Pointer to the resulting block status data.

Return

SDE_ERR_NO_INIT	...	The Sound Library has not been initialized.
SDE_ERR_HANDLE_NULL...		The handle was NULL.
SDE_ERR_HANDLE_ILLEGAL_VALUE...		An illegal address was specified for the handle.

Remarks

Reference

sdMemBlkSetPrm

Sets Memory Block handle parameters.

Prototype

```
#include <sg_sd.h>
EXTERN SDE_ERR sdMemBlkSetPrm( SDMEMBLK handle, const Void *src_blk_ptr, const Sint32
src_blk_sz, const SD_MEMBLK_CALLBACK_FUNC callback_func, const Void *callback_first_arg);
```

Purpose

Sets the parameters for a Memory Block handle.

Parameter

handleMemory Block handle
src_blk_ptr	...	Pointer to first Memory Block to transfer.
src_blk_size	...	Specify the Block size.
callback_func	...	Callback function setting.
SDD_MEMBLK_NO_FUNC...		Do not register the callback function.
SDD_MEMBLK_SYNC_FUNC...		Use the callback function for synchronizing the internal library. Simply specifying this establishes a synchronous mode.
Other	...	An address specified here is registered as a callback function.
cb_first_arg	...	Registers the first parameter of the callback function. Specify NULL if not needed.

Return

SDE_ERR_NO_INIT	...The Sound Library has not been initialized.
SDE_ERR_HANDLE_NULL...	The handle was NULL.
SDE_ERR_HANDLE_ILLEGAL_VALUE...	An illegal address was specified for the handle.

Remarks

Reference

sdMemBlkSetTransferMode

Sets transfer mode for a Memory Block handle.

Prototype

```
#include <sg_sd.h>
EXTERN SDE_ERR sdMemBlkSetTransferMode( SDE_MEMBLK_TRANSFER_MODE transfer_mode);
```

Purpose

Sets the transfer method for Memory Blocks

Parameter

transfer_mode	...	Transfer Mode
---------------	-----	---------------

Return

SDE_ERR_NO_INIT	...	The Sound Library has not been initialized.
-----------------	-----	---

Remarks

Reference

```
SDE_MEMBLK_TRANSFER_MODE
```

18. Dreamcast Middleware Guide

18.1 Purpose

This library is intended to provide simple play of motion pictures and sound. Files in the following formats can be played back as data stream on the GD-ROM.

- 1) MPEG Sofdec data file
This data is created by compressing video using MPEG/Video and audio by Sofdec/Audio and multiplexing them.
- 2) WAVE data file
This is the uncompressed PCM sound data in WAVE format and Dreamcast ADPCM-compressed audio data.
- 3) TrueMotion data file
This data is created by compressing video using TrueMotion and audio by DK3/DK4 and multiplexing them.
- 4) MPEG/Audio data file
This is MPEG/Audio Layer II-compressed audio data.

18.1.1 Modular Structure

Fig. below shows the modular structure of the Middleware.

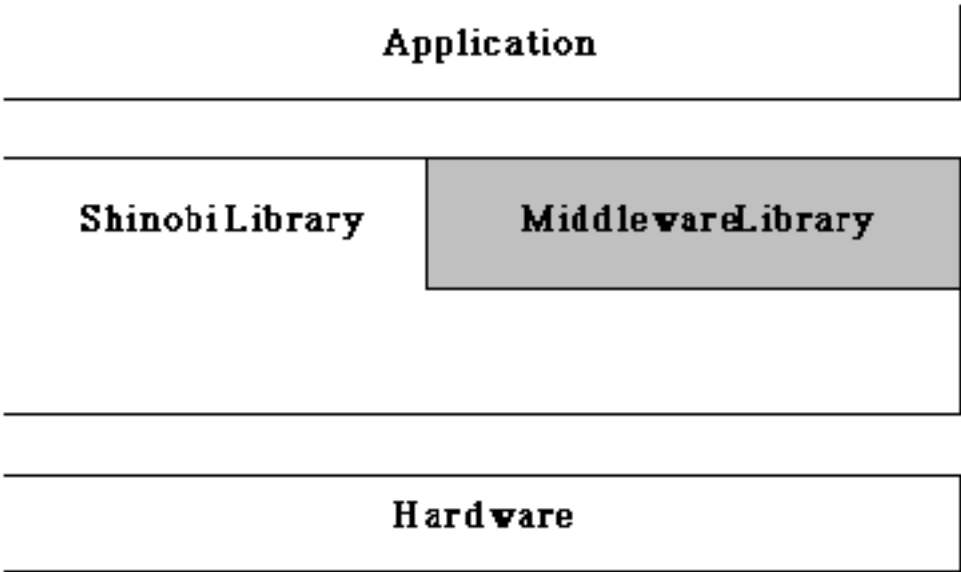


Figure 1.1 Modular Structure

18.2 Middleware Library Internal Structure

The Middleware Library consists of the modules shown in Table 2-1. Applications can easily play motion pictures and sounds using the Middleware API.

Fig. below shows the internal structure of the Middleware Library.

Table 1.1 Middleware Library Internal Modules

Module Name	Description
Module Manager	Assigns decoder CPU time
Stream Controller	Loads interleaved video and sound data
Stream Spriter	Separates interleaved video and sound data, and sends to decoder
Video Decoder	Decompresses video data and sends to video renderer
Audio Decoder	Decompresses sound data and sends to audio renderer
Video Renderer	Outputs decoded video data
Audio Renderer	Outputs decoded audio data

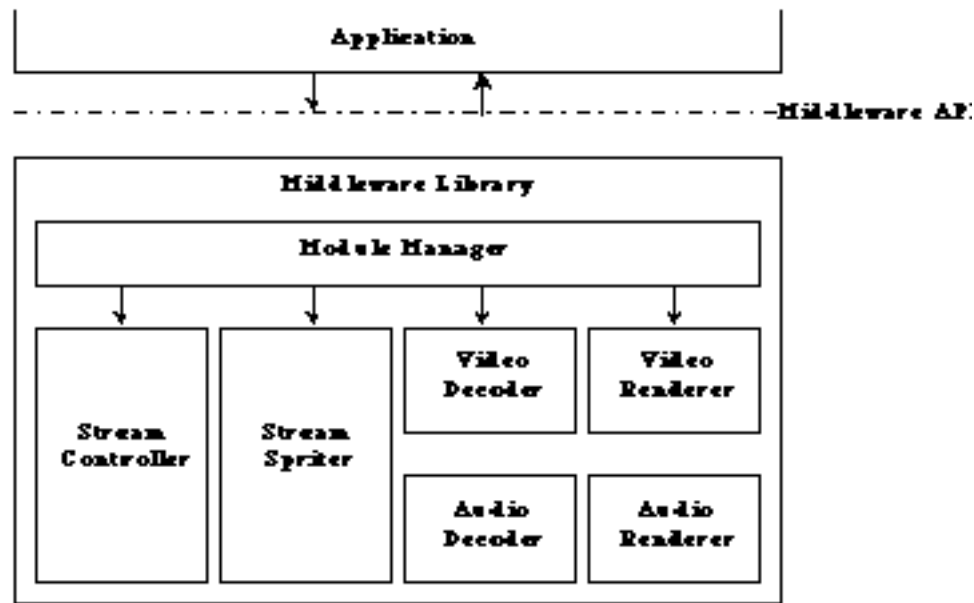


Figure 1.1 *Middleware Library Internal Module*

18.3 Middleware Usage Methods

18.3.1 Initializing the middleware library

To initialize the middleware library, the initialization function for each codec must be called.

<In the case of MPEG Sofdec>

```
mwPlyPreInitSofdec();  
sbInitSystem(NJD_RESOLUTION_640x480_NTSCI, NJD_FRAMEBUFFER_MODE_ARGB8888, 1);  
mwPlyInitSofdec(&iprm);
```

<In the case of WAVE>

```
sbInitSystem(NJD_RESOLUTION_640x480_NTSCI, NJD_FRAMEBUFFER_MODE_ARGB8888, 1);  
mwPlyInitWav(MWD_PLY_SVR_VSYNC);
```

<In the case of TrueMotion>

```
sbInitSystem(NJD_RESOLUTION_640x480_NTSCI, NJD_FRAMEBUFFER_MODE_ARGB8888, 1);  
mwPlyInitTM();
```

<In the case of MPEG/Audio>

```
sbInitSystem(NJD_RESOLUTION_640x480_NTSCI, NJD_FRAMEBUFFER_MODE_ARGB8888, 1);  
mwPlyInitMpa(MWD_PLY_SVR_VSYNC);
```

18.3.2 Middleware playback handle

The audio and video data as compressed by each codec can be played back with the use of the middleware playback handle. First, you must generate the middleware playback handle for each codec. This can be done by using the appropriate API for the codec.

```
MWPLY      plysfd, plywav, plytm, plympa;
MWS_PLY_CPRM_SFD      cprm_sfd;
MWS_PLY_CPRM_TM      cprm_tm;
MWS_PLY_CPRM      cprm_wav, cprm_mpa;

plysfd = mwPlyCrateSofdec(&cprm_sfd);

/* Handle creation for MPEG Sofdec data playback */
plywav = mwPlyCrateWav(&cprm_wav);

/* Handle creation handle for WAVE data playback */
plytm = mwplyCreateTM(&cprm_tm);

/* Handle creation handle for TrueMotion data playback */
Plympa = mwPlyCreateMpa(&cprm_mpa);

/* Handle creation handle for MPEG/Audio data playback */
```

While different API is required for different handle, playback instruction can be done using the same API.

```
mwPlyStartFname(plysfd, îSAMPLE.SFDî);      /* Start playback of MPEG Sofdec data
*/
mwPlyStartFname(plywav, îSAMPLE.WAVî);      /* Start playback of WAVE data */
mwPlyStartFname(plytm, îSAMPLE.AVIî);      /* Start playback of TrueMotion data */
mwPlyStartFname(plypa, îSAMPLE.MP2î);      /* Start playback of MPEG/Audio data */
```

The `mwPlyStart` and `mwPlyStop` functions are used to control playback start and stop of the video and audio data.

18.3.3 How to Use the Middleware Library

The internal status of the middleware library is updated by calling the main routine server function (`mwPlyExecServer`) and then the `njWaitVSync` function. Also, the function `mwPlyStartFrame` must be called directly after the function `njwaitSync` to notify the start of the game frame to the middleware library.

```
while (1) {  
    njWaitVSync();  
    mwPlyStartFrame();           /* Game frame start notice */  
    :  
    (Game progressing)  
    :  
    mwPlyExecServer();          /* Decode processing, etc. */  
}
```

18.3.4 Operational states of the middleware playback handle

The operational states of a middleware playback handle is listed below. Immediately after a middleware playback handle is created, it is in the STOP state. When playback starts, the state changes in this sequence: PREP -> PLAYING -> PLAYEND.

Table 3-1 Handle Status

Status	Description
STOP	Play is stopped
PREP	Preparing to play
PLAYING	Playing in progress
PLAYEND	Finished playing
ERROR	An error has occurred

The changes of state are shown in Fig. below.

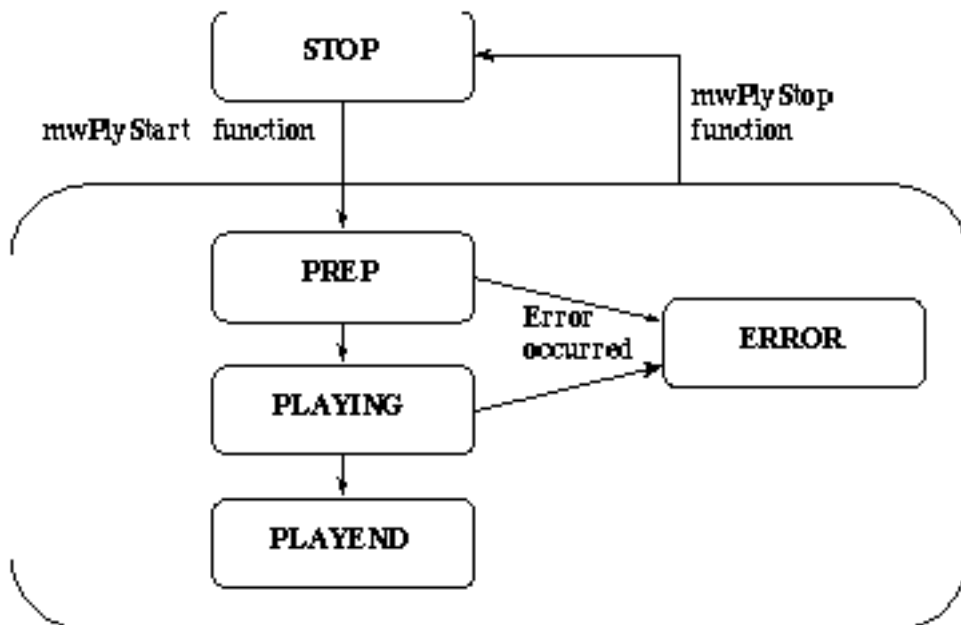


Figure 1.2 State Diagram

18.3.5 Playback of MPEG Sofdec

The following is a sample program for playing back MPEG Sofdec data.

<Set up interrupt stack, etc.>

```
/* Display mode */
#define SYS_MODE          NJD_RESOLUTION_640x480_NTSCI
#define SYS_FRAME        NJD_FRAMEBUFFER_MODE_ARGB8888
#define SYS_COUNT        1

/* Vertex buffer size (if a value is negative, then latency mode is 2V) */
#define SFD_VB_OP         -500000
#define SFD_VB_OM         0
#define SFD_VB_TP         20000
#define SFD_VB_TM         0
#define SFD_VB_PT         0

/* Application main function */
void main(void)
{
    MWPLY                  ply;                /* Middleware playback handle */
    MWE_PLY_STAT           stat;               /* Handle status */
    MWS_PLY_CPRM_SFD       cprm;              /* Handle creation parameter structure */
    /*

    mwPlyPreInitSofdec();                      /* Set up interrupt stack, etc. */
    sbInitSystem(SYS_MODE, SYS_FRAME, SYS_COUNT)
```

```
njInitVertexBuffer(VB_OP, VB_OM, VB_TP, VB_TM, VB_PT);
/*
 * Screen and sound initialization
 */
memset(&iprm, 0, sizeof(iprm));           /* Zero out reserved members */
iprm.mode = SYS_MODE;
iprm.frame = SYS_FRAME;
iprm.count = SYS_COUNT;
iprm.latency = MWD_PLY_LATENCY(VB_OP, VB_OM, VB_TP, VB_TM, VB_PT);
mwPlyInitSofdec(&iprm);                   /* Library initialization */
/* Zero out reserved members */
cprm.ftype = MWD_PLY_FTYPE_SFD;          /* Video + audio */
cprm.dtype = MWD_PLY_DTYPE_AUTO;         /* Give higher priority to images */
cprm.max_bps = 450*1024*8;               /* Bite rate: 450 Kbyte/sec */
cprm.max_width = 320;                   /* Image size: 320x480 */
cprm.max_height = 480;
cprm.nfrm_pool_wk = 3;                  /* Number of frame buffers */
/* Calculation of work area size */
cprm.wksize = mwPlyCalcWorkSofdec(cprm.ftype, cprm.max_bps, cprm.max_width,
cprm.max_height, cprm.nfrm_pool_wk);
cprm.work = syMalloc(cprm.wksize);
ply = mwPlyCreateSofdec(&cprm);           /* Handle creation */
mwPlyStartFname(ply, iSAMPLE.SFDi);      /* Start playing */
for (;;) {
    njWaitVSync();                      /* Wait for V-Sync */
    mwPlyStartFrame();                  /* Notify start of game frame */
    stat = mwPlyGetStat(ply);           /* Get handle status */
    if ( stat == MWE_PLY_STAT_PLAYEND ) {
        break;
    }
    mwPlyExecServer();                  /* Run middleware library */
}
mwPlyStop(ply);                         /* Stop playback */
mwPlyDestroy(ply);                     /* Destroy handle */
syFree(cprm.work);
mwPlyFinishSofdec();                   /* Library initialization */
}
```

18.3.6 Playback of WAVE

The following is a sample program for playing back WAVE file.

<Set up interrupt stack, etc.>

```
/* Application main function */
void main(void)
{
    MWPLY                ply;                /* Middleware playback handle */
    MWE_PLY_STAT          stat;              /* Handle status */
    MWS_PLY_CPRM          cprm;              /* Handle creation parameter structure */
    /*
    * Screen and sound initialization
    */
    mwPlyInitWav(MWD_PLY_SVR_VSYNC);          /* Library initialization */
    cprm.size = MWD_PLY_CALC_AWORK(MWD_PLY_MIN_AWORK);
    cprm.buf = syMalloc(cprm.size);
    cprm.libwork = NULL;
    ply = mwPlyCreateWav(&cprm);              /* Handle creation */
    mwPlyStartFname(ply, iSAMPLE.WAVi);      /* Start playing */
    for (;;) {
        njWaitVSync();                      /* Wait for V-Sync */
        mwPlyStartFrame();                  /* Notify start of game frame */
        stat = mwPlyGetStat(ply);           /* Get handle status */
        if ( stat == MWE_PLY_STAT_PLAYEND ) {
            break;
        }
        mwPlyExecServer();                  /* Run middleware library */
    }
    mwPlyStop(ply);                        /* Stop playback */
    mwPlyDestroy(ply);                     /* Destroy handle */
    syFree(cprm.buf);
    mwPlyFinishWav();                      /* Library initialization */
}
```

18.3.7 Playback of TrueMotion

The following is a sample program for playing back TrueMotion.

<Set up interrupt stack, etc.>

```
/* Application main function */
void main(void)
{
    MWPLY                ply;                /* Middleware playback handle */
    MWE_PLY_STAT         stat;              /* Handle status */

    /*
     * Screen and sound initialization
     */
    mwPlyInitTM();                          /* Library initialization */
    ply = mwPlyCreateTM(NULL);              /* Handle creation */
    mwPlyStartFname(ply, "SAMPLE.AVI");     /* Start playback */
    for (;;) {
        njWaitVSync();                     /* Wait for V-Sync */
        mwPlyStartFrame();                 /* Notify start of game frame */
        stat = mwPlyGetStat(ply);          /* Get handle status */
        if ( stat == MWE_PLY_STAT_PLAYEND ) {
            break;
        }
        mwPlyExecServer();                 /* Run middleware library */
    }
    mwPlyStop(ply);                         /* Stop playback */
    mwPlyDestroy(ply);                     /* Destroy handle */
    mwPlyFinishTM();                       /* Library initialization */
}
```

18.3.8 Playback of MPEG/Audio

The following is a sample program for playing back MPEG/Audio.

<Set up interrupt stack, etc.>

```
#define NUM_HNDL          (1)                                /* Number of handles to create */

/* Application main function */
void main(void)
{
    MWPLY                  ply;                                /* Middleware playback handle */
    MWE_PLY_STAT           stat;                                /* Handle status */
    MWS_PLY_CPRM           cprm;                                /* Handle creation parameter structure */
    /*
     * Screen and sound initialization
     */
    mwPlayInitMpa(MWD_PLY_SVR_VSYNC);                          /* Library initialization */
    cprm.size = MWD_PLY_CALC_AWORK(MWD_PLY_MIN_AWORK);
    cprm.buf = syMalloc(cprm.size);
    cprm.libwork = syMalloc(MWD_MPA_CALC_WORK(NUM_HNDL));
    ply = mwPlayCreateMpa(&cprm);                                /* Handle creation */
    mwPlayStartFname(ply, iSAMPLE.WAVi);                        /* Start playback */
    for (;;) {
        njWaitVSync();                                          /* Wait for V-Sync */
        mwPlayStartFrame();                                     /* Notify start of game frame */
        stat = mwPlayGetStat(ply);                              /* Get handle status */
        if ( stat == MWE_PLY_STAT_PLAYEND ) {
            break;
        }
        mwPlayExecServer();                                     /* Run middleware library */
    }
    mwPlayStop(ply);                                            /* Stop playback */
    mwPlayDestroy(ply);                                         /* Destroy handle */
    syFree(cprm.libwork);
    syFree(cprm.buf);
    mwPlayFinishMpa();                                          /* Library initialization */
}
```


18.4 Data Specifications

Table below shows the Library data.

Table 18.5 Data List

Data Name	Function	No.
Constants		
MWE_PLY_STAT_~	Handle status.	1.1
MWE_PLY_FTYPE~	Type of file to play.	1.2
MWE_PLY_DTYPE~	Display type of video.	1.3
Data Types		
MWPLY	Middleware playback handle.	2.1
MWS_PLY_INIT_SFD	Initialization parameter structure. (for MPEG Sofdec)	2.2
MWS_PLY_CPRM_SFD	Handle creation parameter structure. (for MPEG Sofdec)	2.3
MWS_PLY_CPRM_TM	Handle creation parameter structure. (for TrueMotion)	2.4
MWS_PLY_CPRM	Handle creation parameter structure. (for WAVE, MPEG/ Audio)	2.5

18.5.1 Constants

Title	Data Name	Data	No
Data	MWE_PLY_STAT_~	Handle status.	1.1

The following constants indicate handle status.

Constant Name	Description
MWE_PLY_STAT_STOP	Stopping
MWE_PLY_STAT_PREP	Preparing
MWE_PLY_STAT_PLAYING	Playing
MWE_PLY_STAT_PLAYEND	Finishing playing
MWE_PLY_STAT_ERROR	Error state

Dreamcast Shinobi Library Specification

Title	Data Name	Data	No
Data	MWE_PLY_FTYPE~	Type of file to play.	1.2

The following constants indicate handle status.

Constant Name	Description
MWE_PLY_FTYPE_SFD	MPEG Sofdec (Sound and video)
MWE_PLY_FTYPE_MPV	MPEG/Video (Video only)

Title	Data Name	Data	No
Data	MWE_PLY_DTYPE~	Display type of video.	1.3

The following constants indicate the state of the handle.

Constant Name	Description
MWE_PLY_DTYPE_AUTO	Display size of screen is automatically decided. Use this mode to give picture quality priority. UV coordinates of texture are adjusted. so the image remains in sharp focus even when a binary file is used. The functions mwPlySetDispPos and mwPlySetDispSize are not used when using this mode.
MWE_PLY_DTYPE_FULL	Display is adjusted to fill the full screen.

18.5.2 Data Types

Title	Data Name	Data	No
Data	MWPLY	Middleware playback handle.	2.1

Handle to control playing of motion pictures and sound.

Title	Data Name	Data	No
Data	MWS_PLY_INIT_SFD	Initialization parameter structure.	2.2

Parameter structure for setting the initialization function for MPEG Sofdec. In Ninja, set the same value as the actual set display parameter. Members are as follow.

Member	Type	Description
mode	Sint32	Screen mode
frame	Sint32	Frame buffer color mode
count	Sint32	Frame count number
latency	Sint32	Display latency (2v or 3V)

Title	Data Name	Data	No
Data	MWS_PLY_CPRM_SFD	Handle creation parameter structure.	2.3

Dreamcast Shinobi Library Specification

Parameter structure to set when creating a Middleware playback handle for MPEG Sofdec. The members are as follows.

Member	Type	Description
ftype	Sint32	Type of file to play Select from MWE_PLY_FTYPE_~.
max_bps	Sint32	Maximum bitstream amount (unit: bit/sec) It is also possible to set an amount lower than actual use. If the program freezes during playback, raise this value.
max_width	Sint32	Maximum width of playback screen image size (unit: pixel)
max_height	Sint32	Maximum height of playback screen image size (unit: pixel)
nfrm_pool_wk	Sint32	Number of frame pools in the system area (usually: 3) If frames are dropped due to variations in the processing load, raise this number.
work	Sint8*	Pointer to the buffer for use
wksize	Sint32	Secured buffer size (unit: bytes)
dtype	Sint32	Display type of video Can be selected from MWE_PLY_DTYPE_~.

Title	Data Name	Data	No
Data	MWS_PLY_CPRM_TM	Handle creation parameter structure.	2.4

The parameter structure that is set when creating a handle for playing back middleware when TrueMotion is played back. Because it is not currently used, there are no setting items. Set NULL in the argument of the mwPlyCreateTM function.

Title	Data Name	Data	No
Data	MWS_PLY_CPRM	Handle creation parameter structure.	2.5

The parameter structure that is set when creating the middleware playback handle for audio CODEC (WAVE, MPEG / Audio). Members are as follow.

Member	Type	Description
buf	UInt8*	Pointer to the buffer for use Secure the buffer size calculated by the MWD_PLY_CALC_AWORK macro and set it in this member.
size	Sint32	Buffer size to use (unit: bytes) Set the buffer size calculated by the MWD_PLY_CALC_AWORK macro. MWD_PLY_MIN_AWORK = 48 is the minimum value to set in the MWD_PLY_CALC_AWORK macro and is the required number of sectors for playing back audio smoothly.
libwork	Sint32*	Work area for the decoder This member is not used for WAVE playback, so it should be set to NULL. Secure the work size calculated by the MWD_MPA_CALC_WORK macro for MPEG/Audio playback, and set it in this member. Specify the number of handles to create in the MWD_MPA_CALC_WORK argument.

18.6 Function Specifications

The library functions are listed in Table below.

Table 18.7 Function List

Function Name	Function	No.
Functions for MPEG Sofdec		
mwPlyPreInitSofdec	Set up system initialization.	1.1
mwPlyInitSofdec	Initialize the MPEG Sofdec library.	1.2
mwPlyFinishSofdec	Finish MPEG Sofdec library processing.	1.3
mwPlyCreateSofdec	Create a handle for MPEG Sofdec.	1.4
mwPlyCalcWorkSofdec	Work area size calculation.	1.5
mwPlySetDispPos	Set up the display position.	1.6
mwPlySetDispSize	Set up the display size.	1.7
mwPlySetBright	Set up the brightness.	1.8
mwPlyGetBright	Get the brightness.	1.9
mwPlySetBrightOfst	Set up the brightness offset.	1.10
mwPlyGetBrightOfst	Set up the brightness offset.	1.11
mwPlySetDispZ	Set up the display screen depth value.	1.12
mwPlyGetDispZ	Get the display screen depth value.	1.13
mwPlySetDispMode	Set up the display mode.	1.14
mwPlySetFastHalfpel	Set up the fast half pel process.	1.15
mwPlySetAudioSw	Set up the audio output switch.	1.16
mwPlySetVideoSw	Set up the video display switch.	1.17
mwPlyGetNumDropFrm	Get the number of dropped frames.	1.18
Functions for WAVE		
mwPlyInitWav	Initialize the WAVE playback library.	2.1
mwPlyFinishWav	Finish WAVE playback library processing.	2.2
mwPlyCreateWav	Create a handle for WAVE playback.	2.3
Functions for TrueMotion		
mwPlyInitTM	Initialize the TrueMotion library.	3.1

	mwPlyFinishTM	Finish TrueMotion library processing.	3.2
	mwPlyCreateTM	Create a handle for TrueMotion.	3.3
Functions for MPEG/Audio			
	mwPlyInitMpa	Initialize the MPEG/Audio library.	4.1
	mwPlyFinishMpa	Finish MPEG/Audio library processing.	4.2
	mwPlyCreateMpa	Create a handle for MPEG/Audio.	4.3

Table 18.8 Function List

Function Name	Function	No.
Global functions		
mwPlyDestroy	Destroy a handle.	5.1
mwPlyStartFname	Start playing.	5.2
mwPlyStop	Stop playing.	5.3
mwPlyGetStat	Get the handle status.	5.4
mwPlyGetTime	Get the number of the playing sample.	5.5
mwPlyStartFrame	Notify start of game frame.	5.6
mwPlyPause	Perform pause setting.	5.7
mwPlySetOutVol	Perform volume setting.	5.8
mwPlyGetOutVol	Get the volume's value.	5.9
mwPlySetOutPan	Set up panpot.	5.10
mwPlyGetOutPan	Get the panpot value.	5.11
mwPlyExecServer	Server function.	5.12
mwPlyEntryErrFunc	Register a function to call on error.	5.13

18.8.1 Functions for MPEG Sofdec

This is the function required for playback of MPEG Sofdec.

Title	Function Name	Function	No
Function	mwPlyPreInitSofdec	Set up system initialization.	1.1

[Syntax] void mwPlyPreInitSofdec(void);
[Input] none
[Output] none
[Rtn Val] none
[Purpose] Set up interrupt stack. The interrupt stack size is 16 Kbytes.

Title	Function Name	Function	No
Function	mwPlyInitSofdec	Initialize the MPEG Sofdec library.	1.2

[Syntax] void mwPlyInitSofdec(MWS_PLY_INIT_SFD *iprm);
[Input] iprm : Initialization parameter
[Output] none
[Rtn Val] none
[Purpose] Initializes the library.

It sets up the various server functions, which fall into the following types:

- 1) Main server

Executed within the main routine when mwPlyExecServer is called.

- 2) V-Sync server

Executed at V-Sync interrupt.

- 3) WaitVsync server

Executed within njWaitVsync during the wait time for the next game frame.

The MPEG Sofdec library runs decode in this server.

[Remarks] Set the same value as the one set in the display mode of the initialization parameter in Ninja.

[Example] A usage example is shown below.

```
sbInitSystem(SYS_MODE, SYS_FRAME, SYS_COUNT);
njInitVertexBuffer(VB_OP, VB_OM, VB_TP, VB_TM, VB_PT);
iprm.mode = SYS_MODE;
iprm.frame = SYS_FRAME;
iprm.count = SYS_COUNT;
iprm.latency = MWD_PLY_LATENCY(VB_OP, VB_OM, VB_TP, VB_TM, VB_PT);
mwPlyInitSofdec(&iprm);
```

Title	Function Name	Function	No
Function	mwPlyFinishSofdec	Finish MPEG Sofdec library processing.	1.3

[Syntax] void mwPlyFinishSofdec(void);

[Input] none

[Output] none

[Rtn Val] none

[Purpose] Finishes library processing.

Title	Function Name	Function	No
Function	mwPlyCreateSofdec	Create a handle for MPEG Sofdec.	1.4

[Syntax] MWPLY mwPlyCreateSofdec(MWS_PLY_CPRM_SFD *cprm);

[Input] cprm: Handle creation parameter

[Output] none

[Rtn Val] Middleware playback handle

[Purpose] Creates a handle.

By specifying dtype of the handle creation parameter, the display region can be controlled.

<In the case of automatic adjustment>

dtype = MWD_PLY_DTYPE_AUTO: the display region is adjusted automatically as multiple of integer number.

If the size is smaller than 640 x 480, the display region will be reduced by 1 dot. In the case of 320 x 240, it will become 639 x 479. Because the UV coordinates of texture are adjusted, a high quality image with sharp focus can be gotten.

<In this case of full screen display>

dtype = MWD_PLY_DTYPE_FULL: full screen display

Dreamcast Shinobi Library Specification

This is different from the case of automatic adjustment; the display region is not reduced by 1 dot but the display will be set at 640 x 480.

[Example] A usage example is shown below.

```
cprm.ftype = MWD_PLY_FTYPE_SFD;
cprm.dtype = MWD_PLY_DTYPE_AUTO;          /* Give higher priority to images */
cprm.max_bps = 450*1024*8;                 /* Bit rate: 450 Kbyte/sec */
cprm.max_width = 320;                      /* Image size: 320x480 */
cprm.max_height = 480;
cprm.nfrm_pool_wk = 3;                     /* Number of frame buffers */
/* Calculation of work area size */
cprm.wksize = mwPlyCalcWorkSofdec(cprm.ftype, cprm.max_bps, cprm.max_width,

cprm.max_height, cprm.nfrm_pool_wk);
cprm.work = syMalloc(cprm.wksize);
ply = mwPlyCreateSofdec(&cprm);
```

Title	Function Name	Function	No
Function	mwPlyCalcWorkSofdec	Work area size calculation.	1.5

[Syntax] Sint32 mwPlyCalcWorkSofdec(Sint32 ftype, Sint32
max_bps, Sint32 max_sx, Sint32
max_sy, Sint32 nfb);

[Input] ply: Middleware playback handle

ftype : File type

MWD_PLY_FTYPE_SFD: MPEG Sofdec (audio and video)

MWD_PLY_FTYPE_MPV: MPEG/Video (only video)

max_bps : Maximum bitstream amount (unit: bit/sec)

max_sx : Maximum width of playback screen image size (unit: pixel)

max_sy : Maximum height of playback screen image size (unit: pixel)

nfb : Number of frame pools in the system area

[Output] none

[Rtn Val] Work area size (unit: bytes)

[Purpose] Calculates the work area size to use in MPEG Sofdec playback.

Title	Function Name	Function	No
Function	mwPlySetDispPos	Set up the display position.	1.6

[Syntax] `void mwPlySetDispPos(MWPLY ply, float lx, float ly);`
[Input] ply: Middleware playback handle
lx : X-coordinate
ly : Y-coordinate
[Output] none
[Rtn Val] none
[Purpose] Set up the display position.

Title	Function Name	Function	No
Function	mwPlySetDispSize	Set up the display size.	1.7

[Syntax] `void mwPlySetDispSize(MWPLY ply, float sx, float sy);`
[Input] ply : Middleware playback handle
sx : X-coordinate
sy : Y-coordinate
[Output] none
[Rtn Val] none
[Purpose] Set up the display size.

Title	Function Name	Function	No
Function	mwPlySetBright	Set up the brightness.	1.8

[Syntax] `void mwPlySetBright(MWPLY ply, Sint32 val);`
[Input] ply : Middleware playback handle
val : Brightness (0~255)
[Output] none
[Rtn Val] none
[Purpose] Set up the brightness. By default, it is set to 224.

Dreamcast video output is set as a default to 224 because at 255, the brightness becomes 110IRE. It may be necessary to adjust the brightness value depending on the material.

Dreamcast Shinobi Library Specification

Title	Function Name	Function	No
Function	mwPlyGetBright	Get the brightness.	1.9

[Syntax] `Sint32 mwPlyGetBright(MWPLY ply);`
[Input] ply : Middleware playback handle
[Output] none
[Rtn Val] Brightness
[Purpose] Get the brightness.

Title	Function Name	Function	No
Function	mwPlySetBrightOfst	Set up the brightness offset.	1.10

[Syntax] `void mwPlySetBrightOfst(MWPLY ply, Sint32 val);`
[Input] ply : Middleware playback handle

val : Brightness offset (0~255)
[Output] none
[Rtn Val] none
[Purpose] Set the brightness offset. By default, it is set to 6.

In CG movie, black tends to break down, so by default it is set to 6. This value may have to be adjusted, depending on the material.

You can create the fade-in effect by reducing the value between 255 and 0.

Conversely, you can also create a fade-out effect by gradually increasing the value to 255.

Title	Function Name	Function	No
Function	mwPlyGetBrightOfst	Get the brightness offset.	1.11

[Syntax] `Sint32 mwPlyGetBrightOfst(MWPLY ply);`
[Input] ply : Middleware playback handle
[Output] none
[Rtn Val] Brightness offset
[Purpose] Get the brightness offset.

[Syntax]		<code>void mwPlySetDispZ(MWPLY ply, float z);</code>
[Input]	ply	: Middleware playback handle
	z	: Depth value is from 1.0 (closest to surface) to 65536.0 (deepest)
[Output]	none	
[Rtn Val]	none	
[Purpose]	Sets the display screen depth value.	

[Syntax]	<code>float mwPlyGetDispZ(MWPLY ply);</code>
[Input]	<code>ply</code> : Middleware playback handle
[Output]	none
[Rtn Val]	Depth value from 1.0 (closest to surface) to 65536.0 (deepest)
[Purpose]	Gets the depth value of the display screen being se

[Syntax]	void mwPlySetDispMode(Sint32 mode, Sint32 frame, Sint32 count, Sint32 latency);	
[Input]	mode	: Screen mode
	frame	: Frame buffer color mode
	count	: Number of frames
	latency	: Display latency
[Output]	none	
[Rtn Val]	none	
[Purpose]	Sets display mode in middleware.	

Dreamcast Shinobi Library Specification

[Note] Set the same value as the value set in the actual display mode.

This function is not necessary when set in MWS_PLY_INIT_SFD at the time of initialization.

When the display mode is changed, it is necessary to reset the display mode with this function.

[Remarks] Notes regarding frame count number:

- 1) The display update V-Sync number is 2*count when there is interlacing.
- 2) For example, if the frame count is 2 with interlace, 4V is displayed.
- 3) 1V smoothly displays optional frame rate animation, such as 24fps.
- 4) 2V display gives more CPU time to MPEG Softc and the danger of dropped frames is reduced, but the frame rate for smooth playback is limited.
(NTSC and VGA: 29.97 fps, 30 fps; PAL: 25fps).
- 5) Displays of 3V and above are not recommended for playing back animation.
- 6) For 320x240 display, a 500Kbyte texture area is used when latency is 2V; a 750Kbyte texture area is used when latency is 3V.

Title	Function Name	Function	No
Function	mwPlySetFastHalfpel	Set up the fast half pel process.	1.15

[Syntax] `void mwPlySetFastHalfpel(MWPLY ply, Sint32 sw);`

[Input] ply : Middleware playback handle
sw : Fast half pel switch (0: fast processing off, 1: fast processing on)

[Output] none

[Rtn Val] none

[Purpose] Sets fast half pel processing.

[Remarks] When fast half pel processing is set, the burden on the CPU is reduced, but the picture quality is also diminished somewhat.

Title	Function Name	Function	No
Function	mwPlySetAudioSw	Set up the audio output switch.	1.16

[Syntax] `void mwPlySetAudioSw(MWPLY ply, Sint32 sw);`

[Input] ply : Middleware playback handle
sw : Audio output switch (0: sound output off, 1: audio output on)

[Output] none

[Rtn Val] none

[Purpose] Sets the audio output switch.

[Remarks] As a default, audio is output.

With this function, it is possible to playback only video even for animated files with sound.

Title	Function Name	Function	No
Function	mwPlySetVideoSw	Set up the video display switch.	1.17

[Syntax] `void mwPlySetVideoSw(MWPLY ply, Sint32 sw);`

[Input] ply : Middleware playback handle

sw : Video display switch (0: video display off, 1: video display on)

[Output] none

[Rtn Val] none

[Purpose] Sets the switch for video display

Title	Function Name	Function	No
Function	mwPlyGetNumDropFrm	Get the number of dropped frames.	1.18

[Syntax] `Sint32 mwPlyGetNumDropFrm(MWPLY ply);`

[Input] ply : Middleware playback handle

[Output] none

[Rtn Val] Number of frames

[Purpose] Gets the number of dropped frames.

When the display mode is interlace, this function is only enabled when animation is 29.97 and 30 fps.

18.8.2 Functions for WAVE

This are the functions required for playback of WAVE.

Title	Function Name	Function	No
Function	mwPlyInitWav	Initialize the WAVE playback library.	2.1

[Syntax] `void mwPlyInitWav(Sint32 mode);`

[Input] `mode` : Server function calling mode

MWD_PLY_AT_EXEC_VSYNC :

Execute the server function without V-Sync interrupt.

MWD_PLY_MN_EXEC_VSYNC :

The application calls the server function and performs server processing.

[Output] none

[Rtn Val] none

[Purpose] Initializes the library.

Title	Function Name	Function	No
Function	mwPlyFinishWav	Finish WAVE playback library processing.	2.2

[Syntax] `void mwPlyFinishWav(void);`

[Input] none

[Output] none

[Rtn Val] none

[Purpose] Finishes library processing.

Title	Function Name	Function	No
Function	mwPlyCreateWav	Create a handle for WAVE playback.	2.3

[Syntax] `MWPLY mwPlyCreateWav(MWS_PLY_CPRM *cprm);`

[Input] `cprm` : Parameter for handle creation

[Output] none

[Rtn Val] Middleware playback handle

[Purpose] Creates a handle.

Set the value calculated by the MWD_PLY_CALC_AWORK macro in cprm.size.

Set the sector unit value in the MWD_PLY_CALC_AWORK macro argument. MWD_PLY_MIN_AWORK = 48 is the minimum value to set in the MWD_PLY_CALC_AWORK macro and is the required number of sectors for playing back audio smoothly.

Set by securing an area sufficient for cprm.size in cprm.buf.

You don't have to set any other work area except the buffer area for WAVE playback, so set cprm.libwork to NULL.

[Example] A usage example is shown below.

```
cprm.size = MWD_PLY_CALC_AWORK(MWD_PLY_MIN_AWORK);
cprm.buf = syMalloc(cprm.size);
cprm.libwork = NULL;
ply = mwPlyCreateWav(&cprm);
```

18.8.3 Functions for TrueMotion

These functions are required for TrueMotion playback.

Title	Function Name	Function	No
Function	mwPlyInitTM	Initialize the TrueMotion library.	3.1

[Syntax] `void mwPlyInitTM(void);`

[Input] none

[Output] none

[Rtn Val] none

[Purpose] Initialize the library.

Title	Function Name	Function	No
Function	mwPlyFinishTM	Finish TrueMotion library processing.	3.2

[Syntax] `void mwPlyFinishTM(void);`

[Input] none

[Output] none

[Rtn Val] none

[Purpose] Finishes library processing.

Title	Function Name	Function	No
Function	mwPlyCreateTM	Create a handle for TrueMotion.	3.3

[Syntax] `MWPLY mwPlyCreateTM(MWS_PLY_CPRM_TM *cprm) ;`
[Input] `cprm` : Handle creation parameter
[Output] none
[Rtn Val] Middleware playback handle
[Purpose] Creates a handle.

18.8.4 Functions for MPEG/Audio

These functions are required for MPEG/ Audio playback.

Title	Function Name	Function	No
Function	mwPlyInitMpa	Initialize the MPEG/Audio library.	4.1

[Syntax] `void mwPlyInitWav(Sint32 mode) ;`
[Input] `mode` : Server function call mode
MWD_PLY_SVR_VSYNC:
Runs server function in V-Sync interrupt.
MWD_PLY_SVR_MAIN:
Runs server processes by calling the server function on the application side.
[Output] none
[Rtn Val] none
[Purpose] Initialize the library.

Title	Function Name	Function	No
Function	mwPlyFinishMpa	Finish MPEG/Audio library processing.	4.2

[Syntax] `void mwPlyFinishMpa(void) ;`
[Input] none
[Output] none
[Rtn Val] none
[Purpose] Finishes library processing.

Title	Function Name	Function	No
Function	mwPlyCreateMpa	Create a handle for MPEG/Audio.	4.3

[Syntax] `MWPLY mwPlyCreateMpa(MWS_PLY_CPRM *cprm) ;`

[Input]	cprm :	Handle creation parameter
[Output]	none	
[Rtn Val]	Middleware creation handle	
[Purpose]	Creates a handle.	

Set the value calculated by the MWD_PLY_CALC_AWORK macro in cprm.size.

Set the sector unit value in the MWD_PLY_CALC_AWORK macro argument. MWD_PLY_MIN_AWORK = 48 is the minimum value to set in the MWD_PLY_CALC_AWORK macro and is the required number of sectors for playing back audio smoothly.

Set by securing an area sufficient for cprm.size in cprm.buf.

Set the value calculated by the MWD_PLY_CALC_WORK in cprm.libwork.

Specify the number of handles to create in the MWD_MPA_CALC_WORK macro argument.

[Example] A usage example is shown below.

```
cprm.size = MWD_PLY_CALC_AWORK(MWD_PLY_MIN_AWORK);  
cprm.buf = syMalloc(cprm.size);  
cprm.libwork = syMalloc(MWD_MPA_CALC_WORK(1));  
ply = mwPlyCreateMpa(&cprm);
```

18.8.5 Global functions

These are functions which are not tied to any codec type.

Title	Function Name	Function	No
Function	mwPlyDestroy	Destroy a handle.	5.1

[Syntax]	void mwPlyDestroy(MWPLY ply);
[Input]	ply : Middleware playback handle
[Output]	none
[Rtn Val]	none
[Purpose]	Destroys the handle.

Title	Function Name	Function	No
Function	mwPlyStartFname	Start playing.	5.2

Dreamcast Shinobi Library Specification

[Syntax] `void mwPlyStartFname(MWPLY ply, Sint8 *fname);`
[Input] ply : Middleware playback handle
fname : Name of moving picture and audio file
[Output] none
[Rtn Val] none
[Purpose] Starts playing motion pictures and sound.

Title	Function Name	Function	No
Function	mwPlyStop	Stop playing.	5.3

[Syntax] `void mwPlyStop(MWPLY ply);`
[Input] ply : Middleware playback handle
[Output] none
[Rtn Val] none
[Purpose] Stops playing motion pictures and sound.

Title	Function Name	Function	No
Function	mwPlyGetStat	Get the handle status.	5.4

[Syntax] `MWE_PLY_STAT mwPlyGetStat(MWPLY ply);`
[Input] ply : Middleware playback handle
[Output] none
[Rtn Val] Internal handle status
[Purpose] Gets the internal status of the Middleware playback handle.

Constant Name	Description
MWE_PLY_STAT_STOP	Stopping
MWE_PLY_STAT_PREP	Preparing
MWE_PLY_STAT_PLAYING	Playing
MWE_PLY_STAT_PLAYEND	Finished Playing
MWE_PLY_STAT_ERROR	Error

Title	Function Name	Function	No
Function	mwPlyGetTime	Get the number of the playing sample.	5.5

[Syntax] `void mwPlyGetTime(MWPLY ply, Sint32 *ncount, Sint32 *tscale);`

[Input] ply : Middleware playback handle

[Output] ncount : Playing sample number(time)

tscale : Sampling frequency (unit of time) (Hz)

[Rtn Val] none

[Purpose] Gets the playing time according to the number of the playing sample.

[Remarks] (a) Actual time (rtime) can be calculated by: `rtime = ncount / tscale;`

Title	Function Name	Function	No
Function	mwPlyStartFrame	Notify start of game frame.	5.6

[Syntax] `void mwPlyStartFrame(void);`

[Input] none

[Output] none

[Rtn Val] none

[Purpose] Notify that the game frame has started.

Title	Function Name	Function	No
Function	mwPlyPause	Perform pause setting.	5.7

[Syntax] `void mwPlyPause (MWPLY ply, Sint32 sw);`

[Input] ply : Middleware playback handle

sw : Pause switch (0: continue, 1: pause)

[Output] none

[Rtn Val] none

[Purpose] Perform switching of pause.

Selecting pause again (by specifying 1) during pause advances playback by a single frame.

Note: This function is not included in TrueMotion Ver. 1.25.

Dreamcast Shinobi Library Specification

Title	Function Name	Function	No
Function	mwPlySetOutVol	Perform volume setting.	5.8

[Syntax] `void mwPlySetOutVol (MWPLY ply, Sint32 vol);`

[Input] ply : Middleware playback handle

vol : Volume (-999~0) (unit: 1/10dB)

[Output] none

[Rtn Val] none

[Purpose] Set up the volume. By default, it is set to 0.

You can create a fade-in effect by increasing the value between -999 and 0.

On the other hand, by decreasing the value between 0 and -999 , you can create a fade-out effect.

Note: This function is not included in TrueMotion Ver. 1.25.

Title	Function Name	Function	No
Function	mwPlyGetOutVol	Get the volume's value.	5.9

[Syntax] `Sint32 mwPlyGetOutVol (MWPLY ply);`

[Input] ply : Middleware playback handle

[Output] none

[Rtn Val] Volume (unit: 1/10dB)

[Purpose] Get the value of the volume that is currently set.

Note: This function is not included in TrueMotion Ver. 1.25.

Title	Function Name	Function	No
Function	mwPlySetOutPan	Set up pan pot.	5.10

[Syntax] void mwPlySetOutPan (MWPLY ply, Sint32 chno, Sint32 pan);

[Input] ply : Middleware playback handle

chno : Channel to set

Monaural/stereo (left): MWD_CH_L(0)

Stereo (right): MWD_CH_R(1)

pan : Panpot (-15~15, -128)

MWD_PAN_LEFT = -15, MWD_PAN_RIGHT = 15

MWD_PAN_CENTER = 0, MWD_PAN_AUTO = -128

[Output] none

[Rtn Val] none

[Purpose] Set up panpot for each output channel.

By default, it is set to MWD_PAN_AUTO. In the case of monaural, the value is MWD_PAN_CENTER and for stereo the left is MWD_PAN_LEFT and the right MWD_PAN_RIGHT.

Note: This function is not included in TrueMotion Ver. 1.25.

Title	Function Name	Function	No
Function	mwPlyGetOutPan		5.11

[Syntax] Sint32 mwPlyGetOutPan (MWPLY ply, Sint32 chno);

[Input] ply : Middleware playback handle

chno : Channel to get

Monaural/stereo (left): MWD_CH_L(0)

Stereo (right): MWD_CH_R(1)

[Output] none

[Rtn Val] Panpot value of the supported channel (-15~15)

[Purpose] Get the value of the panpot which is currently set.

Note: This function is not included in TrueMotion Ver. 1.25.

Title	Function Name	Function	No
Function	mwPlyExecServer	Server function.	5.12

Dreamcast Shinobi Library Specification

[Syntax]	<code>void mwPlyExecServer(void);</code>
[Input]	none
[Output]	none
[Rtn Val]	none
[Purpose]	Refreshes the internal status of the handle.

Title	Function Name	Function	No
Function	<code>mwPlyEntryErrFunc</code>	Register a function to call on error.	5.13

[Syntax]	<code>void mwPlyEntryErrFunc (MW_PLY_ERRFN errfn, void *obj);</code>
[Input]	<code>errfn</code> : Error function
<code>obj</code>	: Error object
[Output]	
[Rtn Val]	none
[Purpose]	Registers a function to call on error.
[Remarks]	The registered error function will be called when an error occurs.

A text string is passed as the 2nd parameter of the error function. By using the text string, you can check the content of the error.

```
[Example]      A usage example is shown below.
/* User error function */
void user_error(void *user_obj, char *msg)
{
    for (;;) {
        njPrintC(NJM_LOCATION(3, 3), msg);
    }
}

void main(void)
{
    mwPlyEntryErrFunc(user_error, user_obj);
}
```


18.9 Appendix

18.9.1 Door Status Check

A sample program to check if the door is open is shown below.

<Sample program>

```
/* Function to start when a GD file system error is generated */
void UsrGdErrFunc(void *obj, Sint32 errcode)
{
    if (errcode == GDD_ERR_TRAYOPEND || errcode == GDD_ERR_UNITATTENT) {
        /* Middleware termination processing */
        mwPlyFinishSofdec();
        mwPlyFinishTM();
        mwPlyFinishMpa();
        mwPlyFinishWav();

        sbExitSystem();          /* End processing of Shinobi library */
        syBtExit();             /* Jump to simple player */
    }
}

/* Function to register user in V-sync interrupt */
void UsrVsyncFunc(void)
{
    Sint32 dstat;

    /* Door status check */
    dstat = gdFsGetDrvStat();
    if (dstat == GDD_DRVSTAT_OPEN || dstat == GDD_DRVSTAT_BUSY) {
        gdFsReqDrvStat();
    }
}

/* Application main */
void main(void)
{
    :
    :
    /* Registration of GD file system error callback function */
    gdFsEntryErrFuncAll((void *)UsrGdErrFunc, NULL);
    njSetVSyncFunction(UsrVsyncFunc);
    :
    :
}
```

With this function, a seamless transition from a game scene to a movie scene is made possible by non-synchronously playing back movies while playing back a CD stream of BGM. However, the input buffer size for the ADX handle must be increased when concurrently playing back audio. In the same way, the buffer size for the movie must also be increased. Set the maximum bitstream amount value higher than the actual stream amount value when creating the handle for MPEG Sofdec playback. As a standard, specify about 1.5x the value. This should be done because seek time is expected to be about 1 second, so it may be reduced depending on the way data is located. Also, this value should be raised if the movie breaks.

```

/* Work area size when concurrently playing back 2 streams in 44KHz stereo */
/* 2 streams means BGM and movie */
#define WKSIZ44S ADXT_CALC_WORK(2, ADXT_PLY_STM, 2, 44100)

char          wk44[WKSIZ44S];          /* Work area */
MWPLY         ply;                     /* MWPLY handle */
ADXT           adxt_bgm;               /* ADXT handle */

/* Application */
        :
        :

adxt_bgm = ADXT_Create(2, wk44, WKSIZ44S); /* ADX handle creation */
ADXT_SetReloadSct(adxt_bgm, 25);          /* For reducing seek sound */

/* MWPLY handle creation */
memset(&cprm, 0, sizeof(cprm));           /* Necessary for setting the reserve
                                          member to 0 */

cprm.ftype = MWD_PLY_FTYPE_MPV;
cprm.dtype = MWD_PLY_DTYPE_AUTO;
cprm.max_bps = 900*1024*8;               /* Usually set 1.5x of 600Kbyte/sec */
cprm.max_width = 320;
cprm.max_height = 240;
cprm.nfrm_pool_wk = 3;
cprm.wksize = mwPlyCalcWorkSofdec(cprm.ftype, cprm.max_bps, cprm.max_width,
cprm.max_height, cprm.nfrm_pool_wk);
cprm.work = syMalloc(cprm.wksize);
ply = mwPlyCreateSofdec(&cprm);

ADXT_StartFname(adxt_bgm, iBGM.ADXi);    /* Audio playback start */

if (/* event playback */
mwPlyStartFname(ply, iSCENE01.M1v1);     /* Movie playback */
        :
        .

```

19. Dreamcast Middleware Outline

19.1 Introduction

Middleware is a category of software that stands between applications and the hardware. It is supplied by the middleware vendor in the form of tools and libraries. By using these tools and libraries, the developer can easily create applications for playing compressed images and audio.

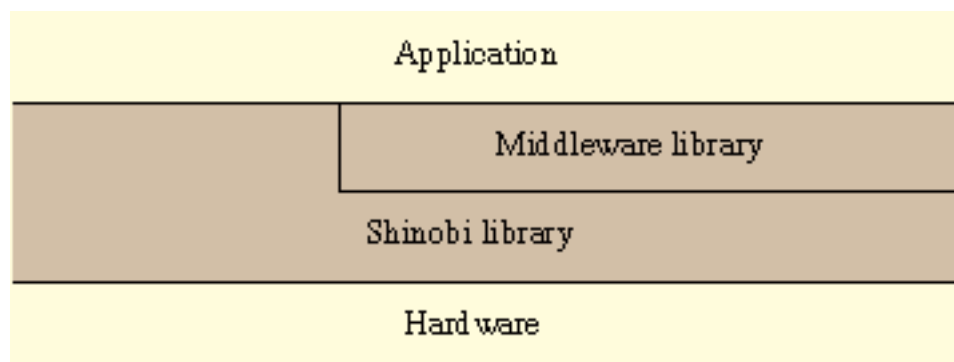


Figure 2.1 *Middleware Configuration*

Currently, the following middleware is available for Dreamcast.

1) CRI MPEG Sofdec

MPEG Sofdec can play back video encoded with MPEG 1 and audio encoded with SofdecAudio. The bit rate is 150 KByte/Sec to 600 KByte/Sec. When the bit rate is about 450 KByte/Sec or higher, there is no noticeable image degradation during playback. The exact CPU load depends on resolution and bit rate, but decoding is possible at about 50%. MPEG standard video data that follows international standards can be played back, so data created using an commercially sold MPEG encoder can also be played back.

2) CRI ADX

CRI ADX (abbreviated ADX) is characterized by the following three features.

- Simultaneous playback of multiple files from GD-ROM is possible. (multistream playback)
- Game data can be read also during streaming playback from GD-ROM.
- ADX encoded, Dreamcast ADPCM encoded, and non-encoded audio data can be played.

ADX encoded audio data (proprietary format) can be played back with CD quality. CPU load is about 0.7% for 44.1 kHz mono sound. Seamless loop playback is also possible. Also, multistream playback such as MPEG Sofdec, WAV decoder and MPEG / Audio can be played back with this middleware.

3) WAV decoder

Can play back Dreamcast ADPCM encoded and non-encoded WAV files from GD-ROM. Only single-stream playback is possible.

4) TrueMotion

TrueMotion plays back PC-compressed video and audio on Dreamcast at a bit rate of 600KByte/sec to 1.2MByte/sec. At a bit rate of about 900 byte/sec for 320x240 or 1.2 byte/sec for 640x480, the picture quality that is gotten is comparatively good. The CPU load is low at about 25% for 320x240.

5) MPEG / Audio

Plays back sound compressed by MPEG1 / AudioLayer2 format that conforms to ISO11172. The CPU load is 15% in stereo.

19.2 Middleware Library

19.2.1 Software Configuration

The middleware library configuration is shown below.

1) Middleware API

The application developer can use this API to implement various CODEC playback functions, using the same interface. Currently, MPEG Sofdec, TrueMotion, WAV decoder and MPEG / Audio support this API. ADX uses its own API for audio playback.

2) Various middleware functions

Software library for playback of video and audio sources compressed using various CODEC methods.

3) Basic middleware library

Comprises the middleware manager, audio renderer, video renderer and other middleware playback functions.

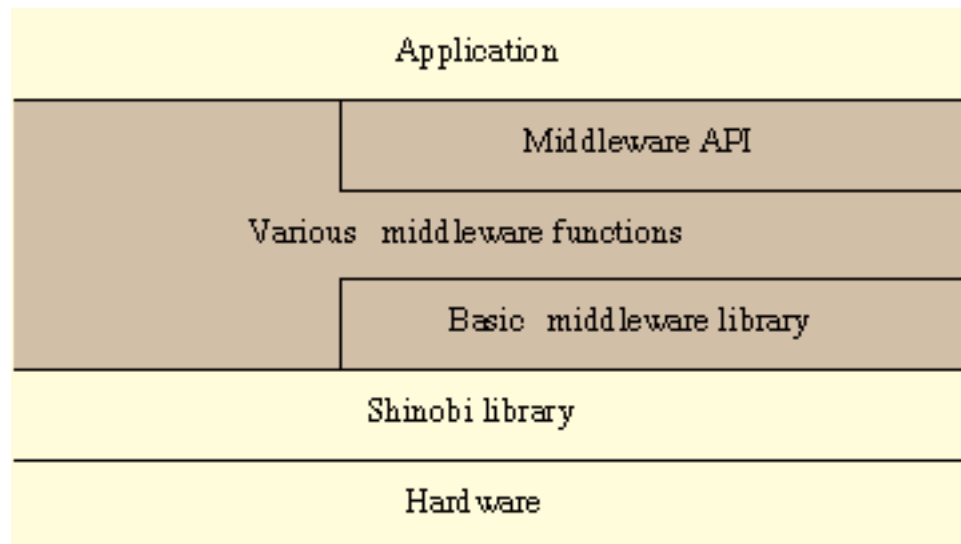


Figure 2.2 *Middleware Library Configuration*

19.2.2 Common Header Files and Library Files

The header files and library files required for middleware playback are as follows.

```

SG_MW.H      :   Middleware API header file
SG_MWPLY.H   :   Middleware playback library header file
                (included from SG_MW.H)
SG_MW.LIB    :   Basic middleware library
  
```

19.3 CRI MPEG Sofdec Outline

19.3.1 Features

MPEG Sofdec serves for playback of MPEG1 encoded video and SofdecAudio encoded audio.

19.3.2 Software Configuration

The software configuration is shown below. For information on MPEG Sofdec playback, refer to the middleware library manual.

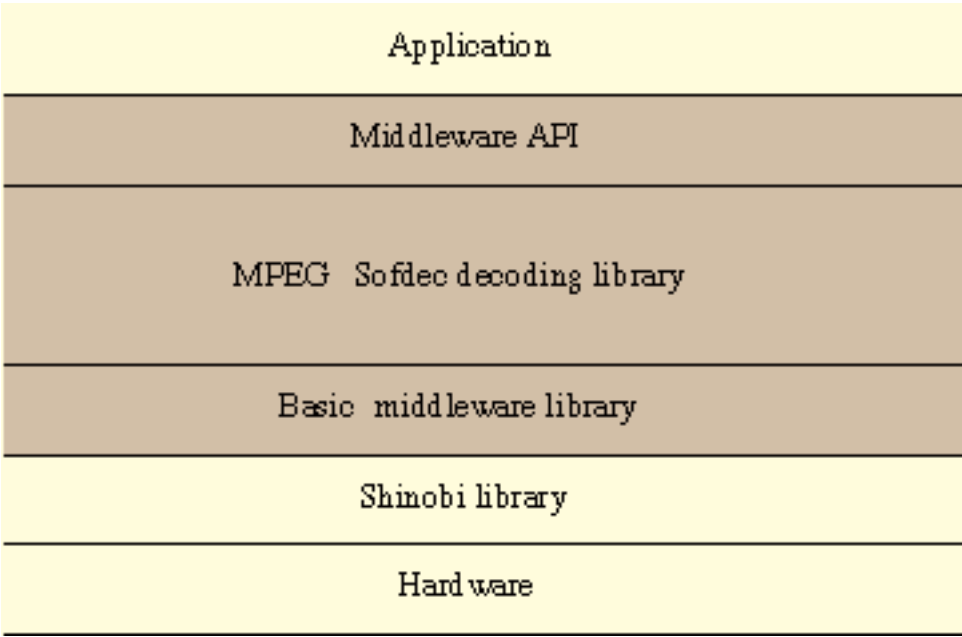


Figure 2.3 *Sofdec Software Configuration*

19.3.3 Data Creation Tools

The following tools for creating MPEG Sofdec data are available.

- | | | |
|-------------|---|---|
| SFVENCD.EXE | : | MPEG/Video encoder |
| SFAENCD.EXE | : | SofdecAudio encoder |
| SFDMUX.EXE | : | Multiplexer |
| SFDMAKE.BAT | : | Batch file for automated encoding operation |

19.3.4 Header Files and Library Files

The following header files are required. By including `SG_MW.H`, the following file are automatically included.

`SFD_MW.H` : MPEG Sofdec decoding library header

To use MPEG Sofdec, the following files must be linked.

`SOFDEC.LIB` : MPEG Sofdec decoding library

`SG_MW.LIB` : Basic middleware library

19.4 CRI ADX Outline

19.4.1 Features

The ADX features are shown below.

1) Audio playback function

ADX encoded, Dreamcast ADPCM encoded, and non-encoded audio data can be easily played back with this API. ADX encoded data (ADX data) can be played in CD quality (44.1 kHz, stereo) with only about 2% CPU load, including data transfer. Seamless loop playback is also possible.

2) Multistream playback function

Several audio and animation files can be played simultaneously from the GD-ROM.

3) ADX file system

The ADX file system allows reading of audio information as well as game data from GD-ROM.

19.4.2 Software Configuration

The software configuration is shown below. The ADX library API is used for audio playback and for reading game data. For details, refer to the ADX playback library implementation manual.

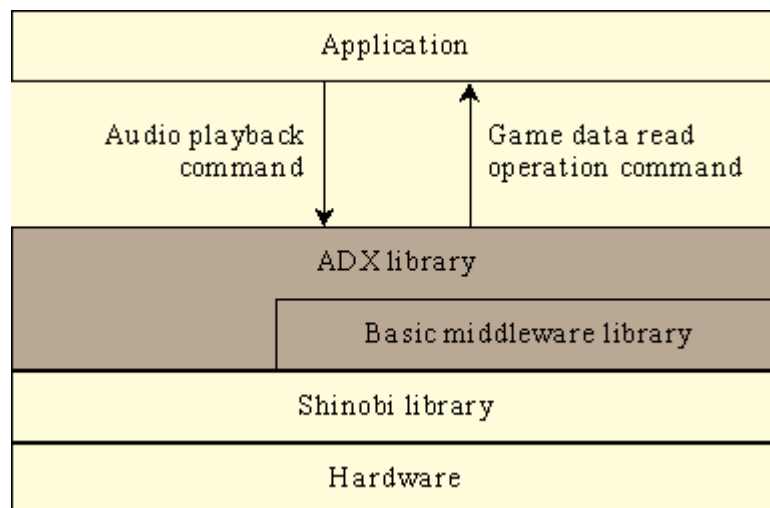


Figure 2.4 ADX Software Configuration

19.4.3 Data Creation Tools

The following tools for creating ADX data are available.

ADXENCD.EXE	:	ADX encoder
ADXCAT.EXE	:	Memory index playback linking tool
AFSLNK.EXE	:	ADX file system linking tool
DADENC.EXE	:	Dreamcast ADPCM encoder
AX.EXE	:	Automated operation tool

19.4.4 Header Files and Library Files

The following header files must be included for using the ADX library.

CRI_ADXT.H	:	ADX library header file
CRI_ADXF.H	:	ADX file system library header file

The following library files must be linked.

CRI_ADXS.LIB	:	ADX library
SG_MW.LIB	:	Basic middleware library

19.4.5 Joint Use With MPEG SOFDEC

To use ADX and MPEG Sofdec together, specify the library files to the linker in the following order, giving priority to the ADX library.

CRI_ADXS.LIB	:	ADX library
SOFDEC.LIB	:	MPEG Sofdec decoding library
SG_MW.LIB	:	Basic middleware library

19.4.6 Multistream playback with CODEC

Multistreaming can also be done for other CODEC functions (MPEG Sofdec, WAV decoder and MPEG/Audio) by linking ADX. When linking to other CODEC functions, set the ADX library to be linked first.

CRI_ADXS.LIB	:	ADX library
SOFDEC.LIB	:	MPEG Sofdec decoding library
SG_MWAV.LIB	:	WAV decoding library
MP1A_L2.LIB	:	MPEG/Audio decoding library
SG_MW.LIB	:	Basic middleware library

19.5 WAV Decoder Outline

19.5.1 Features

The WAV decoder serves for playback of non-encoded or Dreamcast ADPCM encoded WAV files.

19.5.2 Software Configuration

The software configuration is shown below. For information on WAV file playback, refer to the middleware library manual.

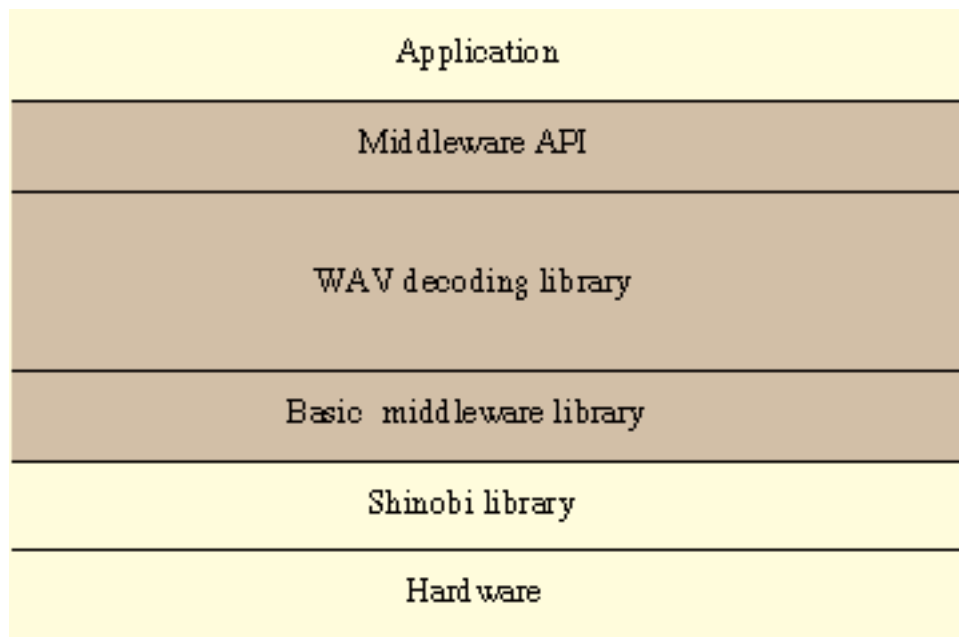


Figure 2.5 WAV Decoder Software Configuration

19.5.3 Data Creation Tools

The following tools for creating Dreamcast ADPCM encoded WAV data is available.

DADENC.EXE : Dreamcast ADPCM encoder

19.5.4 Header Files and Library Files

The following header files are required. By including SG_MW.H, the following file are automatically included.

WAV_MW.H : WAV decoding library header

To play back WAV files, link the following files.

SG_MW.WAV.LIB : WAV decoder library
 SG_MW.LIB : Basic middleware library

19.6 TrueMotion Outline

19.6.1 Features

TrueMotion plays PC-compressed video and audio on Dreamcast. Basically, playback can be carried out by Middleware API, but use the TruePlay API offered by Duck if you want more detailed control.

19.6.2 Software Configuration

The software configuration is shown below. For information on TrueMotion playback, refer to the middleware library manual.

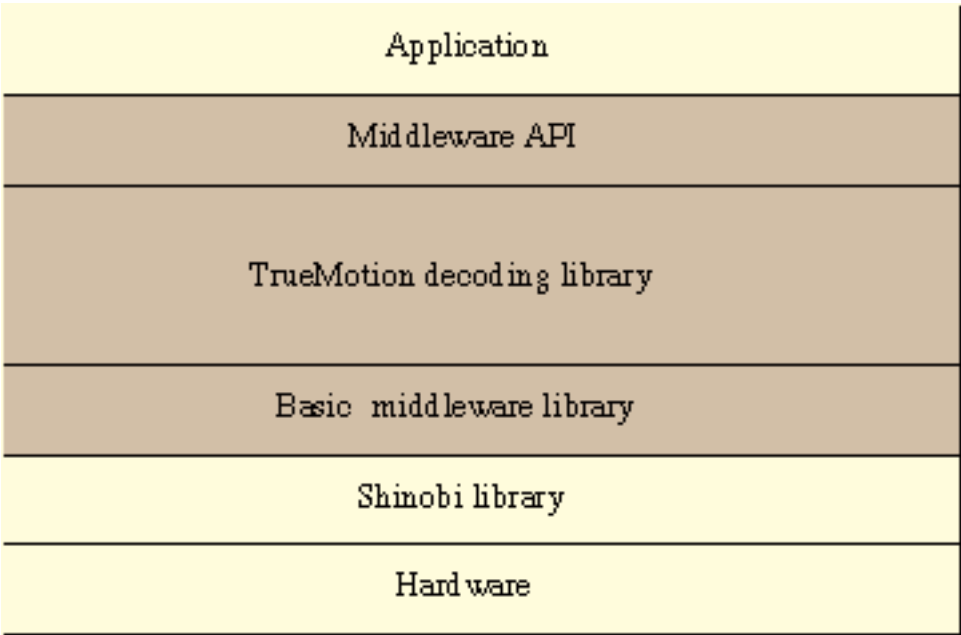


Figure 2.6 *TrueMotion Software Configuration*

19.6.3 Data Creation Tools

To create TrueMotion data, use the VFW (Video for Windows) driver included in the CTK (Compression Tool Kit) setup program. Use this driver from the animation editor tool, such as Adobe Premiere, to create compressed data.

19.6.4 Header File and Library File

The following file is necessary as a header file, but it is automatically included by including SG_MW.H.

TM_MW.H : TrueMotion decoding library header

To use TrueMotion, link the following files.

DUCK_TM.LIB : TrueMotion decoding library
SG_MW.LIB : Basic middleware library

19.7 MPEG/Audio Outline

19.7.1 Features

MPEG/Audio Layer2 data that conforms to ISO11172 can be played back by MPEG/Audio middleware.

19.7.2 Software Configuration

The software configuration is shown below. For information on MPEG/Audio Layer2 playback, refer to the middleware library manual.

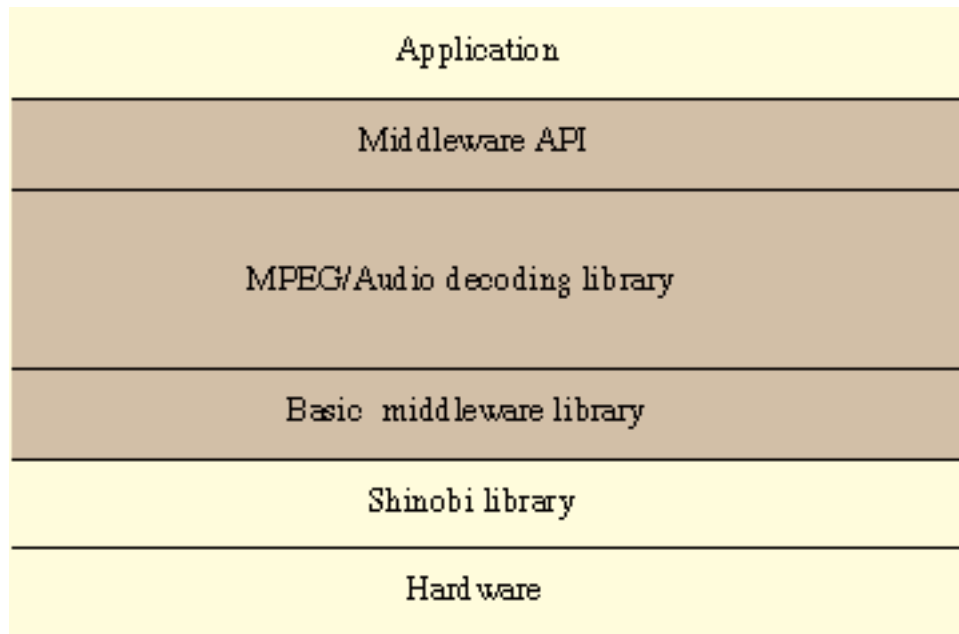


Figure 2.7 MPEG/Audio Software Configuration

19.7.3 Data Creation Tools

MPEG/Audio data is created using commercial tools.

19.7.4 Header Files and Library Files

The following file is necessary as a header file, but it is automatically included by including SG_MW.H.

```
MPA_MW.H      :      MPEG/Audio decoding library header
```

To play back MPEG/Audio Layer2 files, link the following files.

```
MP1A_L2.LIB   :      MPEG/Audio decoding library
SG_MW.LIB     :      Basic middleware library
```


20. ADX Playback Library Implementation Manual

20.1 Introduction

The ADX playback library controls the ADX playback system that is part of the Dreamcast configuration. It is used for playback of ADX-encoded sound data. By using the functions in this library, you can easily implement playback of compressed sound.

The ADX playback library has the following features:

Playback of up to 8 voices with very little CPU load.

CPU load for decoding 44.1 kHz sound data

1voice	8voices
Approx. 0.6%	Approx. 5%

- 1) Playback of various kinds of sound data, including BGM, dialog, sound effects, etc. Data types handled by library:

Data type	Content
ADX data	Compressed single sound file data such as music or dialog.
ACX data (ADX sound effect data)	Compressed multiple ADX data such as music or dialog combined in single file. For memory index playback.
AFS data (ADX file system partition data)	Combination of multiple files such as ADX data and graphics data. For CD index playback.

- 2) Support for various playback methods.

Playback methods supported by library

Playback method	Content
Memory playback	Playback of ADX data in memory
Memory index playback	Playback of ACX data in memory
CD stream playback	Playback of ADX data on CD
CD index playback	Playback of AFS data on CD

- 3) Simultaneous use together with sound driver is possible.
- 4) Multiple CD stream playback operations and CD index playback can be performed simultaneously.
- 5) Seamless loop playback from CD is possible.
- 6) The ADX file system library (AFS) allows reading of game data or other files from CD during CD stream playback.
- 7) AFS makes it easy to handle a large number of files (using approximately 2 bytes per file for file management).
- 8) Playback of 16-bit non-encoded and 4-bit ADPCM encoded WAV data files is possible.

20.2 ADX Playback System

This section shows how the ADX playback system controller works in the ADX Playback Library.

20.2.1 System Configuration

After being decoded in the work RAM of the main CPU, ADX data is transferred to the sound RAM for playback.

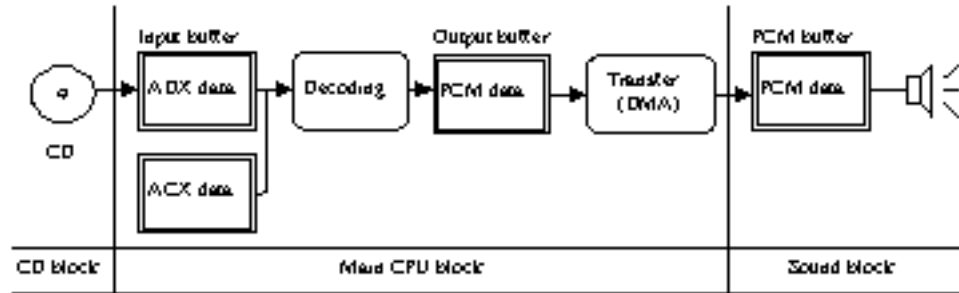


Figure 2.8 System Configuration

Input buffer : Stores ADX data read from the CD.

The size required in the work RAM depends on the sampling frequency and the number of concurrent playback streams. For a monaural 44 kHz source with 3 concurrent playback streams, approximately 100 Kbyte are required. For playback from memory, this is not required.

Output buffer : Stores the decoded 16-bit PCM data.

The size required in the work RAM is (4096+32) samples (4040H bytes) per channel. The combined size of the input buffer and output buffer can be calculated with the ADXT_CALC_WORK macro.

PCM buffer : Stores the PCM data in the sound RAM.

The required area is obtained by the sound library from the end of the sound RAM, according to the number of playback streams. The sound creator must avoid overlap with this area when creating multi-unit files.

The size is 4040H bytes per channel.

ADX data : Data compressed according to the ADX method.

One sound file becomes one set of ADX data.

ACX data : (ADX sound effect data)

Combination of multiple ADX data.

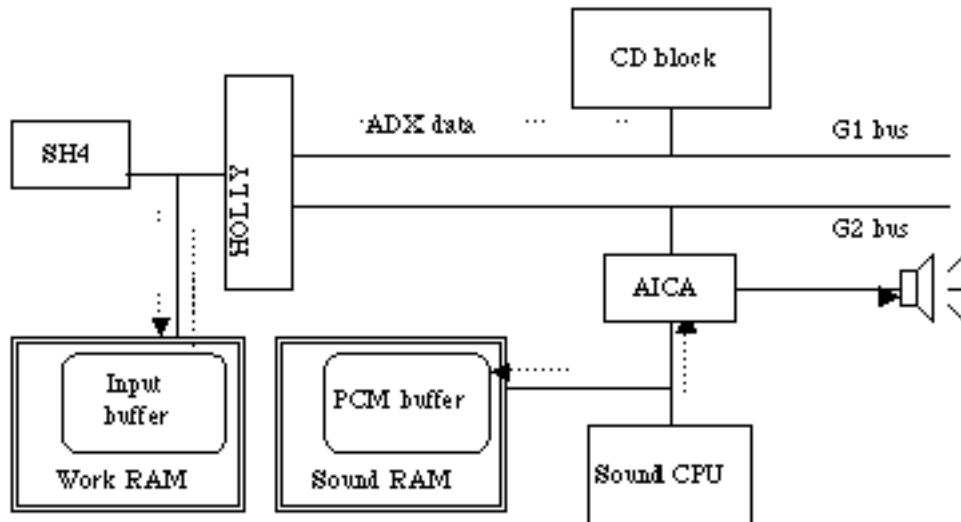
During creation, numbers starting from 0 are assigned to the sound data according to the sequence in the file list.

AFS data : (ADX file system partition data)

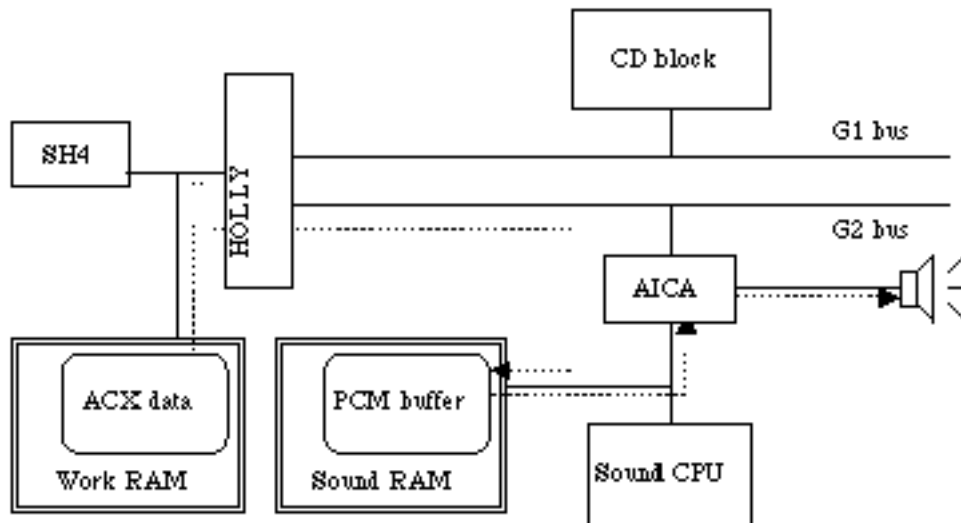
Combination of multiple ADX data and game data. During creation, numbers starting from 0 are assigned as file IDs according to the sequence in the file list. Differently from ACX data, these are positioned on the sector boundary.

The ADX data flow is shown below.

- 1) Playback of ADX data on CD (CD stream playback, CD index playback)



- 2) Playback of ACX data in memory (memory playback, memory index playback)



20.2.2 ADX File System

The ADX file system allows for convenient handling of a large number of files. Several files are linked to create an ADX file system partition file (called an AFS file in this document) which is stored on the CD. The ADX file system can read any specified file or files within the AFS file. This makes playback for example of a large amount of dialog data easy.

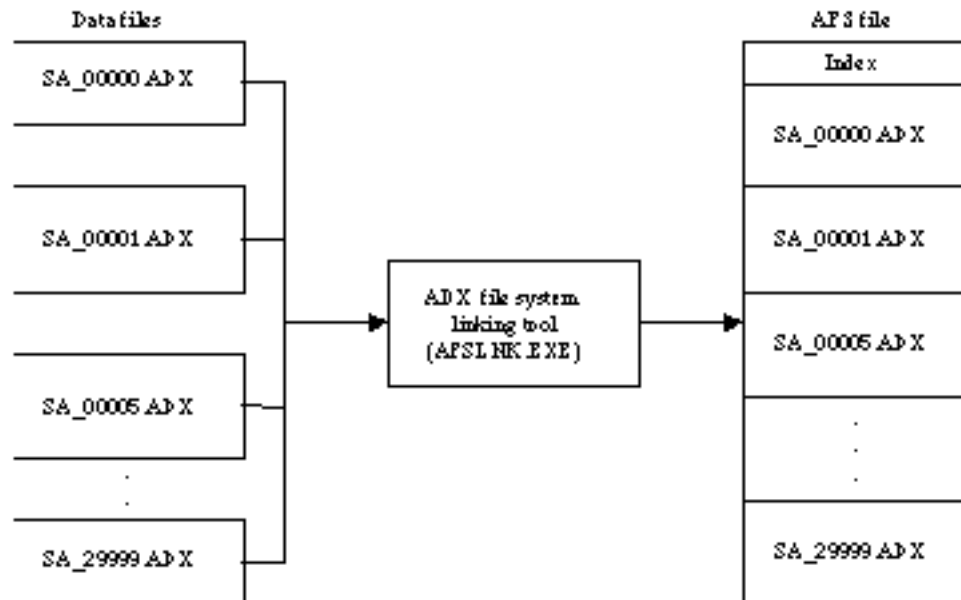


Figure 2.9 AFS File Creation

ADX file system access is performed with the following two keys.

- 1) Partition ID
User-specified number (0 - 255)
The ADXF_LoadPartition function is used to read partition information, and the partition ID is correlated to the AFS file.
- 2) File ID
Internal AFS file number
The linking tool (AFSLNK.EXE) outputs the file ID as a header file.

ADX file system access is performed according to the following procedure.

- 1) Read partition information
The ADXF_LoadPartition function is used to read partition information.
There are approximately 2 bytes of partition information per file.
- 2) Playback of ADX data in AFS file
The ADXT_StartAfs function is used to specify the partition ID and file ID for playback.
- 3) Read-out of data in AFS file
The ADXF_OpenAfs function is used to open the file handle and read game data or similar.

The ADX file system can read data from CD also during CD stream playback.

20.2.3 Module Configuration

Because the ADX playback system can coexist with the sound library and sound drivers, the programmer can use the sound library and ADX library together. For example, ADX could be used for stream playback while playing a MIDI sequence.

The module configuration of the library is shown below.

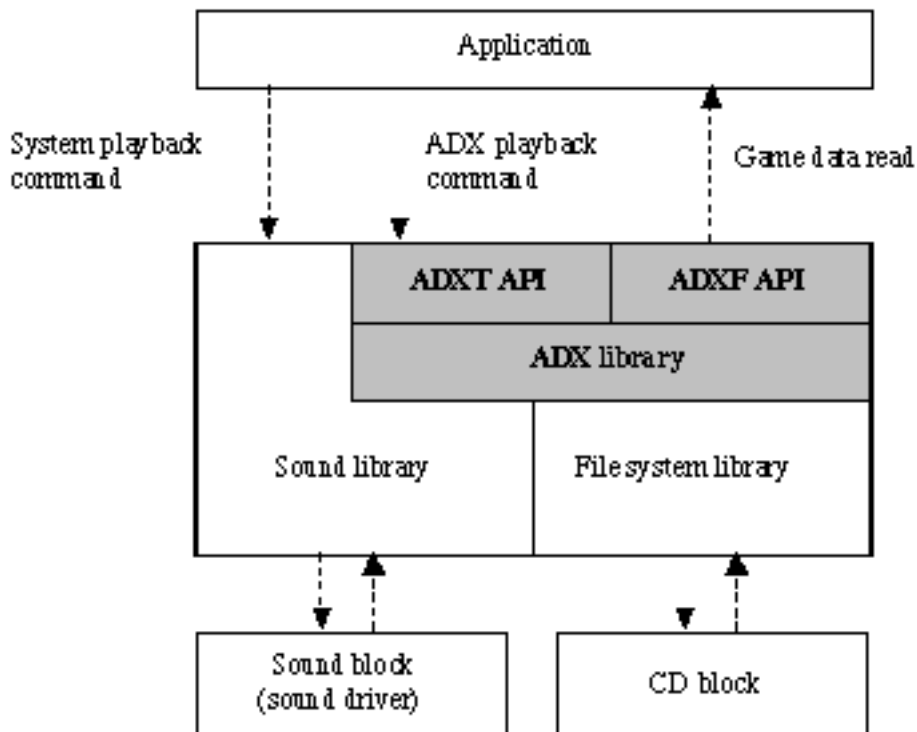
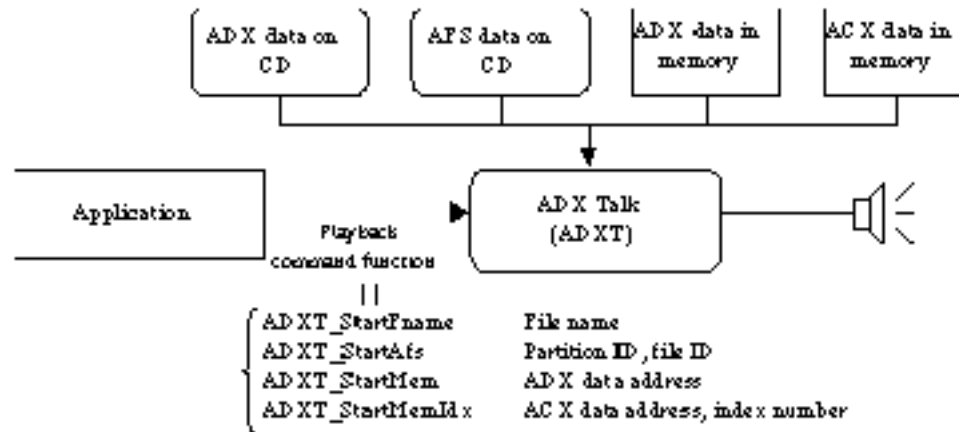


Figure 2.10 *Module Configuration Diagram*

20.2.4 Application Interface (API)

The ADX library API is an object oriented interface. First, the handle for ADX playback 'ADX Talk' (ADXT) is generated. Thereafter, playback of ADX data on CD or in memory can be performed simply by giving the playback command to this handle. One handle plays one ADX data. ADX data can be played back either in mono or in stereo. It is also possible to play several ADX data simultaneously.



20.3 ADX Data Creation

This section describes how to create ADX data, ACX data, and AFS data.

1) ADX data creation

ADX data are PCM sound data compressed with ADX. The original WAV or AIFF format file is processed using the 'adxencd.exe' program. To compress a file, run 'adxencd.exe' from the command line while giving the file name as an argument, as shown below. A file with the extension 'ADX' is created in the current directory.

```
C:\TEMP>adxencd sample.wav      'sample.wav' is compressed to 'sample.adx'.
```

To create ADX data with a different sampling frequency from the original, specify the 'sf' option. The resulting sampling frequencies is an integer fraction of the original. When converting the sampling frequency, the new cutoff frequency is calculated as 0.45 times the specified sampling frequency, and a Butterworth filter with that cutoff frequency is used to attenuate the high range. If the sound seems to be lacking in highs, use an off-the- shelf tool to convert the sampling frequency beforehand.

```
C:\TEMP>adxencd sample.wav  -sf=22050
```

Converts 'sample.wav' to 'sample.adx' with sampling frequency 22050 Hz.

To perform loop playback, specify the loop start point and loop end point in sample units, as shown below.

```
C:\TEMP>adxencd sample.wav  -lps1500 -lpe200000
```

Converts 'sample.wav' to 'sample.adx', with loop playback from sample 1500 to sample 199999.

Note: When specifying a loop, do not specify sampling frequency conversion. Otherwise noise may occur during loop playback. To use both sampling frequency conversion and loop playback, perform sampling frequency beforehand with a separate tool.

Sound Input file format

File format	WAV or AIFF format
Number of quantization bits	16 or 8
Sampling frequency	Arbitrary

1) One-pass creation of ADX data

ax.exe can be used to encode a large amount of sound data in one operation. Create a file list with the dir/b command and use this list to encode multiple files.

```
C:\TEMP>dir /b *.wav > wavlist.flx
```

Creates a list of WAV files

```
C:\TEMP>ax iadxencd %s% wavlist.flx
```

Encodes all WAV files in list

2) ACX data creation

ACX data are a linked series of multiple ADX data (sound effects, dialog etc.) created according to the above method. Linking is performed with the program 'adxcat.exe', using a file list as argument, as shown below.

```
C:\TEMP>dir /b *.adx > se.flx      Creates the linked file 'se.acx' consisting of all ADX
C:\TEMP>adxcat se.flx              files in the current directory.
```

3) AFS file creation

An AFS file consists of linked ADX data and graphic data or other data. Linking is performed with the program 'adxlnk.exe', using a file list as argument, as shown below.

```
C:\TEMP>dir /b *.adx > pat01.als    Creates the linked file 'pat01.afs'
                                      consisting of all
C:\TEMP>afslnk pat01.als             files in the current directory.
```

To link data from other locations besides the current directory, use the following syntax.

```
C:\TEMP>dir /b pat01\*.adx > pat01.als    Creates the linked file 'pat01.afs'
                                      consisting of all
C:\TEMP>afslnk pat01.als /dir=pat01      files in the directory 'pat01'.
```

20.4 Using the Library Functions

This section describes how to initialize the library, play sound effects in memory, and play ADX data files from CD.

20.4.1 Library Initialization and Shutdown

To initialize the entire library, use the `ADXT_Init` function when starting the application. This initialization should basically be performed only once, at the start of the game application. To shut down the library, use the `ADXT_Finish` function.

```
/* application */
void main()
{
    SoundInit();
    /* set transfer mode to DMA */
    sdMemBlkSetTransferMode(SDE_MEMBLK_TRANSFER_MODE_DMA);
    ADXT_Init();      /* initialize library */
    :
    ADXT_Finish();    /* shut down library */
}
```

20.4.2 ADXT Handle Creation

After initialization, allot the work area and create the ADXT handles as shown below. This function serves to open the sound library PCM stream port and allocate other resources. Creating the necessary handles before the start of a scene and avoiding dynamic handle creation and deletion helps to stabilize the CPU load.

```
/* work area size parameter */
#define MAX_CH    (2)    /* stereo (mono also possible) */
#define CDSTM     (1)    /* CD stream playback */
#define MAX_SFREQ (44100) /* sampling frequency 44100 Hz */

/* work area size */
#define WORKSIZE (ADXT_CALC_WORK(MAX_CH, ADXT_PLY_STM, MAX_CDSTM, MAX_SFREQ))
/* work area */
static char work[WORKSIZE];

void adx_play_cdstm(void)
{
    ADXT adxt;          /* ADXT handle */

    :

    :

    (process 1 game scene)
    :
    :

    ADXT_Destroy(adxt); /* delete handle */
}
```

20.4.3 Playback of ADX Data in Memory (Memory Playback)

ADX data in memory can be played back with the ADXT_StartMem function. Specify the ADX data address to the ADXT handle and execute the ADXT_StartMem function. For playback from memory, the work area does not require an input buffer, therefore the size approx. 16 Kbyte x number of channels.

```
/* work area size parameter */
#define MAX_CH                (2)                /* stereo (mono also possible) */
#define MAX_SFREQ             (22050)           /* sampling frequency 22050 Hz */

/* work area size */
#define WORKSIZE (ADXT_CALC_WORK(MAX_CH, ADXT_PLY_MEM, 0, MAX_SFREQ))
/* work area */
static char work[WORKSIZE];

void adx_play_mem(void)
{
    ADXT adxt;                                /* ADXT handle */
    void *adx = (void *)0x8c100000;           /* ADX data address */

    adxt = ADXT_Create(MAX_CH, work, WORKSIZE); /* create handle */
    :
    ADXT_StartMem(adxt, adx);                  /* playback adx */
    :
}
```

20.4.4 Playback of ACX Data in Memory (Memory Index Playback)

ACX data in memory can be played with the ADXT_StartMemIdx function. Specify the ADXT handle and index number to the ACX data address and execute the ADXT_StartMemIdx function.

```
/* work area size parameter */
#define MAX_CH                (1)                /* mono only */
#define MAX_SFREQ             (22050)           /* sampling frequency 22050 Hz */

/* work area size */
#define WORKSIZE (ADXT_CALC_WORK(MAX_CH, ADXT_PLY_MEM, 0, MAX_SFREQ))
/* work area */
static char work[WORKSIZE];

void adx_play_memidx(void)
{
    ADXT adxt;                                /* ADXT handle */
    void *acx = (void *)0x8c100000;           /* ACX data address */

    adxt = ADXT_Create(MAX_CH, work, WORKSIZE); /* create handle */
    :
    ADXT_StartMemIdx(adxt, acx, no)            /* play specified data number in
                                                acx data */
    :
}
```

20.4.5 Playback of ADX Data on CD (CD Stream Playback)

ADX data on CD can be played with the ADXT_StartFname function. Specify the file name to the ADXT handle and execute the ADXT_StartFname function.

```
/* work area size parameter */
#define MAX_CH (2) /* work area size parameter */
#define MAX_CDSTM (3) /* number of CD streams for
concurrent playback (3) */
#define MAX_SFREQ (44100) /* sampling frequency 44100 Hz */

/* work area size */
#define WORKSIZE (ADXT_CALC_WORK(MAX_CH, ADXT_PLY_STM, MAX_CDSTM, MAX_SFREQ))
/* work area */
static char work[WORKSIZE];

void adx_play_cdstm(void)
{
    ADXT adxt; /* ADXT handle */

    adxt = ADXT_Create(MAX_CH, work, WORKSIZE); /* create handle */
    :
    ADXT_StartFname(adxt, "SE007.ADX"); /* playback of "SE007.ADX" */
    :
    ADXT_StartFname(adxt, "SA019.WAV"); /* playback of "SA019.WAV" */
    :
}
```

20.4.6 Playback of AFS Data on CD (CD Index Stream Playback)

ADX data on CD can be played according to the following procedures.

- 1) Use ADXF_LoadPartition function to read partition information during initialization.
The partition ID, AFS file name, partition information area, and maximum number of files are used as arguments.
- 2) Specify partition ID and file ID to ADXT handle and execute ADXT_StartAfs function. Partition ID is the value defined in (1) and the file ID is the sequence number in the file list created during linking/
The file ID can also be specified using the header file output by AFSLNK.EXE.

```
#include ipat01.h /* header file defining file ID */
/* output by AFSLNK.EXE */

/* partition ID */
#define PAT01 (0)
/* maximum number of files in partition */
#define MAX_PAT_NFILES (10000)

/* work area size parameter */
#define MAX_CH (2) /* stereo (mono also possible) */
```



```
#define MAX_CDSTM          (3)                      /* number of CD streams for
concurrent playback (3) */
#define MAX_SFREQ(44100) /* sampling frequency 44100 Hz */
/* work area size */
#define WORKSIZE (ADXT_CALC_WORK(MAX_CH, ADXT_PLY_STM, MAX_CDSTM, MAX_SFREQ))

/* partition information area */
static char pinfo[ADXF_CALC_PTINFO_SIZE(MAX_PAT_NFILES)];
/* work area */
static char work[WORKSIZE];

void main(void)
{
/* partition information read-in */
ADXF_LoadPartition(PAT01, ipat01.afs, pinfo, MAX_PAT_NFILES);
adx_play_afs();
}

void adx_play_afs(void)
{
ADXT adxt;          /* ADXT handle */

adxt = ADXT_Create(MAX_CH, work, WORKSIZE);          /* create handle */
:
ADXT_StartAfs(adxt, PAT01, SE007_ADX); /* playback of "SE007.ADX" */
/* SE007_ADX defined in "pat01.h" */
}
```

20.4.7 Get Playback Status

The ADX handle status can be obtained with the ADXT_GetStat function. By monitoring this status, it can be determined whether ADX data playback is completed or not.

When ADXT_STAT_PLAYING and ADXT_STAT_DECEND are returned, sound is being output. When ADXT_STAT_PLAYEND is returned, playback has ended. When the automatic error recovery mode ADXT_RMODE_STOP is active, a read error or other error during CD-ROM playback will trigger the ADXT_STAT_STOP condition and playback stops.

The ADXT handle status constants are listed in the table below.

Constant	Value	Status	Sound output
ADXT_STAT_STOP	0	Stopped	Off
ADXT_STAT_DECINFO	1	Getting ADX header information	Off
ADXT_STAT_PREP	2	Preparing for playback	Off
ADXT_STAT_PLAYING	3	Decoding and playback in progress	On
ADXT_STAT_DECEND	4	Decoding completed	On
ADXT_STAT_PLAYEND	5	Playback ended	Off

<Getting playback status>

```
long stat;

adxt = ADXT_Create(MAX_CH, work, WORKSIZE);          /* create ADXT handle */
ADXT_StartFname(adxt, "SE007.ADX");                  /* playback of "SE007.ADX" */

for (;;) {
    stat = ADXT_GetStat(adxt);                        /* get playback status */
    if ( stat == ADXT_STAT_PLAYING || stat == ADXT_STAT_DECEND ) {
        /* audio output processing */
        :
        :
    } else if ( stat == ADXT_STAT_PLAYEND ) {
        /* playback end processing */
        :
        :
    }
    break;
    } else if ( stat == ADXT_STAT_STOP ) {
        /* error stop */
        :
        :
    }
    break;
}
```

20.5 Table of Library Functions

The ADX library functions are listed in the following table.

Function	Function Name	No.
ADX Talk functions (high-level functions)		
ADX library initialization	ADXT_Init	6.1
ADX library shutdown	ADXT_Finish	6.2
Work area size calculation	ADXT_CALC_WORK	6.3
ADXT handle creation	ADXT_Create	6.4
ADXT handle deletion	ADXT_Destroy	6.5
Start playback of specified file	ADXT_StartFname	6.6
Start memory playback	ADXT_StartMem	6.7
Start memory index playback	ADXT_StartMemIdx	6.8
Start CD index playback	ADXT_StartAfs	6.9
Stop playback	ADXT_Stop	6.10
Get status	ADXT_GetStat	6.11
Get playback time in sample units	ADXT_GetTime	6.12
Get playback time in real time	ADXT_GetTimeReal	6.13
Get total number of samples	ADXT_GetNumSmpl	6.14
Get sampling frequency	ADXT_GetSfreq	6.15
Set output volume	ADXT_SetOutVol	6.16
Get output volume setting	ADXT_GetOutVol	6.17
ADXT_SetOutPan	ADXT_SetOutPan	6.18
Get panpot setting	ADXT_GetOutPan	6.19
Set server function call frequency	ADXT_SetSvrFreq	6.20
Server function (update internal status)	ADXT_ExecServer	6.21
Get error code	ADXT_GetErrCode	6.22
Clear error code	ADXT_ClearErrCode	6.23
Set auto error recovery	ADXT_SetAutoRcvr	6.24
Set error callback function	ADXT_EntryErrFunc	6.25
Playback pause and resume	ADXT_Pause	6.26

20.6 Description of Library Functions

Title	Function	Function Name	No
Function specification	ADX library initialization	ADXT_Init	6.1

[Format] `void ADXT_Init(void);`

[Input] None

[Output] None

[Function value] None

[Function] Initializes the ADX library.

Registers the server function with the V-Sync interrupt.

[Remarks] In the Shinobi environment, transfer load can be reduced by specifying DMA transfer mode for the sound library. This is performed as follows.

```
/* Normal sound library/driver initialization */
SoundInit(snddrv);
/* DMA transfer mode */
sdMemBlkSetTransferMode(SDE_MEMBLK_TRANSFER_MODE_DMA);
[Example]
SoundInit(snddrv);
sdMemBlkSetTransferMode(SDE_MEMBLK_TRANSFER_MODE_DMA);
/* ADX library initialization */
ADXT_Init();
```

Title	Function	Function Name	No
Function specification	ADX library shutdown	ADXT_Finish	6.2

[Format] `void ADXT_Finish(void);`

[Input] None

[Output] None

[Function value] None

[Function] Performs shutdown processing for the ADX library.
Deletes created handles and deletes the server function from interrupt processing.

```
[Example]
/* ADX library initialization */
ADXT_Init();
:
:
:
/* ADX library shutdown */
ADXT_Finish();
```

Title	Function	Function Name	No
Macro specification	Work area size calculation	ADXT_CALC_WORK	6.3

```

[Format]                                     int ADXT_CALC_WORK(maxch,
flg_cdstm, max_cdstm, max_sfreq);
[Input]           maxch                       :   maximum number of playback
flg_cdstm         :   playback mode

```

ADXT_PLY_MEM: memory playback

ADXT_PLY_STM: stream playback

```

max_cdstm       :   maximum number of CD streams
max_sfreq       :   max_sfreq: maximum playback sampling frequency
[Output]        None
[Function value] Work area size [bytes]
[Function]       Calculates the work area size for each ADXT handle.

```

The maximum number of CD streams specifies the number of streams that can be read from CD simultaneously.

Normally, this is the sum of ADXT handles and ADXF handles used simultaneously.

[Remarks] The work area size calculation formula is as follows.

work area = input buffer size + output buffer size

Table 20.7 sampling frequency 44100Hz, mono

	Memory playback	CD stream playback
Input buffer size	0 byte	25Kbyte + 25Kbyte * number of concurrent playback streams
Output buffer size	16Kbyte	16Kbyte

- 1) For stereo playback, double the area is required.
- 2) The input buffer can be reduced according to the sampling frequency.

Title	Function	Function Name	No
Function specification	create ADXT handle	ADXT_Create	6.4

Dreamcast Shinobi Library Specification

[Format] ADXT ADXT_Create(long maxch, void *work, long worksize);

[Input] maxch: maximum number of playback channels (mono: 1, stereo: 2)

work : work area

worksize : work area size

[Output] None

[Function value] Pointer to generated ADXT structure element (ADXT handle)

[Function] Creates an ADXT handle.

[Remarks] maxnch should be set to "1" for mono playback only and to "2" for stereo playback.

Also when maxnch is set to "2", mono playback is possible.

The work area is allocated by the syMalloc function.

Title	Function	Function Name	No
Function specification	ADXT handle deletion	ADXT_Destroy	6.5

[Format] void ADXT_Destroy(ADXT adxt);

[Input] adxt : ADXT handle

[Output] None

[Function value] None

[Function] Deletes the specified ADXT handle.

[Example]

```
/* work area size */
#define WKSIZe ADXT_CALC_WORK(2, ADXT_PLY_STM, 1, 44100)

char *work[WKSIZe];
ADXT adxt;

adxt = ADXT_Create(2, work, WKSIZe);
:
:
:
ADXT_Destroy(adxt);
```

/* work area */
/* ADXT handle */

/* create ADXT handle */

/* delete ADXT handle */

Title	Function	Function Name	No
Function specification	Start CD stream playback	ADXT_StartFname	6.6

[Format] void ADXT_StartFname(ADXT adxt, char *fname);

[Input] adxt : ADXT handle

fnmae : sound file name

[Output] None

[Function value] None

[Function] Starts playback of file specified by fname. When the function is executed for a handle that is already being played, sound output stops and playback of the specified file starts. ADX files and WAV files can be specified as sound files. By executing the function for different handles, several sound files can be played simultaneously.

[Example]

```
/* playing back 1 stream of 44 kHz stereo data */
#define WKSIZe      ADXT_CALC_WORK(2, ADXT_PLY_STM, 1, 44100)

char *work[WKSIZe];          /* work area */
ADXT adxt;                   /* ADXT handle */

adxt = ADXT_Create(2, work, WKSIZe);          /* create ADXT handle */
ADXT_StartFname(adxt, "MUSIC.ADX");          /* start playback */
:
:
ADXT_StartFname(adxt, "SE2.WAV");            /*start playback */
:
:
```

Title	Function	Function Name	No
Function specification	Start memory playback	ADXT_StartMem	6.7

[Format] void ADXT_StartMem(ADXT adxt, void *adxdatt);

[Input] adxt : ADXT handle

adxdatt : sound data address

[Output] None

[Function value] None

[Function] Plays sound data in memory specified by adxdatt. Allows sound playback with better response than CD stream playback. Delay time is approx. 50 milliseconds.

[Example]

```
/* When playing mono data */
#define WKSIZEx ADXT_CALC_WORK(1, ADXT_PLY_MEM, 0, 0)

char *work[WKSIZEx]; /* work area */
char adxdatt[ADXDAT_SIZE]; /* data area */
ADXT adxt; /* ADXT handle */

load_data(adxdatt); /* read sound data */
adxt = ADXT_Create(1, work, WKSIZEx); /* create ADXT handle */
ADXT_StartMem(adxt, adxdatt); /* start playback */
:
:
```

Title	Function	Function Name	No
Function specification	Start memory index playback	ADXT_StartMem	6.8

[Format] void ADXT_StartMemIdx(ADXT adxt, void *acxdatt, long no);

[Input] adxt : ADXT handle

acxdatt : ACX data address

no : index number

[Output] None

[Function value] None

[Function] Plays data of specified number within ACX data.

[Example]

```
/* mono data playback */
#define WKSIZEx ADXT_CALC_WORK(1, ADXT_PLY_MEM, 0, 0)

char *work[WKSIZEx]; /* work area */
char acxdatt[ADXDAT_SIZE]; /* data area */
ADXT adxt; /* ADXT handle */

load_data(acxdatt); /* read sound data */
adxt = ADXT_Create(1, work, WKSIZEx); /* create ADXT handle */
ADXT_StartMem(adxt, acxdatt, 5); /* start playback */
:
:
ADXT_StartMem(adxt, acxdatt, 8); /* start playback */
:
:
```


Title	Function	Function Name	No
Function specification	Start CD index playback	ADXT_StartAfs	6.9

[Format] void ADXT_StartAfs(ADXT adxt, long pat_id, long file_id);

[Input] adxt : ADXT handle

pat_id : partition ID

file_id : file ID

[Output] None

[Function value] None

[Function] Plays data from AFS file on CD.

The ADXF_LoadPartition function serves for correlating the par_id and the AFS file name. The file_id refers to the sequential number inside the AFS file.

AFSLNK.EXE outputs a header file specifying the file ID. A message corresponding to the file name can also be specified.

[Example]

```

/* simultaneous playback of 44 kHz stereo and 22 kHz mono */
/* 44 kHz stereo playback work area */
#define WKSIZ44                                ADXT_CALC_WORK(2, ADXT_PLY_STM, 2, 44100)
/* 22 kHz mono playback work area */
#define WKSIZ22                                ADXT_CALC_WORK(1, ADXT_PLY_STM, 2, 44100)
/* maximum number of files in partition */
#define MAX_NFILES                             10000
/* partition information area */
char ptinfo[ADXF_CALC_PTINFO_SIZE(MAX_NFILES)];
char *work1[WKSIZ44];                          /* work area */
char *work2[WKSIZ22];                          /* work area */
ADXT adxt1, adxt2;                             /* ADXT handle */
/* partition information read-in */
ADXF_LoadPartition(PATID = 0, ipat01.afsî, ptinfo, MAX_NFILES);
adxt1 = ADXT_Create(2, work1, WKSIZ44);        /* create ADXT handle */
adxt2 = ADXT_Create(1, work2, WKSIZ22);        /* create ADXT handle */
ADXT_StartAfs(adxt1, PATID, BGM);              /* start playback */
:
:
ADXT_StartAfs(adxt1, PATID, SRF137);            /* start playback */
:
:

```

Title	Function	Function Name	No
Function specification	Stop playback	ADXT_Stop	6.10

[Format]

void ADXT_Stop(ADXT adxt);

[Input]

adxt : ADXT handle

[Output]

None

[Function value]

None

[Function]

Stops ADX playback.

[Example]

```
/* 44 kHz stereo data 1 stream playback */
#define WKSIZE
ADXT_PLY_STM, 1, 44100)

char *work[WKSIZE];
ADXT adxt;

adxt = ADXT_Create(2, work, WKSIZE);
ADXT_StartFname(adxt, "MUSIC.ADX");
:
:
/* B button pressed for stop */
if ( per->press & PDD_DGT_TB )
    ADXT_Stop(adxt);
```

```
ADXT_CALC_WORK(2,
/* work area */

/* create ADXT handle */
/* start playback */

/* stop playback */
```

Title	Function	Function Name	No
Function specification	Get status	ADXT_GetStat	6.11

[Format] long ADXT_GetStat(ADXT adxt);

[Input] adxt : ADXT handle

[Output] None

[Function value] Operation status of current ADXT handle

[Function] Gets current operation condition of adxt.

Possible status modes are as follows.

Mode definition	Constant	Status
ADXT_STAT_STOP	0	Stopped
ADXT_STAT_DECINFO	1	Getting ADX header information
ADXT_STAT_PREP	2	Preparing for playback
ADXT_STAT_PLAYING	3	Decoding & playback in progress
ADXT_STAT_DECEND	4	Decoding completed
ADXT_STAT_PLAYEND	5	Playback completed

[Example]

```
/* 44 kHz mono data 1 stream playback */
#define WKSIZE                                ADXT_CALC_WORK(1,
ADXT_PLY_STM, 1, 44100)

char *work[WKSIZE];      /* work area */
ADXT adxt;                /* ADXT handle */
long stat;                /* operation status */

adxt = ADXT_Create(1, work, WKSIZE);          /* create ADXT handle */
ADXT_StartAfs(adxt, pat_id, 0);               /* start playback */
for (;;) {
    /* activate animation while sound is produced */
    stat = ADXT_GetStat(adxt);
    if ( stat == ADXT_STAT_PLAYING && stat == ADXT_STAT_DECEND )
        user_animate_obj();

    /* play next sound when playback ends */
    if ( stat == ADXT_STAT_PLAYEND && stat == ADXT_STAT_STOP )
        ADXT_StartAfs(adxt, pat_id, 1);
}
```

Dreamcast Shinobi Library Specification

Note: Completion of playback can be judged by the PLAYEND or STOP status. When a CD read error has occurred, the status is STOP.

Title	Function	Function Name	No
Function specification	Get playback time in sample units	ADXT_GetTime	6.12

[Format] void ADXT_GetTime(ADXT adxt, long *ncount, long *tscale);

[Input] adxt : ADXT handle

[Output] ncount : playback sample number

tscale : sampling frequency [Hz]

[Function value] None

[Function] Gets playback time in sample units.

[Example]

```
/* 44 kHz stereo data 1 stream playback */
#define WKSIZE      ADXT_CALC_WORK(2, ADXT_PLY_STM, 1, 44100)

char *work[WKSIZE];          /* work area */
ADXT adxt;                   /* ADXT handle */
long nsmpl, sfreq;           /* playback time */
long nfrm, hh, mm, ss, ff;   /* time */

adxt = ADXT_Create(2, work, WKSIZE); /* create ADXT handle */
ADXT_StartFname(adxt, "music.adx"); /* start playback */
for (;;) {
    /* playback time display (frame unit 1/60) */
    ADXT_GetTime(adxt, &nsmpl, &sfreq);
    nfrm = nsmpl*60/sfreq; /* total number of frames */
    hh = nfrm / (60*60*60); /* hours */
    mm = (nfrm ñ hh*60*60*60) / (60*60); /* minutes */
    ss = (nfrm ñ (hh*60+mm)*60*60) / 60; /* seconds */
    ff = nfrm % 1000; /* frames */
    disp_time(hh, mm, ss, ff);
}
```

Note: When 44 kHz sound data are handled as shown in the example above, nfrm is calculated with a multiplication factor of 60. Therefore the maximum playback time value is 811 seconds. With the ADXT_GetTime function, the maximum is approx. 13.5 hours, but when units are converted, the above limitation must be taken into consideration.

Title	Function	Function Name	No
Function specification	Get playback time in real time	ADXT_GetTimeReal	6.13

[Format] long ADXT_GetTimeReal(ADXT
adxt);

[Input] adxt : ADXT handle

[Output] None

[Function value] Playback time [1/100sec]

[Function] Gets playback time in real time, using units of 1/100 seconds.

[Example]

```
/* 44 kHz stereo data 1 stream playback */
#define WKSIZE ADXT_CALC_WORK(2,
ADXT_PLY_STM, 1, 44100)

char *work[WKSIZE]; /* work area */
ADXT adxt; /* ADXT handle */
long time; /* playback time */
long hh, mm, ss, ff; /* time */

adxt = ADXT_Create(2, work, WKSIZE); /* create ADXT handle */
ADXT_StartFname(adxt, imusic.adxi); /* start playback */
:
for (;;) {
/* playback time display (frame unit 1/100) */
time = ADXT_GetTime(adxt, &nsmpl, &sfreq);
hh = time / (60*60*100); /* hours */
mm = (time ñ hh*60*60*100)/(60*100); /* minutes */
ss = (time ñ (hh*60+mm)*60*100)/100; /* seconds */
ff = time % 100; /* frames */
disp_time(hh, mm, ss, ff);
}
```

Title	Function	Function Name	No
Function specification	Get total number of samples	ADXT_GetNumSmpl	6.14

[Format] long ADXT_GetNumSmpl(
ADXT adxt);

[Input] adxt : ADXT handle

[Output] None

[Function value] Total number of sound data samples

[Function] ets total number of sound data samples for current playback.

[Remarks]

This information can be obtained in the interval
extending from playback preparation
(ADXT_STAT_PREP = 2) to playback end
(ADXT_STAT_PLAYEND = 5).

[Example]

```
/* 44 kHz stereo data 1 stream playback */
#define WKSIZE                                ADXT_CALC_WORK(2,
ADXT_PLY_STM, 1, 44100)

char *work[WKSIZE];                          /* work area */
ADXT adxt;                                   /* ADXT handle */
long total_nsmp1;                             /* total number of sample */
long nsmp1, sfreq;                            /* playback time */
long percent;                                /* percent */

adxt = ADXT_Create(2, work, WKSIZE);          /* create ADXT handle */
ADXT_StartFname(adxt, imusic.adx1);          /* start playback */
for (;;) {
/* display playback progress (percent) */
if ( ADXT_GetStat(adxt) >= ADXT_STAT_PREP ) {

ADXT_GetNumSmp1(adxt);                        total_nsmp1 =
                                              ADXT_GetTime(adxt, &nsmp1,
&sfreq);                                     percent = nsmp1 * 100 /

total_nsmp1;
} else {
                                              percent = 0;

}
disp_progress(percent);
}
```

Title	Function	Function Name	No
Function specification	Get sampling frequency	ADXT_GetSfreq	6.15

[Format]

long ADXT_GetSfreq(ADXT adxt);

[Input]

adxt : ADXT handle

[Output]

None

[Function value]

Sound data sampling frequency [Hz]

[Function]

Gets sampling frequency of currently playing sound data.

[Remarks]

This information can be obtained in the interval extending from playback
preparation (ADXT_STAT_PREP = 2) to playback end
(ADXT_STAT_PLAYEND = 5).

[Example]

```

/* 44 kHz stereo data 1 stream playback */
#define WKSIZE          ADXT_CALC_WORK(2, ADXT_PLY_STM, 1, 44100)

char *work[WKSIZE];          /* work area */
ADXT adxt;                   /* ADXT handle */
long sfreq;                   /* sampling frequency */

adxt = ADXT_Create(2, work, WKSIZE);          /* create ADXT handle */
ADXT_StartFname(adxt, imusic.adx1);           /* start playback */
for (;;) {
/* display playback frequency (Hz) */
if ( ADXT_GetStat(adxt) >= ADXT_STAT_PREP ) {
    sfreq = ADXT_GetSfreq(adxt);
    disp_sampling_frequecy(sfreq);
}
}

```

Title	Function	Function Name	No
Function specification	Set output volume	ADXT_SetOutVol	6.16

[Format]		void ADXT_SetOutVol(ADXT adxt, long vol);
[Input] adxt	:	ADXT handle
vol	:	attenuation level (from 0: -0dB to -960: -96dB)
[Output]		None
[Function value]		None
[Function]		Sets the output volume. Can be used before or during playback.
vol setting values 0	:	-0dB no attenuation (maximum)
-30	:	-3dB approx. 70%
-60	:	-6dB approx. 50%
-999	:	-99.9dB mute

Title	Function	Function Name	No
Function specification	Get output volume setting	ADXT_GetOutVol	6.17

Dreamcast Shinobi Library Specification

[Format] void ADXT_GetOutVol(ADXT adxt);

[Input] adxt : ADXT handle

[Output] None

[Function value] Output volume setting value (from 0: -0dB to -999: -99.9dB)

[Function] Gets the output volume setting.

[Example]

```
/* 44 kHz stereo data 1 stream playback */
#define WKSIZE                                ADXT_CALC_WORK(2,
ADXT_PLY_STM, 1, 44100)

char *work[WKSIZE];                          /* work area */
ADXT adxt;                                   /* ADXT handle */
long vol;                                    /* current volume */
long fade_vol;                               /* fader range */

fade_vol = 1000/180;                          /* fade in 3 seconds */
adxt = ADXT_Create(2, work, WKSIZE);          /* create ADXT handle */
ADXT_StartFname(adxt, "music.adx");           /* start playback */
for (;;) {
/* fade-in */
if ( per->on & PDD_DGT_TA ) {
        vol = ADXT_GetOutVol(adxt);
        ADXT_SetOutVol(adxt, vol+fade_vol);
}
/* fade-out */
if ( per->on & PDD_DGT_TB ) {
        vol = ADXT_GetOutVol(adxt);
        ADXT_SetOutVol(adxt, vol-fade_vol);
}
}
```

Title	Function	Function Name	No
Function specification	Set panpot	ADXT_SetOutPan	6.18

[Format] void ADXT_SetOutPan(ADXT adxt, long ch, long pan);

[Input] adxt : ADXT handle

ch : channel number (0, 1)

ADXT_CH_L(0): left channel, ADXT_CH_R(1): right channel

Pan : panpot setting value (from -15 to +15, -128)

ADXT_PAN_LEFT = -15, ADXT_PAN_CENTER = 0

ADXT_PAN_RIGHT = 15, ADXT_PAN_AUTO = -128

[Output] None

Dreamcast Shinobi Library Specification

Title	Function	Function Name	No
Function specification	Set server function call frequency	ADXT_SetSvrFreq	6.20

[Format] void ADXT_SetSvrFreq(ADXT adxt, long freq);

[Input] adxt : ADXT handle

freq : server function call frequency (number of calls per second)

[Output] None

[Function value] None

[Function] Sets the call frequency for the server function (ADXT_ExecServer function).

This function can be used to control the amount of decoding performed by each server function.

The default setting is 60 (V-Sync).

[Remarks] Normally, this function is not required because the server function is called by the V-Sync interrupt.

Title	Function	Function Name	No
Function specification	Server function (update internal status)	ADXT_ExecServer	6.21

[Format] void ADXT_ExecServer(void);

[Input] adxt : ADXT handle

[Output] None

[Function value] None

[Function] Updates the internal status of the library and performs decoding processing.

[Remarks] Must be called with each V-Sync.

Because the ADXT_Init function registers V-Sync interrupt processing, the user does not need to call this function.

Title	Function	Function Name	No
Function specification	Get error code	ADXT_GetErrCode	6.22

[Format] long ADXT_GetErrCode(ADXT adxt);

[Input] adxt : ADXT handle

[Output] None

[Function value] Error code

[Function] Gets error code.

This value is held until cleared by the ADXT_ClearErrCode function.

Error code table

Mode definition	Constant	Error condition
ADXT_ERR_OK	0	Normal
ADXT_ERR_SHRTBUF	1	No data supplied to input buffer GD read error has occurred
ADXT_ERR_SNDBLK	2	Sound block error Sound block has stopped.

Title	Function	Function Name	No
Function specification	Clear error code	ADXT_ClearErrCode	6.23

[Format] void ADXT_ClearErrCode(ADXT adxt);

[Input] adxt : ADXT handle

[Output] None

[Function value] None

[Function] Clears error code.

[Example]

```
/* 44 kHz mono data 1 stream playback */
#define WKSIZE                                ADXT_CALC_WORK(1, ADXT_PLY_STM, 1,
                                                44100)

char *work[WKSIZE];                          /* work area */
ADXT adxt;                                   /* ADXT handle */
long errcode;                                /* error code */

adxt = ADXT_Create(1, work, WKSIZE);          /* create ADXT handle */
ADXT_StartFname(adxt, \isrf01.adx\);         /* start playback */
for (;;) {
    errcode = ADXT_GetErrCode(adxt);
    if ( errcode != ADXT_ERR_OK ) {

                                                ADXT_ClearErrCode(adxt);
                                                ADXT_Stop(adxt);

    }
}
```

Title	Function	Function Name	No
Function specification	Set auto error recovery	ADXT_SetAutoRcvr	6.24

[Format] `void ADXT_SetAutoRcvr(ADXT adxt, long mode);`

[Input] `adxt` : ADXT handle

`mode` : error recovery mode

(`ADXT_RMODE_NOACT` , `ADXT_RMODE_STOP` , `ADXT_RMODE_REPLAY`)

[Output] None

[Function value] None

[Function] Sets the error recovery mode.

When automatic error recovery is active, the following action is carried out approx. 1 second after an error has occurred.

[Remarks] The default setting is `ADXT_RMODE_STOP`.

Mode definition	Constant	Error recovery processing
<code>ADXT_RMODE_NOACT</code>	0	No error recovery
<code>ADXT_RMODE_STOP</code>	1	Automatic stop. Mode changes to <code>ADXT_STAT_STOP</code> .
<code>ADXT_RMODE_REPLAY</code>	2	When data supply from CD has stopped, perform playback from start of file. For other errors, change to stop mode.

[Example]

```
/* 44 kHz stereo data and 22 kHz mono data playback */
#define WKSIZ44      ADXT_CALC_WORK(2, ADXT_PLY_STM, 2, 44100)
#define WKSIZ22      ADXT_CALC_WORK(1, ADXT_PLY_STM, 2, 22050)

char *work44[WKSIZ44];
char *work22[WKSIZ22];
ADXT adxt;
ADXT adxt2;
long pan;

/* work area (for BGM) */
/* work area (for dialog) */
/* ADXT handle(for BGM) */
/* ADXT handle(for dialog) */
/* current panpot setting */

adxt = ADXT_Create(2, work44, WKSIZ44);
ADXT_SetAutoRcvr(adxt,ADXT_RMODE_REPLAY);
/* for BGM ADXT handle */
/* auto- repeat mode */
adxt = ADXT_Create(1, work22, WKSIZ22);
ADXT_SetAutoRcvr(adxt,ADXT_RMODE_STOP);
/* ADXT handle for dialog */
/* auto stop mode */
```

```
      :
      :
ADXT_StartFname(adxt, iBGM.ADXi);           /* start BGM playback */
      :
      :
ADXT_StartFname(adxt2, iSRF01.ADXi);        /* start dialog playback */
      :
      :
```

Title	Function	Function Name	No
Function specification	Set error callback function	ADXT_EntryErrFunc	6.25

[Format] void ADXT_EntryErrFunc(void
(*func)(void *obj, char *msg), void *obj);

[Input] adxt : ADXT handle

func : user callback function

obj : 1st argument of callback function

[Output] None

[Function value] None

[Function] Registers an error callback function.

When an error occurs, the registered function is called in the following format.

```
(*func)(*obj, char *msg);
```

The 1st argument (obj) of this function becomes the 2nd argument of the ADXT_EntryErrFunc function. The second argument (msg) is the error message. Because this function is intended for debugging purposes, it should be replaced with a do-nothing function during mastering.

[Example]

```
/* error callback function */
void user_adx_error(void *obj, char *msg)
{
    /* When an error occurs, this function is called. */
    /* The error message is handed to msg. */
    /* msg is stored in R5 register and can be checked */
    /* by dumping the R5 register address. Because this */
    /* function may be called by the V-sync interrupt, */
    /* Ninja functions should not be used. */
    /* Before releasing the application, this function */
    /* should be set to return without doing anything. */

    for (;;) /* take out in release version */

}

/* main function */
void main(void)
{
    /* initialize ADX library */
    ADXT_Init();
    /* register error callback function */
    ADXT_EntryErrFunc(user_adx_error, NULL);
    :
    :
}
```

Title	Function	Function Name	No
Function specification	Playback pause and resume	ADXT_Pause	6.26

[Format]	void ADXT_Pasue(ADXT adxt, long sw);		
[Input]	adxt	:	ADXT handle
	sw	:	pause switch (1 = pause, 0 = resume)
[Output]	None		
[Function value]	None		
[Function]	Carries out playback pause and resume. When the switch is set to 1, play pauses; when the switch is set to 0, play resumes. When pause is set when playback is in the STOP state, sound cannot be played back even if the start function is run. When pause is set in the PLAYING state, playback pauses. By releasing pause in the PLAYING state, sound instantly plays back.		

[Example]

```
(1)                                Normal pause
                                   /* Playback for only one stream of 44Khz stereo data */
                                   #define WKSIZe                ADXT_CALC_WORK(2,
ADXT_PLY_STM, 1, 44100)

char *work[WKSIZe];                /* work area */
ADXT adxt;                        /* ADXT handle */
long pause_flag;                  /* pause flag */

                                   adxt = ADXT_Create(2, work, WKSIZe);

/* ADXT handle creation */

                                   pause_flag = 0;
                                   ADXT_StartFname(adxt,imusic.adxî);

/* start playback */

                                   for (;;) {

if ( per->on & PDD_DGT_TA ) {

                                   if (pause_flag == 0 )/* pause */

ADXT_Pause(adxt, pause_flag = 1);

                                   else /* resume playback */

ADXT_Pause(adxt, pause_flag = 0);

                                   }

                                   }

(2)                                Immediately playing back sound
data on a CD
adxt = ADXT_Create(2, work, WKSIZe); /* ADXT handle creation */
ADXT_Pause(adxt, 1);                /* pause */
ADXT_StartFname(adxt, imusic.adxî); /* playback start */
is PLAYING */                       /* Sound is not played back, state

                                   for (;;) {

/* When the A button is pressed, sound is immediately output */
if ( per->on & PDD_DGT_TA ) {        ADXT_Pause(adxt, 0);
}
```

20.8 ADX Library

Files Required for Using To use the ADX library, the following files are required.

< ADX library>

CRI_ADXT.H	:	ADX playback library header file
CRI_ADXF.H	:	ADX file system library header file
CRI_ADXS.LIB	:	SHINOBI version library file

<ADX tools>

ADXENCD.EXE	:	ADX encoder
ADXCAT.EXE	:	Sound data linking tool
AFSLNK.EXE	:	ADX file system linking tool
AX.EXE	:	One-pass encoder utility program

20.9 Sound Data Creation Precautions

While playing sound data with the ADX playback library, sound effects and MIDI sequence data in the sound block can also be played back. In the Shinobi environment, the ADX playback library realizes the PCM stream function of the sound library. For this purpose, a PCM buffer is allocated in 4040H blocks from the end of the sound RAM. Therefore the RAM buffer size must be calculated from the number of channels using ADXT handles, and a multi-unit file must be created to prevent overlap in this area.

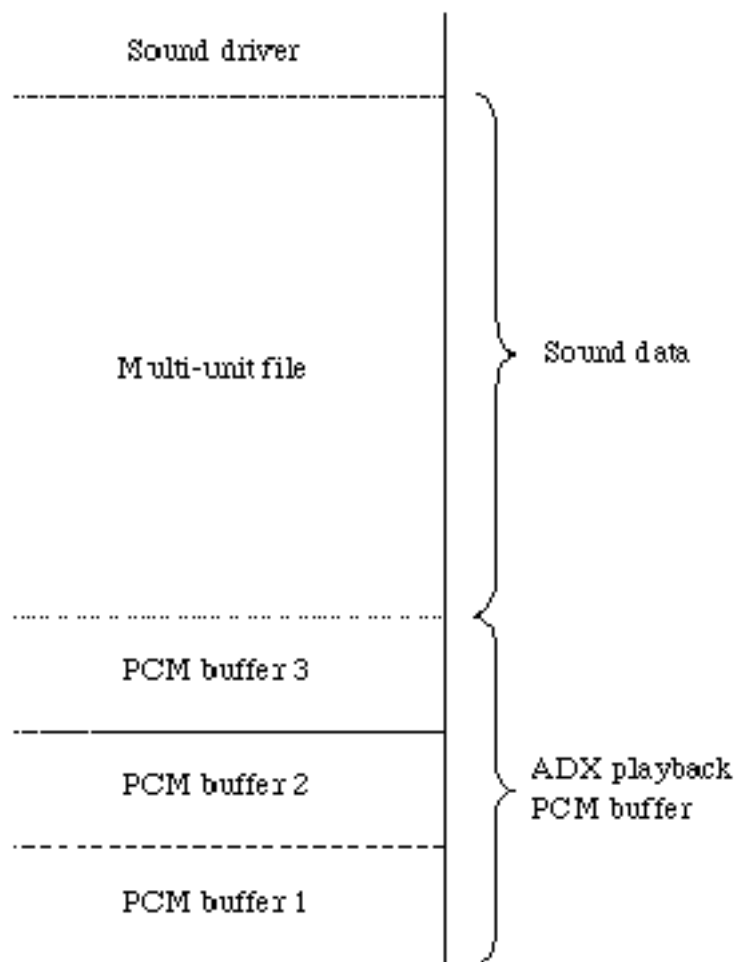


Figure 2.11 Sound RAM allocation map

20.10 Increasing Data Reading Speed

The ADX file system can read data while simultaneously playing sound from the CD. However, in the default setting, data reloading begins at the input buffer 85% threshold. When several game data files are read, this will result in the sound data and game data being read alternately rather than simultaneously. The `ADXT_SetReloadSct` function can be used to adjust the start volume where sound data are reloaded, to speed up game data loading. Reloading should occur at the equivalent of about 1 second of sound data. For example, the data rate of 44.1 kHz stereo material is about 50 Kbyte per second, therefore 25 sectors should be specified. By making the input buffer large, the reload interval can also be made larger, which further speeds up data loading. Because this is a preliminary function, `ADXT_SetReloadSct` is not included in the reference manual. In future, data loading will be speeded up also without these measures.


```
/* 44 kHz stereo data playback */
#define WKSIZE                                ADXT_CALC_WORK(2,
ADXT_PLY_STM, 3, 44100)

char work[WKSIZE];                            /* work area */
ADXT adxt;                                    /* ADXT handle */
ADXF adxf;                                    /* ADXF handle */

adxt = ADXT_Create(2, work, WKSIZE);           /* create ADXT handle */
ADXT_SetReloadSct(adxt, 25);                   /* set reload volume */
:
ADXT_StartFname(adxt, "BGM.ADX");              /* start sound playback */
:
adxf = ADXF_Open("GAMEDAT1.BIN", NULL);        /* open file */
ADXF_ReadNw32(adxf, nsct, buf);                /* read data */
while ( ADXF_GetStat(adxf) != ADXF_STAT_READEND )
;
ADXF_Close(adxf);                             /* close file */
:
(If GAMEDAT1 is read within 2 seconds, there is no reloading of sound data.)
:
adxf = ADXF_Open("GAMEDAT2.BIN", NULL);        /* open file */
ADXF_ReadNw32(adxf, nsct, buf);                /* read data */
while ( ADXF_GetStat(adxf) != ADXF_STAT_READEND )
;
ADXF_Close(adxf);                             /* close file */
```

Note: By increasing the maximum number of CD streams (3rd argument of ADXT_CALC_WORK), the input buffer can be increased. With each stream, an increase equivalent to one second can be achieved. In situations where BGM and dialog are played back together but dialog will never be played during data readout, the buffer set aside for dialog can be allocated to the data read-in process.

20.11 How to Reduce Sound Seek

By default, the ADX playback library carries out buffering control focused on response speed. When 85% of the buffer is allocated to the input buffer, data read is generated. Therefore, when data location positions are separated on the CD, sound seek frequency occurs. Sound seek can be reduced by the function ADXT_SetReloadSct.

For example, when playing back BGM and dialog, the number of times seek occurs can be lowered by making the interval to read dialog larger. When playing back 22Khz monoaural dialog, the bit rate is about 12.5 KB/sec. Therefore, it is possible to concurrently play back 2 streams if you have buffering in one-second intervals, so it is better to have 7 sectors for reloading on the dialog side. Also, by specifying larger numbers of concurrent playback streams, the input buffer can be made larger when calculating the work area for dialog. In this way, you can also make the seek intervals larger.

```
/* Work area for concurrently playing back 2 streams in 44Khz stereo */
#define WKSIZ44S ADXT_CALC_WORK(2, ADXT_PLY_STM, 2, 44100)
/* Work area for concurrently playing back 3 streams in 22Khz monoaural */
#define WKSIZ22M ADXT_CALC_WORK(1, ADXT_PLY_STM, 3, 22050)
/* Originally, only 2 streams could be played back, but now 3 streams are possible */
/* Increasing the input buffer is possible */

char wk44[WKSIZ44S], wk22[WKSIZ22M];          /* work area */
ADXT adxt_bgm, adxt_srv;                      /* ADXT handle */

adxt_bgm = ADXT_Create(2, wk44, WKSIZ44S);    /* ADXT handle creation */
adxt_srf = ADXT_Create(1, wk22, WKSIZ22M);    /* ADXT handle creation */
ADXT_SetReloadSct(adxt_srf, 7);               /* setting the reload amount */
:
ADXT_StartFname(adxt_bgm, "BGM.ADX");         /* start sound playback */
:
if (event occurs)
ADXT_StartFname(adxt_srf, "SRF001.ADX");
```

20.12 Concurrent playback with MPEG Sofdec

Concurrent playback with MPEG Sofdec is possible with the ADX playback library. A seamless transition from a game scene to a movie scene is possible by non-synchronously playing back a movie while playing a CD stream of BGM. However, to do this, you must increase the buffer size for movies. When creating a Sofdec handle, set the maximum bit stream size to a value larger than the actual stream size. As a standard, specify about 1.5x the value. This is done because seek time is expected to be about 1 second, so it can be reduced depending on the way data is located. Also, raise this value if the movie breaks.

```
/* Work area for concurrently playing back 2 streams in 44Khz stereo */
#define WKSIZ44S                                ADXT_CALC_WORK(2, ADXT_PLY_STM, 2,
44100)
/* 2 streams are BGM and movie */
char wk44[WKSIZ44S];                            /* work area */

MWPLY ply;                                       /* MWPLY handle */
ADXT adxt_bgm;                                  /* ADXT handle */

adxt_bgm = ADXT_Create(2, wk44, WKSIZ44S);      /* ADXT handle creation */
ADXT_SetReloadSct(adxt_bgm, 25);               /* for reducing sound seek */

/* MWPLY handle creation */
memset(&cprm, 0, sizeof(cprm));                 /* necessary for setting the
reserve member to 0 */
cprm.opt.first_hpel = ON;
cprm.ftype = MWD_PLY_FTYPE_MPV;
cprm.dtype = MWD_PLY_DTYPE_AUTO;
cprm.max_bps = 900*1024*8;                      /* usually set to 1.5x of 600Kbyte/
sec */
cprm.max_width = 320;
cprm.max_height = 240;
cprm.nfrm_pool_wk = 3;
cprm.wksize = mwPlyCalcWorkSofdec(
cprm.ftype,
cprm.max_bps,
cprm.max_width,
cprm.max_height,
cprm.nfrm_pool_wk);
cprm.work = syMalloc(cprm.wksize);
ply = mwPlyCreateSofdec(&cprm);

ADXT_StartFname(adxt_bgm, iBGM.ADXi);          /* start sound playback */
:
if (event occurs)
mwPlyStartFname(ply, iSCENE01.M1Vi);          /* play movie */
```

20.13 Joint Use with GDFS/NINJA Library

Because the ADX library uses the Shinobi library, joint use with GDFS is in principle possible. However, when GDFS takes over the GD-ROM pickup, sound data cannot be read, causing sound playback to be interrupted. Therefore it is recommended to use the ADX file system. When functions such as `njLoadTexture` make use of GDFS unavoidable, the `ADXT_IsIbufSafety` function or `ADXT_GetNumSctIbuf` function should be used to verify the input buffer status before using GDFS.

The `ADXT_IsIbufSafety` function returns 1 when the number of data in the input buffer exceeds the reload start sector number.

The `ADXT_GetNumSctIbuf` function returns the number of sectors in the input buffer. Determine the time until sound cutoff from the bit rate of the sound stream to make the setting.

```
/* 44 kHz stereo data playback */
#define WKSIZE                                ADXT_CALC_WORK(2, ADXT_PLY_STM, 3, 44100)

char *work[WKSIZE];                          /* work area */
ADXT adxt;                                  /* ADXT handle */
ADXF adxf;                                  /* ADXF handle */

adxt = ADXT_Create(2, work, WKSIZE);          /* create ADXT handle */
:
ADXT_StartFname(adxt, "BGM.ADX");             /* start sound playback */
:
while ( ! ADXT_IsIbufSafety(adxt) )
;
njLoadTexture(&tlist);
```

20.14 Sample Programs

This section lists the main sample programs supplied with this library.

- 1) `adxsm01`: simple CD stream playback
'`smpadx01.c`' is a simple program for playing an sound data file on CD. The file '`sample.adx`' is played.
- 2) `adxsm02`: memory playback & memory index playback
'`smpadx02.c`' is a program that reads the files '`sample.adx`' and '`smp1_mem.acx`' from CD into work RAM and performs memory playback and memory index playback using the pad.
Button A : memory playback
Any other button : memory index playback
- 3) `adxsm03`: CD index playback
'`smpadx03.c`' reads partition information from '`pat01.afs`' on the CD and performs CD index playback using the pad. Four streams are played back simultaneously.
- 4) `adxsm04`: ADX file system
'`smpadx04.c`' reads partition information from '`pat02.afs`' on the CD and reads a file in the partition using the pad.

A. Gun Controller Appendix

A.1 Gun Controller Sample #1 Single Shot

A.1.1 Overview

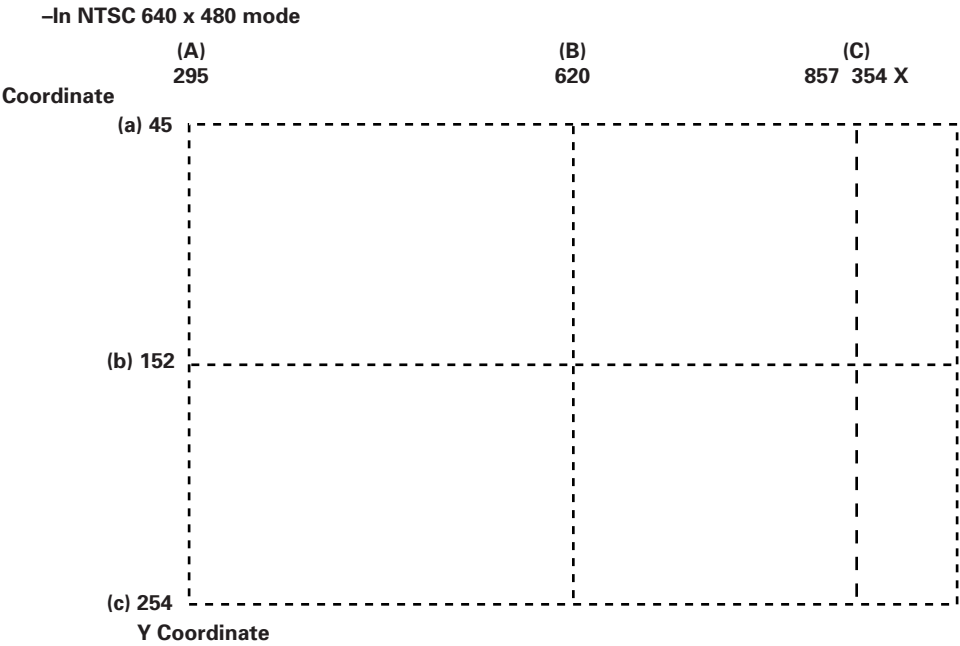
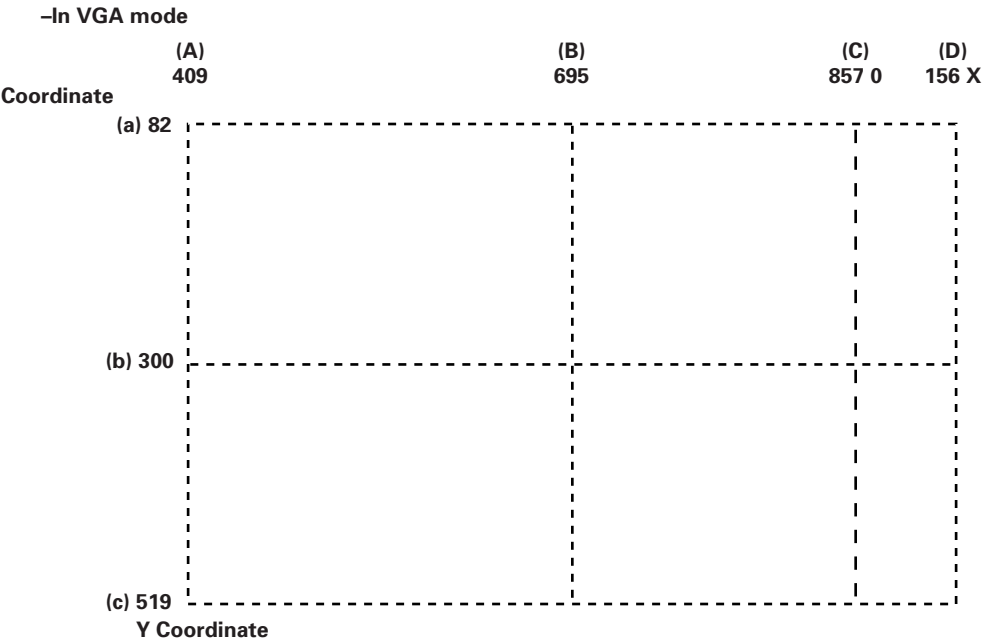
The gun function in the controller library simply gets the gun coordinates. When using in actual applications, coordinate conversion, compensation and off-screen decisions are necessary. This sample is based on the method used in an actual application

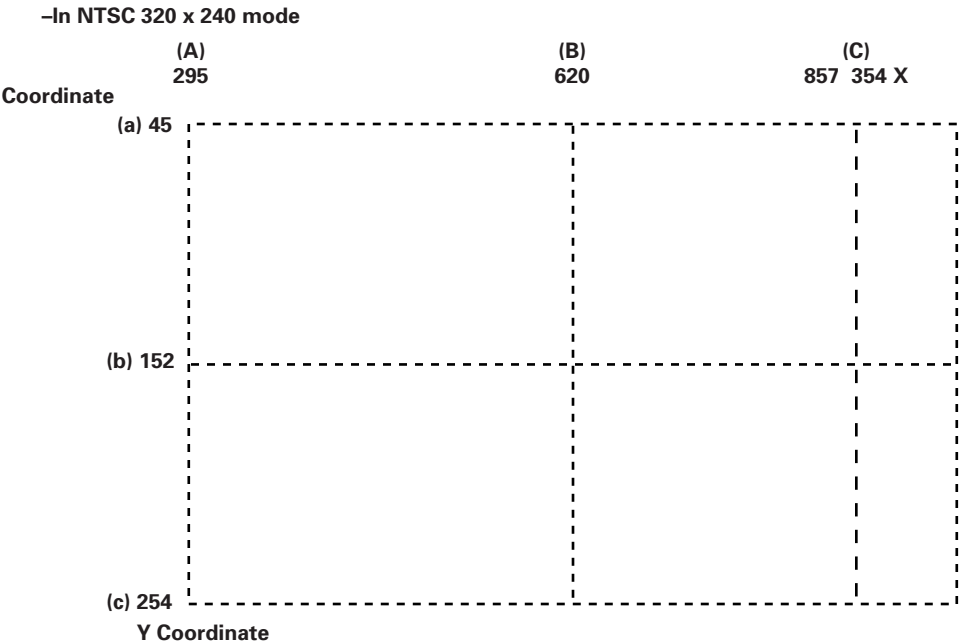
A.1.2 Details of Coordinate Compensation

Coordinates obtained from the gun controller (raw data) have the following values. These are approximate values, and be advised that errors or blurring may occur in actual use.

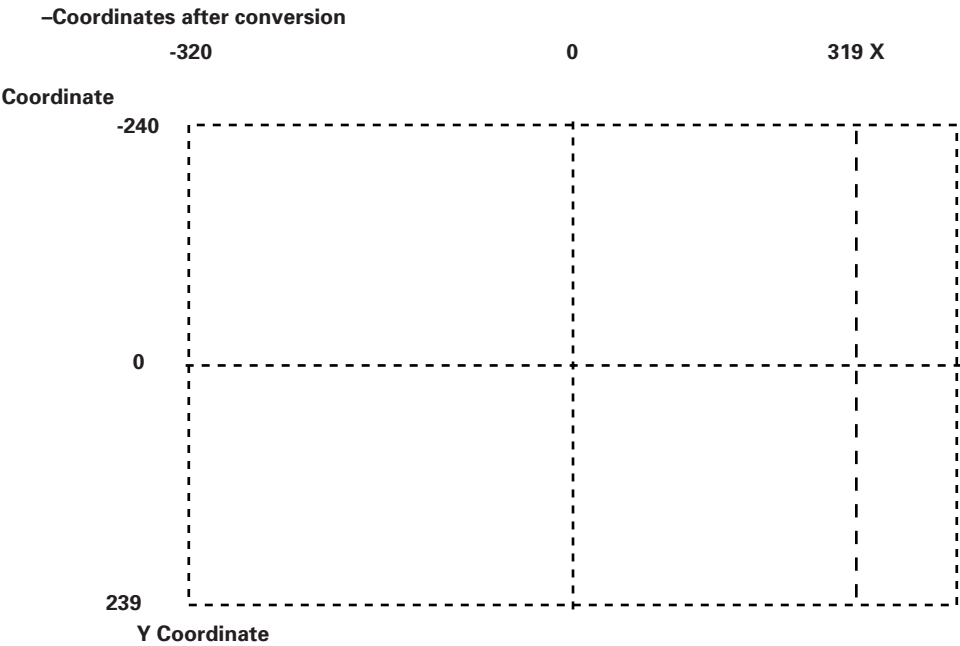
As you can see on the graph below, on the right side of the screen there is a point where the value returns to 0 after the x coordinate value has increased to about 857. Coordinates beyond this point on the right side are either ignored (go offscreen), or the effective value is taken by adding the offset value. This sample follows the latter method.

When in NTSC screen mode, the screen edges don't always show on the screen (overscan) due to the characteristics of TV screens. How much of the periphery is displayed on the screen differs from TV to TV, but be aware that your actual results may differ from the values shown on the chart.





The raw data shown above are converted to the following coordinates by direct compensation. With NTSC, compensation results in a slight inward offset.



When the trigger is pulled with the gun aimed offscreen, the hardware can't get the gun coordinates and instead uses the value from the previous time the trigger was pulled. Using this value, the hardware compares the coordinates gotten both times to judge if the gun was shot offscreen.

With this method, a shot is judged to be offscreen even if it was shot at exactly the same coordinates as the previous time, but since the chances of that happening are extremely rare, this is considered to be the best method to determine if a gun was shot offscreen.

A.1.3 Fine Adjustments with Coordinate Compensation

Fine-tuning by coordinate compensation is possible in the source file `gun.c` by changing the values of the structure arrays `CalibTblX` and `CalibTblY`. For example, when a user test shoots a gun on the option screen in the application, the test shot coordinates can be used in gun calibration by reflecting the raw data of coordinates here.

A.2 Gun Controller Sample # Continuous Shooting

–In the gun mode, the library gets coordinates when it detects the gun trigger has been pulled. However, in certain applications, there are times when you want to detect coordinates even when the gun trigger has not been pulled: for example, during continuous shooting from a machine gun.

–In this sample, the B button on the gun controller toggles between single shot or continuous shooting when it is pressed.

B. Peripheral Data Acquisition Function Group Gun Controller

B.1 Overview

This function group presents the functions for handling the gun controller (tentative name).

B.1.1 Gun Controller Specifications

The Dreamcast Gun uses the same method that Saturn previously used for detecting scan lines on a TV screen.

B.2 Gun Mode

To use the gun, it is necessary to set the special mode "Gun Mode" in the controller library. This setting is required because the access system for the gun is very different from the usual kinds of controllers and keyboard or VM peripherals. Gun mode has a number of limitations.

B.2.1 Limitations with Extension Peripherals

In gun mode, there are many limitations to accessing extension peripherals.

Memory Card, starting with VM (Visual Memory)

The memory card cannot be accessed at all. Because of this, the following libraries can't be used: LCD library, backup library, peripheral timer library

Be sure to exit the backup library using the `buExit()` function. Even if the `buExit()` function is used, mount processing fails and the memory card cannot be recognized.

Puru Puru Pack (Vibration Device)

There are no significant limitations to the use of this device. Vibration can be implemented with Puru Puru Pack connected to the gun by using the vibration library.

SIP

Currently use with SIP is unconfirmed, but to avoid possible problems, it is better not to use it. Because the backup library cannot be accessed, use the following method to access VM from the application:

Note: Operate the gun mode as usual, and set the regular mode only for the save and load screens.

In this setup, initialization of the backup library is carried out at the level in which the save screen is opened and finishes when the screen is exited.

B.2.2 Screen Flash

In gun mode, the library internally makes an automatic flash of the screen at 1INT to detect the gun coordinates when the gun trigger is pulled. The flash color is white (0x00c0c0c0) as a default. When a flash does not occur, it means the gun coordinates have not been properly detected.

B.2.3 Gun Priority Order

In hardware specifications, the gun can only get data from one port in 1INT. In the library, data is gotten with port A having the highest priority. Therefore, when more than one gun trigger is simultaneously pulled, detection starts from the lowest port number first.

B.3 Function Specifications

B.3.1 Function List

pdGunEnter	Enters gun mode
pdGunLeave	Exits gun mode
pdGunGetLatchedPort	Gets control port where coordinates were detected
pdGunGetPosition	Gets gun coordinates
pdGunSetFlashColor	Sets screen flash color
pdGunSetCallback	Registers call back for trigger information setting
pdGunSetTrigger	Sets trigger information

B.3.2 Function API

pdGunEnter

Enters gun mode.

Format

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

Input

portbit Port set to Gun Mde

Output

None

Return value

None

Function

Gets the controller library to gun mode

Reference

```
pdGunLeave( )
```

Example

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

pdGunLeave

Exits gun mode.

Format

```
void pdGunLeave(void)
```

Input

None

Output

None

Return value

None

Function

Sets controller library to regular mode

Reference

```
pdGunEnter()
```

Example

```
pdGunLeave();
```

pdGunGetLatchedPort

Gets control ports where coordinates were detected.

Format

```
Sint32 pdGunGetLatchedPort(void)
```

Input

None

Output

None

Return value

Port number that detected gun coordinates (PDD_PORT_A0/B0/C0/D0) If gun coordinates are not detected -1

Function

Returns the number of the port that detected gun coordinates.

Coordinate detection is carried out when the trigger is pulled or when dummy trigger information is set by the pdGunSetTrigger() function.

Reference

Example

```
Sint32 gunno, x, y;
```

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

pdGunGetPosition

Gets gun coordinates.

Format

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

Input

x, y Pointer to get gun coordinates

Output

x, y Gun coordinates

Return Value

None

Function

Gets gun coordinates.

Coordinate detection is carried out when the trigger is pulled or when dummy trigger information is set by the function `pdGunSetTrigger()`.

Reference

Example

Refer to the `pdGunGetLatchedPort()` example.

pdGunSetFlashColor

Sets screen flash color.

Format

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

Input

color screen flash color

Output

None

Return Value

None

Function

Sets the color the screen flashes when the trigger is pulled.

- When 0 is set, screen flash is not carried out. This can be set when the screen is designed to be comparatively bright, and coordinate detection can be done without making a flash.
- Detecting coordinates when the screen is dark (has low brightness) may result in coordinates being incorrectly detected, or detection not being carried out at all.
- The initial setting for flash color is white (0x00c0c0c0).
- Color setting is based on the border color setting in the graphics library.
- Brightness levels in VGA screen mode are low, so set a slightly brighter color in VGA mode.

Reference

Example

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


pdGunSetCallback

Registers call back for trigger information setting.

Format

```
void pdGunSetCallback(Sint32 (*func)(void))
```

Input

func call back function address

Output

None

Return Value

None

Function

Registers the call back function for trigger setting

- When the call back function returns FALSE, coordinate detection cannot be carried out even when the actual trigger is pulled. In this case, set the trigger with the `pdGunSetTrigger()` function mentioned below, as necessary. When the call back function returns TRUE, coordinate detection can be carried out normally from the trigger.

Reference

`pdGunSetTrigger()`

Example

```
static Sint32 callback(void)
{
    static Uint32 count = 0;

    if ((count & 3) == 0) {
        pdGunSetTrigger(PDD_PORT_A0);
    }
    count++;

    return TRUE;
}
```

pdGunSetTrigger

Sets trigger information.

Format

```
void pdGunSetTriger(Uint32 port)
```

Input

port Port number (PDD_PORT_A0/B0/C0/D0)

Output

None

Return Value

None

Function

Pulling the trigger is simulated by the software and coordinates are gotten.

- This function is invalid if it is used outside of the callback function for trigger setting.
Be sure to register the callback function in `pdGunSetCallback()` and call it up in that function.
- This function is used for continuous shooting in the software even if the actual gun trigger is not pulled.

Reference

`pdGunSetCallback()`

Example

Refer to the example of `pdGunSetCallback()`

Dreamcast Menu Screens

Table of Contents

1. Main Menu	EMS-1
The Main Menu	EMS-1
2. Memory Card Selection Screen	EMS-3
Memory Card Selection Screen	EMS-3
3. Save Data	EMS-5
File Menu Screen	EMS-5
4. Game Names and Sorting	EMS-11
Game Names and Character Maps	EMS-11
Examples of Japanese Language Titles	EMS-12
5. Format Settings	EMS-13
Body Color	EMS-13
Formatting Visual Memory	EMS-14
6. CD Playback Screen	EMS-17
CD Playback Screen	EMS-17
Changing the CD/GD Texture	EMS-18
7. Setting Screen	EMS-19
Set Icon from the Settings Screen	EMS-19

1. Main Menu

1.1 The Main Menu

This screen appears after the opening animation of the BootROM is completed. It is called the Main Menu screen.



Figure 1.1 *The Main Menu*

Four icons are displayed on the main menu: the "Play icon", "Sound icon", "File icon" and "Set icon".

Icon	Function
Play	Starts Dreamcast disk
Sound	Plays CD and GD audio disks
File	Manages memory card
Set	User configuration

Several of these functions can be operated from the disk or memory card.

Dreamcast Menu Screens

The following chapters explain the functions and their format.

Note: In development versions of the BootROM, version information is displayed above the date and time information at the top right of the screen.

2. Memory Card Selection Screen

2.1 Memory Card Selection Screen

Selecting the “File icon” from the main menu brings up the following, “Memory Card Selection Screen”.



Figure 2.2 Memory Card Selection Screen

On this example screen, two memory cards are inserted in control port A.

In expansion slot 1, Visual Memory with “Atsumete Godzilla Kaiju Daishugo”, is inserted. In expansion slot 2, a Visual Memory Unit without an application on it has been inserted.

The Godzilla icon that is displayed is called the “Volume Icon”, and in Visual Memory applications, data must be prepared separately for this icon from the program data.

Dreamcast Menu Screens

This data has the file name ICONDATA_VMS and must have the following format.

Table 2.2

	+0	+1	+2	+3	+4	+5	+6	+7	+8	+9	+A	+B	+C	+D	+E	+F
+0000	VM comment data															
+0010	Monochrome icon data offset				Color icon data offset				Reserved							
+0020	Monochrome icon data															
+00A0	Color icon palette data															
+00C0	Color icon pixel data															
+02B0																

The meaning of each term is listed below.

Table 2.3

Term	Meaning
VM comment data	File name displayed on the file menu screen.
Monochrome icon data offset	Byte offset up to the location of where monochrome icon data is stored in a file. (00000020h is entered)
Color icon data offset	Byte offset from the file start of the location where color icon palette data is stored. (000000A0h is entered)
Reserved	(Fill with 00h)
Monochrome icon data	The bitmap that is displayed on the LCD of Visual Memory when the menu is displayed; information is in 32x32 dots, with 1 dot = 1 bit and 8 bits to 1 byte; if the bit is raised (if it is 1), the color is black and if it is not raised (it is 0), the color is white; pixel information is from left to right, proceeding from the upper bit to the lower bit.
Color icon palette data	A total of 16 colors are stored in ARGB4444 color data for palette data of the volume icon.
Color icon pixel data	Volume icon pixel data is displayed in palette format. 4 bits is 1 dot, and pixel data is expressed with the upper 4 bits as left, the lower 4 bits as right.
Note: All data must be expressed in Little-Endian format.	

3. Save Data

3.1 File Menu Screen

When a memory card is selected and the A button is pressed on the memory card selection screen, the following “File Menu Screen” appears.

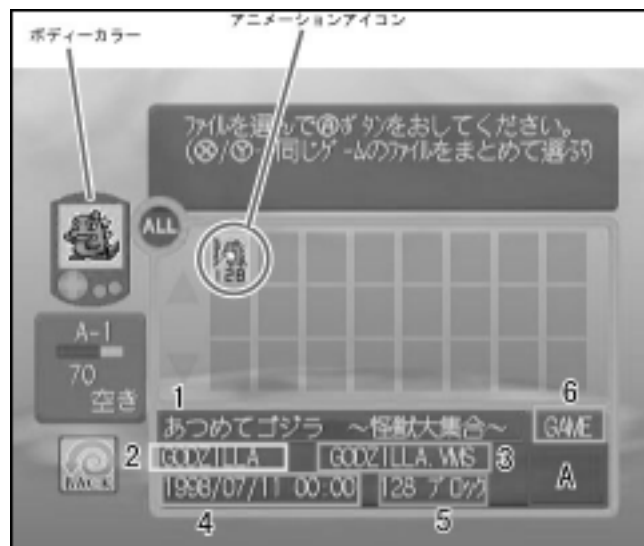


Figure 3.1 The File Menu screen

This screen shows a list of files saved on the selected memory card, as well as details about the selected file. The points on the screen indicated by numbers and letters are called by the following names.

Table 3.1

Number/Letter	Name
1	Game title (BootROM comment)
2	File name
3	Comment (VM comment)
4	Date/Time
5	Number of used blocks
6	GAME/DATA
A	Visual comment

Visual comment displays information of a given game scene that has been miniaturized and saved, and can be used to show the progress conditions that are being saved in visual representation.

The icon to show saved data can hold animation information. If animation information is included in the file icon displayed on the file menu screen, that icon is animated during display. For the above-written scene, the animation of Godzilla breathing fire is displayed.

Next, what kind of file format data other than volume icon data have when saved on Visual Memory is explained.

Besides data or programs, saved files must contain an information fork.

The information fork has the same structure for Visual Memory programs and data, but the position in the file is different.

Visual Memory program files contain the reset vector and other resource information, followed by the information fork, followed by the actual program. Other files such as saved data contain the information fork followed by the actual data.

Table 3.2

Visual Memory program		Data
+0000h	Visual Memory resources (vector information table etc.)	_____
+0200h	Information fork	Information fork _____
+xxxxh	Visual Memory program data	Data _____

The information fork is divided into two sections. One is required and the other is optional, depending on the functions that are used.

The format of the data is as follows.

Table 3.3 - Required section

+0		+1	+2	+3	+4	+5	+6	+7	+8	+9	+A	+B	+C	+D	+E	+F
+0000	VM comment data															
+0010	Game title data (BootROM comment)															
+0020																
+0030	Game name for sorting															
+0040	Number of icons	Animation speed	Visual type	CRC	Saved data size				(*)							
+0050	(*)															
+0060	Icon palette data															
+0070																
+0080	Icon no. 1 pixel data															
+0270																

Table 3.4 - Function-dependent section

	+0	+1	+2	+3	+4	+5	+6	+7	+8	+9	+A	+B	+C
+0280	Icon no. 2 pixel data												
+xxxx													
+xxxx	Icon no. 3 pixel data												
+xxxx													
+xxxx	Palette and pixel data for visual comment												
+xxxx													
Note: All data must be expressed in Little-Endian format.													

Dreamcast Menu Screens

The terms used in the above table have the following meaning.

VM comment data

- The file name to be displayed on the Visual Memory file management screen. The name uses ASCII characters and can be up to 16 characters long. For information on allowable characters, see the separate table.

Game title data (BootROM comment)

- The game title displayed on the main menu file management screen. Up to 32 ASCII characters.

Game name for sorting

- Character string that is used for sorting the file.
- Uses special codes that are explained below.

Number of icons

- The number of icons used for animation (1 - 3).

Animation speed

- The rate at which animation icons are switched. Taking the value specified here as n, the switching interval is n/30 seconds. (The switching sequence is 1 -> 2 -> 3 -> 1 ->...)

Visual type

- Specifies the type of visual comment.
- 0 - No visual comment is used.

Table 3.5 Type A used

1	Size	72 x 56 dots
	Color format	Direct color (ARGB4444)
	Palette	Not used
	Pixel data	For 1 dot, 2 bytes specify information from top left to bottom right
	Number of blocks to be used as visual comment data	16

Table 3.6 Type B used

2	Size	72 x 56 dots
	Color format	256 color palette (8 bit)
	Palette	ARGB4444 format
	Pixel data	For 1 dot, 1 byte specifies information from top left to bottom right
	Number of blocks to be used as visual comment data	8

Table 3.7 Type C used

3	Size	72 x 56 dots
	Color format	16 color palette (4 bit)
	Palette	ARGB4444 format
	Pixel data	For 2 dots, 1 byte specifies information from top left to bottom right
	Number of blocks to be used as visual comment data	4

CRC

- Stores CRC data. When the backup utility functions are used, CRC data are generated and checked automatically. User applications are free to use or disregard CRC data.

Save Data Size

- Specifies the block size of Dreamcast application save data (except for information fork) in numbers of bytes.
- For executable files on Visual Memory, specify the number of bytes for the whole file.

Note: This data is not referred to on the BootROM file management screen.

Icon palette data

- Stores the icon palette data for animated icons. The ARGB4444 format with 16 colors (4 bits) is used.
- Multiple icons (for animation) use the same palette.

Icon no. 1 pixel data

- Defines the file icon in 4-bit palette format.
- The pixel size is 32 dots x 32 dots.

This only needs to be set or non-animated icons. For animated icons, the following items also need to be set.

Icon no. 2 pixel data

- Used for animated icons with two or three images.
- Specifications are the same as for icon no. 1.

Icon no. 3 pixel data

- Used for animated icons with three images.
- Specifications are the same as for icon no. 1.

Visual comment palette and pixel data

- Used for adding a visual comment. The data type depends on the visual comment type (type A - C). When a palette is used, it must be defined first, followed by the pixel data definition.

Because visual comments require a considerable amount of backup space, the decision whether to use comments and which comment type to use should be made after referring to the guidelines for writing Visual Memory software.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0			SP	0	P	`	p					-	タ	ミ		
1			!	1	A	O	a	q				。	ア	チ	ム	
2			"	2	B	R	b	r				「	イ	ツ	メ	
3			#	3	C	S	c	s				」	ウ	テ	モ	
4			\$	4	D	T	d	t				、	エ	ト	ヤ	
5			%	5	E	U	e	u				・	オ	ナ	ユ	
6			&	6	F	V	f	v				ヲ	カ	ニ	ヨ	
7			'	7	G	W	g	w				ァ	キ	ヌ	ヲ	
8			(8	H	X	h	x				ィ	ク	ネ	リ	
9			>	9	I	Y	i	y				ゥ	ケ	ノ	ル	
A			+	:	J	Z	j	z				エ	コ	ハ	レ	
B			+	:	K	[k]				オ	サ	ヒ	ロ	
C			,	<	L	V	l	v				ャ	シ	フ	ワ	
D			-	=	M]	m]				ユ	ス	ヘ	ン	
E			.	>	N	^	n	^				ヨ	ゼ	ホ	^	
F			/	?	O	_	o	_				ッ	ソ	マ	^	

Figure 3.2 Allowable characters for VM comments and save file names
(characters shown in shaded boxes).

4. Game Names and Sorting

4.1 Game Names and Character Maps

You must set a game name for sorting to sort files on the file menu screen.

Only certain characters are allowed for game names, and the length is limited. The name is converted into special codes using a conversion table.

The characters that can be used in game names for sorting are shown below.

No	Chr	Hex	No	Chr	Hex	No	Chr	Hex	No	Chr	Hex	No	Chr	Hex
00	None	00	20	コ	14	40	デ	28	60	ホ	3C	80	キ	50
01	ア	01	21	ゴ	15	41	ト	29	61	ボ	3D	81	エ	51
02	ァ	02	22	サ	16	42	ド	2A	62	ポ	3E	82	ヲ	52
03	イ	03	23	ザ	17	43	ナ	2B	63	マ	3F	83	ン	53
04	ィ	04	24	シ	18	44	ニ	2C	64	ミ	40	84	0	54
05	ウ	05	25	ジ	19	45	ヌ	2D	65	ム	41	85	1	55
06	ゥ	06	26	ス	1A	46	ネ	2E	66	メ	42	86	2	56
07	ゥ	07	27	ズ	1B	47	ノ	2F	67	モ	43	87	3	57
08	エ	08	28	セ	1C	48	ハ	30	68	ヤ	44	88	4	58
09	ェ	09	29	ゼ	1D	49	バ	31	69	ャ	45	89	5	59
10	オ	0A	30	ソ	1E	50	バ	32	70	ユ	46	90	6	5A
11	ォ	0B	31	ゾ	1F	51	ヒ	33	71	ュ	47	91	7	5B
12	カ	0C	32	タ	20	52	ビ	34	72	ヨ	48	92	8	5C
13	ガ	0D	33	ダ	21	53	ピ	35	73	ョ	49	93	9	5D
14	キ	0E	34	チ	22	54	フ	36	74	ラ	4A			
15	ギ	0F	35	ヂ	23	55	ブ	37	75	リ	4B			
16	ク	10	36	ツ	24	56	ブ	38	76	ル	4C			
17	グ	11	37	ヅ	25	57	ヘ	39	77	レ	4D			
18	ケ	12	38	ッ	26	58	ベ	3A	78	ロ	4E			
19	ゲ	13	39	テ	27	59	ベ	3B	79	ワ	4F			

Figure 4.3 Allowable characters in game names for sorting purposes

The maximum length of the file name is 16 characters, using characters from the above table.

Some examples are shown below, along with an explanation of naming conventions.

4.2 Examples of Japanese Language Titles

- Non-Japanese titles should be filled with 00 from the table.
- Applications whose name exceeds 16 characters and which use an abbreviation for the GUI comment should be denoted by their abbreviation.
- The entries 84 - 93 in the table (numerals) are assumed to be used only at the end of a name. A name should *not* start with a numeral.
- For multi-part games which should be kept together when sorting, use similar names with numerals at the end.
- If the first part uses an abbreviated name, the same abbreviation should be used for other parts as well.
- If years are used in a title, the game name should be set while keeping the 4-digit year 2000 in mind.

Ex.

1. "Greatest Nine 98"
2. "Greatest Nine 99"
3. "Greatest Nine 2000"

As is, these would be sorted in the order 3 -> 1 -> 2. To avoid this, use the names

1. "Greatest_Nine_1"
2. "Greatest_Nine_2"
3. "Greatest_Nine_3"

For titles with volume numbering where dual-digit numbers are to be expected, two digits should be used from the outset.

Ex.

"World Literature Collected Works 3" and
"World Literature Collected Works 12" should be named
"World_Lit_03"
"World_Lit_12"

etc.

Also if there are no numbers used in the titles of multi-part works, numbers should be used in the game names to achieve proper sorting.

Once the actual game name has been decided, it must be converted into the dedicated code.

Replace the characters in the created game name for sorting with the numbers shown on the table on the previous page, following the conditions above. 16 characters results in 16 bytes of data, this data should be passed to the library function as the game name for sorting when saving the data.

5. Format Settings

5.1 Body Color

There is an item called Body Color on the file menu screen.

This item is also called the volume data. It can be set when formatting (initializing) the Visual Memory. The body color is set to the default for all Visual Memory cards when shipping. It can only be changed during formatting.

Table 5.4 *The formatting process is as follows (unit is bytes).*

0	1	2	3	4	5 - 32
Initialization flag	Blue	Green	Red	Alpha	Not used

Initialization flag (1 byte)

- 00h No color data (factory default)
- 01h Color data present

Body color data (4 bytes)

- Order is Blue, Green, Red, Alpha (8 bits each).
- For Alpha, 00h means fully transparent. Increasing values cause increasing opaqueness, with FFh being fully opaque.
- For the other colors, 00h means dark, and increasing values cause increasing lightness.
- Specifying 00h for all items results in the default color (white).

5.2 Formatting Visual Memory

When formatting the Visual Memory, there are also other items that can be set.

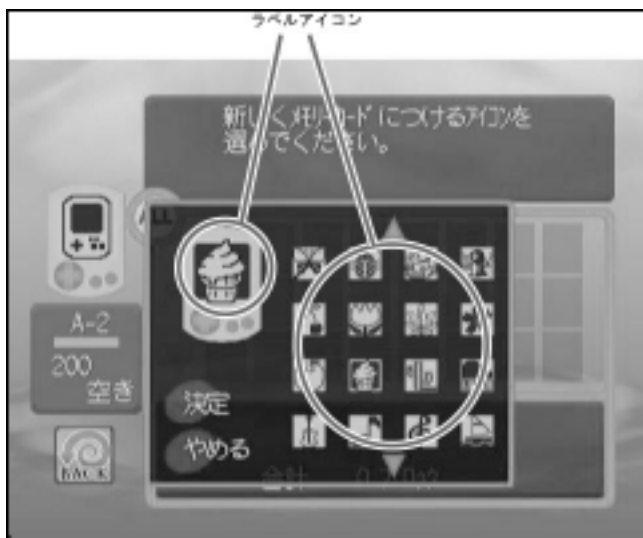


Figure 5.4 *Formatting Visual Memory*

The illustration above shows one of the setting screens that appears when formatting. A Visual Memory Unit also contains “label icons”.

The Main Menu file management screen shows icons that represent the Visual Memory cards. The icon can be set on the LCD as an original icon.

This is called the label icon. The BootROM contains 124 types of default icons, which are available for selection during formatting.

The default label icon can be incorporated into a program as picture data, using the BootROM font access function.



Figure 5.5 Default Label Icons

The icon shape can be obtained using the BootROM font function which is one of the BootROM service functions. However, if a volume icon is set, that setting will have priority.

6. CD Playback Screen

6.1 CD Playback Screen

Selecting the "Sound icon" from the main top menu brings up this screen.



Figure 6.6 *CD Playback screen*

6.2 Changing the CD/GD Texture

When a CD or GD is inserted in the CD player, the CD graphic is displayed and rotated on the screen. This graphic can also be prepared on the application side.

When the following file 0GDTEX.PVR is created according to the following conditions in the application in the GD-ROM, the graphic data for that file is printed as the CD label and used in CD animation.

- 1) 256 x 256 dots
- 2) ARGB4444 or ARGB1555 color format
- 3) TWIDDLE texture
- 4) Global index 0
- 5) MIPMAP is optional

Texture data must be prepared as a round shape, so *Adobe Photoshop* data is offered as a sample. Create original texture data referring to this sample data.

After replacing, the image looks as follows.



Figure 6.7 *Image after texture replacement*

Note: File names must be set so no file (except for the 1st Read file or directory) comes before the 0GDTEX.PVR file when sorted.

7. Setting Screen

7.1 Set Icon from the Settings Screen

Selecting “SET icon” from the main menu brings up a screen for making various settings.

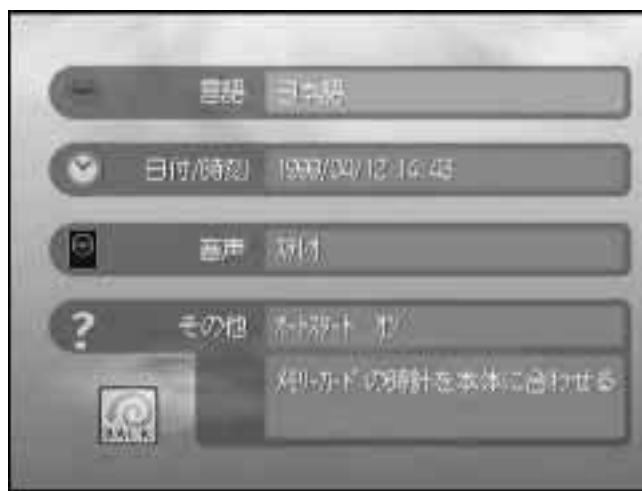


Figure 7.2 *Selecting “Set Icon”*

There are four main settings that can be made and saved here, but the SEGA library can also reference or update some of these settings.

The only settings that can be changed using the SEGA library are the date and time information and the stereo/mono information.

Use the real-time clock functions for updates and references to date and time information.

Table 7.5

Usage	Function
Date and time information reference	<code>syRtcGetDate</code> function
Date and time information update	<code>syRtcSetDate</code> function

Mono/stereo information can be referenced and updated with the following configuration functions:

Table 7.6

Usage	Function
Stereo/monoaural information reference	<code>syCfgGetSoundMode</code> function
Stereo/monoaural information update	<code>syCfgSetSoundMode</code> function

Also, language settings must be read-only.

Table 7.7

Usage	Function
Language setting information acquisition	<code>syCfgGetLanguage</code> function

Note: Stereo/monoaural information only overwrites the contents of internal flash memory. Even if the contents are overwritten, application output does not automatically switch to stereo or monoaural.

Use the function `syCfgGetSoundMode` to check the setting made by the user in the application, and then use the sound function `sdSndSetSpace` to change the sound driver setting.