

General Notice

When using this document, keep the following in mind:

1. This document is confidential. By accepting this document you acknowledge that you are bound by the terms set forth in the non-disclosure and confidentiality agreement signed separately and /in the possession of SEGA. If you have not signed such a non-disclosure agreement, please contact SEGA immediately and return this document to SEGA.
2. This document may include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new versions of the document. SEGA may make improvements and/or changes in the product(s) and/or the program(s) described in this document at any time.
3. No one is permitted to reproduce or duplicate, in any form, the whole or part of this document without SEGA'S written permission. Request for copies of this document and for technical information about SEGA products must be made to your authorized SEGA Technical Services representative.
4. No license is granted by implication or otherwise under any patents, copyrights, trademarks, or other intellectual property rights of SEGA Enterprises, Ltd., SEGA of America, Inc., or any third party.
5. Software, circuitry, and other examples described herein are meant merely to indicate the characteristics and performance of SEGA'S products. SEGA assumes no responsibility for any intellectual property claims or other problems that may result from applications based on the examples describe herein.
6. It is possible that this document may contain reference to, or information about, SEGA products (development hardware/software) or services that are not provided in countries other than Japan. Such references/information must not be construed to mean that SEGA intends to provide such SEGA products or services in countries other than Japan. Any reference of a SEGA licensed product/program in this document is not intended to state or simply that you can use only SEGA'S licensed products/programs. Any functionally equivalent hardware/software can be used instead.
7. SEGA will not be held responsible for any damage to the user that may result from accidents or any other reasons during operation of the user'S equipment, or programs according to this document.

NOTE: A reader's comment/correction form is provided with this document. Please address comments to :

SEGA of America, Inc., Developer Technical Support (att. Evelyn Merritt)
150 Shoreline Drive, Redwood City, CA 94065

SEGA may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.



SEGA OF AMERICA, INC.
Consumer Products Division

SEGA Confidential

Program Library User's Guide 1 CD Library

Doc. # ST-136-R2-093094

READER CORRECTION/COMMENT SHEET

Keep us updated!

If you should come across any incorrect or outdated information while reading through the attached document, or come up with any questions or comments, please let us know so that we can make the required changes in subsequent revisions. Simply fill out all information below and return this form to the Developer Technical Support Manager at the address below. Please make more copies of this form if more space is needed. Thank you.

General Information:

Your Name _____ Phone _____

Document number ST-136-R2-093094 Date _____

Document name Program Library User's Guide 1 CD Library

Corrections:

Chpt.	pg. #	Correction

Questions/comments: _____

Where to send your corrections:

Fax: (415) 802-1440
Attn: Sr. Coordinator,
Technical Publications Group

Mail: SEGA OF AMERICA
Attn: Sr. Coordinator,
Technical Publications Group
130 Shoreline Dr.
Redwood City, CA 94065

REFERENCES

In translating/creating this document, certain technical words and/or phrases were interpreted with the assistance of the technical literature listed below.

1. *Dictionary of Science and Engineering, 350,000 words, 3rd Edition*
Inter Press
Tokyo, Japan
1990
2. *Computer Dictionary*
Kyoritsu Publishing Co., LTD.
Tokyo, Japan
1978
3. *IBM Dictionary of Computing*
McGraw-Hill, Inc.
New York, New York
1994

(This page was blank in the original Japanese document.)

SEGA Confidential

CONTENTS

1.0	Outline	1
1.1	Features	1
1.2	Summary of Functions	2
1.3	Module Configuration	4
2.0	Basics	5
2.1	Glossary	5
2.2	Notation	6
2.3	Name Restrictions	6
2.4	Access Macros	6
3.0	Directory Operations	7
3.1	Initialization	7
3.2	File Identifiers	8
3.3	Sub-Directory Operations	9
3.4	Conversion Between File Names and File Identifiers	11
3.5	CD Block File System	12
4.0	File Access	13
4.1	Access Models	13
4.2	Access Pointers	13
4.3	Parameters Controlled for Each File	14
5.0	Access Modes	15
5.1	Return-Upon-Completion Access	15
5.2	Immediate-Return Access	15
6.0	Other Functions	19
6.1	Development Support Functions	19
6.2	Error Processing Functions	21
6.3	Multiprocessing	21
7.0	Data Specifications	23
7.1	Basic Data	24
7.2	Constants	25
7.3	Data Types	29

8.0	Function Specifications	32
8.1	Directory Control	33
8.2	File Operations	35
8.3	Return-Upon-Completion Read	37
8.4	Immediate-Return Read	38
8.5	Read Parameter Settings	41
8.6	Other	42
Appendix A Utilization of Development Support Functions		43
A.1	Procedure for Using Memory Files	43
A.2	Procedure for Using DOS Files	43
A.3	Precautions	44
Appendix B Error Processing Methods		45
Appendix C		48
C.1	Additional Explanation	48
C.2	Changes from the Previous Version	48

(This page was blank in the original Japanese document.)

SEGA Confidential

1.0 Outline

Below is an explanation of the libraries for accessing files on a CD.

1.1 Features

This library has the following features.

Compatible CD ROMs

- Capable of accessing ISO9660 level files.
- This library does not cover access that utilizes CD ROM XA sub-header information.

Data Buffering

- Access is performed that assumes a CD block buffer control mechanism.
- In addition to simply reading files, pre-reads using the buffer are possible.

File Identifiers

- Access is based on file identifiers (order in directory).
- Eliminate a drop in speed caused by searching directories.
- Access by file name is possible by using a function that converts from the file name to a file identifier.

Development Support Functions

- Memory files and DOS files can be accessed as a development support function.
- In the case of small amounts of data, files on a CD can be interchanged with memory files.
- DOS files on an IBM PC can be accessed in the same way as memory files via the SCSI interface. Even data too large to load into SIMM can be replaced by CD files.
- Two types of libraries are provided: one for building into the product and one that includes development support functions.

1.2 Summary of Functions

The functions of the file system library are summarized below, and are categorized into the following six types.

Directory Operations

Library initialization, directory information reads, current directory settings and other functions are provided.

Function	Function performed
GFS_Init	Initializes the library and mounts CDs
GFS_LoadDir	Reads directory information
GFS_SetDir	Sets the current directory
GFS_NameToId	Converts a file name to a file identifier
GFS_IdToName	Converts a file identifier to a file name

File Operations

Opens, closes, seeks and performs the other common operations on files listed below.

Function	Action
GFS_Open	Opens a file
GFS_Close	Closes a file
GFS_Seek	Moves the access pointer
GFS_Tell	Gets an access pointer
GFS_IsEof	Checks if an access pointer is at the end of a file
GFS_ByteToSct	Converts the unit from byte to sector
GFS_GetFileSize	Gets the file size
GFS_GetFileInfo	Gets file information

Return-Upon-Completion Read

Reads data from files. Control does not return from the function until the reading of data is complete.

Function	Action
GFS_Fread	Reads data from opened files
GFS_Load	Specifies a file identifier and then reads data



Immediate-Return Read

Reads data from files by means of a request function and a server function. The request issued by the request function is processed by the server function one processing unit at a time. The server function must be recalled repeatedly. By inserting application processing into the server function call loop, execution of the application can be continued until completion of the data read.

Function	Action
GFS_NwFread	Issues a data read request
GFS_NwCdRead	Issues a read request to the CD buffer
GFS_NwIsComplete	Checks if read processing is complete
GFS_NwStop	Stops read processing
GFS_NwGetStat	Gets the access status
GFS_NwExecOne	A server function for one file
GFS_NwExecServer	A server function for multiple files

Read Parameter Setting

Sets the various parameters for the return-upon-completion and immediate-return functions. Determines how the CD buffer is to be used, the transfer mode (DMA, CPU, etc.) and the transfer unit.

Function	Function performed
GFS_SetGmode	Sets the mode for fetching from the CD buffer
GFS_SetTmode	Sets the transfer mode
GFS_SetReadPara	Sets the unit for reading to the CD buffer
GFS_SetTransPara	Sets the unit for transferring from the CD buffer

Other

CD pickup control, registration of error processing functions, and getting the error status are provided. The error processing function is called when an error occurs.

Function	Function performed
GFS_CdMovePickup	Moves the CD pickup
GFS_SetErrFunc	Registers the error processing function
GFS_GetErrStat	Gets the error status

1.3 Module Configuration

The positioning of this library with respect to the hardware and other software is shown below in Figure 1.1. The area enclosed in the dotted line is the module included in the library for debugging.

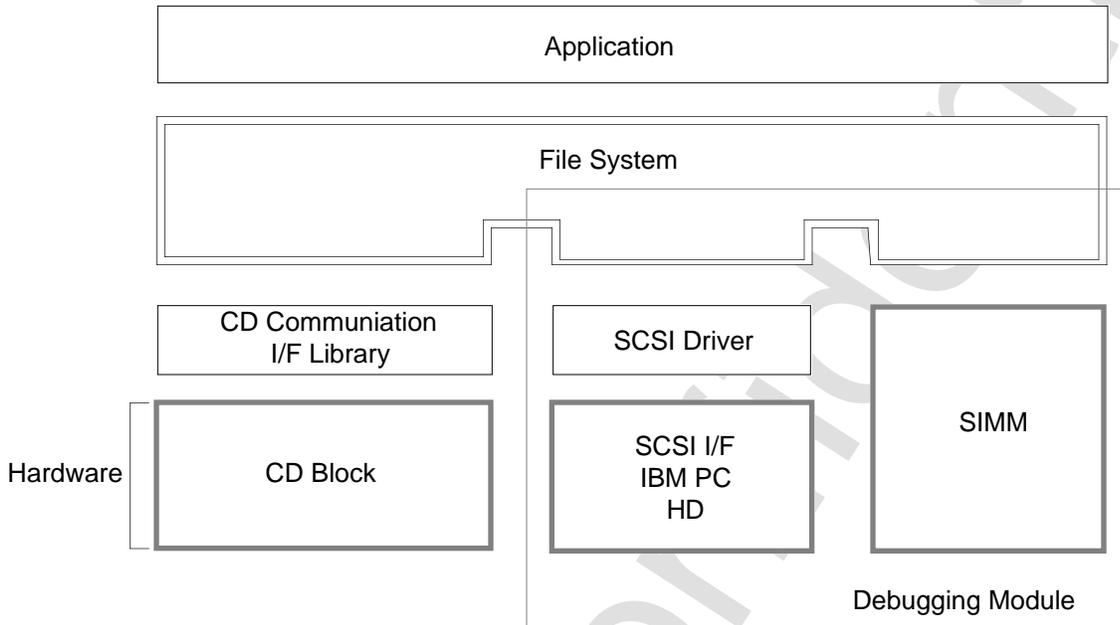


Figure 1.1 Module Configuration

In order to use this library, it is necessary to link the following libraries at the same time.

- SHCNPIC.LIB Library not compatible with position-independent code for SH7600
- SEGA_CDC.LIB CD communications interface library
- SEGA_DMA.LIB DMA library
- SEGA_CSH.LIB Cache library



2.0 Basics

2.1 Glossary

The terms used to explain the file system library are defined in Table 2.1.

Table 2.1 Glossary

Term	Meaning
CD buffer	A buffer that stores data read from the CD in sector units. It has a 200-sector capacity.
DOS file	Files on an IBM PC that can be accessed via a SCSI interface. These can be used in a debugger library.
Memory file	A file located on SIMM. These files can be used in the debugger library.
Access pointer	Position at which a file is accessed (unit: sector).
Current directory	The directory referred to when opening files.
Debug file	General designation for DOS files and memory files.
Buffer partition	One part of a CD buffer divided up into several logical parts. One buffer partition is dedicated to each file that is opened.
File identifier	A sequential number in a directory for identifying files. The values used range from 0 to (number of directory records - 1). Where, 0 indicates the current directory and 1 indicates the parent directory.
Frame address (FAD)	Number continuously assigned in frame units assuming the absolute time on the CD is 00:00:00. This number has a 1-to-1 correspondence to the absolute time. The CD is accessed using the frame address as a key, not the absolute time.
Main process	The series of processes that is begun when the CPU is reset. This term refers to interrupt processing.
Interrupt process	Processing that is started by an interrupt. This term refers to the main process.

2.2 Notation

The notation used in explaining the file system is explained below.

Grouping of names

"ABC_-" indicates several names beginning with ABC_. For example, ABC_X, ABC_Y and ABC_Z.

Symbol Specification

"!MMM/SSS" indicates the symbol SSS defined by MMM. It is also a notation used with E7000 commands.

Hexadecimal Notation

Numbers with an "H" affixed to them at the end are hexadecimal numbers.

2.3 Name Restrictions

In file system libraries, the following names are used for functions, variables, types, and macros.

Functions and variables	GF- or gf-
Type s	Gf-
Macros	GF-

In the applications that use these libraries, be careful not to use designations that conflict with these naming conventions.

2.4 Access Macros

In CD libraries that include file system libraries, members of the structure are referenced using a structure called an access macro. Access macros are capable of getting and setting the values of members. Using access macros has the following advantages.

- Member access format is uniform.
- Parts accessing specific structure members can be easily extracted.



3.0 Directory Operations

3.1 Initialization

Before using this library, GFS_Init must be executed. GFS_Init performs the following processing:

- Initialization of library work area
- Mount processing

Initialization

Sets the work area used by the library and initializes it. The application must provide the work area and the directory information storage area.

Since the size of the area changes with the number of files opened at the same time, it should be obtained using the following macro. The statement open_max is the maximum number of files that can be opened at once.

```
GFS_WORK_SIZE(open_max)
```

Mount Processing

The root directory is read from the CD ROM and this is made the current directory. It also initializes the CD block and erases all the sector data in the CD buffer. Since the file system only holds the top address of the directory information storage area, the application must not change the contents of the area.

The directory information control structure is initialized and GFS_Init called as shown below.

```
#define OPEN_MAX      20          /*maximum number of files to be opened at
                                the same time                    */
#define MAX_DIR       10          /*maximum number of directories      */

Uint32 work[GFS_WORK_SIZE(OPEN_MAX)/4]; /*library work area                */
GfsDirTbl dirtbl; /*directory information control structure */
GfsDirId dir[MAX_DIR]; /*directory information storage area */

GFS_DIRTBL_TYPE(&dirtbl) = GFS_DIR_ID; /*directory information storage area type */
GFS_DIRTBL_NDIR(&dirtbl) = MAX_DIR; /*maximum number of elements in directory
                                information
                                */
/*storage area                    */
GFS_DIRTBL_DIRID(&dirtbl) = dir; /*address of directory information storage*/
/*area                             */

GFS_Init(OPEN_MAX, work, &dirtbl);
```

GFS_Init must be called again when the CD ROM is changed.

3.2 File Identifiers

In this library, files that are accessed are specified by file identifiers. If the file name is used to access a file, then the file name is converted to a file identifier. The file identifier is valid for the current directory.

Example: Accessing FILE2.DAT below.

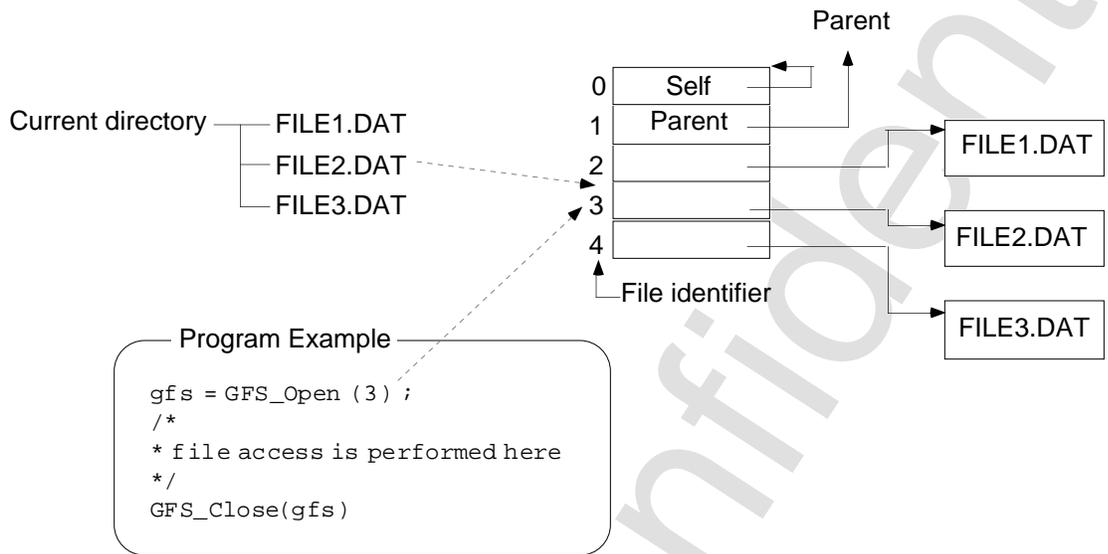


Figure 3.1 Access Using File Identifiers



3.3 Sub-Directory Operations

In order to access files in a sub-directory, it is necessary to set the current directory information by calling the following functions.

- Read directory information (GFS_LoadDir)
- Set current directory (GFS_SetDir)

A method in which the directory information is read in advance and CD ROM access is lost when the file is opened is allowed.

Read Directory Information (GFS_LoadDir)

This specifies the sub-directory file and reads and saves the directory information. The following two types of directory information save areas can be selected.

(a) GFS_DIR_ID

- Does not save the file name. Files can only be accessed by means of a file identifier.

(b) GFS_DIR_NAME

- Saves the file name and therefore an area larger than (a) above is required.
- Access by file name is allowed.

Current Directory Setting (GFS_SetDir)

The directory information area read by GFS_LoadDir is used as the current directory.

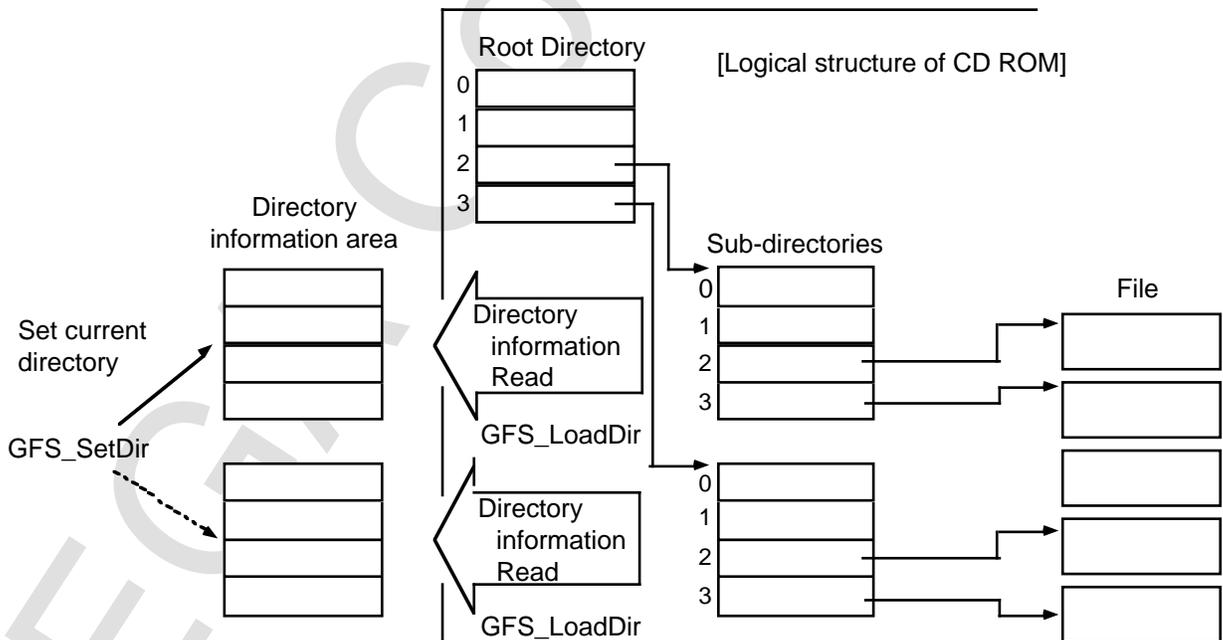
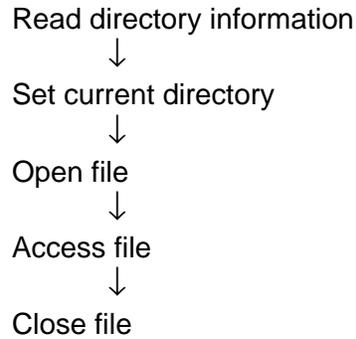


Figure 3.2 Setting Directory Information

The following procedure must be followed to access files in a sub-directory.



Two examples of this procedure are shown below.

Example: Accessing a file in a directory other than the root directory
An example of a procedure program for accessing a file in a sub-directory is shown below. Here, the file to be accessed is in the directory specified by `dir_fid` in the current directory.

```
#define MAX_DIR      10                /*maximum number of directories      */

GfsDirTbl dirtbl;                      /*Directory information storage area  */
GfsDirId dirid[MAX_DIR];              /*Directory information storage area  */
Sint32 dir_fid;                       /*enters directory file identifier    */
Sint32 fid;                           /*enters accessed file identifier     */
GfsHn gfs;                            /*file handler of accessed file      */

GFS_DIRTBL_TYPE(&dirtbl) = GFS_DIR_ID;
GFS_DIRTBL_NDIR(&dirtbl) = MAX_DIR;
GFS_DIRTBL_DIRID(&dirtbl) = dirid;
GFS_LoadDir(dir_fid, &dirtbl);        /*reads directory information        */

GFS_SetDir(&dirtbl);                  /*sets current directory              */

/*sets identifier of file accessed in fid */
gfs = GFS_Open(fid);
/*
*file access performed here
*/
GFS_Close(gfs)
```



Example: Simultaneous access of multiple files in different directories

In order to access files in different directories, the target files must be opened while switching the current directory. An example is shown in which two files in the two sub-directories directly below the current directory are accessed simultaneously. This file identifiers of the two sub-directories with the files to be accessed are respectively specified by `dir_fid1` and `dir_fid2`.

```
#define MAX_DIR      10                /*maximum number of directories */
GfsDirTbl curdir;    /*current directory at this point */
GfsDirTbl dirtbl1, dirtbl2;          /*control area for directory information*/
GfsDirId diridl[MAX_DIR];            /*storage area for directory information*/
GfsDirId dirid2[MAX_DIR];            /*storage area for directory information*/
Sint32 dir_fid2, dir_fid2;          /*enters identifiers of directory files */
Sint32 fid1, fid2;                  /*enters identifiers of access files */
GfsHn gfs1, gfs2                    /*file handlers of access files */

/*loads directory information of current directory dir_fid1 */
GFS_DIRTBL_TYPE(&dirtbl1) = GFS_DIR_ID;
GFS_DIRTBL_NDIR(&dirtbl1) = MAX_DIR;
GFS_DIRTBL_DIRID(&dirtbl1) = diridl;
GFS_LoadDir(dir_gfs1, &dirtbl1);

/*loads directory information of current directory dir_fid2 */
GFS_DIRTBL_TYPE(&dirtbl2) = GFS_DIR_ID;
GFS_DIRTBL_NDIR(&dirtbl2) = MAX_DIR;
GFS_DIRTBL_DIRID(&dirtbl2) = dirid2;
GFS_LoadDir(dir_gfs2, &dirtbl2);

/*opens the file fid1 of the directory dir_fid1 */
GFS_SetDir(&dirtbl1);
gfs1 = GFS_Open(fid1);

/*opens the file fid2 of the directory dir_fid2 */
GFS_SetDir(&dirtbl2);
gfs2 = GFS_Open(fid2);
/*
*file access is performed here
*/
GFS_Close(gfs1);
GFS_Close(gfs2);
```

3.4 Conversion Between File Names and File Identifiers

When directory information containing file names is set to the current directory, functions for converting between file names and file identifiers can be used.

If directory information not containing a file name is set to the current directory, an error results if these functions are called. An example defining a function that uses this function to open a file by its file name is shown below.

```
Example: /*opens file specified by file name */
GfsHn OpenByName(UINT8 *fname)
{
    Sint32 fid = GFS_NameToId(fname);

    if (fid < 0) {
        return NULL;
    }
    return GFS_Open(fid);
}
```

3.5 CD Block File System

Directories can be controlled using the CD block file system (file system built into the CD block; CDBFS below).

The processes for initialization, reading directory information and setting the current directory using the CDBFS are shown below.

Initialization

In order to utilize the functions of the CDBFS, NULL must be specified for the pointer to the directory control structure and GFS_Init must be called. Upon completion of processing by GFS_Init, the root directory is set by the CDBFS.

Reading Directory Information

In order to read sub-directory information, NULL is specified for the pointer to the directory control structure and GFS_LoadDir is called to indicate that the storage destination of the directory information is in the CD block.

Setting Current Directory

In order to set the directory information set in the CD block to the current directory, NULL is specified for the pointer to the directory control structure and GFS_SetDir is called.

Even if settings that use the CDBFS are performed, directory control can be partially performed with this library. In that case, always be aware of which directory control function being utilized. The advantages and disadvantages of using the CDBFS are listed in Table 3.1.

Table 3.1 Advantages and Disadvantages of Using CDBFS

Advantages	Disadvantages
· Uses small amount of host memory.	· The CD ROM is accessed each time a file in a different directory is accessed.
	· The amount of CD buffer that can be used by applications is reduced by one sector.
	· File names cannot be used.

The functions GFS_Init and GFS_LoadDir, which read directory information, return the number of directories read as the function value. When the CDBFS is used, that number becomes the number of directories the CDBFS is holding.

An error results if a file name is used when the current directory of the CDBFS is set.



4.0 File Access

4.1 Access Models

A diagram of a file access model is shown in Figure 4.1.

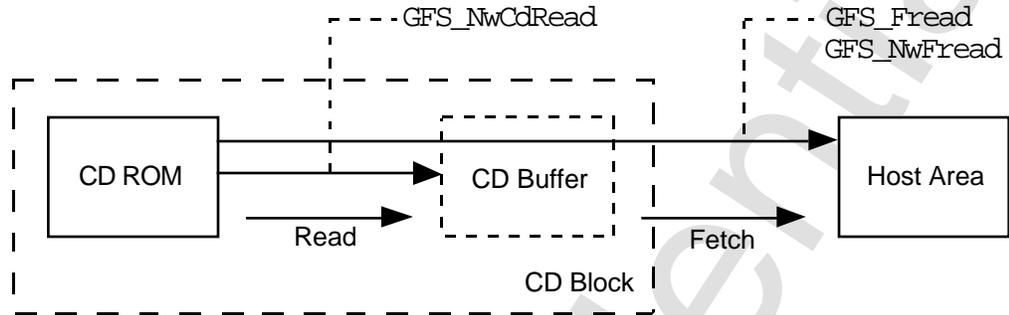


Figure 4.1 Access Function Model

Transferring data from the CD ROM to the CD buffer is called “reading”, and transferring data from the CD buffer to the host area is called “fetching”.

By utilizing `GFS_Fread` and `GFS_NwFread`, an application can transfer data from the CD ROM without being aware of read processing. To control read processing from the application, use `GFS_NwCdRead`.

4.2 Access Pointer

Since the access pointer is updated by reading, it moves in sector units. Movement of the access pointer when the following expression is executed to read 5000 bytes to the host area buffer is shown in Figure 4.2. After execution, the access pointer moves from AP1 to AP2.

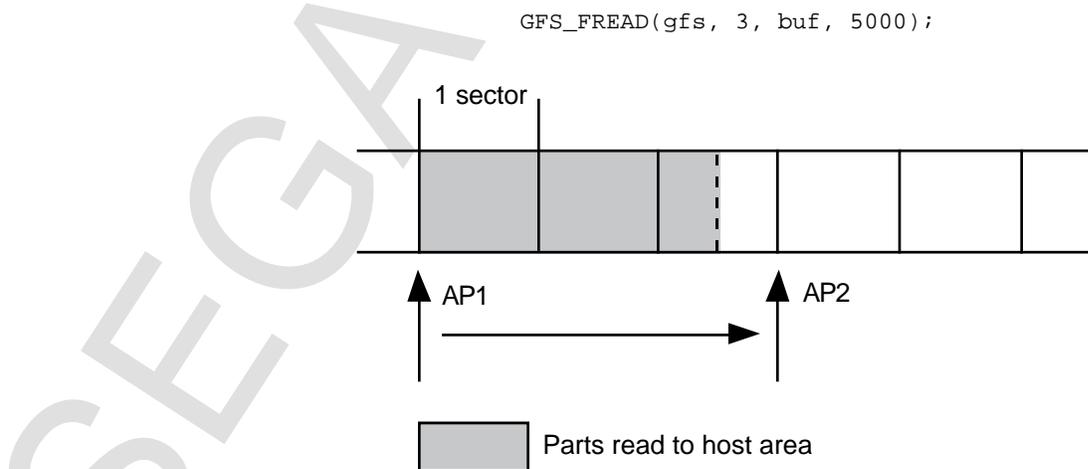


Figure 4.2 Movement of access pointer

Sizes of sectors for each type of file are shown in Table 4.1

Table 4.1 Sector Lengths for Each Type of File

File type	Sector length (bytes)
CD ROM mode 1	2048
CD ROM mode 2 form 1 only	2048
CD ROM mode 2 form 2 only	2324
Includes CD ROM mode 2	Undefined
DOS file	2048
Memory file	2048

While reading is performed in sector units, fetching is performed in 4-byte units.

4.3 Parameters Controlled for Each File

Of the parameters controlled by the library for each file opened, an application can change up to five. These parameters are shown in Table 4.2.

Table 4.2 Parameters for Each File

Parameter	Description	Function changed	Initial value
Access pointer	Offset of file at which reading is started (unit: sector)	GFS_Seek	0
Fetch mode	Specifies whether sector data in CD buffer is erased or left after fetching	GFS_SetGmode	GFS_GMODE_ERASE
Transfer mode	Specifies device that performs fetching	GFS_SetTmode	GFS_TMODE_CPU
Read parameter	Maximum number of sectors transferred in one read	GFS_SetReadPara	GFS_RPARA_DFL
Fetch parameter	Number of sectors transferred in one fetch	GFS_SetTransPara	1

The opened file occupies one filter, which is a CD block resource, and one buffer partition at a time.



5.0 Access Modes

The two access modes provided by this library are described below.

- Return-upon-completion access
Control is not returned to the application until completion of access.
- Immediate-return access
Control is returned as soon as an access request is received.

5.1 Return-Upon-Completion Access

Return-upon-completion access is similar to the file access function in the standard C library. An example return-upon-completion access program is shown below.

Example:

```
#define BUF_SIZE 2048

GfsHn gfs; /*file handler */
Sint32 fid; /*file identifier */
Sint32 nsct = 1; /*number of read sectors */
Uint32 buf[BUF_SIZE] /*read area */

gfs = GFS_Open(fid); /*opens file */

GFS_Fread(gfs, nsct, buf, BUF_SIZE); /*nsct sectors read to buf */

GFS_Close(gfs); /*closes file */
```

5.2 Immediate-Return Access

Immediate-return access is performed by using a request function and a server function. The request function executes only request acceptance and immediately returns. Actual access is performed by repeatedly calling the server function while monitoring the completion status. The application can also be processed in the call loop of the server function.

A file handle that has issued a request cannot issue another request until processing of the first one is completed.

Immediate-Return Access for Single Files

The server function for single file access is GFS_NwExecOne. An example of a program that accesses one file using immediate-return access is shown below. In this example, the request function is GFS_NwFread.

Example:

```
#define BUF_SIZE 2048*2

GfsHn gfs; /*file handler */
Sint32 nsct = 2; /*number of read sectors */
Sint32 stat; /*server status */
Uint32 buf[BUF_SIZE/4] /*read area */

gfs = GFS_Open(fid); /*opens file */
/*request function */
GFS_NwFread(gfs, nsct, buf, BUF_SIZE); /*reads nsct sectors into buf */
/*returns control immediately */

for (;;) {
    /*server function */
    stat = GFS_NwExecOne(GFS); /*executes read */
    if (stat == GFS_SVR_COMPLETED) { /*read complete? */
        break;
    }
    user(); /*optional user process */
}

GFS_Close(gfs); /*closes file */
```



Immediate-ReturnAccess for Multiple Files

The server function for continuous access of multiple files is GFS_NwExecServer. The request function is used in common with access of single files.

The application issues a request for access of multiple files. Following this, access is executed sequentially by periodically passing control to the server. Access is performed one at a time in the order of the requests.

Here is an example of a program that performs user processing while reading three files.

Example:

```
/*number of sectors read from each file */
#define NSCT1 1
#define NSCT2 2
#define NSCT3 3
/*size of data storage area of each file (unit: byte) */
#define BSIZE1 2048*NSCT1
#define BSIZE2 2048*NSCT2
#define BSIZE3 2048*NSCT3

Sint32 fid1, fid2, fid3; /*file identifier of each file */
GfsHn gfs1, gfs2, gfs3; /*file handle of each file */
Uint32 buf1[BSIZE1/4]; /*data storage area of each file */
Uint32 buf2[BSIZE2/4];
Uint32 buf3[BSIZE3/4];
GfsHn now_gfs; /*file handle during access */
Sint32 stat; /*server status */

gfs1 = GFS_Open(fid1); /*opens file */
gfs2 = GFS_Open(fid2);
gfs3 = GFS_open(fid3);
GFS_NwFread(gfs1, NSCT1, buf1, BSIZE1); /*starts read operation */
GFS_NwFread(gfs2, NSCT2, buf2, BSIZE2);
GFS_NwFread(gfs3, NSCT3, buf3, BSIZE3);
for (;;) {
    stat = GFS_NwExecServer(&now_gfs); /*executes read */
    if (stat == GFS_SVR_COMPLETED) { /*is there work to execute? */
        break;
    }
    user(); /*optional user process */
}
GFS_Close(gfs1);
GFS_Close(gfs2);
GFS_Close(gfs3);
```

Pre-loading to CD Buffer

If pre-reading to the CD buffer is specified by GFS_NwCdRead when a large file is continuously loaded a little at a time, then full advantage can be taken of the read speed of the CD.

In the program example shown below, a 1000 sector file is continuously read 10 sectors at a time. GFS_NwCdRead specifies to look ahead 1000 sector pre-reads, and therefore the 1000 sectors of the target file are continuously played and stored in the buffer. Since data is fetched from the buffer at the same time this processing is performed, the buffer does not get full and playback is not interrupted.

If this processing is performed without pre-reading, then playback of the CD is interrupted every 10 sectors and time is wasted.

An example of a program that pre-reads to the CD buffer is shown.

Example:

```
#define SECT_SIZE    2048
#define FILE_SIZE   10000*SECT_SIZE
#define RD_UNIT     10

Uint8 *rd_bp, *proc_bp;           /*read buffer and processing buffer */
Uint32 buf1[RD_UNIT*SECT_SIZE/4]; /*data storage area 1 */
Uint32 buf2[RD_UNIT*SECT_SIZE/4]; /*data storage area 2 */
GfsHn gfs;
Sint32 i, stat, nbyte;

gfs = GFS_Open(fid);
GFS_NwCdRead(gfs, FILE_SIZE);     /*pre-read specification for CD buffer */
GFS_SetTransPara(gfs, RD_UNIT);   /*maximum RD_UNIT sector fetched once */
for (i = 0; i < FILE_SIZE / RD_UNIT; ++i) {
    /*read and processing buffer settings */
    if (i & 1) {
        rd_bp = buf1;
        proc_bp = buf2;
    } else {
        rd_bp = buf2;
        proc_bp = buf1;
    }
    /*executes fetch from CD buffer */
    GFS_NwFread(gfs, RD_UNIT, rd_bp, RD_UNIT * SECT_SIZE);
    do {
        if (i > 0) {
            user_process(proc_bp); /*processing of read data */
        } else {
            user_process0();       /*processing before data is read */
        }
        GFS_NwExecOne(gfs);
        GFS_NwGetStat(gfs, &stat, &nbyte);
    } while (nbyte < RD_UNIT * SECT_SIZE);
}
```



6.0 Other Functions

6.1 Development Support Functions

The file system provides a function to access memory files and DOS files to support debugging. This function makes it possible to access files that have not been prepared on the CD ROM yet or files that have been changed since the CD ROM was made in the same way files on the CD ROM are accessed. However, this function cannot be used together with the CDBFS.

The mechanism that facilitates access of debugging files in the same way as files on the CD ROM lies in the directory information read process.

When directory information is read from the CD ROM, directory information is also read from the debugger file. The information in the debugger file is processed either in place of or in addition to the information in the CD ROM file.

After reading directory information from a directory, the following processing is performed.

1) Substitution

The directory information from a debugger file of the same name as the CD ROM file is set to the directory information area in place of the corresponding CD ROM file.

2) Addition

A debugger file for which substitution was not performed is added to the directory information storage area.

In the substitution of the debugger file, the memory file takes precedence over the DOS file. An example in which these processes are performed is shown below. Figure 6.1 shows the file configuration.

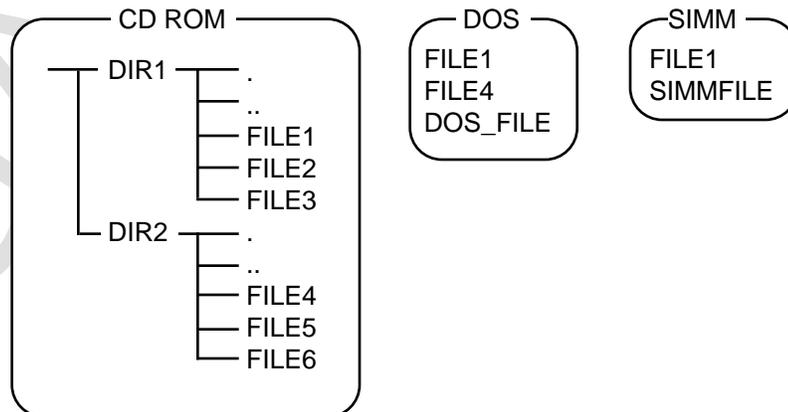


Figure 6.1 File Configuration Example

The results of obtaining the directory information of DIR1 with this file configuration are shown in Table 6.1 and the results of obtaining the directory information of DIR2 are shown in Table 6.2

Table 6.1 Directory Information of DIR1

Directory information obtained	CD ROM	DOS	SIMM
.	.		
..	..		
FILE1	FILE1	FILE1	FILE1
FILE2	FILE2		
FILE3	FILE3		
FILE4		FILE4	
DOS_FILE		DOS_FILE	
SIMMFILE			SIMMFILE

The files with lines through them indicate that they have been replaced.

Table 6.1 Directory Information of DIR2

Directory information obtained	CD ROM	DOS	SIMM
.	.		
..	..		
FILE4	FILE4	FILE4	
FILE5	FILE5		
FILE6	FILE6		
FILE1		FILE1	FILE1
DOS_FILE		DOS_FILE	
SIMMFILE			SIMMFILE

The files with lines through them indicate that they have been replaced.

As shown in the examples above, substitution and addition by a debugger file can be performed on all directories of a CD ROM.



6.2 Error Processing Functions

By registering error processing functions, it is possible to specify that an error processing function be called in the event an error should occur during execution of a library function. Error processing functions are not registered in the initial state.

When an error does occur, the error processing function is executed using the following call format.

```
void *(func)(void *obj, Sint32 err); /*error processing function      */
void *obj;                          /*pointer to registered object  */
Sint32 err_code;                     /*generated error code         */

(*func)(obj, err_code);              /*calls error processing function */
```

When control returns from the error processing function, the library function returns control to the application using the error code as the function value.

6.3 Multiple Processing

This is the processing required when the functions of this library are used (multiple processing) for both the main processing and the interrupt processing.

When an example is made to execute the functions of this library simultaneously for main processing and interrupt processing, the function value called last becomes error code GFS_ERR_BUSY. In that case, the following measures must be taken depending on whether this occurs during main processing or interrupt processing.

During Main Processing

Wait until the called function stops returning GFS_ERR_BUSY or postpone calling of this library function until the next opportunity.

During Interrupt Processing

Postpone calling of this library function until the next opportunity.

(This page was blank in the original Japanese document.)

SEGA Confidential



7.0 Data Specifications

A list of file system data is shown in Table 7.1.

Table 7.1 Data Table

Data	Data Name	No.
Basic data		1.0
Constants		2.0
File attribute	GFS_ATR_ -	2.1
Access status	GFS_NWSTAT_ -	2.2
Access Server status	GFS_SVR_ -	2.3
Seek mode	GFS_SEEK_ -	2.4
Get (fetch) mode	GFS_GMODE_ -	2.5
Transfer mode	GFS_TRANS_ -	2.6
Error code	GFS_ERR_ -	2.7
Other		2.8
Data types		3.0
File handle	GfsHn	3.1
Directory information control	GfsDirTbl	3.2
Directory information with no file name	GfsDirId	3.3
Directory information with file name	GfsDirName	3.4
Error processing function	GfsErrFunc	3.5
Error status	GfsErrStat	3.6

7.1 Basic Data

Title	Data	Data Name	No.
Data specification	Basic data		1.0

1) Basic Data Types

A table of the basic data structure is shown below.

Type name	Explanation
UInt8	1-byte integer without sign
Sint8	1-byte integer with sign
UInt16	2-byte integer without sign
Sint16	2-byte integer with sign
UInt32	4-byte integer without sign
Sint32	4-byte integer with sign
Bool	logic type 4-byte integer

2) Logical Constants

These are used as logical (Bool) values.

Constant name	Value	Explanation
FALSE	0	False logical value
TRUE	1	True logical value

3) NULL Pointer

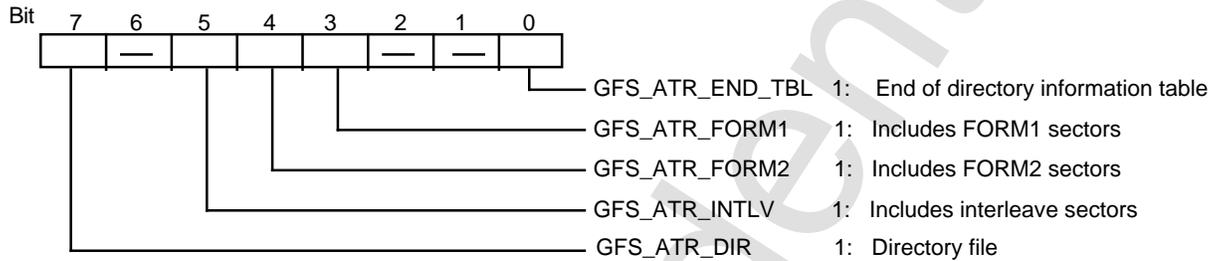
Constant name	Value	Explanation
NULL	(void *)0	Null pointer



7.2 Constants

Title	Data	Data Name	No.
Data specification	File attribute	GFS_ATR_-	2.1

The constants shown below indicate the presence or absence of their respective attributes. These constants are used for attributes in directory information read by GFS_Init and GFS_LoadDir and for attributes output by GFS_GetFileInfo. The bits not shown here are undefined.



Title	Data	Data Name	No.
Data specification	Access status	GFS_NWSTAT_-	2.2

The constants shown below indicate the access status of the server. Therefore, output using GFS_NwGetStat.

Constant name	Explanation
GFS_NWSTAT_NOACT	No action
GFS_NWSTAT_FREAD	GFS_NwFread is being executed
GFS_NWSTAT_CDREAD	GFS_NwCdRead is being executed

Title	Data	Data Name	No.
Data specification	Access Server status	GFS_SVR_-	2.3

The constants shown below are functions of GFS_NwExecOne and GFS_NwExecServer. Their execution status is shown.

Constant name	Explanation
GFS_SVR_COMPLETED	Processing complete
GFS_SVR_BUSY	Processing in progress
GFS_SVR_CDPAUSE	Temporary pause because CD buffer full
GFS_SVR_ERROR	Error has occurred during access

Title	Data	Data Name	No.
Data specification	Seek mode	GFS_SEEK_-	2.4

The constants below indicate the reference when moving an access pointer. These are used as arguments for GFS_Seek.

Constant name	Explanation
GFS_SEEK_SET	Top of file
GFS_SEEK_CUR	Current position
GFS_SEEK_END	End of file

Title	Data	Data Name	No.
Data specification	Fetch mode	GFS_GMODE_-	2.5

The constants below indicate the method by which data are fetched from the CD ROM buffer. These are used as arguments for GFS_SetGmode.

Constant name	Explanation
GFS_GMODE_ERASE	Delete from buffer after transferring
GFS_GMODE_RESIDENT	Leave in CD buffer after transferring



Title	Data	Data Name	No.
Data specification	Transfer mode	GFS_TRANS_	2.6

The constants below indicate the device that executes transfer from the CD buffer. These are used as arguments of GFS_SetTmode.

Constant name	Explanation
GFS_TMODE_SCU	DMA transfer by SCU
GFS_TMODE_SDMA0	DMA cycle steal transfer channel 0
GFS_TMODE_SDMA1	DMA cycle steal transfer channel channel 1
GFS_TMODE_CPU	Software transfer

SEGA Confidential

Title	Data	Data Name	No.
Data specification	Error codes	GFS_ERR_	2.7

The value of GFS_ERR_OK is "0". Other error codes have negative values.

Constant name	Explanation
GFS_ERR_OK	Normal end
GFS_ERR_CDRD	CD read error
GFS_ERR_CDNODISC	No CD is set in the player
GFS_ERR_CDROM	A non-CD ROM disc is set in the player
GFS_ERR_DIRTBL	Contents of directory control table are incorrect
GFS_ERR_OPENMAX	The value for the maximum number of opens is incorrect
GFS_ERR_DIR	The specified file is not a directory
GFS_ERR_CDBFS	CD block file system error
GFS_ERR_NONAME	File names cannot be used in the current directory
GFS_ERR_NEXIST	File name does not exist
GFS_ERR_FID	Incorrect file identifier
GFS_ERR_HNDL	File handle is incorrect
GFS_ERR_SEEK	Seek position is incorrect
GFS_ERR_ORG	Reference position is incorrect
GFS_ERR_NUM	Byte number is negative
GFS_ERR_OFS	Incorrect offset
GFS_ERR_FBUSY	Processing of specified file handle not complete
GFS_ERR_PARA	Incorrect mode
GFS_ERR_BUSY	Library function is being executed
GFS_ERR_NOHNDL	No open file handle
GFS_ERR_PUINUSE	Pickup is being used
GFS_ERR_ALIGN	Data read area is not in 4-byte boundary
GFS_ERR_TMOUT	Internal processing time out
GFS_ERR_CDOPEN	Tray is open
GFS_ERR_BFUL	Read stopped because buffer is full

Title	Data	Data Name	No.
Data specification	Other		2.8

Other constants used in this library are shown below.

Constant name	Explanation
GFS_RPARA_DFL	Initial value of read parameter
GFS_BUFSIZ_INF	Specifies read regardless of the size of the read area in GFS_Load



7.3 Data Types

Title	Data	Data Name	No.
Data specification	File handle	GfsHn	3.1

Holds information regarding file access for each file. The information is generated by GFS_Open. Most functions that access files reference this data.

Title	Data	Data Name	No.
Data specification	Directory information control	GfsDirTbl	3.2

Directory information control structures can control directory information without file names and directory information with files names. The following constants specify which is controlled.

Constant name	Explanation
GFS_DIR_ID	Does not have file name information
GFS_DIR_NAME	Has file name information

These are data types for controlling directory information. The directory information table classification and its size and substance are held.

Access macro	Type	Explanation
GFS_DIRTBL_TYPE(dirtbl)	Sint32	Directory information table classification
GFS_DIRTBL_NDIR(dirtbl)	Sint32	Maximum number of elements in directory information table
GFS_DIRTBL_DIRID(dirtbl)	GfsDirId *	Pointer to directory information table with no file names
GFS_DIR_NAME(dirtbl)	GfsDirName *	Pointer to directory information table with file names

(a) When directory information with no file names is used

```
GfsDirTbl dirtbl;
GfsDirId dir_id[MAX_DIR_ID]
GFS_DIRTBL_TYPE(&dirtbl) = GFS_DIR_ID; /*specifies directory information with no */
/*file name */
GFS_DIRTBL_NDIR(&dirtbl) = MAX_DIR_ID; /*maximum number of elements */
GFS_DIRTBL_DIRID(&dirtbl) = dir_id; /*directory information table with no file */
/*name */
```

(b) When directory information with file names is used

```
GfsDirTbl dirtbl;
GfsDirName dir_name[MAX_DIR_NAME]
GFS_DIRTBL_TYPE(&dirtbl) = GFS_DIR_NAME; /*specifies directory information with */
/*file name */
GFS_DIRTBL_NDIR(&dirtbl) = MAX_DIR_NAME; /*maximum number of elements */
GFS_DIRTBL_DIRNAME(&dirtbl) = dir_name; /*directory information table with file */
/*name */
```

Title	Data	Data Name	No.
Data specification	Directory information without file names	GfsDirId	3.3

These are directory information structures with no file names. GFS_DIR_ID is used to specify the type of directory information table (GFS_DIRTBL_TYPE).

GfsDirId *dir

Access macro	Type	Explanation
GFS_DIR_FAD(dir)	Sint32	Top FAD of file
GFS_DIR_SIZE(dir)	Sint32	Size of file (unit: byte)
GFS_DIR_FN(dir)	Uint8	File number
GFS_DIR_ATR(dir)	Uint8	File attribute
GFS_DIR_UNIT(dir)	Uint8	Unit size of file (unit: sector)
GFS_DIR_GAP(dir)	Uint8	Gap size of file (unit: sector)

Title	Data	Data Name	No.
Data specification	Directory information with file names	GfsDirName	3.4

These are directory information structures which include file names. GFS_DIR_NAME is used to specify the type of directory information table (GFS_DIRTBL_TYPE).

GfsDirId *dir

Access macro	Type	Explanation
GFS_DIR_FAD(dir)	Sint32	Top FAD of file
GFS_DIR_SIZE(dir)	Sint32	Size of file (unit: byte)
GFS_DIR_FN(dir)	Uint8	File number
GFS_DIR_ATR(dir)	Uint8	File attribute
GFS_DIR_UNIT(dir)	Uint8	Unit size of file (unit: sector)
GFS_DIR_GAP(dir)	Uint8	Gap size of file (unit: sector)
GFS_DIR_FNAME(dir)	Uint8[]	File name *1

*1: The area size is 12 bytes. When the file name length is 12 bytes, the character string does not end with '¥0.'



Title	Data	Data Name	No.
Data specification	Error processing function	GfsErrFunc	3.5

These are functions set with GFS_SetErrFunc.

Syntax: void (*GfsErrFunc) (void *obj, Sint32 ec)
 Input: obj : Object required for error processing
 ec : Error code
 Output: none

Title	Data	Data Name	No.
Data specification	Erro status	GfsErrStat	3.6

Data output by GFS_GetErrStat.

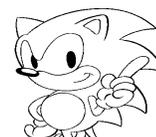
GfsErrStat *stat		
Access macro	Type	Explanation
GFS_ERR_FUNC(err)	GFSErrFunc	Pointer to error processing function
GFS_ERR_OBJ(err)	void *	First argument of error processing function
GFS_ERR_CODE(err)	Sint32	Error code

8.0 Function Specifications

A list of file system library functions is shown in Table 8.1.

Table 8.1 File System Library Function Table

Action	Function Name	No.
Directory operations		
Initialization of file system	GFS_Init	1.1
Read directory information	GFS_LoadDir	1.2
Set current directory	GFS_SetDir	1.3
Convert from file names to file identifiers	GFS_ToId	1.4
Convert from identifiers to file names	GFS_IdToName	1.5
File operations		
Open file	GFS_Open	2.1
Close file	GFS_Close	2.2
Move access pointer	GFS_Seek	2.3
Get access pointer	GFS_Tell	2.4
Check file end	GFS_IsEof	2.5
Convert from byte length to sector length	GFS_ByteToSct	2.6
Get file size	GFS_GetFileSize	2.7
Get file information	GFS_GetFileInfo	2.8
Return-Upon-Completion Loading		
File batch read	GFS_Load	3.1
Read data	GFS_Fread	3.2
Immediate-Return Reading		
Start reading data	GFS_Load	4.1
Start pre-read to CD buffer	GFS_NwCdRead	4.2
Check completion of access operation	GFS_NwIsComplete	4.3
Stop access operation	GFS_NwStop	4.4
Get current access status	GFS_NwGetStat	4.5
Execute access in file units	GFS_NwExecOne	4.6
Execute multiple file access operation	GFS_NwExecServer	4.7
Read Parameter Settings		
Get (fetch) mode setting (resident / destructive)	GFS_SetGmode	5.1
Transfer mode setting (software, DMA, etc.)	GFS_SetTmode	5.2
Amount read to CD buffer setting	GFS_SetReadPara	5.3
Amount transferred from CD buffer setting	GFS_SetTransPara	5.4
Other		
Move the CD pickup	GFS_CdMovePickup	6.1
Error processing function settings	GFS_SetErrFunc	6.2
Get error status	GFS_GetErrStat	6.3



8.1 Directory Control

Title	Function	Function Name	No.
Function specification	Initialize file system: mount	GFS_Init	1.1

Syntax Sint32 GFS_Init(Sint32 open_max, void *work, GfsDirTbl *dirtbl)
 Input open_max : maximum number of files that can be opened at one time (1 to 24)
 work : work area for library
 dirtbl : directory information control structure
 Output dirtbl : directory information control structure (directory information storage area)
 Function value Number of directories read. A negative error code is returned when an error occurs.
 Function Initializes the file system and mounts CD ROM's. The directory control function is specified by dirtbl.

dirtbl	Directory control functions used
Directory control structure	Directory control by this library
NULL	Directory control of CD block file system

Note: Work must be positioned in 4-byte boundaries. The CD block initialization flag, standby time, ECC time and number of retries do not change.

Title	Function	Function Name	No.
Function specification	Read directory information	GFS_LoadDir	1.2

Syntax Sint32 GFS_LoadDir(Sint32 fid, GfsDirTbl *dirtbl)
 Input fid : directory file identifier
 dirtbl : directory information control structure
 Output dirtbl : directory information control structure (directory information storage area)
 Function value Number of directories read. A negative error code is returned when an error occurs.
 Function Reads directory information from the specified directory file. The storage destination of the directory information will change according to the specification by dirtbl.

dirtbl	Directory information storage area
Directory control structure	Directory information storage area of dirtbl
NULL	File control information area in CD block

When NULL is specified for dirtbl, an error will result if use of the CD block file system is not declared with GFS_Init.

However, it is always possible to pass a pointer to an appropriate directory information control structure other than NULL to dirtbl.

SEG

Title	Function	Function Name	No.
Function specification	Set current directory	GFS_SetDir	1.3

Syntax Sint32 GFS_SetDir(GfsDirTbl *dirtbl)
 Input dirtbl : directory information control structure
 Output none
 Function value Error code
 Function Sets the specified directory information to the current directory. The directory information used by the dirtbl specification changes.

dirtbl	Directory information used
Directory control structure	Contents of dirtbl
NULL	File control information in CD block

When NULL is specified for dirtbl, an error will result if use of the CD block file system is not declared with GFS_Init.
 However, it is always possible to pass a pointer to an appropriate directory information control structure other than NULL to dirtbl.

Title	Function	Function Name	No.
Function specification	Convert from name to file identifier	GFS_NameToId	1.4

Syntax Sint32 GFS_NameToId(UInt8 *fname)
 Input fname : file name
 Output none
 Function value File identifier. A negative error code is returned when an error occurs.
 Function Returns a file identifier corresponding to the file name.

Title	Function	Function Name	No.
Function specification	Convert from identifier to file name	GFS_IdToName	1.5

Syntax const UInt8 *GFS_IdToName(Sint32 fid)
 Input fname : file identifier
 Output none
 Function value Pointer to file name. NULL when an error occurs.
 Function Returns a pointer to the file name corresponding to the file identifier. This point specifies the conversion table area used by this library.



8.2 File Operations

Title	Function	Function Name	No.
Function specification	Open file	GFS_Open	2.1

Syntax GfsHn GFS_Open(Sint32 fid)
 Input fid file identifier
 Output none
 Function value File handler. NULL is returned in the case of an error.
 Function Opens the specified file and returns the file handler.

Title	Function	Function Name	No.
Function specification	Close file	GFS_Close	2.2

Syntax void GFS_Close(GfsHn gfs)
 Input gfs : file handle
 Output none
 Function value none
 Function Closes the specified file handle.

Title	Function	Function Name	No.
Function specification	Move access pointer	GFS_Seek	2.3

Syntax Sint32 GFS_Seek(GfsHn gfs, Sint32 off, Sint32 org)
 Input gfs : file handle
 off : amount access point is moved (unit: sector)
 org : reference for moving (seek mode: GFS_SEEK_-)
 Output none
 Function value Position of access point after moving. A negative error code is returned if there is an error.
 Function The access pointer is moved to a position off sectors from org. If movement to a position past the file end is specified, the access pointer is moved on the assumption that the file exists up to that position.

Title	Function	Function Name	No.
Function specification	Get access pointer	GFS_Tell	2.4

Syntax Sint32 GFS_Tell(GfsHn gfs)
 Input gfs : file handle
 Output none
 Function value Position of access pointer. A negative error code is returned if there is an error.
 Function Gets the position of the access pointer.

Title	Function	Function Name	No.
Function specification	Check file end	GFS_IsEof	2.5

Syntax Bool GFS_IsEof(GfsHn gfs)
Input gfs : file handle
Output none
Function value file end flag
Function Checks whether or not the access pointer has reached the end of a file. The function values have the following meanings.
TRUE: reached file end
FALSE: has not reached file end
If an incorrect file handle is entered, then it is considered that the file end has been reached.

Title	Function	Function Name	No.
Function specification	Convert from byte size to sector length	GFS_ByteToSct	2.6

Syntax Sint32 GFS_ByteToSct(GfsHn gfs, Sint32 nbyte)
Input gfs : file handle
nbyte : number of bytes
Output none
Function value Number of sectors; returns a negative error code when an error occurs.
Function Converts the unit from byte to sector (nsct). The length nsct of the sector unit is obtained by the following equation.

$$nsct = \frac{nbyte + sector\ length\ of\ file - 1}{file\ sector\ length}$$

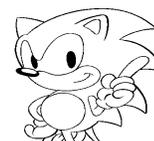
In cases in which the sector length is not defined (Form1 and Form2 are mixed), then "0" is returned.

Title	Function	Function Name	No.
Function specification	Get file size	GFS_GetFileSize	2.7

Syntax void GFS_GetFileSize(GfsHn gfs, Sint32 *sctsize, Sint32 *nsct, Sint32 *lastsize)
Input gfs : file handle
Output sctsize : number of sectors
nsct : number of sectors (does not include last sector)
Function value none
Function Gets information for seeking the file size. If NULL is specified for sctsize, nsct and lastsize, the output of the information can be suppressed. The file size is obtained by the following equation.

$$fsize = sctsize * (nsct - 1) + lastsize:$$

Note: When lastsize for the file of form2 is 2048 bytes, then it must be processed as the last sector having 2324 bytes of data.



Title	Function	Function Name	No.
Function specification	Get file information	GFS_GetFileInfo	2.8

Syntax void GFS_GetFileInfo(GfsHn gfs, Sint32 *fid, Sint32 *fn, Sint32 *fsize, Sint32 *atr)

Input gfs : file handle

Output fid : file identifier
fn : file number
fsize : file size (unit: byte)
atr : attribute

Function value none

Function Gets file information. If NULL is specified for fid, fn, fsize and atr, the output of the information can be suppressed. The file size is recorded in the directory information, and therefore the size of one sector is calculated as 2048 bytes.

8.3 Return-Upon-Completion Read

Title	Function	Function Name	No.
Function specification	Batch load a file	GFS_Load	3.1

Syntax Sint32 GFS_Load(Sint32 fid, Sint32 off, void *buf, Sint32 bsize)

Input fid : file identifier
off : offset (unit: sector)
bsize : top limit of number of data to be loaded (unit: byte)

Output buf : data load area

Function value Number of loaded data (unit: byte); negative error code is returned in case of error.

Function Specifies file identifier and loads data from file. Open and close are performed within the function.
If the file size is less than bsize, then data is loaded up to the end of the file. When GFS_BUFSIZ_INF is specified for bsize, then data from the specified position up to the end of the file is loaded

Note: Buf must be located at 4-byte boundaries.
The default values for the fetch mode, transfer mode, load parameters and fetch parameters are used.

Title	Function	Function Name	No.
Function specification	Load data	GFS_Fread	3.2

Syntax Sint32 GFS_Fread(GfsHn gfs, Sint32 nsct, void *buf, Sint32 bsize)

Input gfs : file handle
nsct : number of sector loaded
bsize : top limit of number of data to be loaded (unit: byte)

Output buf : data load area

Function value Number of bytes actually loaded.

Function Specifies an opened file handle and loads data from the file.
Loads nsct sectors of data from the access pointer. Of the data loaded, the data up to the maximum bsize byte are written to buf.
The access pointer advances nsct sectors.

Note: There are restrictions on the address boundaries of buf depending on the transfer mode.
· GFS_TMODE_SCU : no restriction
· Other than above : locate at 4-byte boundaries
Even if the access pointer is outside the file range specified by the file handle, it undergoes read processing as part of the file. Even if the number of sectors specified straddles the file end, the specified number of sectors undergo read processing.
Regardless of the value specified by GFS_SetReadPara, the default value is used for the read parameter.

8.4 Immediate-Return Read

Title	Function	Function Name	No.
Function specification	Start data loading	GFS_NwFread	4.1

Syntax Sint32 GFS_NwFread(GfsHn gfs, Sint32 nsct, void *buf, Sint32 bsize)

Input fid : file handle
nsct : number of sectors loaded
bsize : size of load area (unit: number of bytes)

Output buf : data load area

Function value error code

Function Issues a request for data load in response to a server function. Upon completion of the requested access operation, the access pointer advances nsct sectors.

Note: The same precaution as noted for GFS_Fread applies to the address boundary of buf.
Even if the access pointer is outside the file range specified by the file handle, it undergoes read processing as part of the file. Even if the number of sectors specified straddles the file end, the specified number of sectors undergo read processing.



Title	Function	Function Name	No.
Function specification	Start pre-read to CD buffer	GFS_NwCdRead	4.2

Syntax Sint32 GFS_NwCDread(GfsHn gfs, Sint32 nsct)

Input gfs : file handle
nsct : number of sectors loaded

Output none

Function value error code

Function Issues requests to server function for pre-reads to the CD buffer. Completion of the requested read operation does not cause the access pointer to change. If the following conditions are not met, the access operation for the specified file handle is not terminated.

- The nsct sector data are transferred to the host area by the GFS_Fread function or the GFS_NwFread function.

- The access operation is stopped by the GFS_NwStop function.

Note: Perform the following operation to find out whether transfer to the host area by the GFS_NwFread function after the start of a pre-read has been completed.

- Use GFS_NwGetStat to get the number of bytes transferred.

- Check if the number of bytes transferred is equal to the target number of bytes. (If equal to the target number of bytes, then transfer is complete.)

If a pre-read is performed by the GFS_NwRead function, please take note that completion of the GFS_NwFread function cannot be checked by the GFS_NwIsCompleted function.

Title	Function	Function Name	No.
Function specification	Check completion of access operation	GFS_NwIsComplete	4.3

Syntax Bool GFS_NwIsComplete(GfsHn gfs)

Input gfs : file handle

Output none

Function value status of access operation

Function Check whether the access operation of the server function is complete. The function values have the following meanings.

TRUE : access complete

FALSE : operation in progress

Title	Function	Function Name	No.
Function specification	Stop access operation	GFS_NwStop	4.4

Syntax Sint32 GFS_NwStop(GfsHn gfs)

Input gfs : file handle

Output none

Function value Stop Point access pointer. Negative error code if an error occurs.

Function Stops the access operation of a server function. GFS_NwExecServer continues the access operation on the next file.

Title	Function	Function Name	No.
Function specification	Get current access status	GFS_NwGetStat	4.5

Syntax void GFS_NwGetStat(GfsHn gfs, Sint32 *stat, Sint32 *ndata)
Input gfs : file handle
Output stat : current access status
ndata : number of data

Function value none

Function Used to get the access status of a server function. The meaning of data for each access status is shown in the table below.

Access status	Processing	Meaning of data number
GFS_NWSTAT_NOACT	None	No meaning
GFS_NWSTAT-FREAD	Read from CD to host area	Number of bytes read into host area
GFS_NWSTAT_CDREAD	Pre-read to CD buffer	Number of sectors read to CD buffer

Title	Function	Function Name	No.
Function specification	Execute access operation in file units	GFS_NwExecOne	4.6

Syntax Sint32 GFS_NwExecOne(GfsHn gfs)

Input gfs : file handle

Output none

Function value execution status (GFS_EXEC_-)

Function This is an access server function for single files. It performs the following access to the file handle in accordance with the access operation called immediately prior to it.

- GFS_NwFread: reads to CD buffer and transfers to host.
- GFS_NwCdRead: reads to CD buffer.

Title	Function	Function Name	No.
Function specification	Execute access operation for multiple files	GFS_NwExecServer	4.7

Syntax Sint32 GFS_NwExecServer(GfsHn *now_gfs)

Input none

Output now_gfs : file handle to be executed

Function value access server status (GFS_SVR_-)

Function This is an access server function for multiple files. It performs the actual access (GFS_NwExecOne function) in the order in which requests were issued.

Upon completion of the access operation for one file, processing moves to the next file.



8.5 Read Parameter Settings

Title	Function	Function Name	No.
Function specification	Set get (fetch) mode (resident/ destructive)	GFS_SetGmode	5.1

Syntax Sint32 GFS_SetGmode(GfsHn gfs, Sint32 gmode)
 Input gfs : file handle
 gmode : get (fetch) mode (GFS_GMODE_-)
 Output none
 Function value Get (fetch) mode before setting. Negative error code in the case of an error.
 Function Sets the get (fetch) mode.

Title	Function	Function Name	No.
Function specification	Set transfer mode (software/DMA, etc.)	GFS_SetTmode	5.2

Syntax Sint32 GFS_SetTmode(GfsHn gfs, Sint32 gmode)
 Input gfs : file handle
 tmode : transfer mode (GFS_TMODE_-)
 Output none
 Function value Transfer mode before setting; a negative error code in the case of an error.
 Function Sets the method of transfer from the CD buffer.

Title	Function	Function Name	No.
Function specification	Set amount read to CD buffer	GFS_SetReadPara	5.3

Syntax Sint32 GFS_SetReadPara(GfsHn gfs, Sint32 cdrsize)
 Input gfs : file handle
 cdrsize : maximum amount read at one time to CD buffer
 (unit: sector)
 Output none
 Function value Read amount before setting; a negative error code in the case of an error.
 Function Sets the maximum value for the amount read at one time to the CD buffer.

Title	Function	Function Name	No.
Function specification	Set amount transferred from CD buffer	GFS_SetTransPara	5.4

Syntax Sint32 GFS_SetTransPara(GfsHn gfs, Sint32 tsize)
 Input gfs : file handle
 tsize : amount transferred at one time to a specified area
 (unit: sector)
 Output none
 Function value Transfer amount before setting. Negative error code in the case of an error.
 Function Sets the amount of data transferred to the destination area in one get (fetch) operation.

8.6 Other

Title	Function	Function Name	No.
Function specification	Move CD pickup	GFS_CdMovePickup	6.1

Syntax Sint32 GFS_CdMovePickup(GfsHn gfs)
 Input gfs : file handle
 Output none
 Function value Error code
 Function Moves the CD pickup to the position of the access pointer. This is used to shorten pickup seek time when reading from the CD with the GFS_Fread, GFS_NwFread or GFS_NwCdRead functions.

Title	Function	Function Name	No.
Function specification	Set error processing function	GFS_SetErrFunc	6.2

Syntax void GFS_SetErrFunc(void (*func)(GfsErrFunc func), void *obj)
 Input func : function called when error occurs
 obj : first argument of func function
 Output none
 Function value none
 Function Records the function called when an error occurs. When NULL is set to func, then no error processing function is registered.

Title	Function	Function Name	No.
Function specification	Get error status	GFS_GetErrStat	6.3

Syntax void GFS_GetErrStat(GfsErrStat *stat)
 Input none
 Output stat : error
 Function value none
 Function Gets the error status of the library function executed last.



AppendixA Utilization of Development Support Functions

A.1 Procedure for Using Memory Files

The following operations must be followed in the order shown here to use memory files.

- **Write file data**
Make a file that will become the memory file and convert it to a memory file. Use the MFCAT.EXE application to convert it to a memory file.
- **Load memory file**
Set the memory file in the memory area. This is done using the ICE command. This must be done each time the contents of the memory area are destroyed.
- **Load the target program**
Load the program file being developed to the target.
- **Declare use of the memory file**
Before executing GFS_Init, the top address in memory is set to !gfsd_mmc/GFMC_base. By doing this, the SIMM directory information is obtained with GFS_Init. Since the initial value of !gfsd_mmc/GFMC_base is "0", the SIMM directory information is not referenced in this state.

GFMC_base is defined as follows:

```
Sint8 *GFMC_base = 0;
```

Refer to the CD Tool Manual regarding MFCAT.EXE.

A.2 Procedure for Using DOS Files

The following operations must be followed in the order shown here to use a DOS file.

- **Write corresponding table file**
Make a file that becomes the DOS file and make a corresponding table file for the filenames on the CD ROM and the file names in DOS.
- **Boot**
Confirm that the DIP switches are set to allow use of SCSI and boot the target.
- **Load the target program**
After confirming that the IBM PC has recognized the target as a SCSI device, break the target and load the target program.
- **Execute CDSIM.EXE**
Execute CDSIM.EXE on the IBM PC.
See the CD Tool Manual regarding corresponding table files and CDSIM.EXE.

A.3 Precautions

The precautions that should be taken when using the development support tools are listed below.

- **When the CD ROM is not used**

In order to prevent differences in programs between when they are all CD ROM files and when they are all debugger files, information for the parent directory and the current directory is added even when the CD ROM is not used. This directory information which is automatically added is set as files on the CD ROM.

- **Temporarily restricting access**

Normally, when both memory files and DOS files are accessed, the DIP switches of the target are set so that the SCSI is not used in order to temporarily stop DOS file access, and the computer is rebooted. In order to access only DOS files, however, "0" is set in !gfsd_mmc/GFMC_base.

- **File name access recommended**

During use of debugger files, the file identifier may change due to a change in the file structure. Therefore, the use of access by file name is recommended.

- **Debugger file directory information**

Debugger file directory information is set as follows:

FAD	: file identifier in debugger file
File number	: 0
Gap size	: 0
Unit size	: 0



Appendix B Error Processing Methods

The causes and remedies of the following error codes is shown below.

GFS_ERR_CDRD

Cause: Read error in CD block.

Remedy: Check CD ROM hardware and CD ROM media.

GFS_ERR_CDNODISC

Cause: The CD ROM is not set in place.

Remedy: Reset the CD ROM in place.

GFS_ERR_CDRROM

Cause: A disc that is not a CD ROM has been inserted.

Remedy: Insert a disc that is a CD ROM.

GFS_ERR_DIRTBL

Cause: The contents of the directory control structure are not correct.

Remedy: Check whether a value or a correct value has been set in each member of the directory control structure and whether that value is correct before calling GFS_Init and GFS_LoadDir.

GFS_ERR_OPENMAX

Cause: The value for the maximum number of open files is incorrect.

Remedy: Check if the specification for the maximum number of files opened by calling GFS_Init exceeds the range of 1 to 24.

GFS_ERR_DIR

Cause: The specified file is not a directory.

Remedy: Check correspondence between file identifiers and files.

GFS_ERR_CDBFS

Cause: An attempt to use the CD block file system was made even though there was no use declaration.

Remedy: If the CD block file system is to be used, then specify NULL for the directory control area address and call GFS_Init.

If the CD block file system is not going to be used, do not specify NULL for GFS_LoadDir and GFS_SetDir.

GFS_ERR_NONAME

Cause: File names cannot be handled by the current directory.

Remedy: Specify a directory control area in which GFS_DIR_NAME has been set to GFS_DIRTBL_TYPE and call GFS_Init or GFS_LoadDir.

GFS_ERR_NEXIST

Cause: The specified file name does not exist.

Remedy: Check if the current directory setting or the file name specification is incorrect.

GFS_ERR_FID

Cause: The file identifier specification is incorrect.

Remedy: Check if the specified file identifier has exceeded the range of GFS_DIR_NDIR of the directory control structure set in the current directory.

GFS_ERR_HNDL

Cause: File handle is incorrect.

Remedy: Check if the function value is set of GFS_Open in the variable in which the file handle is stored, or that the contents of the variable have not been destroyed.

GFS_ERR_SEEK

Cause: The seek location is incorrect.

Remedy: Check the seek position calculated from the reference position and the offset.

GFS_ERR_ORG

Cause: The reference position of GFS_Seek is incorrect.

Remedy: Check to be sure the reference position is at one of GFS_SEEK_SET, GFS_SEEK_CUR or GFS_SEEK_END.

GFS_ERR_NUM

Cause: A negative number of bytes was specified.

Remedy: Check the number of bytes specified by GFS_ByteToSct.

GFS_ERR_OFS

Cause: The offset is incorrect.

Remedy: Check the read start sector position specified by GFS_Load.

GFS_ERR_FBUSY

Cause: Processing of specified file handle remains to be performed.

Remedy: Correct the program so that it accesses again after completing access of the target file, or reconsider the file structure.

GFS_ERR_PARA

Cause: Incorrect mode.

Remedy: Make sure that correct arguments are given to GFS_SetGMode, GFS_SetTmode, GFS_SetReadPara and GFS_SetTransPara.

GFS_ERR_BUSY

Cause: Multiplex processing was attempted.

Remedy: Refer to "6.3 Multiple Processing".



GFS_ERR_NOHNDL

Cause: No open file handles.

Remedy: Either increase the specification for the maximum number of files that can be opened at the same time by GFS_Init or reduce the number of files opened at the same time.

GFS_ERR_PUINUSE

Cause: GFS_CdMovePickup was called while the pickup was in use.

Remedy: Call GFS_CdMovePickup in a state in which file access is not being performed.

GFS_ERR_ALIGN

Cause: The read destination of a file is not located at a word boundary.

Remedy: Position the read area at a word boundary.

GFS_ERR_TMOUT

Cause: A response was not received from the CD block in the prescribed time period.

Remedy: Something may be wrong with the hardware.

GFS_ERR_CDOPEN

Cause: The tray on the CD drive is open.

Remedy: Close the tray and then continue.

GFS_ERR_BFUL

Cause: The CD buffer becomes full and reading is stopped when the fetch mode is GFS_GMODE_RESIDENT.

Remedy: Adjust the order of access and the amount read to prevent the CD buffer from becoming full, when the fetch mode is GFS_GMODE_RESIDENT.

Appendix C

This is an addition to the main part of the manual.

C.1 Additional Explanation

DOS File Parameters

The initial value of the fetch parameters for DOS files is "1". Since only one sector can be transferred at a time, settings other than "1" for the fetch parameters are invalid.

C.2 Changes from the Previous Version

1) Changes in CD Pre-read Processing

- Change in how GFS_NwCdRead is used
Even if GFS_NwExecOne is not called after calling GFS_NwCdRead, pre-read from the CD is enabled.

When look ahead from the CD is used, either all of the specified number of sectors is transferred to the host, or it is not terminated until access is stopped (GFS_NwIsComplete does not return TRUE).

- Access Complete Check (Important)

Since GFS_NwCdRead is not terminated until either data transfer ends or access is stopped (except access of CDDA files), do not wait for termination. As is shown in the example in the old manual, an endless loop will result if waiting for termination of GFS_NwExecOne, so use caution.

If GFS_NwFread does not perform pre-read with NwCdRead, then termination can be checked using the same procedure as in the previous version. If pre-read from the CD is being used, then use GFS_NwGetStat to check for termination while also monitoring the number of bytes transferred. An example is shown below.

```
GfsHn  gfs;
Sint32 fid, stat, nbyte;
Uint32 buf[10*2048/4];

gfs = GFS_Open(fid);
GFS_NwRead(gfs, 100)
for (i = 0, i < 10; ++i) {
    GFSNwFread(gfs, 10, buf, 10*2048);
    while (GFS_NwExecOne (gfs) != GFS_SVR_COMPLETE) {
        GFS_NwGetStat(gfs, &stat, &nbyte);
        /*checks whether number of bytes specified by GFS_NwFread has been read */
        if (nbyte >= 10*2048) {
            break;
        }
    }
    user();          /*application processing */
}
}
```



2) Addition of CDDA File Processing Function

A CDDA file processing function has been added. When a CDDA file is read, the music track specified by that file is played. However, in order to output sound, SCSP must be set by the application. CDDA files and regular files have the following differences.

- Control of files

The only controls the file system performs on CDDA files are playback and playback range. The playback mode is an omitted value (no repeat, moves pickup).

- Pre-read

Since the data from CDDA files does not enter the CD buffer, when they are accessed, pre-read processing and normal access are equivalent.

- Parameters relating to file operation

The fetch mode, transfer mode, read parameters and fetch parameters cannot be changed for CDDA files. An error is returned when the following functions are called for CDDA files.

```
GFS_SetGmode
GFS_SetTmode
GFS_SetReadPara
GFS_SetTransPara
```

3) File attributes

The following changes were made to make the values of file attributes output by GFS_GetFileInfo conform with the CD-ROM XA standard.

```
GFS_ATR_DIR      0x80
GFS_ATR_CDDA    0x40
GFS_ATR_INTLV   0x20
GFS_ATR_FORM2   0x10
GFS_ATR_FORM1   0x80
GFS_ATR_END_TBL 0x01
```

GFS_ATR_CDD was added because of the addition of CDDA file processing functions. Its bit are "1" in the CDDA files. Other constant names and meanings are unchanged.

4) Function values of GFS_Init and GFS_LoadDir

When NULL is specified for the pointer to the directory information control structure of an argument, the number of directories being held by the CD block file system is returned as a function value.

5) Addition of error codes

The following error codes were added.

- GFS_ERR_BFUL

This error code is generated if the CD buffer becomes full while a resident mode (GFS_GMODE_RESIDENT) file is being read. Adjust the order of access, etc., to prevent a buffer full condition during resident mode file access.

- GFS_ERR_FATAL

This error code serves notice that the CD drive is in a fatal condition. When the file system detects this condition, CD playback is stopped (seek home position) and recovery from the error condition is attempted. If this error condition is detected, try processing again.

6) Recognition of tray open condition

A "1" value for the DCHG bit (bit 5) of the interrupt factor register (HIRQREQ) of the CD block is also treated as a tray open condition.

7) Precautions when using SCU-DMA

When the transfer mode is GFS_TMODE_SCU, the file system library uses a SCU level 0DMA end interrupt (vector number 4B). Upon completion of transfer, the interrupt vector and interrupt mask used are restored to their original state.

8) Debug file-related items

- GFMC_base

The variable GFMC_base which sets the top address of the SIMM file is defined in both sega_gfs.lib and segadgfs.lib.

GFMC_base in sega-gfs.lib exists only to establish compatibility with segadgfs.lib. It does not affect the operation of the file system.

- File identifiers

We eliminated the function that automatically added "." and ".." (current directory and parent directory) when a CD file is not used. Because of this, file identifiers agree at the time of boot up.

