SEGA™

SEGA OF AMERICA, INC.
Consumer Products Division

External Specifications

Saturn File System Library

Doc. #ST-39-R2-011094

NOTICE

When using this document, keep the following in mind:

1. This document may, wholly or partially, be subject to change without notice.

2. All rights are reserved: No one is permitted to reproduce or duplicate, in any form, the whole or part of this document without SEGA's permission.

3. 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.

4. 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.

5. No license is granted by implication or otherwise under any patents or other rights of any third party or SEGA Enterprises, Ltd.

6. This document is confidential. By accepting this document you acknowledge that you are bounded 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.

SECTION I

ST-39-R2-011094

1.0 Overview

The Saturn File System Library is a library specification for accessing files on CD. It includes the following features.

(1) File Access Format
- Supports ISO9660 Level File Access.
- The library is not accessible if using CD-ROM XA subheader information.

(2) Data Buffering
- Accessible if a CD block buffer control mechanism is present.
- Besides reading simple files, the buffer can be used for pre-reading.

(3) File Identifier
- Access is based on the file identifier (sequence number within the directory).
- Faster directory searches.
- File name → file identifier conversion function is supported, but can be removed after the file configuration on the CD is confirmed.

(4) Development Support Function
- The development support can access both a memory file and a DOS file.
- The memory files are used in place of a CD file for small amounts of data.
- Using the SCSI, DOS files on the IBM-PC can be accessed in the same way as memory files. Files that are too large to fit into memory can be used in place of files on the CD.

2.0 Definitions

Vocabulary used are shown in Table 2.1.

Table 2.1 Definition of Terms (1)

<table>
<tr><th>TERM</th><th>MEANING</th></tr>
<tr><td>CD Buffer</td><td>(4M DRAM) buffer that stores sector data read from the CD</td></tr>
<tr><td>CD Buffer Size</td><td>Size of sector units of the CD buffer</td></tr>
<tr><td>CN</td><td>Channel Number</td></tr>
<tr><td>DOS File</td><td>Files on the IBM-PC that are accessible through the SCSI interface.
These can be used in the debug library.</td></tr>
<tr><td>FN</td><td>File Number</td></tr>
<tr><td>Access Pointer</td><td>Shows the access position of the file.
The access pointer moves in sector units.
Sector size changes according to the type of file.
<table>
<tr><th>Type of File</th><th>Sector Length (byte)</th></tr>
<tr><td>CD-ROM mode 1
CD-ROM mode 2 form1 only
CD-ROM mode 2 form2 only
CD-ROM mode 2 Mixed
Memory File</td><td>2048
2048
2324
Undecided
2048</td></tr>
</table>
</td></tr>
<tr><td>Current Directory</td><td>Directory referred to when opening a file.</td></tr>
<tr><td>Sector Position</td><td>Position of sector units as designated within the sector set by the FN in a buffer partition. (Belongs to the buffer partition.) A specific sector within a sector belonging to a file name can be designated by the sector position. The host keys the sector position and accesses sector data within a buffer partition.</td></tr>
<tr><td>Partial RAM</td><td>Area accompanying the CD media that can be read and written to.</td></tr>
</table>

TERM	MEANING
Buffer Partition	CD buffer is divided into several logic partitions. Access data read from the CD is stored, in order, in one buffer partition.
Buffer Partition Number	Number assigned to the buffer partition.
Buffer Physical Position	Sector unit position within the CD buffer. Takes its value from 0 to the CD buffer size (-1).
Buffer Logical Position	Position of sector units within the buffer partition (belongs to the buffer partition.) Takes its value from 0 to the buffer partition size (-1).
File Identifier	Sequential number in the directory that is used to identify the file. Takes its value from 0 to the directory record count (-1). 0: shows its own directory, 1 shows the parent directory.
Frame Address (FAD)	With the absolute time 00:00:00 on the CD at 0, numbers are continuously assigned to frame units. Corresponds to absolute time on a one-to-one basis. The frame address, not the absolute time, accesses the CD block as the key. (Also for CD-ROM sector data and CD-DA music data.)
Logic Sector Number (LSN)	With the absolute time 00:02:00 on the CD at 0, numbers are continuously assigned to sector (frame) units. Logic sector number = frame address - 150 (2 sec segments). Logic sector number is used in the directory table (ISO9660).

3.0 Module Configuration

The file system library consists of the following two libraries.

- GFS (General File System): Commonly used file system library.
- FNIT (File Name to ID Table): File name/file identifier conversion library.

A diagram of the module configuration is shown in Figure 3.1.

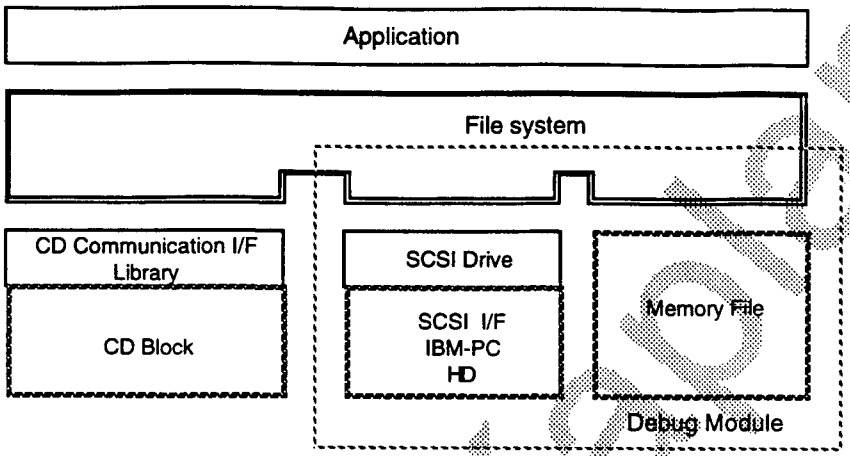

Figure 3.1 Module Configuration

Figure 3.2 demonstrates the data flow when the file system library is used.

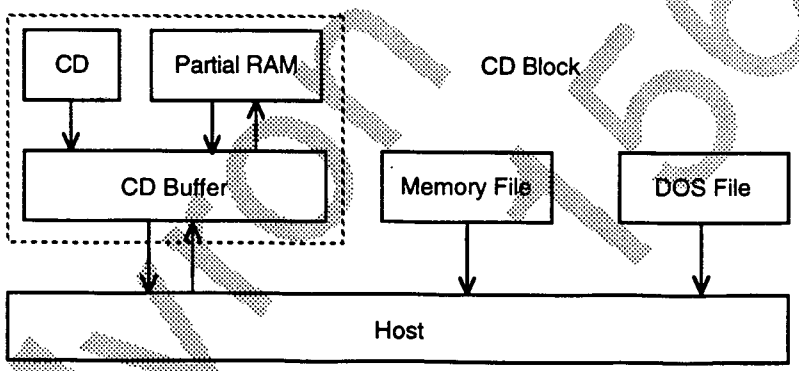

Figure 3.2 Data Flow

Name Limits

The file system library uses the following types of names for functions, variables, types and macros.

Functions, Variables	Names that start with GF, gf, GP, gp
Forms	Names that start with Gf, Gp
Macros	Names that start with GF, GP

Make certain that applications using this library don't conflict with these names.

4.0 File Access Overview

4.1 File Identifier

In the file system, the file designated by the file identifier is accessed (see the example in Figure 4.1.)

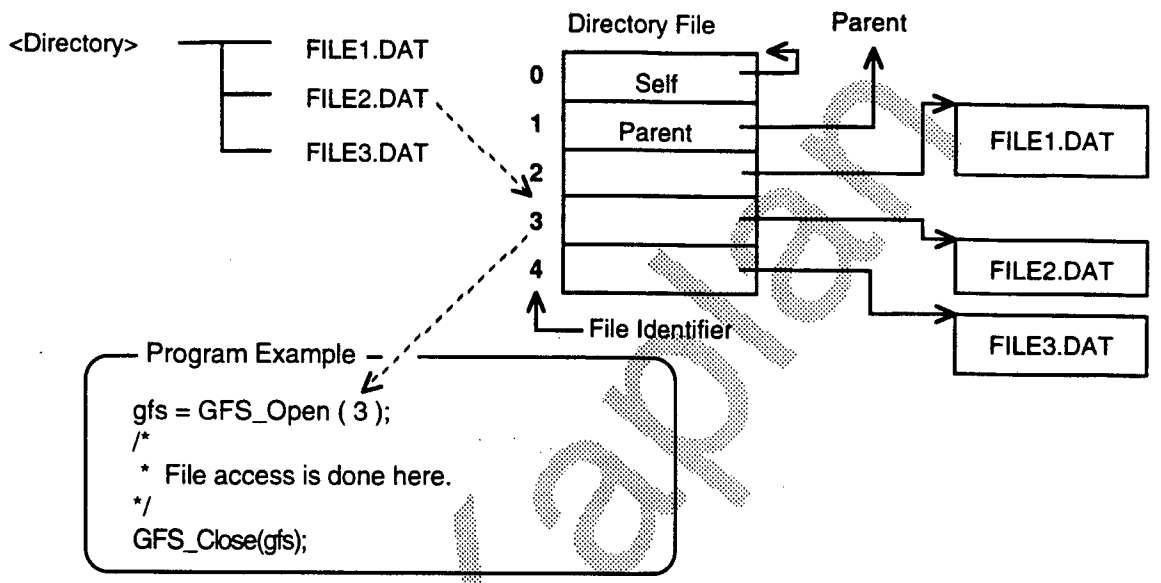

Figure 4.1 Accessing File2.dat with the File Identifier

4.2 Directory

In order to access subdirectory files, set the current directory information by following the steps below.

(1) Read Directory Information (GFS_LoadDir)

Selects the subdirectory file, reads the directory information and saves.
Here you can select between 2 types of directory information to be saved.

(a) File identifier access
- Doesn't save the file name; only access by file identifier.

(b) File name access
- Saves the file name, requiring a larger area.
- Files can be accessed by their file name.

In addition, if NULL is designated in the directory information read area, then directory information is saved in the CD block. In this case, data for the first 256 directories can be saved, but cannot be accessed by file name.

(2) Setting the Current Directory (GFS_SetDir)

The GFS_LoadDir makes the read directory information area the current directory.

If several subdirectories are to be accessed, save the information in each directory in advance, then switch using the GFS_SetDir. This way, the directory code isn't accessed on the CD each time directories are switched.

In addition, if the directory area is designated as NULL, the directory information currently saved in the CD block can be used as the current directory.

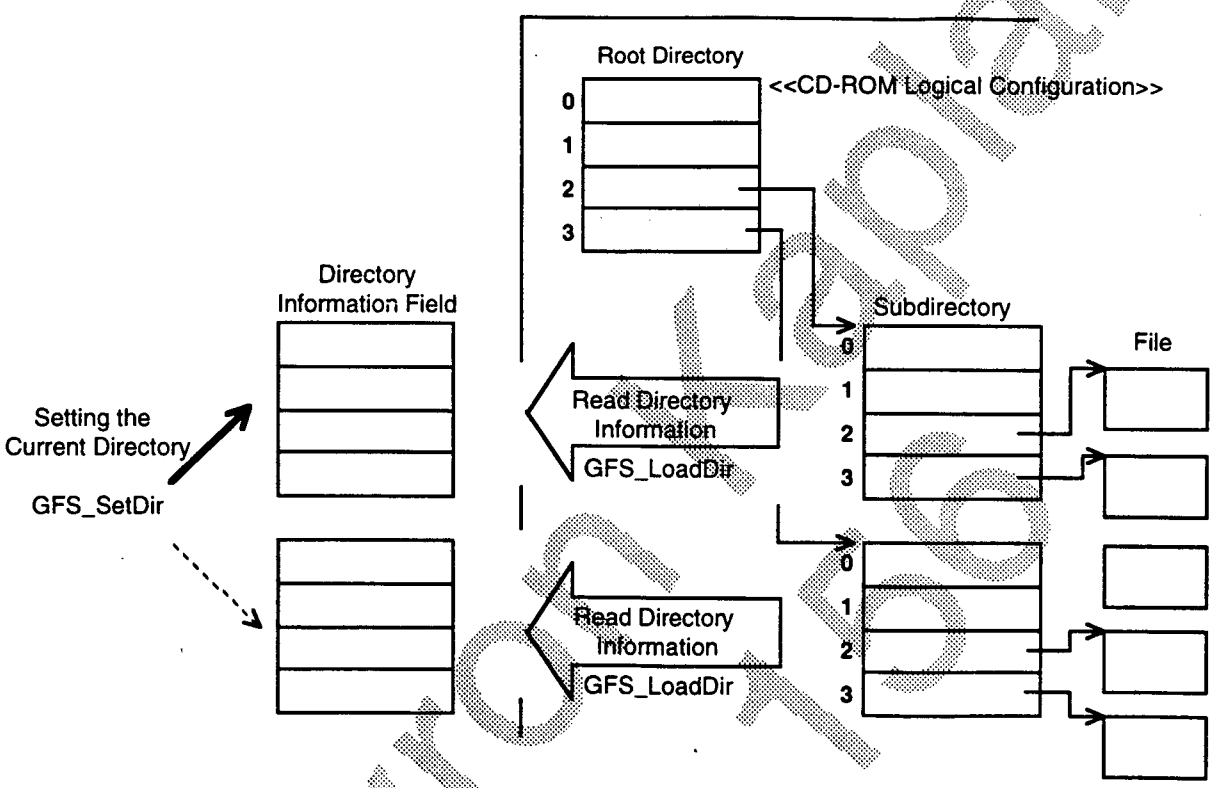

Figure 4.2 Setting Directory Information

For example, to access a file in a directory other than the root directory, or to access a file in a subdirectory, follow these steps.

Read Directory Information
↓
Set the Current Directory
↓
Open the File
↓
Access the File
↓
Close the File

The following program example shows the accessing of a file in a subdirectory. The file to be access is in the directory "dir_fid" located in the current directory.

```
GfsDirTbl dirtbl:        /* Area containing directory information */
GfsDirId dirid[MAX_DIR]; /* Area containing directory information */
GfsFid dir_fid;          /* The directory file identifier goes here */
GfsFid fid;              /* The access file identifier goes here */
GfsHn gfs;               /* File handle of the access file */

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

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

/* Set the identifier of the file to access fid */
gfs = GFS_Open(fid);
/*
* File is accessed here
*/
GFS_Close(gfs);
```

To access multiple files in different directories, open the target file while switching directories.

The following shows a program example of the simultaneous opening of two files from two sub-directories located directly below the current directory. The file identifier for file to be accessed in the sub-directories are designated by "dir_fid1", "dir_fid2", respectively.

```
GfsDirTbl curdir;           /* Current directory at this point */
GfsDirTbl dirtbl1, dirtbl2; /*Directory information control area */
GfsDirId dirid1[MAX_DIR];   /* Area containing directory information */
GfsDirId dirid2[MAX_DIR];   /* Area containing directory information */
GfsFid dir_fid1, dir_fid2;  /* The directory file identifier goes here */
GfsFid fid1, fid2;          /* The access file identifier goes here */
GfsHn gfs1, gfs2;           /* File handle of the access file */

GFS_DIRTBL_TYPE(&dirtbl1) = GFS_DIR_ID;
GFS_DIRTBL_NDIR(&dirtbl1) = MAX_DIR;
GFS_DIRTBL_DIR(&dirtbl1) = dirid1;
GFS_DIRTBL_TYPE(&dirtbl2) = GFS_DIR_ID;
GFS_DIRTBL_NDIR(&dirtbl2) = MAX_DIR;
GFS_DIRTBL_DIR(&dirtbl2) = dirid2;
/* Reads the directory information of the current directory dir_fid1 */
GFS_LoadDir(dir_gfs1, &dirtbl1);
/* Reads the directory information of the current directory dir_fid2 */
GFS_LoadDir(dir_gfs2, &dirtbl2);

GFS_SetDir(&dirtbl1);

gfs1 = GFS_Open(fid1);

GFS_SetDir(&dirtbl2);

gfs2 = GFS_Open(fid2);
/*
* File is accessed here
*/
GFS_Close(gfs1);
GFS_Close(gfs2);
```

4.3 Reading Batch Files

GFS_Load ( ) is a function that simplifies the reading of files. It allows file data to be read in the host area, and won't return until the read has finished. An example of a program that reads all file data is shown below.

```
GfsFid fid;                    / *  file identifier  */
Uint8 buf [BUF_SIZE];  / *  read area  */

fsize = GFS_Load (fid, 0, buf, BUF_SIZE);
                          ÷
                          :
                          :.....offset
```

- Data is read from the file fid sector 0 to buf [BUF_SIZE].
- When this function is finished, fsize byte data can be read in buf [ ].
- If the BUF_SIZE is bigger than the file size, the file size in fsize is changed.

4.4 Access Function Model

The high-level access function model is shown in Figure 4.3.

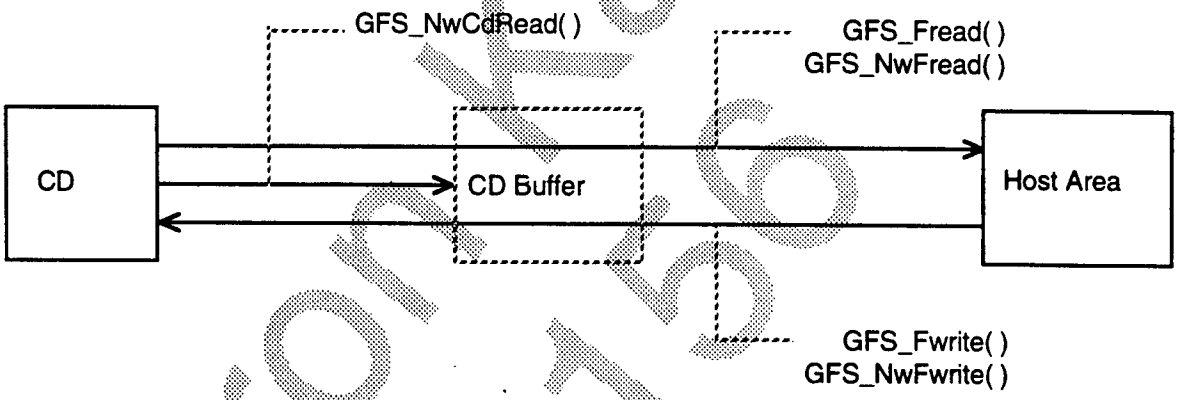

Figure 4.3 High-Level Access Model

The low-level access function model is shown in Figure 4.4.

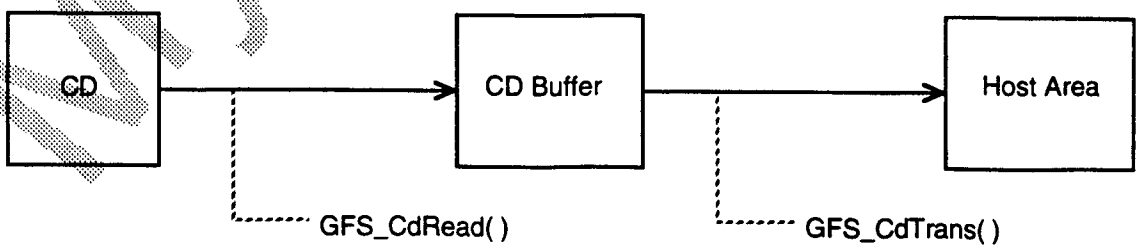

Figure 4.4 Low-Level Access Model

4.5 High-Level Access

The high-level access function allows a file to be opened without knowledge of the CD buffer inside the CD block. There are two methods of high-level access:

- Completion Return Access
- Immediate Return Access

(1) Completion Return Access

Does not return until entry from the CD is complete. An example of the Completion Return Access is shown below.

```
GfsHn  gfs;                                  / *  File Handle * /
GfsFid  fid;                                 / *  File Identifier * /
Sint32 nsct;                                 / *  Read sector count * /
Uint8 buf [BUF_SIZE];                        / * Read Area  * /

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

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

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

(2) Immediate Return Access

Returns immediately after the input command to the CD has been sent. As a result, other processing can be done simultaneous with the input completion condition. Data inputted to the CD block must first be checked. The data send process to the read area must be activated separately. An example of Immediate Return Access is shown below.

```
gfs = GFS_Open (fid);                        / * open file * /

GFS_NwFread (gfs, nsct, buf, BUF_SIZE);      / * read nsct sector to buf */
                                             / * return immediately */

for (; ;)  {
stat = GFS_NwExecOne (gfs) ;                 / * execute reading * /
if (stat = = GFS_EXEC_COMPLETED) {           / * end reading   ?* /
break;
}
user ( );                                    / * optional user process * /
}

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

4.6 Low-Level Access

Data read from the CD is fetched through the CD buffer to the host side.
In low-level access, reading and retrieving data from the CD buffer can be controlled.

An example of the low level access program is shown below:

```
gfs = GFS_Open (fid);

GFS_CdRead (gfs, nsct);                          / * read command to CD buffer * /
tr_flag=0;
for (;;)  {
   if (tr_flag = = 0)  {                         / * has transmission occurred? * /
     / * if not currently transmitting* /
     if (  GFS_CdGetRcnt (gfs) = = nsct){        / * has data accumulated in CD buffer? * /
           GFS_CdTrans (gfs, nsct, buf, BUF_SIZE); / *  start sending * /
           tr_flag = 1;
   }
   }  else  {
        / *  if currently sending    * /
        if   (GFS_CdIsTrans (gfs)  )             / *  finished sending ? * /
             break;
        }
   }
   user ( ) ;                                    / *  optional user process * /
}
GFS_Close (gfs) ;                                / *  close file * /
```

4.7 Multiple File Access

(1) Parallel-processing of Multiple Read Files

An "access server" is provided for application processing of files while multiple files are being read. Access operations are executed one after another by periodically transferring control to the server.

```
gfs 1= GFS_Open(fid1);                      / * open file              * /
gfs 2= GFS_Open(fid2);
gfs 3= GFS_Open(fid3);
GFS_NwFread(gfs1, NSCT1, buf1, BSIZE1);     / * start reading          * /
GFS_NwFread(gfs2, NSCT2, buf2, BSIZE2);
GFS_NwFread(gfs3, NSCT3, buf3, BSIZE3);
for  (; ;)  {
    stat  =  GFS_NwExecServer(&now_gfs);    / * execute read           * /
                                            / * now_gfs is an accessing file* /
    if (stat == GFS_SVR_NOFILES)    {       / * Is execute task gone?  * /
        break;
                                            }
    user ( ) ;                              / * optional user process  * /
      }
    }
GFS_Close(gfs1);
GFS_Close(gfs2);
GFS_Close(gfs3);
```

(2) Pre-reading to the CD Buffer Multiple File

Multiple files can be read to the CD buffer simultaneous with other processes. By reading in advance, data can be sent when specifically needed. The following is a sample program of Lead processing to the CD buffer:

```
gfs 1= GFS_Open(fid1);                          / * opens file              * /
gfs 2= GFS_Open(fid2);
gfs 3= GFS_Open(fid3);
GFS_NwCdRead(gfs1, NSCT1);                      / * start reading to CD buffer * /

GFS_NwCdRead(gfs2, NSCT2);
GFS_NwCdRead(gfs3, NSCT3);
for  (; ;)  {
     stat  =  GFS_NwExecServer(&now_gfs);       / * execute read            * /
                                                / * now_gfs is being accessed * /
     if (stat == GFS_SVR_NOFILES)               / * Is execute task gone?    * /
     break;
     }
     user ( ) ;                                 / * optional user process    * /
}
GFS_Fread(gfs1, NSCT1, buf1, BSIZE1);   / * data sent to host area at one time* /
GFS_Fread(gfs2, NSCT2, buf2, BSIZE2);
GFS_Fread(gfs3, NSCT3, buf3, BSIZE3);
GFS_Close(gfs1);
GFS_Close(gfs2);
GFS_Close(gfs3);
```

SECTION II

(ST-39-R2-011094)

1.0 DATA SPECIFICATIONS

Title	Function	No
Data Specification	Basic Data	

Table 1.1 shows basic data configuration.

Table 1.1 Basic Data Configuration

Type	Explanation
Uint8	1 byte integer with no flag
Sint8	1 byte integer with flag
Uint16	2 byte integer with no flag
Sint16	2 byte integer with flag
Uint32	4 byte integer with no flag
Sint32	4 byte integer with flag
Bool	Logical types. Takes the following values: FALSE TRUE

Title	Function	No
Data Specification	Library Data	

(1) Structure

Table 1.2 shows the structure.

Table 1.2 Data Structure Table

Type	Explanation
GfsFid	File Identifier
GfsHn	File handler. Created by GFS_Open, the file access function refers to this data.
GfsDirTbl	Directory Information Control Structure. Saves the directory information table type, size, and state.
GfsDirId	Directory information control structure without the directory name. Designates GFS_DIR_ID for directory information table type.
GfsDirName	Directory information control structure including the directory name. Designates GFS_DIR_NAME for directory information table type.
GfsErr	Error control structure.
GfsErrFunc	Pointer to the error processing function.

(2) Structure Access Macro

The access macro for GfsDirTbl is shown in Table 1.3.

Table 1.3 GfsDirTbl Access Macros

GfsDirTbl dirtbl;

Access Macro	Type	Explanation
GFS_DIRTBL_TYPE(&dirtbl)	Sint32	Type of directory information table. Sets GFS_DIR_ID or GFS_DIR_NAME.
GFS_DIRTBL_NDIR(&dirtbl)	Sint32	Maximum number of elements in the directory information table.
GFS_DIRTBL_PATH(&dirtbl)	Uint8[ ]	Directory path name being saved by the directory information table. The size in GFS_DIRTBL_PATHSIZE is only valid when using the debugger library.
GFS_DIRTBL_DIR(&dir)	void*	Pointer to the directory information table. Following the directory information table type, the pointer is set for the arrangement of either GfsDirId or GfsDirName.

2.0 FUNCTION SPECIFICATIONS

Table 2.1 shows a table of file system functions.

Table 2.1 Universal File System Functions

Function	Name	No.
Load		1.0
Batch reading of files	GFS_Load	1.1
All Levels Common		2.0
Initializes File System	GFS_Init	2.1
Opens File	GFS_Open	2.2
Closes File	GFS_Close	2.3
Moves Access Pointer	GFS_Seek	2.4
Gets Position of Access Pointer	GFS_Tell	2.5
Checks File End	GFS_IsEof	2.6
Converts Byte size to Sector length	GFS_ByteToSct	2.7
Gets File Size	GFS_GetFileSize	2.8
Gets File Information	GFS_GetFileInfo	2.9
Sets Get Mode (Permanent / Destructive)	GFS_SetGmode	2.10
Sets Transfer Mode (software / DMA, etc.)	GFS_SetTmode	2.11
Directory Operation		3.0
Reads Directory Information	GFS_LoadDir	3.1
Sets Current Directory	GFS_SetDir	3.2
Converts from File Name to File Identifier	GFS_NameToId	3.3
Converts from Identifier to File Name	GFS_IdToName	3.4
High-Level End Return		4.1.0
Reads Data	GFS_Fread	4.1.1
Writes Data	GFS_Fwrite	4.1.2
High-Level Immediate Return		4.2.0
Starts Data Read	GFS_NwFread	4.2.1
Starts CD buffer read	GFS_NwCdRead	4.2.2
Starts Data Write	GFS_NwFwrite	4.2.3
Checks Access Operation End	GFS_NwIsComplete	4.2.4
Stops Access Operation	GFS_NwStop	4.2.5
Gets Current Access State	GFS_NwGetStat	4.2.6
Executes File Level Access Operation	GFS_NwExecOne	4.2.7
Executes Access of All files	GFS_NwExecServer	4.2.8
High-Level Common		4.3.0
Sets CD buffer Read Parameter	GFS_SetReadPara	4.3.1
Sets Transfer Amount from CD buffer	GFS_SetTransPara	4.3.2

Table 2.1 Universal File System Functions (Part 2)

Function		Name	No.
Low-Level			5.0
	Reads to CD buffer	GFS_CdRead	5.1
	Stops read to CD buffer	GFS_CdStopRead	5.2
	Gets number of sectors read in the CD buffer	GFS_CdGetRcnt	5.3
	Transfers data from the CD buffer	GFS_CdTrans	5.4
	Stops transfer of data from the CD buffer	GFS_CdStopTrans	5.5
	Checks if reading or not	GFS_CdIsRead	5.6
	Checks whether or not data is being sent	GFS_CdIsTrans	5.7
	Moves CD pickup	GFS_CdMovePickup	5.8
	Gets CD buffer operating status	GFS_CdGetStat	5.9
Error Handling			6.0
	Sets error processing function	GFS_SetErrorFunc	6.1
	Gets error status	GFS_GetErrStat	6.2

2.1 Load

Title	Function	Function Name	No
Function Specification	Reading Files in Batch	GFS_Load	1

[Format] Sint32 GFS_Load(GfsFid fid, Sint32 off, void *buf, Sint32 bsize)
[Input] fid: File Identifier
off: Offset (In sector units)
bsize: Size of the area to be read (byte count: must be even number)
[Output] buf: Data read area
[Function value] Amount of data read (byte units)
[Function]

Reads beginning with file fid "off" sector to a buf by a bsize byte amount. Data that is larger than the file size is not read. If GFS_BUFSIZ_INF is designated in bsize, it will read from the designated point to the end of the file (no matter how large), and return the actual number of data points read. It will not return until reading is finished.

[Example]

(a) Reads from sector 0 to the end of the file. Returns the file size.

```
fsize = GFS_Load(fid, 0, buf, GFS_BUFSIZ_INF);
```

(b) Reads from sector 10 in the file for 5000 bytes to buf.

```
GFS_Load(fid, 10, buf, 5000);
```

2.2 All Levels Common

Title	Function	Function Name	No
Function Specification	Initializes File System	GFS_Init	2

[Format] Sint32 GFS_Init(GfsDirTbl *dirtbl)
[Input] None
[Output] dirtbl: Directory Information Control Structure
[Function value] Number of directories read
[Function]

Initializes the file system, reads the root directory from the CD and sets it as the root directory. Before calling this function, the following members must be set:

```
GFS_DIRTBL_TYPE(dirtbl)  Directory information area type
GFS_DIRTBL_NDIR(dirtbl)  Maximum directory elements in the
                         directory information area
GFS_DIRTBL_DIR(dirtbl)   Directory information area address
```

If NULL is entered into dirtbl, the directory information will not be read into host memory, but will be read into the CD block file information control table.

Title	Function	Function Name	No
Function Specification	Opens Files	GFS_Open	3

[Format] GfsHn GFS_Open(GfsFid fid)
[Input] fid: File Identifier
[Output] None
[Function value] File Handle
[Function]

Opens the designated file and returns the file handle. The access pointer indicates sector 0. If the designated file does not exist, the error processing function is called up.

Title	Function	Function Name	No
Function Specification	Closes Files	GFS_Close	4

[Format] void GFS_Close(GfsHn gfs)
[Input] gfs: File Handle
[Output] None
[Function value] None
[Function]

Closes the designated file handle.

Title	Function	Function Name	No
Function Specification	Moves Access Pointer	GFS_Seek	5

[Format] Sint32 GFS_Seek(GfsHn gfs, Sint32 off, Sint32 org)
[Input] gfs: File Handle
off: Amount to move the access pointer (in sector units)
org: Base point to move from
GFS_SEEK_SET: Head of the file
GFS_SEEK_CUR: Current position of the access pointer
GFS_SEEK_END: End of the file
[Output] None
[Function value] Access pointer position after movement (in sector units)
[Function]

Moves the access pointer from the org to a position separated by off sectors. The access pointer moves in increments of sector units. If the file is interleaved with other files, only moving to sector 0 is allowed.

Title	Function	Function Name	No
Function Specification	Gets Position of Access Pointer	GFS_Tell	6

[Format] Sint32 GFS_Tell(GfsHn gfs)
[Input] gfs: File Handle
[Output] None
[Function value] Access pointer position (in sector units)
[Function]

Gets the position of the access pointer.

Title	Function	Function Name	No
Function Specification	Checks File End	GFS_IsEof	7

[Format] Bool GFS_IsEof(GfsHn gfs)
[Input] gfs: File Handle
[Output] None
[Function value] End of file flag
TRUE: Went to end of file
FALSE: Do not go to end of file
[Function]

Checks to see if the access pointer imoved to the end of the file.

Title	Function	Function Name	No
Function Specification	Converts Byte Size to Sector Length	GFS_ByteToSct	8

[Format] Sint32 GFS_ByteToSct(GfsHn gfs, Sint32 nbyte)
[Input] gfs: File handle
nbyte: Byte count
[Output] none
[Function value] Sector Count
[Function]

Converts from byte size to sector count (nsct). nsct is found with the equation below:

```
nsct = (nbyte + the file sector length -1)/file sector length;
```

If the sector length is not known (Form1, Form2 combined), 0 is returned.

Title	Function	Function Name	No
Function Specification	Get File Size	GFS_GetFileSize	9

[Format] void GFS_GetFileSize(GfsHn gfs, Sint32 *sctsize, Sint32 *nsct, Sint32 *lastsize)
[Input] gfs: File Handle
[Output] sctsize: Sector Length
nsct: Sector Count
lastsize: Number of bytes the file occupies in the last sector.
[Function value] None
[Function]

Gets the size of the file. All file sizes (fsize) are found with the following equation:

```
fsize = sctsize * (nsct-1) + lastsize;
```

Depending on the file type, the sector length will return the following values.

File Type	Sector Length (bytes)
CD-ROM mode1	2048
CD-ROM mode2 form 1 only	2048
CD_ROM mode2 form2 only	2324
CD-ROM mode2 combined	0
Memory File	2048
DOS File	2048

Title	Function	Function Name	No
Function Specification	Gets File Information	GFS_GetFileInfo	10

[Format] void GFS_GetFileInfo(GfsHn gfs, GfsFid *fid, Sint32 *fn, Sint32 *fsize, Sint32 *atr)

[Input] gfs: File Handle

[Output] fid: File Identifier

fn: File Number

fsize: File size recorded in directory information.

atr: Attribute (Defined in the XA System Use Information)

[Function value] None

[Function]

Gets the file information. The file size returns the value recorded in the directory information. One sector is calculated at 2048 bytes.

Title	Function	Function Name	No
Function Specification	Sets Get Mode (Permanent/Destructive)	GFS_SetGmodet	11

[Format] void GFS_SetGmode(GfsHn gfs, Sint32 gmode)

[Input] gfs: File Handle

gmode: Get mode

[Output] None

[Function value] None

[Function]

Sets the get mode. Get modes are as follows.

Get Mode	Operation
GFS_ERASE	Erases information from CD buffer after transfer to host area.
GFS_RESIDENT	Doesn't erase information from CD buffer after transfer to host area.

In the GFS_RESIDENT mode, access is faster when the same information is used over and over.

Title	Function	Function Name	No
Function Specification	Sets Transfer Mode (Software/DMA, etc.)	GFS_SetTmode	12

[Format] void GFS_SetTmode(GfsHn gfs, Sint32 tmode)
[Input] gfs: File Handle
tmode: Transfer Mode
[Output] None
[Function value] None
[Function]

Sets the method of transfer from the CD buffer. Depending on the transfer mode, the following will change: bus occupation time, invalid interrupts, or transfer rate.

Constants and transfer methods that can be designated by tmode are shown below:

Transfer Mode	Transfer Method	Load on the CPU
GFS_TRANS_SCU	DMA SCU	If transfer is on bus B, CPU is fully independent.
GFS_TRANS_BDMA0	DMA burst channel 0	CPU stops
GFS_TRANS_BDMA1	DMA burst channel 1	CPU stops
GFS_TRANS_SDMA0	DMA Cycle Steal channel 0	CPU capacity drops
GFS_TRANS_SDMA1	DMA Cycle Steal channel 1	CPU capacity drops
GFS_TRANS_CPU	Software	Occupies the CPU but interrupt processing is allowed.

[Reference] GFS_SetTransPara Function

2.3 Directory Control

Title	Function	Function Name	No
Function Specification	Reads Directory Information	GFS_LoadDir	13

[Format] Sint32 GFS_LoadDir(GfsFid fid, GfsDirTbl *dirtbl)
[Input] fid: Directory File Identifier
[Output] dirtbl: Directory Information Set Area
[Function value] Number of directories read
[Function]

Reads the directory information from the directory file to the directory information set in area dirtbl 1. The user must secure the directory information area.

Before calling this function, the following members must be set.

```
GFS_DIRTBL_TYPE(dirtbl)   Directory information area type
GFS_DIRTBL_NDIR(dirtbl)   Maximum directory elements in the
                          directory information area
GFS_DIRTBL_DIR(dirtbl)    Directory information area address
```

When NULL is selected, the selected directory data will be read to file management data in the CD block. When using the debugging library, memory files and DOS files are automatically read. If there are two files with the same name, the directory information of the file with lower priority (according to the chart below) is overwritten.

	Priority	File Type
High ↑	1	Memory File
	2	DOS File
↓ Low	3	CD File

Title	Function	Function Name	No
Function Specification	Sets Current Directory	GFS_SetDir	14

[Format] void GFS_SetDir(GfsDirTbl *dirtbl);
[Input] dirtbl: Directory Information Control Data
[Output] None
[Function value] None
[Function]

Makes the directory information control data the current directory. After this file has been executed, the designated file identifier indicates the current directory order number. In the debugging library, the GFS_DIRTBL_PATH(dirtbl) sets the current directory path name. If dirtbl is set as NULL, the CD block file control information is set as the current directory.

Title	Function	Function Name	No
Function Specification	Converts from File Name to File Identifier	GFS_NameToId	15

[Format] GfsFid GFS_NameToId(Uint8 *fname)
[Input] fname: File Name
[Output] None
[Function value] File Identifier
[Function]

Returns the file identifier corresponding to the file name.

[Example]

```
GfsDirName dirtbl[FILE_MAX];                          /* Conversion table area */

ndir = GFS_LoadDir(dirtbl, FILE_MAX, dir_pos, ndir); /* Create conversion table */

fid = GFS_NameToId(dirtbl, ndir, "sprite.dat");       /* Gets the identifier */

gfs = GFS_Open(fid);                                  /* Open File */
```

[Comments]

If the GFS_SetDir function in the GFS_DIR_NAME directory information is not called, error processing will be called.

Title	Function	Function Name	No
Function Specification	Converts from File Identifier to File Name	GFS_IdToName	16

[Format] Uint8 *GFS_IdToName(GfsFid fid)
[Input] fid: File Identifier
[Output] None
[Function value] Pointer to the File Name
[Function]

Returns the pointer to the file name corresponding to the file identifier. The pointer shows an area within the conversion table.

[Comments]

If the GFS_SetDir function in the GFS_DIR_NAME directory information is not called, error processing will be called.

2.4 High-Level End Return

Title	Function	Function Name	No
Function Specification	Reads Data	GFS_Fread	17

[Format] Sint32 GFS_Fread(GfsHn gfs, Sint32 nsct, void *buf, Sint32 bsize)

[Input] gfs: File handle
nsct: Number of sectors to be read
bsize: Size of the area to be read (byte count: must be even number)

[Output] buf: Area to be read

[Function value] Actual number of bytes read

[Function]

Reads nsct sector area data from the position of the access pointer, and doesn't return until read is finished. The sector part is read, but data that exceeds bsize bytes are thrown out. The access pointer then advances to the nsct sector address.

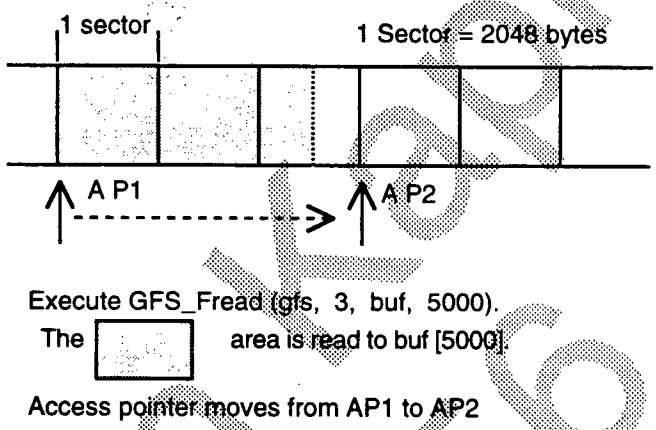

Figure 2.4.1 Example of the Read Operation

[Example]

Data is read from the sector 30 to buf [BUF_SIZE]

```
gfs  =  GFS_Open (fid) ;
GFS_Seek (gfs,  30, GFS_SEEK_SET) ;    / * move access pointer to sector 30* /
nsct = GFS_ByteToSct (BUF_SIZE) ;      / * Get sector count      * /
GFS_Fread (gfs, nsct, buf, BUF_SIZE); / * Does not return until read if finished* /

GFS_Close (gfs) ;
```

Title	Function	Function Name	No
Function Specification	Writes Data	GFS_Fwrite	18

[Format] Sint32 GFS_Fwrite(GfsHn gfs, Sint32 nsct, void *buf, Sint32 bsize)
[Input] gfs: File handle
nsct: Sector Count
bsize: Size of the area to be written (byte count: must be even number)
[Output] buf: Data to be written
[Function value] Actual number of bytes written
[Function]

Writes bsize bytes of data from the position of the access pointer to nsct sector area, and doesn't return until write is finished. The access pointer progresses to the front of the nsct sector, and is only valid for partial RAM. If the file can't be written, the error processing function is called up.

2.5 High-Level Immediate Return

Title	Function	Function Name	No
Function Specification	Starts Data Read	GFS_NwFread	19

[Format] void GFS_NwFread(GfsHn gfs, Sint32 nsct, void *buf, Sint32 bsize)
[Input] gfs: File handle
nsct: Number of sectors to be read
bsize: Size of the area to be read (byte count: must be even number)
[Output] buf: Area to be read
[Function]

Starts reading data from the position of the access pointer; sets internal variables only and returns immediately. Reading the CD buffer and transferring to the host area are done with the GFS_NwExecOne and GFS_NwExecServer functions. When the access operation is finished, the access pointer advances nsct sectors.

[Comments]

If this function is called before the designated input file is completely accessed, an error will occur.

Title	Function	Function Name	No
Function Specification	Starts Reading CD Buffer	GFS_NwCdRead	20

[Format] void GFS_NwCdRead(GfsHn gfs, Sint32 nsct)
[Input] gfs: File handle
nsct: Number of sectors to be read
[Output] None
[Function value] None
[Function]

Starts reading data from the position of the access pointer to the CD buffer; sets internal variables only and returns immediately. Reading the CD buffer and transferring to the host area are done with the GFS_NwExecOne and GFS_NwExecServer functions. The access pointer does not change.

[Comments]

If this function is called before the designated input file is completely accessed, an error will occur.

Title	Function	Function Name	No
Function Specification	Starts Data Write	GFS_NwFwrite	21

[Format] void GFS_NwFwrite(GfsHn gfs, Sint32 nsct, void *buf, Sint32 bsize)

[Input]
- gfs: File handle
- nsct: Sector count
- bsize: Size of the data (byte count: must be even number)

[Output]
- buf: Area to write data to

[Function]

Starts writing data from the position of the access pointer; sets internal variables only and returns immediately. Actual writing is done with the GFS_NwExecOne and GFS_NwExecServer functions. When the access operation is finished, the access pointer advances nsct sectors.

[Comments]

If this function is called before the designated input file is completely accessed, an error will occur.

Title	Function	Function Name	No
Function Specification	Checks Access Operation End	GFS_NwIsComplete	22

[Format] Bool GFS_NwIsComplete(GfsHn gfs)

[Input]
- gfs: File handle

[Output] None

[Function value] Access operation State
- TRUE End
- FALSE Operating

[Function]

Checks for end of operation.

[Example]

Data from the 30 sector is read to buf [BUF_SIZE].

```
gfs = GFS_Open (fid) ;
GFS_Seek (gfs,  30,  GFS_SEEK_SET) ;
nsct = GFS_ByteToSct (BUF_SIZE) ;                 / *  get sector number    * /
GFS_NwFread ( gfs, nsct, buf, BUF_SIZE) ;         / *  return immediately    * /
while  ( !GFS_NwIsComplete (gfs)  )  {
     GFS_NwExecOne (gfs) ;                        / *  actual read operation * /
     user  ( ) ; / *  other user processes        * /
}
GFS_Close (gfs) ;
```

Title	Function	Function Name	No
Function Specification	Stops Access Operation	GFS_NwStop	23

[Format] Sint32 GFS_NwStop(GfsHn gfs)
[Input] gfs: File handle
[Output] None
[Function value] Value of the access pointer at the point it stops
[Function]
Stops the access operation.
Returns to the position where access was stopped.

Title	Function	Function Name	No
Function Specification	Stops Access Operation	GFS_NwGetStat	24

[Format] void GFS_NwGetStat(GfsHn gfs, Sint32 *stat, Sint32 *ndata)
[Input] gfs: File handle
[Output] stat: status of current access operation
ndata: data count
[Function value] None
[Function]
Gets status of current access operation.
Access status is a constant name, as shown below.

Access Status	Processing in Progress Function	What Data Count ndata means
GFS_NWSTAT_NOACT	None	No meaning
GFS_NWSTAT_FREAD	GFS_Fread	Number of bytes read in the host area
GFS_NWSTAT_CDREAD	GFS_CdRead	Number of sectors read in the CD buffer
GFS_NWSTAT_FWRITE	GFS_Fwrite	Number of bytes written

Title	Function	Function Name	No
Function Specification	Executes File Level Access Operation	GFS_NwExecOne	25

[Format] Sint32 GFS_NwExecOne(GfsHn gfs)
[Input] gfs: File handle
[Output] None
[Function value] Current Execution Status
[Function]

This function performs Immediate return access.
The following is conducted after the access operation called prior to this one.

Access Operation	Processing
GFS_NwFread	Reads CD buffer and transfers data to the host.
GFS_NwCdRead	Reads to the CD buffer.
GFS_NwFwrite	(Undetermined)

The constant name and meaning of the current execution status are shown below.

Exec Status	Meaning
GFS_EXEC_COMPLETE	File access has ended
GFS_EXEC_DOING	Accessing File
GFS_EXEC_CDPAUSE	The CD buffer is full and reading is paused.

[Comments]

The amount of data read to the CD buffer in one read is set in the GFS_SetReadPara function.
The amount of data transferred from the CD buffer in one transfer is set in the GFS_SetTransPara function.

Title	Function	Function Name	No
Function Specification	Executes Access of All Files	GFS_NwExecServer	26

[Format] Sint32 GFS_NwExecServer(GfsHn *now_gfs)
[Input] None
[Output] now_gfs: File handle of target file(s)
[Function value] Current Access Status of the Server
[Function]

Files are accessed (GFS_NwExecOne Function) in the order the access operation was started. When one file has finished, the next file is started. As a result, other user processing and file access procedures can be done simultaneously. The current access server returns File Handle during processing to *now_gfs.

Access Server Status	Meaning
GFS_SVR_NOFILES	All files have been accessed.
GFS_SVR_EXEC	Currently accessing files.
GFS_SVR_CDPAUSE	CD buffer is full, paused reading.

[Example]

The following is an example of reading two files simultaneously.

```
gfs 1 = GFS_Open(fid1);
gfs 2 = GFS_Open(fid2);
nsct  = GFS_ByteToSct(BUF_SIZE);          /*Gets the number of sectors */
GFS_NwFread(gfs1, nsct, buf1, BUF_SIZE);  / * Returns immediately  * /
GFS_NwFread(gfs2, nsct, buf2, BUF_SIZE);  / * Returns immediately  * /
for  (; ;)  {
  stat  =  GFS_NwExecServer(&now_gfs);    / * Actual read operation * /
  if (stat == GFS_SVR_NOFILES)   {
      break;
  }
  user ( ) ;                              / * Other user processing  * /
}
  GFS_Close(gfs1);
  GFS_Close(gfs2);
```

Title	Function	Function Name	No
Function Specification	Sets Read Amount from CD Buffer	GFS_SetReadPara	27

[Format] void GFS_SetReadPara(GfsHn gfs, Sint32 cdrsize)
[Input] gfs: File Handle
cdrsize: Number of sectors that can be read to the CD buffer.
[Output] None
[Function value] None
[Function]

Sets the maximum value that can be read to the CD buffer at one time in high-level.

Title	Function	Function Name	No
Function Specification	Sets Transfer Amount from CD Buffer	GFS_SetTransPara	28

[Format] void GFS_SetTransPara(GfsHn gfs, Sint32 tsize)
[Input] gfs: File Handle
tsize: Number of sectors that can be transferred to the designated area at one time.
[Output] None
[Function value] None
[Function]

Sets the amount of data that can be transferred to the destination area at one time.

[Comments]

If the GFS_Open function is called, tsize is 1 sector.
Reading to and from the CD buffer is repeated in units of tsize.

2.7 Low-Level

Title	Function	Function Name	No
Function Specification	Reads to CD Buffer	GFS_CdRead	29

[Format] void GFS_CdRead(GfsHn gfs, Sint32 nsct)
[Input] gfs: File Handle
nsct: Number of sectors read
[Output] None
[Function value] None
[Function]

Reads to the CD buffer (from the CD) the number of sectors in nsct. If the pickup is in operation when another file executes either the GFS_CdRead or GFS_CdSeek function, an error will occur.

Title	Function	Function Name	No
Function Specification	Stops Read to the CD Buffer	GFS_CdStopRead	30

[Format] void GFS_CdStopRead(GfsHn gfs)
[Input] gfs: File Handle
[Output] None
[Function value] None
[Function]

Stops read to the CD buffer.

Title	Function	Function Name	No
Function Specification	Number of Sectors Read to the CD Buffer	GFS_CdGetRcnt	31

[Format] Sint32 GFS_CdGet Rcnt(GfsHn gfs)
[Input] gfs: File Handle
[Output] None
[Function value] None
[Function]

Returns the value of the current number of sectors read by the GFS_CdRead function.

Title	Function	Function Name	No
Function Specification	Transfers Data from the CD Buffer	GFS_CdTrans	32

[Format] void GFS_CdTrans(GfsHn gfs, Sint32 nsct, void *buf, Sint32 bsize)
[Input] gfs: File Handle
nsct: Number of sectors read from the CD buffer
bsize: Size of the destination area (bytes: must be even)
[Output] buf: Transfer destination area
[Function value] None
[Function]

Transfers data from the CD buffer to buf.
Data is read from the CD buffer by nsct sectors, but only bsize bytes are transferred to the host area.
When a transfer is over, the access pointer advances nsct sectors.
If another file is transferring data, an error will occur.

Title	Function	Function Name	No
Function Specification	Stops Data Transfer from CD Buffer	GFS_CdStopTrans	33

[Format] void GFS_CdStopTrans(GfsHn gfs)
[Input] gfs: File Handle
[Output] None
[Function value] None
[Function]

Stops transfer from the CD buffer to the host area.

Title	Function	Function Name	No
Function Specification	Checks whether reading or not	GFS_CdIsRead	34

[Format] Bool GFS_CdIsRead(GfsHn gfs)
[Input] gfs: File Handle
[Output] None
[Function value] Returns read condition
TRUE: Reading
FALSE: Not Reading
[Function]

Checks whether reading CD buffer or not.

Title	Function	Function Name	No
Function Specification	Checks whether Transferring or not	GFS_CdIsTrans	35

[Format] Bool GFS_CdIsTrans(GfsHn gfs)
[Input] gfs: File Handle
[Output] None
[Function value] Returns the transfer status
TRUE: Transferring
FALSE: Not Transferring

[Function]

Checks to see if the GFS_CdTrans function is transferring.

[Example]

Read data from the 30 sector to buf[BUF_SIZE]

```
gfs = GFS_Open (fid);
GFS_Seek (gfs,  30,  GFS_SEEK_SET) ;
nsct = GFS_ByteToSct(BUF_SIZE) ;              / *  get sector count    * /
GFS_CdRead (gfs,  nsct) ;                     / *  read command to CD buffer * /
for  (ntrn=0;  ntrn<BUF_SIZE;  )  {
  if  ( GFS_CdGetRcnt(gfs)  >  0  )  {        / *  read to CD buffer ?      * /
     if  (! GFS_CdIsTrans (gfs)  )  {         / *  is transferring in progress? * /
           tsize = min(sct_size,  BUF_SIZE  -  ntrn)  ;
           GFS_CdTrans (gfs,  1,  &buf[ntrn],  tsize) ;     / * transferring * /
           ntrn += tsize ;
     }
     }
     user ( ) ; / *  other user processes  * /
  }
  GFS_Close (gfs) ;
```

Title	Function	Function Name	No
Function Specification	Moves CD Pickup	GFS_CdMovePickup	36

[Format] void GFS_CdMovePickup(GfsHn gfs)
[Input] gfs: File Handle
[Output] None
[Function value] None
[Function]

Moves the CD pickup to the access pointer's position.
Used to shorten the pickup seek time when using the GFS_Read, GFS_NwRead or GFS_CdRead functions.
If the pickup is in operation because functions GFS_CdRead or GFS_CdSeek are operating in another file, an error will result.

Title	Function	Function Name	No
Function Specification	Gets the Operation Status of the CD Buffer	GFS_CdGetStat	37

[Format] void GFS_CdGetStat(GfsHn *rdgfs, Sint32 *rdstat, GfsHn *trgfs, Sint32 *trstat)

[Input] None

[Output]
- rdgfs: File handle that controls the CD pickup
- rdstat: Pickup operational status
  - GFS_STAT_READ: Reading
  - GFS_STAT_SEEK: Moving
  - GFS_STAT_NOACT: Not Moving
- trgfs: File handle transferring data from the CD buffer
- trstat: Transfer status
  - GFS_STAT_TRANS: Transferring
  - GFS_STAT_NOACT: Not Moving

[Function value] None

[Function]

Returns the status of the CD buffer to operational.

2.8 Error Handling

Title	Function	Function Name	No
Function Specification	Sets the Error Processing Function	GFS_SetErrorFunc	38

[Format] void GFS_SetErrFunc(GfsErrFunc func, void *obj)
[Input] func: Functions called when there is an error.
obj: First factor of the func function
[Output] None
[Function value] None
[Function]

Records the function called when there is an error.
When an error occurs, the following function executes if it is set.

```
(*func)(obj, err_code);
```

Note: If not set, nothing will execute.

Error code (err_code) details are undetermined.

Title	Function	Function Name	No
Function Specification	Gets Error Status	GFS_GetErrStat	39

[Format] GfsErr *GFS_GetErrStat(void)
[Input] None
[Output] None
[Function value] Pointer to the structure that saves the error status
[Function]

Gets the pointer that indicates which structure the error status is saved in.
This structure provides the conditions surrounding the error.
Details are undetermined.