

Sega  Dreamcast™

Dreamcast Middleware Outline

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 or can compress audio taken in from external devices.

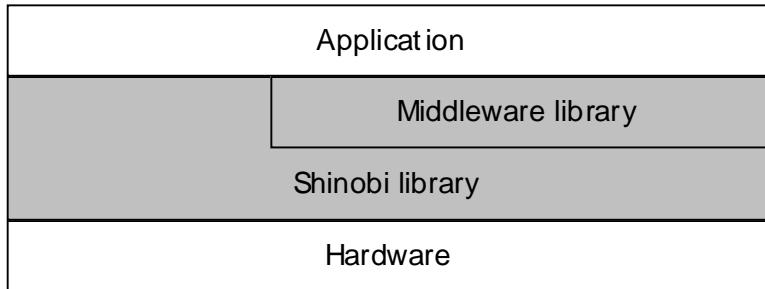


Fig. 1 Middleware Configuration

Currently, the following middleware is available for Dreamcast.

(1) CRI MPEG SofdecF/X

MPEG SofdecF/X 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 a commercially sold MPEG encoder can also be played back. In addition, this library can perform special playbacks such as visual effect, multiwindow display, seamless continuous playback and multistreaming playback.

(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 SofdecF/X, WAVE decoder and MPEG/Audio can be played back with this middleware.

(3) WAVE decoder

Can play back Dreamcast ADPCM encoded and non-encoded WAVE 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.

(6) SAN

SAN converts BMP files into YUV420 format and facilitates the display of continuous data. Also, composition can be made at the pixel level and display is possible as textures.

2. Middleware Library

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 or recording functions separately, using the same interface. Currently, MPEG SofdecF/X, TrueMotion, WAVE decoder, 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 and for recording audio after compressed by various CODEC methods.

(3) Basic middleware library

Basic library to play back and record middlewares such as middleware manager, audio renderer, video renderer and audio capture.

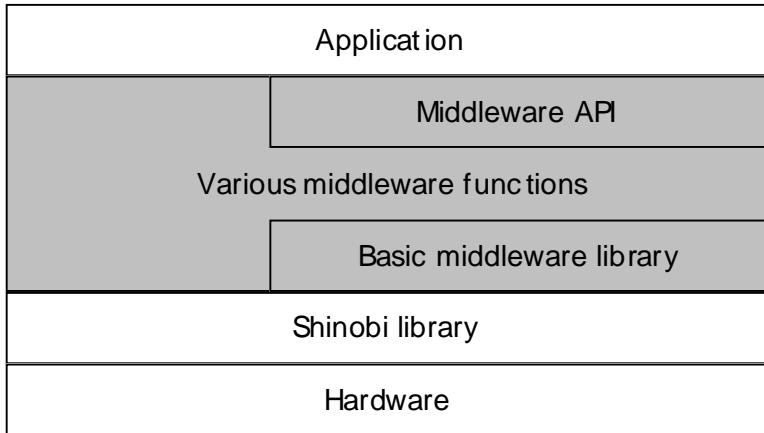


Fig. 2 Middleware Library Configuration

2.2. Common Header Files and Library Files

The header files and library files required for middleware playback and recording are as follows.

SG_MW.H	: Middleware API header file
SG_MWPLY.H	: Middleware playback library header file (included from SG_MW.H)
SG_MWREC.H	: Middleware recording library header file (included from SG_MW.H)
SG_MW.LIB	: Basic middleware library

3. CRI MPEG SofdecF/X Outline

3.1. Features

MPEG SofdecF/X serves for playback of MPEG1 encoded video and SofdecAudio encoded audio. In addition, this library can perform special playbacks such as visual effect, multiwindow display, seamless continuous playback and multistreaming playback.

3.2. Software Configuration

The software configuration is shown below. For information on MPEG SofdecF/X playback, refer to “CRI MPEG SofdecF/X Library Manual”.

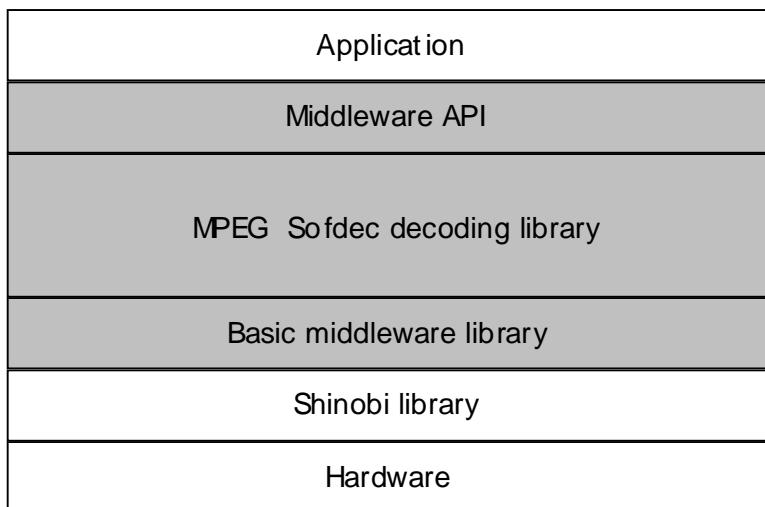


Fig. 3 MPEG SofdecF/X Software Configuration

3.3. Data Creation Tools

The following tools (CRI MPEG CRAFT) for creating MPEG SofdecF/X data are available.

SFVENCD.EXE	: MPEG/Video encoder
SFAENCD.EXE	: SofdecAudio encoder
SFDMUX.EXE	: Multiplexer
SFDMAKE.BAT	: Batch file for automated encoding operation

3.4. Header Files and Library Files

The following header files are required. By including SG_MW.H, the following file are automatically included.

SFDFX_MW.H	: MPEG SofdecF/X decoding library header
------------	--

To use MPEG SofdecF/X, the following files must be linked.

SOFDEC.LIB	: MPEG SofdecF/X decoding library
SG_MW.LIB	: Basic middleware library

4. CRI ADX Outline

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.

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.

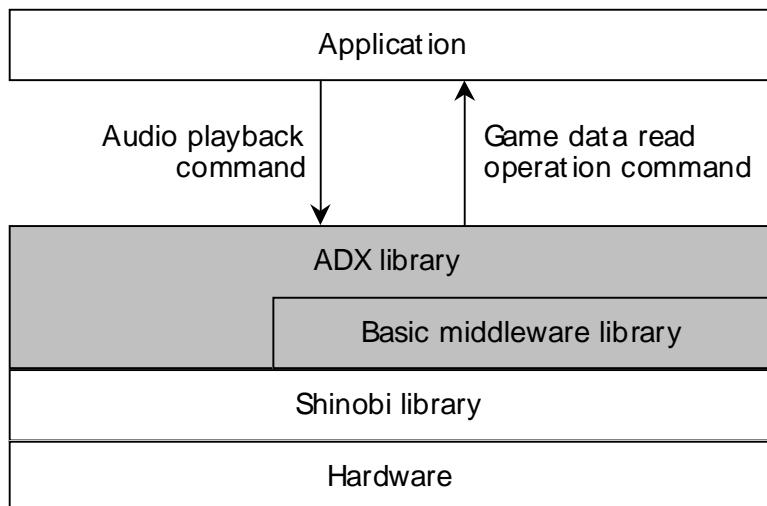


Fig. 4 ADX Software Configuration

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

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

4.5. Joint Use With MPEG SofdecF/X

To use ADX and MPEG SofdecF/X 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 SofdecF/X decoding library
SG_MW.LIB : Basic middleware library

4.6. Multistream playback with CODEC

Multistreaming can also be done for other CODEC functions (MPEG SofdecF/X, WAVE 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 SofdecF/X decoding library
SG_MWWAV.LIB : WAVE decoding library
MP1A_L2.LIB : MPEG/Audio decoding library
SG_MW.LIB : Basic middleware library

5. WAVE Decoder Outline

5.1. Features

The WAV decoder serves for playback of non-encoded or Dreamcast ADPCM encoded WAVE files.

5.2. Software Configuration

The software configuration is shown below. For information on WAVE file playback, refer to the middleware library manual.

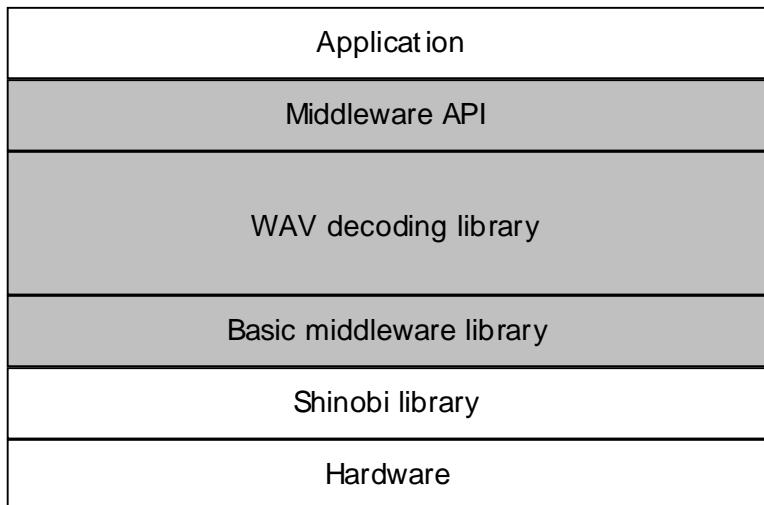


Fig. 5 WAVE Decoder Software Configuration

5.3. Data Creation Tools

The following tools for creating Dreamcast ADPCM encoded WAVE data is available.

DADENC.EXE : Dreamcast ADPCM encoder

5.4. Header Files and Library Files

The following header files are required. By including SG_MW.H, the following file are automatically included.

WAVE_MW.H : WAVE decoding library header

To play back WAVE files, link the following files.

SG_MWWAV.LIB : WAVE decoder library

SG_MW.LIB : Basic middleware library

6. TrueMotion Outline

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.

6.2. Software Configuration

The software configuration is shown below. For information on TrueMotion playback, refer to the middleware library manual.

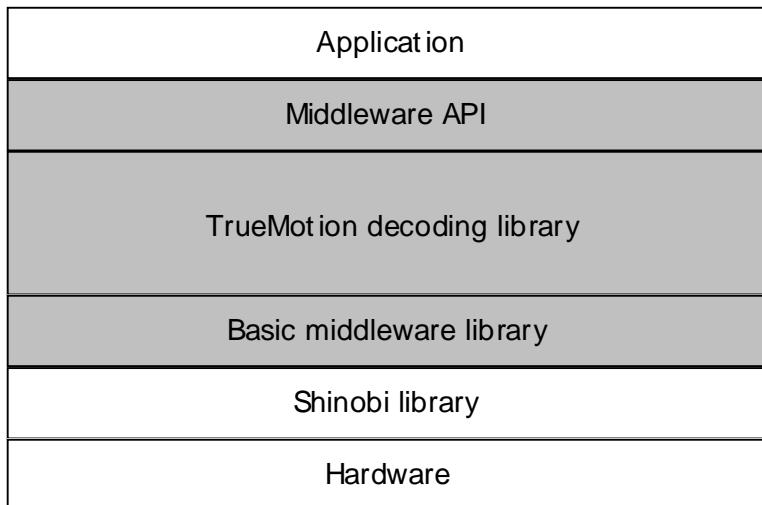


Fig. 6 TrueMotion Software Configuration

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.

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

7. MPEG/Audio Outline

7.1. Features

MPEG/Audio Layer2 data that conforms to ISO11172 can be played back by MPEG/Audio middleware.

7.2. Software Configuration

The software configuration is shown below. For information on MPEG/Audio Layer2 playback, refer to the middleware library manual.

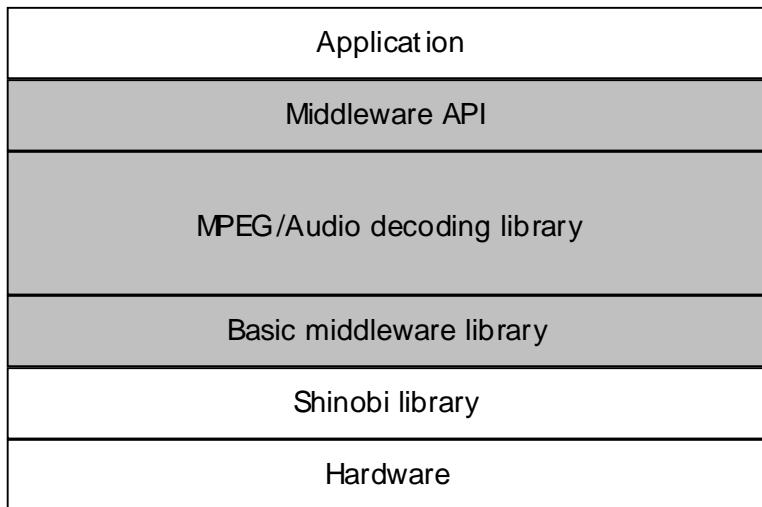


Fig. 7 MPEG/Audio Software Configuration

7.3. Data Creation Tools

MPEG/Audio data is created using Dreamcast Movie Creator.

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

8. CRI SAN Outline

8.1. Features

This library converts BMP files into YUV420 format and facilitates the display of continuous data. Also, composition can be made at the pixel level and display is possible as textures.

8.2. Software Configuration

The software configuration is shown below. For information on usage of SAN data, refer to the user's manual for simple animation library.

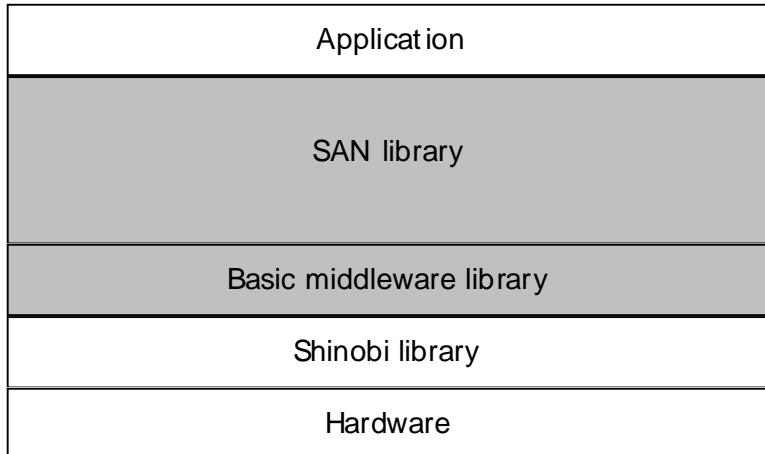


Fig. 8 SAN Software Configuration

8.3. Data Creation Tools

The following tool is used for creating SAN data.

BMP2SAN.EXE : SAN converter



***Dreamcast
Middleware Player
External
Specifications***

1. Overview

1.1. Purpose

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

(1) MPEG SofdecF/X 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) MPEG/Audio data file

This is MPEG/Audio Layer II-compressed audio data.

1.2. Modular Structure

The modular structure of the Middleware Library is shown in the following diagram.

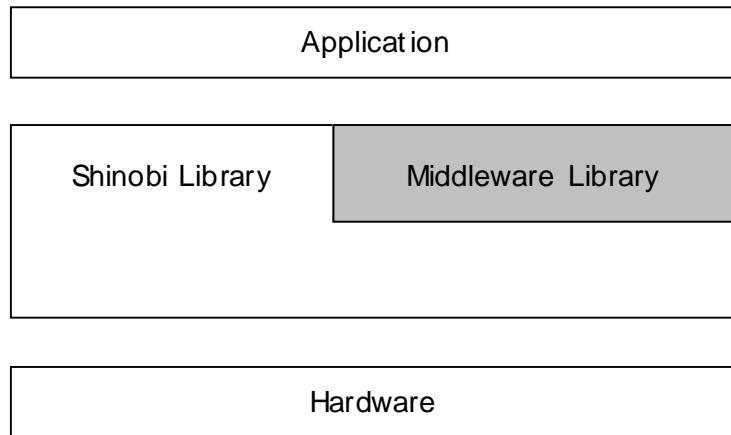


Fig. 1-1 Modular Structure

2. Middleware Library Internal Structure

The Middleware Library consists of the following modules. Applications can easily play motion pictures and sounds using the Middleware API.

Table 2-1 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

The following figure shows the internal structure of the Middleware Library.

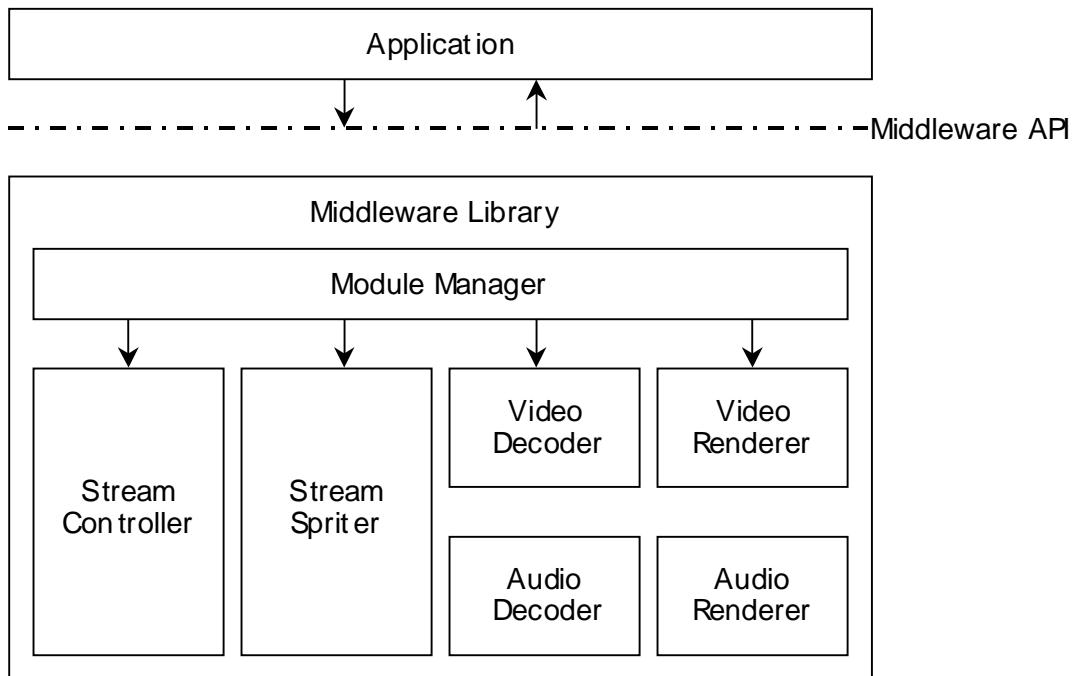


Fig. 1.2-1 Middleware Library Internal Module

3. Middleware Usage Methods

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 SofdecF/X>

```
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);
```

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 need to 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, plydlsd;
MWS_PLY_CPRM_SFD cprm_sfd;
MWS_PLY_CPRM_TM  cprm_tm;
MWS_PLY_CPRM     cprm_wav, cprm_mpa, cprm_dlsd;

plysfd          =      mwPlyCreateSofdec(&cprm_sfd);
/* Handle creation for MPEG SofdecF/X data playback */
*
plywav          =      mwPlyCreateWav(&cprm_wav);
/* Handle creation for WAVE data playback */
plytm           =      mwPlyCreateTM(&cprm_tm);
/* Handle creation for TrueMotion data playback */
plympa          =      mwPlyCreateMpa(&cprm_mpa);
/* Handle creation for MPEG/Audio data playback */
```

While different API is required for different handle, playback instruction can be done using the same API. In the case of GD-ROM playback, mwPlyStartFname function controls playback start of the video and audio data.

```
mwPlyStartFname(plysfld, "SAMPLE.SFD"); /* Start playback of MPEG SofdecF/X  
data */  
mwPlyStartFname(plywav, "SAMPLE.WAV"); /* Start playback of WAVE data */  
mwPlyStartFname(plytm, "SAMPLE.AVI"); /* Start playback of TrueMotion data */  
mwPlyStartFname(plympa, "SAMPLE.MP2"); /* Start playback of MPEG/Audio data */
```

In the case of streaming playback by stream joint, mwPlyStartSj function controls playback start of the video and audio data.

```
mwPlyStartSj(plysfld, sj); /* Start playback of MPEG  
SofdecF/X data */  
mwPlyStartSj(plywav, sj); /* Start playback of WAVE  
data */  
mwPlyStartSj(plympa, sj); /* Start playback of  
MPEG/Audio data */
```

In the case of playback of data in memory, mwPlyStarMem function controls playback stop of the video and audio data.

```
mwPlyStartMem(plysfld, addr, len); /* Start playback of MPEG  
SofdecF/X data */  
mwPlyStartMem(plywav, addr, len); /* Start playback of WAVE  
data */  
mwPlyStartMem(plympa, addr, len); /* Start playback of  
MPEG/Audio data */
```

Regardless of the playback type, mwPlyStop function controls playback stop of the video and audio data.

3.3. How to Use the Middleware Library

The internal status of the middleware library is updated by calling the main routine server function (mwExecMainServer) and then the njWaitSync 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) {  
    /* Process to get peripheral information */  
    :  
    mwExecMainServer(); // Drawing processing of motion  
    data, etc.  
    /* Drawing processing */  
    :  
    njWaitVSync();  
}
```

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. When GD read error occurs, the state changes to ERROR.

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 the following figure.

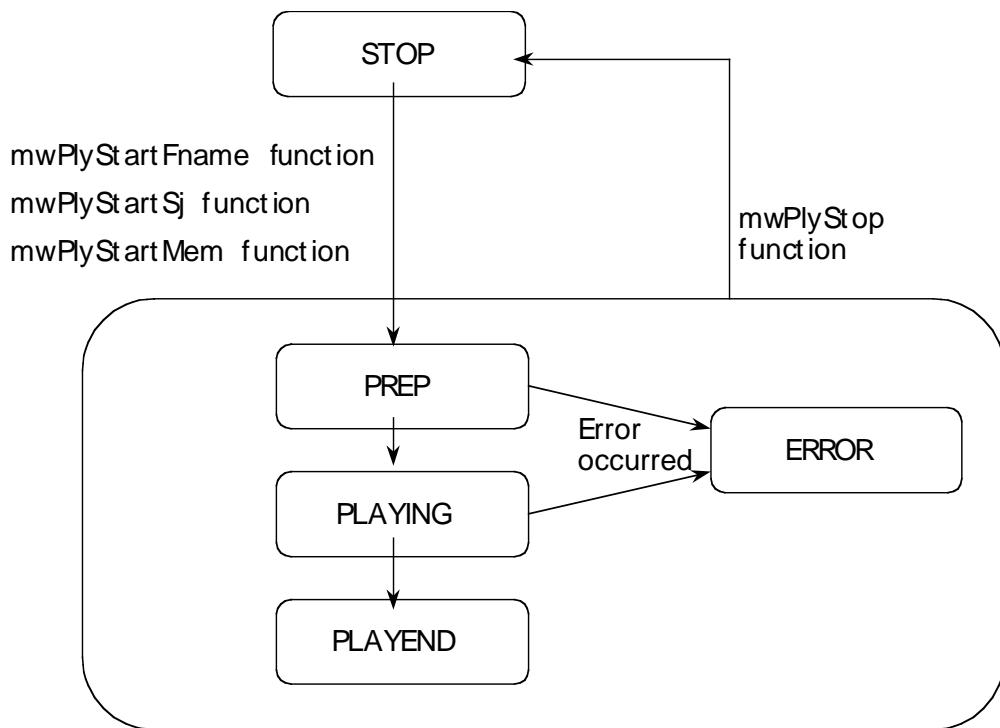


Fig. 3-1 State Diagram

3.5. Playback of MPEG SofdecF/X

The following is a sample program for playing back MPEG SofdecF/X 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_CONT)
    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 */
}
```

```

memset(&cprm, 0, sizeof(cprm));           /* 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;                    /* Number of frame buffers */
cprm.nfrm_pool_wk = 3;                   /* Calculation of work area
                                           size */
cprm.work = syMalloc(cprm.wksize);
ply = mwPlyCreateSofdec(&cprm);          /* Handle creation */
mwPlyStartFname(ply, "SAMPLE.SFD");      /* Start playing */
for (;;) {
    mwExecMainServer();                  /* Run middleware library */
    stat = mwPlyGetStat(ply);           /* Get handle status */
    if ( stat == MWE_PLY_STAT_PLAYEND ) {
        break;
    }
    njWaitVSync();                     /* Wait for V-Sync */
}
mwPlyStop(ply);                         /* Stop playback */
mwPlyDestroy(ply);                      /* Destroy handle */
syFree(cprm.work);                     /* Library initialization */
}

```

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 */
memset(&cprm, 0, sizeof(cprm));        /* Zero out reserved members
*/
cprm.size = MWD_PLY_CALC_AWORK(MWD_PLY_MIN_AWORK);
cprm.buf = syMalloc(cprm.size);
cprm.sample = MWD_PLY_SAMPLE_NUM;
ply = mwPlyCreateWav(&cprm);           /* Handle creation */
mwPlyStartFname(ply, "SAMPLE.WAV");    /* Start playing */
for (;;) {
    mwExecMainServer();                 /* Run middleware library */
    stat = mwPlyGetStat(ply);          /* Get handle status */
    if ( stat == MWE_PLY_STAT_PLAYEND ) {
        break;
    }
    njWaitVSync();                    /* Wait for V-Sync */
}
mwPlyStop(ply);                      /* Stop playback */
mwPlyDestroy(ply);                   /* Destroy handle */
syFree(cprm.buf);                   /* Library initialization */
}
```

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 (;;) {
        mwExecMainServer();             /* Run middleware library */
        stat = mwPlyGetStat(ply);       /* Get handle status */
        if ( stat == MWE_PLY_STAT_PLAYEND ) {
            break;
        }
        njWaitVSync();                 /* Wait for V-Sync */
    }
    mwPlyStop(ply);                    /* Stop playback */
    mwPlyDestroy(ply);                 /* Destroy handle */
    mwPlyFinishTM();                  /* Library initialization */
}
```

3.8. Playback of MPEG/Audio

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

3.9. Playback by Stream Joint

The following is a sample program for playback using stream joint.

<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 */
    SJ          sj;                                /* Stream joint handle */
    SJCK       ck, ck2;                            /* Chunk */
    Sint8      *buf;                               /* Stream joint work area */
    Sint8      *data;                             /* Pointer to data */
    Sint32     size;                               /* Data size */
    Sint32     pos;                                /* Data position */
    Sint32     len;                                /* Amount of data transfer */
    Sint32     nroom;                             /* Free buffer capacity */

    /*
     * Screen and sound initialization
     */
    mwPlyInitWav(MWD_PLY_SVR_VSYNC);        /* Library initialization */
    buf = syMalloc(2048 * 24);
    sj = SJRBF_Create(buf, 2048 * 24, 0);
    memset(&cprm, 0, sizeof(cprm));
    cprm.size = MWD_PLY_CALC_AWORK(MWD_PLY_MIN_AWORK);
    cprm.buf = syMalloc(cprm.size);
    cprm.sample = MWD_PLY_SAMPLE_NUM;
    ply = mwPlyCreateWav(&cprm);             /* Handle creation */
    /*
     * Prepare to playback data
     */
    mwPlyStartSj(ply, sj);                   /* Start playback */
    pos = 0;
```

```

for (;;) {
    If (pos < size) {
        nroom = SJ_GetNumData(sj, SJ_LIN_FREE);
        SJ_GetChunk(sj, SJ_LIN_FREE, nroom, &ck); /* Get free chunk */
        len = MIN(ck.len, (size - pos));
        memcpy(&ck.data, data[pos], ck.len);
        SJ_SplitChunk(&ck, len, &ck ,&ck2);
        SJ_PutChunk(sj, SJ_LIN_DATA, &ck); /* Pass data chunk */
        SJ_UngetChunk(sj, SJ_LIN_FREE, &ck2); /* Return free chunk */
        pos += len;
    }
    mwExecMainServer(); /* Run middleware library */
    stat = mwPlyGetStat(ply); /* Get handle status */
    if ( stat == MWE_PLY_STAT_PLAYEND ) {
        break;
    }
    njWaitVSync(); /* Wait for V-Sync */
}
mwPlyStop(ply); /* Stop playback */
mwPlyDestroy(ply); /* Destroy handle */
syFree(buf);
SJ_Destroy(sj);
syFree(cprm.buf);
mwPlyFinishWav(); /* Library initialization */
}

```

3.10. Playback of Memory

The following is a sample program for playing back data in memory.

<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 */
    Sint8      *data;                   /* Pointer to data */
    Sint32     size;                   /* Data size */

    /*
     * Screen and sound initialization
     */
    mwPlyInitWav(MWD_PLY_SVR_VSYNC);    /* Library initialization */
    memset(&cprm, 0, sizeof(cprm));      /* Zero out reserved members */
    cprm.size = MWD_PLY_CALC_AWORK(MWD_PLY_MIN_AWORK);
    cprm.buf = syMalloc(cprm.size);
    cprm.libwork = NULL;
    cprm.sample = MWD_PLY_SAMPLE_NUM;
    ply = mwPlyCreateWav(&cprm);        /* Handle creation */
    /*
     * Prepare to playback data
     */
    mwPlyStartMem(ply, (void *)data, size); /* Start playback */
    for (;;) {
        mwExecMainServer();             /* Run middleware library */
        stat = mwPlyGetStat(ply);       /* Get handle status */
        if ( stat == MWE_PLY_STAT_PLAYEND ) {
            break;
        }
        njWaitVSync();                /* Wait for V-Sync */
    }
    mwPlyStop(ply);                    /* Stop playback */
    mwPlyDestroy(ply);                /* Destroy handle */
    syFree(cprm.buf);                /* Library initialization */
}
```

4. Data Specifications

The Library data list is shown below.

Table 4-1 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_DTYPTE~	Display type of video	1.3
MWD_PLY_COMPO~	Composition mode	1.4
Data Types		
MWPLY	Middleware playback handle	2.1
MWS_PLY_INIT_SFD	Initialization parameter structure (for MPEG SofdecF/X)	2.2
MWS_PLY_CPRM_SFD	Handle creation parameter structure (for MPEG SofdecF/X)	2.3
MWS_PLY_CPRM	Handle creation parameter structure (for WAVE, MPEG/Audio, DualSpeech)	2.5

4.1. 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

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

The following constants indicate type of file to play for MPEG Sofdec F/X.

Constant Name	Description
MWE_PLY_FTYPE_SFD	MPEG SofdecF/X (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 display type of video.

Constant Name	Description
MWE_PLY_DTYPE_AUTO	Display size of screen is automatically adjusted. 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. This mode is left to be able to cooperate with Ver. 1.
MWE_PLY_DTYPE_WND	Adjust the display position and size according to the window level This constant is set by the mwPlySetDispPos and mwPlySetDispSize functions.
MWE_PLY_DTYPE_SRF	Adjust the display position and size according to the surface level Use the mwPlySetSrf~ functions to set the display position and brightness for each vertex.

Title	Data Name	Data	No
Data	MWE_PLY_COMPO~	Composition mode.	1.4

The following constants indicate the composition mode.

Constant Name	Description
MWE_PLY_COMPO_Opeq	Opaque
MWE_PLY_COMPO_Trnsp	Transparent
MWE_PLY_COMPO_Add	Additive composition
MWE_PLY_COMPO_Lumi	Luminance key composition
MWE_PLY_COMPO_Alph3	3-step alpha composition
MWE_PLY_COMPO_Alph5	5-step alpha composition
MWE_PLY_COMPO_Alph256	Full alpha composition
MWE_PLY_COMPO_Mix	Mixed composition mode Mixes multiple composition modes in one stream.

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 SofdecF/X. In Ninja, set the same value as the one set in 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

Parameter structure to set when creating a Middleware playback handle for MPEG SofdecF/X. 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_~.
compo_mode	Sint32	Composition mode Can be selected from MWD_PLY_COMP_~.

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).

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.
extsize[2]	Sint32	Extra buffer size (unit: bytes) In the case extra area is required for internally used buffer, specify the size in this member. (MPEG/Audio needs this to be specified.) For how to specify each CODEC, refer to function specification and description on handle creation.
sample	Sint32	Number of playback samples to start playing back audio renderer Set “MWD_PLY_SAMPLE_NUM=8192” for standard. As for DualSpeech, specify the following; MWD_DLSD_SAMPLE_NUM=1.
limit	Sint32	Limit value to stop the memory playback automatically. Set “MWD_PLY_MEMORY_LIMIT=0” for standard.

5. Function Specifications

The library functions are listed below.

Table 5-1 Function List

Function Name	Function	No.
Functions for MPEG SofdecF/X		
mwPlyPreInitSofdec	Set up system initialization	1.1
mwPlyInitSofdec	Initialize the MPEG SofdecF/X library	1.2
mwPlyFinishSofdec	Finish MPEG SofdecF/X library processing	1.3
mwPlyCreateSofdec	Create a handle for MPEG SofdecF/X	1.4
mwPlyCalcWorkCprmSfd	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	Get 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	Initializes 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	Initializes 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	Initializes the MPEG/Audio library	4.1
mwPlyFinishMpa	Finish MPEG/Audio library processing	4.2
mwPlyCreateMpa	Create a handle for MPEG/Audio	4.3

Table 5-2 Function List

Function Name	Function	No.
Global functions		
mwPlyDestroy	Destroy a handle	6.1
mwPlyStartFname	Start playing (playback of GD-ROM)	6.2
mwPlyStartSj	Start playing (playback of stream joint)	6.3
mwPlyStartMem	Start playing (playback of memory)	6.4
mwPlyStop	Stop playing	6.5
mwPlyGetStat	Get the handle status	6.6
mwPlyGetTime	Get the number of the playing sample	6.7
mwPlyPause	Perform pause setting	6.8
mwPlySetOutVol	Perform volume setting	6.9
mwPlyGetOutVol	Get the volume's value	6.10
mwPlySetOutPan	Set up panpot	6.11
mwPlyGetOutPan	Get the panpot value	6.12
mwPlyEntryErrFunc	Register a function to call on error	6.13
mwExecMainServer	Server function	6.14

5.1. Functions for MPEG SofdecF/X

This is the function required for playback of MPEG SofdecF/X.

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 SofdecF/X library.	1.2

[Syntax] void mwPlyInitSofdec(MWS_PLY_INIT_SFD *iprm);
 [Input] iprm : Initialization parameter
 [Output] none
 [Rtn Val] none
 [Purpose] It 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 mwExecMainServer is called.
 (2) V-Sync server
 Executed at V-Sync interrupt.
 (3) Idle server
 Executed within njWaitVsync during the wait time for the next game frame.
 The MPEG SofdecF/X 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 SofdecF/X 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 SofdecF/X.	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.
 [Example] A usage example is shown below.

```
memset(&cprm, 0, sizeof(cprm)); /* Zero out reserved members */
cprm.compo_mode = MWD_PLY_COMPO_OPEQ; /* Composition mode */
cprm.ftype = MWD_PLY_FTYPE_SFD; /* Playback file type */
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 */
cprm.wksize = mwPlyCalcWorkCprmSfd(cprm); /* Calculation of work
area size */
cprm.work = syMalloc(cprm.wksize); /* Secure working area */
ply = mwPlyCreateSofdec(&cprm);
```

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

[Syntax] Sint32 mwPlyCalcWorkCprmSfd(MWS_PLY_CPRM_SFD *cprm);

[Input] cprm : Handle creation parameter

[Output] none

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

[Purpose] Calculates the work area size to use in MPEG SofdecF/X 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] **p**ly : M i d d l e w a r e p l a y b a c k h a n d l e

lx : X-coordinate

ly : Y-coordinate

[Output] none

[Rtn Val] none

[Purpose] Set up the display position.

Title Function	Function Name MwPlySetDispSize	Function Set up the display size.	No 1.7
-------------------	-----------------------------------	--------------------------------------	-----------

[Syntax] void mwPlySetDispSize(MWPLY ply, float sx, float sy);
 [Input] ply : M i d d l e w a r e p l a y b a c k h a n d l e
 sx : X-coordinate
 sy : Y-coordinate
 [Output] none
 [Rtn Val] none
 [Purpose] Set up the display size.

Title Function	Function Name MwPlySetBright	Function Set up the brightness.	No 1.8
-------------------	---------------------------------	------------------------------------	-----------

[Syntax] void mwPlySetBright(MWPLY ply, Sint32 val);
 [Input] ply : M i d d l e w a r e p l a y b a c k h a n d l e
 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.

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	: M i d d l e w a r e	p l a y b a c k h a n d l e
	val	: Brightness offset (0~255)	
[Output]	none		
[Rtn Val]	none		
[Purpose]	<p>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.</p>		

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.

Title	Function Name	Function	No
Function	Function NamemwPlySetDispZ	Set up the display screen depth value.	1.12

[Syntax]	void mwPlySetDispZ(MWPLY ply, float z);
[Input]	ply : M i d d l e w a r e p l a y b a c k h a n d l e 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.

Title	Function Name	Function	No
Function	mwPlyGetDispZ	Get the display screen depth value.	1.13

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

Title	Function Name	Function	No
Function	mwPlySetDispMode	Set up the display mode.	1.14
Function	mwPlyGetDispMode	Get the display mode.	1.15

[Syntax]	void mwPlySetDispMode(Sint32 mode, Sint32 frame, Sint32 count, Sint32 latency);
[Input]	mode : S c r e e n m o d e frame : F r a m e b u f f e r c o l o r m o d e count : N u m b e r o f f r a m e s latency : Display latency
[Output]	none
[Rtn Val]	none
[Purpose]	Sets display mode in middleware.
[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]	<p>(1) Notes regarding frame count number:</p> <ol style="list-style-type: none"> 1 The display update V-Sync number is 2*count when there is interlacing. For example, if the frame count is 2 with interlace, 4V is displayed. 2 1V smoothly displays optional frame rate animation, such as 24fps. 3 1V smoothly displays 29.97fps movie for NTSC when playback is on PAL. 4 2V display gives more CPU time to MPEG SofdecF/X and the danger of dropped frames is reduced, but the frame rate for smooth playback is limited. (NTSC: 29.97 fps, VGA: 30 fps, PAL: 25fps). 5 Displays of 3V and above are not recommended for playing back animation. <p>(2) Notes regarding latency:</p> <p>For 320x240 display, a 500Kbyte texture area is used for 2V latency and a 750Kbyte texture area is used for 3V latency.</p>

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 : M i d d l e w a r e p l a y b a c k h a n d l e
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 : M i d d l e w a r e p l a y b a c k h a n d l e
sw : Audio output switch (0: sound output off, 1: audio output on)

[Output] none

[Rtn Val] none

[Purpose] Sets the audio output switch.

[Remarks] A s a d e f a u l t , a u d i o i s o u t p u t .
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.

5.2. 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. Specify MWD_PLY_EXTRA_SIZE in cprm.extsize. Specify MWD_PLY_SAMPLE_NUM in cpam.sample. Specify MWD_PLY_MEMORY_LIMIT in cpam.limit.

[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;
cprm.extsize[MWD_CH_L] = MWD_PLY_EXTRA_SIZE;
cprm.extsize[MWD_CH_R] = MWD_PLY_EXTRA_SIZE;
cprm.sample = MWD_PLY_SAMPLE_NUM;
cprm.limit = MWD_PLY_MEMORY_LIMIT;
ply = mwPlyCreateWav(&cprm);
```

5.3. 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.

5.4. 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 mwPlyInitMpa(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. Specify MWD_MPA_EXTRA_SIZE in cprm.extsize. Specify MWD_PLY_SAMPLE_NUM in cpam.sample. Specify MWD_PLY_MEMORY_LIMIT in cpam.limit.

[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));
cprm.extsize[MWD_CH_L] = MWD_MPA_EXTRA_SIZE;
cprm.extsize[MWD_CH_R] = MWD_MPA_EXTRA_SIZE;
cprm.sample = MWD_PLY_SAMPLE_NUM;
cprm.limit = MWD_PLY_MEMORY_LIMIT;
ply = mwPlyCreateMpa(&cprm);
```

5.5. Global functions

These are functions which are not tied to any codec type.

Title	Function Name	Function	No
Function	mwPlyDestroy	Destroy a handle.	6.1

[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. (Playback of GD-ROM)	6.2

[Syntax] void mwPlyStartFname(MWPLY ply, Sint8 *fname);
 [Input] ply : M i d d l e w a r e p l a y b a c k h a n d l e
 fname : Name of moving picture and audio file
 [Output] none
 [Rtn Val] none
 [Purpose] Starts playing motion pictures and sound on GD-ROM.

Title	Function Name	Function	No
Function	MwPlyStartSj	Start playing. (Playback of stream joint)	6.3

[Syntax] void mwPlyStartSj(MWPLY ply, SJ sj);
 [Input] ply : M i d d l e w a r e p l a y b a c k h a n d l e
 sj : Stream joint handle
 [Output] none
 [Rtn Val] none
 [Purpose] Starts playing motion pictures and sound that are provided for stream joint.

Note: This function is not supported by Truemotion Ver. 2.24.

Title	Function Name	Function	No
Function	MwPlyStartMem	Start playing. (Playback from memory)	6.4

[Syntax] void mwPlyStartMem(MWPLY ply, void *addr, Sint32 len);
 [Input] ply : M i d d l e w a r e p l a y b a c k h a n d l e
 addr : Pointer to where data of motion pictures or sound are
 len : Length of data
 [Output] none
 [Rtn Val] none
 [Purpose] Starts playing motion pictures and sound in memory.

Note: This function is not supported by Truemotion Ver. 2.24.

Title Function	Function Name mwPlyStop	Function Stop playing.	No 6.5
--------------------------	-----------------------------------	----------------------------------	------------------

[Syntax] void mwPlyStop(MWPLY ply);
 [Input] ply : Middleware playback handle
 [Output] none
 [Rtn Val] none
 [Purpose] Stops playing motion pictures and sound.

Title Function	Function Name mwPlyGetStat	Function Get the handle status.	No 6.6
--------------------------	--------------------------------------	---	------------------

[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	Function Name	Function	No
	mwPlyGetTime	Get the number of the playing sample.	6.7

[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 (time) can be calculated by:
 rtime = ncount / tscale;

Title Function	Function Name	Function	No
	mwPlyPause	Perform pause setting.	6.8

[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.

Title	Function Name	Function	No
Function	mwPlySetOutVol	Perform volume setting.	6.9

[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 the 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 the fade-out effect.

Title	Function Name	Function	No
Function	mwPlyGetOutVol	Get the volume's value.	6.10

[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.

Title	Function Name	Function	No
Function	mwPlySetOutPan	Set up panpot.	6.11
Function	mwPlyGetOutPan	Get panpot.	6.12

[Syntax]	void mwPlySetOutPan (MWPLY ply, Sint32 chno, Sint32 pan);
[Input]	<p>ply : Middleware playback handle</p> <p>chno : C h a n n e l t o s e t Monaural / stereo (left) : M W D _ C H _ L (0) Stereo (right): MWD_CH_R(1)</p> <p>pan : P a n p o t (- 1 5 ~ 1 5 , - 1 2 8) MWD_PAN_LEFT = -15, MWD_PAN_RIGHT = 15 MWD_PAN_CENTER = 0, MWD_PAN_AUTO = -128</p>
[Output]	none
[Rtn Val]	none
[Purpose]	<p>Set up pan pot 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.</p>

Title	Function Name	Function	No
Function	mwPlyGetOutPan	Get the panpot value.	6.12

[Syntax]	Sint32 mwPlyGetOutPan (MWPLY ply, Sint32 chno);
[Input]	<p>ply : Middleware playback handle</p> <p>chno : C h a n n e l t o g e t M o n a u r a l / s t e r e o (l e f t) : M W D _ C H _ L (0) Stereo (right): MWD_CH_R(1)</p>
[Output]	none
[Rtn Val]	Panpot value of the supported channel (-15~15)
[Purpose]	Get the value of the panpot which is currently set.

Title	Function Name	Function	No
Function	mwPlyEntryErrFunc	Register a function to call on error.	6.13

[Syntax] void mwPlyEntryErrFunc (MW_PLY_ERRFN errfn, void *obj);

[Input] errfn : E r r o r f u n c t i o n
obj : 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);
}
```

Title	Function Name	Function	No
Function	mwExecMainServer	Server function.	6.14

[Syntax] void mwExecMainServer(void);

[Input] none

[Output] none

[Rtn Val] none

[Purpose] Refreshes the internal status of the handle.

6. Appendix

6.1. 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_TRAYOPEN || 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);
    :
    :
}
```

6.2. Concurrent playback of MPEG SofdecF/X and ADX

Concurrent playback of MPEG SofdecF/X and ADX is possible using the ADX playback library.

With this function, a seamless transition from a game scene to a movie scene is 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 SofdecF/X 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 WKSIZE44S ADXT_CALC_WORK(2, ADXT_PLY_STM, 2, 44100)

char    wk44[WKSIZE44S];                      /* Work area */
MWPLY  ply;                                  /* MWPLY handle */
ADXT   adxt_bgm;                            /* ADXT handle */

/* Application */
:
:
adxt_bgm = ADXT_Create(2, wk44, WKSIZE44S);      /* 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 = mwPlyCalcWorkSfdCprm(cprm);
cprm.work = syMalloc(cprm.wksize);
ply = mwPlyCreateSofdec(&cprm);

ADXT_StartFname(adxt_bgm, "BGM.ADX");          /* Audio playback start */

if /* event playback */
  mwPlyStartFname(ply, "SCENE01.M1V");        /* Movie playback */
:
:
```

6.3. System Configuration

The system configuration for the middleware playback library is shown below.

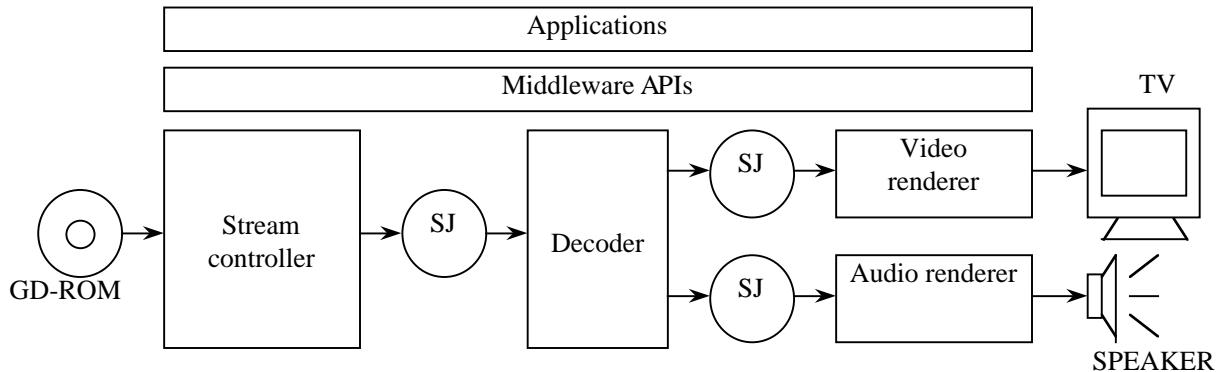


Fig. 6-1 Middleware Playback Library System Configuration

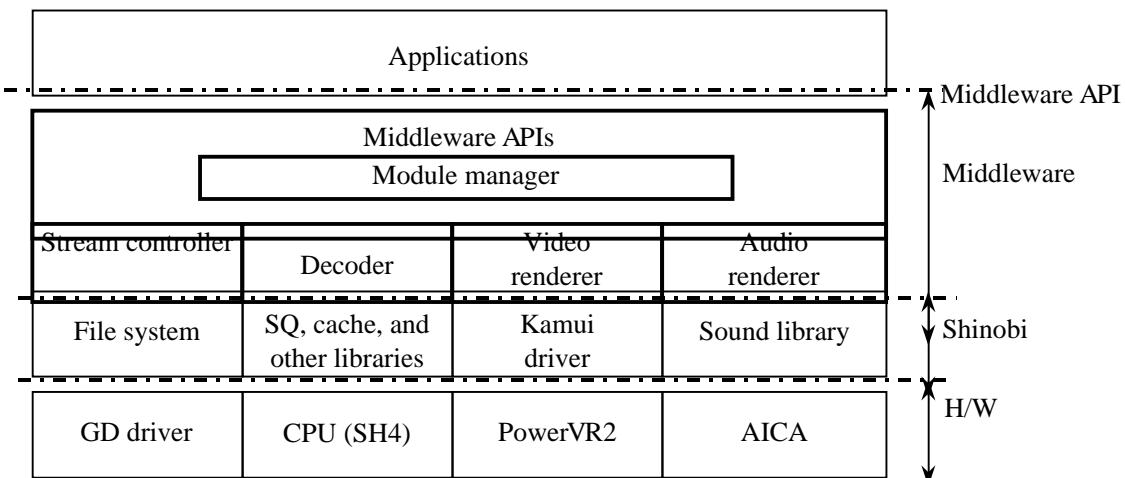
6.4. Module Configuration

The middleware playback library consists of the following modules.

Table 6-1 Internal Modules in the Middleware Playback Library

Module	Symbol	Description
Middleware player	MWPLY	Controls all processes from loading data to output, and plays back video and audio.
Module manager	MWMNG	Allocates CPU time for the decoder.
Stream controller	MWGSC	Loads video and audio data.
Decoder	_____	Decodes compressed data.
Video renderer	MWRNV	Outputs decoded video data.
Audio renderer	MWRNA	Outputs decoded audio data.

The internal configuration of the middleware playback library is shown below.



* SQ: Store Queue

Fig. 6-2 Internal Configuration of Middleware Playback Library

6.5. Flow of Control

The times at which the three main types of middleware processing are performed is described below.

Table 6-2 Types of Server Processing

Server processing	Description
V-Sync interrupt server processing	This processing is executed when a V-Sync interrupt is generated. This is processing that must be performed on a regular basis, such as audio data decoding and audio output.
Main server processing	This processing is executed when the mwExecMainServer function is called within the main processing. This processing transfers and renders video data.
Idle server processing	This processing is executed within the frame flip function during the time until the next game frame. In the Ninja library, this processing is executed within the njWaitVSync function. This is processing for which the CPU load fluctuates, such as video data decoding.

The flow of control is illustrated below.

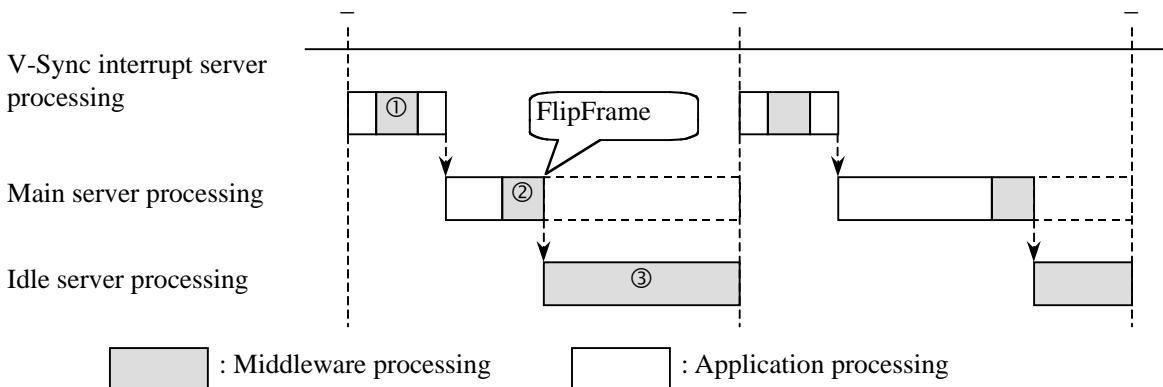


Fig. 6-3 Flow of Control

The timing at which the processing is performed varies according to the codec.

Table 6-3 Middleware Processing with Different Codecs

	Sofdec	TrueMotion	WAVE	ADX
① V-Sync interrupt server processing	Supported	Supported	Supported	Supported
② Main server processing	Supported	Supported	-	-
③ Idle server processing	Supported	-	-	-

* MPEG/Audio and DualSpeech are the same as WAVE.

The breakdown of server processing in Sofdec is as follows:

- ① : Audio data decoding and audio output processing
- ② : Transfer of video data to texture memory and rendering processing
- ③ : Video data decoding

The breakdown of server processing in TrueMotion is as follows:

- ① : Audio output processing
- ② : Video data decoding and video output processing

6.6. Sound Resources

The sound driver and the sound library are used for audio output in the middleware playback library. Allocate sound resources for the middleware playback library as described in the table below. Because the maximum

Table 6-4 Sound Resources

Sound Resource	Required amount
PCM stream ports	1
Memory block handles	4
Sound RAM	4040H(16448)byte × Number of channels

It is possible to play sound effects and MIDI sequences stored in sound RAM while also playing back sound data through the middleware playback library. In the Sega library environment, the middleware playback library is implemented by using the sound library "PCM stream playback functions. The PCM stream buffer for middleware playback is allocated from the end of sound RAM in units of 4040H (16,448) bytes. Accordingly, it is necessary to determine the PCM stream buffer size based on the number of channels used by the middleware playback library, and then create a multi-unit file so that those areas do not overlap.

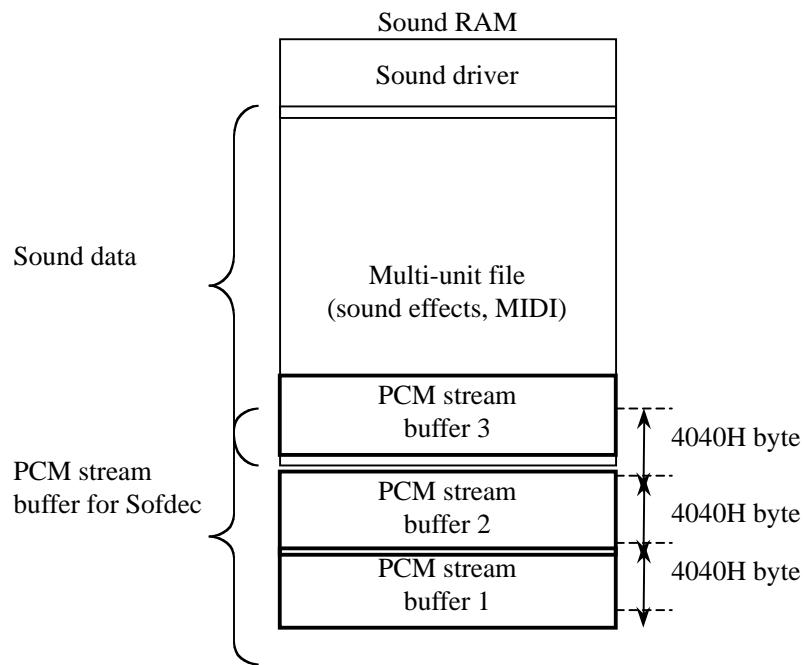


Fig. 6-4 Position of PCM Stream Buffer for Middleware Playback

6.7. Video Resources

The relationship of video resources with the graphics library varies according to the codec.

(1) Sofdec

Sofdec is implemented on top of the Kamui driver. During initialization, Sofdec accesses the Ninja library, and gets information on vertex buffers, etc. Outside of initialization, Sofdec operates independently from the Ninja library.

The relationship between Sofdec and the graphics library is illustrated below.

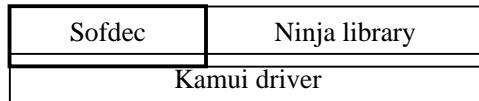


Fig. 6-5 Relationship Between Sofdec and Graphics Library

TrueMotion

TrueMotion is implemented on top of the Ninja library.

The relationship between TrueMotion and the graphics library is illustrated below.

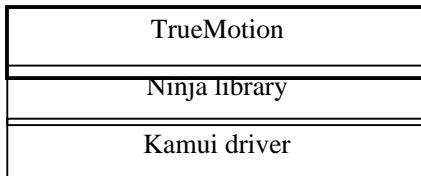


Fig. 6-6 Relationship Between TrueMotion and Graphics Library

When playing back a movie, YUV422 textures are used. When creating handles, Sofdec uses the Kamui driver to allocate texture areas. TrueMotion decodes textures in areas allocated by the user

Sega Dreamcast™

**Dreamcast
CRI MPEG
SofdecF/X Library**

1. Overview

1.1. What Is Sofdec F/X?

Sofdec F/X provides special playback functions, such as visual effects. Sofdec F/X consists of the following two libraries:

(1) **CRI MPEG Sofdec F/X library**

This library can combine and play back MPEG movies with polygons.

(2) **SAN library**

This library can make compositions with noncompressed full-color still images at the pixel level.

This document describes the CRI MPEG Sofdec F/X library. For details on the SAN library, refer to the "Simple Animation Library User's Manual."

The features of this library are described below.

(1) **Upwards compatibility with MPEG Sofdec**

This library is upwardly compatible with the CRI MPEG Sofdec library. The basic APIs conform with the middleware APIs. Therefore, you can transition to MPEG Sofdec F/X simply by recompiling, without needing to make any modifications to the source program. This library can also play back MPEG Sofdec data.

(2) **Visual effect functions**

This library possesses alpha composition and additive composition functions that permit composition with polygons at the pixel level.

(3) **Multiwindow display function**

This library has a function that allows you to cut out a specific portion of a movie image and reshape it for display at any position on the display. Images can be displayed in multiple windows.

(4) **Texture display function**

This library has a function that allows it to display a movie as a polygon texture. Because this library is capable of alpha composition, such textures can be extracted in any desired shape.

(5) **Multistreaming function**

This library has a function that allows it to simultaneously play back multiple movie files in parallel. Parallel playback of up to four files is possible.

(6) **Seamless continuous playback function**

This library has a function that allows it to consecutively play separate movie files continuously. It can also seamlessly loop the same file repeatedly. Currently, this function only supports video files.

2. CRI MPEG Sofdec F/X

2.1. System Configuration

The system configuration of the MPEG Sofdec F/X is shown below.

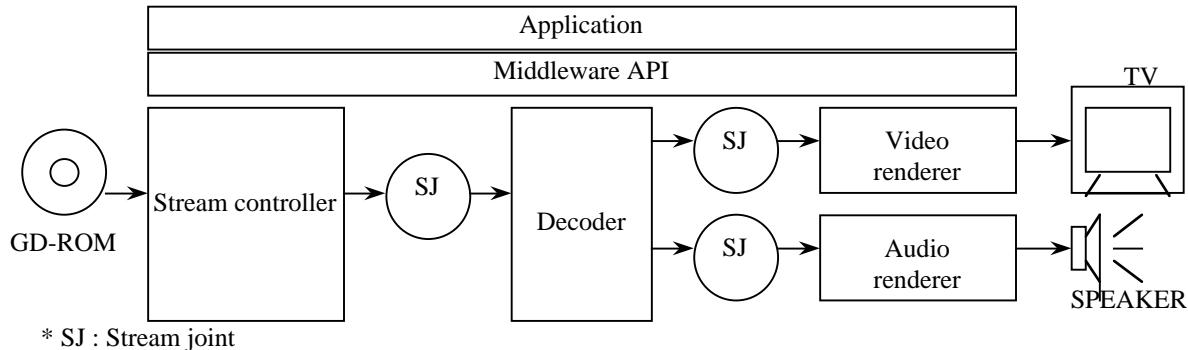


Fig. 2-1 MPEG Sofdec F/X Library System Configuration

2.2. Module Configuration

The MPEG Sofdec F/X consists of the following modules.

Table 2-1 Internal Modules

Module	Symbol	Description
Middleware player	MWPLY	Manages all phases of processing from loading data to outputting data, and plays back video and audio.
Module manager	MWMNG	Allocates CPU time for the decoder.
Stream controller	MWGSC	Loads video and audio data.
Decoder	CRI_SFD	Expands compressed data.
Video renderer	MWRNV	Outputs expanded video data.
Audio renderer	MWRNA	Outputs expanded audio data.

The internal configuration of the MPEG Sofdec F/X is shown below.

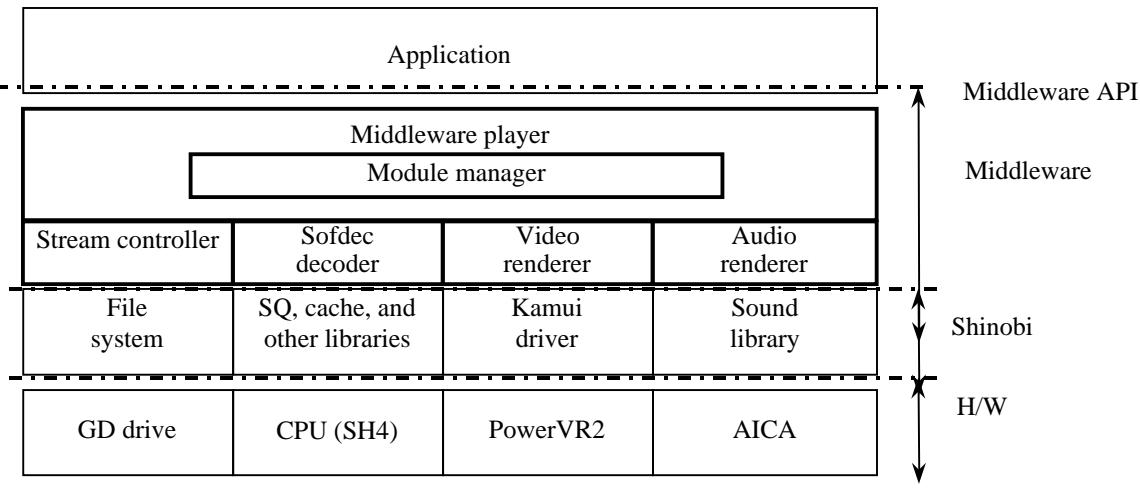


Fig. 2-2 MPEG Sofdec F/X Internal Configuration

2.3. Controller Flow

The timing of MPEG Sofdec F/X processing is described below.

Table 2-2 Types of Server Processing

Server processing	Description
V-Sync interrupt server processing	This processing is executed when a V-Sync interrupt is generated. This executes processing that is always performed periodically, such as audio data decoding and audio output.
Main server processing	This processing is executed in the main processing when the "mwExecMainServer" function is called. This function transfers and draws video data.
Idle server processing	This processing is executed within the frame flip function in the interval before the next game frame. In the Ninja library, this processing is executed within the njWaitVSync function. This processing handles processing for which the CPU load fluctuates, such as video data decoding.

The control flow chart is shown below.

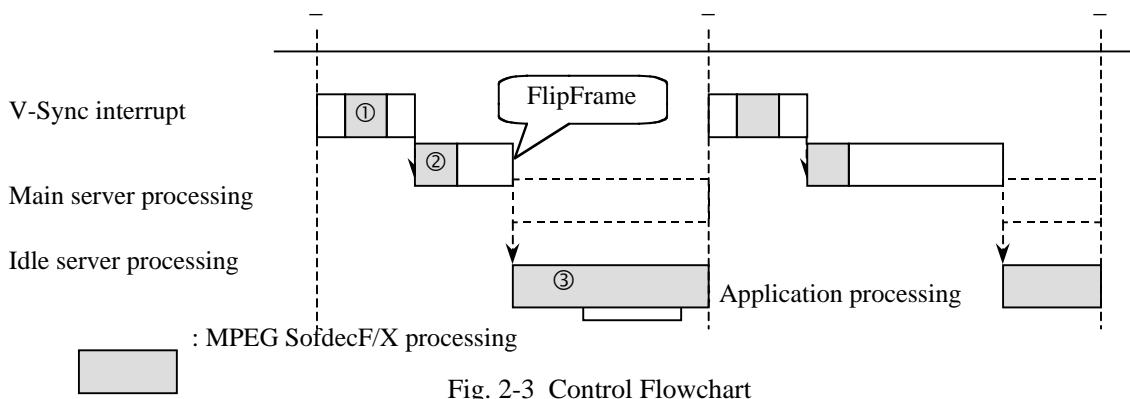


Fig. 2-3 Control Flowchart

The breakdown of each type of server processing in MPEG Sofdec F/X is as follows:

- ① : Audio data decoding and audio output processing
- ② : Transfer of video data to texture memory and drawing processing
- ③ : Video data decoding

2.4. Sound Resources

The sound driver and sound library are used for audio output in the MPEG Sofdec F/X. Allocate sound resources for the MPEG Sofdec F/X as shown in the table below. Because the maximum value for the memory block handle is 64, using MPEG Sofdec F/X reduces the number of memory block handles that can be used by the application.

Table 2-3 Sound Resources (one handle)

Sound resource	Required amount
PCM stream ports	1
Memory block handles	4
Sound RAM	4040H (16,448) bytes × number of channels

* With Ver. 2.24, the number of channels is fixed to 2.

Sound effects in sound RAM and MIDI sequences can be played back simultaneously with playback of movie data. In the Sega library environment, the MPEG Sofdec F/X is implemented by using sound library PCM stream playback functions. The MPEG Sofdec F/X PCM stream buffers are allocated in units of 4040H (16,448) bytes, starting from the end of the sound RAM. Accordingly, it is necessary to determine the size of the PCM stream buffers on the basis of the number of channels used by the middleware playback library, and then to create a multi-unit file in which those areas do not overlap.

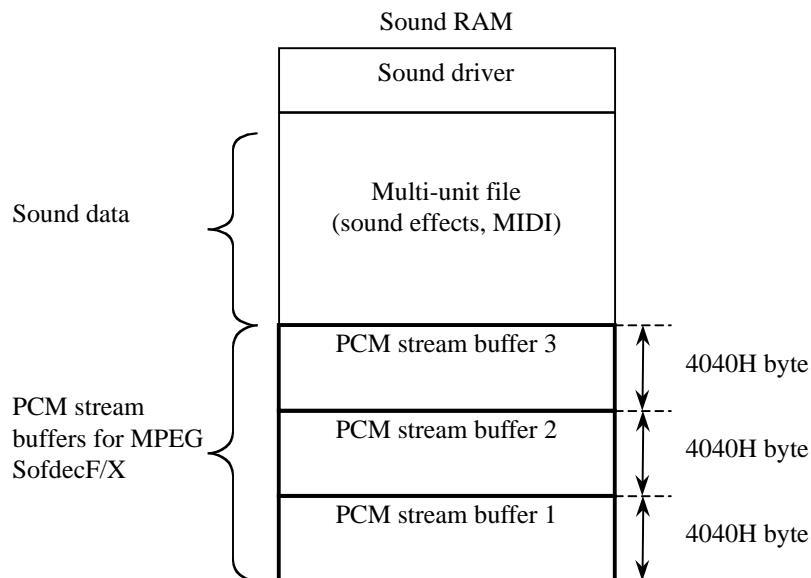


Fig. 2-4 Position of the PCM Stream Buffers for MPEG Sofdec F/X

2.5. Video Resources

MPEG Sofdec F/X is installed in the Kamui driver. During initialization, it accesses the Ninja library and gets information concerning the vertex buffer, etc. At times other than initialization, MPEG Sofdec F/X operates independently of the Ninja library.

The relationship between MPEG Sofdec F/X and the graphic library is shown below.

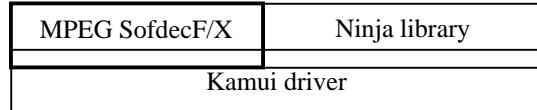


Fig. 2-5 Relationship between MPEG Sofdec F/X and Graphic Library

YUV422 textures are used for video playback. MPEG Sofdec F/X uses the Kamui driver to allocate memory when creating a handle.

3. Simple Playback

3.1. Shinobi System Setup

MPEG Sofdec F/X requires that the mwPlyPreInitSofdec function be executed before the system is initialized (sbInitSystem).

```
mwPlyPreInitSofdec();  
sbInitSystem(SYS_MODE, SYS_FRAME, SYS_COUNT); // Shinobi system initialization
```

3.2. Library Initialization and Termination Processing

(1) When using the Ninja library

If the "mwPlyInitSofdec" function is used to initialize the MPEG Sofdec F/X library, the MPEG Sofdec F/X library internally gets the pointer for the vertex buffer in the Ninja library and uses that pointer for rendering video.

```
mwPlyInitSofdec(&iprm);  
/* Video playback */  
mwPlyFinish();
```

(2) When using only the Kamui driver

Use the "mwPlyInitSfdFx" function to initialize the library.

Use the "mwPlySetVertexBuffer" function to set the vertex buffer separately.

```
mwPlyInitSfdFx(&iprm);  
mwPlySetVertexBuffer(vbuf); // Vertex buffer setting  
/* Video playback */  
mwPlyFinishSfdFx();
```

3.3. Handle Generation/Deletion

In order to play back MPEG Sofdec F/X data, set the generation parameters and generate a handle.

```
MWPLY          ply;  
MWS_PLY_CPRM_SFD  cprm;  
  
memset(&cprm, 0, sizeof(cprm)); /* For zero setting for reserved members */  
cprm.compo_mode = MWD_PLY_COMPO_OPEQ; /* Composition mode */  
cprm.ftype = MWD_PLY_FTYPE_SFD; /* Playback file type */  
cprm.dtype = MWD_PLY_DTYPE_AUTO; /* Image quality priority */  
cprm.max_bps = 450*1024*8; /* Bit rate : 450 Kbyte/sec */  
cprm.max_width = 320; /* Image size: 320×480 */  
cprm.max_height = 480;  
cprm.nfrm_pool_wk = 3; /* Number of pages in frame buffer */  
cprm.wksize = mwPlyCalcWorkCprmSfd(cprm); /* Work area size calculation */  
cprm.work = syMalloc(cprm.wksize); /* Work area allocation */  
ply = mwPlyCreateSofdec(&cprm); // Handle generation  
  
/* Movie playback */  
mwPlyDestroy(ply); // Handle deletion
```

3.4. Playback Start/Stop

When playing data from a GD-ROM, the mwPlyStartFname function controls the start of playback. Regardless of the playback method that is used, the mwPlyStop function controls the stop of playback.

```
mwPlyStartFname(ply, "SAMPLE.SFD");           // Playback start
/* Movie playback */
mwPlyStop(ply);                            // Playback stop
```

<Sample program>

```
void main(void)
{
    MWS_PLY_INIT_SFD    iprm;           /* Library initialization parameter */
    MWS_PLY_CPRM_SFD    cprm;           /* Handle generation parameter */
    MWPLY                ply;             /* Playback handle */

    mwPlyPreInitSofdec();               /* Shinobi system setting */
    /*
     * Screen and sound initialization
     */
    memset(&iprm, 0, sizeof(iprm));      /* For zero setting for reserved members */
    iprm.mode = NJD_RESOLUTION_640x480_NTSCNI; /* Display mode */
    iprm.frame = NJD_FRAMEBUFFER_MODE_ARGB8888; /* Frame buffer mode */
    iprm.count = 1;                      /* System count */
    iprm.latency = 2;                   /* Display latency */
    mwPlyInitSofdec(&iprm);           /* Library initialization */

    memset(&cprm, 0, sizeof(cprm));      /* For zero setting for reserved members */
    cprm.compo_mode = MWD_PLY_COMPO_OPEQ; /* Composition mode */
    cprm.ftype = MWD_PLY_FTYPE_SFD;      /* Playback file type */
    cprm.dtype = MWD_PLY_DTYPE_AUTO;     /* Image quality priority */
    cprm.max_bps = 450*1024*8;          /* Bit rate:450 Kbyte/sec */
    cprm.max_width = 320;                /* Image size:320×480 */
    cprm.max_height = 480;
    cprm.nfrm_pool_wk = 3;              /* Number of pages in frame buffer */
    cprm.wksize = mwPlyCalcWorkCprmSfd(cprm); /* Work area size calculation */
    cprm.work = syMalloc(cprm.wksize);   /* Work area allocation */
    ply = mwPlyCreateSofdec(&cprm);    /* Handle generation */

    mwPlyStartFname(ply, "SAMPLE.SFD"); /* Playback start */
}
```

4. Memory Playback

Sofdec data in memory can be played back by using the mwPlyStartMem function.

<Sample program>

```
/* Main application function */
void main(void)
{
    void      *data;                      /* Pointer to Sofdec data */
    Sint32    size;                       /* Data size */

    mwPlyPreInitSofdec();                 /* Shinobi system set up */
    /*
     * Screen and sound initialization
     */
    mwPlyInitSofdec(&iprm);
    ply = mwPlyCreateSofdec(&cprm);       /* Handle generation */

mwPlyStartMem(ply, (void *)data, size); /* Playback start */
    for (;;) {
        mwExecMainServer();                /* Middleware library execution */
        stat = mwPlyGetStat(ply);          /* Handle status acquisition */
        if (stat == MWE_PLY_STAT_PLAYEND)
            break;
        njWaitVSync();                    /* Wait for V-sync */
    }
    mwPlyDestroy(ply);                  /* Handle deletion */
    mwPlyFinishSofdec();                /* Library end processing */
}
```

5. Stream Joint Playback

The mwPlyStartSj function plays back Sofdec data that the user has supplied to a stream joint. Although playback is possible with a stream joint created by the user, it is also possible to get a playback handle's internal stream joint and use that for playback.

<Sample program>

```
/* Main application function */
void main(void)
{
    SJ             sj;                      /* Stream joint handle */
    SJCK          ck, ck2;                 /* Chunks */

    mwPlyPreInitSofdec();                  /* Shinobi system set up */

    /* Screen and sound initialization */
    mwPlyInitSofdec(&iprm);
    ply = mwPlyCreateSofdec(&cprm);       /* Handle generation */
    sj = mwPlyGetInputSj(ply);            /* Input stream joint acquisition */
    mwPlyStartSj(ply, sj);                /* Playback start */

    for (;;) {
        nroom = SJ_GetNumData(sj, SJ_LIN_FREE);
        SJ_GetChunk(sj, SJ_LIN_FREE, nroom, &ck);   /* Free chunk acquisition */
        nbytes = user_supply_data(ck.data, ck.len);  /* nbytes data supply */
        SJ_SplitChunk(&ck, len, &ck, &ck2);        /* Splits chunk */
        SJ_PutChunk(sj, SJ_LIN_DATA, &ck);          /* Passes data chunk */
        SJ_UngetChunk(sj, SJ_LIN_FREE, &ck2);        /* Returns free chunks */

        mwExecMainServer();                     /* Main server processing execution */
        stat = mwPlyGetStat(ply);              /* Handle status acquisition */
        if (stat == MWE_PLY_STAT_PLAYEND)
            break;
        njWaitVSync();                         /* Wait for V-sync */
    }
    mwPlyDestroy(ply);                     /* Handle deletion */
    mwPlyFinishSofdec();                  /* Library end processing */
}
```

5.1. Server Function

The internal status of MPEG Sofdec F/X is updated by calling the server function for the main processing routine (mwExecMainServer) before the njWaitVSync function.

```
whlie (1) {
    /* Peripheral information acquisition processing */
    :
    mwExecMainServer();                                // Movie data rendering processing, etc.
    /* Rendering processing */
    :
    njWaitVSync();
}
```

5.2. Handle Operation Status

The handle operation statuses are listed below. Immediately after a handle is generated, it is in the STOP state. After playback starts, the status changes as follows PREP -> PLAYING -> PLAYEND. If a GD read error, etc., occurs, the status changes to ERROR.

Table 5-1 Handle Status

Status	Description
STOP	Playback is stopped
PREP	Preparing for playback
PLAYING	Playback in progress
PLAYEND	Playback has ended
ERROR	Error has occurred

The state transition diagram is shown below.

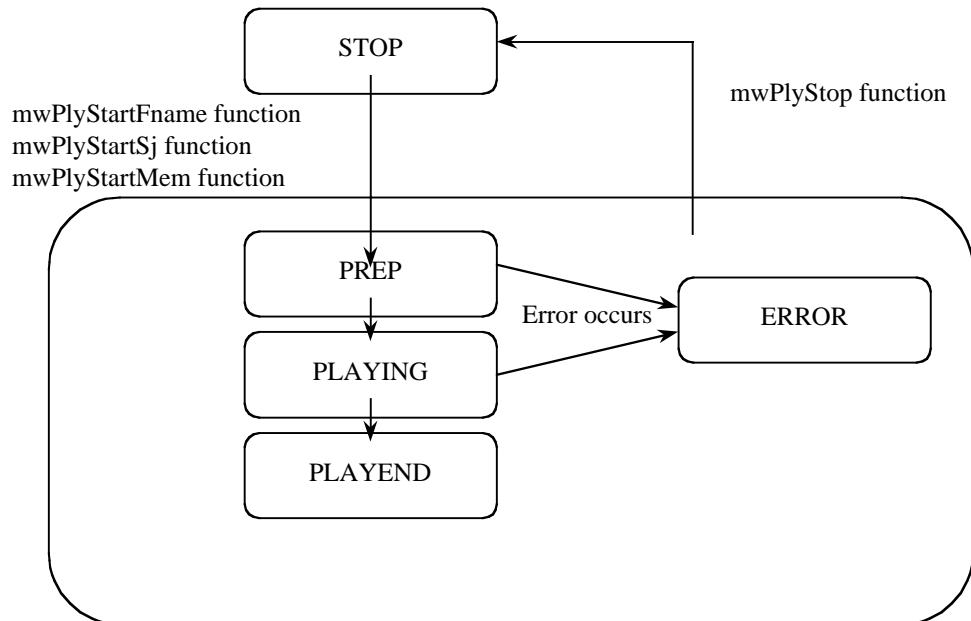


Fig. 5-1 State Transition Diagram

6. Cautions Concerning Movie Playback

6.1. Handle Generation/Deletion

The following video resources are allocated when a handle is generated.

(1) YUV422 texture surface

The surface size is defined by a rectangle with sides that are powers of two that are equal to or greater than the image size of the movie.

<Example> Movie size Surface size
 320 × 240 → 512 × 256

(2) 8-bit palette texture size

The surface size is defined by a square with sides that are powers of two that are equal to or greater than the image size of the movie.

<Example> Movie size Surface size
 320 × 240 → 512 × 512

When a handle is deleted, the texture surface is released. Therefore, if a handle is to be deleted during playback, stop playback, wait 1V or 2V cycles and then delete the handle. Also note, the mwExecMainServer function renders video. Execute this function after executing the mwPlyDestroy function.

<Sample program>

```
for (;;) {
    if ( /* Button B was pressed */ ) {
        mwPlyStop(ply);
        njWaitVSync();           // Once when latency is 2V
        njWaitVSync();           // Twice when latency is 3V
        mwPlyDestroy(ply);
    }
    mwExecMainServer();         // Execute after the mwPlyDestroy function
    njWaitVsync();
}
```

6.2. Playback Start/Stop

If two movies are to be played consecutively, and you wish to start the second movie while the first movie is still being displayed, follow the procedure described below to play the movies.

```
mwPlyStartFname  
  - 1st movie  
  mwPlyStartFname  
    - 2nd movie
```

{ Execute this function while the last frame of the first movie is being displayed; the screen will not change to the background color.

The mwPlyStop function stops movie output (rendering). If the mwPlyStop function is executed while a movie is being played back, the background color is displayed.

```
mwPlyStartFname  
  - 1st movie  
    mwPlyStop  
  - Background color screen  
    mwPlyStartFname  
  - 2nd movie
```

- * For the sake of simplicity, in this explanation the movie is depicted as being output on the screen immediately after the mwPlyStartFname function is executed. In reality, however, the movie does not appear on the screen immediately after the mwPlyStartFname function is executed. The movie is first output on the screen after the status of the middleware playback handle changes to MWE_PLY_STAT_PLAYING.

1.3. Displaying the Last Frame of a Movie

Basically, when movie playback ends, the final frame continues to be rendered on the screen. Rendering stops when the mwPlyStop function is executed.

Furthermore, one of the characteristics of MPEG is that sometimes the last frame is not displayed, and two or three images are left in the decoder. Therefore, in order to display the end of a movie as a still image on the screen, the creator of the movie should insert the same image for the last two or three frames of the movie. Because the bit rate is high, the picture quality of the I picture in MPEG Sofdec F/X movies is poor; make adjustments so that the movie stops at a P picture or a B picture. It is important to note that this characteristic concerning picture quality is not true for normal MPEG data.

<Picture Quality by Picture Type with MPEG Sofdec F/X>

I picture < P picture = B picture

1.4. Caching

The MPEG Sofdec F/X operates at peak performance when the operand cache is set to index cache mode. Although the library will operate even if the operand cache is not set to index cache mode, performance will be lower by as much as 20%.

If the MPEG Sofdec F/X library was initialized by using the "mwPlyInitSofdec" function or the "mwPlyInitSfdFx" function, the cache mode is switched to index cache mode. Furthermore, the "mwPlyFinishSofdec" function and the "mwPlyFinishSfdFx" function restore the Shinobi default cache mode setting. To permit an application to operate efficiently, either perform initialization and termination processing frequently, or else use the cache control functions to switch the cache mode.

When a polygon and a movie both exist, determine the cache mode on the basis of the movie load. As a guideline, if the movie resolution is 160×120 or lower, set the normal cache mode; if the movie resolution is higher, set the index cache mode.

<Cache control example 1>

```
#defineCACHE_SHINOBI                                // Normal cache mode
    (SYD_CACHE_FORM_IC_ENABLE | SYD_CACHE_FORM_OC_ENABLE |
SYD_CACHE_FORM_P1_CB )

#defineCACHE_SOFDEC\                               // MPEG Sofdec F/X cache mode
    (SYD_CACHE_FORM_OC_INDEX | CACHE_SHINOBI)

mwPlyInitSofdec(...)_                           // Index cache mode
syCacheInit(CACHE_SHINOBI);                     // Normal mode
/* Polygon rendering */
syCacheInit(CACHE_SOFDEC);                      // Index cache mode
mwPlyStartFname("sample.sfd");                  // Movie playback
```

<Cache control example 2>

```
mwPlyInitSofdec(...);                          // Index cache mode
mwPlyFinishSofdec();                          // Normal mode
/* Polygon rendering */
mwPlyInitSofdec(...);                          // Index cache mode
mwPlyStartFname("sample.sfd");                // Movie playback
```

* The name of the "syCacheInit" function may be changed in Shinobi 2.

1.5. Reducing the CPU Load

The following methods can be used to reduce the CPU load and the frequency of dropped frames.

(1) Simplifying Halfpel processing

Simplifying the Halfpel processing for the B picture can increase speed by about 3 or 4%, with only a very slight deterioration in picture quality.

```
mwPlySetFastHalfpel(ply, ON);
```

(2) Deleting the B picture

During encoding, encode the stream without including the B picture. For example, a stream that does not include the B picture can be created by specifying the parameters N = 6 and M = 1 for decoding. Because decoding the B picture generates a greater load than the other pictures, the CPU load can be reduced by about 6 to 9% by eliminating the B picture.

```
c:\>sfvencd -in=sample.avi -out=sample.m1v -gop_n=6 -gop_m=1
```

(3) Reducing the bit rate

The CPU load increases proportionally to the SFD data bit rate. The CPU load can be decreased by decreasing the data rate.

(4) Adjusting the resolution

The CPU load can be reduced by reducing the resolution. For example, on an NTSC TV, the top and bottom 16 dots are not displayed at 640×480 resolution. As a result, the CPU load can be reduced by about 6% by cropping a 320×480 video to a 320×448 video.

(5) Increasing the number of frame pools

Dropped frames can be reduced by increasing the number of frame pools. Although this method does not reduce the CPU load, it does absorb some of the fluctuation in the CPU load. In MPEG, the CPU load fluctuates considerably, depending on the picture type and the intricacy of the video.

```
cprm.nfrm_pool_wk = 6; // Dropped frames can be reduced by increasing this value  
ply = mwPlyCreateSofdec(&cprm);
```

7. Display type

MPEG Sofdec F/X specifies the display type as shown below as a handle generation parameter. The functions which can be used differ according to the specified display type.

- (1) Automatic display type **MWD_PLY_DTYPE_AUTO**
- (2) Window display type **MWD_PLY_DTYPE_WND**
- (3) Surface display type **MWD_PLY_DTYPE_SRF**

* "MWD_PLY_DTYPE_FULL" is identical to "MWD_PLY_DTYPE_WND". The name has been changed, starting in MPEG Sofdec F/X.

<Sample program>

```
cprm.dtype = MWD_PLY_DTYPE_AUTO;  
ply = mwPlyCreateSofdec (&cprm);
```

Table 7-1 Display type

Display type	Display method	Functions that can be used
Automatic type AUTO	The size of the display area is determined automatically from the video resolution. Adjust the UV value to reduce blurring caused by the bilinear filter.	<code>mwPlySetDispPos</code>
Window type WND	Displays a rectangular window. Set the parameters for the window as a whole. This method can only handle one window.	<code>mwPlySetDispPos</code> <code>mwPlySetDispSize</code> <code>mwPlySetBright</code> <code>mwPlySetBrightOfst</code>
Surface type SRF	Permits independent control of each of the four vertices of a window. This mode is used for multiwindow display.	<code>mwPlySetSrfPos</code> <code>mwPlyGetSrfPos</code> <code>mwPlySetSrfBright</code> <code>mwPlyGetSrfBright</code> <code>mwPlySetSrfBrightOfst</code> <code>mwPlyGetSrfBrightOfst</code> <code>mwPlySetImgPos</code> <code>mwPlyGetImgPos</code>

7.1. Automatic display type

The automatic display type reduces blurring due to the bilinear filter, and automatically adjusts the display size to provide a sharp image. For example, a 320×240 movie will be displayed with 639×479 dots. Therefore, if a background color has been set, it is essential to be aware that one line of dots of the background color will appear on the right edge and the bottom edge of the screen. The following tables show the relationship between the video resolution and the display size.

Table 7-2 Relationship between Video Size and Display Size in the X Direction

Video X size	320	352	480	640	Others
Display X size	639	640	640	640	640

↑ 16 dots are cut off from both left and right.

Table 7-3 Relationship between Video Size and Display Size in the Y Direction

Video X size	224	240	320	336	448	480	Others
Display X size	447	479	320	448	448	480	480

Bilinear filter processing

With bilinear filter processing, if the UV value is set between 0.0 and 1.0, the operation described in the figure below is performed, resulting in an overall blurriness in the video image.

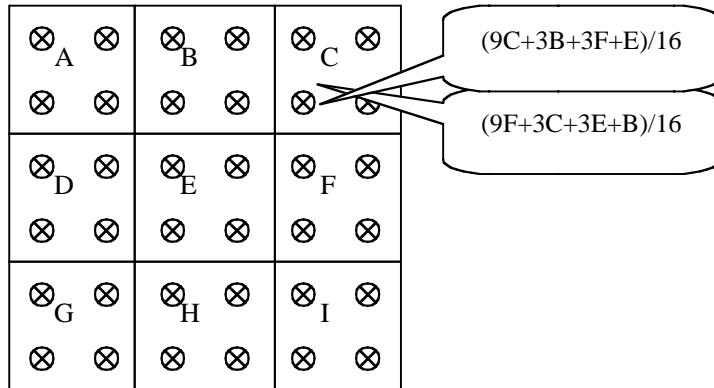


Fig. 7-1 Normal Bilinear Filter Processing

In MPEG Sofdec F/X, when the size of the image is doubled, the UV value is set as shown in the diagram below, making playback with less blurriness possible. Because this method can only generate $(2 \times - 1)$ dots, a 320×240 video is displayed as a 639×479 video.

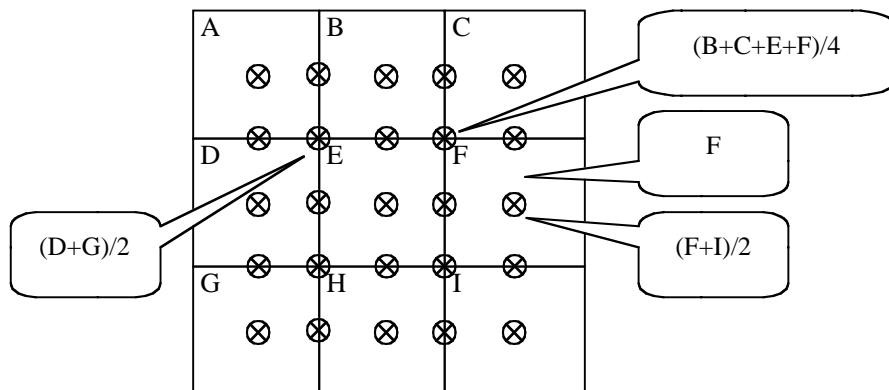


Fig. 7-2 Normal Bilinear Filter Processing in MPEG SofdecF/X

1.2. Window Display type

The window display type can be controlled in units of individual rectangular windows.

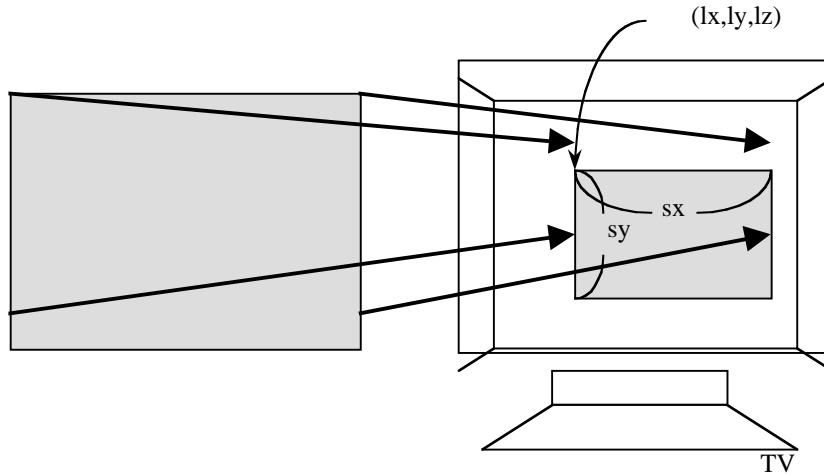


Fig. 7-3 Window Control

<Sample program>

```
ply = mwPlyCreateSofdec(&cprm);
mwPlySetDispPos(ply, lx, ly);                                // Display position setting
mwPlySetDispSize(ply, sx, sy);                               // Display size setting
mwPlySetDispZ(ply, lz);                                     // Depth setting
mwPlySetBright(ply, bright);                                 // Brightness setting
mwPlySetBrightOfst(ply, bofst);                             // Brightness offset setting
mwPlyStartFname(ply, "sample.sfd");
for (;;) {
    if ( Window movement ) {
        mwPlySetDispPos(ply, lx++, ly++);
        mwPlySetDispZ(ply, lz++);
    }
    if ( Window size change ) {
        mwPlySetDispSize(ply, sx++, sy++);
    }
    if ( Brightness change ) {
        mwPlySetBright(ply, bright++);
    }
    if ( Brightness offset change ) {
        mwPlySetBrightOfst(ply, bofst++);
    }
    mwExecMainServer();
    /* Polygon rendering */
    njWaitVSync();
}
```

1.3. Surface Display Type

For the surface display type, the following parameters can be set independently for each of the four vertices of a window. This display type permits video to be displayed in multiple windows.

(1) Display position

The X, Y, and Z coordinates can each be set. The X coordinates and Y coordinates are the frame buffer coordinate values. The coordinates for the upper left corner are (0.0, 0.0), and the coordinates for the lower right corner are (639.0, 479.0). The Z coordinates are also specified, but no perspective conversion is performed on the Z coordinates.

(2) Vertex brightness

These parameters specify the alpha, red, green and blue intensity. Set these values in the range from 1.0 to 0.0. These values are multiplied with the video data. Changing these values from 0.0 to 1.0 produces a "fade in from black" effect, and changing these values from 1.0 to 0.0 produces a "fade out" effect. The alpha value sets the mixing ratio with the background video. Changing these values produces a dissolve effect.

(3) Vertex brightness offset

These parameters specify additive components for the alpha, red, green, and blue values. Set these values in the range from 1.0 to 0.0. The value produced by multiplying this setting by 255 is then added to the video data. Changing these values from 0.0 to 1.0 produces a "fade out to white" effect, and changing these values from 1.0 to 0.0 produces a "fade in from white" effect.

(4) Video position

These parameters specify the position of the video data corresponding to each vertex. Although equivalent to the UV coordinates, the values that are specified are the positions in terms of the pixels corresponding to the video data.

The parameters are set for each vertex in sequence in a "Z" pattern as shown below.

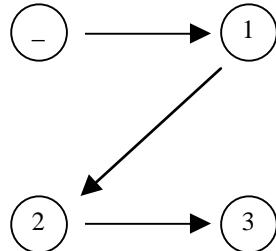


Fig. 7-4 Order of Vertices for Which Parameters Are Set in the Surface Type

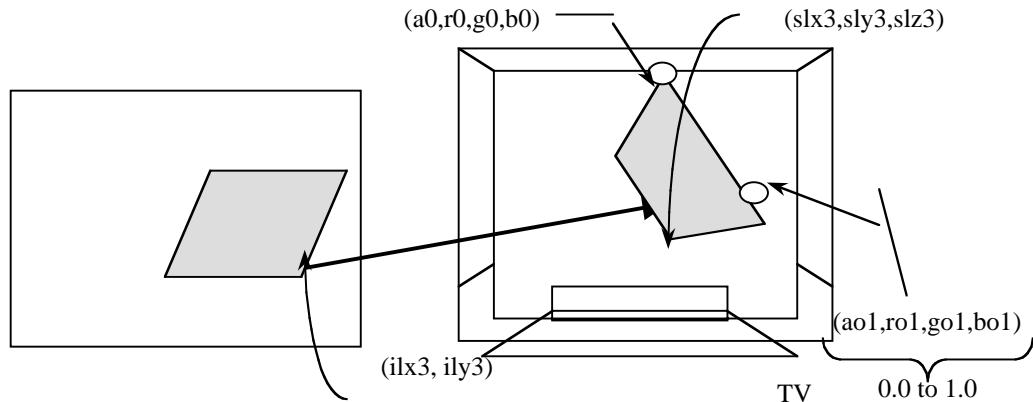


Fig. 7-4 Surface Control

<Sample program>

```

ply = mwPlyCreateSofdec(&cprm);

// Image position setting
mwPlySetImgPos(ply, 0, ilx0, ily0);
mwPlySetImgPos(ply, 1, ilx1, ily1);
mwPlySetImgPos(ply, 2, ilx2, ily2);
mwPlySetImgPos(ply, 3, ilx3, ily3);

// Surface position setting
mwPlySetSrfPos(ply, 0, slx0, sly0, slz0);
mwPlySetSrfPos(ply, 1, slx1, sly1, slz1);
mwPlySetSrfPos(ply, 2, slx2, sly2, slz2);
mwPlySetSrfPos(ply, 3, slx3, sly3, slz3);

mwPlySetSrfBright(ply, 0, a0, r0, g0, b0);           // Brightness setting
mwPlySetSrfBrightOfst(ply, 1, aol, ro1, gol, bol); // Brightness offset setting

mwPlyStartFname(ply, "sample.sfd");
for (;;) {
    mwExecMainServer();
    /* Polygon rendering */
    njWaitVSync();
}

```

8. Multiwindow Playback

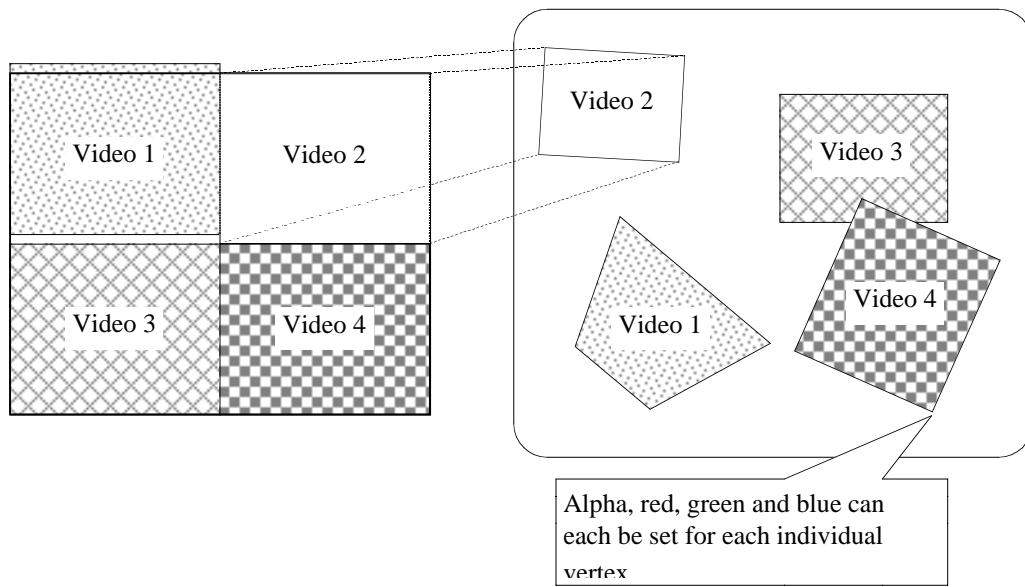


Fig. 8-1 Multiwindow Display

Multiwindow display can be performed through the procedure described below by using the surface display type.

(1) Set surface mode

Set surface mode as the display mode during handle generation.

```
cprm.dtype = MWD_PLY_DTYPE_SRF  
ply = mwPlyCreateSofdec(&cprm);
```

(2) Allocate and set up the buffer for surface points

The "surface points" indicate each vertex of a window, and have the above parameters. A window is described by four surface points. For example, in order to display 25 windows, 100 surface points are needed.

```
size = mwPlyCalcSrfBufSize(ply, 4* 25); // Point buffer size calculation  
_buf = syMalloc(size);  
mwPlySetSrfPntBuf(ply, 4* 25, buf, bsize); // Point buffer settings  
  
/* Settings for 10th window */  
mwPlySetSrfPos(ply, 10*4 + 0, lx0, ly0, lz0); // Pay attention to the point  
numbers  
mwPlySetSrfPos(ply, 10*4 + 1, lx0, ly0, lz0);  
mwPlySetSrfPos(ply, 10*4 + 2, lx0, ly0, lz0);  
mwPlySetSrfPos(ply, 10*4 + 3, lx0, ly0, lz0);
```

9. Composition

9.1. Features

MPEG Sofdec F/X includes composition functions for visual effects.

(1) Composition of a movie with alpha and polygons

These functions can display a movie with alpha in front of polygons, and permit composition at the pixel level.

(2) Support for various composition modes (translucent composition/additive composition/luminance key composition/alpha composition)

These functions support multiple composition modes, depending on circumstances. The load on the CPU and the details of the functions vary according to the different modes.

(3) Permits switching the composition mode during playback

The composition mode can be switched while a movie is in progress. This capability allows you to select the appropriate composition mode for each cut of a movie.

(4) Fade in/out capability

An alpha value can be assigned for a video as a whole. This makes it possible to fade video in and out without increasing the CPU load. Use this function to initiate a visual effect smoothly.

(5) AVI file conversion using a dedicated tool

Use CRI MPEG CRAFT to create movie data for composition purposes from an AVI file with alpha.

9.2. Composition Functions

MPEG SofdecF/X offers composition functions for implementing visual effects. The following composition methods are available:

(1) Opaque display

Renders video without composition

(2) Translucent composition (alpha composition for entire windows)

Alpha composition in individual windows.

(3) Additive composition

Simply adds the RGB values of a polygon to the RGB values of the movie.

(4) Alpha composition (alpha composition for individual pixels)

Alpha composition at the individual pixel level of polygon RGB values and movie RGB values.

Table 9-1 Resources Used by Sofdec F/X and the Load

	Resources used	Rendering load	Increase in CPU load
Translucent/additive composition	None	One layer translucent	None
Alpha composition	256-color palette (ARGB8888)	Two layers translucent	5 to 10%

<What is "alpha composition"?>

"Alpha composition" is expressed as follows by the mixing value " α ":

```

output =  $\alpha \cdot - + (-\alpha) \cdot -$ 
output : RGB value after composition
 $\alpha$  : Mixing ratio
- : Movie RGB value
- : Polygon RGB value
  
```

- * The "polygon RGB value" is the RGB value in the frame buffer after the object that is farthest in the background from the movie has been rendered.

1.3. Alpha Composition

Alpha composition at the individual pixel level is implemented by rendering two translucent polygons as shown below. After rendering the mask surface (multiplication processing), the video surface is added.

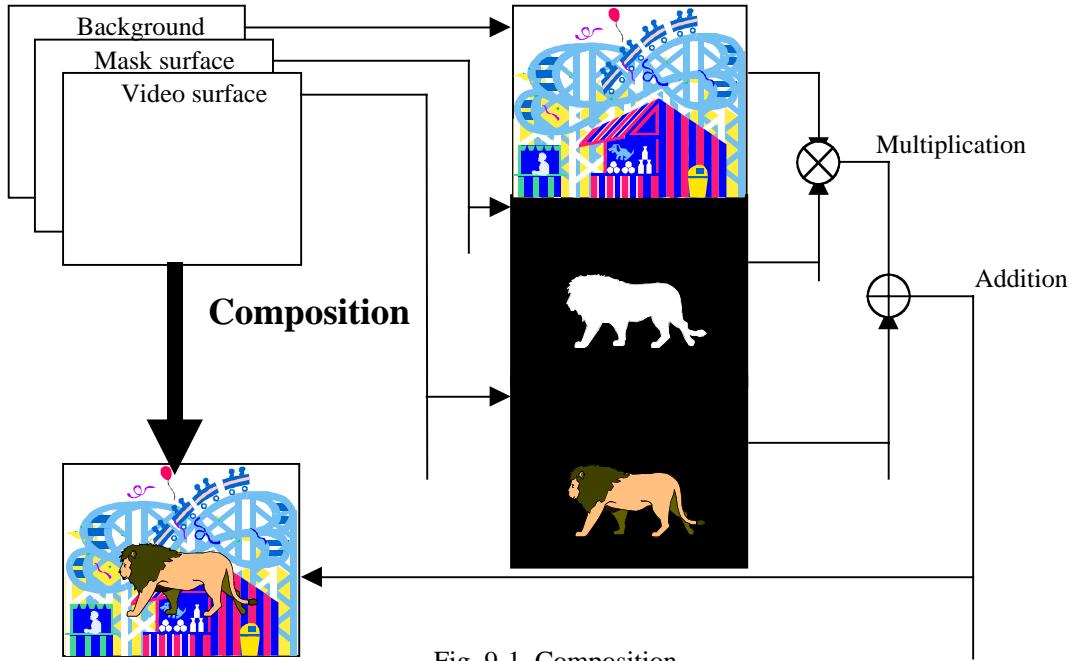


Fig. 9-1 Composition

In Sofdec F/X, the following two texture surfaces can be gotten from the handle.

- (1) **Video surface**
A rectangular-format YUV422 texture. Stores the decoded video.
- (2) **Mask surface**
A twiddled-format 8-bit/pixel palette texture. Stores the mask image.

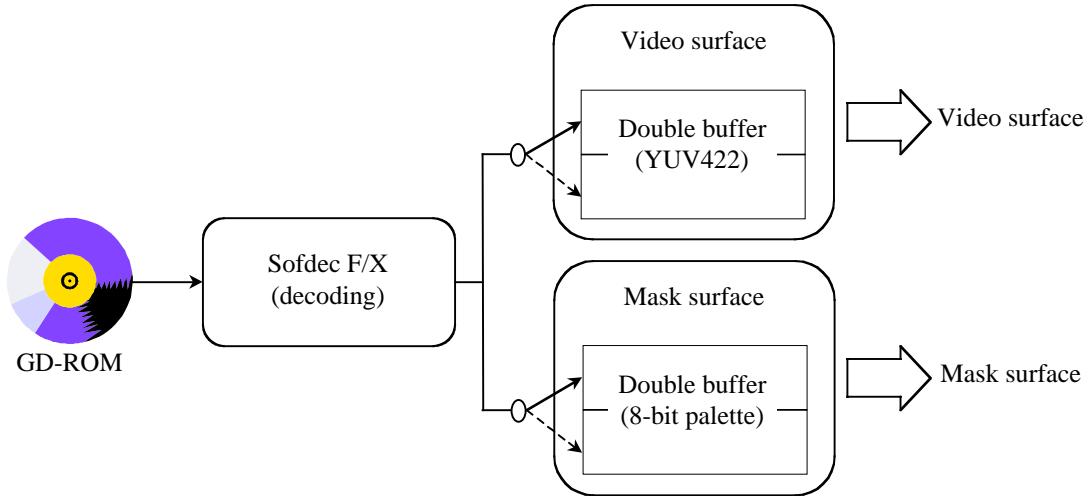


Fig. 9-2 Sofdec F/X Texture Surface

1.4. Composition mode

MPEG Sofdec F/X offers the following composition modes. The mode is set through the "cprm.compo_mode" member during handle generation.

```
cprm.compo_mode = MWD_PLY_COMPO_~; (Composition mode)
ply = mwPlyCreateSofdec(&cprm);
```

- (1) **Opaque display mode (MWD_PLY_COMPO_OPEQ)**

(2) Translucent composition mode (MWD_PLY_COMPO_TRNSP)

In this mode, a movie is displayed as a translucent polygon. Although this mode does not permit composition at the pixel level, it does permit alpha composition for the window as a whole.

(3) Additive composition mode (MWD_PLY_COMPO_ADD)

This mode adds the movie as a whole. As a result, black portions appear to be completely removed. This mode is intended for light effects.

(4) Luminance composition mode (MWD_PLY_COMPO_LUMI)

This mode changes the alpha value according to the luminance. The darker an element is, the more transparent it becomes; brighter elements become opaque. The default settings are shown below. The "mwPlySetAlphTbl" function can set up the conversion table from luminance to alpha value.

0 to 15: Completely transparent

16 to 100: Alpha value gradually increases from 0.0 (transparent) to 1.0 (opaque)

100 to 255: Completely opaque

(5) Three-step alpha composition mode (MWD_PLY_COMPO_ALPH3)

This mode permits three levels of composition: opaque, translucent (alpha value of 50%), and transparent. This mode produces a picture quality with a quantization bit number of about 7 bits.

(6) Five-step alpha composition mode (MWD_PLY_COMPO_ALPH5)

This mode permits five levels of composition: transparent (0%), 25%, 50%, 75%, and opaque (100%). However, the video quality suffers.

(7) Full alpha composition mode (MWD_PLY_COMPO_ALPH256)

This mode permits composition with 256-level alpha values. This mode entails a CPU load and work area that is almost twice as great as that of other modes. This mode is suited for small movies.

(8) Mixed composition mode (MWD_PLY_COMPO_MIX)

This mode permits switching among composition modes other than full alpha composition mode according to the "mwPlySetCompoMode" function.

The following table shows the resources that are used, the CPU load, and the rendering load for each composition mode. This is meant only as a guideline; the actual details will vary according to the movie data.

<Conditions for video>

Resolution : 256×256
 Data rate : 300 Kbyte/sec
 GOP pattern : N=6, M=1

Table 9-2 Resources Used and Load for Each Composition Mode

Composition mode	CPU load	Rendering load	Texture	Palette
Opaque display	40%	One layer opaque	YUV422	Not used
Translucent	40%	One layer opaque	YUV422	Not used
Additive composition	40%	One layer opaque	YUV422	Not used
Luminance composition	50%	Two layers translucent	YUV422 PLT8	Used
Three-step alpha composition	50%	Two layers translucent	YUV422 PLT8	Used
Five-step alpha composition	50%	Two layers translucent	YUV422 PLT8	Used
Full alpha composition	65%	Two layers translucent	YUV422 PLT8	Used
Mixed composition			YUV422 PLT8	Used

<Resources used>

YUV422 texture : 256Kbyte (256×256 , 2 layers)
 8-bit palette texture : 128Kbyte (256×256 , 2 layers)
 Palette : ARGB8888, 256 entries, starting from the 768th

1.5. Additive Composition

In additive composition, the image data from a movie is simply added to the displayed video. This composition mode is intended for implementing light-related effects.

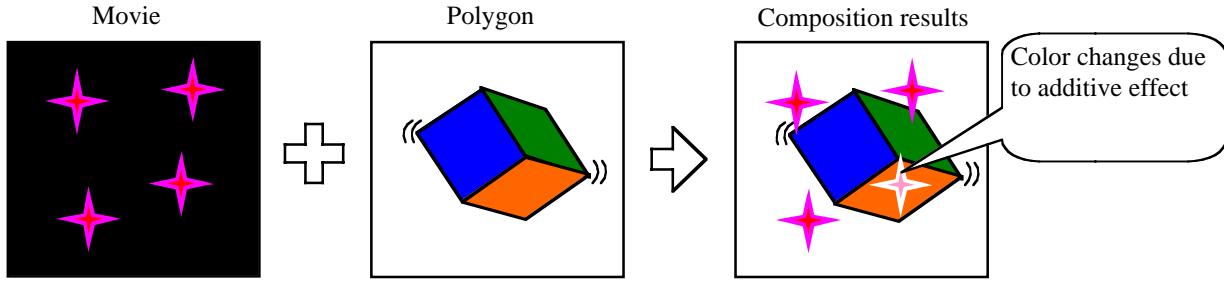


Fig. 9-3 Additive Composition

<Sample program>

```
cprm.compo_mode = MWD_PLY_COMPO_ADD;           // Additive composition mode
:
ply = mwPlyCreateSofdec(&cprm);
mwPlyStartFname(ply, "effect.sfd");
for (;;) {
    mwExecMainServer();
    (Polygon rendering)
    njWaitVSync();
}
```

1.6. Luminance Key Composition

Luminance key composition uses a mixing ratio that varies in accordance with the luminance. This makes it possible to cut out black portions of an image. This mode is effective for effects that have no black portions, such as flames, smoke, lightning, etc.

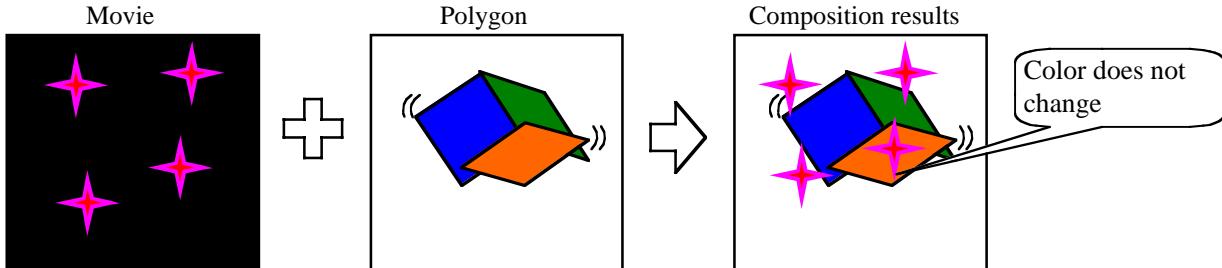


Fig. 9-4 Luminance key composition

<Sample program>

```
cprm.compo_mode = MWD_PLY_COMPO_LUMI; // Luminance key composition mode
:
ply = mwPlyCreateSofdec(&cprm);
mwPlyStartFname(ply, "effect.sfd");
:
```

1.7. Three-step Alpha Composition

Three-step alpha composition permits the use of three levels of alpha values: opaque, translucent, and transparent. Because the alpha information is embedded in the video data, the video quality is equivalent to 7-bit quality. This mode is suitable for animation cell-like composition. It can also be used for composition involving black images that will not work with luminance composition.

<Sample program>

```
cprm.compo_mode = MWD_PLY_COMPO_ALPH3; // Three-step alpha composition mode  
:  
ply = mwPlyCreateSofdec(&cprm);  
mwPlyStartFname(ply, "effect.sfd");  
:
```

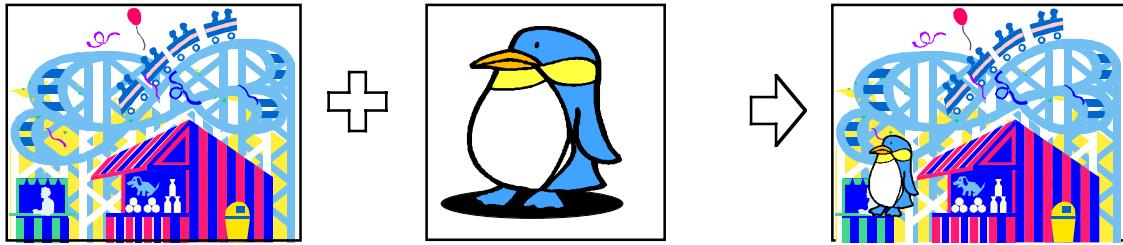


Fig. 9-5 Three-step alpha composition

1.8. Five-step Alpha Composition

Five-step alpha composition permits the use of five levels of alpha values: 0% (transparent), 25%, 50%, 75%, and 100% (opaque). Although this mode can be used for visual effects that require translucence, the alpha information is embedded in the video data, which means that the video quality is equivalent to 5-bit quality.

<Sample program>

```
cprm.compo_mode = MWD_PLY_COMPO_ALPH5; // Five-step alpha composition mode  
:  
ply = mwPlyCreateSofdec(&cprm);  
mwPlyStartFname(ply, "effect.sfd");
```

1.9. 256-step Alpha Composition

You can also use 256-step alpha composition. Although this mode provides composition with the highest image quality, it increases the load on the CPU. This mode also requires more memory resources than the other composition modes.

1.10. Switching Composition Modes

It is possible to switch composition modes for each cut in a movie. All composition modes except for 256-value composition mode can be switched. Set the composition mode to MW_PLY_COMPO_MIX when generating the handle.

Although the composition mode can be changed by using the mwPlySetCompoMode function, execute this function within the effect callback function. This callback function is called immediately before mask generation. Because the frame number is passed as a parameter, change the composition mode in accordance with this number.

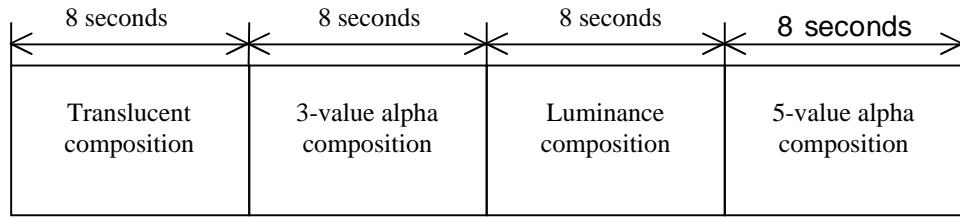


Fig. 9-6 Switching Composition Modes

<Example of Switching Composition Modes>

```
void user_chg_compo(void *obj, Sint32 fno, Sint32 time, Sint32 tunit)
{
    MWPLY ply= (MWPLY) obj;

    if ( fno == 0 ) {
        mwPlySetCompoMode(ply, compo_mode=MWD_PLY_COMPO_TRNSP);
    } else if ( fno == 1*8*30 ) {
        mwPlySetCompoMode(ply, compo_mode=MWD_PLY_COMPO_LUMI);
    } else if ( fno == 2*8*30 ) {
        mwPlySetCompoMode(ply, compo_mode=MWD_PLY_COMPO_ALPH3);
    } else if ( fno == 3*8*30 ) {
        mwPlySetCompoMode(ply, compo_mode=MWD_PLY_COMPO_ALPH5);
    }
}

void main(void)
{
    cprm.compo_mode = MWD_PLY_COMPO_MIX;
    ply = mwPlyCreateSofdec(&cprm);
    mwPlyEntryFxCb(ply, user_chg_compo, (void *)ply);
    mwPlyStartFname(ply, "vfx01.m1v");
}
```

1.11. Fade In/Out

A movie can be faded in and out by manipulating the alpha values for each vertex of the display window. Specify a composition mode other than opaque (MWD_PLY_COMPO_OPEQ). Furthermore, the display type must be "surface display."

<Sample program>

```
cprm.dtype = MWD_PLY_COMPO_TRNSP;      // Translucent composition mode
cprm.dtype = MWD_PLY_DTYPE_SRF;          // Surface display type
ply = mwPlyCreateSofdec(&cprm);
mwPlyStartFname(ply, "vfx01.m1v");
for (;;) {
    njWaitVsync();
    if ( per->on & PDD_DGT_KU ) {
        a += 1.0f/60.0f;           // One-second fade-in
        a = ( a < 1.0f ) ? a : 1.0f;
        for ( i=0; i<4; i++ )
            mwPlySetSrfBright(ply, i, a, 1.0f, 1.0f, 1.0f); // Alpha value
    settings
    }
    if ( per->on & PDD_DGT_KD ) {
        a -= 1.0f/60.0f;           // One-second fade-out
        a = ( a > 0.0f ) ? a : 0.0f;
        for ( i=0; i<4; i++ )
            mwPlySetSrfBright(ply, i, a, 1.0f, 1.0f, 1.0f); // Alpha value
    settings
    }
    mwExecMainServer();
}
```

10. Texture Movies

The MPEG Sofdec F/X allocates video texture surfaces and mask texture surfaces using the Kamui driver. The user can get the surfaces from each library's handle, and can apply a video or mask to the surface of a user object.

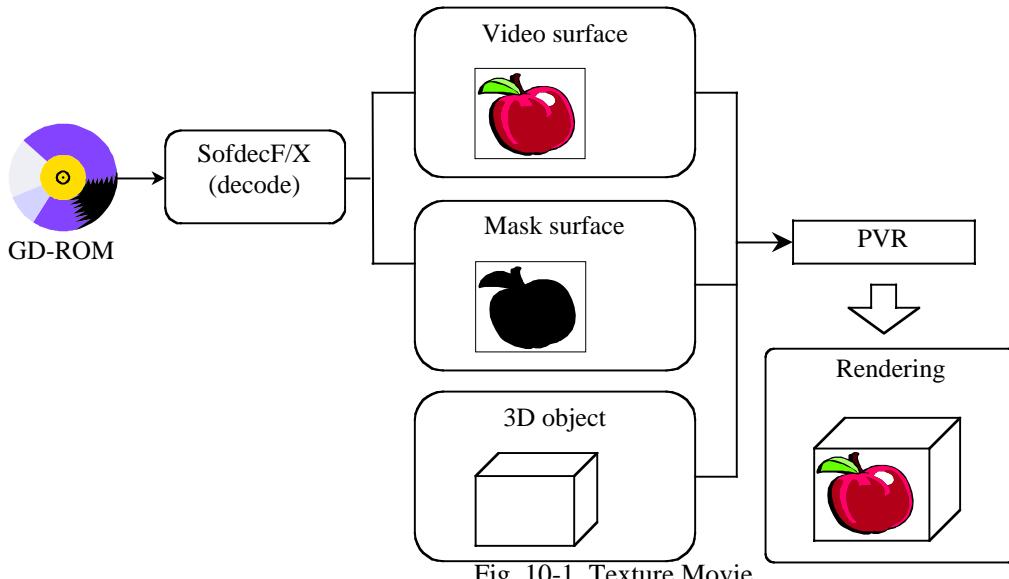


Fig. 10-1 Texture Movie

A normal movie with no mask can be played back as a texture movie just by applying it to a polygon surface. In the case of a movie with a mask, however, it is necessary to apply two polygon surfaces beforehand.

Inside (mask surface): with alpha (FBS_SA|FBD_ISA)

SRCBlendingMode=SRCALPHA, DSTBlendingMode=INVSRCALPHA

Outside (video surface): addition (FBS_SA|FBD_ONE)

SRCBlendingMode=SRCALPHA, DSTBlendingMode=ONE

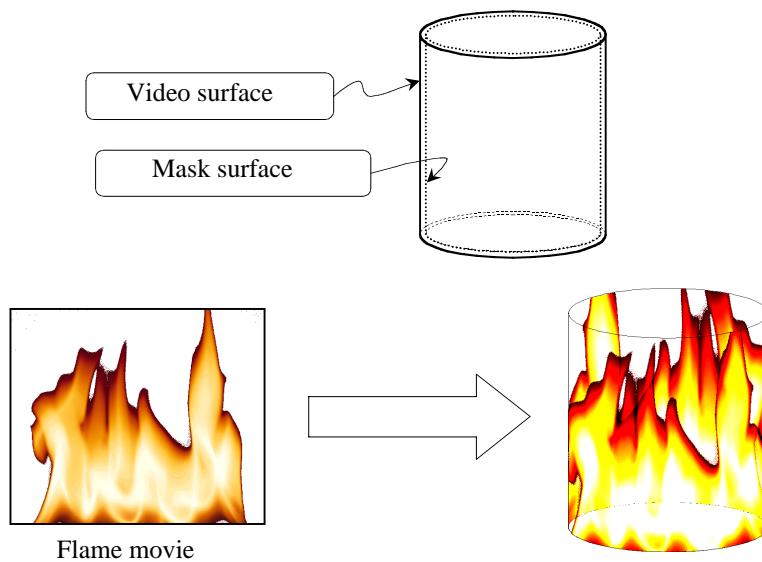


Fig. 10-2 Texture Movie with Mask

<Getting a surface through MPEG Sofdec F/X>

```
mwExecMainServer(); // Transfer to texture RAM
mwPlyGetMvFrm(ply, &frm); // Getting the video surface
user_set_video_surface(frm.srf); // Applying the video surface
mwPlyGetMskFrm(ply, &frm); // Getting the mask surface
user_set_msk_surface(frm.srf); // Applying the mask surface
/* Polygon rendering */
```

<Texture movie through the Ninja library>

The "njSetMvSurface" function can be used to register a movie surface in a texture with a number specified in the texture list. The "njSetMvSurface" function is defined in the middleware sample program.

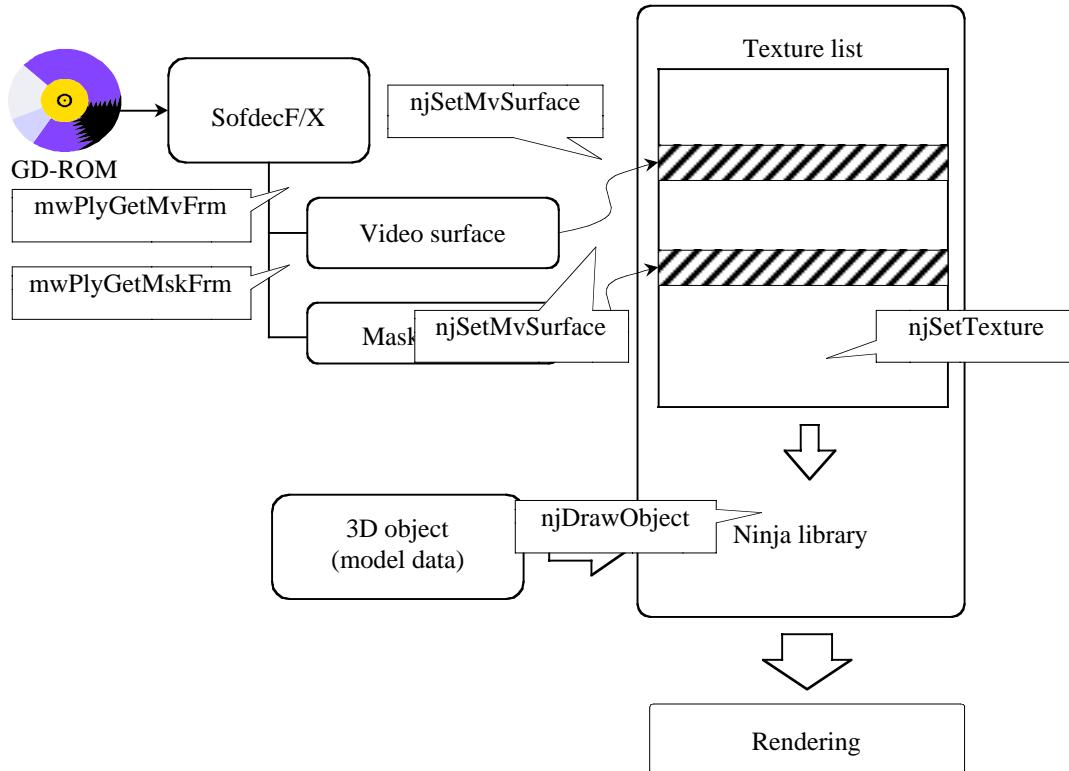


Fig. 10-3 Texture Movie Through the Ninja Library

11. Multistreaming

11.1. Mechanism

MPEG Sofdec F/X is capable of simultaneously playing multiple movie files stored on GD-ROM in separate files. Because this function uses the ADX multistreaming function, it is necessary to link ADX.

The following description is an example of how to generate multiple handles and then play a movie for each handle asynchronously.

```
ply1 = mwPlyCreateSofdec(&cprm1);
ply2 = mwPlyCreateSofdec(&cprm2);
for (;;) {
    if ( The A button was pressed )
        mwPlyStartFname(ply1, "movie1.sfd");
    if ( The B button was pressed )
        mwPlyStartFname(ply2, "movie2.sfd");
}
```

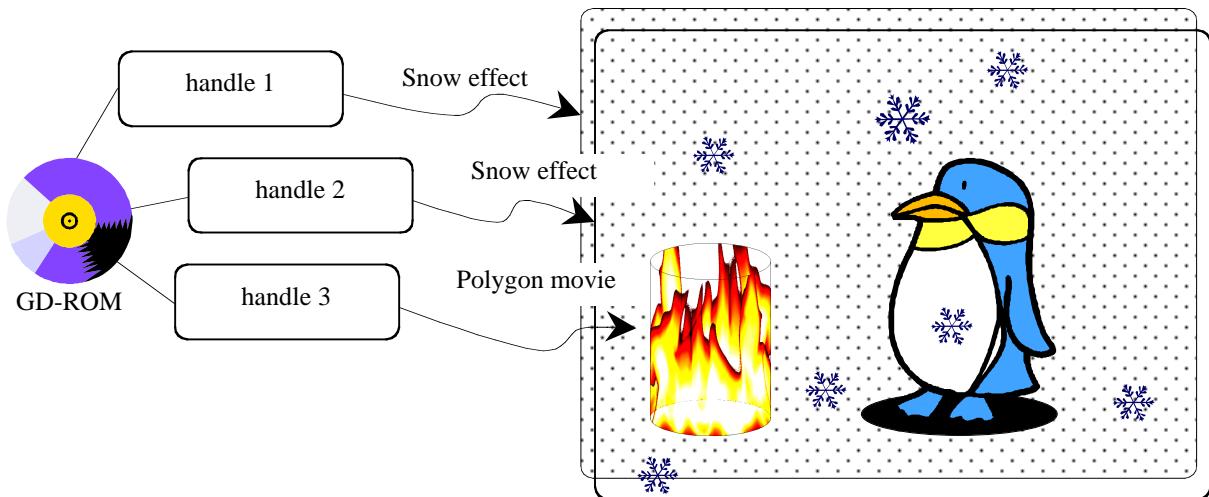


Fig. 11-1 Multistreaming Playback

11.2. Buffer Settings

When using multistreaming, the size of the input buffer must be increased. The size of the input buffer is determined by the "max_bps" generation parameter. For example, if two files will be played back simultaneously, "max_bps" must be doubled.

```
#define NSTM(3)           // Number of stream to be played back simultaneously
#define MAX_BPS1(150*1024*8) // Maximum bit rate1 (150Kbyte/sec)
#define MAX_BPS2(100*1024*8) // Maximum bit rate2 (100Kbyte/sec)
#define MAX_BPS3(75*1024*8)  // Maximum bit rate3 (75Kbyte/sec)

// Handle generation 1 (for 150Kbyte/sec)
cprm1.max_bps = MAX_BPS1*NSTM;
cprm1.wksize = mwPlyCalcWorkCprmSfd(&cprm);
cprm1.work = syMalloc(cprm1.wksize);
ply1 = mwPlyCreateSofdec(&cprm1);

// Handle generation 2 (for 100Kbyte/sec)
cprm2.max_bps = MAX_BPS2*NSTM;
cprm2.wksize = mwPlyCalcWorkCprmSfd(&cprm2);
cprm2.work = syMalloc(cprm2.wksize);
ply2 = mwPlyCreateSofdec(&cprm2);

// Handle generation 3 (for 75Kbyte/sec)
cprm3.max_bps = MAX_BPS3*NSTM;
cprm3.wksize = mwPlyCalcWorkCprmSfd(&cprm3);
cprm3.work = syMalloc(cprm3.wksize);
ply3 = mwPlyCreateSofdec(&cprm3);
```

12. Seamless Continuous Playback

12.1. Seamless Continuous Playback of Multiple Files

MPEG Sofdec F/X is capable of seamlessly and continuously playing multiple, separate movie files. It is also able to repeatedly play the same file. Continuous playback is possible for up to 9 hours. This feature requires the CRI ADX library.

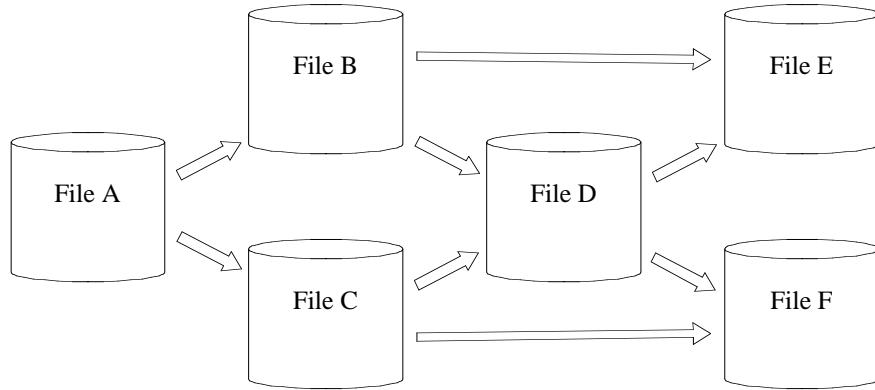


Fig. 12-1 Seamless Continuous Playback

Seamless continuous playback is possible by following the procedure described below.

- (1) Register the files to be played back. ("mwPlyEntryFname" function)
- (2) Start seamless continuous playback. ("mwPlyStartSeamless" function)
- (3) Additional files can still be registered for continuous playback after playback has begun.
- (4) Cancel seamless continuous playback mode. ("mwPlyReleaseSeamless" function)
- (5) The system ends in the playback end state.

Up to eight files can be registered. The next file should be registered at least one second before playback of the last movie file ends.

```
ply = mwPlyCreateSofdec(&cprm);
mwPlyEntryFname(ply, "movie1.m1v");
mwPlyEntryFname(ply, "movie2.m1v");
mwPlyStartSeamless(ply);
for (;;) {
    if ( The A button was pressed )
        mwPlyEntryFname(ply, "movie3.m1v");
    if ( The B button was pressed )
        mwPlyReleaseSeamless(ply);
    if ( mwPlyGetStat(ply) == MWE_PLY_STAT_PLAYEND )
        break;
}
```

12.2. Seamless Loop Playback

The mwPlyStartFnameLp function can be used for simple seamless loop playback of a single file.

```
ply = mwPlyCreateSofdec(&cprm);
mwPlyStartFnameLp(ply, "movie1.m1v");
```

13. Data Specifications

The Library data list is shown below.

Table 13-1 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
MWD_PLY_COMPO~	Composition mode	1.4
Data Types		
MWPLY	Middleware playback handle	2.1
MWS_PLY_INIT_SFD	Initialization parameter structure	2.2
MWS_PLY_CPRM_SFD	Handle creation parameter structure	2.3
MWS_PLY_MVFRM	Video surface information structure	2.4

13.1. 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

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

The following constants indicate the type of file to play for MPEG Sofdec F/X.

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 display type of video.

Constant Name	Description
MWE_PLY_DTYPE_AUTO	Display size of screen is automatically adjusted. 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. This constant is retained in order to preserve compatibility with Ver. 1.
MWE_PLY_DTYPE_WND	Adjust the display position and size according to the window level This constant is set by the mwPlySetDispPos and mwPlySetDispSize functions.
MWE_PLY_DTYPE_SRF	Adjust the display position and size according to the surface level Use the mwPlySetSrf~ functions to set the display position and brightness for each vertex.

Title	Data Name	Data	No
Data	MWD_PLY_COMPO~	Composition mode	1.4

The following constants indicate the composition mode.

Constant Name	Description
MWD_PLY_COMPO_OPEQ	Opaque
MWD_PLY_COMPO_TRNSP	Transparent
MWD_PLY_COMPO_ADD	Additive composition
MWD_PLY_COMPO_LUMI	Luminance key composition
MWD_PLY_COMPO_ALPH3	3-step alpha composition
MWD_PLY_COMPO_ALPH5	5-step alpha composition
MWD_PLY_COMPO_ALPH256	Full alpha composition
MWD_PLY_COMPO_MIX	Mixed composition mode Mixes multiple composition modes in one stream.

1.2. Data Type

Title	Data Name	Data	No
Data	MWPLY	Middleware playback handle	2.1

Handle to control playing of MPEG Sofdec F/X.

Title	Data Name	Data	No
Data	MWS_PLY_INIT_SFD	Initialization parameter structure	2.2

Parameter structure for setting the initialization function for MPEG SofdecF/X. Set the same values as in the display parameters that were actually set during Ninja initialization.

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

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

Member	Type	Description
ftype	Sint32	Type of file to play Select from MWE_PLY_FTYP~
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~
compo_mode	Sint32	Composition mode Can be selected from MWD_PLY_COMP~

Title	Data Name	Data	No
Data	MWS_PLY_MVFRM	Video surface information structure	2.4

This is the movie surface information that is gotten when playing a texture movie.

Member	Type	Description
srf	void*	Texture surface
width	Sint32	Valid surface width (unit: pixels)
height	Sint32	Valid surface height (unit: pixels)

14. Function Specifications

The library functions are listed below.

Table 14-1 Function List

Function Name	Function	No.
Initialization and end processing		
mwPlyPreInitSofdec	Set up system initialization	1.1
mwPlyInitSofdec	Library initialization (for Ninja)	1.2
mwPlyFinishSofdec	Library end processing (for Ninja)	1.3
mwPlyInitSfdFx	Library initialization (for Kamui, Kamui 2)	1.4
mwPlyFinishSfdFx	Library end processing (Kamui, Kamui 2)	1.5
mwPlySetDispMode	Set up the display mode	1.8
mwPlySetVertexBuffer	Vertex buffer setting	1.9
Server function		
mwExecMainServer	Server function	2.1
mwPlyExecTexSvr	Texture transfer server functions (for Kamui 2)	2.2
mwPlyExecDrawSvr	Video rendering server functions (for Kamui 2)	2.3
Basic operation processing		
mwPlyCreateSofdec	Handle generation	3.1
mwPlyCalcWorkCprmSfd	Work area size calculation	3.2
mwPlyDestroy	Destroy a handle	3.3
mwPlyStartFname	Start playing (playback of GD-ROM)	3.4
mwPlyStartSj	Start playing (playback of stream joint)	3.5
mwPlyStartMem	Start playing (playback of memory)	3.6
mwPlyStop	Stop playing	3.7
mwPlyGetStat	Get the handle status	3.8
mwPlyGetTime	Get the number of the playing sample	3.9
mwPlyGetInputSj	Input stream joint acquisition	3.10
mwPlyPause	Perform pause setting	3.11
Audio output control		
mwPlySetOutVol	Perform volume setting	4.1
mwPlyGetOutVol	Get the volume's value	4.2
mwPlySetOutPan	Set up panpot	4.3
mwPlyGetOutPan	Get the panpot value	4.4

Table 14-2 Function List

Function Name	Function	No.
Window control		
mwPlySetDispPos	Set up the display position	5.1
mwPlySetDispSize	Set up the display size	5.2
mwPlySetBright	Set up the brightness	5.3
mwPlyGetBright	Get the brightness	5.4
mwPlySetBrightOfst	Set up the brightness offset	5.5
mwPlyGetBrightOfst	Get the brightness offset	5.6
mwPlySetDispZ	Set up the display screen depth value	5.7
mwPlyGetDispZ	Get the display screen depth value	5.8
Surface control		
mwPlyCalcSrfBufSize	Surface pointer buffer size calculation	6.1
mwPlySetSrfPntBuf	Surface point buffer setting	6.2
mwPlySetSrfPos	Set up the display position	6.3
mwPlyGetSrfPos	Display position acquisition	6.4
mwPlySetSrfBright	Set up the brightness	6.5
mwPlyGetSrfBright	Get the brightness	6.6
mwPlySetSrfBrightOfst	Set up the brightness offset	6.7
mwPlyGetSrfBrightOfst	Get the brightness offset	6.8
mwPlySetImgPos	Image position setting	6.9
mwPlyGetImgPos	Image position acquisition	6.10
mwPlyGetImgSize	Image size acquisition	6.11
Effects		
mwPlySetCompoMode	Composition mode setting	7.1
mwPlyEntryFxCb	Effect callback function registration	7.2
mwPlySetAlphTbl	Setup of table for conversion from luminance to alpha value	7.3
Texture movie		
mwPlyGetMvFrm	Movie frame acquisition	8.1
mwPlyGetMskFrm	Mask frame acquisition	8.2
Seamless continuous playback		
mwPlyEntryFname	Seamless continuous playback file registration	9.1
mwPlyStartSeamless	Seamless continuous playback start	9.2
mwPlyReleaseSeamless	Seamless continuous playback cancel	9.3
mwPlySetLpFlg	Seamless loop playback setup	9.4
mwPlyStartFnameLp	Seamless loop playback start	9.5

Table 14-3 Function List

Function Name	Function	No.
Playback mode setting		
mwPlySetFastHalfpel	Set up the fast half pel process	10.1
mwPlySetAudioSw	Set up the audio output switch	10.2
mwPlySetVideoSw	Set up the video display switch	10.3
Internal status acquisition		
mwPlyGetNumDropFrm	Get the number of dropped frames	11.1
mwPlyGetNumDecPool	Acquisition of number of decoded frames	11.2
mwPlyGetNumTotalDec	Acquisition of total number of pictures decoded	11.3
mwPlyGetNumTotalSkip	Acquisition of total number of pictures skipped	11.4
mwPlyGetNumIbufRoom	Acquisition of free space in input buffer	11.5
mwPlyGetNumDispWait	Acquisition of number of frames waiting to be displayed	11.6
mwPlyGetNumRend	Acquisition of number of frames being rendered	11.7
mwPlyGetNumLoadFrm	Acquisition of number of frames transferred to texture memory	11.8
Error processing		
mwPlyEntryErrFunc	Register a function to call on error	12.1

14.1. Initialization and end processing

These functions are used for initialization and end processing.

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

[Syntax]	void mwPlyPreInitSofdec(void);
[Input]	None
[Output]	None
[Rtn Val]	None
[Purpose]	<p>Set up interrupt stack. The interrupt stack size is 16 Kbytes.</p> <p>If this function is executed, the stack is switched when interrupt processing is executed. This function must be executed before the sbInitSystem function.</p>

Title Function	Function Name mwPlyInitSofdec	Function Library initialization (for Ninja)	No 1.2
-------------------	----------------------------------	--	-----------

[Syntax]	void mwPlyInitSofdec(MWS_PLY_INIT_SFD *iprm);
[Input]	iprm : Initialization parameter
[Output]	None
[Rtn Val]	None
[Purpose]	<p>This function initializes the library. This function switches operand cache mode to index cache mode. When using Ninja, use this function in tandem with the mwPlyFinishSofdec function. This function sets up all types of server functions. The different types of server functions are listed below.</p> <ul style="list-style-type: none"> (1) Main server Executed within the main routine when mwPlyExecServer is called. (2) V-Sync server Executed at V-Sync interrupt. (3) Idle server In njWaitVSync, this server is executed during the wait until the next game frame. The MPEG Sofdec library decodes movies within this server.
[Remarks]	Set the same value as the one set in the display mode of the initialization parameter in Ninja.
[Example]	<pre>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);</pre>

Title Function	Function Name mwPlyFinishSofdec	Function Library end processing (for Ninja)	No 1.3
-------------------	------------------------------------	--	-----------

[Syntax] void mwPlyFinishSofdec(void);
 [Input] None
 [Output] None
 [Rtn Val] None
 [Purpose] This function performs the library end processing. This function restores the Shinobi default setting for the cache mode that was switched by the mwPlyInitSofdec function.
 When using Ninja, use this function in tandem with the mwPlyInitSofdec function.

Title Function	Function Name mwPlyInitSfdFx	Function Library initialization (for Kamui, Kamui 2)	No 1.4
-------------------	---------------------------------	--	-----------

[Syntax] void mwPlyInitSfdFx(MWS_PLY_INIT_SFD *iprm);
 [Input] iprm : Initialization parameter
 [Output] None
 [Rtn Val] None
 [Purpose] This function initializes the library. This function switches operand cache mode to index cache mode. When using Kamui, use this function in tandem with the mwPlyFinishSfdFx function.
 Also, be certain to use the mwPlySetVertexBuffer function to set the vertex buffers for rendering.
 This function sets up all types of server functions. The different types of server functions are listed below.

- (1) Main server
 Executed within the main routine when mwPlyExecServer is called.
- (2) V-Sync server
 Executed at V-Sync interrupt.
- (3) Idle server
 In njWaitVSync, this server is executed during the wait until the next game frame. The MPEG Sofdec library decodes movies within this server.

[Remarks] Set the same value as the one set in the display mode of the initialization parameter in Kamui.
 [Example] A usage example is shown below.

```
sbInitSystem(SYS_MODE, SYS_FRAME, SYS_COUNT);
iprm.mode = SYS_MODE;
iprm.frame = SYS_FRAME;
iprm.count = SYS_COUNT;
iprm.latency = 2;
mwPlyInitSofdec(&iprm);
mwPlySetVertexBuffer(vbuf);
```

Title	Function Name	Function	No
Function	mwPlyFinishSfdFx	Library end processing (for Kamui, kamui 2)	1.5

[Syntax] void mwPlyFinishSfdFx(void);
 [Input] None
 [Output] None
 [Rtn Val] None
 [Purpose] Finishes library processing. This function restores the Shinobi default setting for the cache mode that was switched by the mwPlyInitSfdFx function.
 When using Kamui, use this function in tandem with the mwPlyInitSfdFx function.

Title	Function Name	Function	No
Function	mwPlySetDispMode	Set up the display mode	1.6

[Syntax] void mwPlySetDispMode(Sint32 mode, Sint32 frame, Sint32 count, Sint32 latency);
 [Input] mode : Screen mode
 frame : Frame buffer color mode
 count : Frame count number
 latency : Display latency
 [Output] None
 [Rtn Val] None
 [Purpose] This function sets the display mode.
 [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] (1) Notes regarding frame count number:
 ① The display update V-Sync number is 2*count when there is interlacing.
 For example, if the frame count is 2 with interlace, 4V is displayed.
 ② 1V smoothly displays optional frame rate animation, such as 24fps.
 ③ 1V smoothly displays 29.97fps movie for NTSC when playback is on PAL.
 ④ 2V display gives more CPU time to MPEG SofdecF/X and the danger of dropped frames is reduced, but the frame rate for smooth playback is limited.
 (NTSC: 29.97 fps, VGA: 30 fps, PAL: 25fps).
 (2) Latency
 When playing back in 320 × 240 mode, 500K of texture area is used with 2V latency, and 750K of texture area is used with 3V latency.

Title Function	Function Name mwPlySetVertexBuffer	Function Vertex buffer setting	No 1.7
-------------------	---------------------------------------	-----------------------------------	-----------

[Syntax] void mwPlySetVertexBuffer (void *vbuf);
[Input] vbuf : Vertex buffer
[Output] None
[Rtn Val] None
[Purpose] This function sets a vertex buffer. When using Kamui, this function must be used to set up a vertex buffer to be used for rendering. When using Ninja, this function is not needed.

14.2. Server function

These functions are called regularly within the main routine.

Title	Function Name	Function	No
Function	mwExecMainServer	Server function	2.1

[Syntax]	void mwExecMainServer(void);
[Input]	None
[Output]	None
[Rtn Val]	None
[Purpose]	This function transfers textures and renders video. This function is equivalent to the earlier mwPlyExecServer function. This function is used both in playback middleware and recording middleware. In order to avoid multiple calls of the internal module, this function is a common function. An application should only call this function once per game frame. For the sake of compatibility, the mwPlyExecServer function is supported, but should not be used.

Title	Function Name	Function	No
Function	mwPlyExecTexSvr	Texture transfer server functions (for Kamui 2)	2.2

[Syntax]	void mwPlyExecTexSvr(void);
[Input]	None
[Output]	None
[Rtn Val]	None
[Purpose]	This function transfers texture data to video memory. This function is intended for use by Kamui 2.

Title	Function Name	Function	No
Function	mwPlyExecDrawSvr	Video rendering server functions (for Kamui 2)	2.3

[Syntax] void mwPlyExecDrawSvr(void);

[Input] None

[Output] None

[Rtn Val] None

[Purpose] This function renders video. This function is intended for use by Kamui 2.

14.3. Basic operation processing

These are the bare minimum functions that are needed for simple playback of movies.

Title	Function Name	Function	No
Function	mwPlyCreateSofdec	Handle generation	3.1

[Syntax] MWPLY mwPlyCreateSofdec(MWS_PLY_CPRM_SFD *cprm);

[Input] cprm : Handle generation parameter

[Output] None

[Rtn Val] Middleware playback handle

[Purpose] This function generates a handle. This function allocates video resources such as texture areas.

[Example] A usage example is shown below.

```

memset(&cprm, 0, sizeof(cprm));           /* For zero setting for reserved
members*/
cprm.compo_mode = MWD_PLY_COMPO_OPEQ;      /* Composition mode */
cprm.ftype = MWD_PLY_FTYPE_SFD;            /* Playback file type */
cprm.dtype = MWD_PLY_DTYPE_AUTO;            /* Picture quality priority */
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 */
*/
cprm.wksize = mwPlyCalcWorkCprmSfd(cprm);   /* Work area size calculation */
*/
cprm.work = syMalloc(cprm.wksize);          /* Work area allocation */
ply = mwPlyCreateSofdec(&cprm);

```

Title	Function Name	Function	No
Function	mwPlyCalcWorkCprmSfd	Work area size calculation	3.2

[Syntax] Sint32 mwPlyCalcWorkCprmSfd(MWS_PLY_CPRM_SFD *cprm);
 [Input] cprm : Handle generation parameter
 [Output] None
 [Rtn Val] Work area size (unit: bytes)
 [Purpose] Calculates the work area size to use in MPEG SofdecF/X playback.

Title	Function Name	Function	No
Function	mwPlyDestroy	Destroy a handle	3.3

[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 (playback of GD-ROM)	3.4

[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 on GD-ROM.

Title	Function Name	Function	No
Function	mwPlyStartSj	Start playing (playback of stream joint)	3.5

[Syntax] void mwPlyStartSj(MWPLY ply, SJ sj);
 [Input] ply : Middleware playback handle
 sj : Stream joint
 [Output] None
 [Rtn Val] None
 [Purpose] Starts playing motion pictures and sound that are provided for stream joint.

Title	Function Name	Function	No
Function	mwPlyStartMem	Start playing (playback of memory)	3.6

[Syntax] void mwPlyStartMem(MWPLY ply, void *addr, Sint32 len);
 [Input] ply : Middleware playback handle
 addr : Pointer to movie data
 len : Length of data
 [Output] None
 [Rtn Val] None
 [Purpose] This function starts playback of a movie in memory.

Title	Function Name	Function	No
Function	mwPlyStop	Stop playing	3.7

[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	3.8

[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. Normally, the status changes in the sequence STOP -> PREP -> PLAYING -> PLAYEND, but if a GD read error, etc., occurs, the status changes to ERROR.

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

Title	Function Name	Function	No
Function	mwPlyGetTime	Get the number of the playing sample	3.9

[Syntax] void mwPlyGetTime(MWPLY ply, Sint32 *ncount, Sint32 *tscale);
 [Input] ply : Middleware playback handle
 [Output] ncount : time
 tscale : unit of time (Hz)
 [Rtn Val] None
 [Purpose] This function gets the playback time.
 [Remarks] Actual time (rtime) can be calculated by:

$$rtime = ncount / tscale;$$

Title	Function Name	Function	No
Function	mwPlyGetInputSj	Input stream joint acquisition	3.10

[Syntax] SJ mwPlyGetInputSj(MWPLY ply);
 [Input] ply : Middleware playback handle
 [Output] None
 [Rtn Val] Stream joint
 [Purpose] This function gets the stream joint in which the data that is being input to the decoder is being buffered. Playback can then be accomplished with the mwPlyStartSj function using the stream joint that was gotten.

Title	Function Name	Function	No
Function	mwPlyPause	Perform pause setting	3.11

[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.

14.4. Audio output control

These functions control audio output.

Title Function	Function Name mwPlySetOutVol	Function Perform volume setting	No 4.1
-------------------	---------------------------------	------------------------------------	-----------

[Syntax] void mwPlySetOutVol (MWPLY ply, Sint32 vol);
[Input] ply : Middleware playback handle
vol : Volume (-999 to 0) (unit: 1/10dB)
[Output] None
[Rtn Val] None
[Purpose] Set up the volume. By default, it is set to 0.
You can create the 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 the fade-out effect.

Title Function	Function Name mwPlyGetOutVol	Function Get the volume's value	No 4.2
-------------------	---------------------------------	------------------------------------	-----------

[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.

Title Function	Function Name mwPlySetOutPan	Function Set up panpot	No 4.3
-------------------	---------------------------------	---------------------------	-----------

[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 to 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.

Title Function	Function Name mwPlyGetOutPan	Function Get the panpot value	No 4.4
-------------------	---------------------------------	----------------------------------	-----------

[Syntax] Sint32 mwPlyGetOutPan (MWPLY ply, Sint32 chno);

[Input] ply : Middleware playback handle
 chno : Channel to set
 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.

14.5. Window control

These functions control the video window display.

Title Function	Function Name mwPlySetDispPos	Function Set up the display position	No 5.1
-------------------	----------------------------------	---	-----------

[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	Function Name mwPlySetDispSize	Function Set up the display size	No 5.2
-------------------	-----------------------------------	-------------------------------------	-----------

[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	5.3

[Syntax] void mwPlySetBright(MWPLY ply, Sint32 val);
 [Input] ply : Middleware playback handle
 val : Brightness (0 to 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.

Title	Function Name	Function	No
Function	mwPlyGetBright	Get the brightness	5.4

[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	5.5

[Syntax] void mwPlySetBrightOfst(MWPLY ply, Sint32 val);
 [Input] ply : Middleware playback handle
 val : Brightness offset (0 to 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	5.6

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

Title	Function Name	Function	No
Function	mwPlySetDispZ	Set up the display screen depth value	5.7

[Syntax] void mwPlySetDispZ(MWPLY ply, float z);
 [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.

Title	Function Name	Function	No
Function	mwPlyGetDispZ	Get the display screen depth value	5.8

[Syntax] float mwPlyGetDispZ(MWPLY ply);
 [Input] ply : Middleware playback handle
 [Output] None
 [Rtn Val] Depth value is from 1.0 (closest to surface) to 65536.0 (deepest)
 [Purpose] Gets the depth value of the display screen being set.

14.6. Surface control

These functions control the video surface display.

Title Function	Function Name mwPlyCalcSrfBufSize	Function Surface pointer buffer size calculation	No 6.1
-------------------	--------------------------------------	---	-----------

[Syntax] Sint32 mwPlyCalcSrfBufSize(MWPLY ply, Sint32 npnt);

[Input] ply : Middleware playback handle
npnt : Number of surface points

[Output] None

[Rtn Val] Surface point buffer size (unit: bytes)

[Purpose] This function gets the surface point buffer size.

Title Function	Function Name mwPlySetSrfPntBuf	Function Surface point buffer setting	No 6.2
-------------------	------------------------------------	--	-----------

[Syntax] void mwPlySetSrfPntBuf(MWPLY ply, Sint32 npnt, void *buf, Sint32 bsize);

[Input] ply : Middleware playback handle
npnt : Number of surface points
buf : Buffer
bsize : Buffer size (unit: bytes)

[Output] None

[Rtn Val] None

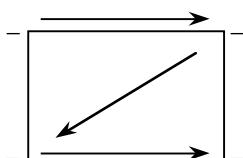
[Purpose] This function sets the specified buffer as a surface point buffer. This buffer must be assigned when using multi-window display.

[Example] When displaying 25 windows (100 surface points)

```
size = mwPlyCalcSrfBufSize(ply, 4*25);
buf = syMalloc(size);
mwPlySetSrfPntBuf(ply, 4*25, buf, size);
```

Title Function	Function Name mwPlySetSrfPos	Function Set up the display position	No 6.3
-------------------	---------------------------------	---	-----------

[Syntax] void mwPlySetSrfPos(MWPLY ply, Uint32 no, float lx, float ly, float lz);
 [Input] ply : Middleware playback handle
 no : Vertex number (from 0 to 3, in a Z pattern)
 lx : X-coordinate
 ly : Y-coordinate
 lz : Depth value is from 1.0 (closest to surface) to 65536.0 (deepest)
 [Output] None
 [Rtn Val] None
 [Purpose] This function sets the position of the surface to be displayed for one vertex.
 [Remarks] The "Z pattern" for the vertex numbers is as shown below.



Title Function	Function Name mwPlyGetSrfPos	Function Display position acquisition	No 6.4
-------------------	---------------------------------	--	-----------

[Syntax] void mwPlyGetSrfPos(MWPLY ply, Uint32 no, float *lx, float *ly, float *lz);
 [Input] ply : Middleware playback handle
 no : Vertex number (from 0 to 3, in a Z pattern)
 [Output] lx : X-coordinate
 ly : Y-coordinate
 lz : Depth value is from 1.0 (closest to surface) to 65536.0 (deepest)
 [Rtn Val] None
 [Purpose] This function gets the position of the surface for one vertex.

Title	Function Name	Function	No
Function	mwPlySetSrfBright	Set up the brightness	6.5

[Syntax] void mwPlySetSrfBright(MWPLY ply, Uint32 no, float a, float r, float g, float b);

[Input] ply : Middleware playback handle
 no : Vertex number (from 0 to 3, in a Z pattern)
 a : Alpha value (0.0 to 1.0)
 r : Red component value (0.0 to 1.0)
 g : Green component value (0.0 to 1.0)
 b : Blue component value (0.0 to 1.0)

[Output] None

[Rtn Val] None

[Purpose] This function sets the brightness values for one vertex.

Title	Function Name	Function	No
Function	mwPlyGetSrfBright	Get the brightness	6.6

[Syntax] void mwPlyGetSrfBright(MWPLY ply, Uint32 no, float *a, float *r, float *g, float *b);

[Input] ply : Middleware playback handle
 no : Vertex number (from 0 to 3, in a Z pattern)

[Output] a : Alpha value (0.0 to 1.0)
 r : Red component value (0.0 to 1.0)
 g : Green component value (0.0 to 1.0)
 b : Blue component value (0.0 to 1.0)

[Rtn Val] None

[Purpose] This function gets the brightness values for one vertex.

Title	Function Name	Function	No
Function	mwPlySetSrfBrightOfst	Set up the brightness offset	6.7

[Syntax] void mwPlySetSrfBrightOfst(MWPLY ply,Uint32 no,float a,float r,float g,float b);
 [Input] ply : Middleware playback handle
 no : Vertex number (from 0 to 3, in a Z pattern)
 a : Alpha value (0.0 to 1.0)
 r : Red component value (0.0 to 1.0)
 g : Green component value (0.0 to 1.0)
 b : Blue component value (0.0 to 1.0)
 [Output] None
 [Rtn Val] None
 [Purpose] This function sets the brightness offset values for one vertex.

Title	Function Name	Function	No
Function	mwPlyGetSrfBrightOfst	Get the brightness offset	6.8

[Syntax] void mwPlyGetSrfBrightOfst(MWPLY ply,Uint32 no,float *a,float *r,float *g,float *b);
 [Input] ply : Middleware playback handle
 no : Vertex number (from 0 to 3, in a Z pattern)
 [Output] a : Alpha value (0.0 to 1.0)
 r : Red component value (0.0 to 1.0)
 g : Green component value (0.0 to 1.0)
 b : Blue component value (0.0 to 1.0)
 [Rtn Val] None
 [Purpose] This function gets the brightness offset for one vertex.

Title	Function Name	Function	No
Function	mwPlySetImgPos	Image position setting	6.9

[Syntax] void mwPlySetImgPos(MWPLY ply, Uint32 no, float lx, float ly);
 [Input] ply : Middleware playback handle
 no : Vertex number (from 0 to 3, in a Z pattern)
 lx : X-coordinate
 ly : Y-coordinate
 [Output] None
 [Rtn Val] None
 [Purpose] This function sets the image position for one vertex.

Title	Function Name	Function	No
Function	mwPlyGetImgPos	Image position acquisition	6.10

[Syntax] void mwPlyGetImgPos(MWPLY ply, Uint32 no, float *lx, float *ly);
 [Input] ply : Middleware playback handle
 no : Vertex number (from 0 to 3, in a Z pattern)
 [Output] lx : X-coordinate
 ly : Y-coordinate
 [Rtn Val] None
 [Purpose] This function gets the image position for one vertex.

Title Function	Function Name mwPlyGetImgSize	Function Image size acquisition	No 6.11
-------------------	----------------------------------	------------------------------------	------------

[Syntax] void mwPlyGetImgSize(MWPLY ply, Sint32 *isx, Sint32 *isy);

[Input] ply : Middleware playback handle

[Output] isx : X-coordinate

isy : Y-coordinate

[Rtn Val] None

[Purpose] This function gets the image size.

14.7. Effects

These functions are used during effect movie playback.

Title Function	Function Name mwPlySetCompoMode	Function Composition mode setting	No 7.1
-------------------	------------------------------------	--------------------------------------	-----------

[Syntax] void mwPlySetCompoMode(MWPLY ply, Sint32 mode);
 [Input] ply : Middleware playback handle
 mode : Composition mode(MWD_PLY_COMPO~)
 [Output] None
 [Rtn Val] None
 [Purpose] This function sets the composition mode. This function is used to switch the composition mode during playback. The composition mode is switched at the same time that the callback function registered by the mwPlyEntryFxCb function is executed.

Title Function	Function Name mwPlyEntryFxCb	Function Effect callback function registration	No 7.2
-------------------	---------------------------------	---	-----------

[Syntax] void mwPlyEntryFxCb(MWPLY ply,
 void (*fn)(void *obj, Sint32 fno, Sint32 time, Sint32 tunit), void *obj);
 [Input] ply : Middleware playback handle
 fn : Callback function
 obj : First parameter of callback function
 [Output] None
 [Rtn Val] None
 [Purpose] This function registers the function that is executed before composition processing. Change the composition mode when this function is executed. The callback function parameters are as follows:
 fno : Frame serial number in movie data ("0" is the first)
 time : Playback time ("time/tunit" gives the actual time)
 tunit : Playback time unit (Hz)
 [Example] A usage example is shown below.

```
void user_chg_compo(void *obj, Sint32 fno, Sint32 time, Sint32 tunit)
{
    MWPLY ply=(MWPLY) obj;

    if ( fno == 0 ) {
        mwPlySetCompoMode(ply, compo_mode=MWD_PLY_COMPO_TRNSP);
    } else if ( fno == 1*8*30 ) {
        mwPlySetCompoMode(ply, compo_mode=MWD_PLY_COMPO_LUMI);
    } else if ( fno == 2*8*30 ) {
        mwPlySetCompoMode(ply, compo_mode=MWD_PLY_COMPO_ALPH3);
    }
}
mwPlyEntryFxCb(ply, user_chg_compo, (void *)ply);
```

Title	Function Name	Function	No
Function	mwPlySetAlphTbl	Setup of table for conversion from luminance to alpha value	7.3

[Syntax] void mwPlySetAlphTbl(MWPLY ply, long bno, float alph[256]);

[Input] ply : Middleware playback handle
 bno : Bank number
 alph : Alpha table

[Output] None

[Rtn Val] None

[Purpose] Setup of table for conversion from luminance to alpha value.

[Example] luminance value Y<128 : $\alpha=0.0$
 luminance Y_128 : When $\alpha = 1.0$

```
for (y=0; y<128; y++) {
    alph[ y ] = 0.0;
}
for ( ; y<256; y++) {
    alph[ y ] = 1.0;
}
mwPlySetAlphTbl(ply, 0, alph);
```

1.8. Texture movie

These functions are used in texture movies.

Title Function	Function Name mwPlyGetMvFrm	Function Movie frame acquisition	No 8.1
-------------------	--------------------------------	-------------------------------------	-----------

[Syntax] Sint32 mwPlyGetMvFrm(MWPLY ply, MWS_PLY_MVFRM *frm);
[Input] ply : Middleware playback handle
[Output] frm : Video surface information
[Rtn Val] Movie frame acquisition result (1: acquired successfully; 0: did not acquire)
[Purpose] This function gets a movie frame.
When making a texture movie, use the surface from the movie frame that was gotten by this function to render the polygon. When using Ninja, set this surface in a texture list.

Title Function	Function Name mwPlyGetMskFrm	Function Mask frame acquisition	No 8.2
-------------------	---------------------------------	------------------------------------	-----------

[Syntax] Sint32 mwPlyGetMskFrm(MWPLY ply, MWS_PLY_MVFRM *frm);
[Input] ply : Middleware playback handle
[Output] frm : Video surface information
[Rtn Val] Mask frame acquisition result (1: acquired successfully; 0: did not acquire)
[Purpose] This function gets a mask frame.
When making a texture movie, use the surface from the mask frame that was gotten by this function to render the polygon. When using Ninja, set this surface in a texture list.

1.9. Seamless continuous playback

These functions are used for seamless continuous playback.

Title Function	Function Name mwPlyEntryFname	Function Seamless continuous playback file registration	No 9.1
-------------------	----------------------------------	--	-----------

[Syntax] void mwPlyEntryFname(MWPLY ply, char *fname);
[Input] ply : Middleware playback handle
fname : File name
[Output] None
[Rtn Val] None
[Purpose] This function registers a file for seamless continuous playback.

Title Function	Function Name mwPlyStartSeamless	Function Seamless continuous playback start	No 9.2
-------------------	-------------------------------------	--	-----------

[Syntax] void mwPlyStartSeamless(MWPLY ply);
[Input] ply : Middleware playback handle
[Output] None
[Rtn Val] None
[Purpose] This function starts seamless continuous playback of the registered file.

Title	Function Name	Function	No
Function	mwPlyReleaseSeamless	Seamless continuous playback cancel	9.3

[Syntax] void mwPlyReleaseSeamless(MWPLY ply);
 [Input] ply : Middleware playback handle
 [Output] None
 [Rtn Val] None
 [Purpose] This function cancels seamless continuous playback. After playback of the currently registered file is completed, the handle changes to the PLAYEND status.

Title	Function Name	Function	No
Function	mwPlySetLpFlg	Seamless loop playback setup	9.4

[Syntax] void mwPlySetLpFlg(MWPLY ply, long flg);
 [Input] ply : Middleware playback handle
 flg : Loop playback flag (1: loop; 0: loop cancelled)
 [Output] None
 [Rtn Val] None
 [Purpose] When this function sets the loop playback flag to the "ON" (1) state, seamless loop playback of all registered files is enabled.

Title	Function Name	Function	No
Function	mwPlyStartFnameLp	Seamless loop playback start	9.5

[Syntax] void mwPlyStartFnameLp(MWPLY ply, char *fname);
 [Input] ply : Middleware playback handle
 fname : File name
 [Output] None
 [Rtn Val] None
 [Purpose] This function initiates seamless loop playback of the specified file.

1.10. Playback mode setting

These functions set the playback mode.

Title	Function Name	Function	No
Function	mwPlySetFastHalfpel	Set up the fast half pel process	10.1

[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	10.2

[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	10.3

[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. Specify "0" for the video switch to display the video as a texture movie.

1.11. Internal status acquisition

These functions are debugging functions that are used to get the internal statuses of the middleware.

Title Function	Function Name mwPlyGetNumDropFrm	Function Get the number of dropped frames	No 11.1
-------------------	-------------------------------------	--	------------

[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.

Title Function	Function Name mwPlyGetNumDecPool	Function Acquisition of number of decoded frames	No 11.2
-------------------	-------------------------------------	---	------------

[Syntax] Sint32 mwPlyGetNumDecPool(MWPLY ply);
[Input] ply : Middleware playback handle
[Output] None
[Rtn Val] Number of decoded frames
[Purpose] This function gets the number of decoded frames. This value will always be "0" if any dropped frames occur.

Title	Function Name	Function	No
Function	mwPlyGetNumTotalDec	Acquisition of total number of pictures decoded	11.3

[Syntax] Sint32 mwPlyGetNumTotalDec(MWPLY ply);
 [Input] ply : Middleware playback handle
 [Output] None
 [Rtn Val] Total number of pictures decoded
 [Purpose] This function gets the total number of pictures that have been decoded.

Title	Function Name	Function	No
Function	mwPlyGetNumTotalSkip	Acquisition of total number of pictures skipped	11.4

[Syntax] Sint32 mwPlyGetNumTotalSkip(MWPLY ply);
 [Input] ply : Middleware playback handle
 [Output] None
 [Rtn Val] Total number of pictures skipped
 [Purpose] This function gets the total number of pictures skipped.

Title	Function Name	Function	No
Function	mwPlyGetNumIbufRoom	Acquisition of free space in input buffer	11.5

[Syntax] Sint32 mwPlyGetNumIbufRoom(MWPLY ply);
 [Input] ply : Middleware playback handle
 [Output] None
 [Rtn Val] Free space in input buffer (unit :bytes)
 [Purpose] This function gets the free space in input buffer.

Title	Function Name	Function	No
Function	mwPlyGetNumDispWait	Acquisition of number of frames waiting to be displayed	11.6

[Syntax] Sint32 mwPlyGetNumDispWait(MWPLY ply);
 [Input] ply : Middleware playback handle
 [Output] None
 [Rtn Val] Number of frames waiting to be displayed
 [Purpose] This function gets the number of frames waiting to be displayed.

Title	Function Name	Function	No
Function	mwPlyGetNumRend	Acquisition of number of frames being rendered	11.7

[Syntax] Sint32 mwPlyGetNumRend(MWPLY ply);
 [Input] ply : Middleware playback handle
 [Output] None
 [Rtn Val] Number of frames being rendered
 [Purpose] This function gets the number of frames being rendered.

Title	Function Name	Function	No
Function	mwPlyGetNumLoadFrm	Acquisition of number of frames transferred to texture memory	11.8

[Syntax] Sint32 mwPlyGetNumLoadFrm(MWPLY ply);
 [Input] ply : Middleware playback handle
 [Output] None
 [Rtn Val] Number of frames transferred to texture memory
 [Purpose] This function gets the number of frames transferred to texture memory.

1.12. Error processing

These functions are used for error processing.

Title	Function Name	Function	No
Function	mwPlyEntryErrFunc	Register a function to call on error	12.1

[Syntax] void mwPlyEntryErrFunc(MW_PLY_ERRFN errfn, void *obj);
[Input] errfn : Error function
obj : 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);
}
```

Sega Dreamcast™

***Dreamcast
Stream Joint
External
Specifications***

1. Overview

The stream joint library is a library of stream joints that are used to pass streaming data between modules. Using stream joints makes it possible to develop highly independent stream processing modules. For example, in the development of a mixer module such as the one indicated below, the developer only needs to be aware of the input and output stream joints. Because the mixer module is not dependent on the modules that are connected on the other side of the stream joints, the mixer module can be re-used in other systems.

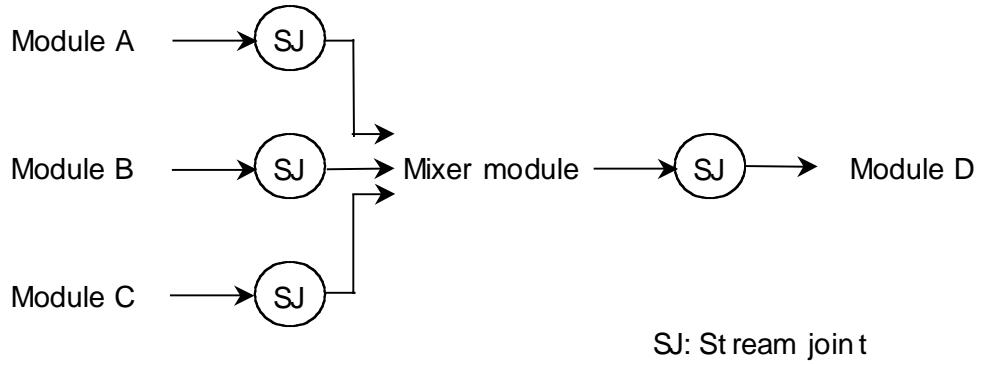


Fig. Example of Module Connections Using Stream Joints

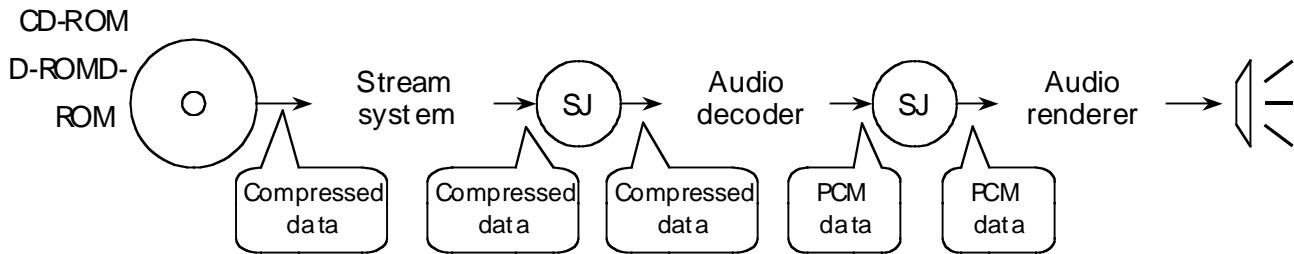


Fig. Example of Use of Stream Joints

2. Stream Joints

2.1. What Is a Stream Joint?

A stream joint is a mechanism that is used to pass streaming data between modules. A stream joint has two lines, a data line and a free line. The data line stores valid data, and the free line stores data that has already been used. The user can get data from each line, and can put data to each line. Each line operates as a FIFO buffer, so data can be gotten in the order in which it was put to the line.

Data can be gotten in chunks from each line. A data chunk consists of a pointer to the data area, and the size of the chunk. Unused data in a data chunk that was gotten can be returned (by using "Unget"). A data chunk is defined as shown below.

```
/* Data chunk */
typedef struct {
    char *data;          /* Pointer to data area */
    long len;           /* Size of data area */
} SJCK;
```

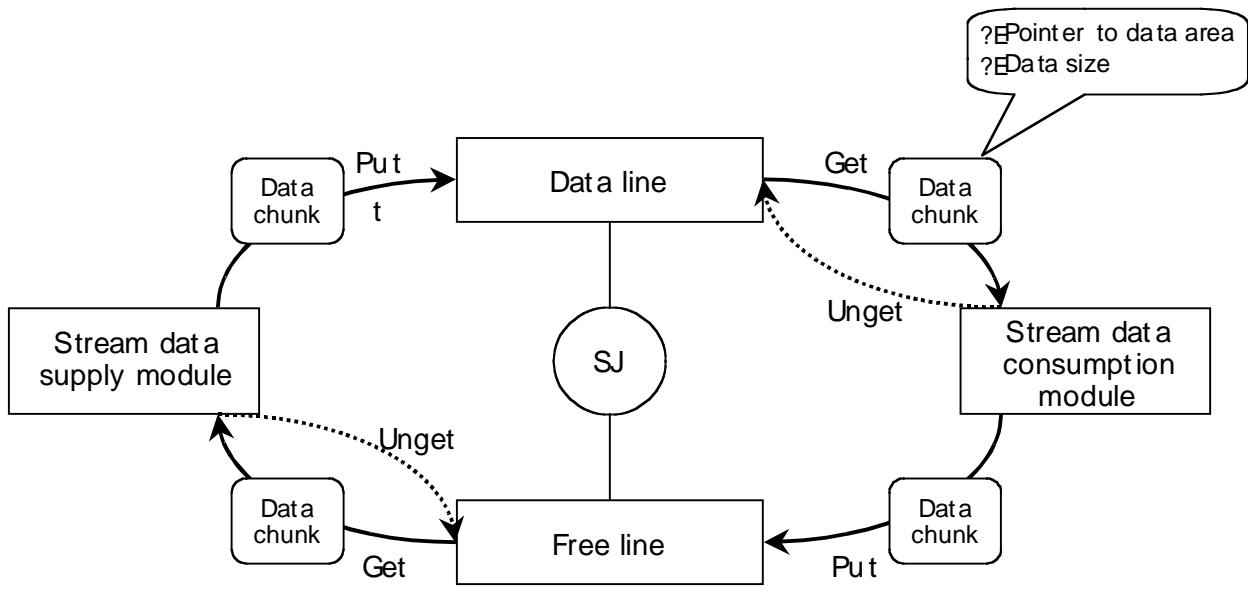
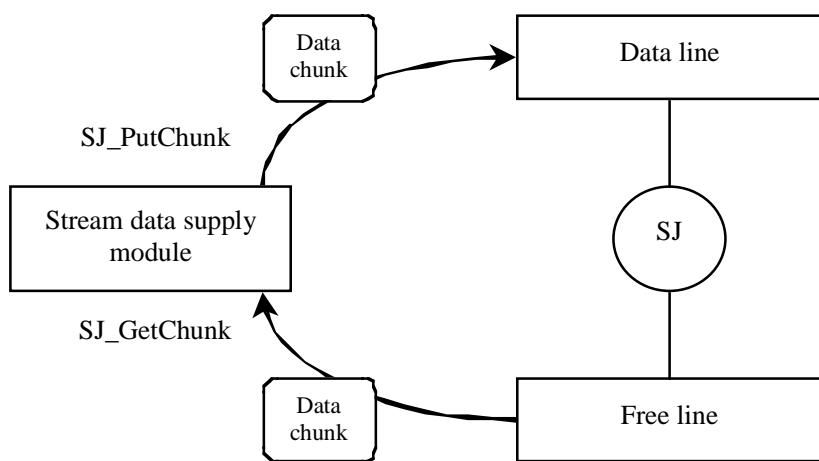


Fig. Stream Joint Mechanism

2.2. Data Supply Module

The data supply module accepts used data from the free line, fills the data chunk with new data, and passes the data chunk onto the data line. The procedure for supplying new data is described below.

- (1) After specifying the data size that can be supplied (nbyte), the data supply module accepts a data chunk f r o m t h e f r e e l i n e . The size of the data chunk that is obtained is the maximum size of continuous data that can be gotten. Therefore, there is no guarantee that the size of the data chunk that is obtained is equal to the value of "nbyte".
SJ_GetChunk(sj, SJ_LIN_FREE, nbyte, &ck);
- (2) The module copies data into the data chunk that was received.
memcpy(ck.data, user_data, ck.len);
- (3) The module puts the data chunk on the data line.
SJ_PutChunk(sj, SJ_LIN_DATA, &ck);



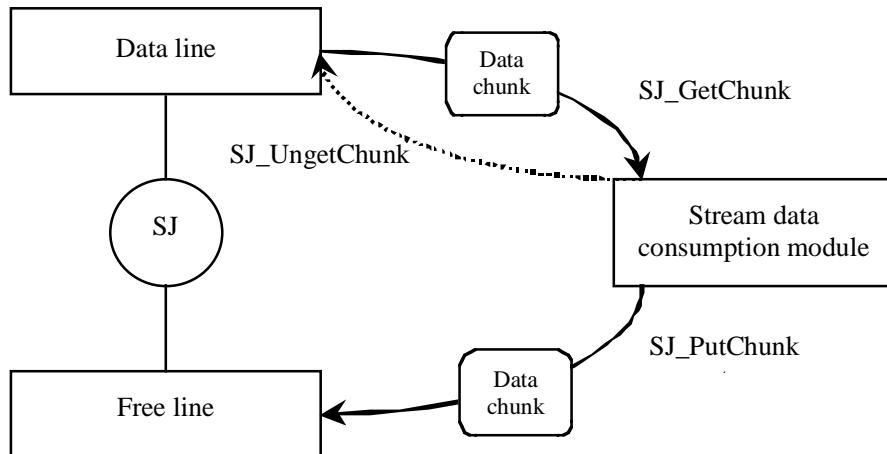
<Program example>

```
SJ sj; /* Stream joint */  
SJCK ck; /* Data chunk */  
  
SJ_GetChunk(sj, SJ_LIN_FREE, 1024, &ck); /* Gets used data */  
user_supply_func(ck.data, ck.len); /* Supplies data */  
SJ_PutChunk(sj, SJ_LIN_DATA, &ck); /* Puts data on data line */
```

2.3. Data Consumption Module

The data consumption module accepts valid data from the data line, and passes the consumed data to the free line. The procedure for accepting data is described below.

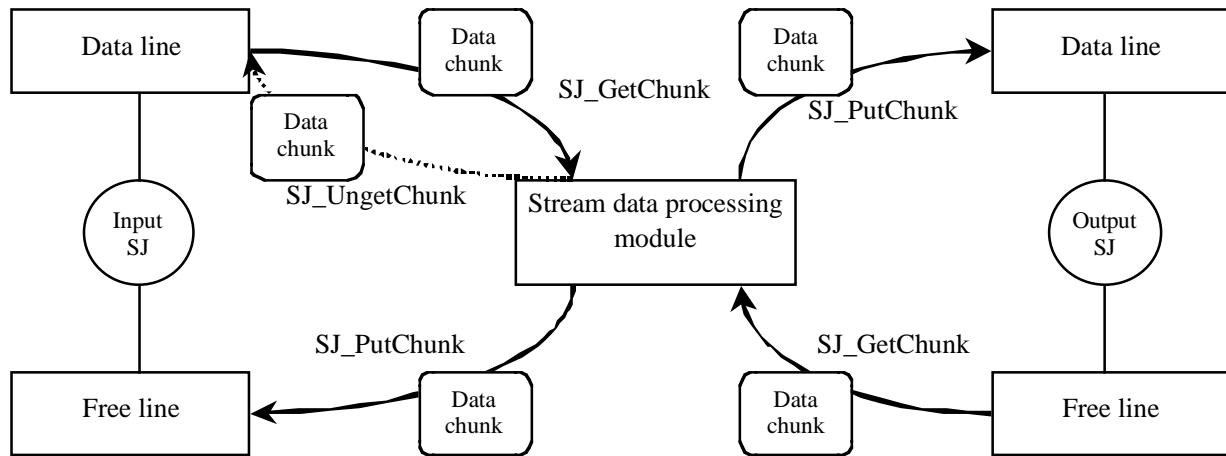
- (1) After specifying the data size that is required from the data line (nbyte), the data consumption module accepts a data chunk.
`SJ_GetChunk(sj, SJ_LIN_DATA, nbyte, &ck);`
 - (2) The module processes the data chunk that was accepted. ("nused" is the size of the data that was used.)
`nused = user_consume_func(ck.data, ck.len);`
 - (3) The module splits the data chunk into a chunk containing data for which processing is finished, and a chunk containing data for which processing is not finished.
`SJ_SplitChunk(&ck, nused, &ck, &ck2);`
 - (4) The module puts the data chunk for which processing is finished on the free line.
`SJ_PutChunk(sj, SJ_LIN_FREE, &ck);`
 - (5) The module returns ("ungets") the data that could not be processed to the data chunk.
`SJ_UngetChunk(sj, SJ_LIN_DATA, &ck2);`



<Program example>

2.4. Data Processing Module

The data processing module combines the data consumption module and the data supply module.



<Program example>

```

SJ_sji;                      /* Input stream joint */
SJ_sjo;                      /* Output stream joint */
SJCK_cki, cki2;              /* Data chunk for input SJ */
SJCK_cko, cko2;              /* Data chunk for output SJ */
long nused;                   /* Amount of data consumed */
long ngen;                    /* Amount of data supplied */

SJ_GetChunk(sji, SJ_LIN_DATA, nbytes1, &cki);
SJ_GetChunk(sjo, SJ_LIN_FREE, nbytes2, &cko);           /* Gets valid data */
                                                /* Gets an empty data area */

user_proc_func(&cki, &cko, &nused, &ngen);          /* Processes data */

SJ_SplitChunk(&cki, nused, &cki, &cki2);            /* Splits input data chunk */
SJ_PutChunk(sji, SJ_LIN_FREE, &cki);                  /* Puts used chunk on free line */
SJ_UngetChunk(sji, SJ_LIN_DATA, &cki2);                /* Returns unused chunk to data line */

SJ_SplitChunk(&cko, ngen, &cko, &cko2);            /* Splits output data chunk */
SJ_PutChunk(sjo, SJ_LIN_DATA, &cko);                  /* Puts used chunk on data line */
SJ_UngetChunk(sjo, SJ_LIN_FREE, &cko2);                /* Returns unused chunk to free line */

```

3. Ring Buffer-type Stream Joint

A ring buffer-type stream joint permits access to a ring buffer using a stream joint API.

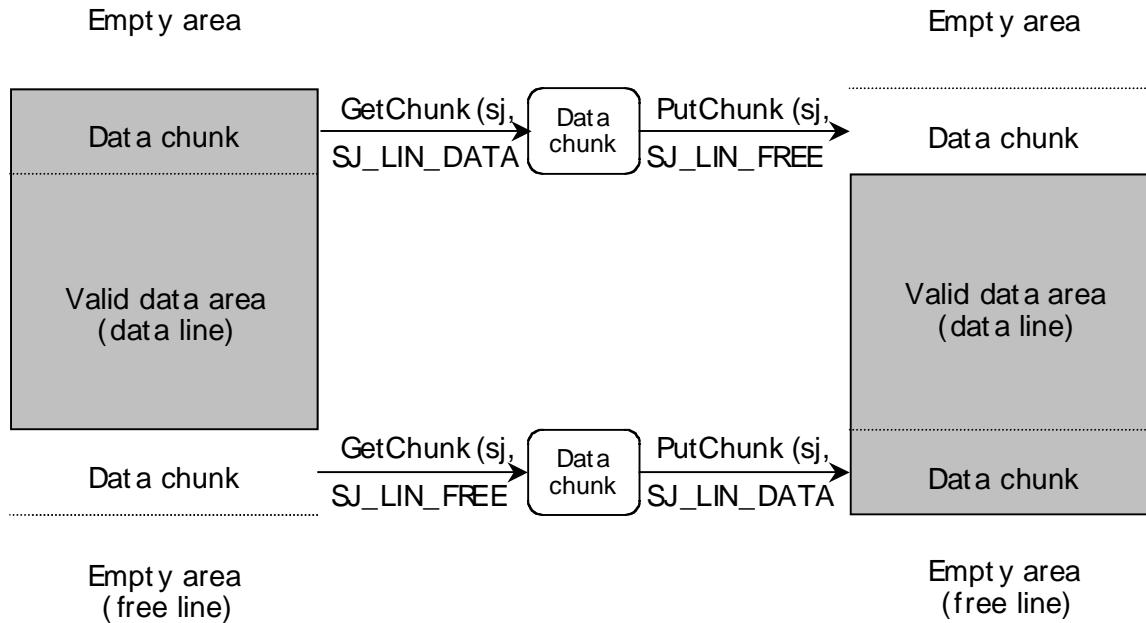


Fig. Ring Buffer-type Stream Joint

When creating a ring buffer-type stream joint, an extra area can be specified in the third parameter. The data from the beginning of the ring buffer is always copied in the extra area. As a result, it is guaranteed that the data is continuous up to the size of the extra area. For example, if it is necessary to process data in block units, specify the block size as the size of the extra area. As a result, thanks to the extra area, processing can be performed without being concerned about buffer wraparound, even though the buffer is a ring buffer.

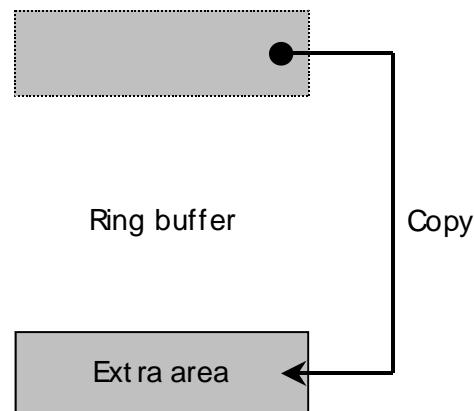


Fig. Extra Area

The following is a listing of a sample program.

<Sample Program>

```
void main(long ac, char *av[])
{
    SJ          sj;
    SJCK       ck, ck1, ck2;
    static char buf[1024+32];
    FILE        *fp, *fp2;
    long        nbyte, rsize;

    if ( (fp=fopen("test2a.bin", "rb")) == NULL ) {
        exit(1);
    }
    if ( (fp2=fopen("test2b.bin", "wb")) == NULL ) {
        exit(1);
    }

    SJRBF_Init();                         /* Initialization */
    sj = SJRBF_Create(buf, 1024, 32);      /* Handle creation */
    for (;;) {
        if ( SJ_GetNumData(sj, SJ_LIN_DATA) == 0 && feof(fp) )
            break;
        /* Read processing */
        nbyte = rand() % 1025;
        SJ_GetChunk(sj, SJ_LIN_FREE, nbyte, &ck);
        rsize = fread(ck.data, sizeof(char), ck.len, fp);
        SJ_SplitChunk(&ck, rsize, &ck1, &ck2);
        SJ_PutChunk(sj, SJ_LIN_DATA, &ck1);
        SJ_UngetChunk(sj, SJ_LINE_FREE, &ck2);
        /* Write processing */
        nbyte = rand() % 1025;
        SJ_GetChunk(sj, SJ_LIN_DATA, nbyte, &ck);
        fwrite(ck.data, ck.len, sizeof(char), fp2);
        SJ_PutChunk(sj, SJ_LIN_FREE, &ck);
    }
    SJ_Destroy(sj);
    fclose(fp);
    fclose(fp2);
    getchar();
}
```

4. Memory-type Stream Joint

A memory-type stream joint permits access to data in memory using a stream joint API. This type of stream joint is used to playback audio or video data that has already been loaded into memory. This type of stream joint is basically identical to the ring buffer-type stream joint, except for the following differences:

- (1) In the initial state, a data chunk exists in the data line.
With the SJ_Reset function, all data exists in the data line.
- (2) There is no extra area.
- (3) Normally, the data chunk from the free line is not normally gotten.

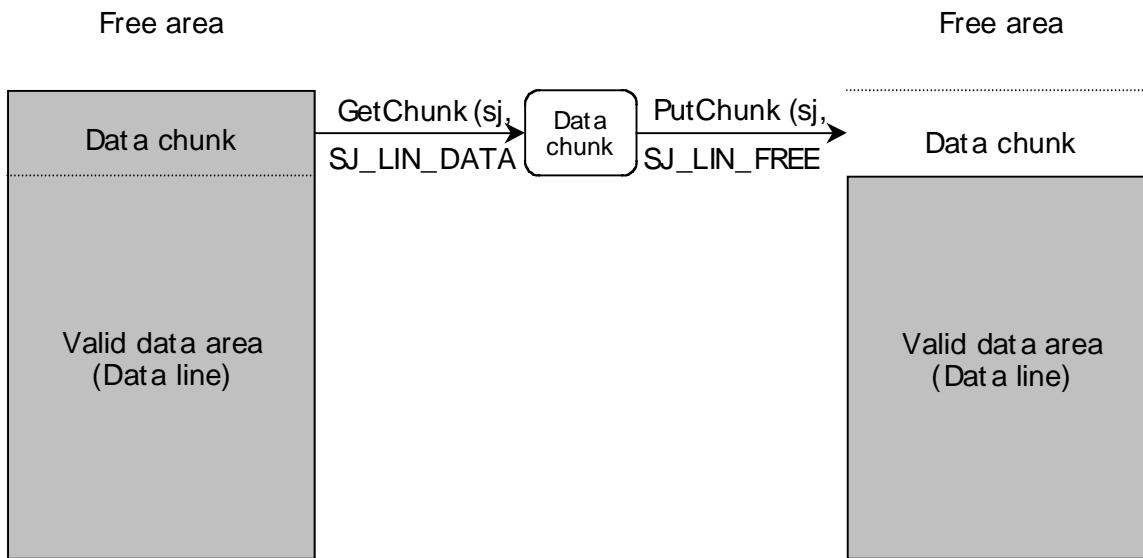


Fig. Memory-type Stream Joint

The following is a listing of a sample program.

<Sample Program>

```
void main(long ac, char *av[])
{
    SJ          sj;
    SJCK       ck, ck1, ck2;
    static char data[1024*1024]; /* 1MB data area */
    FILE        *fp, *fp2;
    long        nbyte, rsize, dtsize;

    /* Data read */
    fp = fopen("test2a.bin", "rb");
    dtsize = fread(data, sizeof(char), sizeof(data), fp);
    fclose(fp);

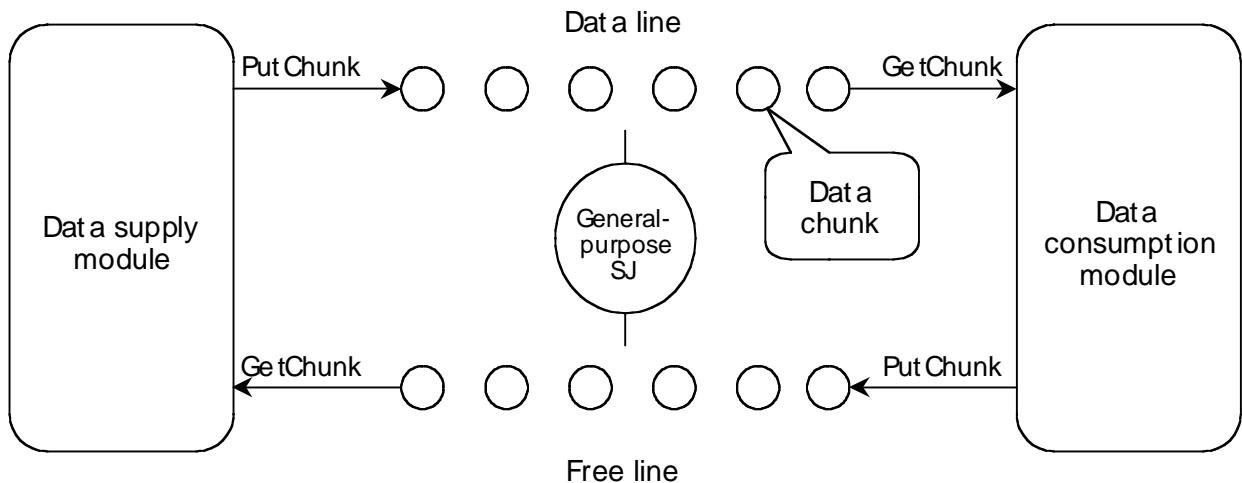
    /* Opens file for data write */
    fp2 = fopen("test2b.bin", "wb");

    SJMEM_Init();                                /* Initialization */
    sj = SJMEM_Create(data, dtsize);             /* Handle creation */
    for (;;) {
        /* Write processing */
        nbyte = (rand() % 1025) + 1;
        SJ_GetChunk(sj, SJ_LIN_DATA, nbyte, &ck);
        if ( ck.len == 0 )
            break;
        fwrite(ck.data, ck.len, sizeof(char), fp2);
        SJ_PutChunk(sj, SJ_LIN_FREE, &ck);
    }
    SJ_Destroy(sj);
    fclose(fp2);
}
```

5. General-purpose Stream Joint

Unlike ring buffer-type stream joints and memory-type stream joints, a general-purpose stream joint does not possess its own storage area. The data supply module provides data chunks to the general-purpose stream joint, and the data consumption module accepts those data chunks directly from the stream joint. This method permits efficient data streaming, without copying data unnecessarily.

A general-purpose stream joint resembles a conveyor belt. The data supply module packs data into packets (data chunks) and places those packets on the conveyor belt (the data line). The data consumption module accepts the data in the order in which it was placed on the conveyor belt. Once the data that was received has been consumed, the module places the packet (data chunk) on the return conveyor belt (the free line). The data supply module accepts the empty packet from the free line, packs it with new data, and places it back on the data line again.



In addition, the unused portion of a packet that was gotten by GetChunk can be returned by the UnGetChunk function.

Furthermore, the general-purpose stream joint has a function that links data chunks on the line. If a data chunk that is placed on the line is in an area that is contiguous with the area of the previous data chunk, this function links the new data chunk with the previous data chunk. In other words, when placing a packet on the conveyor belt, it is possible to place the contents of that packet and the contents of the previous packet together in a new packet, and then place that new packet on the conveyor belt.

This function can be used for a ring buffer for playing back voice data. If the data size on the supplying side differs from the data size on the consuming side, the supplied data becomes linked together gradually, and the consuming side gets the linked data a little bit at a time.

The setting that enables or disables this linking function is specified when the stream joint is created.

5.1. Method for Creating a General-purpose Stream Joint

When creating a general-purpose stream joint, it is necessary to provide a work area that can accommodate the maximum number of data chunks that the stream joint will handle. For example, if the stream joint will handle up to 100 data chunks, the definition would be as shown below.

```
#define MAX_NCHUNK (100)
SJ sj;
static char sj_work[SJUNI_CALC_WORK(MAX_NCHUNK)];

sj = SJUNI_Create(SJUNI_MODE_SEPA, sj_work, sizeof(sj_work));
```

Furthermore, when linking data as for a ring buffer, the definition would be as shown below. Because a ring buffer is divided into a maximum of three buffer areas, the maximum number of data chunks would be three.

```
#define MAX_NCHUNK (3)
SJ sj;
static char sj_work[SJUNI_CALC_WORK(MAX_NCHUNK)];

sj = SJUNI_Create(SJUNI_MODE_JOIN, sj_work, sizeof(sj_work));
```

<Sample Program (Ring Buffer Implemented through the General-purpose Type)>

```
void main(long ac, char *av[])
{
    SJ          sj;
    SJCK       ck, ck1, ck2;
    static char buf[1024];
    static char sjwork[SJUNI_CALC_WORK(3)];
    FILE        *fp, *fp2;
    long         nbyte, rsize;

    if ( (fp=fopen("test2a.bin", "rb")) == NULL ) {
        exit(1);
    }
    if ( (fp2=fopen("test2b.bin", "wb")) == NULL ) {
        exit(1);
    }

    SJUNI_Init();                                     /* Initialization */
    sj = SJUNI_Create(SJUNI_MODE_JOIN, sjwork, sizeof(sjwork));      /* Handle creation */
    ck.data = buf, ck.len = sizeof(buf);
    SJ_PutChunk(sj, SJ_LIN_FREE, &ck);                  /* Supplies the buffer area */
    for (;;) {
        if ( SJ_GetNumData(sj, SJ_LIN_DATA) == 0 && feof(fp) )
            break;
        /* Read processing */
        nbyte = rand() % 1025;
        SJ_GetChunk(sj, SJ_LIN_FREE, nbyte, &ck);
        rsize = fread(ck.data, sizeof(char), ck.len, fp);
        SJ_SplitChunk(&ck, rsize, &ck1, &ck2);
        SJ_PutChunk(sj, SJ_LIN_DATA, &ck1);
        SJ_UngetChunk(sj, SJ_LINE_FREE, &ck2);
        /* Write processing */
        nbyte = rand() % 1025;
        SJ_GetChunk(sj, SJ_LIN_DATA, nbyte, &ck);
        fwrite(ck.data, ck.len, sizeof(char), fp2);
        SJ_PutChunk(sj, SJ_LIN_FREE, &ck);
    }
    SJ_Destroy(sj);
    fclose(fp);
    fclose(fp2);
    getchar();
}
```

<Sample Program (Memory-type Stream Joint Implemented through the General-purpose Type)>

```
void main(long ac, char *av[])
{
    SJ          sj;
    SJCK       ck, ck1, ck2;
    static char sjwork[SJUNI_CALC_WORK(3)];
    static char data[1024*1024]; /* 1MB data area */
    FILE        *fp, *fp2;
    long         nbytes, rsize, dtsize;

    /* Data read */
    fp = fopen("test2a.bin", "rb");
    dtsize = fread(data, sizeof(char), sizeof(data), fp);
    fclose(fp);

    /* Opens file for writing data */
    fp2 = fopen("test2b.bin", "wb");

    SJUNI_Init();                                /* Initialization */
    sj = SJUNI_Create(SJUNI_MODE_JOIN, sjwork, sizeof(sjwork)); /* Handle creation */
    ck.data = data, ck.len = dtsize;
    SJ_PutChunk(sj, SJ_LIN_DATA, &ck);           /* Supplies the data area */
    for (;;) {
        /* Write processing */
        nbytes = (rand() % 1025) + 1;
        SJ_GetChunk(sj, SJ_LIN_DATA, nbytes, &ck);
        if ( ck.len == 0 )
            break;
        fwrite(ck.data, ck.len, sizeof(char), fp2);
        SJ_PutChunk(sj, SJ_LIN_FREE, &ck);
    }
    SJ_Destroy(sj);
    fclose(fp2);
}
```

6. Data Specifications

The library data is listed below.

Table 6-1 Data List

Data name	Function	No
Constants		
SJ_LIN_	Data line	1.1
SJ_ERR_	Error code	1.2
SJUNI_MODE_	Linking mode	1.3
Data Types		
UUID	Library identifier	2.1
SJ	Stream joint handle	2.2
SJCK	Data chunk	2.3

6.1. Constants

Title	Data Name	Data	No
Data	SJ_LIN__	Data line	1.1

This is the line which supplies chunks.

Constant	Description
SJ_LIN_DATA	Data line Identifier for getting valid data
SJ_LIN_FREE	Free line Identifier for getting used data

Title	Data Name	Data	No
Data	SJ_ERR__	Error code	1.2

The value of MWD_SJ_ERR_OK is "0"; the other error codes have negative values.

Constant	Description
SJ_ERR_OK	Normal end
SJ_ERR_FATAL	Fatal error (does not normally occur)
SJ_ERR_INTERNAL	Internal error (does not normally occur)
SJ_ERR_PRM	Parameter error

Title	Data Name	Data	No
Data	SJUNI_MODE__	Linking mode	1.3

This parameter is specified when creating a general-purpose stream joint. This parameter determines whether or not to link data chunks that are supplied by the SJ_PutChunk function.

Constant	Description
SJUNI_MODE_SEPA	Do not link data chunks
SJUNI_MODE_JOIN	Link data chunks

6.2. Data Types

Title	Data Name	Data	No
Data	UUID	Library identifier	2.1

This is a library identifier that is uniquely defined in terms of time and space. Executing UUIDGEN.EXE or some similar function of Visual C++ yields a code; these codes are defined within each library for each of the following members.

Member	Type	Description
Data1	long	ID1
Data2	short	ID2
Data3	short	ID3
Data4[8]	char	ID4

```
C:\>uuidgen
211b2800-a6a9-11d1-8f3f-0060089448bc
```

```
#define SmD_LID { \
    0x211b2800, 0xa6a9, 0x11d1, 0x8f, 0x3f, 0x00, 0x60, 0x08, 0x94, 0x48, 0xbc\ \
}
```

Title	Data Name	Data	No
Data	SJ	Stream joint handle	2.2

This handle is used to control the stream joint.

Title	Data Name	Data	No
Data	SJCK	Data chunk	2.3

Member	Type	Description
data	char *	Starting address
len	long	Number of bytes

7. Function Specifications

A list of the library functions is shown below.

Table 7-1 List of Functions

Function name	Description of function	No
Common Functions		
SJ_Destroy	Deletes stream joint	1.1
SJ_Reset	Resets stream joint	1.2
SJ_GetChunk	Gets data chunk	1.3
SJ_UngetChunk	Returns data chunk	1.4
SJ_PutChunk	Puts data chunk	1.5
SJ_GetNumData	Gets the amount of data that can be read/written	1.6
SJ_GetUuid	Gets the stream joint UUID	1.7
SJ_EntryErrFunc	Registers an error processing function	1.8
SJ_SplitChunk	Splits a data chunk	1.9
Ring Buffer		
SJRBF_Init	Initialization	2.1
SJRBF_Create	Stream joint creation	2.2
Memory		
SJMEMP_Init	Initialization	3.1
SJMEMP_Create	Stream joint creation	3.2
General-purpose		
SJUNI_Init	Initialization	4.1
SJUNI_Create	Stream joint creation	4.2

7.1. Common Functions

These are common functions, regardless of the stream joint type.

Title Function	Function Name SJ_Destroy	Function Deletes stream joint	No 1.1
-------------------	-----------------------------	----------------------------------	-----------

[Format] void SJ_Destroy(SJ sj);
[Inputs] sj : Stream joint
[Outputs] None
[Function value] None
[Description] Deletes a stream joint.

Title Function	Function Name SJ_Reset	Function Resets stream joint	No 1.2
-------------------	---------------------------	---------------------------------	-----------

[Format] void SJ_Reset(SJ sj);
[Inputs] sj : Stream joint
[Outputs] None
[Function value] None
[Description] Resets a stream joint.

Title Function	Function Name	Function Gets data chunk	No 1.3
--------------------------	----------------------	------------------------------------	------------------

[Format] void SJ_GetChunk(SJ sj, long id, long nbyte, SJCK *ck);

[Inputs] sj : Stream j o i n t
 id : Chunk input / output I D
 nbyte : Amount of data requested (unit: bytes)
 ck : Data chunk (data starting address/size)

[Outputs] None

[Function value] None

[Description] This function gets the area (data chunk) that can be read or written from a stream joint when writing or reading a buffer.

Title Function	Function Name	Function Returns data chunk	No 1.4
--------------------------	----------------------	---------------------------------------	------------------

[Format] void SJ_UngetChunk(SJ sj, long id, SJCK *ck);

[Inputs] sj : Stream j o i n t
 id : Chunk input / output I D
 ck : Data chunk (data starting address/size)

[Outputs] None

[Function value] None

[Description] When a data chunk that was gotten could not actually be written or read, this function returns the area (data chunk) that could not be read/written back to the stream joint.

Title Function	Function Name	Function	No
	SJ_PutChunk	Puts data chunk	1.5

[Format] void SJ_PutChunk(SJ sj, long id, SJCK *ck);
 [Inputs] sj : Stream
 id : Chunk
 ck : Data chunk (data starting address/size)
 [Outputs] None
 [Function value] None
 [Description] When a data chunk that was gotten was actually written or read, this function puts the area (data chunk) that was read/written back to the stream joint.

Title Function	Function Name	Function	No
	SJ_GetNumData	Gets the amount of data that can be read/written	1.6

[Format] long SJ_GetNumData(SJ sj, long id);
 [Inputs] sj : Stream
 id : Chunk input/output ID
 [Outputs] None
 [Function value] Amount of data that can be read/written (unit: bytes)

- When writing: Amount of free space
- When reading: Amount of data

 [Description] When writing data to a stream joint, this function gets the free space in the buffer; when reading data, this function gets the amount of data in the buffer.

Title	Function Name	Function	No
Function	SJ_GetUuid	Gets the stream joint UUID	1.7

[Format]	UUID *SJ_GetUuid(SJ sj);
[Inputs]	sj : Stream joint
[Outputs]	None
[Function value]	Stream joint UUID
[Description]	This function gets the stream joint UUID.

Title	Function Name	Function	No
Function	SJ_EntryErrFunc	Registers an error processing function	1.8

Title	Function Name	Function	No
Function	SJ_SplitChunk	Splits a data chunk	1.9

[Format] void SJ_SplitChunk(SJCK *ck, long nbytes, SJCK *ck1, SJCK *ck2);
 [Inputs] ck : Data chunk to be split
 nbytes : Size of ck1 (unit: bytes)
 [Outputs] ck1 : First half of data chunk that was split
 ck2 : Second half of data chunk that was split
 [Function value] Size of data chunk ck2 (unit: bytes)
 [Description] This function splits the data chunk ck into data chunks ck1 and ck2. ck1 has a length indicated by "nbytes". If the size of ck1 is less than the value of "nbytes", the chunk is not split and ck is substituted into ck1. The size of ck2 is then "0". The same value may be specified for ck1 and ck. To simply split a chunk, use the following description.
 [Example]
 SJ_SplitChunk(&ck, 100, &ck1, &ck2);

7.2. Ring Buffer-type Stream Joint

Title Function	Function Name SJRBF_Init	Function Initialization	No 2.1
-------------------	-----------------------------	----------------------------	-----------

[Format] void SJRBF_Init(void);
 [Inputs] None
 [Outputs] None
 [Function value] None
 [Description] This function initializes a ring buffer-type stream joint.

Title Function	Function Name SJRBF_Create	Function Stream joint creation	No 2.2
-------------------	-------------------------------	-----------------------------------	-----------

[Format] SJ SJRBF_Create(char *buf, long bsize, long xsiz);
 [Inputs] buf : R i n g b u f f e r
 bsize : S i z e o f r i n g b u f f e r (unit: bytes)
 xsiz : Size of extra buffer (unit: bytes)
 [Outputs] None
 [Function value] Stream joint
 [Description] This function creates a ring buffer-type stream joint.

7.3. Memory-type Stream Joint

Title Function	Function Name SJMEM_Init	Function Initialization	No 3.1
-------------------	-----------------------------	----------------------------	-----------

[Format] void SJMEM_Init(void);
 [Inputs] None
 [Outputs] None
 [Function value] None
 [Description] This function initializes a memory-type stream joint.

Title Function	Function Name SJMEM_Create	Function Stream joint creation	No 3.2
-------------------	-------------------------------	-----------------------------------	-----------

[Format] SJ SJMEM_Create(char *data, long bsize);
 [Inputs] data : D a t a
 bsize : Data size (unit: bytes) a r e a
 [Outputs] None
 [Function value] Stream joint
 [Description] This function creates a memory-type stream joint.

7.4. General-purpose Stream Joint

Title	Function Name	Function	No
Function	SJUNI_Init	Initialization	4.1

[Format]	void SJUNI_Init(void);
[Inputs]	None
[Outputs]	None
[Function value]	None
[Description]	This function creates a memory-type stream joint.

Title	Function Name	Function	No
Function	SJUNI_Create	Stream joint creation	4.2

[Format]	SJ SJUNI_Create(long mode, char *work, long wksize);
[Inputs]	mode : Linking mode (SJUNI_MODE_SEPA, SJUNI_MODE_JOIN) work : W o r k a r e a wksize : Size of work area
[Outputs]	None
[Function value]	Stream joint
[Description]	This function creates a general-purpose stream joint. If this stream joint will handle separated data chunks, specify SJUNI_MODE_SEPA. If this stream joint will handle linked data chunks, specify SJUNI_MODE_JOIN. Allocate a work area that can accommodate the maximum number of data chunks that the stream joint will handle, and pass "work" and "wksize" as parameters.

```
[Example] #define M A X _ N C K ( 2 5 6 )
           c h a r   w o r k [ S J U N I _ C A L C _ W O R K ( M A X _ N C K ) ];
           sj = SJUNI_Create(SJUNI_MODE_SEPA, work, sizeof(work));
```

Sega Dreamcast™

Dreamcast
ADX Playback Library
User's Manual

1. Introduction

The ADX playback library serves for playback of ADX encoded sound data. By using the functions of this library, playback of compressed sound can be implemented easily. This library controls the ADX playback system that is part of the Dreamcast configuration.

This library has the following features.

- (1) Playback of up to 8 voices with very little CPU load.

CPU load for decoding 44.1 kHz sound data	
_voice	_voices
Approx. 0.6%	Approx. 5%

- (2) 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 GD index playback.

- (3) 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
GD stream playback	Playback of ADX data on GD
GD index playback	Playback of AFS data on GD

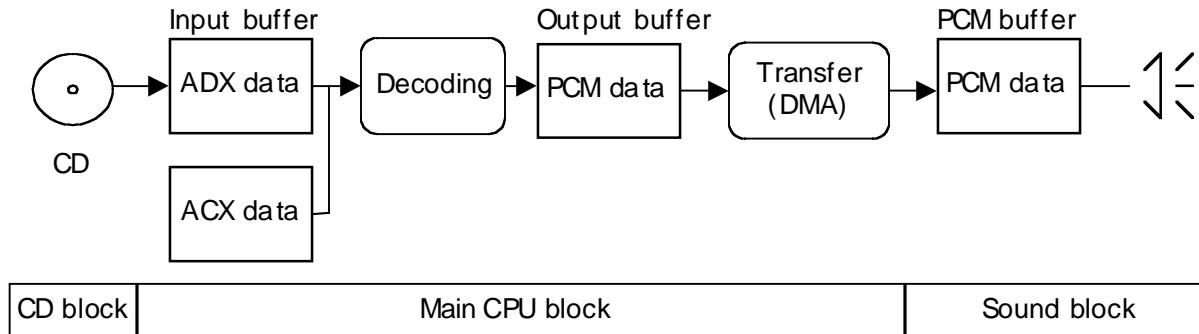
- (4) Simultaneous use together with sound driver possible.
- (5) Multiple GD stream playback operations and GD index playback can be performed simultaneously.
- (6) Seamless loop playback from GD possible.
- (7) ADX file system library (AFS) allows reading of game data or other files from GD during GD stream playback.
- (8) AFS makes it easy to handle a large number of files. (using approximately 2 byte per file for file management)
- (9) Playback of 16-bit non-encoded and 4-bit ADPCM encoded WAV data files possible.

2. ADX Playback System

This section describes the ADX playback system controller by this library.

2.1. System Configuration

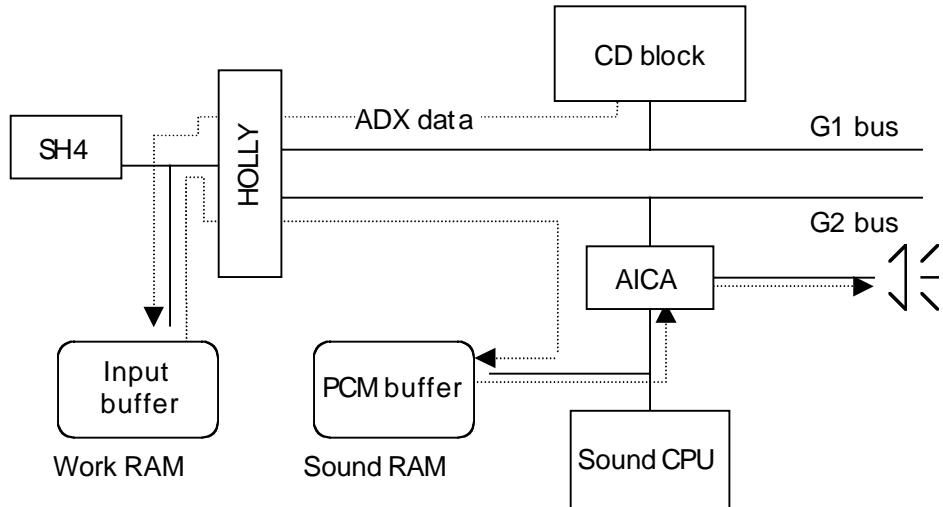
After being decoded in the work RAM of the main CPU, ADX data are transferred to the sound RAM for playback.



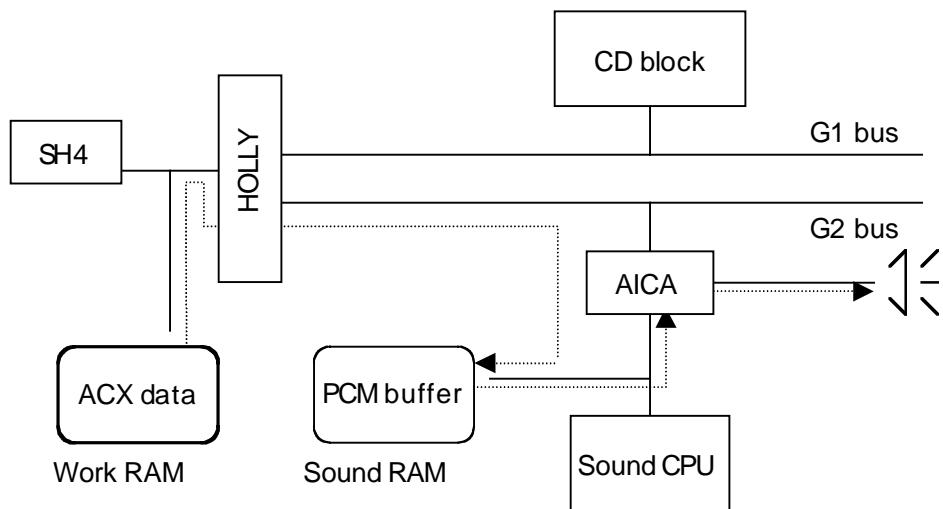
Input buffer	<p>: Stores ADX data read from the GD.</p> <p>The size required in the work RAM depends on the sampling frequency and the number of concurrent playback streams. For a monaural 44 kHz source with 3 concurrent playback streams, approximately 100 Kbyte are required.</p> <p>For playback from memory, this is not required.</p>	
Output buffer	<p>: Stores the decoded 16-bit PCM data.</p> <p>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.</p>	
PCM buffer	<p>: Stores the PCM data in the sound RAM.</p> <p>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.</p> <p>The size is 4040H bytes per channel.</p>	
ADX data	<p>: Data compressed according to the ADX method.</p> <p>One sound file becomes one set of ADX data.</p>	
ACX data	<p>: (ADX sound effect data)</p> <p>Combination of multiple ADX data.</p> <p>During creation, numbers starting from 0 are assigned to the sound data according to the sequence in the file list.</p>	
AFS data	<p>: (ADX file system partition data)</p> <p>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.</p>	

The ADX data flow is shown below.

- (1) Playback of ADX data on GD (GD stream playback, GD index playback)



- (2) Playback of ACX data in memory (memory playback, memory index playback)



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 GD. 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.

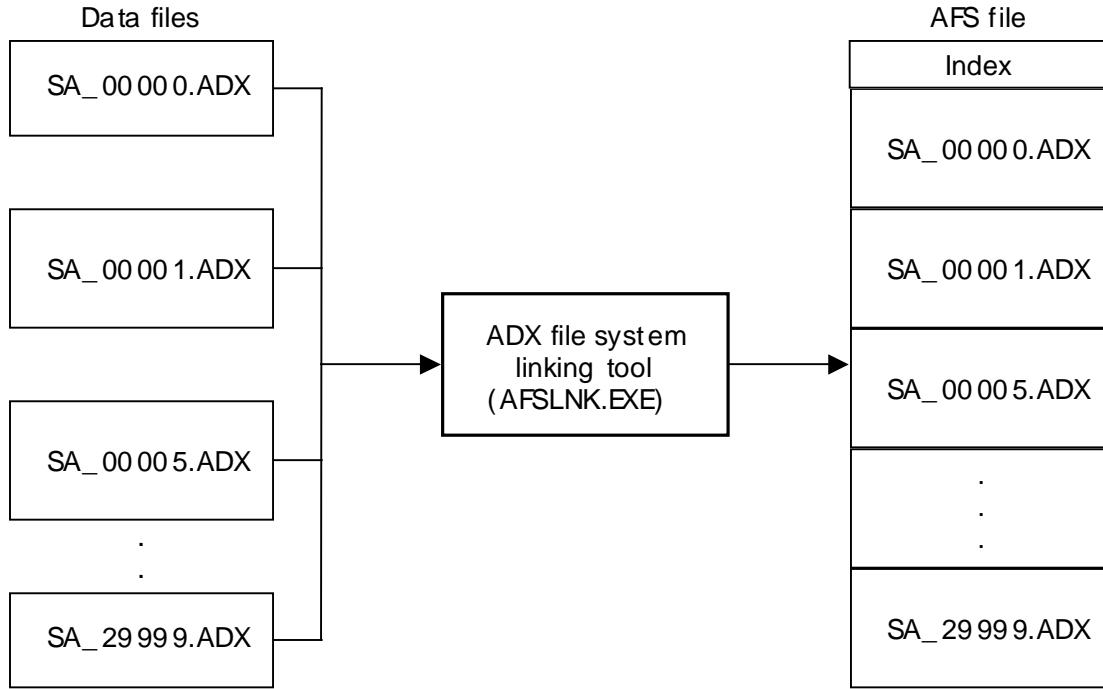


Fig. AFS File Creation

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

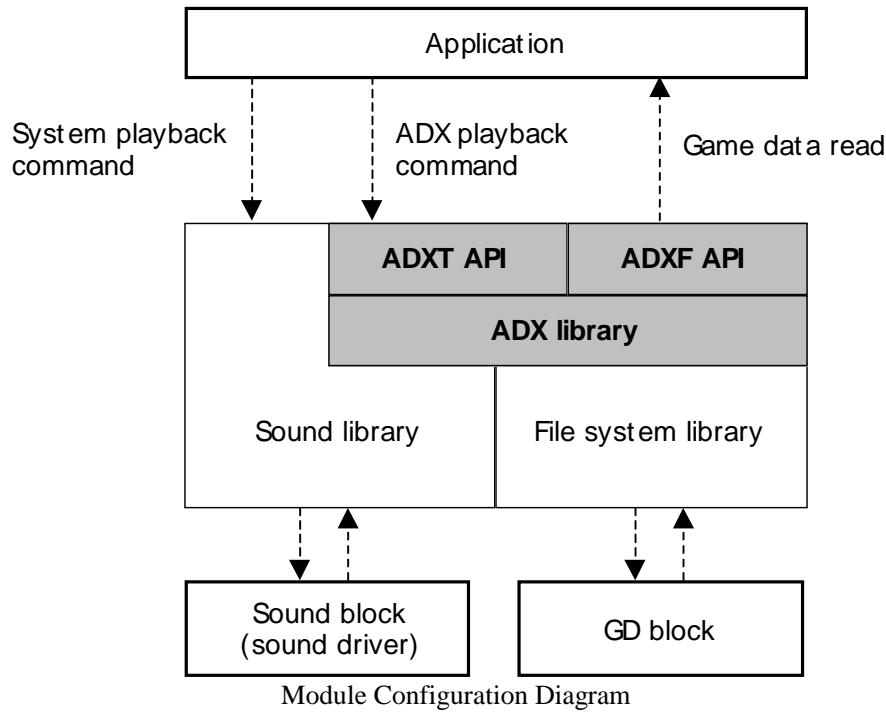
The ADXF_OpenAfs function is used to open the file handle and read game data or similar.

The ADX file system can read data from GD also during GD stream playback.

2.3. Module Configuration

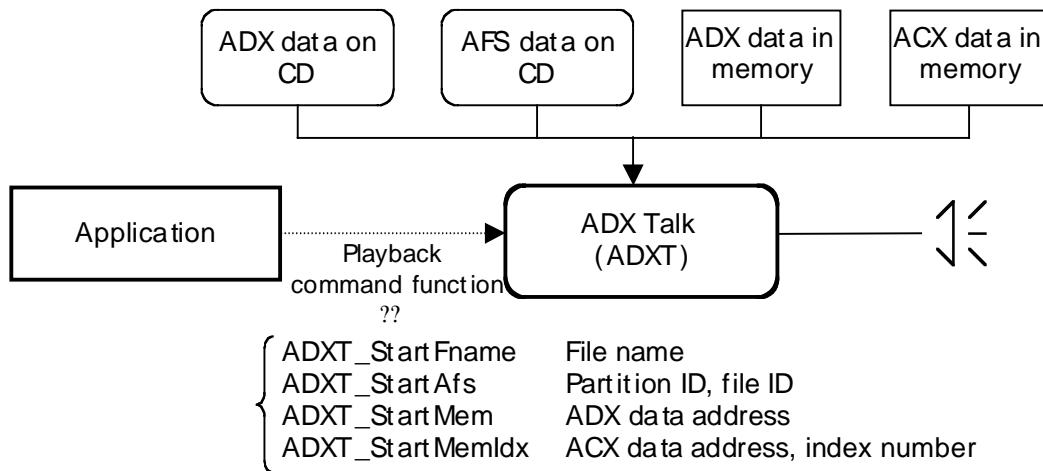
Because the ADX playback system can coexist with the sound library and sound drivers, the programmer can use the sound library and ADX library together. For example, ADX could be used for stream playback while playing a MIDI sequence.

The module configuration of the library is shown below.



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 GD 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.



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 : \ T E M P > a d x e n c d s a m p l e . w a v -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 : \ T E M P > a d x e n c d s a m p l e . w a v -lps1500 -lpe2000000

Converts 'sample.wav' to 'sample.adx', with loop playback from sample 1500 to sample 199999.

* 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.

In addition, the whole file can be set to perform loop playback by using lpa option.

C : \ T E M P > a d x e n c d s a m p l e . w a v -lpa

<Sound Input file format>

File format	WAV or AIFF format
Number of quantization bits	16 or 8
Sampling frequency	Arbitrary

(2) 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.fls Creates a list of WAV files

C:\TEMP>ax "adxencd %%s" wavlist.fls Encodes all WAV files in list

(3) 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.fl
C:\TEMP>adxcat se.fl

Creates the linked file 'se.acx' consisting of all ADX files in the current directory.

(4) 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
C:\TEMP>afslnk pat01.als

Creates the linked file 'pat01. afs' consisting of all 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
C:\TEMP>afslnk pat01.als -dir=pat01

Creates the linked file 'pat01. afs' consisting of all files in the directory 'pat01'.

4. Using the Library Functions

This section describes how to initialize the library, play sound effects in memory, and play ADX data files from GD.

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 */
}
```

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 GDSTM       (1)                      /* GD stream playback */
#define MAX_SFREQ   (44100)                  /* sampling frequency 44100 Hz */

/* work area size */
#define WORKSIZE    (ADXT_CALC_WORK(MAX_CH,     ADXT_PLY_STM,     MAX_GDSTM,
MAX_SFREQ))
/* work area */
static char work[WORKSIZE];

void adx_play_gdstm(void)
{
    ADXT adxt;                                /* ADXT handle */

    adxt = ADXT_Create(MAX_CH, work, WORKSIZE); /* create handle */
    :
    :
    (process 1 game scene)
    :
    :
    ADXT_Destroy(adxt);                     /* delete handle */
}
```

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 */
    :
}
```

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 */
    :
}
```

4.5. Playback of ADX Data on GD (GD Stream Playback)

ADX data on GD 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_GDSTM   (3)          /* number of GD 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_GDSTM,
MAX_SFREQ))
/* work area */
static char work[WORKSIZE];

void adx_play_gdstm(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" */
    :
}
```

4.6. Playback of AFS Data on GD (GD Index Stream Playback)

ADX data on GD 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 "pat01.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_GDSTM   (3)          /* number of GD 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_GDSTM,
MAX_SFREQ))

/* partition information area */
static char ptinfo[ADXF_CALC_PTINFO_SIZE(MAX_PAT_NFILES)];
/* work area */
static char work[WORKSIZE];

void main(void)
{
    /* partition information read-in */
    ADXF_LoadPartition(PAT01, "pat01.efs", ptinfo, 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" */
}
```

4.7. Get Playback Status

The ADXT 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 GD-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;
    }
}
```

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 GD 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

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]	I n i t i a l i z e s t h e A D X l i b r a r y . 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.		
	<pre>/* Normal sound library/driver initialization */ SoundInit(snddrv); /* DMA transfer mode */ sdMemBlkSetTransferMode(SDE_MEMBLK_TRANSFER_MODE_DMA);</pre>		
[Example]	<pre>SoundInit(snddrv); sdMemBlkSetTransferMode(SDE_MEMBLK_TRANSFER_MODE_DMA); /* ADX library initialization */ ADXT_Init();</pre>		

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]	<pre>/* ADX library initialization */ ADXT_Init(); : : : /* ADX library shutdown */ ADXT_Finish();</pre>		

Title	Function	Function Name	No
Macro specification	Work area size calculation	ADXT_CALC_WORK	6.3
[Format]	int ADXT_CALC_WORK(nch, stmflg, nstm, sfreq);		
[Input]	nch : maximum number of playback (monaural:1, stereo:2) stmflg : p l a y b a c k A D X T _ P L Y _ M E M : m e m o r y ADXT_PLY_STM: stream playback	m o d e p l a y b a c k	
	nstm : maximum number of GD streams		
	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 GD streams specifies the number of streams that can be read from G D 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

<sampling frequency 44100Hz, mono>

	Memory playback	GD 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
[Format]	ADXT ADXT_Create(long maxch, void *work, long worksize);		
[Input]	maxch : maximum number of playback channels (mono: 1, stereo: 2) work : w o r k worksize : work area size	a r e a	
[Output]	None		
[Function value]	Pointer to generated ADXT structure element (ADXT handle)		
[Function]	Creates an ADXT handle.		
[Remarks]	maxch should be set to "1" for mono playback only and to "2" for stereo playback. Also when maxch 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]	<pre>/* work area size */ #define WKSIZEx ADXT_CALC_WORK(2, ADXT_PLY_STM, 1, 44100) char *work[WKSIZEx]; /* work area */ ADXT adxt; /* ADXT handle */ adxt = ADXT_Create(2, work, WKSIZEx); /* create ADXT handle */ * : : ADXT_Destroy(adxt); /* delete ADXT handle */</pre>		

Title	Function	Function Name	No
Function specification	Start GD stream playback	ADXT_StartFname	6.6

[Format] void ADXT_StartFname(ADXT adxt, char *fname);
 [Input] adxt : ADXT
 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 *adxdat);		
[Input]	adxt : ADXT adxdat : sound data address		h a n d l e
[Output]	None		
[Function value]	None		
[Function]	Plays sound data in memory specified by adxdat. Allows sound playback with better response than GD stream playback. Delay time is approx. 50 milliseconds.		
[Example]	<pre>/* When playing mono data */ #define WKSIZE ADXT_CALC_WORK(1, ADXT_PLY_MEM, 0, 0) char *work[WKSIZE]; /* work area */ char adxdat[ADXDAT_SIZE]; /* data area */ ADXT adxt; /* ADXT handle */ load_data(adxdat); /* read sound data */ adxt = ADXT_Create(1, work, WKSIZE); /* create ADXT handle */ ADXT_StartMem(adxt, adxdat); /* start playback */ :</pre>		

Title	Function	Function Name	No
Function specification	Start memory index playback	ADXT_StartMem	6.8
[Format]	void ADXT_StartMemIdx(ADXT adxt, void *acxdat, long no);		
[Input]	adxt : ADXT acxdat : A C X no : index number	h a n d l e d a t a a d d r e s s	
[Output]	None		
[Function value]	None		
[Function]	Plays data of specified number within ACX data.		
[Example]	<pre>/* mono data playback */ #define WKSIZE ADXT_CALC_WORK(1, ADXT_PLY_MEM, 0, 0) char *work[WKSIZE]; /* work area */ char acxdat[ADXDAT_SIZE]; /* data area */ ADXT adxt; /* ADXT handle */ load_data(acxdat); /* read sound data */ adxt = ADXT_Create(1, work, WKSIZE); /* create ADXT handle */ */ ADXT_StartMem(adxt, acxdat, 5); /* start playback */ : : ADXT_StartMem(adxt, acxdat, 8); /* start playback */ : :</pre>		

Title	Function	Function Name	No
Function specification	Start GD index playback	ADXT_StartAfs	6.9
[Format]	void ADXT_StartAfs(ADXT adxt, long pat_id, long file_id);		
[Input]	adxt : ADXT pat_id : p a r t i t i o n file_id : file ID	h a n d l e I D	
[Output]	None		
[Function value]	None		
[Function]	Plays data from AFS file on GD. 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]	<pre>/* simultaneous playback of 44 kHz stereo and 22 kHz mono */ /* 44 kHz stereo playback work area */ #define WKSIZE44 ADXT_CALC_WORK(2, ADXT_PLY_STM, 2, 44100) /* 22 kHz mono playback work area */ #define WKSIZE22 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[WKSIZE44]; /* work area */ char *work2[WKSIZE22]; /* work area */ ADXT adxt1, adxt2; /* ADXT handle */ /* partition information read-in */ ADXF_LoadPartition(PATID = 0, "pat01.afs", ptinfo, MAX_NFILES); adxt1 = ADXT_Create(2, work1, WKSIZE44); /* create ADXT handle */ adxt2 = ADXT_Create(1, work2, WKSIZE22); /* create ADXT handle */ ADXT_StartAfs(adxt1, PATID, BGM); /* start playback */ : : ADXT_StartAfs(adxt1, PATID, SRF137); /* start playback */ : :</pre>		

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]	<pre>/* 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 */ adxt = ADXT_Create(2, work, WKSIZE); /* create ADXT handle */ */ ADXT_StartFname(adxt, "MUSIC.ADX"); /* start playback */ : : /* B button pressed for stop */ if (per->press & PDD_DGT_TB) ADXT_Stop(adxt); /* stop playback */</pre>		

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);
}
```

* Completion of playback can be judged by the PLAYEND or STOP status. When a GD 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 : p l a y b a c k s a m p l e 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 - hh*60*60*60) / (60*60); /* minutes */
    ss = (nfrm - (hh*60+mm)*60*60) / 60; /* seconds */
    ff = nfrm % 1000; /* frames */
    disp_time(hh, mm, ss, ff);
}
```

* 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]	<pre>/* 44 kHz stereo data 1 stream playback */ #define WKSIZEx ADXT_CALC_WORK(2, ADXT_PLY_STM, 1, 44100) char *work[WKSIZEx]; /* work area */ ADXT adxt; /* ADXT handle */ long time; /* playback time */ long hh, mm, ss, ff; /* time */ adxt = ADXT_Create(2, work, WKSIZEx); /* create ADXT handle */ */ ADXT_StartFname(adxt, "music.adx"); /* start playback */ : for (;;) { /* playback time display (frame unit 1/100) */ time = ADXT_GetTime(adxt, &nsmpl, &sfreq); hh = time / (60*60*100); /* hours */ mm = (time - hh*60*60*100)/(60*100); /* minutes */ ss = (time - (hh*60+mm)*60*100)/100; /* seconds */ ff = time % 100; /* frames */ disp_time(hh, mm, ss, ff); }</pre>		

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]	Gets 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]	<pre>/* 44 kHz stereo data 1 stream playback */ #define WKSIZEx ADXT_CALC_WORK(2, ADXT_PLY_STM, 1, 44100) char *work[WKSIZEx]; /* work area */ ADXT adxt; /* ADXT handle */ long total_nsmpl; /* total number of sample */ long nsmpl, sfreq; /* playback time */ long percent; /* percent */ adxt = ADXT_Create(2, work, WKSIZEx); /* create ADXT handle */ */ ADXT_StartFname(adxt, "music.adx"); /* start playback */ for (;;) { /* display playback progress (percent) */ if (ADXT_GetStat(adxt) >= ADXT_STAT_PREP) { total_nsmpl = ADXT_GetNumSmpl(adxt); ADXT_GetTime(adxt, &nsmpl, &sfreq); percent = nsmpl * 100 / total_nsmpl; } else { percent = 0; } disp_progress(percent); }</pre>		

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]	<pre>/* 44 kHz stereo data 1 stream playback */ #define WKSIZEx ADXT_CALC_WORK(2, ADXT_PLY_STM, 1, 44100) char *work[WKSIZEx]; /* work area */ ADXT adxt; /* ADXT handle */ long sfreq; /* sampling frequency */ adxt = ADXT_Create(2, work, WKSIZEx); /* create ADXT handle */ */ ADXT_StartFname(adxt, "music.adx"); /* start playback */ for (;;) { /* display playback frequency (Hz) */ if (ADXT_GetStat(adxt) >= ADXT_STAT_PREP) { sfreq = ADXT_GetSfreq(adxt); disp_sampling_frequency(sfreq); } }</pre>		

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
 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

[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.

```

/* 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 specification	Function Set panpot	Function Name ADXT_SetOutPan	No 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		1)
[Output]	None		
[Function value]	None		
[Function]	S e t s t h e o u t p u t p a n p o t . When the setting is AUTO, panning is performed automatically, depending on whether the sound data are mono or stereo. This setting can be made also during sound playback.		

Title Function specification	Function Get panpot setting	Function Name ADXT_GetOutPan	No 6.19
[Format]	long ADXT_GetOutPan(ADXT adxt, long ch);		
[Input]	adxt : ADXT handle ch : channel number (0, 1) ADXT_CH_L(0): left channel, ADXT_CH_R(1): right channel		n u m b e r
[Output]	None		
[Function value]	Panpot setting value (from -15 to +15, -128)		
[Function]	Gets the output panpot setting.		
[Example]	<pre>/* 44 kHz mono data 1 stream playback */ #define WKSIZEx ADXT_CALC_WORK(1, ADXT_PLY_STM, 1, 44100) char *work[WKSIZEx]; /* work area */ ADXT adxt; /* ADXT handle */ long pan; /* current panpot value */ adxt = ADXT_Create(1, work, WKSIZEx); /* create ADXT handle */ ADXT_StartFname(adxt, "srf01.adx"); /* start playback */ for (;;) { /* shift panpot to right */ if (per->on & PDD_DGT_KR) { pan = ADXT_GetOutPan(adxt, 0); pan = min(ADXT_PAN_RIGHT, pan+1); ADXT_SetOutPan(adxt, pan+1); } /* shift panpot to left */ if (per->on & PDD_DGT_KL) { pan = ADXT_GetOutPan(adxt, 0); pan = max(ADXT_PAN_LEFT, pan-1); ADXT_SetOutPan(adxt, pan); } }</pre>		

Title	Function	Function Name	No
Function specification	Set server function call frequency	ADXT_SetSrvFreq	6.20
[Format]	void ADXT_SetSrvFreq(ADXT adxt, long freq);		
[Input]	adxt : ADXT freq : server function call frequency (number of calls per second)	h a n d l e	
[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]	M u s t b e c a l l e d w i t h e a c h V - S y n c . 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] G e t s e r r o r c o d e .
 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, "srf01.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
ADXT_RMODE_NOACT	0	No error recovery
ADXT_RMODE_STOP	1	Automatic stop. Mode changes to ADXT_STAT_STOP.
ADXT_RMODE_REPLAY	2	When data supply from GD 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 WKSIZE44  ADXT_CALC_WORK(2, ADXT_PLY_STM, 2, 44100)
#define WKSIZE22  ADXT_CALC_WORK(1, ADXT_PLY_STM, 2, 22050)

char *work44[WKSIZE44];           /* work area (for BGM) */
char *work22[WKSIZE22];           /* work area (for dialog) */
ADXT adxt;                      /* ADXT handle(for BGM) */
ADXT adxt2;                     /* ADXT handle(for dialog) */
long pan;                        /* current panpot setting */

adxt = ADXT_Create(2, work44, WKSIZE44); /* for BGM ADXT handle */
ADXT_SetAutoRcvr(adxt,ADXT_RMODE_REPEAT); /* auto- repeat mode */
adxt = ADXT_Create(1, work22, WKSIZE22); /* ADXT handle for
dialog */
ADXT_SetAutoRcvr(adxt,ADXT_RMODE_STOP); /* auto stop mode */
:
:
ADXT_StartFname(adxt, "BGM.ADX"); /* start BGM playback */
:
:
ADXT_StartFname(adxt2, "SRF01.ADX"); /* 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 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.		

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_Pause(ADXT adxt, long sw);		
[Input]	adxt : ADXT sw : pause switch (1 = pause, 0 = resume)	h a n d l e	
[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		
	<pre>/* 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, "music.adx"); /* 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); } }</pre>		
	(2) Immediately playing back sound data on a GD		
	<pre>adxt = ADXT_Create(2, work, WKSIZE); /* ADXT handle creation */ * ADXT_Pause(adxt, 1); /* pause */ ADXT_StartFname(adxt, "music.adx"); /* playback start */ /* Sound is not played back, state is PLAYING */ for (;;) { /* When the A button is pressed, sound is immediately output */ if (per->on & PDD_DGT_TA) { ADXT_Pause(adxt, 0); } }</pre>		

Appendix 1. ADX Library

Appendix 1. 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

Appendix 2. 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.

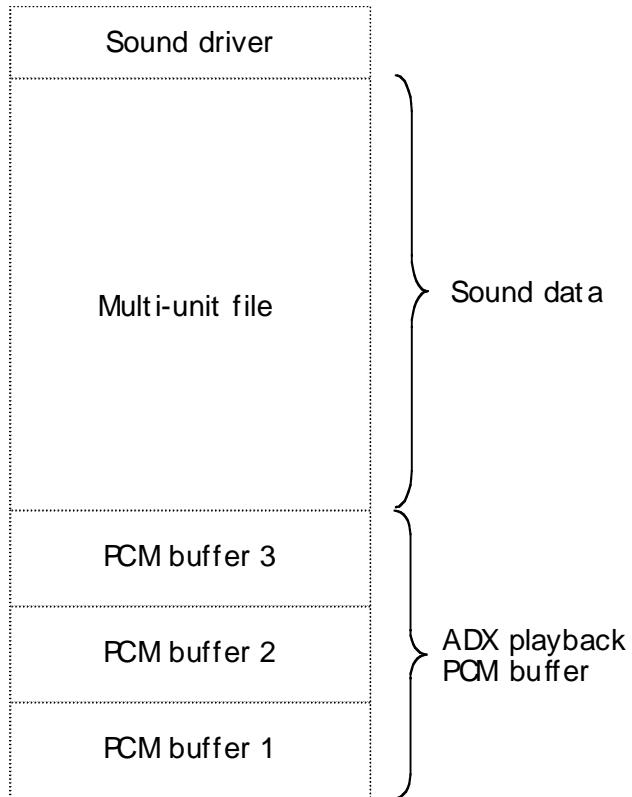


Fig. Sound RAM allocation map

Appendix 3. Increasing Data Reading Speed

The ADX file system can read data also while playing sound from the GD. 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 */
```

- * By increasing the maximum number of GD 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.

Appendix 4. 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 GD, 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 WKSIZE44S ADXT_CALC_WORK(2, ADXT_PLY_STM, 2, 44100)
/* Work area for concurrently playing back 3 streams in 22Khz monoaural */
#define WKSIZE22M 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[WKSIZE44S], wk22[WKSIZE22M]; /* work area */
ADXT adxt_bgm, adxt_srv; /* ADXT handle */

adxt_bgm = ADXT_Create(2, wk44, WKSIZE); /* ADXT handle creation */
adxt_srf = ADXT_Create(1, wk22, WKSIZE); /* 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_srv, "SRF001.ADX");
```

Appendix 5. 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 GD stream of BGM. However, to do this, you need to 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 WKSIZE44S ADXT_CALC_WORK(2, ADXT_PLY_STM, 2, 44100)
/* 2 streams are BGM and movie */
char wk44[WKSIZE44S];                                /* work area */

MWPLY ply;                                              /* MWPLY handle */
ADXT adxt_bgm;                                         /* ADXT handle */

adxt_bgm = ADXT_Create(2, wk44, WKSIZE);           /* 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, "BGM.ADX"); /* start sound playback */
:
if (event occurs)
    mwPlyStartFname(ply, "SCENE01.M1V");      /* play movie */
```

Appendix 6. 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 WKSIZEx    ADXT_CALC_WORK(2, ADXT_PLY_STM, 3, 44100)

char *work[WKSIZEx];           /* work area */
ADXT adxt;                   /* ADXT handle */
ADXFX adxf;                 /* ADXF handle */

adxt = ADXT_Create(2, work, WKSIZEx); /* create ADXT handle */
:
ADXT_StartFname(adxt, "BGM.ADX"); /* start sound playback */
:
while ( ! ADXT_IsIBufSafty(adxt) )
;
njLoadTextrure(&tlist);
```

Appendix 7. ADX Multi-stream System Arrangement

The ADX multi-stream system can be used to play, in parallel, audio and movie files that are stored separately in GD-ROM.

Up to four streams can be played simultaneously, if they are from audio files on GD-ROM. For example, it is possible to play 44KHz stereo BGM, sound effects, and two 44KHz monaural dialogue streams simultaneously. Furthermore, when playing a movie file from MPEG Sofdec F/X, it is possible to play a separate audio file simultaneously.

By using the ADX file system and the ADX multi-stream system together, it becomes possible to load graphics data, etc., while simultaneously playing audio data from a GD-ROM.

The following diagram illustrates the ADX mutli-stream system arrangement.

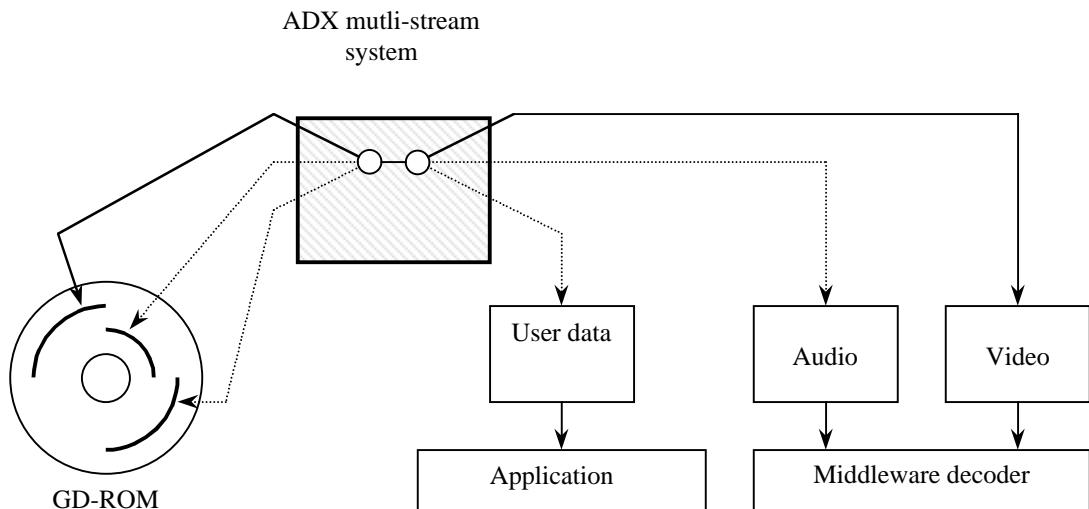


Fig. ADX Multi-stream System Arrangement

The ADX multi-stream system controls the amount of data that flows. Each stream has its own data buffer, and data is loaded whenever the amount of data falls below the number of sectors that triggers the start of a reload. The default number of sectors that triggers a reload is set at 85% of the buffer size.

The following diagram illustrates the ADX data flow control arrangement.

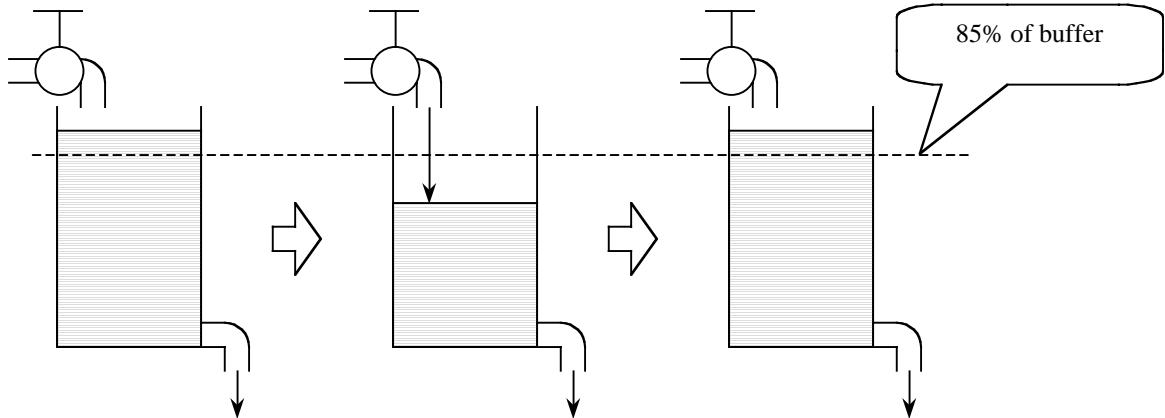


Fig. Data Flow Control Arrangement

Appendix 8. Sofdec F/X Movie and Audio Multi-streaming

When multi-streaming a Sofdec F/X movie and audio, the work area can be eliminated if the buffers are set up as described below.

```
#define NADX          ( 3 )      // Number of audio streams for parallel
                                playback
#define NSFD           ( 2 )      // Number of movie streams for parallel
                                playback

#define ADX_NSTM        (NADX*( (NSFD != 0) ? (NSFD+1) : 1 ))
#define SFD_NSTM        (NSFD+( (NADX != 0) ? 1 : 0 ))
#define MOVIE_BPS        (300*1024*8)

// Movie playback handle generation
cprm.max_bps = MOVIE_BPS * SFD_NSTM;                      // Maximum bit rate
for (i=0; i<NSFD; i++ )
    ply[i] = mwPlyCreateSofdec(&cprm);
// ADX playback handle generation
wksize=ADXT_CALK_WORK(2, ADXT_PLY_STM, ADX_NSTM, 44100));
    for (i=0; i<NADX; i++) {
        wk = syMalloc(wksize);
        adxt[i] = ADXT_Create(2, wk, wksize);
    }
// Audio playback start
ADXT_StartFname(adxt[0], "music.adx");
for (;;) {
    if (Button A was pressed)
        mwPlyStartFname(ply[0], "sample.sfd"); // Movie playback start
}
```

Appendix 9. Loop Playback

ADX can perform seamless loop playback of a portion of audio data. The ADXT_SetLpFlg function can be used to set whether to loop or not. If a loop is cancelled while playback is in progress, the audio data in question plays until its end is reached. A loop can only be cancelled once playback is in progress.

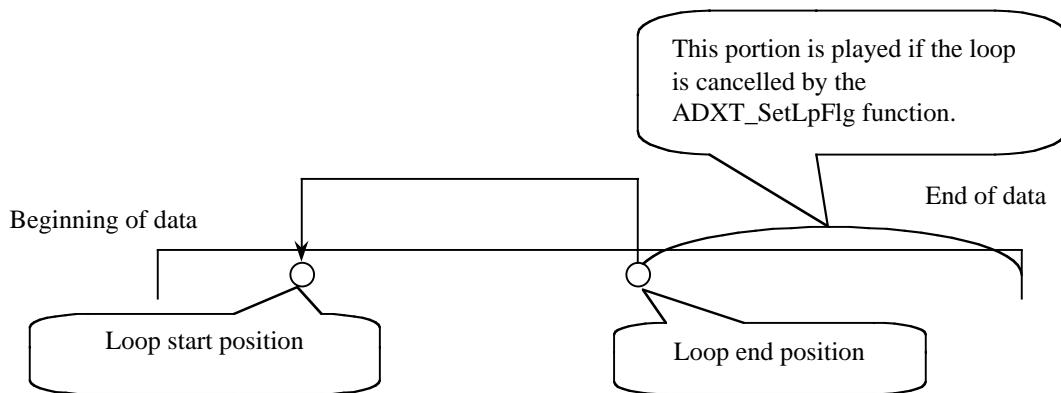


Fig. Loop Playback

Appendix 10. Method for Eliminating the Time Lag Before Audio Is Output

The flow of operations until audio is output is shown below.

(3) Audio data is read from a GD-ROM.



The header of the audio data is analyzed.



The audio data is decoded.



The data is stored in the PCM stream buffer in sound RAM.



Audio output starts

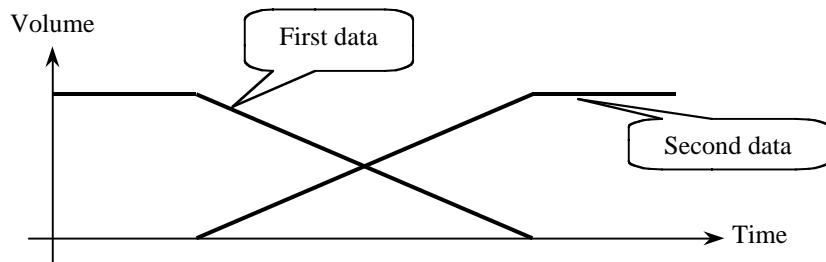
As you can see, audio output does not start right away when he ADXT_StartFname function is executed. A method for eliminating the time lag before the audio output starts is shown below.

```
—  
—  
ADXT_Pause(adxt, ON); /* Pause  
*/  
  
ADXT_StartFname(adxt, "sample.adx"); /* Playback start */  
  
for (;;) {  
    if —(Timing for audio output) {  
        ADXT_Pause(adxt, OFF); /* Release pause  
*/  
    }  
    —  
    —
```

Appendix 11. Crossfading

Crossfading makes it possible to switch between game scenes naturally without suddenly interrupting the music.

Fig. Change in Volume Over Time When Crossfading



```
vol1 = 0;
vol2 = -1000;
for (;;) {
    if (Timing for starting playback of first data) {
        ADXT_SetOutVol(adxt1, vol1);
        ADXT_StartFname(adxt1, "sample1.adx");
    }
    if (Timing for starting playback of second data) {
        ADXT_SetOutVol(adxt2, vol2);
        ADXT_StartFname(adxt2, "sample2.adx");
    }
    if ( ADXT_GetStat(adxt2) == ADXT_STAT_PLAYING ) {
        ADXT_SetOutVol(adxt1, vol1-=10);
        ADXT_SetOutVol(adxt2, vol2+=10);
    }
}
```

Sega Dreamcast™

***Dreamcast
ADX File System
User's Manual***

1. Outline

1.1 Purpose

This system is designed to allow data reading while streaming sound playback is in progress.

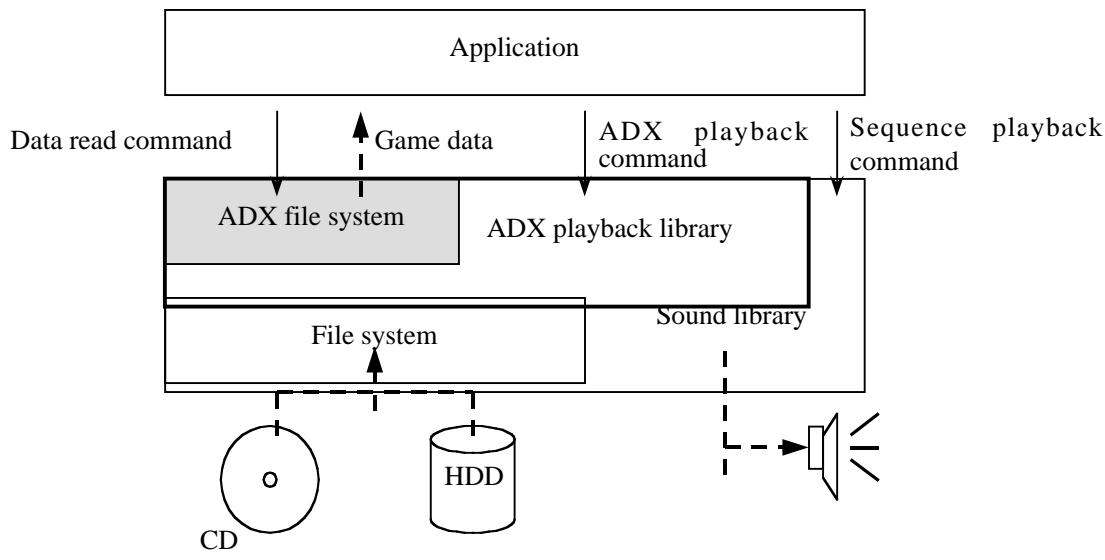
1.2 Features

The features of this system are as follows.

- (1) Data can be read while streaming sound playback is in progress.
- (2) Besides ISO9660 files (conventional files), AFS files consisting of multiple linked files can be read.
- (3) Multiple files can be opened at the same time and read in parallel.

1.3 Module Configuration

The module configuration of the ADX file system is shown below.



2. Basic Points

2.1 Terminology

Some important terms related to the ADX file system are explained below.

Table 2-1 Terminology

Term	Meaning
ISO9660 file	ISO9660 format file (conventional file)
AFS file	Multiple ISO9660 files linked on a PC. Up to 65535 files can be linked.
Partition	Files inside an AFS file are handled as files and the AFS file is handled as a partition. This term is used in the explanation of the disk read process by the library.
Inside file	File linked inside an AFS file (partition) Maximum size is 128 MByte, in 2048 byte (sector) units
Partition ID	Identifier for a partition
File ID	Identifier for an inside file

3. Using the ADXF File System

Data are transferred from disk to memory by the following process.

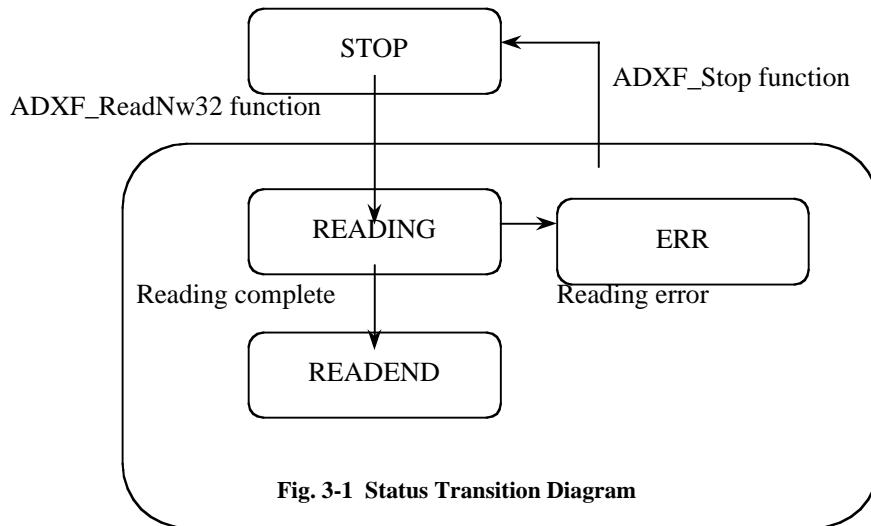
- (1) Open file (ADXF_Open/ADXF_OpenAfs)
- (2) Issue read request (ADXF_ReadNw32)
- (3) Monitor status until data read process is complete (ADXF_GetStat)
- (4) When process is complete, close file (ADXF_Close)

The handle status is as shown below.

Table 3-1 Handle Status

Status	Description
STOP	Data reading is stopped
READING	Data reading is in progress
READEND	Data reading is completed
ERR	Error

The handle status changes from STOP to READING in response to the ADXF_ReadNw32 function. When the specified number of sectors has been read, the status automatically changed to READEND.



3.1 Reading ISO9660 Files

A sample program for reading ISO9660 files is shown below.

```
<Sample program>
/* application main function */
void main(void)
{
    ADXF      adxf;                      /* ADXF handle          */
    long      rd_nsct = 1000;             /* number of read sectors*/
    char     *rd_buf = (char *)0x8C800000; /* buffer address       */

    /*
     * Initialization of hardware and other components
     */
    ADXT_Init();                         /* initialize library   */
    adxf = ADXF_Open("SAMPLE0.DAT");     /* open file            */
    ADXF_ReadNw32(adxf, rd_nsct, rd_buf); /* start reading        */
    for (;;) {
        if (ADXF_GetStat(adxf) == ADXF_STAT_READEND)           /* check for read end*/
            break;
        /*
         * wait for V-Sync
         */
    }
    ADXF_Close(adxf);                   /* close file           */
}
```

3.2 Accessing AFS Files

The AFS file system allows easy access to a large number of files. Multiple files are linked beforehand and stored on the CD. The ADX file system then can read any specified file within the AFS file.

3.2.1 Creating an AFS File

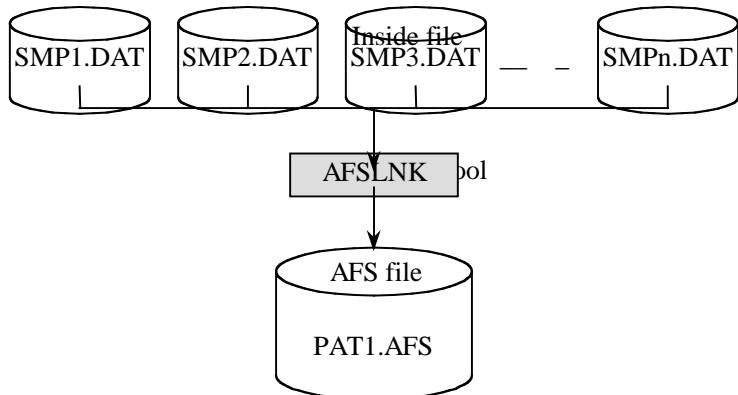


Fig. 3-2 Creating an AFS File

An AFS file contains linked files with ADX data, graphics data etc. The afslnk.exe program is used to link the files, using a file list as argument, as shown below.

```
C:\TEMP>dir /b *.adx > pat01.als  
C:\TEMP>afslnk pat01.als
```

Creates the linked file 'pat01. afs' consisting of all files in the current directory.

To link data from other locations besides the current

```
C:\TEMP>dir /b pat01\*.adx > pat01.als  
C:\TEMP>afslnk pat01.als -dir=pat01
```

Creates the linked file 'pat01. afs' consisting of all files in the directory 'pat01'.

3.2.2 Reading an Inside File

Because this section describes how to read an inside file, an AFS file here is called a partition.

Access to an inside file is made using 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 inside file.

```
static char ptinfo[ ADXF_CALC_PTINFO_SIZE(5) ];           /*partition information area */
long          pi_sz;                                     /* partition information size */

ADXT_Init();                                         /* initialize library */
ADXF_LoadPartition(0, "PAT0.AFS", ptinfo, 5); /* load partition */
```

To handle multiple partitions, use the following procedure. This allows using the partition information area of one partition for multiple partitions.

```
static char ptinfo[ ADXF_CALC_PTINFO_SIZE(256) ];           /* partition information area */
long          pi_sz;                                     /* partition information size */

ADXT_Init();                                         /* initialize library */
ADXF_LoadPartition(0, "PAT0.AFS", ptinfo, 5); /* load partition */
pi_sz = ADXF_GetPtinfoSize(0);                      /* get partition information size */
ADXF_LoadPartition(1, "PAT1.AFS", ptinfo+pi_sz, 8);
:
```

(2) Read data in partition

The ADXF_OpenAfs function serves for opening files, reading graphic data, etc.

```
ADXF      adxf;                                /* ADXF handle */
Long       pati_id=1;                            /* partition ID */
long       file_id = 3;                           /* file ID */
long       rd_nsct = 1000;                         /* number of read sectors */
char      *rd_buf = (char *)0x8C800000;           /* buffer address */

adxf = ADXF_OpenAfs(pati_id, file_id);           /* open file */
ADXF_ReadNw32(adxf, rd_nsct, rd_buf);           /* start reading */
for (;;) {
    if (ADXF_GetStat(adxf) == ADXF_STAT_READEND) /* check for end of reading */
        break;
    /* wait for V-Sync */
}
ADXF_Close(adxf);                                /* close file */
```

3.2.3 Reading an Add-on Inside File

In the case of adding or changing inside file at the middle stage of the development, combining AFS files leads to poor work performance. Explanation on add-on function is omitted because it is still not packaged in this current version.

4. Data Specifications

A table of library data organization is shown below.

Table 4-1 Data Organization

Data name	Function	No.
Constant		
ADXF_STAT_...	Handle status	1.1
ADXF_SEEK_...	Seek type	1.2
Data type		
ADXF	ADXF handle	2.1

4.1 Constants

Title Data	Data Name ADXF_STAT_...	Data Handle status	No 1.1
---------------	----------------------------	-----------------------	-----------

The constants shown below indicate the handle status.

Constant	Description
ADXF_STAT_STOP	Data reading stopped
ADXF_STAT_READING	Data reading in progress
ADXF_STAT_READEND	Data reading ended
ADXF_STAT_ERR	Error on reading data

Title Data	Data Name ADXF_SEEK_...	Data Seek type	No 1.2
---------------	----------------------------	-------------------	-----------

The following constants are used as reference for moving the access pointer. They are specified as arguments of the ADXF_Seek function.

Constant	Description
ADXF_SEEK_SET	Top of file
ADXF_SEEK_CUR	Current position
ADXF_SEEK_END	End of file

4.2 Data Type

Title Data	Data Name ADXF	Data ADXF handle	No 2.1
---------------	-------------------	---------------------	-----------

Stores information about file access for each file. Generated by the functions ADXF_Open and ADXF_OpenAfs. These data are referenced in almost every file access.

5. Function Specifications

The library functions are listed in the table below.

Table 5-1 Function List

Function name	Function	No.
Library initialization and shutdown		
ADXF_Init	Library initialization	1.1
ADXF_Finish	Library shutdown	1.2
ADXF_LoadPartition	Load partition information	1.3
ADXF_AddPartition	Read add-on partition information	1.4
ADXF_GetPtinfoSize	Get partition size information	1.5
Data loading		
ADXF_Open	Open file (ISO9660 format)	2.1
ADXF_OpenAfs	Open file (AFS format)	2.2
ADXF_Close	Close file	2.3
ADXF_ReadNw32	Start data reading	2.4
ADXF_ReadSj32	Start reading to stream joint	2.5
ADXF_Stop	Stop data reading	2.6
ADXF_ExecServer	Server function	2.7
Control access pointer		
ADXF_Seek	Move access pointer	3.1
ADXF_Tell	Get access pointer	3.2
Information loading		
ADXF_GetFsizeSct	Get file size	4.1
ADXF_GetNumReqSct	Get read request information	4.2
ADXF_GetNumReadSct	Get number of read sectors	4.3
ADXF_GetStat	Get handle status	4.4

5.1 Library Initialization and Shutdown

These functions serve to initialize and terminate the library.

Title Function	Function Name	Function	No
	ADXF_Init	Library initialization	1.1

[Format]	void ADXF_Init(void);
[Input]	None
[Output]	None
[Function value]	None
[Function]	Initializes the library
[Remarks]	Since this is called by the ADXT_Init function (ADX playback library initialization), it normally needs not be called by an application.

Title Function	Function Name	Function	No
	ADXF_Finish	Library shutdown	1.2

[Format]	void ADXF_Finish(void);
[Input]	None
[Output]	None
[Function value]	None
[Function]	Performs shutdown processing for the library.

Title Function	Function Name ADXF_LoadPartition	Function Read partition information	No 1.3
-------------------	-------------------------------------	--	-----------

[Format] long ADXF_LoadPartition(long ptid, char *fname, void *ptinfo, long nfile);
 [Input] ptid: partition ID (0 - 255)
 fname: AFS file name
 ptinfo: partition information read-in area
 nfile : number of files
 [Output] None
 [Function value] Error code
 [Function] Reads AFS file partition information and sets the partition ID.

Title Function	Function Name ADXF_AddPartition	Function Read add-on partition information	No 1.4
-------------------	------------------------------------	---	-----------

[Format] long ADXF_AddPartition(long ptid, char *fname, void *ptinfo, long nfile);
 [Input] ptid: partition ID (from 0 to 255)
 fname : AFS file name
 ptinfo: partition information read-in area
 nfile : number of files
 [Output] None
 [Function value] Error code
 [Function] Reads AFS file partition information and sets the add-on partition ID.

Title Function	Function Name ADXF_GetPtinfoSize	Function Get partition information size	No 1.5
-------------------	-------------------------------------	--	-----------

[Format] long ADXF_GetPtinfoSize(long ptid);
[Input] ptid: partition ID
[Output] None
[Function value] Partition information size (unit: bytes)
[Function] Gets the size of the set partition information.

5.2 Data Loading

These functions serve for reading data.

Title Function	Function Name	Function Open file (ISO9660 format)	No
	ADXF_Open		2.1

[Format] ADXF ADXF_Open(char *fname, void *atr);
[Input] fname: file name
atr : file attribute
[Output] None
[Function value] ADXF handle, NULL in case of error
[Function] Opens the specified file and returns the ADXF handle.

Title Function	Function Name	Function Open file (AFS format)	No
	ADXF_OpenAfs		2.2

[Format] ADXF ADXF_OpenAfs(long ptid, long flid);
[Input] ptid: partition ID
flid: file ID
[Output] None
[Function value] ADXF handle, NULL in case of error
[Function] Opens AFS file specified by partition ID and file ID, and returns ADXF handle.

Title Function	Function Name ADXF_Close	Function Close file	No 2.3
-------------------	-----------------------------	------------------------	-----------

[Format] void ADXF_Close(ADXF adxf);
 [Input] adxf : ADXF handle
 [Output] None
 [Function value] None
 [Function] Closes the specified ADXF handle.

Title Function	Function Name ADXF_ReadNw32	Function Start data read	No 2.4
-------------------	--------------------------------	-----------------------------	-----------

[Format] long ADXF_ReadNw32(ADXF adxf, long nsct, void *buf);
 [Input] adxf: ADXF handle
 nsct: read data volume (unit: sectors)
 buf : read-in area
 [Output] None
 [Function value] Read data volume (unit: sectors)
 [Function] Issues data read request. When requested access is completed, the access pointer advances to the sector specified by nsct.
 [Remarks] The buf address boundary must be 32 byte-aligned
 When the specified number of read sectors extends beyond the file end, the number of sectors to the file end is returned, and file read is performed to the file end.

Title Function	Function Name ADXF_ReadSj32	Function Start reading to stream joint	No 2.5
-------------------	--------------------------------	---	-----------

[Format] long ADXF_ReadSj32(ADXF adxf, long nsct, SJ sj);
 [Input] adxf : ADXF handle
 nsct : read data volume (unit: sectors)
 SJ : stream joint
 [Output] None
 [Function value] Read data volume (unit: sectors)
 [Function] Issue data read request to stream joint.
 Buffer size of stream joint needs to be multiple of whole number.
 When user starts to read data from stream joint, data is automatically read into stream joint.
 When requested access is completed, the access pointer advances to the sector specified by nsct.

Title Function	Function Name ADXF_Stop	Function Stop data read	No 2.6
-------------------	----------------------------	----------------------------	-----------

[Format] long ADXF_Stop(ADXF adxf);
 [Input] adxf : ADXF handle
 [Output] None
 [Function value] Access pointer at stop point
 [Function] Stops the data loading process.

Title Function	Function Name ADXF_ExecServer	Function Server function	No 2.7
-------------------	----------------------------------	-----------------------------	-----------

[Format] void ADXF_ExecServer(void);
[Input] None
[Output] None
[Function value] None
[Function] Updates the internal state.
[Remarks] Applications do not need to call this function, because it is called automatically by the V-Sync interrupt.

5.3 Access Pointer Control

These functions serve to control the access pointer.

Title Function	Function Name ADXF_Seek	Function Move access pointer	No 3.1
-------------------	----------------------------	---------------------------------	-----------

[Format] long ADXF_Seek(ADXF adxf, long pos, long type);
 [Input] adxf : ADXF handle
 pos: access pointer move amount (unit: sectors)
 type: move reference (seek type: ADXF_SEEK_-)
 [Output] None
 [Function value] Access pointer position
 [Function] Moves the access pointer to a position at "pos" sectors from "type". When a move beyond the end of the file is specified, the access pointer moves to the end of the file.

Constant	Description
ADXF_SEEK_SET	Top of file
ADXF_SEEK_CUR	Current position
ADXF_SEEK_END	End of file

Title Function	Function Name ADXF_Tell	Function Get access pointer	No 3.2
-------------------	----------------------------	--------------------------------	-----------

[Format] long ADXF_Tell(ADXF adxf);
 [Input] adxf: ADXF handle
 [Output] None
 [Function value] Access pointer position (unit: sectors)
 [Function] Gets the access pointer position.

5.4 Getting Information

Title Function	Function Name ADXF_GetFsizeSct	Function Get file size	No 4.1
-------------------	-----------------------------------	---------------------------	-----------

[Format] long ADXF_GetFsizeSct(ADXF adxf);
[Input] adxf: ADXF handle
[Output] None
[Function value] File size (unit: sectors)
[Function] Gets the size of the specified file.

Title Function	Function Name ADXF_GetNumReqSct	Function Get read request information	No 4.2
-------------------	------------------------------------	--	-----------

[Format] long ADXF_GetNumReqSct(ADXF adxf, long *seekpos);
[Input] adxf: ADXF handle
[Output] seekpos: read position
[Function value] Requested read data volume (unit: sectors)
[Function] Gets the read position and data volume requested by the ADXF_ReadNw32 function.

Title Function	Function Name ADXF_GetNumReadSct	Function Get actual number of read sectors	No 4.3
-------------------	-------------------------------------	---	-----------

[Format] long ADXF_GetNumReadSct(ADXF adxf);
 [Input] adxf: ADXF handle
 [Output] None
 [Function value] Read data volume (unit: sectors)
 [Function] Gets the amount of data actually read.

Title Function	Function Name ADXF_GetStat	Function Get handle status	No 4.4
-------------------	-------------------------------	-------------------------------	-----------

[Format] long ADXF_GetStat(ADXF adxf);
 [Input] adxf: ADXF handle
 [Output] None
 [Function value] ADXF handle internal status (status: ADXF_STAT_-)
 [Function] Gets the internal status of the ADXF handle.

Constant	Description
ADXF_STAT_STOP	Data reading stopped
ADXF_STAT_READING	Data reading in progress
ADXF_STAT_READEND	Data reading completed
ADXF_STAT_ERR	Error on reading data

6. Appendix

6.1 Door Open check

The following is a sample program that checks whether the door is open.

<Sample program>

```
/* GDFunction that is initiated when a GD file system error occurs */
void UsrGdErrFunc(void *obj, Sint32 errcode)
{
    if (errcode == GDD_ERR_TRAYOPEND || errcode == GDD_ERR_UNITATTENT) {
        ADXF_Finish();                                /* ADXF end processing */

        sbExitSystem();                               /* Shinobi library end processing */
        syBtExit();                                  /* Jump to simple player */
    }
}

/* Function registered by the user for the V-SYNC interrupt */
void UsrVsyncFunc(void)
{
    Sint32 dstat;

    /* Door open check */
    dstat = gdFsGetDrvStat();
    if (dstat == GDD_DRVSTAT_OPEN || dstat == GDD_DRVSTAT_BUSY) {
        gdFsReqDrvStat();
    }
}

/* Main body of application */
void main(void)
{
    :
    :
    /* Registration of GD file system error callback function */
    gdFsEntryErrFuncAll((void *)UsrGdErrFunc, NULL);
        njSetVSyncFunction(UsrVsyncFunc);
    :
    :
}
```

6.2 Read Error

If a read error occurs in the ADXF file system, the handle status changes to the error state. Therefore, applications should monitor the handle status and take appropriate action when a read error occurs.

```
ADXF_ReadNw32(adxf, nsct, buf);
for (;;) {
    if (ADXF_GetStat(adxf) == ADXF_STAT_ERROR) {
        /* Read error processing */
    }
}
```

6.3 Reducing Seek Noise

6.3.1 Directory Changes

When changing directories, a seek is generated and reading data takes about one second longer. Normally, game data is located in the outermost tracks of the disc, while directory information resides in the innermost tracks. As a result, when a directory change occurs, the head must execute a seek to the innermost tracks of the disk in order to get the directory information. Because a seek from the outermost tracks to the innermost tracks requires about 600ms to execute, the reading of data is delayed by more than one second as the head travels from the outer tracks to the inner tracks and then back again.

By using AFS files, and positioning the data that is to be used in close proximity, the data can be managed in the same manner as directories and the noise caused by seek operations can be reduced.

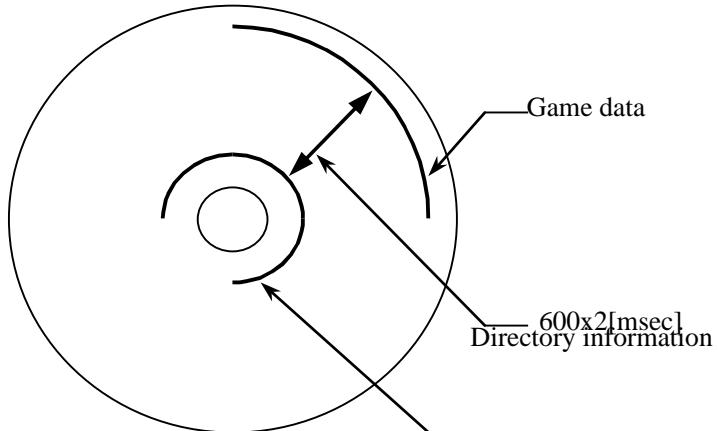


Fig.6-1 Directory Change

6.3.2 Effective Use of AFS Files

Instead of creating different AFS files for BGM data, dialogue data, etc., create separate AFS files for each scene. In a case where dialog will be played back and graphics data will be loaded while BGM plays by means of multistreaming, the seek time will be longer if separate AFS files are created for each type of data.

The frequency of seeks that occur when using multistreaming can be reduced by creating a separate AFS file for each scene.

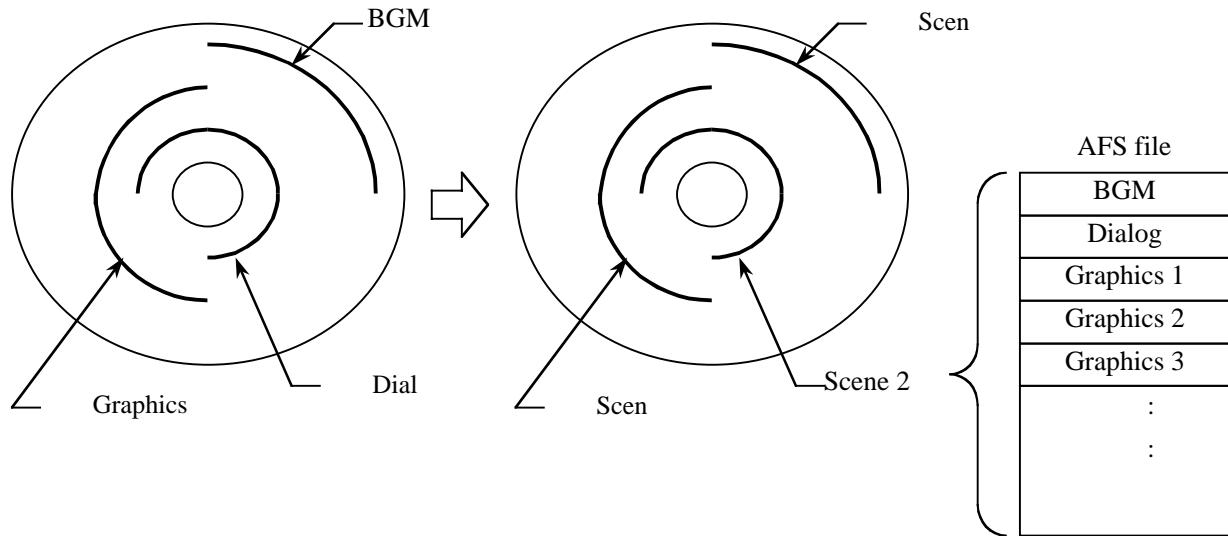


Fig.6-2 Effective Use of AFS Files

6.4 Reading Data Faster

The ADX file system permits game data to be read while audio is being played from a GD-ROM. However, under the default settings, because the buffer is reloaded if the data in the ADX playback buffer falls below 85% of capacity, reading multiple units of game data can result in game data and audio data being loaded in alternation.

Game data can be loaded faster by using the ADXT_SetReloadSct function to adjust the level at which audio data reloading starts. Because audio playback can be performed smoothly if a minimum of one second of data is loaded in the input buffer, specify one second of audio data as the minimum amount of data needed in the ADX playback buffer before reloading starts.

For example, because 44.1KHz stereo sound requires about 50K of data per second, 25 sectors (1 sector = 2048 bytes) should be specified. Furthermore, because it is possible to increase the interval between buffer reloads by increasing the size of the input buffer, the speed at which game data is loaded can be increased even further.

Because the specifications for the ADXT_SetReloadSct function are still tentative, the function is not included in the reference manual. In the future, it will be possible to load game data quickly without needing to specify this function.

```
/* When playing 44KHz stereo data */
#define WKSIZE          ADXT_CALC_WORK(2, ADXT_PLY_STM, 3, 44100)

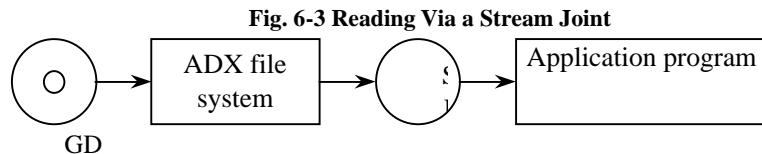
char work[ WKSIZE ] ;                                /* Work area   */
ADXT adxt;                                         /* ADXT handle */
ADXF adxf;                                         /* ADXF handle */

adxt = ADXT_Create(2, work, WKSIZE); /* ADXT handle generation      */
ADXT_SetReloadSct(adxt, 25);                      /* Setting of reloading
amount */                                           :
ADXT_StartFname(adxt, "BGM.ADX"); /* Sound playback start */
:
adxf = ADXF_Open("GAMEDAT1.BIN", NULL);           /* File open   */
ADXF_ReadNw32(adxf, nsct, buf);                   /* Data read   */
while ( ADXF_GetStat(adxf) != ADXF_STAT_READEND );
ADXF_Close(adxf);                                 /* File close */
:
(If GAMEDAT1 is read within two seconds, audio data reloading does not
occur)
:
adxf = ADXF_Open("GAMEDAT2.BIN", NULL);           /* File open   */
ADXF_ReadNw32(adxf, nsct, buf);                   /* Data read   */
while ( ADXF_GetStat(adxf) != ADXF_STAT_READEND );
ADXF_Close(adxf);                                 /* File close */
```

- The size of the input buffer can be increased by increasing the maximum number of GD-ROM streams (the third parameter of ADXT_CALC_WORK). Each increase of "1" in the value of the parameter provides one more second's worth of extra space in the buffer. In addition, although BGM and dialogue play simultaneously, if it is certain that no dialogue will be played while data is being read, the portion of the buffer that is allocated for dialogue can be allocated for reading data instead.

6.5 Reading Via a Stream Joint

An application can perform streaming easily by reading data via a stream joint. If there is free space in a stream joint, the ADX file system automatically reads data.



As an example, this function makes it possible to read a Sofdec file as an inside file of an AFS file.

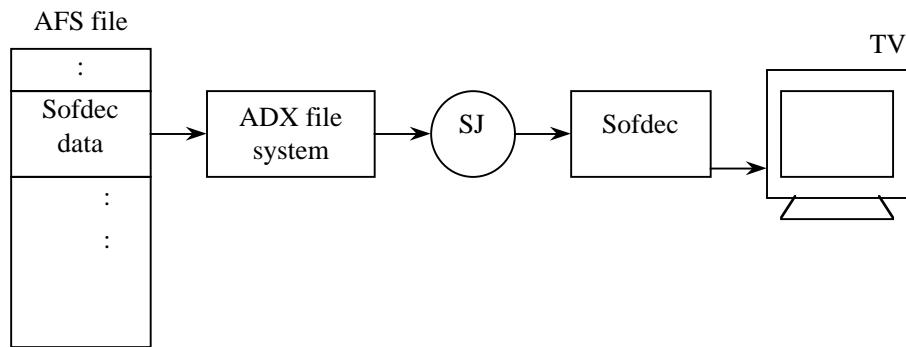


Fig. 6-4 Sofdec Playback Via a Stream Joint

```
<Sample program>
adx = ADXF_OpenAfs(PAT0, FID1);
sj = mwPlyGetInputSj(sj);
ADXF_ReadSj32(adx, fsize, sj);
:
```

The following sample program reads and decompresses compressed data of a variable block length by using a stream joint.

```
<Sample program>
char      buf[ 64*1024+2048 ] ;                                // Maximum block length: 2048
bytes

sj = SJRBF_Create(buf, 64*1024, 2048); // Specification of extra area for
the maximum block length
// This specification guarantees the allocation of 2048 continuous
bytes.
ADXF_ReadSj32(adx, 1000, sj);
for (;;) {
    SJ_GetChunk(sj, SJ_LIN_DATA, 2048, &ck);
    nused = user_decode(ck.data, ck.len); // User decoder
    SJ_SplitChunk(&ck, nused, &ck, &ck2);
    SJ_PutChunk(sj, SJ_LIN_FREE, &ck);
    SJ_UngetChunk(sj, SJ_LIN_DATA, &ck2);
}
```



Dreamcast
Simple Animation
Library
User's Manual

1. Overview

1.1. Purpose

This library converts BMP files into YUV420 format and facilitates the display of continuous data.

1.2. Module Configuration

The configuration of the modules in the Simple Animation (SAN) library is shown below.

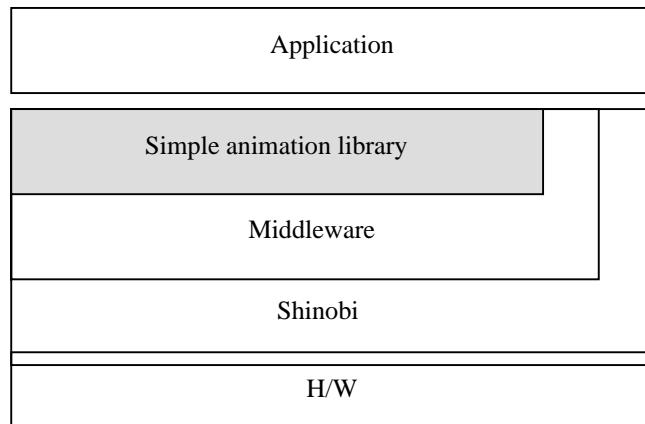


Fig. 1-1 Module Configuration

2. The SAN Concept

The SAN_LoadTex function transfers one picture from SAN data to a texture buffer. The SAN_Draw function transfers this data that was transferred to the texture buffer to the frame buffer. The data format of one picture in the SAN data is YUV420. When the data is transferred to a texture buffer, it is converted to YUV422 format.

The SAN concept is illustrated below.

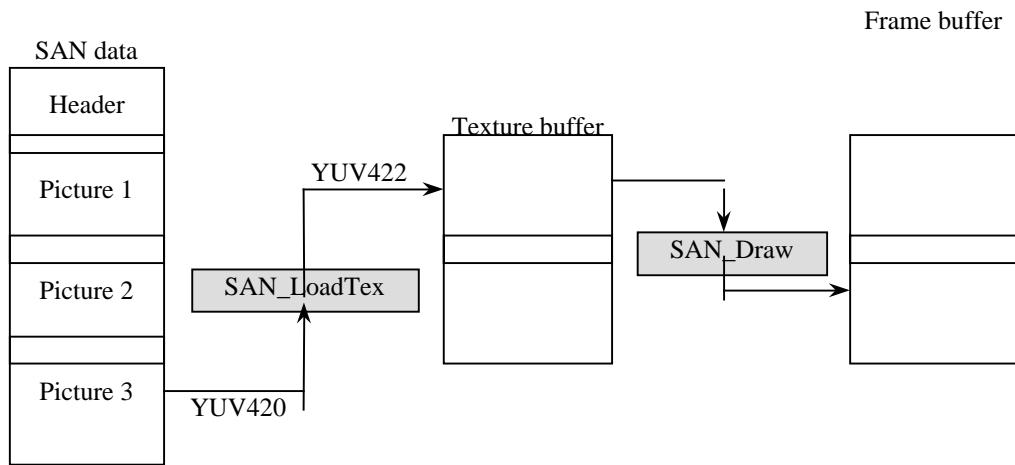


Fig. 2-1 The SAN Concept

3. Using the Library

A sample program is shown below.

<Sample program>

4. Multiwindows

SAN is capable of displaying video in multiple windows. The following parameters can be set independently for each of the four vertices of a window.

(1) Display position

The X, Y, and Z coordinates can each be set. The X coordinates and Y coordinates are the frame buffer coordinate values. The coordinates for the upper left corner are (0.0, 0.0), and the coordinates for the lower right corner are (639.0, 479.0). The Z coordinates are also specified, but no perspective conversion is performed on the Z coordinates.

(2) Vertex luminance

These parameters specify the alpha, red, green and blue intensity. Set these values in the range from 1.0 to 0.0. These values are multiplied with the video data. Changing these values from 0.0 to 1.0 produces a "fade in from black" effect, and changing these values from 1.0 to 0.0 produces a "fade out" effect. The alpha value sets the mixing ratio with the background video. Changing these values produces a dissolve effect.

(3) Vertex luminance offset

These parameters specify additive components for the alpha, red, green, and blue values. Set these values in the range from 1.0 to 0.0. The value produced by multiplying this setting by 255 is then added to the video data. Changing these values from 0.0 to 1.0 produces a "fade out to white" effect, and changing these values from 1.0 to 0.0 produces a "fade in from white" effect.

(4) Video position

These parameters specify the position of the video data corresponding to each vertex. Although equivalent to the UV coordinates, the values that are specified are the positions in terms of the pixels corresponding to the video data.

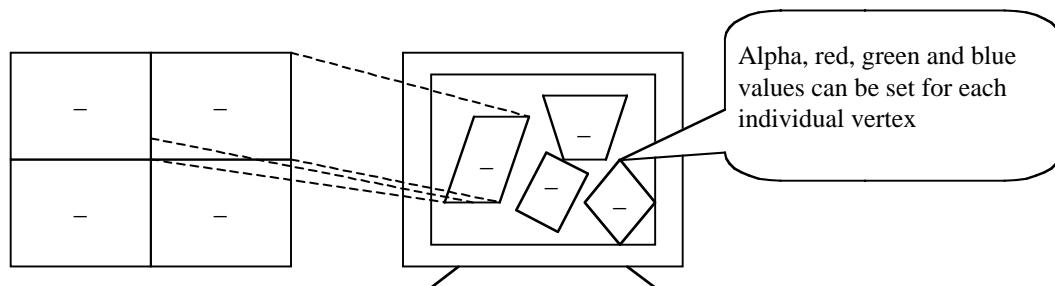


Fig. 4-1 Multi-window Display

In order to use multi-window display, buffers for the surface points need to be allocated and set up. A "surface point" is one of the vertices of a window. A window is defined by four surface points. For example, 100 surface points are needed in order to display 25 windows.

The procedure for allocating and setting up surface point buffers is shown below.

<Surface point buffer allocation and setup>

```
size = SAN_CalcSrfBufSize(san, 4*25);  
buf = syMalloc(size);  
SAN_SetSrfPntBuf(san, 4*25, buf, size);
```

5. SAN Textures

SAN allocates texture surfaces for video using the Kamui driver, and texture surfaces for masks. An application can get a surface from a SAN handle and then apply a video and mask to the object surface. The SAN texture concept is illustrated below.

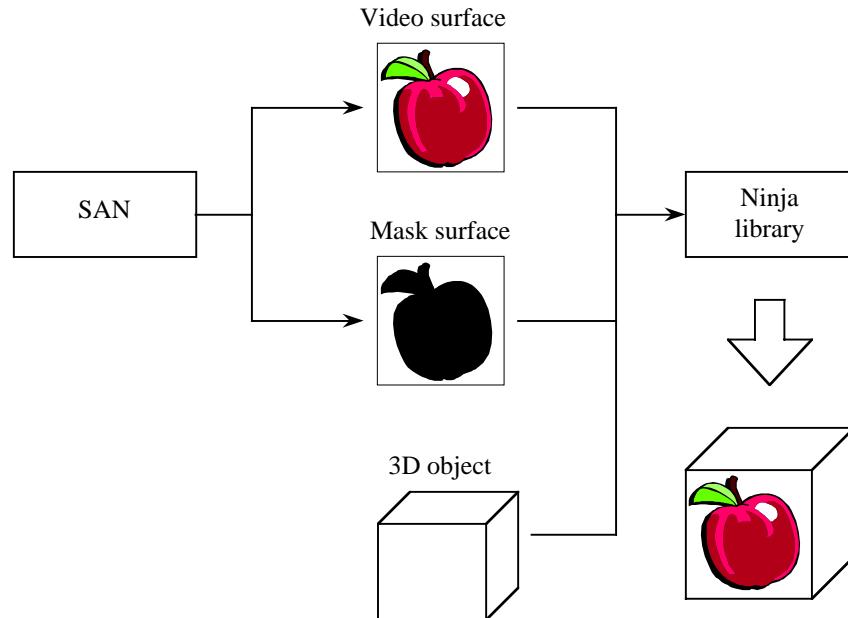


Fig. 5-1 SAN Texture Concept

With SAN, the following two texture surfaces can be gotten from a handle.

(1) Video surface

Rectangle-format YUV422 texture

(2) Mask surface

Twiddled-format 8-bit/pixel palette texture

<Surface acquisition method>

```
SAN_LoadTex(san, 0);                                     /* Transfer to  
texture RAM                                         */  
SAN_GetVideoPic(san, 0, &pic);                         /* Video surface acquisition */  
user_set_video_srf(pic.srf);                           /* Video surface application */  
SAN_GetMskPic(san, 0, &pic);                          /* Mask surface acquisition */  
user_set_msk_srf(pic.srf);                            /* Mask surface application */  
/* Polygon rendering */
```

* "pic.srf" is a surface assigned by the Kamui driver.

<SAN Textures with the Ninja Library>

The njSetMvSurface function can be used to register a SAN surface in a texture with a number specified in the texture list. The njSetMvSurface function is defined in the middleware sample program.

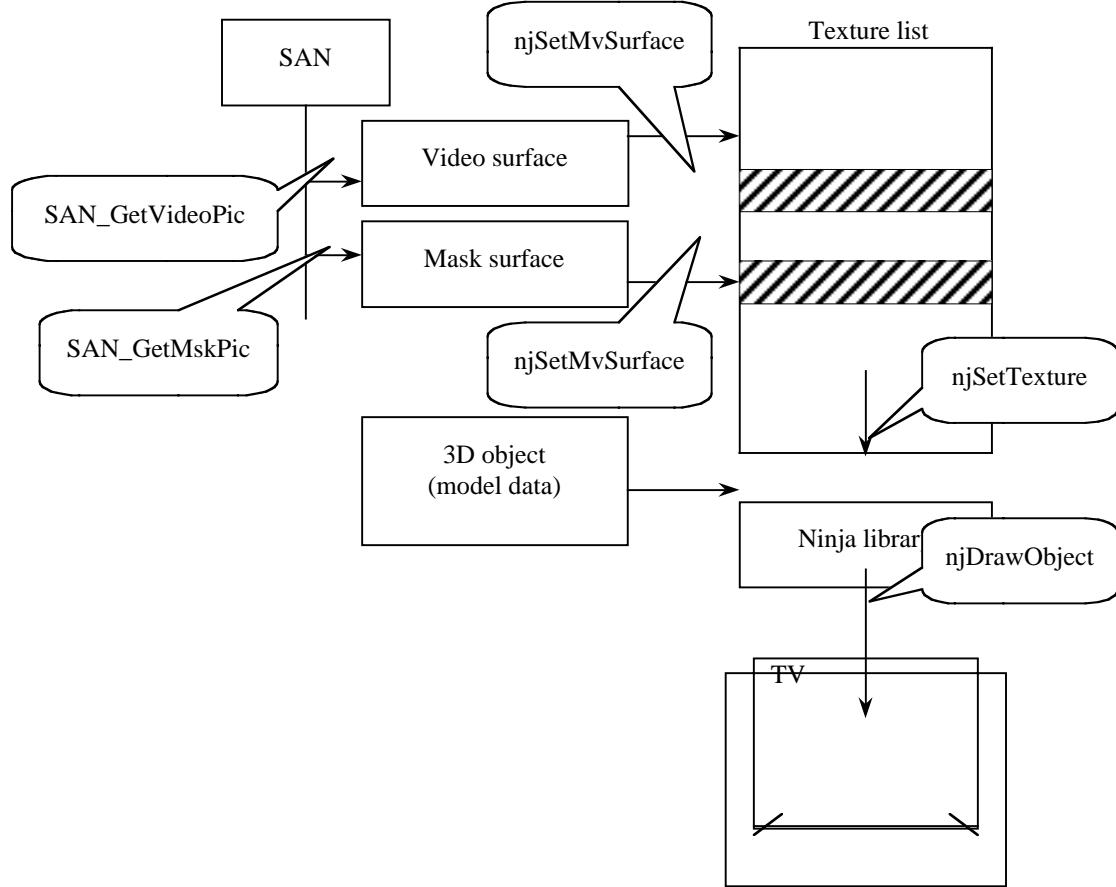


Fig. 5-2 SAN Textures with the Ninja Library

6. Composition

6.1. Composition Functions

SAN offers composition functions for implementing visual effects. The following composition methods are available:

(1) Opaque display

Renders video without composition.

(2) Translucent composition

Alpha blending in individual windows.

(3) Additive composition

Simply adds the RGB values of a polygon to the RGB values of the movie.

(4) Alpha composition

Alpha blending at the individual pixel level of polygon RGB values and movie RGB values.

<"alpha blending">

"Alpha blending" is expressed as follows by the mixing value "α":

$$\begin{aligned} \text{output} &= \alpha \cdot s + (1 - \alpha) \cdot d \\ \text{output} &: \text{RGB value after composition} \\ \alpha &: \text{Mixing ratio} \\ s &: \text{Movie RGB value} \\ d &: \text{Polygon RGB value} \end{aligned}$$

6.2. Alpha Composition

Alpha composition at the individual pixel level is implemented by rendering two translucent polygons as shown below. After rendering the mask surface (multiplication processing), the video surface is added. The Composition is shown below.

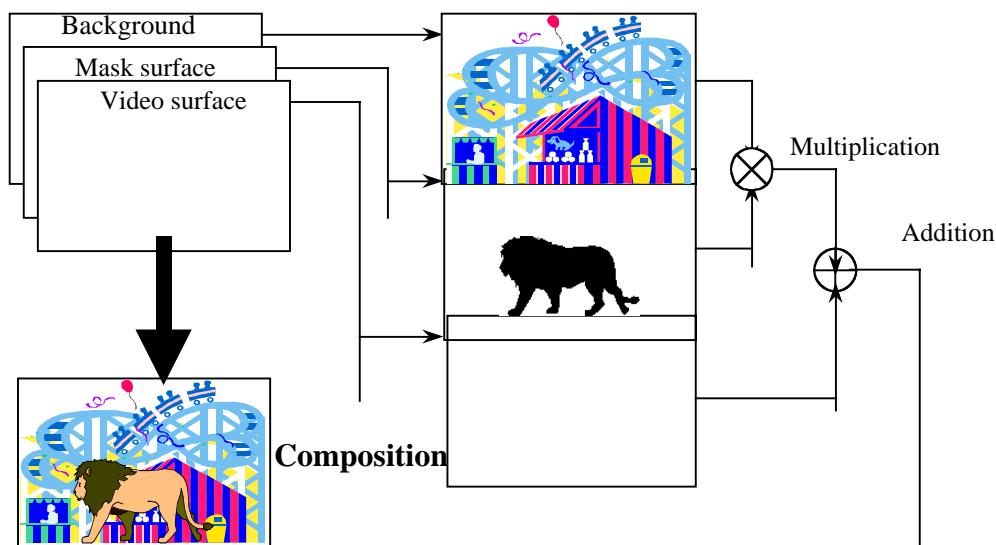


Fig. 6-1 Composition

7. Data Format

SAN data is linked to YUV420 data. The data is created from a BMP file (24 bits, noncompressed) by the bmp2san.exe program.

The SAN data format is shown below.

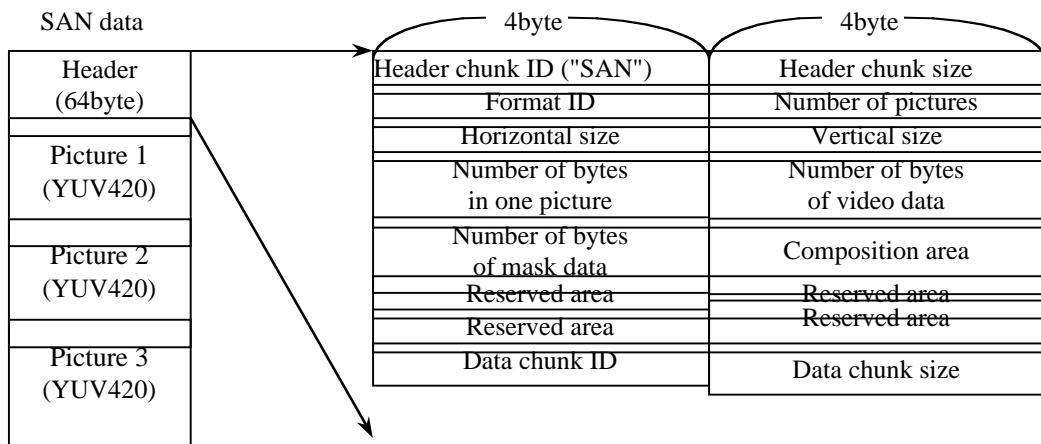


Fig. 7-1 SAN Data Format

8. Data Creation Method

8.1. Opaque Display, Translucent Composition, Additive Composition

The bmp2san.exe program converts multiple BMP files into YUV420 format and links the data together.

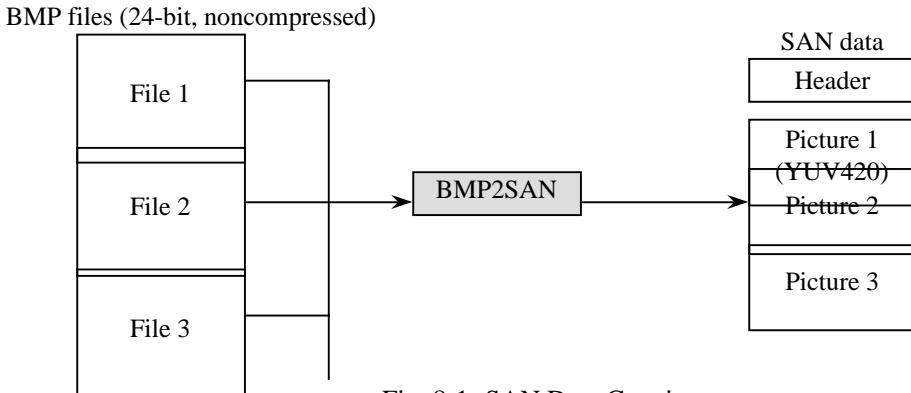


Fig. 8-1 SAN Data Creation

Execute the following command at the command prompt. The last parameter is the output file.

C:\TEMP>bmp2san file1.bmp file2.bmp file3.bmp sample.san



Extensions may be omitted.

C:\TEMP>bmp2san file1 file2 file3 sample

The format for using a subcommand file is as follows:

C:\TEMP>bmp2san -sub=san.sub

Subcommand file (san.sub)

```
file1.bmp  
file2.bmp  
file3.bmp  
sample.san
```

Follow the procedure described below to convert all of the BMP files in a directory to SAN data.

C:\TEMP>dir /b *.bmp > tmp.sub

Subcommand file (tmp.sub)

C:\TEMP>bmp2san -sub=tmp.sub sample1.san

```
file1.bmp  
file2.bmp  
file3.bmp
```

<Translucent composition>

Add the options shown below.

C:\TEMP>bmp2san -compo=trnsp -sub=tmp.sub sample1.san

<Additive composition>

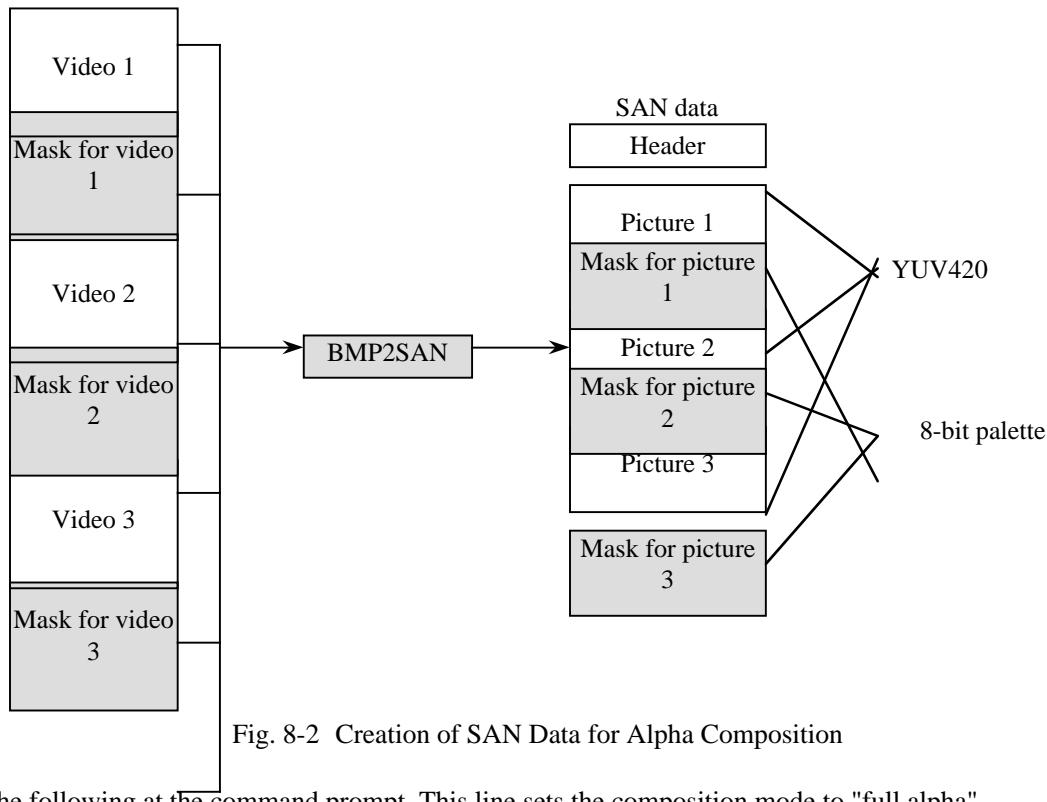
Add the options shown below.

C:\TEMP>bmp2san -compo=add -sub=tmp.sub sample1.san

8.2. Alpha Composition

SAN data for alpha composition is created from video data and mask data by a graphics tool, and is then linked together by the bmp2san.exe program. The mask data is converted into twiddled-format 8-bit palette data.

BMP file (24 bits, noncompressed)



Execute the following at the command prompt. This line sets the composition mode to "full alpha" ("alph256"), and then inputs the BMP files in the order video data, mask data.

```
C:\TEMP>bmp2san -compo=alph256 file1v.bmp file1a.bmp file2v.bmp  
file2a.bmp sample.san
```

```
file1v.bmp, file2v.bmp .. Video data  
file1a.bmp, file2a.bmp .. Mask data
```

As with opaque display, a subcommand file can be specified.

9. Data Specifications

The library data list is shown below.

Table 9-1. Data List

Data name	Function	Number
Data type		
SAN	SAN handle	1.1
SAN_PIC	Surface information structure	1.2

9.1. Data type

Title	Data Name	Data	No
Data	SAN	SAN handle	1.1

This handle is used to control SAN. This handle is generated by the SAN_Create function.

Title	Data Name	Data	No
Data	SAN_PIC	Surface information structure	1.2

This structure is used to pass surface information to the graphics library when using SAN as a texture.

Member	Type	Description
srf	void *	Texture surface
width	long	Valid surface width (unit: pixels)
height	long	Valid surface height (unit: pixels)

10. Function Specifications

The library functions are listed shown below.

Table 10-1 List of Functions

Function name	Function	Number
Initialization and end processing		
SAN_Init	Initialization	1.1
SAN_Finish	End processing	1.2
SAN_InitKm	Initialization (for Kamui)	1.3
SAN_FinishKm	End processing (for Kamui)	1.4
Basic operation processing		
SAN_CalcWorkSize	Work area size calculation	2.1
SAN_CalcWorkSizeAlph	Work area size calculation (for composition)	2.2
SAN_Create	Handle generation	2.3
SAN_Destroy	Handle deletion	2.4
SAN_LoadTex	Transfer of YUV420 data to V-RAM	2.5
SAN_Draw	Texture display	2.6
SAN_GetNumPic	Acquisition of number of pictures	2.7
Display control		
SAN_SetDispPos	Display position setting	3.1
SAN_SetDispSize	Display size setting	3.2
SAN_SetDispZ	Display screen depth value setting	3.3
SAN_GetDispZ	Display screen depth value acquisition	3.4
SAN_SetDispBright	Luminance setting	3.5
SAN_GetDispBright	Luminance acquisition	3.6
SAN_SetDispBrightOfst	Luminance offset setting	3.7
SAN_GetDispBrightOfst	Luminance offset acquisition	3.8
Surface control		
SAN_CalcSrfBufSize	Surface point buffer size calculation	4.1
SAN_SetSrfPntBuf	Surface point buffer setting	4.2
SAN_SetSrfPos	Display position setting	4.3
SAN_GetSrfPos	Display position acquisition	4.4
SAN_SetSrfBright	Luminance setting	4.5
SAN_GetSrfBright	Luminance acquisition	4.6
SAN_SetSrfBrightOfst	Luminance offset setting	4.7
SAN_GetSrfBrightOfst	Luminance offset acquisition	4.8
SAN_SetImgPos	Image position setting	4.9
SAN_GetImgPos	Image position acquisition	4.10
SAN_GetImgSize	Image size acquisition	4.11
Texture		
SAN_GetVideoPic	Video picture acquisition	5.1
SAN_GetMskPic	Mask picture acquisition	5.2
Error processing		
SAN_EntryErrFunc	Error callback function registration	6.1

10.1. Initialization and end processing

Title Function	Function Name SAN_Init	Function Initialization	No 1.1
--------------------------	----------------------------------	-----------------------------------	------------------

[Syntax] void SAN_Init(void);
[Input] None
[Output] None
[Rtn Val] None
[Purpose] This function initializes the library.

Title Function	Function Name SAN_Finish	Function End processing	No 1.2
--------------------------	------------------------------------	-----------------------------------	------------------

[Syntax] void SAN_Finish(void);
[Input] None
[Output] None
[Rtn Val] None
[Purpose] This function performs the library end processing.

Title	Function Name	Function	No
Function	SAN_InitKm	Initialization (for Kamui)	1.3

[Syntax] void SAN_InitKm(void);
 [Input] None
 [Output] None
 [Rtn Val] None
 [Purpose] This function initializes the library.

Title	Function Name	Function	No
Function	SAN_FinishKm Function	End processing (for Kamui)	1.4

[Syntax] void SAN_FinishKm(void);
 [Input] None
 [Output] None
 [Rtn Val] None
 [Purpose] This function performs the library end processing.

10.2. Basic Operation Processing

Title Function	Function Name SAN_CalcWorkSize	Function Work area size calculation	No 2.1
-------------------	-----------------------------------	--	-----------

[Syntax] long SAN_CalcWorkSize (long ntex);
 [Input] ntex : Number of texture buffers
 [Output] None
 [Rtn Val] Work area size (unit : bytes)
 [Purpose] This function calculates the size of the work area.

Title Function	Function Name SAN_CalcWorkSizeAlph	Function Work area size calculation (for composition)	No 2.2
-------------------	---------------------------------------	---	-----------

[Syntax] long SAN_CalcWorkSizeAlph(long ntex, long alph_flg);
 [Input] ntex : Number of texture buffers
 alph_flg : Composition flag
 (1: Perform composition; 0: Do not perform composition)
 [Output] None
 [Rtn Val] Work area size (unit : bytes)
 [Purpose] This function calculates the size of the work area.

Title	Function Name	Function	No
Function	SAN_Create	Handle generation	2.3

[Syntax] SAN SAN_Create(void *sandat, long ntex, void *wk);
 [Input] sandat : SAN data
 ntex : Number of texture buffers
 wk : Work area
 [Output] None
 [Rtn Val] SAN handle
 [Purpose] This function generates a handle.

Title	Function Name	Function	No
Function	SAN_Destroy	Handle deletion	2.4

[Syntax] void SAN_Destroy(SAN san);
 [Input] san : SAN handle
 [Output] None
 [Rtn Val] None
 [Purpose] This function deletes a handle.

Title	Function Name	Function	No
Function	SAN_LoadTex	Transfer of YUV420 data to V-RAM	2.5

[Syntax] void SAN_LoadTex(SAN san, long sno, long dno);
 [Input] san : SAN handle
 sno : Picture number
 dno : Texture buffer number
 [Output] None
 [Rtn Val] None
 [Purpose] This function transfers YUV420 data to a texture buffer (V-RAM).
 [Remarks] The picture number must be within a range from 0 to [the number of pictures in the SAN data - 1]. The number of pictures in the SAN data can be gotten with the SAN_GetNumPic function. The texture buffer number must be within a range from 0 to [the number of texture pictures - 1]. The number of texture buffers is indicated by the "ntex" parameter in the SAN_CalcWorkSize function and the SAN_Create function.

Title	Function Name	Function	No
Function	SAN_Draw	Texture display	2.6

[Syntax] void SAN_Draw(SAN san, long dno);
 [Input] san : SAN handle
 dno : Texture buffer number
 [Output] None
 [Rtn Val] None
 [Purpose] This function displays a texture.
 [Remarks] The texture buffer number must be within a range from 0 to [the number of texture pictures - 1]. The number of texture buffers is indicated by the "ntex" parameter in the SAN_CalcWorkSize function and the SAN_Create function.

Title	Function Name	Function	No
Function	SAN_GetNumPic	Acquisition of number of pictures	2.7

[Syntax] long SAN_GetNumPic(SAN san);

[Input] san : SAN HANDLE

[Output] None

[Rtn Val] Number of pictures

[Purpose] This function gets the number of pictures (the number of linked files) in the SAN data.

10.3. Display control

Title Function	Function Name SAN_SetDispPos	Function Display position setting	No 3.1
-------------------	---------------------------------	--------------------------------------	-----------

[Syntax] void SAN_SetDispPos(SAN san, float lx, float ly);

[Input] san : SAN HANDLE

lx : X coordinate

ly : Y coordinate

[Output] None

[Rtn Val] None

[Purpose] This function sets the display position.

Title Function	Function Name SAN_SetDispSize	Function Display size setting	No 3.2
-------------------	----------------------------------	----------------------------------	-----------

[Syntax] void SAN_SetDispSize(SAN san, float sx, float sy);

[Input] san : SAN HANDLE

sx : X coordinate

sy : Y coordinate

[Output] None

[Rtn Val] None

[Purpose] This function sets the display size.

Title Function	Function Name	Function	No
	SAN_SetDispZ	Display screen depth value setting	3.3

[Syntax] void SAN_SetDispZ(SAN san, float z);
 [Input] san : SAN HANDLE
 z : Depth value 1.0 (closest to front) 65536 (farthest in back)
 [Output] None
 [Rtn Val] None
 [Purpose] This function sets the display screen depth value.

Title Function	Function Name	Function	No
	SAN_GetDispZ	Display screen depth value acquisition	3.4

[Syntax] float SAN_GetDispZ(SAN san);
 [Input] san : SAN handle
 [Output] None
 [Rtn Val] Depth value 1.0 (closest to front) 65536 (farthest in back)
 [Purpose] This function gets the display screen depth value

Title	Function Name	Function	No
Function	SAN_SetDispBright	Luminance setting	3.5

[Syntax] void SAN_SetDispBright(SAN san, long val);
 [Input] san : SAN handle
 val : Luminance (0 to 255)
 [Output] None
 [Rtn Val] None
 [Purpose] This function sets the luminance.
 [Remarks] The default value for this setting is "224." This is the same as the Sofdec default value. Because the Dreamcast video output has a luminance of 110IRE when this value is 255, the default is set to 224. Depending on the material, it may be necessary to adjust this value.

Title	Function Name	Function	No
Function	SAN_GetDispBright	Luminance acquisition	3.6

[Syntax] long SAN_GetDispBright(SAN san);
 [Input] san : SAN handle
 [Output] None
 [Rtn Val] Luminance (0 to 255)
 [Purpose] This function gets the luminance.

Title	Function Name	Function	No
Function	SAN_SetDispBrightOfst	Luminance offset setting	3.7

[Syntax] void SAN_SetDispBrightOfst(SAN san, long val);
 [Input] san : SAN handle
 val : Luminance offset (0 to 255)
 [Output] None
 [Rtn Val] None
 [Purpose] This function sets the luminance offset.
 Remarks The default value for this setting is "6." This is the same as the Sofdec default value. The default value is set to "6" because black tends to get overwhelmed in computer graphics. Depending on the material, it may be necessary to adjust this value. You can implement a fade-in effect by setting this value to 255 and then gradually decreasing the value. Similarly, you can achieve a fade-out effect by gradually increasing this value up to 255.

Title	Function Name	Function	No
Function	SAN_GetDispBrightOfst	Luminance offset acquisition	3.8

[Syntax] long SAN_GetDispBrightOfst(SAN san);
 [Input] san : SAN handle
 [Output] None
 [Rtn Val] Luminance offset (0 to 255)
 [Purpose] This function gets the luminance offset.

10.4. Surface control

Title Function	Function Name SAN_CalcSrfBufSize	Function Surface point buffer size calculation	No 4.1
-------------------	-------------------------------------	---	-----------

[Syntax] long SAN_CalcSrfBufSize(SAN san, long npnt);
 [Input] san : SAN handle
 npnt : Number of surface points
 [Output] None
 [Rtn Val] Surface point buffer size (unit: bytes)
 [Purpose] This function gets the surface point buffer size.

Title Function	Function Name SAN_SetSrfPntBuf	Function Surface point buffer setting	No 4.2
-------------------	-----------------------------------	--	-----------

[Syntax] void SAN_SetSrfPntBuf(SAN san, long npnt, void *buf, long bsize);
 [Input] san : SAN handle
 npnt : Number of surface points
 buf : Buffer
 bsize : Buffer size (unit: bytes)
 [Output] None
 [Rtn Val] None
 [Purpose] This function sets the specified buffer as a surface point buffer.

Title Function	Function Name	Function	No
	SAN_SetSrfPos	Display position setting	4.3

[Syntax] void SAN_SetSrfPos(SAN san, unsigned long no, float lx, float ly, float lz);
 [Input] san : SAN handle
 no : Vertex number (numbered from 0 to 3, in a pattern that resembles a “z”)
 lx : X coordinate
 ly : Y coordinate
 lz : Z coordinate (1.0 (closest to front) 65536 (farthest in back))
 [Output] None
 [Rtn Val] None
 [Purpose] This function sets the position where the surface is to be displayed for each individual vertex.

Title Function	Function Name	Function	No
	SAN_GetSrfPos	Display position acquisition	4.4

[Syntax] void SAN_GetSrfPos(SAN san, unsigned long no, float *lx, float *ly, float *lz);
 [Input] san : SAN handle
 no : Vertex number (numbered from 0 to 3, in a pattern that resembles a “z”)
 [Output] lx : X coordinate
 ly : Y coordinate
 lz : Z coordinate (1.0 (closest to front) 65536 (farthest in back))
 [Rtn Val] None
 [Purpose] This function gets the position of the surface for each individual vertex.

Title Function	Function Name SAN_SetSrfBright	Function Luminance setting	No 4.5
-------------------	-----------------------------------	-------------------------------	-----------

[Syntax] void SAN_SetSrfBright(SAN san, unsigned long no,
 float a, float r, float g, float b);

[Input] san : SAN handle
 no : Vertex number (numbered from 0 to 3, in a pattern that resembles a “z”)
 a : Alpha value (0.0 to 1.0)
 r : Red component (0.0 to 1.0)
 g : Green component (0.0 to 1.0)
 b : Blue component (0.0 to 1.0)

[Output] None

[Rtn Val] None

[Purpose] This function sets the luminance for each individual vertex.

Title Function	Function Name SAN_GetSrfBright	Function Luminance acquisition	No 4.6
-------------------	-----------------------------------	-----------------------------------	-----------

[Syntax] void SAN_GetSrfBright(SAN san, unsigned long no,
 float *a, float *r, float *g, float *b);

[Input] san : SAN handle
 no : Vertex number (numbered from 0 to 3, in a pattern that resembles a “z”)

[Output] a : Alpha value (0.0 to 1.0)
 r : Red component (0.0 to 1.0)
 g : Green component (0.0 to 1.0)
 b : Blue component (0.0 to 1.0)

[Rtn Val] None

[Purpose] This function gets the luminance for each individual vertex.

Title Function	Function Name	Function	No
	SAN_SetSrfBrightOfst	Luminance offset setting	4.7

[Syntax] void SAN_SetSrfBrightOfst(SAN san, unsigned long no,
 float a, float r, float g, float b);

[Input] san : SAN handle
 no : Vertex number (numbered from 0 to 3, in a pattern that resembles a “z”)
 a : Alpha value (0.0 to 1.0)
 r : Red component (0.0 to 1.0)
 g : Green component (0.0 to 1.0)
 b : Blue component (0.0 to 1.0)

[Output] None

[Rtn Val] None

[Purpose] This function sets the luminance offset for each individual vertex.

Title Function	Function Name	Function	No
	SAN_GetSrfBrightOfst	Luminance offset acquisition	4.8

[Syntax] void SAN_GetSrfBrightOfst(SAN san, unsigned long no,
 float *a, float *r, float *g, float *b);

[Input] san : SAN handle
 no : Vertex number (numbered from 0 to 3, in a pattern that resembles a “z”)

[Output] a : Alpha value (0.0 to 1.0)
 r : Red component (0.0 to 1.0)
 g : Green component (0.0 to 1.0)
 b : Blue component (0.0 to 1.0)

[Rtn Val] None

[Purpose] This function gets the luminance offset for each individual vertex.

Title	Function Name	Function	No
Function	SAN_SetImgPos	Image position setting	4.9

[Syntax] void SAN_SetImgPos(SAN san, unsigned long no, float lx, float ly);
 [Input] san : SAN handle
 no : Vertex number (numbered from 0 to 3, in a pattern that resembles a “z”)
 lx : X coordinate
 ly : Y coordinate
 [Output] None
 [Rtn Val] None
 [Purpose] This function sets the position of the image for each individual vertex.

Title	Function Name	Function	No
Function	SAN_GetImgPos	Image position acquisition	4.10

[Syntax] void SAN_GetImgPos(SAN san, unsigned long no, float *lx, float *ly);
 [Input] san : SAN handle
 no : Vertex number (numbered from 0 to 3, in a pattern that resembles a “z”)
 [Output] lx : X coordinate
 ly : Y coordinate
 [Rtn Val] None
 [Purpose] This function gets the position of the image for each individual vertex.

Title Function	Function Name SAN_GetImgSize	Function Image size acquisition	No 4.11
-------------------	---------------------------------	------------------------------------	------------

[Syntax] void SAN_GetImgSize(SAN san, long *isx, long *isy);

[Input] san : SAN handle

[Output] isx : X coordinate

isy : Y coordinate

[Rtn Val] None

[Purpose] This function gets the image size.

10.5. Texture

Title Function	Function Name SAN_GetVideoPic	Function Video picture acquisition	No 5.1
-------------------	----------------------------------	---------------------------------------	-----------

[Syntax] void SAN_GetVideoPic(SAN san, long dno, SAN_PICTURE *pic);
 [Input] san : SAN handle
 dno : Texture buffer number
 [Output] pic : Surface information structure
 [Rtn Val] None
 [Purpose] This function gets a video surface from the texture buffer with the specified number.
 [Remarks] The texture buffer number can be specified in a range from 0 to [the number of textures - 1]. The number of texture buffers is indicated by the "ntex" parameter in the SAN_CalcWorkSize function and the SAN_Create function.

Title Function	Function Name SAN_GetMskPic	Function Mask picture acquisition	No 5.2
-------------------	--------------------------------	--------------------------------------	-----------

[Syntax] void SAN_GetMskPic(SAN san, long dno, SAN_PICTURE *pic);
 [Input] san : SAN handle
 dno : Texture buffer number
 [Output] pic : Surface information structure
 [Rtn Val] None
 [Purpose] This function gets a mask surface from the texture buffer with the specified number.
 [Remarks] The texture buffer number can be specified in a range from 0 to [the number of textures - 1]. The number of texture buffers is indicated by the "ntex" parameter in the SAN_CalcWorkSize function and the SAN_Create function.

10.6. Error processing

Title	Function Name	Function	No
Function	SAN_EntryErrFunc	Error callback function registration	6.1

- [Syntax] void SAN_EntryErrFunc(SAN_ERRFN errfn, void *obj);
[Input] errfn : User callback function
obj : First parameter of the callback function
[Output] None
[Rtn Val] None
[Purpose] This function registers the error callback function. If an error occurs, the registered callback function is called in the format indicated below.

```
(* func) (void *obj, char *msg);
```

The first parameter of this function, "obj", becomes the second parameter of the SAN_EntryErrFunc function. The second parameter, "msg", is the error message. Because this function is used for debugging purposes, replace this function with a NOP function when creating the master for the application.

Sega Dreamcast™

***Dreamcast
CRI MPEG Sofdec
Data Creation Manual***

1. Introduction

1.1. Overview

This document describes cautions concerning the creation of material for MPEG Sofdec movie data ("Sofdec data" hereafter), methods for creating Sofdec data, and details concerning the tool needed in order to create Sofdec data.

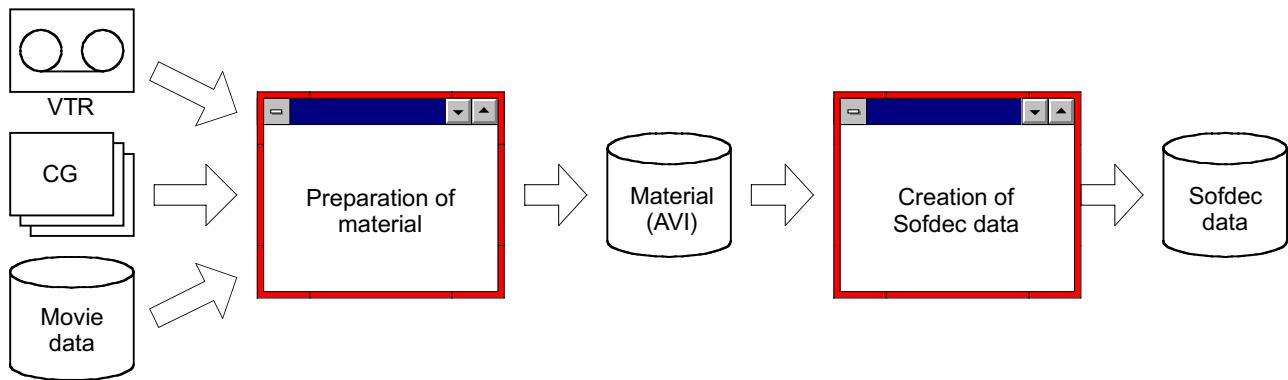


Fig. 1.1 Flow of Sofdec Data Creation

1.2. Operating Environment

Each Sofdec data creation tool operates in the environment described below.

Table 1.1 Operating Environment

Item	Description
Hardware	PC machine with a Pentium Series (or compatible) CPU (A CPU that supports MMX is recommended.)
OS	MS Windows 95 or later, MS Windows NT 4.0 or later
Memory	At least 32MB

1.3. Description of Terms

Some of the terms used in this document are described below.

Table 1.2 List of Terms

Term	Definition
Sofdec data	A movie file that contains video and audio data and which can be played back by CRI MPEG Sofdec. MPEG1 Video is used as the video codec, and Sofdec Audio is used as the audio codec.
Material	A file that is the source for Sofdec data. Includes video and audio. Non-compressed AVI format is recommended as a standard.
Video stream	Encoded data representing the video in a material file.
Audio stream	Encoded data representing the audio in a material file.
Frame	Refers to one frame of video for display.
Frame rate	Refers to the video frequency of the material, or the update frequency on the output screen. The unit is "fps" (frames per second). In the NTSC system, the frame rate is 29.97fps. In the PAL system, the frame rate is 25fps. This is known as the "picture rate" in the MPEG convention.
Pixel	One picture element (or "dot") that is displayed on the screen.
Pixel aspect ratio	The ratio of the height versus the width of a pixel. Normally expressed as a value that is calculated by dividing the height by the width. In the NTSC system, this value is $1/0.91 \approx 1.0950$ (approximately). Also called the "pel aspect ratio."
Cropping	Refers to the process of cutting off unneeded portions of a picture.
Vsync	The vertical sync signal. One of the video signals in a TV or monitor, this signal indicates the timing at which the scan line returns from the bottom of the screen to the top of the screen.

2. Preparation of Material

Preparation of the material involves creating the material from the source Sofdec data and then converting it into an AVI file.

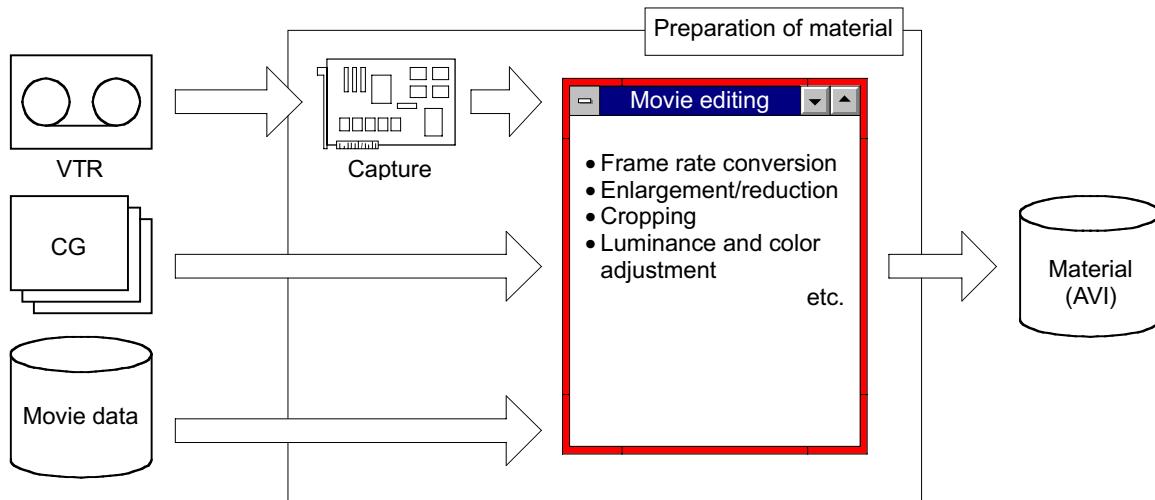


Fig. 2.1 Preparation of Material

2.1. Material File Format

The Sofdec data creation tool uses the AVI format for their standard input.

Table 2.1 Video Files That Can Be Encoded

Item	Description
Format	AVI format (*1)
Picture size	Multiple of 32 (H) × 16 (V) pixels
Number of colors	24-bit color (approximately 16 million colors)
Compression	Non-compressed recommended
Frame rate (fps)	10, 15, 23.976, 24, 25, 29.97, and 30 (10fps and 15fps are supported only for the Sega Dreamcast Movie Creator)

Table 2.2 Audio Files That Can Be Encoded

Item	Description
Format	AVI format (interleaved with video), WAV format, AIFF format
Sampling frequency (Hz)	44100, 22050, 14700, 11025, 8820, 7350, 6300
Compression	Non-compressed recommended
Number of quantization bits	8 bits or 16 bits
Audio channel	Monoaural or stereo

(*1) Movie formats

The Sega Dreamcast Movie Creator plug-in can use all movie formats supported by Adobe Premiere. In addition to AVI, Premiere supports QuickTime, bitmap sequences, TARGA sequences, and others. The plug-in can create Sofdec data from any of these formats.

2.2. Frame Rates

Keep the following points in mind when setting and converting frame rates.

(1) When creating new material

Dreamcast supports the NTSC and PAL video signal systems. Adjust the frame rate of the material in accordance with the video signal system that will be used.

Table 2.3 Video Signal System

Video signal system	Frame rate	Pixel aspect ratio	Region where used	Remarks
NTSC system	29.97fps	1.0950	Japan, U.S.	Interlaced system
PAL system	25fps	0.9157	Europe	Interlaced system
VGA	59.94fps	1.0000	(for PC monitors)	Non-interlaced system Create movies at 29.97fps

(2) When using existing material

The Sofdec data creation tool supports the frame rates shown in the table below.

If material with one of these frame rates already exists, do not convert the frame rate.

Table 2.4 Frame Rates Supported by the Sofdec Data Creation Tool

Item	Supported values
Frame rate (fps)	10, 15, 23.976, 24, 25, 29.97, 30

(3) When converting existing material

Conversion is necessary if the existing material falls into one of the following categories:

- The frame rate of the material is not a frame rate that is supported by the Sofdec data creation tool

→Use movie editing software to convert the frame rate.

- 24fps film material (such as animation) was recorded by a VCR

→If the material has been captured as VCR material, the frame rate becomes 29.97fps. When capturing film material, perform reverse television-cinema conversion to convert the frame rate to 24fps.

2.3. Pixel Aspect Ratio

The pixel aspect ratio of a television depends on the video signal system. When creating material, adjust the pixel aspect ratio in accordance with the video signal system to be used.

If the pixel aspect ratio of the display differs from that of the material, the video will not be displayed properly. For example, if the screen consists of 100×100 square pixels, an NTSC system video will appear stretched out vertically, a PAL system video will appear stretched out horizontally, and a VGA signal will be displayed normally. Make sure that the pixel ratios of the material and the display match.

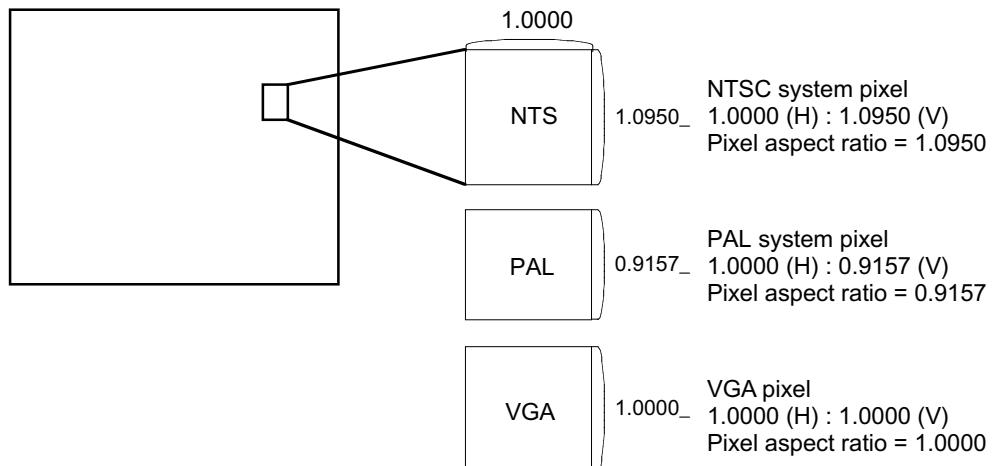


Fig. 2.2 Pixel Aspect Ratios

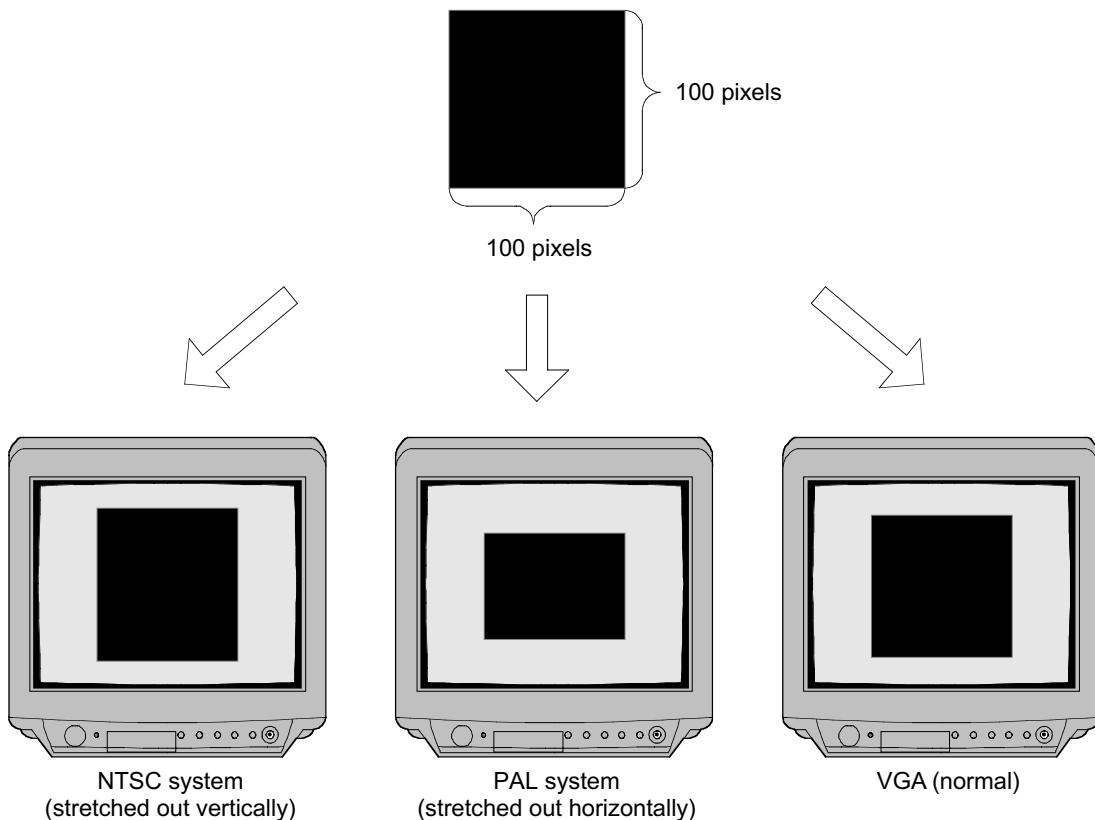


Fig. 2.3 Display differences with different video systems

2.4. Luminance and Color

The video that is output by the Dreamcast will appear to have different colors on a PC monitor. When creating material, always use the Dreamcast's TV output video to check the colors and luminance. Pay special attention to the following two points in particular:

③ Dreamcast luminance signal output characteristics

The luminance signal that is output by the Dreamcast is amplified by about 10%. During playback, the brightness value can be used to adjust the luminance. Because doing so will change the color tones, it is necessary beforehand to use movie editing software to either reduce the luminance by 10% or else emphasize the color.

✿ National characteristics of video signal systems

There are differences among television video signals between countries. For example, even though Japan and the U.S. both use the same NTSC standard, the signal offset values are different, so the U.S. system appears slightly darker. When checking the color and luminance, use a TV from the actual target country.

2.5. Picture Size

Adjust the picture size of the material in order to reduce the load on the CPU during decoding and to improve picture quality. There are two types of adjustment: cropping and reduction.

③ Cropping

When the material has been captured from a VCR, etc., areas that are not actually visible on the screen are included. These unnecessary portions can be cropped in order to eliminate wasted processing. For example, with the Dreamcast's NTSC system output, the picture size that is actually visible to the viewer is approximately 640×448 . Note that the picture size that the user can actually see differs according to the video signal system.

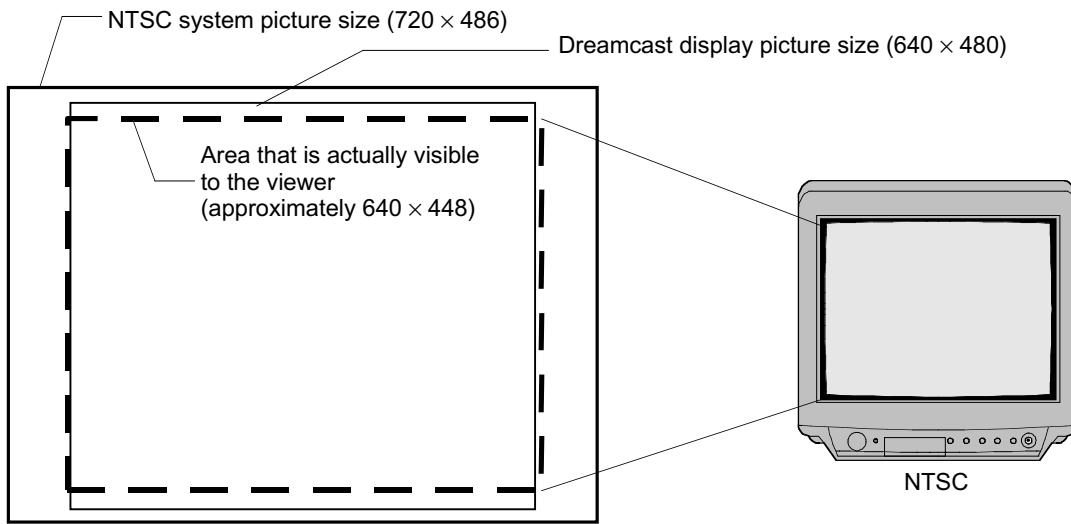


Fig. 2.4 NTSC System TV Display Size

✿ Reduction

Reduce the picture size in order to reduce the processing load during decoding. Select an appropriate size on the basis of factors such as the state of CPU usage during playback, the material, and the application.

2.6. Interlacing

The NTSC system and the PAL system are interlaced systems in which each frame of video is divided into two groups of alternating lines that are displayed in alternation. These two groups are called "field 1" and "field 2."

When handling interlaced video, it is important to make sure that field 1 and field 2 are not reversed. Also keep the following points in mind:

(1) Capture

When capturing interlaced video, give the field 2 priority and capture the fields so that field 1 includes the topmost line. Pay careful attention to the field in which capture starts, and to the specification of the coordinates for the capture range.

(2) Cropping

When cropping, do so in such a way that the topmost line is part of field 1. If the cropped picture starts with field 2, the fields will be reversed during playback, and the image may not be displayed correctly.

(3) Reduction

In the interlaced system, the field 1 and field 2 video are displayed at slightly different times. If an interlaced video signal is reduced as is, the video in the two fields will be mixed, and the image will be ghosted. When reducing an interlaced video signal, extract one of the fields first, and then reduce that field.

(4) Playback

Depending on the playback position, field 1 and field 2 might be reversed. In this case, slide the fields one pixel in the vertical direction.

2.7. Material for Both the NTSC and PAL Systems

When creating Sofdec data that will be played back on both the NTSC system and the PAL system, create the Sofdec data for the NTSC system, which has the higher frame rate, and then pay attention to the following points in regards to playing the video back on a PAL system.

(1) Sync processing

The CRI MPEG Sofdec performs sync processing which keeps the video and audio in sync if the frame rate of the material differs from the output frame rate. When the display mode is set to noninterlaced, sync is performed in Vsync units (field units). (Playback is much smoother this way then when compared to movie editing software that performs compensation in frame units.) When playing back the video in the PAL system, set the screen display mode to PAL non-interlaced mode.

- Playback by CRI MPEG Sofdec

When NTSC system material is played back in the PAL system, the CRI MPEG Sofdec decoder applies compensation in Vsync units.

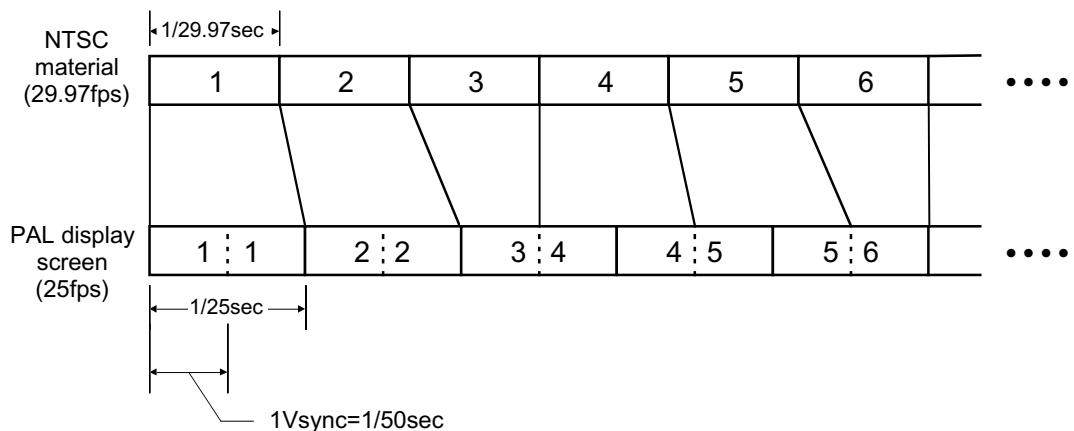


Fig. 2.5 When NTSC System Material Is Played Back in the PAL System

- When converted by movie editing software (reference)

If NTSC system material is converted to the PAL system by movie editing software, compensation is applied in frame units. In the figure below, frame 6 is lost as a result of the compensation.

Do not use movie editing software to convert frame rates.

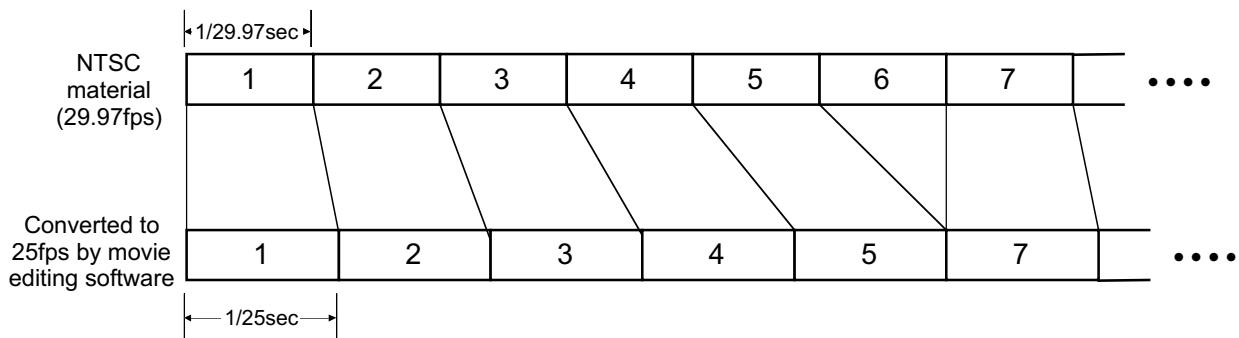


Fig. 2.6 When Movie Editing Software Is Used for Conversion

(2) Pixel aspect ratio

The pixel aspect ratio in the NTSC system is different from that of the PAL system. If no adjustment is made, an NTSC picture will appear to be reduced in the vertical direction when played back in the PAL system. As a remedy, expand the picture in the vertical direction (or reduce it in the horizontal direction) when displaying the picture. The expansion ratio is calculated from the pixel aspect ratio. In the following example, we will give example adjustments, assuming a picture display size of 640×480 .

- Expanding in the vertical direction

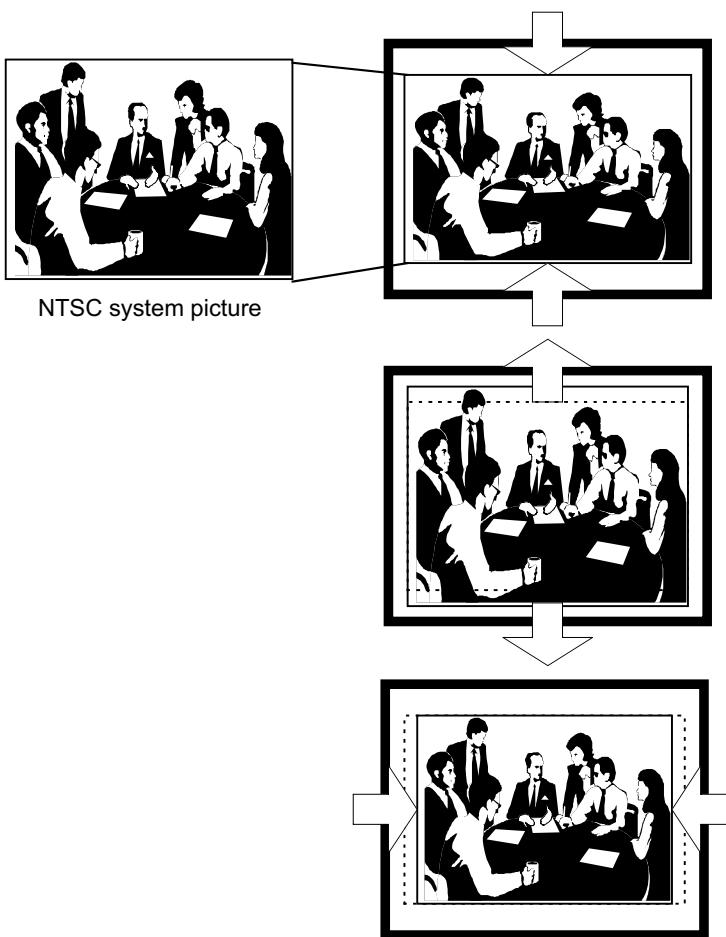
The expansion factor in the vertical direction is $1.0950/0.9157 = 1.1958$.

In the case of a 640×480 picture, $480 \times 1.1958 \approx 574$ (approximately), so the playback area values should be set to 640×574 . (However, it is important to note that the Dreamcast can only display up to 640×480 , so the top and bottom 47 pixels will not be displayed.)

- Reducing in the horizontal direction

The reduction factor in the horizontal direction is $(1/1.0950)/(1/0.9157) = 0.8363$.

In the case of a 640×480 picture, $640 \times 0.8363 \approx 536$ (approximately), so the playback area values should be set to 536×480 .



- When displayed as is

When an NTSC picture is displayed in the PAL system, the different pixel aspect ratios result in the picture being reduced in the vertical direction.

- When enlarged in the vertical direction

Here, the picture is enlarged in the vertical direction in accordance with the pixel aspect ratio, resulting in a normal aspect ratio for the displayed picture. However, it is important to note that the portion of the picture that extends beyond the display area of the Dreamcast system (640×480) is not displayed.

- When reduced in the horizontal direction

Here, the picture is reduced in the horizontal direction in accordance with the pixel aspect ratio.

Fig. 2.7 Notes on Displaying an NTSC Picture in the PAL System

3. Sofdec Data Creation Methods

3.1. Recommended Values for the Encoding Parameters

The recommended values for the encoding parameters are shown below. These values are reference values. Always perform a playback test, and then adjust the parameters according to the playback state and the picture quality of the video.

Table 3.1 Recommended Parameters for General Material

Purpose	Picture Size	Bit Rate	GOP Sequence (N/M)
Highly detailed video	480 × 336	2.4Mbps - 3.6Mbps	N=12, M=2
Interlaced video	320 × 448	2.4Mbps - 3.6Mbps	N=12, M=2
Film material (24fps)	640 × 448	2.4Mbps - 3.2Mbps	N=6, M=1
Fast-moving video	320 × 224	3.2Mbps - 4.8Mbps	N=15, M=3

Depending on the video, noise may appear after encoding, or frames may be skipped during playback. In such a case, follow the procedure described below to address the situation.

- If noise is generated
 - Raise the bit rate.
 - Increase the M value, and raise the percentage of the B picture in the GOP sequence.
- If frames are skipped
 - Reduce the M value, and reduce the percentage of the B picture in the GOP sequence.
 - Lower the bit rate.
 - Reduce the image size.

3.2. Sofdec Data Creation Example

The following is an example of creating Sofdec data with CRI MPEG CRAFT, using the recommended parameters.

- (1) When starting with material created through computer graphics

Encode the material that was created, adjusting for the pixel aspect ratio and frame rate of the NTSC system.

Reduce the picture size to 480×336 , which are the recommended values for highly detailed video.

Table 3.2 Encoding Parameters

Item	Material	Encoding parameters
Picture size	640×480	480×336
Pixel aspect ratio	1.0950	1.0950
Frame rate	29.97fps	29.97fps
Bit rate	–	3.2Mbps
GOP sequence	–	N=12, M=2

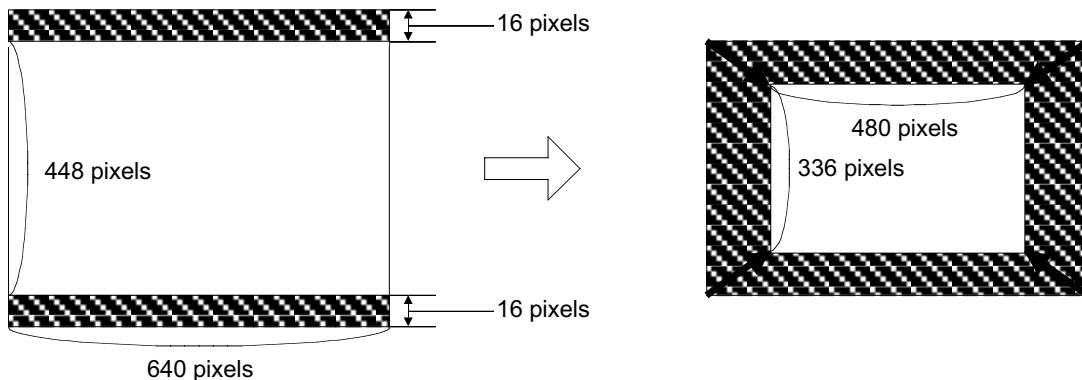


Fig. 3.1 Cropping and Reduction

Cropping and reduction are performed by movie editing software or by functions of the encoder.

Three types of processing are needed in order to create Sofdec data from a material file: video encoding, audio encoding, and multiplexing. The procedure and examples of the actual parameters are shown below.

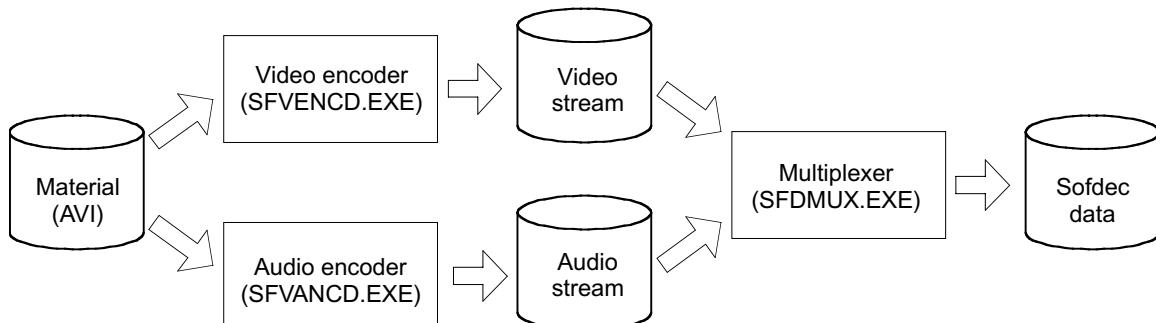


Fig. 3.2 Flow of Sofdec Data Creation

- Video encoding

**C:\>SFVENCD -in=sample.avi -mbps=3.2 -gop_n=12 -gop_m=2 -crop=640,448,0,16
-scale=480,336**

-in=sample.avi	Input file name specification (sample.avi)
-mbps=3.2	Bit rate specification (3.2Mbps)
-gop_n=12 -gop_m=2	GOP sequence specification (N=12, M=2)
-crop=640,448,0,16	Cropping specification (640 × 448)
-scale=480,336	Reduction (480 × 336)

- Audio encoding

C:\>SFAENCD sample.avi

sample.avi	Input file name specification (sample.avi)
------------	--

- Multiplexing

C:\>SFDMUX -V=sample.m1v -A=sample.sfa -S=sample.sfd

-V=sample.m1v	Video stream file name specification (sample.m1v)
-A=sample.sfa	Audio stream file name specification (sample.sfa)
-S=sample.sfd	Sofdec data file name specification (sample.sfd)

(2) When starting with natural material captured from a VCR

When capturing video from a VCR, pay careful attention to the pixel aspect ratio and frame rate settings. The following example will describe Sofdec data creation for video that was captured at 720×486 . The picture size will be 320×448 , which are the recommended values for interlaced video.

Table 3.3 Encoding Parameters

Item	Material	Encoding parameters
Picture size	720×486	320×448
Pixel aspect ratio	1.0950	1.0950
Frame rate	29.97fps	29.97fps
Bit rate	—	3.2Mbps
GOP sequence	—	N=12, M=2

Note that in some cases, the video is recorded by the VCR in interlaced format. When cropping, make sure not to shift the even and odd lines. (In our example, $(486 - 448)/2 = 19$, so in order to crop an even number of lines, the top 20 pixels and the bottom 18 pixels are removed.)

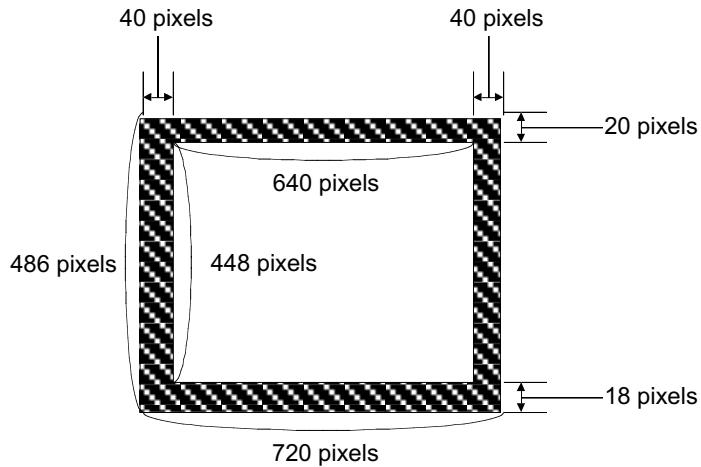


Fig. 3.3 Points Requiring Special Attention When Cropping Interlaced Material

- Video encoding

```
C:\>SFVENCD -in=sample.avi -mbps=3.2 -gop_n=12 -gop_m=2 -crop=640,448,40,20
-scale=320,448
```

-crop=640,448,0,20 Cropping specification (640×448)

Crop from even lines, to prevent a shift in interlacing.

- Audio encoding

```
C:\>SFAENCD sample.avi
```

- Multiplexing

```
C:\>SFDMUX -V=sample.m1v -A=sample.sfa -S=sample.sfd
```

(3) When starting with animated material

When 24fps material, such as animated material, is recorded on a VCR, apply reverse television-cinema conversion to convert the frame rate to 24fps.

In the case of 24fps material, a picture size of 640×448 can be selected. However, depending on the movement in the video, noise may appear or frames may be skipped. If this happens, reduce the image size.

Table 3.4 Encoding Parameters

Item	Material	Encoding parameters
Picture size	720×486	640×448
Pixel aspect ratio	1.0950	1.0950
Frame rate	24fps	24fps
Bit rate	—	3.2Mbps
GOP sequence	—	N=6, M=1

- Video encoding

C:\>SFVENCD -in=sample.avi -mbps=3.2 -gop n=6 -gop m=1 -crop=640,448,40,19 -pr=24000

-pr=24000 Frame rate specification (24fps)

- Audio encoding

C:\>SFAENCD sample.avi

- Multiplexing

C:\>SFDMUX -V=sample.m1v -A=sample.sfa -S=sample.sfd

3.3. Video Encoding

The video encoder encodes video material. The important parameters that need to be set during encoding are described below.

(1) Bit rate

The bit rate parameter specifies how much data will be provided for each second of video. A higher value provides better picture quality. A smaller value compresses the data, reducing the CPU load factor and the total amount of data.

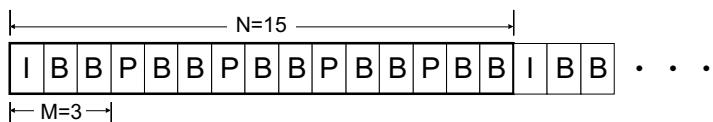
In the case of CRI MPEG CRAFT, the default bit rate is 3.6Mbps. Adjust the bit rate over a range of 2.4Mbps to 4.8Mbps in accordance with the video.

(2) GOP sequence

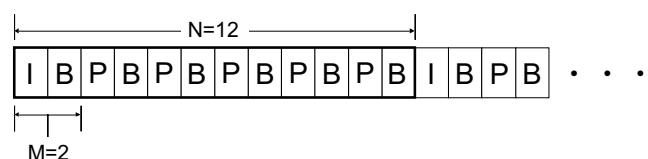
"GOP" stands for "Group of Pictures." This parameter indicates the cycle timing of the key frame (the I picture) and two differential frames (the P picture and the B picture). The cycle setting is made through two values, "N" and "M."

"N" indicates the cycle on which the I picture is displayed. "M" indicates the cycle on which the P picture is displayed. "N" must be a multiple of "M."

Examples: N=15, M=3



N=12, M=2



N=6, M=1

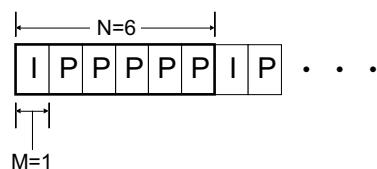


Table 3.5 Picture Types Included in the GOP Sequence

Picture type	Content	Amount of data	Playback CPU load
I picture	Key frame	Large	Small
P picture	Differential from I picture	Medium	Medium
B picture	Differential from I and P pictures	Small	Large

Amount of data: This refers to the amount of data after encoding. The I picture requires more data than the other pictures.

Playback CPU load: This refers to the load placed on the CPU in order to decode and display each picture. Although this can vary greatly, depending on the video material, on average the I picture creates the smallest load, while the P and B pictures create a progressively larger load.

Because the quantization method for the I picture is different from that of the B and P pictures, the I picture quality will not improve beyond a certain level no matter how high a bit rate is used for encoding. (In the case of a bit rate as low as that of a video CD, the I picture may have better picture quality than the B and P pictures.)

I picture-only Sofdec data should only be used to create a low CPU load movie.

(3) Color compensation

MPEG1 video compresses and retains picture information in YUV format. Typical MPEG1 devices are based on the CCIR601 video standard, and convert RGB and YUV. However, during playback, the conversion performed by the Dreamcast is slightly different from that performed by the CCIR601. The Sofdec data creation tool can apply compensation during the encoding stage that is suited for Dreamcast output. Set the parameter to perform Dreamcast color compensation.

3.4. Audio Encoding

The audio encoder encodes audio material. Sofdec Audio is used for the codec. There are no special encoding parameters. Because the compression rate is fixed (approximately 1:4), the data size after encoding is proportional to the sampling frequency and the number of audio channels. The sampling frequencies and audio channels that are supported by the encoder are shown below.

Table 3.6 Corresponding Values in Sofdec Audio

Description	Corresponding Values
Sampling frequency (Hz)	44100, 22050, 14700, 11025, 8820, 7350, 6300
Audio channel	Monaural or stereo

3.5. Multiplexing

The multiplexer interleaves the video stream with the audio stream. Multiplexing must be performed by a multiplexer designed specifically for Sofdec.

There are two multiplexers that are designed specifically for Sofdec:

- CRI MPEG Sofdec multiplexers
 - SFDMUX.EXE, which is provided with CRI MPEG CRAFT
 - The multiplexer used by the Sega Dreamcast Movie Creator

3.6. Long Material

The upper limit for AVI file size is 2GB, except for some codecs. If the material is longer than this, it is necessary to divide the material for encoding and then to link the material up again. The durations of material that can be created with 2GB of data are listed in the table below. If the material being created is longer than what is shown in the table below, processing that divides the material and then links it is needed.

Table 3.7 Duration of 2GB of Non-compressed Material

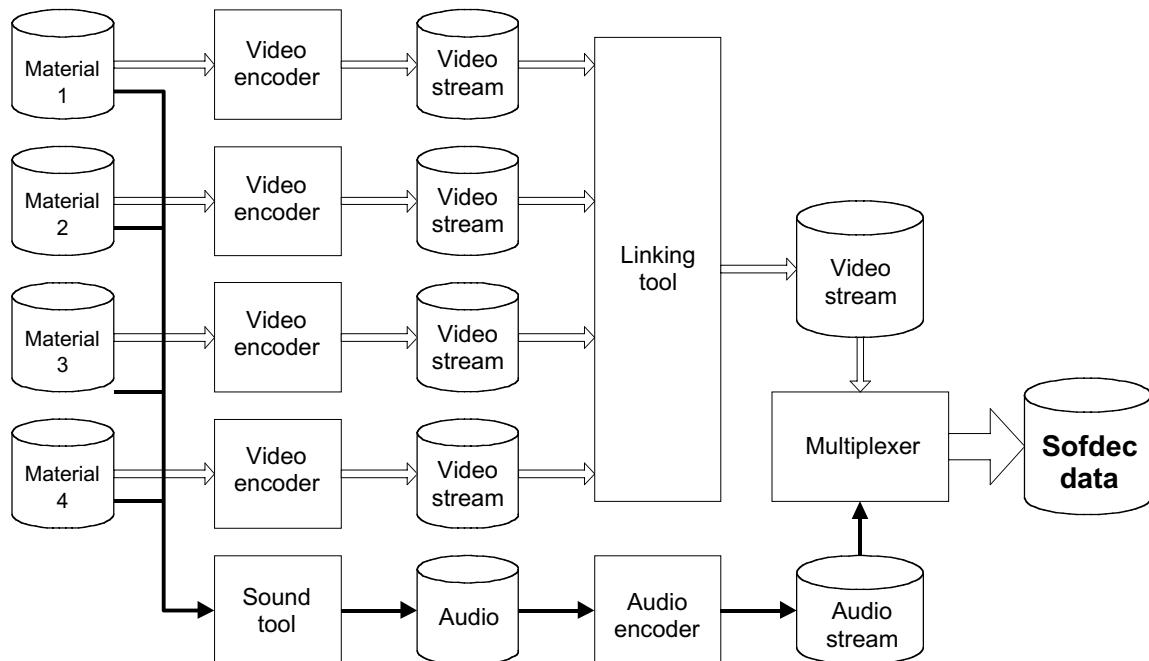
Picture size	Number of bytes per second	Duration of 2GB
320 × 224 (29.97fps)	Approximately 6.15MB/sec	Approximately 5 minutes and 30 seconds
352 × 240 (29.97fps)	Approximately 7.24MB/sec	Approximately 4 minutes and 40 seconds
320 × 448 (29.97fps)	Approximately 12.29MB/sec	Approximately 2 minutes and 45 seconds
320 × 448 (24fps)	Approximately 9.84MB/sec	Approximately 3 minutes and 20 seconds
480 × 336 (29.97fps)	Approximately 13.8MB/sec	Approximately 2 minutes and 25 seconds
640 × 448 (29.97fps)	Approximately 24.58MB/sec	Approximately 1 minute and 20 seconds
640 × 448 (24fps)	Approximately 19.69MB/sec	Approximately 1 minute and 40 seconds
640 × 480 (29.97fps)	Approximately 26.34MB/sec	Approximately 1 minute and 15 seconds
720 × 480 (29.97fps)	Approximately 29.63MB/sec	Approximately 1 minute and 00 seconds
720 × 486 (29.97fps)	Approximately 30.00MB/sec	Approximately 1 minute and 00 seconds

When performing divided encoding, a linking tool (provided by CRI) is needed.

The procedure for creating a long material is described below.

- Long Movie Creation Procedure

- (1) Extract just the audio track from the divided material, and put it into a single file.
- (2) Encode each of the individual video materials.
- (3) Use the linking tool to link the video stream.
- (4) Encode the single audio file.
- (5) Multiplex the linked video stream and audio stream, and create the Sofdec data.



4. Appendix

4.1. List of CRI MPEG CRAFT Commands

CRI MPEG CRAFT is a Sofdec data creation tool that consists of three tools.

(1) Sofdec Video Encoder (SFVENCD.EXE)

SFVENCD is a video encoder designed specifically for Sofdec and provided by CRI. This encoder is optimized for Dreamcast.

- Features

- Encoding characteristics are optimized for Dreamcast.
- Can be controlled through a command line, making flexible batch processing possible.
- Can create Sofdec data for alpha composition. (The CRI MPEG Sofdec F/X library is needed for playback.)

- Parameters

SFVENCD -in=*infile* [-out=*outfile*] [-encode={ON|OFF}]

[-mode={**DEBUG|RELEASE**}][-mbps=*mbrate*][-pr={**23976|24000|25000|29970|30000**}]
[-start=*startno*][-end=*endno*][-gop_n=*gopn*][-gop_m=*gopm*]
[-crop=*crw,crh,crx,cry*][-scope=*spw,sph,spx,spy,[spr,spg,spb]*]
[-scale=*scw,sch*][-alpha={**2|3|5|256|OFF**}][-yuv={**DC|CCIR**}][-sub=*scfile*]

SFVENCD -registry=*keycode*

Table 4.1 List of Parameters

Parameter	Function
-in= <i>infile</i>	Input file name specification.
-out= <i>outfile</i>	Output file name specification.
-encode= <i>value</i>	Encoding execution flag. OFF = Do not encode
-mode= <i>value</i>	Encoding mode. DEBUG = Fast encoding (encodes while placing no emphasis on picture quality)
-mbps= <i>mbrate</i>	Bit rate specification. Unit: Mbps
-pr= <i>value</i>	Picture rate specification. Describes the video frame rate by a factor of 1000x.
-start= <i>startno</i>	Encoding starting frame specification. -1 = from start of file
-end= <i>endno</i>	Encoding ending frame specification. -1 = to end of file
-gop_n= <i>gopn</i>	GOP sequence "N" value specification. "N" must be a multiple of "M."
-gop_m= <i>gopm</i>	GOP sequence "M" value specification.
-crop= <i>crw,crh,crx,cry</i>	Crops crw × crh pixels of the video, starting from the position (crx,cry).
-scope= <i>spw,sph,spx,spy,[spr,spg,spb]</i>	Places the picture in the position (spx,spy) of the background; the background color is (spr,spg,spb), and the background size is (spw × sph).
-scale= <i>scw,sch</i>	(spr,spg,spb), and the background size is (spw × sph).
-alpha= <i>value</i>	Encoding with alpha mask. Select from among 3, 5, and 256. (Refer to a separate manual for details.)
-yuv= <i>value</i>	YUV mode specification.
-sub= <i>scfile</i>	Subfile name specification.
-registry= <i>keycode</i>	Key code registration
-help	Display of usage

SFAENCD is an audio encoder designed especially for Sofdec, and is provided by CRI.

SFAENCD can encode an audio track in an AVI file, and WAV and AIFF format audio files.

- Features

- Audio encoder designed specifically for Sofdec.
- Uses Sofdec Audio, which provides CD-level quality, for its audio codec.
- Can encode any interval in accordance with the frequency conversion and the video in the AVI file.

- Parameters

```
SFAENCD <Infile> [Outfile] [-sf=value][-ch=value][-vsno=value][-veno=value]
```

Table 4.2 List of Parameters

Parameter	Function
Infile	Input file name specification.
Outfile	Output file name or directory specification.
-sf=value	Compressed audio sampling frequency specification. (unit: Hz)
-ch=value	Audio data channel specification. 0 = left; 1 = right; 2 = L-R switching; 3 = (L + R)/2 (MONO)
vsno=value	Video encoding starting frame number
veno=value	Video encoding ending frame number

(3) CRI Sofdec Multiplexer (SFDMUX.EXE)

SFDMUX is a multiplexer designed especially for Sofdec, and is provided by CRI.

SFDMUX loads a video stream and an audio stream, and outputs the result as Sofdec data. In order to play back data on the CRI MPEG Sofdec decoder, this tool must be used for multiplexing.

- Features

- Multiplexer designed specifically for Sofdec data.
- In order to play back data on the CRI MPEG Sofdec decoder, this tool must be used for multiplexing. (The Sega Dreamcast Movie Creator has a built-in function that is identical to this multiplexer.)

- Parameters

```
SFDMUX [-LANG={J|E}][-SUB=filename] -V=video -A=audio -S=output
```

Table 4.3 List of Parameters

Parameter	Function
-V=video	Input video file name specification.
-A=audio	Input audio file name specification.
-S=output	Output Sofdec data file name specification.
-LANG=value	Output language change (J: Japanese; E: English)
-SUB=filename	Parameter file specification.

Sega Dreamcast™

***Dreamcast
Sofdec Video
Linking Tool
(SFVCAT)
User's Manual***

1. Overview

This document explains how to use SFVCAT, the CRI MPEG Sofdec video stream linking tool.

1.1. SFVCAT

SFVCAT has the following functions:

- (1) Linking function

Links multiple CRI MPEG Sofdec video streams into one video stream.

- (2) Time code compensation function

Applies compensation to time codes so that the linked movie can be played back correctly. It is also possible to apply compensation so that the movie can be played back at any desired picture rate.

<Note>

Support for the time code compensation function is currently in the process of being implemented in the Sofdec decoding library. Do not use this function in official titles until support has been officially implemented in the Sofdec decoding library.

1.2. Description of Terms

Some of the terms used in this document are described below.

Table 1.1 Terms

Term	Definition
Video stream	A movie file that was compressed in accordance with the MPEG1 video stream format.
Picture	One frame of a movie.
Picture rate	Value that indicates the number of pictures that are displayed per second. The unit is "fps" (Frames Per Second). The NTSC picture rate is 29.97fps, and PAL is 25fps.
Time code	Value contained in each picture in a video stream that indicates the timing with which the picture must be displayed. (More precisely, the timing is indicated by two values: the TC value and the TR value.)
User-specified picture rate	Mechanism for playing back movies at a rate other than an MPEG standard picture rate, by adjusting the time code for each picture, thus displaying those pictures at intervals specified by the user.

2. Operation

2.1. Operating Environment

SFVCAT runs in the environment described below.

Table 2.1 Operating Environment

Item	Operating conditions, models, etc.
Hardware	
Machine	IBM AT-compatible
CPU	Intel Pentium processor
Main memory	At least 16MB (32MB or more recommended)
Hard disk	Free space requirements for the output file are determined by the total size of the linking files. (The drive that the file is output to does not have to be the same drive on which the linking files are stored.)
Software	
OS	Microsoft Windows 95/98/NT 4.0 or later

2.2. Supported Material

The video streams that can be linked and which SFVCAT supports are those which were encoded by the encoders listed in the table below.

Table 2.2 Supported Encoder Tools

Tool name	Provided by
CRI MPEG Sodec Video Encoder (SFVENCD)	CRI
SEGA Dreamcast Movie Creator	SEGA

3. Operation Method

3.1. Preparation

Check the following before using this tool.

(1) Operating environment

This tool runs at the Windows 95 MS-DOS prompt and at the Windows NT COMMAND prompt.

(2) Setup

Add the directory where this tool resides to the environment variable "PATH". Otherwise, copy this tool into the work directory.

(3) Hard disk free space

Free disk space that is equal to total size of the linking files is required.

(4) Material

Confirm that all streams that will be input satisfy the conditions listed below.

- (a) MPEG1 video streams that were output by a supported tool (*.m1v)
- (b) The same picture size
- (c) Identical display conditions, such as the aspect ratio.
- (d) Consistent color compensation
(Refer to section 4.3 in the SFVENCD User's Manual, "Picture Compensation Parameter "YUV Compensation."

3.2. Execution Method

(1) Example of general usage

Specify the material file names as parameters and execute the command from the command prompt as shown below.

[Example: Compressing "**a.m1v**", "**b.m1v**", and "**c.m1v**" into "**sofdec.m1v**" in the current directory]

C:\WORK>**sfvcat** -in=**a.m1v+b.m1v+c.m1v**
1) 2)

- 1) This tool
- 2) Input file name

Fig. 3.1 Example of Basic Usage from the Command Line

In this example, because the "**-out**" option is omitted, the default name "**sofdec.m1v**" is used for the output file. The linked MPEG1 video file is generated with the name "**sofdec.m1v**" in the current directory.

(2) List file specification

When executing the application, the list of files to be input can be written in a list file.

[Example: Using a list file "**video.lst**" to specify the files to be linked]

C:\WORK>**svencd -sub=video.sub**

1) 2)

Contents of subcommand file

sample.m1v
29.97
c:\work\aa.m1v
c:\work\bb.m1v
c:\work\cc.m1v

1) This tool
2) List file name

Fig. 3.2 Example of Using a List File To Specify the Files To Be Linked

(3) Command line option restrictions

- There is no significance attached to the order of description or the use of upper-case/lower-case letters.
- The file base name can consist of up to eight characters, with a three-character extension.
- All options and parameters should contain no spaces.
- Do not insert a space or tab before or after equal signs "=".

4. Parameter Specifications

Title	Subcommand	Keyword	No.
Subcommand specification	Input file list	in	1.1

[Format] **-in=filelist**

[Input] **filelist** : List of video files to be input

[Function] This parameter specifies the list of video files to be linked.

When specifying multiple streams, link them with "+" signs.

(Example: **-in=movie1.m1v+movie2.m1v+movie3.m1v**)

Up to 64 files may be specified.

Title	Subcommand	Keyword	No.
Subcommand specification	Output file name	out	1.2

[Format] **-out=filename**

[Input] **filename** : Output file name

[Function] This parameter specifies the name of the video file to be output.

If omitted, "**s0fdec.m1v**" is specified.

Title Subcommand specification	Subcommand Picture rate	Keyword pr	No. 1.3
--	-----------------------------------	----------------------	------------

[Format] **-pr=rate**

[Input] **rate** : Picture rate

3.97fps, 24fps, 25fps, 29.97fps, 30fps, 50fps, 59.94fps, 60fps

[Function] This parameter specifies the picture rate for the time codes to be written in the linked video stream.
(unit: fps (frames per second))

If omitted, the picture rate of the first stream in the specified input file list is used.

Title Subcommand specification	Subcommand Operating mode	Keyword mode	No. 1.4
--	-------------------------------------	------------------------	------------

[Format] **-mode=mode**

[Input] **mode** : Operating mode

[Function] This parameter specifies the operating mode when the command is executed.

(1) mode

This parameter specifies the operating mode when the command is executed.

(a) Range : Either "**NONE**" or "**NORMAL**"

If "**NONE**" is specified, linking is not performed; only stream analysis is performed. This mode is used to find out how many frames there are in a stream, etc.

If "**NORMAL**" is specified, linking is performed.

(b) If omitted : "**NORMAL**"

Title	Subcommand	Keyword	No.
Subcommand specification	listfile	list	1.5

[Format] **-list=listfile**

[Input] **listfile** : List file

[Function] This command reads, as a parameter, a list file that includes a list of input files, the name of the output file, and the bit rate. This command cannot be specified if the "**-in**", "**-out**" or "**-pr**" options are specified.

- List file format

The list file must be a text file written in the format described below.

Line 1	sofdec.m1v	←	Output file name
Line 2	29.97	←	Picture rate
Line 3	c:\movie1.m1v	←	Stream to be linked
Line 4	c:\movie2.m1v		Stream to be linked
•	c:\movie3.m1v		Stream to be linked
•	c:\movie4.m1v		Stream to be linked
•	c:\movie5.m1v		Stream to be linked
•	c:\movie6.m1v		Stream to be linked

Fig. 4.1 Example in Which the Files To Be Linked Are Specified in a List File

Sega Dreamcast™

Dreamcast
Sofdec Video Encoder
(SFVENC'D)
User's Manual

1. Overview

The Sofdec Video Encoder "SFVENCD" is a tool for creating MPEG1 video files for Sofdec from video material.

This tool is capable of handling the encoding that is needed to change image sizes as well as special, such as for alpha movies. The tool includes the following functions:

- (1) General encoding that creates an MPEG1 video file from an AVI file

This tool allows you to specify parameters concerning bit rate and picture quality, and then encodes video material.

- (2) Encoding in conjunction with size changes

When the image size of video material is different from the movie size that is needed, you can specify that the SFVENCD change the size.

You can also encode video material without needing to use any external movie editing software.

- (3) Special encoding that is used to create movie data for alpha movies

"Sofdec F/X" is provided as a special playback function for Sofdec. This function can overlay still images and polygons in movies being played back on Dreamcast. SFVENCD can encode movie data for use with Sofdec F/X.

An alpha movie is a method that uses the alpha channel for overlays in Sofdec F/X.

The video material that is used to create an alpha movie must have an alpha channel in the movie data. A detailed description of Sofdec F/X will be found in the "Sofdec Data Creation Guide," which is scheduled for revision in the near future.

2. System Specifications

2.1. Operating Environment

The hardware and software requirements of this system are listed below.

Table 2.1 Operating Environment

Item	Operating Conditions, Models, etc.
Hardware	
Machine	IBM AT-compatible
CPU	Intel Pentium processor (Pentium II 350MHz or faster, 440BX chipset recommended)
Main memory	At least 16MB (32MB or more recommended)
Hard disk	Free space is required for the output file in accordance with the bit rate. (When encoding one minute of material at 4Mbps, approximately 30MB of free space is required.)
Capture card (*1)	Canopus DVREx-M1/DVRaptor recommended
Software	
OS	Microsoft Windows 95/98/NT4.0 or later (Windows NT4.0 or later recommended)
Driver/DLL	LsxMpeg DLI (required; provided with system) If Canopus AVI files will be handled: • Canopus DV Codec (*2)

(*1) Only if video capture capability is needed

(*2) Must be purchased separately if using a machine that does not already have a Canopus capture card installed.

1.2. Input Specifications

The following file formats can be specified for input files.

Table 2.2 Input File Formats

File format	Description
AVI files	
File type	Microsoft AVI file
Compression format	<ul style="list-style-type: none">• Noncompressed• Canopus DV Codec format <p>Either of the above is recommended</p>
Number of colors	24-bit/32-bit color
Image size	128 × 32 to 1024 × 768 (in 32 × 16-pixel units; this restriction does not apply when changing the size through image compensation, however)
Frame rate	30fps or less (29.97fps recommended)
Pel aspect ratio (*1)	1.0950 (ITU-R601 525 lines)

(*1) Ratio of vertical/horizontal size of pixels on screen

[Recommended values]

Image size: 320 × 224 to 320 × 240

Frame rate: 29.97fps

<Supplement>

The current version (Ver. 2.20) of SFVENCD can handle Matrox DigiSuite-format AVI files, with conditions.

The requirements for using such files are listed below:

- (a) OS: Windows NT4.0
- (b) Restrictions: Only up to 1GB of frames are valid (frames beyond 1GB are ignored)
- (c) Driver, etc.: Matrox DigiSuite utilities 3.0 or later (*1)
Matrox software-only Video for Windows Codec (*2)

(*1) On machines in which the DigiSuite hardware is installed

(*2) On machines in which the DigiSuite hardware is not installed

3. Operation Method

3.1. Preparation

Check the following items before using this tool.

(1) Operating environment

This tool operates from the MS-DOS prompt in Windows 95 or the COMMAND prompt in Windows NT.

(2) Setup

(a) Path setup

Either add the directory where this tool resides to the "PATH" environment variable, or else copy this tool to the work directory.

(b) DLL copy

Place "LsxMpeg.dll" (included with this system) either in the same directory as this tool, or else in the Windows system directory. ("c:\Windows\system" in the case of Windows 95, or "c:\winnt\system32" in the case of Windows NT)

(3) Hard disk free space

Allocate the necessary amount of free space on a hard disk in accordance with the bit rate specified during the encoding process and the length of the material. For example, when encoding one minute of material at 4Mbps, approximately 30MB of free space is required.

(4) Material

Use one of the compression formats shown in Table 2.2 for the file containing the input material. Make sure that either 24-bit color or 32-bit color is used.

3.2. Execution Method

(1) General-purpose Usage Example

Using the name of the file containing the material as the parameter, execute the command at the command prompt as shown below. A file with the extension ".M1V" is generated in the current directory.

[Example: Compressing "**sample.avi**" into "**sample.m1v**" in the current directory]

C:\WORK>**sfenccd -in=sample.avi**
1) 2)

- 1) This tool
- 2) Input file name

Fig. 3.1 Example of Basic Usage from the Command Line

You can also specify the bit rate and other encoding parameters as options.

[Example: Creating "**movie200.m1v**" with a bit rate of 2.0Mbit/s]

C:\WORK>**sfenccd -in=sample.avi -out=movie200.m1v -mpbs=2.0**
1) 2) 3) 4)

- 1) This tool
- 2) Input file name
- 3) Output file name
- 4) Bit rate specification

Fig. 3.2 Example in Which the Bit Rate and Output File Name Are Specified

(2) Subcommand file specification

Regarding application execution, it is possible to use a subcommand file (in which the option parameters are specified as a file) to specify the encoding parameters.

[Example: Specifying the encoding parameters through a subcommand file "**video.sub**"]

C:\WORK>**sfenccd -sub=video.sub**
1) 2)

- 1) This tool
2) Subcommand file name

— Contents of subcommand file —

-in=sample.avi
-out=movie200.m1v
-mbps=2.0

Fig. 3.3 Example in Which Parameters Are Specified Through a Subcommand File

(3) Command line option restrictions

- There is no significance attached to the order of description or the use of upper-case/lower-case letters.
- The file base name can consist of up to eight characters, with a three-character extension.
- All options and parameters should contain no spaces.
- Do not insert a space or tab before or after equal signs "=".

3.3. Work Procedure

The following is a specific description of the work procedure using SFVENCD.

3.3.1. Work Steps

Follow the steps described below.

(1) Create material

Create the AVI file that will be the source material for encoding. Use an existing video capture card, etc., to load a movie into the PC.

The AVI file that is output must be of one of the formats shown in Table 2.2.

(2) Encode

Use this tool and begin encoding. Set the encoding parameters in accordance with the desired output file bit rate and image size.

(3) Check output image

Make sure that the encoded image has the right image quality and playback status.

After multiplexing the image, use the actual Dreamcast system to output the image on a TV monitor and check the image quality, color tones, smoothness of playback, etc.

<Remarks>

- The image quality of an output movie will vary greatly according to the content of the images and the parameters. If the initial output quality is inadequate, try changing the encoding parameters and remaking the material file.
- In order to obtain better image quality, it may be necessary to repeat the encoding work several times through a process of trial and error. Because parameters that were used for encoding in the past are very important information, we recommend that you always keep a record of the parameters that were used for encoding.
- If movie playback is not smooth, try adjusting the image size, the bit rate, the GOP N/M value, etc., in order to reduce the decoding load.
- Simple image confirmation is possible using the Windows Media Player or a general-purpose MPEG playback application. However, because there are differences in display methods and color conversion methods, the final check of image quality must be performed on a television monitor using the actual Dreamcast system. In addition, when using the Windows Media Player, the final frame may not be displayed correctly.

3.3.2. Encoding Example

(1) Encoding with a high bit rate

A higher bit rate improves the image quality. It is important to understand, however, that if the bit rate is too high, the decoding load during playback will result in dropped frames.

[Example: Creating "**movie450.m1v**" with a bit rate of 4.5Mbit/s]

```
C:\WORK>sfvencd -in=sample.avi -out=movie450.m1v -mpbs=4.5 -gop m=1
```

Fig. 3.4 Example of Encoding with a High Bit Rate

<Remarks>

- Normally, smooth playback is possible with a bit rate of 4.0Mbps.
- Decreasing the resolution (image size) will decrease the load during playback.
- Reducing the GOP M value (i.e., reducing the B picture) will also decrease the load.

(2) Encoding with high resolution

When displaying fine text information on a black background, such as for the credits at the end of a movie, the picture quality can be improved by increasing the picture size.

[Example: Encoding 720×486 material at 320×448 with high resolution in the vertical direction]

```
C:\WORK>sfvencd -in=sample2.avi -crop=640,448,40,19 -scale=320,448
```

Fig. 3.5 Example of Encoding with High Resolution

<Remarks>

- The picture that is to be used as the material should originally be a high-resolution image.
- Extract the required portion of the original picture (-crop), and then resize the picture (-scale) in accordance with the resolution during playback. You can also use an existing editing application, such as Adobe After Effects.
- If the playback load is high due to the high resolution, reduce the bit rate.
- Reducing the GOP M value (i.e., reducing the B picture) will also decrease the load.

(3) Encoding an alpha movie

This encodes video material that has an alpha channel for a Sofdec F/X alpha movie.

[Example: Encoding for a two-step alpha movie]

C:\WORK>**sfvencd -in=sample3.avi -alpha=2**

Fig. 3.6 Example of Alpha Movie Encoding

<Remarks>

- The alpha movie resolution can be selected from among 2-value, 3-value, 5-value, and 256-value modes.
- The differences in the alpha movie effects due to the different value modes will be explained in the scheduled revision of the "Sofdec Data Creation Guide."

3.3.3. Default Values

The default values for parameters that are not specified during the encoding process are shown in the table below. (These values are identical to the values shown for "Omissible" parameters indicated in Chapter 4.)

Encoding mode	RELEASE
Encoding start frame	0
Encoding end frame	-1
Encoding execution flag	ON
Encoding bit rate	3.6 [Mbps]
Picture rate	29.97 [fps]
GOP cycle N value	15
GOP cycle M value	3
YUV compensation	DC

4. Parameter Specifications

4.1. Input/Output Files

Item	Input file
Format	-in=filename
Parameters	filename Input file path name
Range of values	Path name for material file. Long file names are not supported.
Omissible	No
Function	For details on the format of material files, refer to Chapter 2.

Item	Output file
Format	-out=filename
Parameters	filename Output file path name
Range of values	Path name for material file. Long file names are not supported.
Omissible	Yes (If omitted, the input file name will be used, with the ".M1V" extension.)
Function	This item specifies the path name for the video stream that will be output.

4.2. Encoding Parameters

Item	Encoding mode																				
Format	-mode=mode																				
Parameters	mode Encoding mode																				
Range of values	Select one of DEBUG , NORMAL , or RELEASE .																				
Omissible	Yes (If omitted: RELEASE)																				
Function	<p>This item specifies the encoding mode. Select one of three modes: DEBUG : Gives priority to encoding speed. NORMAL : Encodes with intermediate picture quality at intermediate speed. RELEASE : Gives priority to picture quality.</p> <table border="1"> <thead> <tr> <th></th> <th>DEBUG</th> <th>NORMAL</th> <th>RELEASE</th> </tr> </thead> <tbody> <tr> <td>GOP N value</td> <td>15</td> <td>15</td> <td>15</td> </tr> <tr> <td>GOP M value</td> <td>1</td> <td>3</td> <td>3</td> </tr> <tr> <td>Functional compensation range</td> <td>Small</td> <td>Medium</td> <td>Large</td> </tr> <tr> <td>Functional compensation speed</td> <td>Fastest</td> <td>Fast</td> <td>Fast</td> </tr> </tbody> </table>		DEBUG	NORMAL	RELEASE	GOP N value	15	15	15	GOP M value	1	3	3	Functional compensation range	Small	Medium	Large	Functional compensation speed	Fastest	Fast	Fast
	DEBUG	NORMAL	RELEASE																		
GOP N value	15	15	15																		
GOP M value	1	3	3																		
Functional compensation range	Small	Medium	Large																		
Functional compensation speed	Fastest	Fast	Fast																		

Item	Encoding start frame number
Format	-start=value
Parameters	value Encoding start frame number
Range of values	0 - (Total number of frames in material -1)
Omissible	Yes (If omitted: 0)
Function	This item specifies the frame number where encoding is to start. If " 0 " is specified, encoding starts from the first frame in the material file.

Item	Encoding end frame number
Format	-end=value
Parameters	value Encoding end frame number
Range of values	-1 , or " encoding start frame number " to (Total number of frames in material -1)
Omissible	Yes (If omitted: -1)
Function	This item specifies the frame number where encoding is to end. If " -1 " is specified, encoding continues until the last frame is reached.

Item	Encoding execution flag
------	-------------------------

Parameters	flag Encoding execution flag
Range of values	Either ON or OFF
Omissible	Yes (If omitted: ON)
Function	This item specifies whether encoding processing is actually to be executed or not. If " OFF " is specified, the input file information is displayed and then processing ends.

Item	Encoding bit rate (*) (unit: bits/second)
Format	-bitrate=value
Parameters	value Bit rate
Range of values	4000 - 104000000 [bps]
Omissible	Yes (If omitted: 3600000)
Function	This item specifies the bit rate for encoding a video stream. The value is indicated in decimal notation, and represents bits per second. (The bit rate can also be expressed using 1K = 1000, or 1M = 1000000.)
Remarks	Cannot be specified if " -mbps " or " -byterate " is specified.

Item	Encoding bit rate (*) (unit: Mbits/second)
Format	-mbps=value
Parameters	value Bit rate
Range of values	0.004 - 104 [Mbps]
Omissible	Yes (If omitted: 3.6)
Function	This item specifies the bit rate for encoding a video stream. The value is indicated in decimal notation, and represents Mbits per second.
Remarks	Cannot be specified if " -bitrate " or " -byterate " is specified.

Item	Encoding bit rate (*) (unit: bytes/second)
Format	-byterate=value
Parameters	value Bit rate
Range of values	500 - 13000000 [bytes/sec]
Omissible	Yes (If omitted: 450000)
Function	This item specifies the bit rate for encoding a video stream. The value is indicated in decimal notation, and represents bytes per second.
Remarks	Cannot be specified if " -bitrate " or " -mbps " is specified.

Item	Picture rate
------	--------------

Parameters	value Picture rate
Range of values	Select one of 23976 , 24000 , 25000 , 29970 , or 30000 .
Omissible	Yes (If omitted: 29970 (29.97fps))
Function	This item specifies the picture rate of a video stream. The value is expressed in units of [frames/1000 seconds].

Item	GOP cycle N value
Format	-gop_n=value
Parameters	value GOP cycle N value (I picture cycle)
Range of values	1 - 64
Omissible	Yes (If omitted: 15)
Function	This item specifies the GOP N value (I picture appearance cycle). The N value must be a multiple of the M value.

Item	GOP cycle M value
Format	-gop_m=value
Parameters	value GOP cycle M value (I/P picture cycle)
Range of values	1 - 16
Omissible	Yes (If omitted: 3)
Function	This item specifies the GOP M value (I picture or P picture appearance cycle).

4.3. Picture Compensation Parameters

Item	Cropping
Format	-crop=width,height,xoffset,yoffset
Parameters	<p>Width Width of picture after cropping height Height of picture after cropping xoffset X coordinate offset of picture after cropping versus the original picture yoffset Y coordinate offset of picture after cropping versus the original picture</p>
Range of values	<p>Specify a range of values for the cropped picture that lies within the original picture. (Refer to the diagram below.)</p> <p>If this parameter is not used in conjunction with the resize function, the width and height of the picture after cropping must conform with the image sizes permitted by the input file format.</p>
Omissible	Yes (If omitted, cropping processing is not performed.)
Function	This function extracts and encodes a portion of a picture from the original material.
Remarks	This item can not be specified at the same time as the items that add a white border or that produce an alpha movie.

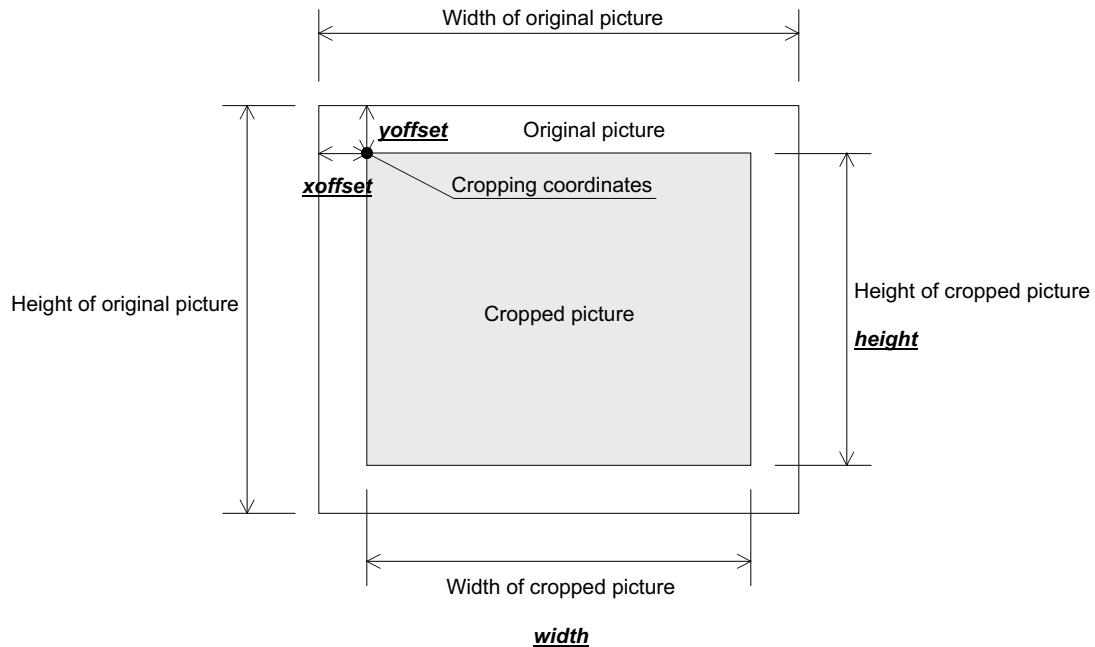


Fig. 4.1 Cropping

Item	White border
Format	-scope=width,height,xoffset,yoffset{,r,g,b}
Parameters	<p>Width Width of picture after adding white border</p> <p>Height Height of picture after adding white border</p> <p>Xoffset X coordinate offset of picture after adding white border versus the original picture</p> <p>Yoffset Y coordinate offset of picture after adding white border versus the original picture</p> <p>r,g,b Pixel value for white border (If omitted: 0,0,0 = black)</p>
Range of values	<p>Specify a range of values that will keep the original picture within the picture after the white border is added. (Refer to the diagram below.)</p> <p>If this parameter is not used in conjunction with the resize function, the width and height of the picture after the white border is added must conform with the image sizes permitted by the input file format.</p>
Omissible	Yes (If omitted, white border processing is not performed.)
Function	This function adds a white border to the original material and then encodes the material.
Remarks	This item can not be specified at the same time as the items that crop the picture or that produce an alpha movie.

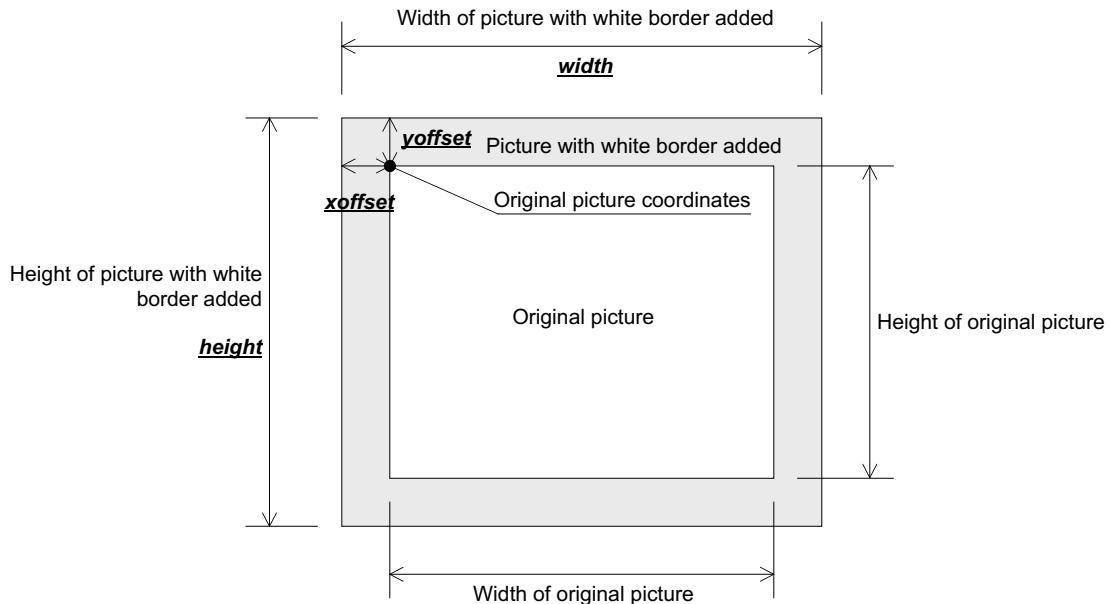


Fig. 4.2 Adding a White Border

Item	Resizing (averaging)
Format	-scale=width,height
Parameters	Width Width of picture after resizing height Height of picture after resizing
Range of values	The width and height of the picture after resizing must conform with the image sizes permitted by the input file format.
Omissible	Yes (If omitted, resizing processing is not performed.)
Function	This function resizes the original picture by averaging the pixel values and then encodes the resulting picture.
Remarks	This item can be combined with the items that crop the picture or that add a white border. This item can not be specified at the same time as resizing (down sampling) and alpha movie.

Item	Resizing (down sampling)
Format	-scaleds=width,height
Parameters	Width Width of picture after resizing height Height of picture after resizing
Range of values	The width and height of the picture after resizing must conform with the image sizes permitted by the input file format.
Omissible	Yes (If omitted, resizing processing is not performed.)
Function	This function resizes the original picture by sampling the pixel values and then encodes the resulting picture.
Remarks	This item can be combined with the items that crop the picture or that add a white border. This item can not be specified at the same time as resizing (averaging) or alpha movie.

Item	Alpha movie creation
Format	-alpha=value
Parameters	Value Alpha composition algorithm
Range of values	Select from among 2 , 3 , 5 , 256 or OFF .
Omissible	Yes (If omitted: OFF)
Function	This function creates an alpha movie video stream. Select the algorithm that is best suited to the application. 2: For 2-step alpha composition 3: For 3-step alpha composition 5: For 5-step alpha composition 256: For 256-step alpha composition OFF: Adds no data for alpha composition
Remarks	This item cannot be specified at the same time as transformation processing (cropping/white border) or resizing processing.

Item	YUV compensation
Format	-yuv=mode
Parameters	mode YUV compensation mode
Range of values	Either DC or CCIR
Omissible	Yes (If omitted: DC)
Function	When " DC " is specified, compensation is applied to the YUV value during encoding in accordance with the NTSC output characteristics of the Dreamcast. When " CCIR " is specified, encoding is performed in compliance with MPEG standard CCIR (with no compensation).

4.4. Miscellaneous Parameters

Item	Log display mode
Format	-log=mode
Parameters	mode Log display mode
Range of values	Select one of NORMAL , CSV , or NONE
Omissible	Yes (If omitted, the file is not loaded.)
Function	This item sets the output format for the log data collected during encoding. Select one of the following three modes: NORMAL: Displays progress in terms of the number of frames, the current picture quality, and the highest and lowest picture quality. CSV: Outputs the picture quality values with comma delimiters. This mode is used for data collection. NONE: Does not display the progress.

Item	Subcommand file
Format	-sub=filename
Parameters	filename Subcommand file path name
Range of values	Path name for the subcommand file. Long file names are not supported.
Omissible	Yes (If omitted, the file is not loaded.)
Function	This item loads the parameters written in the specified file.

5. List of Error Messages

5.1. Option Specification-related Errors

These errors are generated when an option is specified incorrectly.

Error Messages	
ce20: Illegal arguments.	
Cause	A parameter that is not supported by this tool was specified.
Remedy	Delete the invalid parameter string.
ce21: Illegal option or argument.	
Cause	An option that is not supported by this tool was specified.
Remedy	Delete the invalid option from the parameters.
ce25: Multiple option.	
Cause	The same option was specified more than once.
Remedy	Specify each option no more than once.
ce26: Argument of option was not specified.	
Cause	An option parameter (the portion that follows "=") was not specified.
Remedy	Always specify the parameters required by an option.
ce27: Illegal arguments of option.	
Cause	An illegal value was specified in an option parameter (the portion that follows "=").
Remedy	Specify the option parameters correctly.

5.2. Input File-related Errors

These errors are generated if an input file cannot be opened, or if an input file has an unsupported file format.

Error Messages	
ce30: Input file name missing.	
Cause	No input AVI file name was specified.
Remedy	Specify an input file name.
ce31: Can not access input file.	
Cause	Either the file does not exist, or it is being used by another application.
Remedy	Confirm that the file exists and is not in use.
ce32: Can not access input AVI file.	
Cause	Cannot confirm the format of the input AVI file.
Remedy	Confirm that the input file is a legal AVI file.
ce33: Can not get frame size.	
Cause	Cannot get frame size from the input AVI file.
Remedy	Confirm that the input file is a legal AVI file.
ce34: Can not get total frame number.	
Cause	Cannot get the total number of frames from the input AVI file.
Remedy	Confirm that the input file is a legal AVI file.
ce35: Can not get buffer size.	
Cause	Cannot get the buffer size needed for bitmap expansion from the input AVI file.
Remedy	Confirm that the input file is a legal AVI file.
ce36: Illegal frame size value.	
Cause	The input file picture size or the size after picture compensation is illegal.
Remedy	Either use an input file with a legal picture size, or apply picture compensation so that the resulting picture size is correct.
ce37: Illegal start frame number.	
Cause	The starting frame number is incorrect.
Remedy	Specify a number that is within the total number of frames in the material.
ce38: Illegal end frame number.	
Cause	The ending frame number is incorrect.
Remedy	Specify a number that is within the total number of frames in the material, or "-1."

5.3. Parameter-related Errors

These errors are generated when a parameter is specified outside the range supported by this tool.

Error Messages	
ce40: Illegal start/end frame number.	
Cause	A starting frame number that is greater than the ending frame number was specified.
Remedy	Specify a starting frame number that is smaller than the ending frame number.
ce41: Illegal picture rate value.	
Cause	An unsupported picture rate was specified.
Remedy	Specify a supported picture rate.
ce42: Illegal bit rate value.	
Cause	A picture rate outside the supported range was specified.
Remedy	Specify a picture rate inside the supported range.
ce43: Illegal GOP N value.	
Cause	A GOP N value outside the supported range was specified.
Remedy	Specify a GOP N value inside the supported range.
ce44: Illegal GOP M value.	
Cause	A GOP M value outside the supported range was specified.
Remedy	Specify a GOP M value inside the supported range.
ce45: GOP N value was not multiple of M value.	
Cause	The GOP N value is not a multiple of the GOP M value.
Remedy	Always specify an integer multiple of the GOP M value for the GOP N value.
ce46: Must be Dreamcast YUV Mode when use alpha channel.	
Cause	The YUV compensation CCIR option and alpha movie are both specified at the same time.
Remedy	When using alpha movie, specify "DC" for YUV compensation.

ce50: both Transform and Alpha movie were specified.

Cause	Alpha movie and transform processing are both specified at the same time.
Remedy	When using alpha movie, do not specify transform processing.

ce51: both Scaling and Alpha movie were specified.

Cause	Alpha movie and resize processing are both specified at the same time.
Remedy	When using alpha movie, do not specify resize processing.

ce52: Multiple Transform specification.

Cause	Cropping and white border are both specified at the same time.
Remedy	Specify either cropping or white border, not both.

ce53: Multiple Scaling specification.

Cause	Averaging and down sampling resize processing are both specified at the same time.
Remedy	Specify either averaging or down sampling, not both.

ce54: Multiple Bitrate specification.

Cause	Multiple options that specify a bit rate are specified.
Remedy	Either specify the bit rate once, or delete all of the options.

6. MPEG1 Video Data Creation Guide

This chapter provides a simple description of the features of MPEG1 video, and also explains key points concerning the process of actually creating and encoding a video.

6.1. MPEG1 Video File Creation

This section provides a simple description of key points concerning the creation of MPEG1 video.

(1) Number of pixels (screen display size)

When playing back a video with the Sofdec library, the vertical size must be a multiple of 32, and the horizontal size must be a multiple of 16.

(2) Video processing

Pay careful attention to the aspect ratio when trimming and expanding video.

In order to prevent graininess when expanding a video, good results can be obtained through interpolation using an effector, etc.

Using a field interpolation effector can prevent comb-like lines in video that is shot with a video camera.

(3) Checking the picture quality

We recommend checking the picture quality of encoded data on a TV.

Color tones and reproduction are completely different on a computer monitor compared with a TV. An image that appears excellent on a computer monitor can be very poor when displayed on a TV. Also, if the image quality prior to encoding is poor, the image quality after encoding will be worse.

(4) Encoder

The picture quality is dependent on the encoder and on the encoding parameters.

Even if the data will be encoded on another machine, the same key points still apply to the creation of the data.

(5) Do not multiplex on an MPEG encoder

Some encoders handle all processes from encoding to multiplexing. Because it is difficult to simply replace the audio with Sofdec audio, the audio and video should always be created separately.

(6) Allocation of the data

Increasing the bit rate is a comparatively simple method for improving picture quality. When doing so, consider the GD transfer speed and the allocation of capacity for audio.

For example, if the GD transfer speed is 1MB (8Mbits) per second, and the audio requires 0.4Mbps, up to 7.6Mbps can be allocated for video.

6.2. To Encode Pictures with Better Picture Quality

Several points that are important for improving picture quality are explained below.

6.2.1. High Bit Rate Allocation

When encoding the following types of video, we recommend increasing the video bit rate in order to improve the picture quality.

- (a) Fast-moving video (races, sporting events, etc.)
- (b) Video that includes many colors on the screen
- (c) Video in which sand, dust, rain, or other highly granular or fine substances appear
- (d) Video that includes frequent scene changes, camera movement, and fade-in/fade-outs
- (e) Video of a subject that moves in an irregular fashion
- (f) Video with sharp delineations between colors (such as animation, computer graphics, overlays, or subtitles)

6.2.2. Hindrances to Improved Picture Quality

Video that was encoded by MPEG can be subject to block distortion (*1) and mosquito noise (*2), which both adversely affect quality. These problems are particularly obvious when cell animation is encoded, but also become apparent for other subjects.

Specific examples of hindrances to picture quality are described below.

(1) Flickering light source

The flicker of fluorescent lights, flickering light reflected on a wall, etc., can lead to block distortion. Video in which there are sudden changes from bright to dark is also subject to noise.

Extra care is required because these phenomena are difficult to detect when using a TV to check video shot with a video camera.

(2) Video of a lot of moving objects

If there are numerous objects moving simultaneously on the screen, the picture quality after encoding may be poor, and playback may also suffer from problems.

(3) Video with frequent fade-ins/fade-outs

MPEG is not well-suited for processing fade-ins and fade-outs. Because the difference between light and dark is large during fade processing, block distortion can easily appear.

(4) Video with lots of zooming and scrolling

Zooming and scrolling creates a large processing load during playback. Be especially careful of repeated zooming and lengthy scrolling.

[Remarks]

*1: Block distortion

MPEG encoding processing is performed in 8 x 8-pixel blocks; "block distortion" refers to a phenomenon in which the boundaries between adjacent blocks create a sharp linear delineation.

*2: Mosquito noise

In this phenomenon, quantization error spreads across entire blocks, resulting in a blurry appearance. This phenomenon primarily manifests itself along contour lines.

6.2.3. Remedies

The following are some remedies for hindrances to improved picture quality.

(1) To improve the picture quality

- Increase the bit rate.
- Increase the resolution.
- Use the B picture more.

Each of these is an important element for improved picture quality. The drawback is that these increase the processing load during playback. Increased use of the B picture increases the efficiency of compression, increasing the bit rate for the I/P pictures on a relative basis.

(2) Resolving problems during playback

- Lower the bit rate.
- Decrease the resolution.
- Reduce the use of the B picture.

These are the complement of the remedies for improving the picture quality.

The best picture quality is the highest picture quality that can be attained without generating problems during playback.



***Dreamcast
CRI MPEG
Sofdec Multiplexer
(SFDMUX)
User's Manual***

1. Overview

The Sofdec Multiplexer (SFDMUX) is a tool that creates MPEG Sofdec files from MPEG1/Video files and audio files.

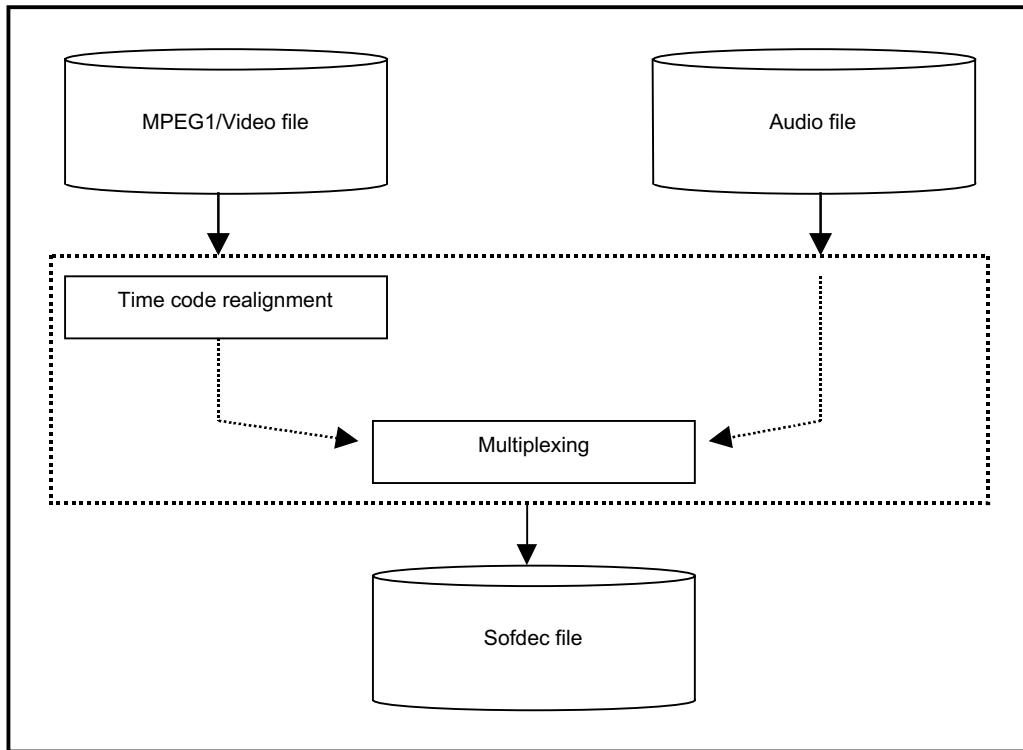


Fig. 1.1 Overview of the Operation of This Tool

2. Operating Environment

2.1. Execution Environment

This tool operates in the following environment. It will not run under conventional MS-DOS.

- (a) Windows 95 MS-DOS prompt
- (b) Windows NT COMMAND prompt

2.2. Environment Setup

Before using this tool, check the following.

- (1) Path setting

Add the directory where this tool resides to the environment variable "PATH". Otherwise, copy this tool into the work directory.

- (2) Hard disk free space

Free disk space that is equal to approximately 1.02 x the total size of the audio and video files to be multiplexed is required.

For example, if a 10MB audio file and a 40MB video file are to be multiplexed, free space of approximately 51MB is required.

3. Input/Output Files

3.1. Input Files

Table 3.1 Input Files

Type	Description
Video file (M1V file)	This is an MPEG1/Video video file. In order to improve playback performance on the Dreamcast, the number of pixels in the horizontal direction must be a multiple of 32, and the number of pixels in the vertical direction must be a multiple of 16.
Audio file (SFA file)	This is an audio file. The current Sofdec library only supports Sofdec/Audio.
Subcommand file	This is a text file that contains commands that control this tool. This file is not specifically required.

3.2. Output File

Table 3.2 Output File

Type	Description
Sofdec file (SFD file)	This file contains the audio and video handled by the Sofdec library.

4. Multiplexing

4.1. Creating Files One at a Time

It is a simple matter to create Sofdec files one at a time by specifying them as parameters in the command line options.

(1) Example

In this example, we will multiplex one audio file and one video file, and create one output file. The most basic example is shown below.

```
Multiplex Sofdec/Audio, and create a Sofdec playback file.

+--- Prompt
|
|       +----- Input files -----+ +- Output file --+
↓      |                         ||   |
C:\>SFDMUX -V=OPENING.M1V -A=OPENING.SFA -S=OPENING.SFD
     1)          2)          3)          4)

1) Name of this tool
2) Video file name
3) Audio file name
4) Sofdec file name
```

Fig. 4.1 Example of Basic Usage from the Command Line

(2) Notes on description

when specifying the command line options, observe the following rules:

- Specify one video file, one audio file, and one output file. (The order in which they are specified does not matter.)
- Type each option and parameter close together. Do not insert tabs or spaces on either side of the equal sign "=".

(3) Command line option commentary

There are three command line options shown in the above Fig. 4.1. Normally, these are the only three options that you need to remember. For further details, refer to Appendix A.

- (a) -V : Specifies the video input file name. If the extension is omitted, "M1V" will be assumed.
- (b) -A : Specifies the audio input file name. If the extension is omitted, "SFA" will be assumed.
- (c) -S : Specifies the output file (Sofdec file) name. If the extension is omitted, "SFD" will be assumed.

4.2. Batch Creation of Multiple Files

The most efficient way to create multiple Sofdec files at one time is to use subcommand files or batch files. This section will explain how to use subcommand files.

(1) Example

Let's change the example shown above in Fig. 4.1 so that it uses a subcommand file. The restrictions on file names for the subcommand file are the same as for command line options.

In the command line, use the "-SUB" command and then specify only the subcommand file name. Write all of the output file names and the input file names in the subcommand file.

```
[Command line]
+-- Prompt
|
|           +- Subcommand -+
↓           |           |
C:>SFDMUX -SUB=MOVIE.SUB
    1)           2)

1)   Tool name
2)   Subcommand file name (contents are shown below)

[Contents of subcommand file]
      +----- ";" Comment follows
[MUX]           ↓
VCH00 = OPENING.M1V
ACH00 = OPENING.SFA
SFDFILE = OPENING.SFD ; Opening
```

Fig. 4.2 Example Using a Subcommand File

(2) When writing the subcommand file, observe the following rules.

- Write one command per line. (One line can be no longer than 250 characters.)
- Specify one each of a video file, audio file and output file. (The order is not important.)
- Spaces and tabs may be inserted between keywords and parameters.
- Write comments after a semi-colon ";".

(3) Subcommand keyword commentary

In the example shown in Fig. 4.2 above, there are four command line options. Normally, these are the only four options that you need to remember. For further details, refer to Appendix A.

- (a) [MUX] : Indicates the start of the multiplexing block.
- (b) VCH00 : Specifies the MPEG1/Video input file name. If the extension is omitted, "M1V" will be assumed.
- (c) ACH00 : Specifies the audio input file name. If the extension is omitted, "SFA" will be assumed.
- (d) SFDFILE : Specifies the output file (Sofdec file) name.

Multiple files can then be created as a batch job by repeatedly writing the group of four keywords (a) through (d) above.

[Contents of subcommand file]

```
[MUX]
VCH00      = OPENING.M1V
ACH00      = OPENING.SFA
SFDFILE   = OPENING.SFD

[MUX]
VCH00      = ENDIGN    ; The extension may be omitted if it is "M1V"
ACH00      = ENDING    ; The extension may be omitted if it is "SFA"
SFDFILE   = ENDING    ; The extension may be omitted if "SFD" is
                      acceptable
```

Fig. 4.3 Example of Batch Processing of Multiple Files

5. Expanded Functions

This tool also offers these functions in addition to the above 4.

The above four functions are sufficient for creating a Sofdec file. The functions described here should be used as needed.

For detailed specifications, refer to Appendix A.

5.1. Multiple Inputs

This tool can multiplex multiple audio and video files. The tool can handle up to 32 audio files and 16 video files simultaneously. The stream IDs are managed internally, with the audio IDs ranging from C0h to DFh, and the video IDs ranging from E0h to EFh.

(1) Command line options

Specify "-A" and "-V" as many times as needed.

The stream IDs are assigned in order, starting from the file specified first.

(2) Subcommand keywords

Do not duplicate ACH00 through ACH31 and VCH00 through VCH15.

ACH00 corresponds to stream ID C0h, and ACH31 corresponds to DFh. Similarly, VCH00 corresponds to stream ID E0h, and VCH15 corresponds to EFh.

6. Q & A

Here are some frequently asked questions, and their answers.

Q1 : Is it possible to use long file names?

A1 : They can be used except in the following cases:

- (a) The length of the command line character string is restricted by the specifications of the operating system. (*1)

Therefore, if the names are long, it may not be possible to specify them unless they are in the 8+3 format "xxxxxx~1.mlv".

- (b) Names that include a period "," will not be processed correctly.

- *1) The command line character string length is normally 126 characters.

If you set "SHELL=COMMAND.COM /P /U:255 /L:1024" in CONFIG.SYS and restart the computer, the command line can be extended to 253 characters.

Q2 : Is it possible to use file names that include spaces?

A2 : Enclose such file names with double quotation marks ("). (This capability was added in Ver. 1.07.) If a file name does not include a space, it makes no difference whether you enclose it with double quotation marks ("") or not. (See the example below.)

Example) SFDMUX -A="VOICE TEST 01.SFA" -V="MOVIE.M1V" -S=TEST01.SFD

Q3 : I can't open a file, even though the file name is correct.

A3 : Make sure that you are not specifying an existing read-only file as the output file.

Q4 : If I specify a file without an extension, I get an error message.

A4 : Add a period "." at the end of the file name.

Normally, this tool automatically deduces the extension. Note that any file name that is displayed when an option error occurs is the file name exactly as the user input it.

Appendix A. List of Commands

This appendix lists the SFDMUX command line options and the subcommand keywords.

A.1 List of Commands

Table A.1 lists the command line options and the subcommand keywords.

Table A.1 List of Command Line Options

Function	Option	Subcommand	No.
Multiplexing options			1.0
Video input file specification	-V	VCH00 to VCH15	1.1
Audio input file specification	-A	ACH00 to ACH31	1.2
Private input file specification	-P1	PCH00	1.3
Output file specification	-S	SFDFILE	1.4
Command line-only commands			2.0
Language specification	-LANG	--	2.1
Subcommand file specification	-SUB	--	2.2
Subcommand-only commands			3.0
Definition block start	--	[MUX]	3.1

A.2 Command Specifications

The commands can be written in any order desired. Furthermore, no distinction is made between upper-case letters and lower-case letters.

Make sure that you do not specify the name of an existing read-only file as the output File.

Title	Option Name	Option	Keyboard	No
Option specification	Video input file specification	-V	VCH00 to VCH15	1.1

[Format] Command line: -V=filename
 Subcommand : VCH??=filename

[Input] filename : Input file name

[Function] This command specifies the video input file and the bit rate.

[Remarks]

- (a) This command can be specified as many times as needed in a command line or in the subcommand [MUX] block. In the case of a subcommand, the same keyword can only be used once.
- (b) When in a command line, the stream IDs are assigned in the order that the files are specified.

In the case of a subcommand, the specified channel (00 to 15) can be specified directly. The channel is converted internally into an actual stream ID (E0h to EFh).

- (c) Multiplexing is not possible if a file from a codec other than MPEG1/Video is specified.
- (d) This command corrects the time codes as it multiplexes.

It does not correct or otherwise process the input file itself.

After correction, the time codes are non-drop time codes that start from 00:00:00.

- (e) If an extension is not specified in "filename", the extension "M1V" is assumed. If specifying a file name that has no extension, add a period "." at the end of the file name.

Title	Option Name	Option	Keybord	No
Option specification	Audio input file specification	-A	ACH00 to ACH31	1.2

[Format] Command line: -A=filename
 Subcommand : ACH??=filename

[Input] filename : Input audio file name

[Function] This command specifies the input audio file name and bit rate.

[Remarks]

- (a) This command can be specified as many times as needed in a command line or in the subcommand [MUX] block. In the case of a subcommand, the same keyword can only be used once.
- (b) When in a command line, the stream IDs are assigned in the order that the files are specified.
 In the case of a subcommand, the specified channel (00 to 31) can be specified directly.
 The channel is converted internally into an actual stream ID (C0h to DFh).
- (c) As long as the bit rate is fixed, the codec does not matter.
- (d) If an extension is not specified in "filename", the extension "M1V" is assumed. If specifying a file name that has no extension, add a period "." at the end of the file name.

Title	Option Name	Option	Keybord	No
Option specification	Private input file specification	-P1	PCH00	1.3

[Format] Command line: -P1=filename
 Subcommand : PCH00=filename

[Input] filename : Private data file name

[Function] This command specifies a private data input file.

[Remarks]

- (a) This command can be specified as many times as needed in a command line or in the subcommand [MUX] block.
- (b) "BDh" is assigned as the stream ID.
- (c) The specified file is always placed at the beginning of the Sofdec file. Note that if this file contains a large amount of data, there may be a noticeable delay before movie playback begins.
- (d) Multiplexing is not possible if only private data is specified. At least one audio or video data file must be specified.

Title	Option Name	Option	Keybord	No
Option specification	Output file specification	-S	SFDFILE	1.4

[Format] Command line: -S=filename
 Subcommand : SFDFILE=filename

[Input] filename : Output file name

[Function] This command specifies the output file name.

[Remarks]

- (a) This command can be specified only once in a command line or in the subcommand [MUX] block.
- (b) At least one input audio file or video file is required.
- (c) If the extension has been omitted, the extension "SFD" is assumed.

Title	Option Name	Option	Keybord	No
Option specification	Language specification	-LANG	--	2.1

[Format] Command line: -LANG={E | J}
 Subcommand : None

[Input] E : English

J : Japanese

[Function] This command specifies which language messages will be displayed in.

[Remarks]

- (a) This command may be omitted. If omitted, the language mode of the console is used.
- (b) If this command is specified more than once, the last specification is valid.
- (c) This command must be specified at the beginning of the file.

Title	Option Name	Option	Keybord	No
Option specification	Subcommand file specification	-SUB	--	2.2

[Format] Command line: -SUB=filename
 Subcommand : None

[Input] filename : Subcommand file name

[Function] This command specifies the subcommand file.

[Remarks]

- (a) This command may be omitted.
- (b) If this command is specified more than once, the last specification is valid.

Option specification	Definition block start	--	[MUX]	3.1
----------------------	------------------------	----	-------	-----

[Format] Command line: None

 Subcommand : [MUX]

[Input] None

[Function] This keyword indicates the start of the multiplexing block.

[Remarks]

- (a) One block is started by this keyword and continues until either the next keyword is found, or the end of the file is found.
- (b) One block consists of one output file and at least one input file.
- (c) Multiple blocks may be written in one subcommand file.

Appendix B. List of Error Messages

This appendix lists the errors that are output by this tool. The messages appear in English and in Japanese.

The messages are displayed in Japanese only if the console mode is Japanese. In all other cases, the display is in English. The display language can be switched by using the "-LANG" option.

B.1 Command Analysis-Related Errors

These errors are generated during analysis of the command line or subcommand files.

Unknown codec.	-300
Cause	The codec specification is incorrect.
Remedy	Specify the correct codec name.
Illegal video codec.	-301
Cause	The video codec specification is incorrect.
Remedy	Specify the correct codec name.
Illegal audio codec.	-302
Cause	The audio codec specification is incorrect.
Remedy	Specify the correct codec name.
Bitrate value is not correct.	-304
Cause	Either the bit rate specification method or the value is incorrect.
Remedy	Specify the correct bit rate.
Illegal language.	-306
Cause	The language mode specification is incorrect.
Remedy	Specify the correct language mode.
Command parameter missing.	-307
Cause	The option format is incorrect.
Remedy	Check the option format and specify the correct parameters.
File name missing.	-308
Cause	No file name is specified in an input/output option.
Remedy	Specify a file name.
Too many input files.	-311
Cause	There are too many audio or video input files.
Remedy	Specify no more than 32 audio files and 16 video files.
Too many output files.	-312
Cause	Multiple output files are specified.
Remedy	Specify only one output file.

Cause	No file name is specified for the input and output files.
Remedy	Specify at least one input file name and one output file name.
Output file name missing.	-314
Cause	No output file name is specified.
Remedy	Specify one output file name.
Input file name missing.	-315
Cause	No input file name is specified.
Remedy	Specify at least one input file name.
Subcommand file name missing.	-316
Cause	No subcommand file name is specified.
Remedy	Specify the subcommand file name for the option parameter.
Cannot open subcommand file.	-317
Cause	The specified subcommand file either does not exist, or is locked.
Remedy	Make sure that the specified subcommand file exists. If it does, make sure that it is not being used by another application. If it is in use, close the file and then run this tool.
Description of subcommand is not correct.	-318
Cause	The subcommand format is incorrect.
Remedy	Check the subcommand format and then write it correctly.
Illegal keyword.	-319
Cause	There is an illegal keyword in the subcommand file.
Remedy	Write the subcommand file correctly.
Illegal option.	-320
Cause	An undefined option was specified.
Remedy	Specify the option correctly.
Too many video files.	-323
Cause	More than 16 video files are being input.
Remedy	Input no more than 16 video files.
Too many audio files.	-324
Cause	More than 32 audio files are being input.
Remedy	Input no more than 32 audio files.
Illegal parameter.	-325
Cause	A parameter includes characters that cannot be interpreted by this tool.
Remedy	Make sure that the parameter contains no control codes or other special characters.

B.2 Errors After the Start of Multiplexing

These errors are displayed during multiplexing processing.

If one of these errors is generated, multiplexing is not performed correctly. Eliminate the cause of the error and then re-execute the multiplexing process.

Illegal Sofdec/Audio file.	-202
Cause	Either the file is damaged, or it is not in the Sofdec/Audio format.
Remedy	Prepare a Sofdec/Audio file of the correct format and then re-execute the multiplexing process.
Not Sofdec/Audio.	-203
Cause	Either the file is damaged, or it is not a Sofdec/Audio file.
Remedy	Prepare a Sofdec/Audio file of the correct format and then re-execute the multiplexing process.
Not MPEG1/Video.	-204
Cause	Either the file is damaged, or it is not an MPEG1/Video file.
Remedy	Prepare a MPEG1/Video file of the correct format and then re-execute the multiplexing process.
Illegal pixel size.	-206
Cause	The video has a number of pixels in the horizontal or vertical direction (i.e., the display size) that is not suitable for the Sofdec library.
Remedy	Prepare video that has a multiple of 32 pixels in the horizontal direction and a multiple of 16 pixels in the vertical direction, such as 320×240 (224), and then re-execute the multiplexing process.
Illegal picture rate value.	-207
Cause	Either the video file is damaged, or it has a picture rate that is not stipulated in the MPEG1/Video standard.
Remedy	Confirm that the specified file is an MPEG1/Video file. If the file is damaged, encode it again. When encoding, check the parameters, prepare the correct MPEG1/Video format, and then re-execute the multiplexing process. Within Japan, the standard picture rate is 29.97fps.
Cannot get picture attributes.	-209
Cause	Either the video file is damaged, or the header is abnormal.
Remedy	Confirm that the specified file is an MPEG1/Video file. If the file is damaged, encode it again. If this does not resolve the problem, contact Sega DTS.

Picture size is too big.	-210
Cause	The I picture of the video is too large. It is possible that the number of pixels is too large (640 × 480, for example), or that the bit rate is too high.
Remedy	Reduce the number of pixels, lower the bit rate, and then re-execute the multiplexing process. If you do not want to change the encoding conditions, use the -ncps option of this tool, note the message that is displayed, and then consult with Sega DTS.
Cannot decode MPEG1/Video header.	-212
Cause	Either the video file is damaged, or the header is abnormal.
Remedy	Confirm that the specified file is an MPEG1/Video file. If the file is damaged, encode it again. If the I picture is too large, lower the bit rate or reduce the number of pixels. (Refer to error code -210.) If this does not resolve the problem, contact Sega DTS.
Cannot find MPEG1/Video header.	-213
Cause	Either the video file is damaged, the header is abnormal, or the I picture is too large.
Remedy	Confirm that the specified file is an MPEG1/Video file. If the file is damaged, encode it again. If the I picture is too large, lower the bit rate or reduce the number of pixels. (Refer to error code -210.) If this does not resolve the problem, contact Sega DTS.
Cannot open output file.	-214
Cause	Either the specified output file already exists and is write protected, or else there is not any free space on the disk.
Remedy	Check the file attributes using the "Explore" and "attrib" commands. If the read-only attribute is set, cancel that attribute. Also make sure that the file is not in use by another application; if the file is in use, close the file and then re-execute the multiplexing process. If neither of these steps resolves the problem, it is possible that there is not enough free space on the disk. Delete any unneeded files in order to create more free space, and then re-execute the multiplexing process.
Cannot multiplex.	-215
Cause	An error occurred during multiplexing, and processing did not end normally.
Remedy	Eliminate the causes of all other errors and then re-execute the multiplexing process.

B.3 Initialization and Termination Errors

These errors appear while the tool is performing its initial preparations and while it is quitting.

If any of these errors are generated, it is possible that multiplexing cannot be performed, or, even if multiplexing appears to be possible, that multiplexing is not performed correctly. Eliminate the cause of the error and then re-execute the multiplexing process.

Cannot allocate memory.	-102
Cause	There is not enough memory available for this tool to run.
Remedy	In the Windows system properties, make sure that there is enough free space in the swap destination drive. If any other applications are running, close those applications, wait until the access indicator on the hard drive is off, and then try again.
Cannot write file.	-104
Cause	Either the specified file is write protected, or else it is locked.
Remedy	Check the file attributes using the "Explore" and "attrib" commands. If the read-only attribute is set, cancel that attribute. Also make sure that the file is not in use by another application; if the file is in use, close the file and then try again.
Cannot open file.	-105
Cause	It is possible that the file does not exist.
Remedy	Make sure that the file exists.
Cannot close file.	-106
Cause	It is possible that there is not enough free space on the disk.
Remedy	Delete any unneeded files in order to create more free space, and then try again.
Cannot seek file.	-107
Cause	There is some sort of file access problem.
Remedy	Quit this tool, restart Windows, and then try again.
Cannot read file.	-108
Cause	There is some sort of file access problem.
Remedy	Quit this tool, restart Windows, and then try again.

B.4 Other Errors

If any errors other than those described above appear, contact Sega DTS.