Ambarella

SDK6 API: File System (AmbaFS)

Version 1.5

January 27, 2015

Ambarella

Confidentiality Notice:

The contents of this document are proprietary and confidential information of Ambarella Inc.

The material in this document is for information only. Ambarella assumes no responsibility for errors or omissions and reserves the right to change, without notice, product specifications, operating characteristics, packaging, ordering, etc. Ambarella assumes no liability for damage resulting from the use of information contained in this document. All brands, product names, and company names are trademarks of their respective owners.

US
3101 Jay Street
Ste. 110
Santa Clara, CA 95054, USA
Phone: +1.408.734.8888
Fax: +1.408.734.0788

Taiwan
Suite C1, No. 1, Li-Hsin Road 1
Science-Based Industrial Park
Hsinchu 30078, Taiwan
Phone: +886.3.666.8828
Fax: +886.3.666.1282

Hong Kong
Unit A&B, 18/F, Spectrum Tower
53 Hung To Road
Kwun Tong, Kowloon
Phone: +85.2.2806.8711
Fax: +85.2.2806.8722

Japan - Yokohama
Shin-Yokohama Business Center Bldg. 5th Floor
3-2-6 Shin-Yokohama, Kohoku-ku,
Yokohama, Kanagawa, 222-0033, Japan
Phone: +81.45.548.6150
Fax: +81.45.548.6151

Korea
6 Floor, Hanwon-Bldg.
Sunae-Dong, 6-1, Bundang-Gu
SeongNam-City, Kyunggi-Do
Republic of Korea 463-825
Phone: +031.717.2780
Fax: +031.717.2782

China - Shenzhen
Unit E, 5th Floor
No. 2 Finance Base
8 Ke Fa Road
Shenzhen, 518057, China
Phone: +86.755.3301.0366
Fax: +86.755.3301.0966

China - Shanghai
9th Floor, Park Center
1088 Fangdian Road, Pudong New District
Shanghai 201204, China
Phone: +86.21.6088.0608
Fax: +86.21.6088.0366

I Contents

II Preface

This document provides technical details using a set of consistent typographical conventions to help the user differentiate key concepts at a glance.

Conventions include:

Example	Description
AmbaGuiGen, **DirectUSB** **Save, File > Save** **Power, Reset**, **Home**	Software names GUI commands and command sequences Computer / Hardware buttons
Flash_IO_control **da**, **status**, **enable**	Register names and register fields. For example, **Flash_IO_control** is the register for global control of Flash I/O, and bit 17 (**da)** is used for DMA acknowledgement.
GPIO81, **CLK_AU**	Hardware external pins
VIL, VIH, VOL, VOH	Hardware pin parameters
INT_O, RXDATA_I	Hardware pin signals
amb_performance_t ***amb_operating_mode_t*** **amb_set_operating_mode()**	API details (e.g., functions, structures, and type definitions)
`/usr/local/bin` `success = amb_set_operating_mode (amb_base_address, & operating_mode)`	User entries into software dialogues and GUI windows File names and paths Command line scripting and Code

Table II-1. Typographical Conventions for Technical Documents.

Additional Ambarella typographical conventions include:

- Acronyms are given in UPPER CASE using the default font (e.g., AHB, ARM11 and DDRIO).
- Names of Ambarella documents and publicly available standards, specifications, and databooks appear in *italic* type.

1 Overview

1.1 Overview: Introduction

This document defines the Ambarella File System (AmbaFS) application programming interface (API) module for AXX digital processing products.

1.2 Overview: Scope of Document

This document focuses strictly on the AXX File System API. Users of this document are assumed to be familiar with the AXX chip hardware, system capabilities, software architecture, and reference applications. The reader is referred to the following for a background overview:

- The chip AXX datasheet provides hardware pin and package details including a feature list with descriptions of chip performance, brief interface descriptions, a complete power-on configuration table, and electrical characteristics.
- The "*AXX Hardware Programming Reference Manual*" is the primary resource for programming peripheral drivers. It lists software-programmable registers accessible from CPU cores, including detailed information on each field of a register. It also provides overviews of the system memory map, power-on configuration options, and ARM interrupts.
- "*AXX RM: System Hardware*" covers power-on timing and sequencing. It provides pin connection details including guidance for unused interfaces and PCB layout guidelines.

2 File System

2.1 File System: Overview

This chapter details the Ambarella File System (AmbaFS) API functions.

2.2 File System: Function List

- AmbaFS_SetBufferingMode
- AmbaFS_Chdir
- AmbaFS_Chdmod
- AmbaFS_Chmod
- AmbaFS_DeleteDir
- AmbaFS_fclose
- AmbaFS_feof
- AmbaFS_fopen
- AmbaFS_Format
- AmbaFS_fread
- AmbaFS_fseek
- AmbaFS_FirstDirEnt
- AmbaFS_NextDirEnt
- AmbaFS_Stat
- AmbaFS_FSync
- AmbaFS_fwrite
- AmbaFS_GetBufferingMode
- AmbaFS_GetDev
- AmbaFS_GetVol
- AmbaFS_fappend
- AmbaFS_Init
- AmbaFS_Mkdir
- AmbaFS_Move
- AmbaFS_remove

- AmbaFS_rename
- AmbaFS_Rmdir
- AmbaFS_SetVol
- AmbaFS_Sync
- AmbaFS_CleanDir
- AmbaFS_ChmodDir
- AmbaFS_Cinsert
- AmbaFS_Cdelete
- AmbaFS_Combine
- AmbaFS_Divide
- AmbaFS_Mount
- AmbaFS_UnMount
- AmbaFS_GetError

2.2.1 AmbaFS_SetBufferingMode

API Syntax:

int **AmbaFS_SetBufferingMode** (char Drive, int Mode)

Function Description:

- The **AmbaFS_SetBufferingMode** function is used to set the File Allocation Table (FAT) buffer and the data buffer read/write-back mode.
- The FAT buffer is an area used by the AMBA File System to internally cache FAT data when the FAT is being read or written-to. The data buffer is an area used by the AMBA File System to internally cache file data. The data buffer can also be used as the directory entry area when reading or writing file data.
- The **AmbaFS_SetBufferingMode** function determines whether or not to immediately write data to media when the data is updated by calling an API. The data may be written to media when the directory entry is changed by the **AmbaFS_fopen** function or when data is refreshed by the **AmbaFS_fread** function or the **AmbaFS_fwrite** function.
- This function will return 0 when it successfully sets the write mode and will return -1 when it fails.
- Call the API **AmbaFS_GetError** to retrieve error information.

Parameters:

Type	Parameter	Description
char	**Drive**	Drive name to be specified (Example: 'A')
int	**Mode**	Read / Write back mode: (See Table 2-2 for additional detail) 0x00: **AMBA_FS_WRITE_BACK_CACHE** (Write back mode) 0x01: **AMBA_FS_EJECT_SAFE_WITH_WRITE_THRU** (Eject-Safe write through mode) 0x02: **AMBA_FS_WRITE_BACK_ON_SIGNIF_API** (Automatic write back mode) 0x06: **AMBA__FS_EJECT_SAFE_WITH_WRITE_BACK** (Eject-Safe write back mode)

*Table 2-1. Parameters for File System API **AmbaFS_SetBufferingMode()**.*

Mode	FAT	Directory, Data
AMBA_FS_EJECT_SAFE_WITH_WRITE_THRU	Write through	Write through
AMBA_FS_EJECT_SAFE_WITH_WRITE_BACK	Write through	Write back (Auto)
AMBA_FS_WRITE_BACK_ON_SIGNIF_API	Write back (Auto)	Write back (Auto)
AMBA_FS_WRITE_BACK_CACHE	Write back (optional)	Write back (Optional)

*Table 2-2. File System API **AmbaFS_SetBufferingMode()** parameter **Mode** detail.*

- When **AMBA_FS_EJECT_SAFE_WITH_WRITE_THRU** is set, data is written immediately to media if an update of one byte occurs. The EjectSafe feature is enabled for all data writing.

- When **AMBA_FS_EJECT_SAFE_WITH_WRITE_BACK** is set, data is written immediately to media when the FAT is written. The directory entry or file data is buffered to the data buffer and written back to media when the file is closed by calling the API **AmbaFS_fclose**. The EjectSafe feature is enabled for all data writing.
- When **AMBA_FS_WRITE_BACK_ON_SIGNIF_API** is set, all data is buffered and written back to media when the file is closed by calling **AmbaFS_fclose** function.
- When **AMBA_FS_WRITE_BACK_CACHE** is set, all data is buffered and will not be written back to media even when the file is closed (**AmbaFS_fclose** function). To write back data, the **AmbaFS_Sync** function or the **AmbaFS_FSync** function must be called.
- There are two parameters for the API **AmbaFS_SetBufferingMode**. The drive name to configure the write mode must be designated in the first parameter, **Drive**. This drive name is not case-sensitive.
- Set the second parameter **mode** to specify the read/write back mode. The four types of parameters described in the previous paragraphs can be set in the parameter **Mode**. The default setting at the startup of AmbaFS File System is **AMBA_FS_WRITE_BACK_ON_SIGNIF_API ()**.

Returns:

Return	Description
0	Completed successfully
-1	Write mode configuration failed

*Table 2-3. Returns for File System API **AmbaFS_SetBufferingMode()**.*

Example:

```
int Rval;
int Slot = SCM_SLOT_SD;

/* Set buffering on Drive */
AmbaPrintf("--- buffering ---");
Rval = AmbaFS_SetBufferingMode('a' + Slot,
AMBA_FS_EJECT_SAFE_WITH_WRITE_BACK);
if (Rval == -1) {
    AmbaPrintf("buffering failed (%d)", Rval);
} else {
    AmbaPrintf("buffering %c Drive",'a' + Slot);
}
```

See Also:

None

2.2.2 AmbaFS_Chdir

API Syntax:

int **AmbaFS_Chdir** (const char *pDirName)

Function Description:

- **AmbaFS_Chdir** is a function used to change the current directory name.
- Set the parameter **pDirName** pointer to the directory name to be changed. The directory designation is the same as the file name designation for the created function.
- When changing the current directory among drives, consider this example that changes the current directory to the A drive:

  AmbaFS_Chdir("A:\\"): Change to the root directory of the A drive
  AmbaFS_Chdir("A:/"): Change to the root directory of the A drive
  AmbaFS_Chdir("A:"): Change to the previous directory of the A drive
  AmbaFS_Chdir("A:\\ABC"): Change to "A:\\ABC" or "A:/ ABC"
  AmbaFS_Chdir("A:/ ABC"): Change to "A:\\ABC" or "A:/ ABC"
- The **AmbaFS_Chdir** function returns 0 when it successfully changes the current directory and returns -1 when it fails.

Parameters:

Type	Parameter	Description
const char*	**pDirName**	Pointer to path and directory name

*Table 2-4. Parameters for File System API **AmbaFS_Chdir()**.*

Returns:

Return	Description
0	Success
-1	Failed to change current directory

*Table 2-5. Returns for File System API **AmbaFS_Chdir()**.*

Example:

```
char path[128];
int Slot = SCM_SLOT_SD;
sprintf(path, "%c:\\test", 'a'+ Slot);
AmbaFS_Mkdir(path);
Rval = AmbaFS_Chdir(path);
if (Rval < 0)
    AmbaPrintf("chdir  fail");
```

See Also:

None

2.2.3 AmbaFS_Chdmod

API Syntax:

int **AmbaFS_Chdmod** (const char *pDirName, int Attr)

Function Description:

- The **AmbaFS_Chdmod** API is a function used to change the directory attribute information.
- There are two parameters.
  - Specify the first parameter, **pDirName**, as a pointer to the directory name and the path. Refer to the **AmbaFS_Mkdir** function for information on designating a directory name. Note that an entry without an actual file name or directory name, such as "." or "..", that exists under a subdirectory may be designated as a directory name.
  - The new attribute information must be specified in the second parameter, **pMode**. Multiple attributes can be combined by using the OR operator (|). The **AMBA_FS_ ATTR_DIR** attribute bit will not be cleared even when the specification of the **AMBA_FS_ ATTR_DIR** attribute is not included in the **Attr** parameter. For example, when only the **AMBA_FS_ ATTR_ARCH** attribute is specified, the directory will have two attributes, **AMBA_FS_ATTR_ARCH** and **AMBA_FS_ATTR_DIR**. If **AMBA_FS_ATTR_NONE** (0x00) is specified in **pMode**, the directory has only the **AMBA_FS_ ATTR_DIR** attribute.
- The **AmbaFS_Chdmod** function returns 0 when the change to the directory attribute is successful and returns -1 when it fails.

Parameters:

Type	Parameter	Description
const char*	**pDirName**	Pointer to path and file name
int	**Attr**	Directory attribute: 0x00: **AMBA_FS_ATTR_NONE** (No attribute) 0x01: **AMBA_FS_ATTR_RDONLY** (Read only) 0x02: **AMBA_FS_ATTR_HIDDEN** (Hidden file) 0x10: **AMBA_FS_ATTR_DIR** (Subdirectory) 0x20: **AMBA_FS_ATTR_ARCH** (Archive)

*Table 2-6. Parameters for File System API **AmbaFS_Chdmod()**.*

Returns:

Return	Description
0	Success
-1	Directory attribute change failed

*Table 2-7. Returns for File System API **AmbaFS_Chdmod()**.*

Example:

```
int rval;

char path[256] ;
int Slot = SCM_SLOT_SD;

sprintf(path, "%c:\\test", 'a' + Slot);
AmbaFS_Chdmod(path, AMBA_FS_ATTR_RDONLY);
```

See Also:

None

2.2.4 AmbaFS_Chmod

API Syntax:

int **AmbaFS_Chmod** (const char *pFileName, int Attr)

Function Description:

- The **AmbaFS_Chmod** function changes the attribute information of a specified file.
- There are two parameters. The first parameter, **pFileName**, is a pointer to the path and file name. The new attribute information must be specified in the second parameter **Attr**.
- When specifying multiple attributes, the attributes can be combined by using the OR operator (|).
- If **pMode** is specified as 0, the file will have no assigned attributes.
- The **AmbaFS_Chmod** function returns 0 when it succeeds in changing the file attribute and returns -1 when it fails.
- The attribute cannot be changed while the designated file is open. This function cannot change the attribute of a directory. Wildcard characters ("*" and "?") cannot be used to specify the file name.

Parameters:

Type	Parameter	Description
const char*	**pFileName**	Pointer to path and file name
int	**Attr**	Directory attribute: 0x00: **AMBA_FS_ATTR_NONE** (No attribute) 0x01: **AMBA_FS_ATTR_RDONLY** (Read only) 0x02: **AMBA_FS_ATTR_HIDDEN** (Hidden file) 0x04: **AMBA_FS_ATTR_SYSTEM** (System file) 0x20: **AMBA_FS_ATTR_ARCH** (Archive)

*Table 2-8. Parameters for File System API **AmbaFS_chmod()**.*

Returns:

Return	Description
0	Success
-1	File attribute change failed

*Table 2-9. Returns for File System API **AmbaFS_chmod()**.*

Example:

```
int Rval;
char path[256];
int Slot = SCM_SLOT_SD;
sprintf(path, "%c :\\test.txt", 'a' + Slot);

/* Change file permission */

AmbaPrintf("--- chmod test dir ---");
Rval = AmbaFS_Chmod(path, AMBA_FS_ATTR_HIDDEN);
if (Rval) {
AmbaPrintf("chmod failed (%d)", Rval);
}
else {
AmbaPrintf("chdmod passed");
}
```

See Also:

None

2.2.5 AmbaFS_DeleteDir

API Syntax:

int **AmbaFS_DeleteDir** (const char *pDirName)

Function Description:

- **AmbaFS_DeleteDir** is a function used to delete a directory (Unicode supported).
- Set the parameter **pDirName** to point to the name of the directory to be deleted.
- The **AmbaFS_DeleteDir** function returns 0 when it successfully deletes the directory and returns -1 when it fails.
- If the directory specified by **pDirName** is the root directory, current directory, parent directory, or a file or directory existing in the specified directory, then the directory cannot be deleted, and an error is returned.
- If the directory specified by **pDirName** is open or a directory with the read-only (**AMBA_FS_ATTR_RDONLY**) attribute, **AmbaFS_DeleteDir** function returns an error without deleting the directory.
- Wildcard characters (“*” and “?”) cannot be used to specify the directory name.

Parameters:

Type	Parameter	Description
const char*	**pDirName**	Pointer to path and directory name

*Table 2-10. Parameters for File System API **AmbaFS_DeleteDir()**.*

Returns:

Return	Description
0	Success
< 0	Failure

*Table 2-11. Returns for File System API **AmbaFS_DeleteDir()**.*

Example:

```
char path[128];

path = "d:\\zzz";
AmbaFS_DeleteDir(path);
```

See Also:

None

2.2.6 AmbaFS_fclose

API Syntax:

int **AmbaFS_fclose** (AMBA_FS_FILE *pFile)

Function Description:

- The **AmbaFS_fclose** function is used to close an open file.
- The file descriptor of the file to be closed should be specified in the parameter **pFile**. Use the file descriptor obtained by the **AmbaFS_fopen** function.
- The **AmbaFS_fclose** function returns 0 when the file specified in the parameter stream is closed and returns -1 when the operation fails.

Parameters:

Type	Parameter	Description
AMBA_FS_FILE*	**pFile**	Pointer to file descriptor

*Table 2-12. Parameters for File System API **AmbaFS_fclose()**.*

Returns:

Return	Description
0	Completed successfully
-1	File close failed

*Table 2-13. Returns for File System API **AmbaFS_fclose()**.*

Example:

```
AMBA_FS_FILE *pFile;
UINT64 SuccessSize;
unsigned char buff[10];


/* open a sample file */
pFile =  AmbaFS_fopen("A:\\sample.txt", "w+");
/* write data to the sample file */
SuccessSize = AmbaFS_fwrite((void*)FOOBAR, 6, 1, pFile);
…
/* seek file pointer to start of the sample file */
AmbaFS_fseek(pFile, 0, AMBA_FS_SEEK_SET);
/* read data from the sample file */
SuccessSize = AmbaFS_fread((void*)&buff, 6, 1, pFile);
…
/* close the sample file */
AmbaFS_fclose(pFile);
```

See Also:

None

2.2.7 AmbaFS_feof

API Syntax:

int **AmbaFS_feof** (AMBA_FS_FILE *pFile)

Function Description:

- The **AmbaFS_feof** function checks whether the end of the file has been reached.
- The file descriptor returned from the **AmbaFS_fopen** function must be specified in the parameter **pFile**.
- The **AmbaFS_feof** function returns 1 when the file position indicator has reached the end of the file or if it has been moved to the end or beyond the end with the **AmbaFS_fseek** function.
- The **AmbaFS_feof** function returns 0 when the file position indicator is located before the end of the file.
- The **AmbaFS_feof** function returns -1 when the file is not opened or the file descriptor is invalid. Note that a subsequent call to the **AmbaFS_GetError** function will not return information relating to this error.

Parameters:

Type	Parameter	Description
AMBA_FS_FILE*	**pFile**	Pointer to file descriptor

*Table 2-14. Parameters for File System API **AmbaFS_feof()**.*

Returns:

Return	Description
-1	The file is not open or invalid file descriptor
0	The I/O pointer has not reached the end of the file
1	The I/O pointer has reached the end of the file

*Table 2-15. Returns for File System API **AmbaFS_feof()**.*

Example:

```
AMBA_FS_FILE *pFile;
AMBA_FS_STAT stat;
UINT64 SuccessSize;
int Result;
…
…
/* get the file status for a sample file */
AmbaFS_Stat("A:\\sample.txt", &stat);
…
/* open a sample file */
pFile = AmbaFS_fopen("A:\\sample.txt", "a+");
…
/* check whether the file pointer is at the end of file */
Result = AmbaFS_feof(pFile);
if (Result != 1)
    AmbaPrintf("feof failed");
…
…
/* close the sample file */
AmbaFs_fclose(pFile);
…
…
```

See Also:

None

2.2.8 AmbaFS_fopen

API Syntax:

AMBA_FS_FILE* **AmbaFS_fopen** (const char *pFileName, const char *pMode)

Function Description:

- The **AmbaFS_fopen** function opens an existing file or creates a new file and opens it.
- There are two parameters. The first parameter, **pFileName**, is a pointer to the buffer that contains the file name and path for the file to be opened.
- The access mode is specified using the **pMode** string:
  - **r** opens a file for reading.
    - Returns an error when the specified file does not exist.
  - **w** opens an empty file for writing.
    - If the specified file already exists, then the contents of the file are destroyed.
  - **a** opens a file for writing.
    - Information will be written starting at the end of the file (append mode).
    - If the file does not exist, this function first creates the file.
  - **r+** opens a file for reading and writing.
    - Returns an error when the specified file does not exist.
  - **w+** opens an empty file for reading and writing.
    - If the specified file already exists, then the contents of the file are destroyed.
  - **a+** opens a file for both reading and writing.
    - Information will be written starting at the end of the file (append mode).
    - If the file does not exist, this function first creates the file.
- When the access mode is specified as 0, the file will be opened as if the mode had been specified as **r+**. When a file is opened, the file position indicator is 0 except for **a** and **a+**, in which case it is at the end of the file.
- The file position indicator is always moved to the end of a file before the first write, and therefore, existing file data cannot be overwritten.
- The attribute of the created file is **AMBA_FS_ATTR_ARCH** (archive) by default. When this attribute needs to be changed, use the **AmbaFS_Chmod** function after file creation.
- When the **AmbaFS_fopen** function opens the file successfully, it returns the file descriptor. When it fails to open the file, a NULL pointer is returned. Call the **AmbaFS_GetError** API to retrieve error information.
- A file that is already open can be reopened with the **AmbaFS_fopen** function by specifying a mode other than **w** or **w+**. Because each file will have a separate file descriptor, care must be exercised when reading or writing to a file that has been opened more than once. For example, when one file descriptor is used to write to a file, and another file descriptor is used to write to the same file, data may not be saved correctly. Therefore, do not execute reading/writing operations on the same file using different file descriptors simultaneously. When such operations are executed, accurate data cannot be guaranteed.

Parameters:

Type	Parameter	Description
const char*	**pFileName**	Pointer to path and file name
const char*	**pMode**	Access mode

*Table 2-16. Parameters for File System API **AmbaFS_fopen()**.*

Returns:

Return	Description
Nonzero	Completed successfully (file descriptor)
0	Open failed

*Table 2-17. Returns for File System API **AmbaFS_fopen()**.*

Example:

```
AMBA_FS_FILE *pFile;
UINT64 SuccessSize;
unsigned char buff[10];

/* open a sample file */
pFile = AmbaFS_fopen("A:\\sample.txt", "w+");
/* write data to the sample file */
SuccessSize = AmbaFS_fwrite((void*)FOOBAR, 6, 1, pFile);
…
/* seek file pointer to start of the sample file */
AmbaFS_fseek(pFile, 0, AMBA_FS_SEEK_SET);
/* read data from the sample file */
SuccessSize = AmbaFS_fread((void*)&buff, 6, 1, pFile);
…
/* close the sample file */
AmbaFS_fclose(pFile);
```

See Also:

None

2.2.9 AmbaFS_Format

API Syntax:

int **AmbaFS_Format** (char Drive, const char *pParam)

Function Description:

- The **AmbaFS_Format** function formats a drive.
- There are two parameters. The drive name for the drive to be formatted must be specified in the first parameter, **Drive**. This drive name is not case-sensitive. A pointer to the string that is passed to the driver must be specified in the second parameter, **pParam**. This string is used if the application needs to notify the driver of the format type.
- Formatting processes can be performed even when a driver is not mounted, as long as the driver is attached and the read/write process is physically enabled.
- The **AmbaFS_Format** function returns 0 when drive formatting is successful and returns -1 when it fails. Call the **AmbaFS_GetError** function to retrieve relevant error information.
- The **AmbaFS_Format** function writes to media immediately, regardless of the mode specified by **AmbaFS_SetBufferingMode**.
- If the formatted drive is the current drive, the current directory is changed to the root of the drive. The volume label will also be deleted.
- If an open file or directory exists on the drive specified by **Drive**, the file and directory descriptor become invalid.

Parameters:

Type	Parameter	Description
char	**Drive**	Drive name to be specified (Example 'A')
const char*	**pParam**	Pointer to the string to pass to the driver: FAT12: Microsoft FAT12 file system format FAT16: Microsoft FAT16 file system format FAT32: Microsoft FAT32 file system format Note: For SDHC media use NULL parameter; the driver would use the SD20 format parameter

*Table 2-18. Parameters for File System API **AmbaFS_Format()**.*

Returns:

Return	Description
0	Completed successfully
-1	Drive format failed

Table 2-19. Returns for File System API **AmbaFS_*Format()***.

Example:

```
int rval;
int slot = SCM_SLOT_SD;
char drv;
char dummy = '\0';

drv = 'a'+ slot;

rval = AmbaFS_Format(drv, &dummy);
if (rval < 0) {
      AmbaPrintf("driver %c format error", drv);
      return -1;
}

AmbaPrintf("format successful!");
return 0;
```

See Also:

None

2.2.10 AmbaFS_fread

API Syntax:

UINT64 **AmbaFS_fread** (void *pBuf, UINT64 Size, UINT64 Count, AMBA_FS_FILE *pFile)

Function Description:

- The **AmbaFS_fread** function reads data from an open file.
- There are four parameters. The first parameter, **pBuf**, is a pointer to the storage area. The second parameter, **size**, specifies the size of the items to be read in bytes. The third parameter, **Count**, specifies the number of items to be read. Thus, **AmbaFS_fread** will read **size * Count** bytes worth of data from the file. The fourth parameter, **stream**, specifies the file descriptor for the file to be read.
- When file data is read, the file position indicator is increased by the number of bytes read (**size * Count**). Use the **AmbaFS_fseek** function to change the file position indicator.
- The **AmbaFS_fread** function returns the actual data item count when file data are read successfully. When the returned value is smaller than the value specified in **nobj**, it indicates that the pointer reached the end of the file, or that the read failed. Use the **AmbaFS_GetError** function to check whether an error has occurred. The **AmbaFS_feof** function can be used to verify that the file position indicator has reached the end of a file. If an error has occurred, the file position indicator will be undefined.
- When the item count in the return value is smaller than **Count**, it indicates that a size between [Return value item **Count** * byte] and [Return value item **Count** * byte + byte – 1] was actually read.
- The AmbaFS API performs NULL pointer checking only. Therefore, proper operation is not guaranteed if an address to an invalid area is specified.
- Data cannot be read if the size is set to 4 GB or larger. In this case, data is read up to the 4 GB limit, and the number of items successfully read is returned as the return value.

Parameters:

Type	Parameter	Description
void*	**pBuf**	Pointer to data storage area for data to be read
UINT64	**size**	Item size for data to be read (bytes) Set to less than 4 GB
UINT64	**Count**	Item count for data to be read
AMBA_FS_FILE*	**pFile**	Pointer to file descriptor

*Table 2-20. Parameters for File System API **AmbaFS_fread()**.*

Returns:

Return	Description
Same value as **Count**	Completed successfully (number of items read)
Different value than **Count**	I/O position indicator reached end of file; read failed

*Table 2-21. Returns for File System API **AmbaFS_fread()**.*

Example:

```
AMBA_FS_FILE *pFile;
UINT64 SuccessSize;
unsigned char buff[10];
…
…
/* open a sample file */
pFile = AmbaFS_fopen("A:\\sample.txt", "w+");
/* write data to the sample file */
SuccessSize = AmbaFS_fwrite((void*)FOOBAR, 6, 1, pFile);
…
/* seek file pointer to start of the sample file */
AmbaFS_fseek(pFile, 0, AMBA_FS_SEEK_SET);
/* read data from the sample file */
SuccessSize = AmbaFS_fread((void*)&buff, 6, 1, pFile);
…
/* close the sample file */
AmbaFS_fclose(pFile);
…
…
```

See Also:

None

2.2.11 AmbaFS_fseek

API Syntax:

int **AmbaFS_fseek** (AMBA_FS_FILE *pFile, UINT64 Offset, int Origin)

Function Description:

- The **AmbaFS_fseek** function moves the file position indicator in an open file.
- There are three parameters. The file descriptor of the file whose file position indicator is to be moved must be specified in the first parameter **pFile**. The byte count from the reference point must be specified in the second parameter **offset**. The reference point (**AMBA_FS_SEEK_CUR**, **AMBA_FS_SEEK_SET** or **AMBA_FS_SEEK_END**) must be specified in the third parameter **Origin**.
- The file position indicator is moved from the origin to the new location specified by the **Offset** bytes. The file position indicator may be moved to a point beyond the end of the file. However, the **AmbaFS_fseek** function returns an error if the pointer is moved ahead of the start of the file.
- The **AmbaFS_fseek** function returns 0 when it has succeeded in moving the file position indicator and returns -1 when it fails. Call the **AmbaFS_GetError** API to retrieve error information.
- When calling this function results in an error, the file data I/O pointer retains the position it held prior to the **AmbaFS_fseek** function being called.
- The file data I/O pointer cannot be moved to an area above 4 GB. If this is attempted, the file data I/O pointer will not change from the position it held prior to the **AmbaFS_fseek** function being called.
- If writing is attempted after moving the file I/O pointer to an area beyond the end of the file, the area from the end of the file to the moved file I/O pointer will maintain the state it was in prior to writing.

Parameters:

Type	Parameter	Description
AMBA_FS_FILE*	**pFile**	Pointer to file descriptor
UINT64	**Offset**	Byte count from reference point
int	**Origin**	Reference point 0x00: **AMBA_FS_SEEK_SET** (Start) 0x01: **AMBA_FS_SEEK_CUR** (Current) 0x02: **AMBA_FS_SEEK_END** (End)

*Table 2-22. Parameters for File System API **AmbaFS_fseek()**.*

Returns:

Return	Description
0	Completed successfully
-1	Failed to move file position indicator

*Table 2-23. Returns for File System API **AmbaFS_fseek()**.*

Example:

```
AMBA_FS_FILE *pFile;
UINT64 SuccessSize;
unsigned char buff[10];

/* open a sample file */
pFile = AmbaFS_fopen("A:\\sample.txt", "w+");
/* write data to the sample file */
SuccessSize = AmbaFS_fwrite((void*)FOOBAR, 6, 1, pFile);
…
/* seek file pointer to start of the sample file */
AmbaFS_fseek(pFile, 0, AMBA_FS_SEEK_SET);
/* read data from the sample file */
SuccessSize = AmbaFS_fread((void*)&buff, 6, 1, pFile);
…
/* close the sample file */
AmbaFS_fclose(pFile);
```

See Also:

None

2.2.12 AmbaFS_FirstDirEnt

API Syntax:

int **AmbaFS_FirstDirEnt** (const char *pDirName, unsigned Attr, AMBA_FS_DTA *pDta)

Function Description:

- The **AmbaFS_FirstDirEnt** API is a function used to search for a file that matches a set of defined attributes.
- There are three parameters. The first parameter **pDirName** specifies a pointer to the buffer address that contains the file name and the path to be searched. Two wildcard characters, "*" and "?", may be used in the file name. When the second parameter **Attr** is set to a file attribute, files with the designated attribute are searched. When this parameter is set to **AMBA_FS_ATTR_NONE**, no attributes will be searched. When set to **AMBA_FS_ATTR_ALL**, all files, subdirectories, and volume labels are searched.
- The **AmbaFS_FirstDirEnt** function normally searches files, but when the attribute is set to **AMBA_FS_ATTR_AND**, the **AmbaFS_FirstDirEnt** function searches mode as well.
- The search result is stored in the third parameter, **pDta**. A long name is stored in the **AMBA_FS_DTA** member **LongName** if the file name is not 8.3 file name and upper case letters are used. In this case, alias name is stored in the **AMBA_FS_DTA** member FileName.

Parameters:

Type	Parameter	Description
const char*	**pDirName**	Pointer to path and file name
unsigned attr	**Attr**	File attribute: 0x01: **AMBA_FS_ATTR_RDONLY** (Read only) 0x02: **AMBA_FS_ATTR_HIDDEN** (Hidden file) 0x04: **AMBA_FS_ATTR_SYSTEM** (System file) 0x08: **AMBA_FS_ATTR_VOLUME** (Volume label) 0x10: **AMBA_FS_ATTR_DIR** (Subdirectory) 0x20: **AMBA_FS_TTR_ARCH** (Archive) 0x40: **AMBA_FS_ATTR_NONE** (No attribute) 0x3F: **AMBA_FS_ATTR_ALL** (All files) Search mode: 0x80: **AMBA_FS_ATTR_AND** (AND mode)
AMBA_FS_DTA*	**pDta**	Pointer to the file search result (See Section 2.2.12.1 for the **AMBA_FS_DTA** definition)

*Table 2-24. Parameters for File System API **AmbaFS_FirstDirEnt()**.*

Returns:

Return	Description
0	Completed successfully
-1	File not found or file search failed

*Table 2-25. Returns for File System API **AmbaFS_FirstDirEnt()**.*

Example:

```
AMBA_FS_DTA DtaTable;


/* search for  a sample file */
AmbaFS_FirstDirEnt("A:\\sample.txt", AMBA_FS_ATTR_ALL, &DtaTable);
/* search for  the sample file again *I
AmbaFS_NextDirEnt(&DtaTable);
```

See Also:

None

2.2.12.1 AmbaFS_FirstDirEnt > AMBA_FS_DTA

```
<<AMBA FS DTA structure>>

typedef struct _AMBA_FS_DTA_s_ {
AMBA_FS_DTA_UNION u;
int FsType;
UINT16 Time;
UINT16 Date;
UINT64 FileSize;
char FileName[AMBA_FS_NAME_LEN];
char LongName[AMBA_FS_NAME_LEN];
char Attribute;
int search_Mode; /* search for single file, wildcard(Regx) */
void *pRomfsCurrentInode; /* current inode for searching files */
} AMBA_FS_DTA;)
```

2.2.13 AmbaFS_NextDirEnt

API Syntax:

int **AmbaFS_NextDirEnt** (AMBA_FS_DTA *pDta)

Function Description:

- The **AmbaFS_NextDirEnt** is a function used to search the next file according to defined file search information.
- Before the **AmbaFS_NextDirEnt** function can be used, the **AmbaFS_FirstDirEnt** function must be executed, and the resultant file search results must be set in the parameter **AMBA_FS_DTA**.
- The search result is stored in **AMBA_FS_DTA** when a file that matches the search condition is located. If the file name is under 8.3 characters, the full file name will be stored in the search result, **LongName**. If the file name is longer than 8.3 characters, or if lowercase letters are used, then a shortened file name (alias) will be stored in **FileName**.
- The pattern matching specification is the same as that of the **AMBAFS_FirstDirEnt** function.
- The **AmbaFS_NextDirEnt** API returns 0 when the file search completes successfully and returns -1 when no file is found or the file search fails. Call the **AmbaFS_GetError** API to retrieve error information.

Parameters:

Type	Parameter	Description
AMBA_FS_DTA*	**pDta**	Pointer to file search information (See Section 2.2.12 "AmbaFS_FirstDirEnt" for the **AMBA_FS_DTA** definition)

*Table 2-26. Parameters for File System API **AmbaFS_NextDirEnt()**.*

Returns:

Return	Description
0	Completed successfully
-1	File not found or file search failed

*Table 2-27. Returns for File System API **AmbaFS_NextDirEnt()**.*

Example:

```
AMBA_FS_DTA DtaTable;


/* search for a sample file */
AmbaFS_FirstDirEnt("A:\\sample.txt", AMBA_FS_ATTR_ALL, &DtaTable);
/* search for the sample file again*/
AmbaFS_NextDirEnt(&DtaTable);
…
```

See Also:

AmbaFS_FirstDirEnt()

2.2.14 AmbaFS_Stat

API Syntax:

int **AmbaFS_Stat** (const char *pName, AMBA_FS_STAT ***pStat**)

Function Description:

- The **AmbaFS_Stat** function retrieves file or directory information.
- There are two parameters.
  - The first parameter, **pName**, is a pointer to the path and file name or directory name. If an entry with no actual file name or directory name, such as "." or "..", that exists under a subdirectory is specified, information other than file size (file byte size) can be acquired. In this case, the **AmbaFS_Stat** function is always set to 0.
  - A pointer to the structure that stores the file and directory information must be specified in the second parameter **pStat**.
- The **AmbaFS_Stat** API returns 0 when it successfully obtains the file or directory information and returns -1 when it fails. Additional error information can be obtained by calling the **AmbaFS_GetError** function.
- The Amba File System module performs **NULL** pointer checking only. Proper operation cannot be guaranteed when an invalid pointer is specified.
- When **pName** specifies an open file, the latest retrievable information is acquired. As a consequence, the data retrieved may not represent the information actually written to the media. In addition, when using the AmbaFS File System cache feature for caching the FAT or other data, the information acquired by the **AmbaFS_Stat** function and the information written to the actual media may also differ.

Parameters:

Type	Parameter	Description
const char*	**pName**	Pointer to path and file name
AMBA_FS_STAT*	**pStat**	Pointer to file information area

*Table 2-28. Parameters for File System API **AmbaFS_Stat()**.*

Returns:

Return	Description
0	Completed successfully
-1	Failed to retrieve file or directory information

*Table 2-29. Returns for File System API **AmbaFS_Stat()**.*

Example:

```
AMBA_FS_FILE *pFile;
AmbaFS_STAT stat;
UINT32 SuccessSize;
int Result;
…
…
/* get the file status  for a sample file */
AmbaFS_Stat("A:\\sample.txt", &stat);
…
/* open a sample file */
pFile = AmbaFS_fopen("A:\\sample.txt", "a+");
…
/* check whether the file pointer is at the end of file */
Result = AmbaFS_feof(pFile);
```

See Also:

None

2.2.14.1 AmbaFS_fstat > AMBA_FS_STAT

```
<<AMBA_FS_STAT structure>>

typedef struct _AMBA_FS_STAT_s_ {
    PF_STAT Stat;
    UINT64     Size;        /* file size in bytes */
    AMBA_FS_FILE_DATE   LastAccDate;  /* Update on read or write */
    AMBA_FS_FILE_TIME   LastAccTime;  /* Update on read or write */
    AMBA_FS_FILE_TIME   LastMdyTime;  /* Update on write */
    AMBA_FS_FILE_DATE   LastMdyDate;  /* Update on write */
    AMBA_FS_FILE_TIME   CreateTime; /* Update on create */
    AMBA_FS_FILE_DATE   CreateDate; /* Update on create */
    AMBA_FS_FILE_COMP_TIME CreateCompTime; /* Update on create */
    UINT32                 Attr;
} AMBA_FS_STAT;
```

2.2.15 AmbaFS_FSync

API Syntax:

int **AmbaFS_FSync** (AMBA_FS_FILE *pFile)

Function Description:

- This function is used to write data contained in the cache of the specified file to media. Cache data that has not been updated will not be written.
- Set the parameter **pFile** to the descriptor for the file to which the cache data will be written.
- The **AmbaFS_FSync** API returns 0 when it successfully writes the cache data and returns -1 when it fails. Call the **AmbaFS_GetError** function to retrieve additional error information.
- Whether the file specified by parameter **pFile** is used or not, all FAT and directory entry areas are written back.

Parameters:

Type	Parameter	Description
AMBA_FS_FILE*	**pFile**	Pointer to the file descriptor

*Table 2-30. Parameters for File System API **AmbaFS_FSync()**.*

Returns:

Return	Description
0	Completed successfully
-1	Failed to write back cache

*Table 2-31. Returns for File System API **AmbaFS_FSync()**.*

Example:

```
AMBA_FS_FILE  *pFile;
UINT64 SuccessSize;
…
…
/* open a simple file */
pFile = AmbaFS_fopen("A:\\sample.txt", "w+");
/* write data to the sample file * /
SuccessSize = AmbaFS_fwrite((void*)FOOBAR, 6, 1, pFile);
…
/* synchronize the file data in cache and the media */
AmbaFS_FSync(pFile);
…
/* close the sample file*/
AmbaFS_fclose(pFile);
…
…
```

See Also:

None

2.2.16 AmbaFS_fwrite

API Syntax:

UINT64 **AmbaFS_fwrite** (const void *pBuf, UINT64 Size, UINT64 Count, AMBA_FS_FILE *pFile)

Function Description:

- The **AmbaFS_fwrite** function writes data into an open file.
- There are four parameters. The first parameter, **pBuf**, is a pointer to the data area. The second parameter, **Size**, specifies the size of the items to write in bytes. The third parameter, **Count**, specifies the number of items to write. Thus, the **AmbaFS_fwrite** function will write **Size * Count** bytes worth of data to the file. The fourth parameter, **pFile**, specifies the file descriptor for the file to be written.

  When file data is written, the file position indicator is increased by the number of bytes written (**Size * Count**). Use the **AmbaFS_fseek** function to change the file position indicator.
- The **AmbaFS_fwrite** function will return the actual data item count when file data are written successfully. When the returned value is smaller than the value specified in **Count**, it indicates that the write failed. Use the **AmbaFS_GetError** function to verify whether an error has occurred. If an error has occurred, the file position indicator becomes undefined.
- When the item count in the returned value is smaller than **Count**, it indicates that a size between [Return value item **Count** * byte] and [Return value item **Count** * byte + byte – 1] is actually written.
- The AmbaFS API performs **NULL** pointer checking only. Therefore, proper operation is not guaranteed if an address to an invalid area is specified.
- Data cannot be written if **Size** is set to 4 GB or larger. In this case, data up to 4 GB is written, and the number of items successfully written is returned as the returned value.

Parameters:

Type	Parameter	Description
const void*	**pBuf**	Pointer to data storage area for data to be read
UINT64	**Size**	Item size for data to be read (bytes). Less than 4 GB.
UINT64	**Count**	Item count for data to be read
AMBA_FS_FILE*	**pFile**	Pointer to file descriptor

*Table 2-32. Parameters for File System API **AmbaFS_fwrite()**.*

Returns:

Return	Description
Same value as **Count**	Completed successfully (number of items written)
Different value than **Count**	Data write failed

*Table 2-33. Returns for File System API **AmbaFS_fwrite()**.*

Example:

```
AMBA_FS_FILE *pFile;
UINT64 SuccessSize;
unsigned char buff[10];

/* open a sample file */
pFile = AmbaFS_fopen("A:\\sample.txt", "w+");
/* write data to the sample file */
SuccessSize = AmbaFS_fwrite((void*)FOOBAR, 6, 1, pFile);
/* seek file pointer to start of the sample file */
AmbaFS_fseek(pFile, 0, AMBA_FS_SEEK_SET);
```

See Also:

None

2.2.17 AmbaFS_GetBufferingMode

API Syntax:

int **AmbaFS_GetBufferingMode** (char Drive)

Function Description:

- The **AmbaFS_GetBufferingMode** function retrieves the current buffering mode of a specified slot.
- Buffering Mode can set by using the **AmbaFS_SetBufferingMode** function.

Parameters:

Type	Parameter	Description
char	**Drive**	The name of the drive

*Table 2-34. Parameters for File System API **AmbaFS_GetBufferingMode()**.*

Returns:

Return	Description
0x00	**AMBA_FS_WRITE_BACK_CACHE**: Buffer Mode - Write back mode
0x01	**AMBA_FS_EJECT_SAFE_WITH_WRITE_THRU:** Buffer Mode - Eject-Safe write through mode
0x02	**AMBA_FS_WRITE_BACK_ON_SIGNIF_API**: Buffer Mode - Automatic write back mode
0x06	**AMBA_FS_EJECT_SAFE_WITH_WRITE_BACK**: Buffer Mode - Eject-Safe write back mode

*Table 2-35. Returns for File System API **AmbaFS_GetBufferingMode()**.*

Example:

```
int buf_Mode;
char Slot = 'A';

buf_Mode = AmbaFS_GetBufferingMode(Slot);
```

See Also:

AmbaFS_SetBufferingMode()

2.2.18 AmbaFS_GetDev

API Syntax:

int **AmbaFS_GetDev** (char Drive, AMBA_FS_DEVINF *pDevInf)

Function Description:

- The **AmbaFS_GetDev** API is a function used to acquire the device capacity.
- There are two parameters.
  - The first parameter, **Drive**, specifies the drive for which the device capacity should be retrieved. This drive name is not case-sensitive. This parameter can be set to any drive name from A to Z.
  - The second parameter, **pDevInf**, is a pointer to the area in which the device capacity information is stored.
- The **AmbaFS_GetDev** API returns 0 when it succeeds in retrieving device capacity and returns -1 when it fails.
- Additional error information can be obtained by calling the **AmbaFS_GetError** function.

Parameters:

Type	Parameter	Description
char	**Drive**	Drive name to be specified (See examples below)
AMBA_FS_DEV_INF *	**pDevInf**	Storage address of device capacity area (See Section 2.2.20.1 below for structure definition)

*Table 2-36. Parameters for File System API **AmbaFS_GetDev()**.*

Returns:

Return	Description
0	Completed successfully
-1	Failed to retrieve device capacity

*Table 2-37. Returns for File System API **AmbaFS_GetDev()**.*

Example:

```
int Rval;
AMBA_FS_DEVINF DevInf;
AMBA_FS_VOLTAB Voltab;


Rval = AmbaFS_GetVol('A', &Voltab);
if (Rval < 0) {
AmbaPrintf("volume is not ready!");
return -2;
}

AmbaFS_GetDev('A', &DevInf);

memset(name, 0, sizeof(name));
strncpy(name, Voltab.name, 11);

AmbaPrintf("volume: %s\n", name);
AmbaPrintf("total clusters: %lld\n", DevInf.Cls);
AmbaPrintf("empty clusters: %lld\n", DevInf.Ecl);
AmbaPrintf("bytes per sector: %d\n", DevInf.Bps);
AmbaPrintf("sectors per cluster: %d\n", DevInf.Spc);
AmbaPrintf("total space: %lld KB",
          (((UINT64)(DevInf.Cls * DevInf.Spc)) * DevInf.Bps) >> 10);
AmbaPrintf("used: %lld KB",
          (((UINT64)((DevInf.Cls - DevInf.Ecl) * DevInf. Spc)) *
            DevInf.Bps) >> 10);
AmbaPrintf("free space: %lld KB",
          (((UINT64)(DevInf.Ecl * DevInf.Spc)) * DevInf.Bps) >> 10);
```

See Also:

None

2.2.18.1 AmbaFS_GetDev > AMBA_FS_DEVINF

```
<<AMBA FS DEVINF structure>>

typedef struct _AMBA_FS_DEVINF_s_ {
    PF_DEV_INF          DevInfo;
    UINT32              Cls;        /* Total clusters */
    UINT32              Ucl;        /* Unused clusters */
    UINT32              Bps;        /* Bytes per sector */
    UINT32              Spc;        /* Sectors per cluster */
    UINT32              Cpg;        /* Clusters per cluster group */
    UINT32              Ucg;        /* Unused cluster groups */
    AMBA_FS_FAT_TYPE_e  Fmt;        /* Format type */
} AMBA_FS_DEVINF;
```

2.2.19 AmbaFS_GetVol

API Syntax:

int **AmbaFS_GetVol** (char Drive, AMBA_FS_VOLTAB *pVolTab)

Function Description:

- The **AmbaFS_GetVol** API is a function used to acquire the volume label information.
- There are two parameters.
  - The name of the drive from which to retrieve the volume label information must be specified in the first parameter, **Drive**. This drive name is not case-sensitive. This parameter can be set to any drive name from A to Z.
  - A pointer to the area in which to store the volume label information must be specified in the second parameter, **pVolTab**.
- The **AmbaFS_GetVol** API returns 0 when it retrieves the volume label information and returns -1 when it fails.

Parameters:

Type	Parameter	Description
char	**Drive**	Drive name to be specified
AMBA_FS_VOLTAB*	**pVolTab**	Pointer to volume label information storage area (See Section 2.2.21.1 below for structure definition)

*Table 2-38. Parameters for File System API **AmbaFS_GetVol()**.*

Returns:

Return	Description
0	Completed successfully
-1	Failed to retrieve volume label information

*Table 2-39. Returns for File System API **AmbaFS_GetVol()**.*

Example:

```
int Rval;
AMBA_FS_DEVINF DevInf;
AMBA_FS_VOLTAB Voltab;


Rval = AmbaFS_GetVol('A', &Voltab);
if (Rval < 0) {
AmbaPrintf("volume is not ready!");
return -2;
}

AmbaFS_GetDev('A', &DevInf);

memset(name, 0, sizeof(name));
strncpy(name, Voltab.name, 11);

AmbaPrintf("volume: %s\n", name);
AmbaPrintf("total clusters: %lld\n", DevInf.Cls);
AmbaPrintf("empty clusters: %lld\n", DevInf.Ecl);
AmbaPrintf("bytes per sector: %d\n", DevInf.Bps);
AmbaPrintf("sectors per cluster: %d\n", DevInf.Spc);
AmbaPrintf("total space: %lld KB",
          (((UINT64)(DevInf.Cls * DevInf.Spc)) * DevInf.Bps) >> 10);
AmbaPrintf("used: %lld KB",
          (((UINT64)((DevInf.Cls - DevInf.Ecl) * DevInf. Spc)) *
DevInf.Bps) >> 10);
AmbaPrintf("free space: %lld KB",
          (((UINT64)(DevInf.Ecl * DevInf.Spc)) * DevInf.Bps) >> 10);
```

See Also:

None

2.2.19.1 AmbaFS_GetVol > AMBA_FS_VOLTAB

```
<<AMBA_FS_VOLTAB structure>>

typedef struct _AMBA_FS_VOLTAB_s_ {
    PF_VOLTAB   VolTab;
    int     FsType;
    char    Name[AMBA_FS_MAX_VOL_NAME_LEN]; /*volume label name*/
    UINT32  VolSrc;
    UINT8   VolAttr; /* volume label attribute */
    UINT32  VolDate; /* volume label creation date */
    UINT32  VolTime; /* volume label creation time */
} AMBA_FS_VOLTAB;
```

2.2.20 AmbaFS_fappend

API Syntax:

int **AmbaFS_fappend** (AMBA_FS_FILE *pFile, UINT64 Size)

Function Description:

- The **AmbaFS_fappend** API is a function used to add consecutive clusters of a specified size to the end of a file.
- There are two parameters.
  - Set the first parameter, **pFile**, to the pointer of the file to have clusters appended to it.
  - Set the second parameter, Size, to the size of the area to be added in bytes. The size specified in bytes is converted to an integer to denote the number of cluster units.
- The **AmbaFS_fappend** function returns the bytes of clusters that are added successfully.

Parameters:

Type	Parameter	Description
AMBA_FS_FILE*	**pFile**	Pointer to the file descriptor
UINT64	**Size**	Size of the area to be added (in bytes)

*Table 2-40. Parameters for File System API **AmbaFS_fappend()**.*

Returns:

Return	Description
Same value as Size	Completed successfully (bytes of Size are added)
Different value from Size	Only the bytes of return are added.

*Table 2-41. Returns for File System API **AmbaFS_fappend()**.*

Example:

```
AMBA_FS_FILE *pFile;
AMBA_FS_STAT Stat;
int SuccessSize;
int result;
…
…
/* get the file status for a sample file */
AmbaFS_Stat("A:\\sample.txt", Stat);
…
/* open a simple file */
pFile = AmbaFS_fopen("A:\\sample.txt", "a+");
…
/* check whether the file I/O pointer is at the end of file */
result = AmbaFS_feof(pFile);
…
```

```
/* append the data to the sample file */
SuccessSize = AmbaFS_fappend(pFile, 1024);
/* write data to the sample file */
SuccessSize = AmbaFS_fwrite((void*)”FOOBAR”, 6, 1, pFile);
…
/* close the sample file */
AmbaFS_fclose(pFile);
…
…
```

See Also:

None

2.2.21 AmbaFS_Init

API Syntax:

int **AmbaFS_Init** (UINT32 CodeMode)

Function Description:

- The **AmbaFS_Init** function executes the initialization process for the AmbaFS File System. The **AmbaFS_Init** function must be executed before using any other AmbaFS APIs.
- The Failure return value is provided as a placeholder for future expansion. Currently, the **AmbaFS_Init** API always returns 0.

Parameters:

Type	Parameter	Description
UINT32	**CodeMode**	0x0: **AMBA_FS_ASCII** 0x1: **AMBA_FS_UNICODE**

*Table 2-42. Parameters for File System API **AmbaFS_Init()**.*

Returns:

Return	Description
0	Success
< 0	Failure

*Table 2-43. Returns for File System API **AmbaFS_Init()**.*

Example:

None

See Also:

None

2.2.22 AmbaFS_Mkdir

API Syntax:

int **AmbaFS_Mkdir** (const char *pDirName)

Function Description:

- The **AmbaFS_Mkdir** API is a function used to create a directory.
- Specify the parameter **pDirName** as a pointer to the name of the directory to be created.
- The attribute of the created directory is **AMBA_FS_ATTR_DIR** (subdirectory) by default. When this attribute needs to be changed, use **AmbaFS_Chdmod** after creation.
- The **AmbaFS_Mkdir** function returns 0 when it successfully creates the directory and returns -1 when it fails.
- Because only one directory can be created with each execution, any directories specified in the path to **pDirName** must already exist. If the path does not exist, the directory is not created and an error is returned.

Parameters:

Type	Parameter	Description
const char*	**pDirName**	Pointer to path and directory name

*Table 2-44. Parameters for File System API **AmbaFS_Mkdir()**.*

Returns:

Return	Description
0	Completed successfully
-1	Directory creation failed

*Table 2-45. Returns for File System API **AmbaFS_Mkdir()**.*

Example:

```
…
/* make a directory */
AmbaFS_Mkdir("A:\\dir");
…
/* change the current directory */
AmbaFS_Chdir("A:\\");
…
/* change the attribute of the directory */
AmbaFS_Chdmod("A:\\dir", AMBA_FS_ATTR_DIR);
…
/* remove directory */
AmbaFS_Rmdir("A:\\dir");

…
…
```

See Also:

AmbaFS_Rmdir()

2.2.23 AmbaFS_Move

API Syntax:

int **AmbaFS_Move** (const char *pSrcName, const char *pDstName)

Function Description:

- The **AmbaFS_Move** API is a function used to move a specified file or directory.
- There are two parameters. Define the first parameter, **pSrcName**, as a pointer to the file or directory to be moved. Define the second parameter, **pDstName**, as a pointer to the destination file or directory name. The path may be included in either the source or destination file/directory name.
- If **pSrcName** does not include a path, the file in the current directory is moved. If **pDstName** does not include a path, the current directory becomes the target.
- A file name can be changed during the process of being moved. When the file name set in **pDstName** is not to be changed from the name set in **pSrcName**, both the file name and the directory name must be set in **pDstName**.
- The **AmbaFS_Move** API returns 0 when it successfully moves the file or directory and returns -1 when it fails.

Parameters:

Type	Parameter	Description
const char*	**pSrcName**	Pointer to source file/directory and path name
const char*	**pDstName**	Pointer to destination file/directory and path name

*Table 2-46. Parameters for File System API **AmbaFS_Move()**.*

Returns:

Return	Description
0	Completed successfully
-1	Failed to move file

*Table 2-47. Returns for File System API **AmbaFS_Move()**.*

Example:

```
…
/* rename a sample file */
AmbaFS_rename("A:\\sample.txt", "foobar.txt");
…
/* move the foobar file */
AmbaFS_Move("A:\\foobar.txt", "A:\\dir\\foobar.txt");
…
/* remove the foobar file */
AmbaFS_remove("A:\\foobar.txt");
…
…
```

See Also:

None

2.2.24 AmbaFS_remove

API Syntax:

AmbaFS_remove (const char *pFileName)

Function Description:

- The **AmbaFS_remove** function deletes a specified file.
- The address of the buffer that contains the file name to delete and its path must be specified in the parameter **pFileName**.
- The **AmbaFS_remove** function returns 0 when it successfully deletes the file and returns -1 when it fails.
- A file cannot be deleted if it is open, or if its attributes are set to read only (**AMBA_FS_ATTR_RDONLY**), volume label (**AMBA_FS_ATTR_VOLUME**), or subdirectory (**AMBA_FS_ATTR_ALL**).

Parameters:

Type	Parameter	Description
const char*	**pFileName**	Pointer to path and name

*Table 2-48. Parameters for File System API **AmbaFS_remove()**.*

Returns:

Return	Description
0	Completed successfully
- 1	Failed to move file

*Table 2-49. Returns for File System API **AmbaFS_remove()**.*

Example:

```
…
/* rename a sample file */
AmbaFS_rename("A:\\sample.txt", "foobar.txt");
…
/* move the foobar file */
AmbaFS_Move("A:\\foobar.txt", "A:\\dir\\foobar.txt");
…
/* remove the foobar file */
AmbaFS_remove("A:\\foobar.txt");
…
…
```

See Also:

None

2.2.25 AmbaFS_rename

API Syntax:

int **AmbaFS_rename** (const char *pOldName, const char *pNewName))

Function Description:

- The **AmbaFS_rename** function changes the name of a specified file or directory.
- There are two parameters:
  - The first parameter, **pOldName**, specifies a pointer to the file name to be renamed, and to its directory and path.
  - A pointer to the new file name or directory name must be specified in the second parameter **pNewName**.
- **AmbaFS_Rename** cannot move files or directories. Therefore, it returns an error if the path designated in **pOldName** and the path designated in **pNewName** are different.
- If **pOldName** does not contain a path, the file in the current directory is targeted. If only a file name or a directory name is designated in **pNewName**, the path will be the same as in **pOldName**.
- The **AmbaFS_Rename** function returns 0 when it successfully changes the file or directory name and returns -1 when it fails.
- If the file or directory designated by **pOldName** is open or has a read-only attribute (AMBA_FS_ATTR_RDONLY), the name cannot be changed. When the directory specified in **pOldName** is the current directory or a parent directory, the name cannot be changed.
- When a file is open in the directory specified in **pOldName** or its child directory, the name cannot be changed. Wildcard characters ("*" and "?") cannot be used to specify the file name.

Parameters:

Type	Parameter	Description
const char*	**pOldName**	Pointer to path and name
const char*	**pNewName**	Pointer to new file name

*Table 2-50. Parameters for File System API **AmbaFS_rename()**.*

Returns:

Return	Description
0	Completed successfully
-1	Failed to change file name or directory name

*Table 2-51. Returns for File System API **AmbaFS_rename()**.*

Example:

```
…
/* rename a sample file */
AmbaFS_rename("A:\\sample.txt", "foobar.txt");
…
/* move the foobar file */
AmbaFS_Move("A:\\foobar.txt", "A:\\dir\\foobar.txt");
…
/* remove the foobar file */
AmbaFS_remove("A:\\foobar.txt");
…
…
```

See Also:

None

2.2.26 AmbaFS_Rmdir

API Syntax:

int **AmbaFS_Rmdir** (const char *pDirName)

Function Description:

- The **AmbaFS_Rmdir** function deletes a directory.
- Define the parameter **pDirName** as a pointer to the directory name to delete.
- The **AmbaFS_Rmdir** function returns 0 when it successfully deletes the directory and returns -1 when it fails.
- If the directory specified by **pDirName** is the root directory, current directory, parent directory, or a file or directory existing in the specified directory, then the directory cannot be deleted, and an error is returned. If the directory specified by **pDirName** is open or a directory with the read-only (AMBA_FS_ATTR_RDONLY) attribute, **AmbaFS_Rmdir** function returns an error without deleting the directory.
- Wildcard characters ("*" and "?") cannot be used to specify the directory name.

Parameters:

Type	Parameter	Description
const char*	**pDirName**	Pointer to path and directory name

*Table 2-52. Parameters for File System API **AmbaFS_Rmdir()**.*

Returns:

Return	Description
0	Completed successfully
-1	Directory creation failed

*Table 2-53. Returns for File System API **AmbaFS_Rmdir()**.*

Example:

```
…
…
/* make a directory */
AmbaFS_Mkdir("A:\\dir");
…
/* change the current directory */
AmbaFS_Chdir("A:\\");
…
/* change the attribute of the directory */
AmbaFS_Chdmod("A:\\dir", AMBA_FS_ATTR_DIR);
…
/* remove directory */
AmbaFS_Rmdir("A:\\dir");
…
…
```

See Also:

None

2.2.27 AmbaFS_SetVol

API Syntax:

int **AmbaFS_SetVol** (char Drive, const char *pVolName)

Function Description:

- The **AmbaFS_SetVol** API is a function used to register the volume label.
- There are two parameters:
  - The first parameter, **Drive**, specifies the drive for which the volume label is to be set. This drive name is not case-sensitive. This parameter can be set to any drive name from A to Z.
  - The second parameter, **pVolName**, is a pointer to the buffer containing the volume label.
- Because the volume label does not have an extension, it must be designated as an 11-byte name.
- The attribute of the registered volume is AMBA_FS_ATTR_VOLUME (volume label) by default.
- The **AmbaFS_SetVol** returns 0 when volume label is set successfully and returns -1 when it fails.

Parameters:

Type	Parameter	Description
char	**Drive**	Drive name to be specified
const char*	**pVolName**	Pointer to volume label name (Example 'A')

*Table 2-54. Parameters for File System API **AmbaFS_SetVol()**.*

Returns:

Return	Description
0	Completed successfully
- 1	Volume label set failed

*Table 2-55. Returns for File System API **AmbaFS_SetVol()**.*

Example:

```
AMBA_FS_VOLTAB volume;


/* set the volume name */
AmbaFS_SetVol('A', FOOBAR);

/* get the volume label information */
AmbaFS_GetVol('A', &volume);
```

See Also:

None

2.2.28 AmbaFS_Sync

API Syntax:

int **AmbaFS_Sync** (char Drive, int Mode)

Function Description:

- The **AmbaFS_Sync** API is a function used to write all data in the cache of the specified drive to media. This function does not write back cache data that is not updated.
- There are two parameters. Set the first parameter, **Drive**, to the drive name to which the cache data will be written. Set the second parameter, **Mode**, to either **AMBA_FS_INVALIDATE** (invalidate cache) or **AMBA_FS_NINVALIDATE** (do not invalidate cache). Setting **AMBA_FS_INVALIDATE** (invalidate cache) invalidates all cache information. When the same area is accessed, data on the medium is read to the cache buffer.
- The **AmbaFS_Sync** function returns 0 when it successfully writes back the cache and returns -1 when it fails.

Parameters:

Type	Parameter	Description
char	**Drive**	Drive name to be specified (Example: 'A')
int	**Mode**	Mode: 0x00: **AMBA_FS_NINVALIDATE** (Does not invalidate cache) 0x01: **AMBA_FS_INVALIDATE** (Invalidates cache)

*Table 2-56. Parameters for File System API **AmbaFS_Sync()**.*

Returns:

Return	Description
0	Completed successfully
-1	Failed to write back cache

*Table 2-57. Returns for File System API **AmbaFS_Sync()**.*

Example:

```
AMBA_FS_FILE *pFile1, *pFile2;
UINT64 SuccessSize;

/* open files */
pFile1 = AmbaFS fopen("A:\\sample.txt", "w+");
pFile2 = AmbaFS fopen("A:\\foobar.txt", "r+");

/* write data */
SuccessSize = AmbaFS_fwrite((void*)SAMPLE, 6, 1, pFile1);
SuccessSize = AmbaFS_fwrite((void*)FOOBAR, 6, 1, pFile2);
/* synchronize the all data in cache and the media */
AmbaFS_Sync('A', AMBA_FS_NINVALIDATE);

/* close the sample file */
AmbaFS_fclose(pFile1);
AmbaFS_fclose(pFile2);
```

See Also:

AmbaFS_FSync()

2.2.29 AmbaFS_CleanDir

API Syntax:

int **AmbaFS_CleanDir** (const char *pDirName, const char *pFileName, UINT32 OpMode, UINT32 *pCount)

Function Description:

- **AmbaFS_CleanDir** is a function used to clean the specified directory (Unicode supported).
- There are four parameters and these parameters are described below:
  - Set a pointer **pDirName** to the directory name and the path to the first parameter.
  - Set a pointer to the buffer address that contains the file name to be deleted. This must be specified in the second parameter, **pFileName**. Two wild card characters, '*' and '?' may be used in this file name.
  - Search mode, Test mode, Delete files attribute, and Attribute comparing mode are specified in the third parameter **OpMode**.
  - The number of deleted files and directories are stored in **pCount**, the fourth parameter. When specifying **NULL** to this parameter, the number of deleted files and directories are not stored.
- The **AmbaFS_CleanDir** function returns 0 when it successfully deletes files and directories, and returns -1 when it fails.
- The directory specified with **pDirName** is not deleted. If the entry to be deleted is the root directory, the current directory, or is open; then the entry cannot be deleted and an error is returned.
- The number of the files and the directories which have been already deleted is stored to count.
- Wildcard characters ('*' and '?') cannot be used for the first parameter.

Parameters:

Type	Parameter	Description
const char*	**pDirName**	Pointer to the path and the directory name
const char*	**pFileName**	Pointer to the file name
UINT32	**OpMode**	Search mode: 0x10000: **AMBA_FS_MODE_SEARCH_BELOW** (below only) 0x20000: **AMBA_FS_MODE_SEARCH_TREE** (tree all) Test mode: 0x100000: **AMBA_FS_MODE_TEST** (test) Delete files attribute: 0x01: **AMBA_FS_ATTR_RDONL** (Read only) 0x02: **AMBA_FS_ATTR_HIDDEN** (Hidden file) 0x0: **AMBA_FS_ATTR_SYSTEM** (System file) 0x20: **AMBA_FS_ATTR_ARCH** (Archive) 0x100: **AMBA_FS_ATTR_FILE_ONLY** (File only) 0x40: **AMBA_FS_ATTR_NONE** (No attribute) 0x3f: **AMBA_FS_ATTR_ALL** (All files) Attributes comparing mode: 0x80: **AMBA_FS_MODE_CMP_AND** (AND) 0x1000: **AMBA_FS_MODE_CMP_MATCH** (Match full)
UINT32*	**pCount**	Pointer to the number of deleted files

*Table 2-58. Parameters for File System API **AmbaFS_CleanDir()**.*

Returns:

Return	Description
0	Completed successfully
- 1	Failed to clean the directories

*Table 2-59. Returns for File System API **AmbaFS_CleanDir()**.*

Example:

```
AMBA_FF_FILE *pfile;
UINT32 count;
…
…
/* make a directory */
AmbaFS_Mkdir("A:\\dir1");
/* make a directory */
AmbaFS_Mkdir("A:\\dir1\\dir2");
/* make a directory */
AmbaFS_Mkdir("A:\\dir1\\dir2\\dir3");
…
/* open a sample file */
pfile = AmabFS_fopen("A:\\dir1\\dir2\\dir3\\file.txt", "r");
/* close a sample file */
AmabFS_fclose(p_file);
…
/* clean directory */
AmbaFS_CleanDir("A:\\dir1",
                "*",
                AMBA_FS_SEARCH_BELOW | ATTR_ALL,
                &count);
```

See Also:

None

2.2.30 AmbaFS_ChmodDir

API Syntax:

int **AmbaFS_ChmodDir** (const char *pPath, const char *pFileName, UINT32 Mode, UINT32 *pCount)

Function Description:

- **AmbaFS_ChmodDir** is a function used to change the attribute of the files/directories under the directory (Unicode supported).
- There are five parameters.
  - Set a pointer **pDirName** to the directory name and the path to the first parameter.
  - Set a pointer to the buffer address that contains the file name to be deleted. This must be specified in the second parameter, **pFileName**. Two wild card characters, '*' and '?' may be used in this file name.
  - Search mode, Test mode, Delete files attribute, and Attribute comparing mode are specified in the third parameter **Mode**.
  - The attribute information is specified in the fourth parameter, **Attr**.
  - The number of the files and the directories whose attributes have been changed is stored in the fifth parameter **Count**. When **NULL** is specified, it is not stored to the parameter.
  - The number of deleted files and directories are stored in **pCount**, the fourth parameter. When specifying **NULL** to this parameter, the number of deleted files and directories are not stored.
- The **AmbaFS_ChmodDir** function returns 0 when it successfully changes the attribute and returns -1 when it fails.
- The directory specified with **pDirName** is not changed. When the entry is open, the error is returned.
- The number of the files and the directories which have been already changed is stored to count.
- Wildcard characters ('*' and '?') cannot be used for the first parameter.

Parameters:

Type	Parameter	Description
const char*	**Parameter**	Pointer to path and directory name
const char*	**pPath**	Pointer to file name
UINT32	**pFileName**	Search mode: 0x10000: **AMBA_FS_MODE_SEARCH_BELOW** (below only) 0x20000: **AMBA_FS_MODE_SEARCH_TREE** (tree all) Test mode: 0x100000: **AMBA_FS_MODE_TEST** (test) Delete files attribute: 0x01: **AMBA_FS_ATTR_RDONL** (Read only) 0x02: **AMBA_FS_ATTR_HIDDEN** (Hidden file) 0x0: **AMBA_FS_ATTR_SYSTEM** (System file) 0x20: **AMBA_FS_ATTR_ARCH** (Archive) 0x100: **AMBA_FS_ATTR_FILE_ONLY** (File only) 0x40: **AMBA_FS_ATTR_NONE** (No attribute) 0x3f: **AMBA_FS_ATTR_ALL** (All files) Attribute comparing mode: 0x80: **AMBA_FS_MODE_CMP_AND** (AND) 0x1000: **AMBA_FS_MODE_CMP_MATCH** (Match full)

Type	Parameter	Description
UINT32	**Attr**	File attribute: 0x01: **AMBA_FS_ATTR_RDONL** (Read only) 0x02: **AMBA_FS_ATTR_HIDDEN** (Hidden file) 0x04: **AMBA_FS_ATTR_SYSTEM** (System file) 0x20: **AMBA_FS_ATTR_ARCH** (Archive) 0x40: **AMBA_FS_ATTR_NONE** (No attribute) Change attribute mode: 0x2000: **AMBA_FS_MODE_ATTR_ADD** (Addition) 0x4000: **AMBA_FS_MODE_ATTR_SUB** (Deletion)
UINT32*	**pCount**	Pointer to number of changes in the attribute

*Table 2-60. Parameters for File System API **AmbaFS_ChmodDir()**.*

Returns:

Return	Description
0	Completed successfully
- 1	Failed to change the attribute

*Table 2-61. Returns for File System API **AmbaFS_ChmodDir()**.*

Example:

```
AMBA_FF_FILE *pfile;
unsigned long count;
…
…
/* make a directory */
AmbaFS_Mkdir("A:\\dir1");
/* make a directory */
AmbaFS_Mkdir("A:\\dir1\\dir2");
/* make a directory */
AmbaFS_Mkdir("A:\\dir1\\dir2\\dir3");
…
/* open a sample file */
pfile = AmabFS_fopen("A:\\dir1\\dir2\\dir3\\file.txt", 0);
/* close a sample file */
AmabFS_fclose(p_file);
…
/* chmod files/directories under the specified directory */
AmbaFS_ChmodDir("A:\\dir1",
                "*",
                AMBA_FS_MODE_SEARCH_BELOW | AMBA_FS_ATTR_ALL,
                AMBA_FS_ATTR_RDONLY,
                &count);
…
…
```

See Also:

None

2.2.31 AmbaFS_Cinsert

API Syntax:

int **AmbaFS_Cinsert** (const char *pFileName, UINT32 Offset, UINT32 Number)

Function Description:

- **AmbaFS_Cinsert** is a function used to insert a number of clusters in a file (Unicode supported).
- There are three parameters.
  - Set the pointer **pFileName** to the name and the path of the file where the unused clusters are to be inserted.
  - Set the second parameter, **Offset**, to the location to insert a cluster at the beginning of the file.
  - Set the third parameter, **Number**, to the number of unused clusters to insert.
- The **AmbaFS_Cinsert** function returns the number of clusters that are actually inserted when it successfully inserts clusters. If the value is smaller than the value designated by the number, it means that the cluster insertion failed. Call the **AmbaFS_GetError** API to get details on error information.

Parameters:

Type	Parameter	Description
const char*	**pFileName**	Pointer to file name
UINT32	**Offset**	Byte count from the reference point
UINT32	**Number**	Byte count for the cluster to be inserted

*Table 2-62. Parameters for File System API **AmbaFS_Cinsert()**.*

Returns:

Return	Description
Same value as **Number**	Completed successfully (bytes of Number are added)
Different value from **Number**	The number of clusters that fail to be inserted (Only bytes of return are inserted)

*Table 2-63. Returns for File System API **AmbaFS_Cinsert()**.*

Example:

```
…
…
/* combine the sample file and the foobar file */
AmbaFS_Combine("A:\\sample.txt", "A:\\dir\\foobar.txt");
…
/* insert cluster to the sample file */
AmbaFS_Cinsert("A:\\sample.txt", 5, 2);
…
/* delete cluster from the sample file */
AmbaFS_Cdelete("A:\\sample.txt", 10, 1);
…
```

```
/* divide the sample file */
AmbaFS_Divide("A:\\sample.txt", "A:\\dir\\foobar.txt", 1024);
…
…
```

See Also:

AmbaFS_Cdelete()

2.2.32 AmbaFS_Cdelete

API Syntax:

int **AmbaFS_Cdelete** (const char *pFileName, UINT32 Offset, UINT32 Number)

Function Description:

- **AmbaFS_Cdelete** is a function used to delete a number of clusters in a file (Unicode supported).
- There are three parameters.
  - Set the pointer **pFileName** to the name and the path of the file where the clusters are to be deleted from.
  - Set the second parameter, **Offset**, to the location to delete clusters in units of number of clusters from the beginning of the file.
  - Set the third parameter, **Number**, to the number of clusters to delete.
- The deleted cluster size is subtracted from the size of the file specified by **pFileName**. If the number of clusters specified from the Offset to the Number exceeds the cluster that includes the **EOF**, all clusters up to the one that includes the **EOF** are deleted. If a cluster is added after the cluster that includes the **EOF** by **AmbaFS_fappend**, the added cluster will not be deleted.
- The **AmbaFS_Cdelete** function returns the number of clusters that are actually deleted when it successfully deletes clusters. If the return value is smaller than Number, cluster deletion fails. Call the **AmbaFS_GetError** API to get details on error information.

Parameters:

Type	Parameter	Description
const char*	**pFileName**	Pointer to file name
UINT32	**Offset**	Location to delete a cluster (cluster offset number)
UINT32	**Number**	Number of clusters to delete

*Table 2-64. Parameters for File System API **AmbaFS_Cdelete()**.*

Returns:

Return	Description
Same value as **Number**	Completed successfully (bytes of Number are added)
Different value from **Number**	Failed to delete the number of clusters (Only the bytes of return are deleted)

*Table 2-65. Returns for File System API **AmbaFS_Cdelete()**.*

Example:

```
…
…
/* combine the sample file and the foobar file */
AmbaFS_Combine("A:\\sample.txt", "A:\\dir\\foobar.txt");
…
/* insert cluster to the sample file */
AmbaFS_Cinsert("A:\\sample.txt", 5, 2);
…
/* delete cluster from the sample file */
AmbaFS_Cdelete("A:\\sample.txt", 10, 1);
…
/* divide the sample file */
AmbaFS_Divide("A:\\sample.txt", "A:\\dir\\foobar.txt", 1024);
…
…
```

See Also:

AmbaFS_Cinsert()

2.2.33 AmbaFS_Combine

API Syntax:

int **AmbaFS_Combine** (char *pFileNameBase, char *pFileNameAdd)

Function Description:

- **AmbaFS_Combine** is a function that is used to combine two files into one file (Unicode supported).
- There are two parameters.
  - Set the first parameter, **pFileNameBase**, to a pointer which points to the name and path of the file to be located at the beginning of the combined file.
  - Set the second parameter, **pFileNameAdd**, to a pointer which points to the name and path of the file to be located at the latter part of the combined file.
- If the end of the file specified by **pFileNameBase** does not reach a cluster boundary, the data in the file specified by **pFileNameAdd** is combined after the data from **EOF** to the cluster boundary.
- If the files are combined successfully, the entry for the file specified by **pFileNameAdd** will be deleted.
- The **AmbaFS_Combine** function returns 0 when it successfully combines the files, and returns -1 when it fails. Call the **AmbaFS_GetError** API to get details on error information.

Parameters:

Type	Parameter	Description
char*	**pFileNameBase**	Pointer to file name of the base file to be combined
char*	**pFileNameAdd**	Pointer to file name of the file to be added

*Table 2-66. Parameters for File System API **AmbaFS_Combine()**.*

Returns:

Return	Description
0	Completed successfully
- 1	Failed to combine the files

*Table 2-67. Returns for File System API **AmbaFS_Combine()**.*

Example:

```
…
…
/* combine the sample file and the foobar file */
AmbaFS_Combine("A:\\sample.txt", "A:\\dir\\foobar.txt");
…
/* insert cluster to the sample file */
AmbaFS_Cinsert("A:\\sample.txt", 5, 2);
…
/* delete cluster from the sample file */
AmbaFS_Cdelete("A:\\sample.txt", 10, 1);
…
/* divide the sample file */
AmbaFS_Divide("A:\\sample.txt", "A:\\dir\\foobar.txt", 1024);
…
…
```

See Also:

AmbaFS_Divide()

2.2.34 AmbaFS_Divide

API Syntax:

int **AmbaFS_Divide** (char *pOriginPath, char *pNewPath, UINT32 Offset)

Function Description:

- **AmbaFS_Divide** is a function used to divide a specified file into two files at a byte boundary (Unicode supported).
- There are three parameters.
  - Set the first parameter, **pOriginPath**, to a pointer to point to the name and path of the file to be divided.
  - Set the second parameter, **pNewPath**, to a pointer to point to the name and path of the file to be created after division.
  - Set the third parameter, **Offset**, to a byte offset from the beginning of the file to the file division location.
- If Offset is not at a cluster boundary, one new cluster is allocated as the first cluster of the file created after division. This new cluster will contain undefined data up to the Offset and the original file data following it.
- The data from the byte offset location to the end of the cluster is copied to a location equivalent to the byte offset of the beginning cluster of the file created after division. If the byte offset is at a cluster boundary, no new cluster will be allocated for the first cluster of the file created after division.
- If a cluster is added to the file specified by **pOriginPath** by calling **AmbaFS_fappend**, the added cluster area is moved to the end of the file created after division.
- The **AmbaFS_Divide** function returns 0 when it successfully divides the file and returns -1 when it fails. Call the **AmbaFS_GetError** API to get the error information.

Parameters:

Type	Parameter	Description
char*	**pOriginPath**	Pointer to path and file name
char*	**pNewPath**	Pointer to new file name
UINT32	**Offset**	Byte count from reference point

*Table 2-68. Parameters for File System API **AmbaFS_Divide()**.*

Returns:

Return	Description
0	Completed successfully
- 1	Failed to divide the file

*Table 2-69. Returns for File System API **AmbaFS_Divide()**.*

Example:

```
…
…
/* combine the sample file and the foobar file */
AmbaFS_Combine("A:\\sample.txt", "A:\\dir\\foobar.txt");
…
/* insert cluster to the sample file */
AmbaFS_Cinsert("A:\\sample.txt", 5, 2);
…
/* delete cluster from the sample file */
AmbaFS_Cdelete("A:\\sample.txt", 10, 1);
…
/* divide the sample file */
AmbaFS_Divide("A:\\sample.txt", "A:\\dir\\foobar.txt", 1024);
…
…
```

See Also:

AmbaFS_Combine()

2.2.35 AmbaFS_Mount

API Syntax:

int **AmbaFS_Mount** (char *pOriginPath, char *pNewPath, UINT32 Offset)

Function Description:

- **AmbaFS_Mount** is a function that is used to mount a drive.
- Set the parameter **Drive** to the drive name to be mounted.
- This drive name is not case sensitive. This parameter can be set to any drive name from A to Z.
- The ***AmbaFS_Mount*** function returns 0 when drive mounting is successful and returns -1 when it fails. Call the **AmbaFS_GetError** API to get the error information.
- A drive cannot be mounted if it is not attached.

Parameters:

Type	Parameter	Description
char	**Drive**	The drive name

*Table 2-70. Parameters for File System API **AmbaFS_Mount()**.*

Returns:

Return	Description
0	Completed successfully
- 1	Failed to mount the drive

*Table 2-71. Returns for File System API **AmbaFS_Mount()**.*

Example:

None

See Also:

AmbaFS_Unmount()

2.2.36 AmbaFS_UnMount

API Syntax:

int **AmbaFS_UnMount** (char *pOriginPath, char *pNewPath, UINT32 Offset)

Function Description:

- **AmbaFS_UnMount** is a function used to release a mounted drive.
- Set the parameter **Drive** to the drive name to be mounted.
- This drive name is not case sensitive. This parameter can be set to any drive name from A to Z.
- The **AmbaFS_UnMount** function returns 0 when it successfully releases the drive and returns -1 when it fails. Call the **AmbaFS_GetError** API to get the error information.

Parameters:

Type	Parameter	Description
char	**Drive**	The drive name

*Table 2-72. Parameters for File System API **AmbaFS_Mount()**.*

Returns:

Return	Description
0	Completed successfully
- 1	Failed to release the drive

*Table 2-73. Returns for File System API **AmbaFS_Mount()**.*

Example:

None

See Also:

AmbaFS_Mount()

2.2.37 AmbaFS_GetError

API Syntax:

void **AmbaFS_GetError** (int *pError)

Function Description:

- **AmbaFS_GerErrpr** is a function which is used to get the latest error code for all APIs.
- Set the parameter **pError** as the pointer to save the error number.
- The ***pError** would be 0 when no error occurs and would be a non-zero value when an error occurs.

Parameters:

Type	Parameter	Description
int*	**pError**	Pointer to save the error number

*Table 2-74. Parameters for File System API **AmbaFS_GetError()**.*

Returns:

None

Example:

None

See Also:

None

Appendix 1 Additional Resources

Please contact an Ambarella representative for related resources.

Appendix 2 Important Notice

All Ambarella design specifications, datasheets, drawings, files, and other documents (together and separately, "materials") are provided on an "*as is*" basis, and Ambarella makes no warranties, expressed, implied, statutory, or otherwise with respect to the materials, and expressly disclaims all implied warranties of noninfringement, merchantability, and fitness for a particular purpose. The information contained herein is believed to be accurate and reliable. However, Ambarella assumes no responsibility for the consequences of use of such information.

Ambarella Incorporated reserves the right to correct, modify, enhance, improve, and otherwise change its products and services at any time and to discontinue any product or service without notice. Customers should obtain the latest relevant information before placing orders and should verify that such information is current and complete.

All products are sold subject to Ambarella's terms and conditions of sale supplied at the time of order acknowledgment. Ambarella warrants performance of its hardware products to the specifications applicable at the time of sale in accordance with its standard warranty. Testing and other quality control techniques are used to the extent Ambarella deems necessary to support this warranty.

Ambarella assumes no liability for applications assistance or customer product design. Customers are responsible for their products and applications using Ambarella components. To minimize the risks associated with customer products and applications, customers should provide adequate design and operating safeguards.

Ambarella does not warrant or represent that any license, either expressed or implied, is granted under any Ambarella patent right, copyright, mask work right, or other intellectual property right relating to any combination, machine, or process in which Ambarella products or services are used. Information published by Ambarella regarding third-party products or services does not constitute a license from Ambarella to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from Ambarella under the patents or other intellectual property of Ambarella.

Reproduction of information from Ambarella documents is not permissible without prior approval from Ambarella.

Ambarella products are not authorized for use in safety-critical applications (such as life support) where a failure of the product would reasonably be expected to cause severe personal injury or death, unless officers of the parties have executed an agreement specifically governing such use. Customers acknowledge and agree that they are solely responsible for all legal, regulatory and safety-related requirements concerning their products and any use of Ambarella products in such safety-critical applications, notwithstanding any applications-related information or support that may be provided by Ambarella. Further, Customers must fully indemnify Ambarella and its representatives against any damages arising out of the use of Ambarella products in such safety-critical applications.

Ambarella products are neither designed nor intended for use in military/aerospace applications or environments. Customers acknowledge and agree that any such use of Ambarella products is solely at the Customer's risk, and they are solely responsible for compliance with all legal and regulatory requirements in connection with such use.

Appendix 3 Revision History

NOTE: Page numbers for previous drafts may differ from page numbers in the current version.

Version	Date	Comments
	1 March 2013	Formatting
0.1	13 Mar 2013	Rename document title Removed APIs - ff_get_env, ff_get_env_vol, AmbaFS_getfileinstance
	15 Mar 2013	Modified examples
	25 Mar 2013	Formatting
	29 Mar 2013	Modifying of parameters
	1 Apr 2013	Update master pages
	12 April 2013	Refine some functions
	26 April 2013	Preliminary version
1.1	18 Oct 2013	Remove Unicode-related functions
1.2	21 Oct 2013	Refine all descriptions; update formatting
1.3	24 Oct 2013	Update function descriptions for 2.2.10, 2.2.11, 2.2.12, and 2.2.13.
1.4	6 May 2014	Remove API - AmbaFS_fstat; Add APIs - AmbaFS_fappend, AmbaFS_CleanDir, AmbaFS_ChmodDir, AmbaFS_Cinsert, AmbaFS_Cdelete, AmbaFS_Combine, AmbaFS_Divide, AmbaFS_Mount, AmbaFS_Unmount, AmbaFS_GerError; Modified the prototype and name style to sync with the latest SDK; Correct the examples.
1.5	27 Jan 2015	Formatted to SDK6

Table A3-1. Revision History.