exFAT and SafeexFAT File System User Guide

Version 1.60

For use with exFAT and SafeexFAT File Systems versions 1.17 and above

Table of Contents

1. System Overview

This chapter contains the fundamental information for this module.

The component sections are as follows:

- Introduction – describes the main elements of the module.
- Feature Check – summarizes the main features of the module as bullet points.
- Packages and Documents – the *Packages* section lists the packages that you need in order to use this module. The *Documents* section lists the relevant user guides.
- Change History – lists the earlier versions of this manual, giving the software version that each manual describes.

1.1. Introduction

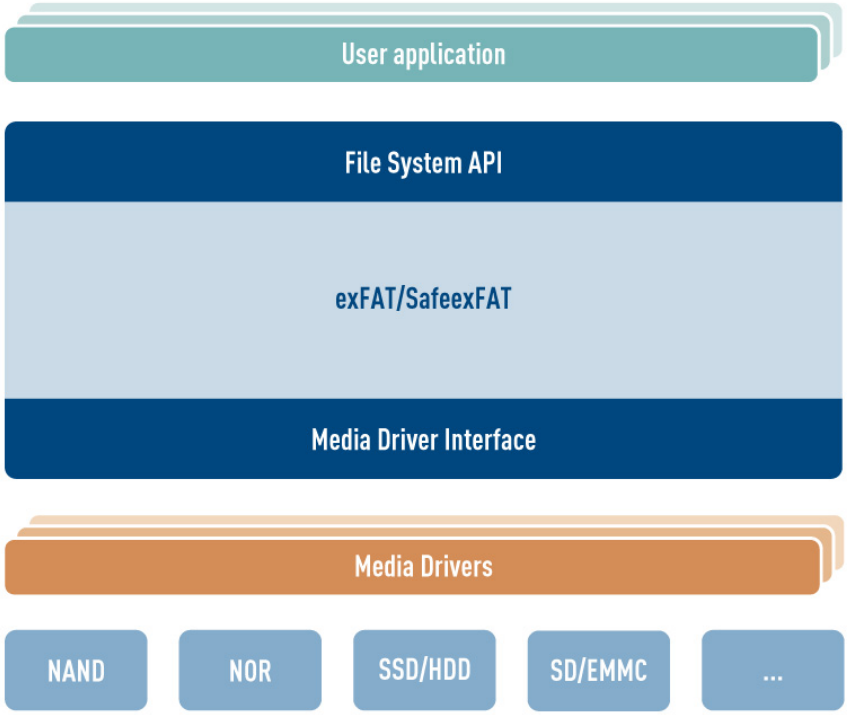
This guide is for those who wish to implement an exFAT or SafeexFAT file system. It describes HCC Embedded's exFAT and SafeexFAT file system products.

The exFAT (Extended File Allocation Table) file system is a proprietary Microsoft system. It is optimized for use with flash memory such as USB flash drives and SD cards. exFAT handles extremely large file sizes such as those used for audio and video and enables seamless file exchange between devices that use removable storage, whatever the device or operating system.

In summary:

- exFAT is a lightweight file system like FAT 32 ("lightweight" because it lacks NTFS's extra features and their associated overheads).
- exFAT supports greater file size and partition size limits than FAT 32. FAT 32 has a 4GB maximum file size and 8TB maximum partition size, whereas you can store files that are larger than 4GB each on a flash drive or SD card formatted with exFAT. exFAT's maximum file size limit is 16EiB (Exbibyte).
- exFAT is compatible with more devices than NTFS, making it the system to use when copying/sharing large files between OSes. The Mac OS X has only read-only support for NTFS, but offers full read/write support for exFAT. exFAT drives can also be accessed on Linux after installing the appropriate exFAT drivers. Note, of course, that much older devices may only support FAT 32 and not exFAT.
- exFAT is designated by the Secure Digital (SD) Card Association as the standard file system for high-capacity, high-speed SDXC cards.
- The SafeexFAT extension is designed to be truly fail-safe, protecting against unexpected reset or power loss.

The diagram below summarizes the exFAT and SafeexFAT architecture.



User applications use the standard file Application Programming Interface (API) to issue file system commands to the exFAT file system. The exFAT/SafeexFAT file system makes use of media drivers to access one or more storage media to execute the requested storage operation.

There are two possible setups. exFAT itself operates in the same way in both of these cases:

- exFAT package alone – in this case the log process files are not used and the **exfat_repair()** function is not available.
- exFAT package plus SafeexFAT – SafeexFAT stores a set of safe log files that contain system information for the last correct state. If an operation is interrupted, for example by a power outage, the **exfat_repair()** function can be used to process these log files and restore the last correct state.

This file system can be run through Linux FUSE (Filesystem in User space) by using the separate *exFAT and SafeexFAT for Linux FUSE* package.

HCC Embedded is a licensed supplier of exFAT implementations and can provide a full technology and patent license solution for incorporation into customers' devices. This means:

- For those who already have a Microsoft license for exFAT, HCC can supply its exFAT software implementation.
- For those who do not have a Microsoft license for exFAT, HCC can provide a Microsoft-approved license for exFAT and supply its exFAT software implementation.

Note:

- HCC offers hardware and firmware development consultancy to assist developers with the implementation of various types of file system.
- Although every attempt has been made to simplify the system's use, developers must have a good understanding of the requirements of the systems they are designing in order to obtain the maximum practical benefits.

1.2. Feature Check

The main features of Microsoft's exFAT are the following:

- Almost unlimited card storage – exFAT means devices can handle growing requirements for media file storage, raising capacity from 32 GB to 256 TB.
- Handles vast amounts of media in one directory – exFAT can handle more than 100 HD movies, 4000 RAW images, or 60 hours of HD recording in a single directory.
- Interoperability between systems and devices – exFAT supports interoperability between many operating systems, so there's no need to keep reformatting files and media.
- Fast transfer speeds – file saves on SDXC cards can achieve their full speed of 300 MBps.
- Provides an extensible format – this includes parameters that OEMs can define to customize exFAT for specific devices.

The main features of the HCC system are the following:

- Conforms to the HCC Advanced Embedded Framework.
- Designed for integration with both RTOS and non-RTOS based systems.
- Provides fail-safety (when used with SafeexFAT extension).
- Linux FUSE integration available.
- Cache options for optimal performance.
- Code size 35 KB (exFAT) or 47 KB (with SafeexFAT).
- RAM usage >16 KB (exFAT) or >18 KB (with SafeexFAT).
- ANSI ‘C’.
- Unicode 16.
- Multiple open files.
- Multiple users of open files.
- Multiple volumes.
- Multi-sector read/write.
- Variable sector sizes.
- Partition handling.
- Handles media errors.
- Test suite.
- Zero copy.
- Re-entrant.
- Boundary alignment offset for the FAT table.
- Boundary alignment offset for the data region.

1.3. Packages and Documents

Packages

This table lists the packages that need to be used with this module, and also optional linked modules:

Package	Description
hcc_base_doc	This contains the two guides that will help you get started.
fs_exfat	The exFAT file system package described in this document.
fs_exfat_safe	The SafeexFAT extension for the file system, also described in this document.
fs_exfat_test	The exFAT and SafeexFAT test suite; this has its own manual.
psp_template_base	The Platform Support Package (PSP) base package.
oal_base	The OS Abstraction Layer (OAL) base package.
media_drv_base	The Media Driver base package that provides the base for all media drivers that attach to the file system.
fs_fuse_exfat	This is needed to run exFAT through Linux FUSE; this has its own manual.

Documents

For an overview of HCC file systems and guidance on choosing a file system, see Product Information on the main HCC website.

Readers should note the points in the HCC Documentation Guidelines on the HCC documentation website.

HCC Firmware Quick Start Guide

This document describes how to install packages provided by HCC in the target development environment. Also follow the *Quick Start Guide* when HCC provides package updates.

HCC Source Tree Guide

This document describes the HCC source tree. It gives an overview of the system to make clear the logic behind its organization.

HCC exFAT and SafeexFAT File System User Guide

This is this document.

HCC exFAT and SafeexFAT Test Suite User Guide

This document describes the test suite used to test exFAT and SafeexFAT operation.

HCC exFAT and SafeexFAT for Linux FUSE User Guide

This document describes the package that can be used to run exFAT and SafeexFAT through Linux FUSE.

1.4. Change History

This section describes past changes to this manual.

- To download this manual see File System PDFs.
- For the history of changes made to the package code itself, see History: fs_exfat.

The current version of this manual is 1.60. The full list of versions is as follows:

Manual version	Date	Software version	Reason for change
1.60	2020-01-28	1.17	Moved EXFAT_ENABLE_LAST_ACCESS_UPDATE configuration option and added EXFAT_COVERAGE option. Changed **exfat_get_volume_list()** and also **exfat_getcwd()**, **exfat_getdcwd()**, **exfat_open()**, **exfat_open_nonsafe()**, **exfat_truncate()**, **exfat_getlabel()** and **exfat_setlabel()**. Changed *space.free* to *space.free_*
1.50	2019-10-02	1.15	Removed EXFAT_SYSTEM_ARCH_64BIT configuration option. EXFAT_ENABLE_CACHE_CFG_CHECK default changed to 0.
1.40	2019-09-18	1.14	Changed **exfat_open()**, added **exfat_open_nonsafe()**. Added EXFAT_ENABLE_NONSAFE configuration option, removed two "consts" from **config_exfat.c**. Added one line to **f_setlabel()**.
1.30	2019-08-06	1.12	Added configuration option EXFAT_ENABLE_REMOVE_CONTENT and function **exfat_remove_content()**. Changed code in **config_exfat.c**.
1.20	2019-07-23	1.08	Added configuration option EXFAT_SECTOR_BUFFER_COUNT.
1.10	2019-07-23	1.06	Added "and SafeexFAT" to title. Added SafeexFAT information including **exfat_repair()**, and the *t_exfat_format_param* structure. Removed description of **psp_w16xxx()** functions from *PSP Porting* as these are now part of the **psp_base** template. Added section *Unicode string literals* to PSP Porting and HCC_UTF macro to many code examples.
1.00	2019-02-13	1.01	First version.

2. System Description

This section describes the fundamental elements of exFAT/SafeexFAT.

2.1. Lower Layer Requirements

In order for a file system that claims fail-safety to be able to ensure correct operation, it has to specify the minimum requirements that must be satisfied by the media interface below it. For example, suppose that a low level HDD driver has a large cache that can be written to the disk. If, when an unexpected reset occurs there's no guarantee that all data are written, it is unlikely that any system will be able to ensure a consistent state of that disk.

For SafeexFAT the requirements are:

- Any sectors written to the disk are committed to the disk before the next write is started.
- Any sector written to the disk is updated atomically. That is, in all cases either the original contents of the sector are present or the new data are present; there are no intermediate states.
- If an unexpected reset condition is reached, the file system is restarted. No attempt is made to continue to use the system after a serious condition is detected.

If these conditions are not met, the system cannot be guaranteed fail-safe. However, even if they are not met, the system is much safer than a standard unprotected exFAT file system.

Guaranteeing that these conditions are fulfilled is not always easy. The vast majority of flash card vendors do not provide detailed information about how their cards work. This makes it very difficult to define how a system will behave when used with media whose behavior is undefined.

HCC Embedded works closely with a number of card manufacturers to provide solutions in which target devices have been designed to meet the above criteria. HCC has a test system in place to verify whether flash cards meet the required standards. Although HCC's tests cannot prove conclusively that a card is reliable, as defined above, they give a very good indication of the level of reliability that can be expected.

2.2. Volume Boot Record: VBR

The Volume Boot Record (VBR) is a 12 sector area that contains the boot records, BIOS Parameter Block (BPB), OEM parameters, and the checksum sector.

There is also a backup VBR that contains a copy of the first 12 sectors of the volume.

2.3. Metadata and Checksums

exFAT introduces metadata integrity through the use of three checksums:

- The VBR checksum sector. This is a checksum of the previous 11 sectors in the VBR, except for three bytes in the boot sector (used for flags and the percentage used). This tests the integrity of the VBR by checking whether it was modified. The commonest cause is a boot sector virus, but the check detects any other corruption of the VBR.
- An UpCase table checksum. This table is a static table and should never change. Any corruption in the table could prevent files from being located because the table is used to convert the filenames to upper case when searching for a file.
- The directory file sets checksum. Multiple directory records are used to define a single file and this is called a file set. This file set has metadata including the file name, time stamps, attributes, address of first cluster location of the data, file lengths, and the file name. A checksum is taken over the entire file set and a mismatch occurs if the directory file set is accidentally or maliciously changed.

When the file system is mounted, the integrity check is conducted and these hashes are verified. At mount time the exFAT file system version is checked by the driver to make sure the driver is compatible with the file system it wants to mount, and to check that none of the required directory records are missing. For example, the directory records for the UpCase table and Allocation Bitmap are required; the file system cannot run without them. If any of these checks fails, the file system should not be mounted, although it may mount as read-only in certain cases.

2.4. Allocation Bitmap

The allocation bitmap keeps track of the clusters' allocation status. This makes the process of determining whether a cluster is free for writing very fast.

The bitmap comprises a number of 8 bit bytes, treated as a bit sequence. Each bit corresponds to a data cluster. It has a value of 1 if the cluster is allocated, or 0 if the cluster is unallocated.

The bitmap tracks clusters by using the least significant bit within a byte to represent the allocation status of the first cluster in the range. The least significant bit of the bitmap table refers to the first cluster, cluster 2. The first byte covers clusters 2 to 9, the second byte maps clusters 10 to 17, and so on. For example, if just the first four clusters covered by a byte are allocated, its binary value would be 0000 1111.

2.5. UpCase Table

exFAT is case-insensitive and it has to convert to upper case the characters in file names during search operations. The UpCase table holds the data used for conversion from lower case to upper case characters.

The UpCase table is an array of Unicode characters. It has an index that represents each Unicode character to be converted to upper case, then the value which is the target upper case character. The UpCase table must contain at least 128 mandatory Unicode mappings.

This example shows typical entries from the table. These are part of the mandatory set:

Index	Value	Note
0x0061	0x0041	"a" is mapped to "A".
0x0062	0x0042	"b" is mapped to "B".
0x0063	0x0043	"c" is mapped to "C".

Normally the UpCase table is located right after the allocation bitmap but it can be placed anywhere in the cluster heap. It has a corresponding primary critical directory entry in the root directory.

2.6. Pre-allocating File Space

exFAT can pre-allocate disk space for a file by simply marking arbitrary space on the disk as 'allocated'. A file can be pre-allocated as a very large size in an attempt to obtain many contiguous clusters in a single allocation.

For each file, exFAT uses two separate 64 bit length fields in the directory:

- Valid Data Length (VDL) – the real size of the file, the amount of data written.
- Physical data length.

Windows 10 saves the same value for Valid Data Length and Physical data length and the HCC implementation does the same.

2.7. File Name Lookup

exFAT uses a hash-based filename lookup phase to speed up certain searches.

2.8. Directory File Sets

Like other FAT file systems, exFAT does not use indexes for file names. When a file is accessed, the directory is searched sequentially until a match is found. For file names shorter than 16 characters in length, one file name record is required but the entire file is represented by three 32-byte directory records. This is called a directory file set. A 256 MiB sub-directory can hold up to 2,796,202 file sets. (If files have longer names, this number is reduced but this is the maximum based on the minimum three-record file set.)

When searching for a file:

1. The file name is converted to upper case using the UpCase table (as explained above, file names are case-insensitive).
2. Each record in the directory is searched by comparing the file name.
3. When a match is found, the file names are compared to ensure that the proper file was located.

This improves performance as only two bytes need to be compared for each file. This significantly cuts the CPU cycles because most file names are more than two characters (bytes) in size; almost every comparison is performed on just two bytes at a time until the desired file is located.

2.9. Timestamps

This table shows the components of an exFAT timestamp:

Bits	Size	Description	Note
0-4	5	Seconds (number of 2-second increments). For example 10 equals 20 seconds.	0..59
5-10	6	Minutes	0 .. 59
11-15	5	Hour	0 .. 23
16-20	5	Day	1 .. 31
21-24	4	Month	1 .. 12
25-31	7	Year (the offset from 1980)	0 equals 1980

Note:

- The timestamp format records seconds in two second intervals. 10ms increments are used to improve the precision of creation and modification times from two seconds to 10 milliseconds. The valid values are from 0 to 199 in 10ms intervals; these are added to the timestamp.
- Timestamp granularity for Last Access time is only to double seconds (FAT has date only).
- Timestamps use the time of the local time zone.

2.10. SafeexFAT Log Files

SafeexFAT stores a set of safe log files that contain system information for the last correct state. If an operation is interrupted, for example by a power outage, the **exfat_repair()** function can be used to process these log files and restore the last correct state.

3. Other File System Information

This section:

- describes the system, stack, and real time requirements.
- describes the functions exFAT provides for creating and managing multiple drives, partitions and volumes.

3.1. System Requirements

The exFAT system is designed to be as open and portable as possible. No assumptions are made about the functionality or behavior of the underlying operating system.

For the system to work at its best, perform the porting work outlined in the following sections. This is a straightforward task for an experienced engineer.

3.2. Stack Requirements

File system functions are always called in the context of the calling thread or task. Naturally, the functions require stack space and you should allow for this in applications that call file system functions. Typically, calls to the file system use less than 2 KB of stack.

3.3. Real-Time Requirements

The bulk of the file system is code that executes without delay. There are exceptions at the driver level, where delays in reading and writing from/to the physical media, and in the communication itself, cause the system to wait on external events. The points at which delays occur are documented in the relevant driver documents.

Modify drivers to meet the exFAT system's requirements, either by implementing interrupt control of the relevant events, or scheduling other parts of the system that can proceed without completion of the events. Refer to the relevant driver documents for details.

3.4. Drives, Partitions and Volumes

This HCC implementation of exFAT supports multiple drives and currently supports only one partition per drive.

First, note the following definitions:

- A drive consists of a physical medium that is controlled by a single driver. Examples are an HDD and a Compact Flash card.
- All drives contain zero or more partitions. If a drive is not partitioned, there is just a single volume on that drive.
- A single volume may be added to each partition. A volume can exist on a drive without partitions.

exFAT operates on volumes. You can have one volume or a set of volumes. Additional functions are provided to work with multi-volume sets (A:, B:, C:, and so on).

Note: The API functions **exfat_getdrive()**, **exfat_chdrive**(), and **exfat_getdcwd()** refer to drives by name because this is the convention, but the names are really references to volumes.

Partitions are created on a single volume such as an HDD, so a single driver is used to access the volume even though there are multiple partitions on it. These volumes need to be controlled by a single lock.

Note: Some operating systems do not recognize multiple partitions on removable media. It is therefore "normal" to restrict the use of multiple partitions to fixed drives.

4. Source File List

This section lists and describes all the source code files included in the system. These files follow HCC Embedded's standard source tree system, described in the *HCC Source Tree Guide*. All references to file pathnames refer to locations within this standard source tree, not within the package you initially receive.

Note: Do not modify any files except the configuration files.

API Header File

The file **src/api/api_exfat.h** must be included by any application using the system. It includes all that is required to access the system. The use of these API functions is defined in Application Programming Interface. **This file should only be modified by HCC**.

Configuration Files

The following files in the directory **src/config** contain all the configurable parameters of the system. Configure these as required.

File	Package	Description
config_exfat.c	**fs_exfat**	Defines the FAT/directory cache used, based on the number of sectors on the media. Also defines the cluster sizes used by **exfat_format()**.
config_exfat.h	**fs_exfat**	Configuration options (with EXFAT_ENABLE_SAFE set to 0).
config_exfat.h	**fs_exfat_safe**	Configuration options (with EXFAT_ENABLE_SAFE set to 1).

Version File

The file **src/version/ver_exfat.h** contains the version number of this module. This version number is checked by all modules that use this module to ensure system consistency over upgrades.

Log Process Files

These files, used in a SafeexFAT system by **exfat_repair()**, are in the **fs_exfat_safe** package in the directory **src/exfat/safe**:

File	Description
exfat_log.c and **.h**	Log functions and values.
exfat_log_entry.c and **.h**	Log entry functions and values.

Source Code

These files are in the directory **src/exfat/common**. **These files should only be modified by HCC**.

File	Description
exfat.c and **.h**	Main functions.
exfat_bitmap.c and **.h**	Bitmap functions.
exfat_cache.c and **.h**	Cache buffer functions.
exfat_convert.c and **.h**	Various conversion functions.
exfat_dir.c and **.h**	Directory functions.
exfat_direntry.c and **.h**	Directory entry functions.
exfat_driver.c and **.h**	Driver functions (reading/writing sectors and clusters).
exfat_driver_low.c and **.h**	Driver low level sector operations.
exfat_fat.c and **.h**	Main functions.
exfat_file.c and **.h**	File functions.
exfat_types.h	Various definitions and structures.
exfat_upcase_table.c and **.h**	Upcase table.

5. Configuration Options

The following sections describe the configuration files:

- **config_exfat.h** – lists the available configuration options and their default values.
- **config_exfat.c** – contains parameters used for cache and volume formatting.

5.1. config_exfat.h

Set the following configuration options in the file **src/config/config_exfat.h**. This section lists the available configuration options and their default values.

> **Note:** There is a copy of this file in both the **fs_exfat** and **fs_exfat_safe** packages. The only difference between these is in the default setting of EXFAT_ENABLE_SAFE.

EXFAT_MAX_VOLUME_COUNT

The maximum number of volumes allowed on the system. The default is 1. Volumes are given drive letters as specified by **exfat_initvolume()**.

The system is designed so that access to a specific volume is entirely independent of any other volumes. That is, if an operation is being performed on a volume it does not block access to other volumes.

EXFAT_MAX_TASK_COUNT

The number of tasks that are allowed to access the file system simultaneously. The default is 2.

EXFAT_MAX_FILE_HANDLE_COUNT

The total number of files that may be open simultaneously across all volumes. The default is 5.

EXFAT_MAX_DIR_HANDLE_COUNT

The total number of directories that may be open simultaneously across all volumes. The default is 5.

EXFAT_MAX_SECTOR_SIZE

The maximum sector size of the attached media. Valid values are 512, 1024, 2048, and 4096. The default is 512.

For devices whose native sector size is not 512 bytes (for example, 2K page NAND flash-based devices), it can be most efficient to use another value.

EXFAT_MAX_PATH_LENGTH

The minimum EXFAT_MAX_FILE_NAME_LENGTH + 1, excluding the trailing zero character. The default is 512.

EXFAT_MAX_FILE_NAME_LENGTH

The maximum length of a filename with the full path (excluding the trailing zero character). The default is 255, which is also the maximum allowed.

EXFAT_PATH_SEPARATOR_CHAR

The default is '/'. Set this to '\\' to use backslash as the pathname separator character. These two are the only valid values.

EXFAT_CLUSTER_CACHE_SIZE

Use of cluster cache can speed up seeking in large files. The default is 32. Every element of cache adds 8 bytes for each file handle; to disable this and save the space, set this option to 0.

EXFAT_MAX_DIR_DEPTH

The maximum directory depth to handle. If this is too small, an EXFAT_ERR_DIRECTORY_TOO_DEEP error may be returned. The default is 32 and the minimum is 1.

EXFAT_SEEK_FILL_CHARACTER

The fill character to use when seeking beyond the file size. The default is 0.

EXFAT_ENABLE_CACHE_CFG_CHECK

This has two options:

- 0 - disables the check. This is the default.
- 1 - check the cache configuration, calculate the optimal values for pool size and buffer count, and print these using **psp_printf()**. We recommend setting this to 1 after changing the EXFAT_CACHE_POOL_SIZE, EXFAT_CACHE_MAX_BUF_COUNT, or *g_exfat_cache_config[]*.

EXFAT_ENABLE_CACHE

This has two options:

- 0: disables sector caching. This runs the system with the absolute minimum configuration.
- 1: enables sector caching using the cache configuration in **config_exfat.c**. This is the default.

> **Note:** The following two options only apply if EXFAT_ENABLE_CACHE is set to 1.

EXFAT_CACHE_POOL_SIZE

The number of bytes in the cache pool for each volume. The default is (259584).

The amount of RAM needed in bytes is `EXFAT_CACHE_POOL_SIZE * EXFAT_MAX_VOLUME_COUNT.`

EXFAT_CACHE_MAX_BUF_COUNT

The maximum number of buffers for the cache. The default is 83.

EXFAT_SECTOR_BUFFER_COUNT

This is used by **exfat_format()** and **exfat_mkdir()**. The minimum and default are 1 but a larger value may speed up formatting and directory creation, though it uses more static memory.

Used memory is calculated as:
EXFAT_MAX_VOLUME_COUNT * EXFAT_MAX_SECTOR_SIZE * EXFAT_SECTOR_BUFFER_COUNT

EXFAT_ENABLE_SAFE

This has two options:

- 0: Normal (non-safe) operation. This is the default for this value in the **fs_exfat** package.
- 1: Safe operation (the **fs_exfat_safe** package is needed to use this and this is the default for this value in that package).

EXFAT_ENABLE_LAST_ACCESS_UPDATE

This option is only used if safe mode is disabled (EXFAT_ENABLE_SAFE is 0). Set this to 1 to enable updating of the 'last access' timestamp. File entries are updated on every call of **exfat_close()** for reading as well. The default is 0.

EXFAT_ENABLE_NONSAFE

This option is only used if EXFAT_ENABLE_SAFE is set non-zero. It has two options:

- 0: Only safe operations are available.
- 1: Safe and non-safe file operations are available. This is the default for this value in both packages.

EXFAT_ENABLE_REMOVE_CONTENT

This has two options:

- 0: **exfat_remove()** is the only **remove** function available. This removes the file without clearing its content.
- 1 (the default): **exfat_remove_content()** is also available. This removes the file clusters and erases all data before removing the file.

EXFAT_COVERAGE

Use this option to disable part of the code for coverage testing purposes. Setting it to 1 disables a few PSP_ASSERT calls, but is necessary to run the test cases EXFAT_TEST_VOLUME_LABEL, EXFAT_TEST_FIND_ALLOCATION_BITMAP_ERROR, EXFAT_TEST_GETFREESPACE_ERROR, EXFAT_TEST_MKDIR_ERROR, and EXFAT_TEST_SAFE_LOG_OPEN_ERROR. Otherwise the code will run into an assertion error. The default is 0.

5.2. config_exfat.c

Cache Memory

The memory allocated dynamically at initialization for the file system to use is controlled by the *g_exfat_cache_config[]* array in the file **src/config/config_exfat.c**. This array is only used when EXFAT_ENABLE_CACHE is set to 1; otherwise static memory buffers are used, with their sizes configured by EXFAT_CACHE_POOL_SIZE and EXFAT_CACHE_MAX_BUF_COUNT in **config_exfat.h**.

The *g_exfat_cache_config_default[]* array defines the FAT/directory cache used, based on the number of sectors present on the media. Also see Cache Definitions.

```
static t_exfat_cache_config  g_exfat_cache_config_default[EXFAT_CACHE_TYPE_COUNT] =
{
/* Type of cache: directory entry, allocation bitmap, etc.                    */
/* See EXFAT_CACHE_TYPE_*                                                     */
/*      |                     Count of sectors for each cache buffer (min 1). */
/*      |                                    |   Count of buffers (min 1).     */
/*      |                                    |   |                             */
  {   EXFAT_CACHE_TYPE_FILE_CONTENT,      1,  EXFAT_MAX_FILE_HANDLE_COUNT   }
  , { EXFAT_CACHE_TYPE_UPCASE_TABLE,      12, 1                             }
  , { EXFAT_CACHE_TYPE_ALLOCATION_BITMAP, 1,  1                             }
  , { EXFAT_CACHE_TYPE_BOOT,              1,  1                             }
  , { EXFAT_CACHE_TYPE_FAT,               1,  16                            }
  , { EXFAT_CACHE_TYPE_DIRECTORY,         8,  UINT16_MAX                    }
};

t_exfat_cache_config * g_exfat_cache_config[EXFAT_MAX_VOLUME_COUNT] =
{
  g_exfat_cache_config_default
 #if ( EXFAT_MAX_VOLUME_COUNT > 1 )
  , g_exfat_cache_config_default
 #endif
 #if ( EXFAT_MAX_VOLUME_COUNT > 2 )
  , g_exfat_cache_config_default
 #endif
 #if ( EXFAT_MAX_VOLUME_COUNT > 3 )
  , g_exfat_cache_config_default
 #endif
};
```

exfat_format() Cluster Sizes

The following code is used by the **exfat_format()** function. If the media capacity is less than or equal to *capacity*, the corresponding sectors/cluster and boundary units are used in creating the file system.

```
static const t_exfat_format_param  g_exfat_format_params_default[] =
{
  /* capacity <=        sectors/  boundary unit */
  /* (MiB)              cluster   (sectors)     */
  {   128,              8,        512   }
  , { 256,              8,        1024  }
  , { 2088,             64,       8192  }

  /* The following values are from */
  /* "SD Specifications Part 2 File System Specification Version 3.00 */
  /* April 16, 2009", Table A-15 */
  , { 32896,            256,      32768 }
  , { 512 * 1024,       512,      65536 }
  , { 2 * 1024  * 1024, 1024,     131072 }
  , { 0, 0, 0 } /* End of table */
};

const t_exfat_format_param * g_exfat_format_params = g_exfat_format_params_default;
```

6. Application Programming Interface

This section describes all the Application Programming Interface (API) functions. It includes all the functions that are available to an application program.

6.1. Module Management

The functions are the following:

Function	Description
exfat_init()	Initializes the file system and allocates the required resources.
exfat_start()	Starts the file system.
exfat_stop()	Stops the file system.
exfat_delete()	Releases resources allocated during the initialization of the file system.

exfat_init

Use this function to initialize the file system. Call it once at start-up.

Format

```
t_exfat_ret exfat_init ( void )
```

Arguments

None.

Return Values

Return value	Description
EXFAT_NO_ERROR	Successful execution.
EXFAT_ERR_NOT_ENOUGH_RESOURCE	Insufficient resources available.

Example

```
void main()
{
    exfat_init();   /* Initialize the file system */
    exfat_start();  /* Start the file system */
    .
    .
}
```

exfat_start

Use this function to start the file system.

This function must complete successfully before the file system can be used.

> **Note:** Call **exfat_init()** before this to initialize the file system.

Format

```
t_exfat_ret exfat_start ( void )
```

Arguments

None.

Return Values

Return value	Description
EXFAT_NO_ERROR	Successful execution.
Else	See Error Codes.

exfat_stop

Use this function to stop the file system.

After this, the file system cannot be used until a new call to **exfat_start()** is successfully completed.

Format

```
t_exfat_ret exfat_stop ( void )
```

Arguments

None.

Return Values

Return value	Description
EXFAT_NO_ERROR	Successful execution.
Else	See Error Codes.

exfat_delete

Use this function to release resources allocated during the initialization of the file system.

Note: All volumes must be deleted before this function is called.

Format

```
t_exfat_ret exfat_delete ( void )
```

Arguments

None.

Return Values

Return value	Description
EXFAT_NO_ERROR	Successful execution.
EXFAT_ERR_BUSY	A volume has not been deleted and this prevented the successful completion of this function.

6.2. File System API

This section describes the available Application Programming Interface (API) functions. It is split into functions for task management, volume management, directory management, file access, and file management.

Task Management

The functions are the following:

Function	Description
exfat_enter_task()	Adds the current task to the list of tasks.
exfat_exit_task()	Removes the current task from the list of tasks.

exfat_enter_task

Use this function to add the current task to the list of tasks.

Format

```
t_exfat_ret exfat_enter_task ( void )
```

Arguments

None.

Return values

Return value	Description
EXFAT_NO_ERROR	Successful execution.
EXFAT_ERR_NOT_ENOUGH_RESOURCE	Insufficient resources available for this call.

Example

```
void task_add_to_list()
{
    exfat_enter_task(); /* Add the current task to current task list */
    .
    .
    .
}
```

exfat_exit_task

Use this function to remove the current task from the list of tasks.

Format

```
void exfat_exit_task ( void )
```

Arguments

None.

Return values

None.

Example

```
void task_cut_from_list()
{
    exfat_exit_task(); /* Remove the current task from current task list */
    .
    .
    .
}
```

Volume Management

Note: The API functions **exfat_getdrive()**, **exfat_chdrive()** and **exfat_getdcwd()** use the term "drive" because this is the convention. This is equivalent to the term "volume".

The functions are the following:

Function	Description
exfat_initvolume()	Initializes a volume.
exfat_delvolume()	Deletes an existing volume.
exfat_repair()	Repairs an existing volume (**SafeexFAT only**). This processes the log files and restores the last good state.
exfat_get_volume_count()	Gets the number of volumes currently available to the user.
exfat_get_volume_list()	Gets a list of volumes currently available to the user.
exfat_get_volume_info()	Gets a volume's cluster size and sector size.
exfat_format()	Formats the specified drive.
exfat_chdrive()	Changes to a new current drive.
exfat_getdrive()	Gets the current drive number.
exfat_getfreespace()	Fills a structure with information about the drive space usage: total space, free space, used space, and bad (damaged) size.
exfat_getlabel()	Returns the label as a function value.
exfat_setlabel()	Sets a volume label.

exfat_initvolume

Use this function to initialize an exFAT volume. This attaches the media driver and also validates the Master Boot Record (MBR) and Volume Boot Record (VBR). It always initiates the first partition on the media.

This function works independently of the status of the hardware; that is, it does not matter whether a card is inserted or not.

In non-multitask systems this call must be followed by a call to **exfat_chdrive()** to select the current drive for relative file path accessing.

Format

```
int exfat_initvolume (
    t_exfat_drive const   drivenum,
    F_DRIVERINIT          driver_init,
    uint32_t              driver_param )
```

Arguments

Argument	Description	Type
drivenum	The drive to initialize (0='A', 1='B', and so on).	t_exfat_drive
driver_init	The media driver's initialization function. This is called to retrieve drive configuration information from the relevant driver.	F_DRIVERINIT
driver_param	This can optionally be used to pass information to the low level driver. Its use is driver-dependent. When the **xxx_initfunc()** of the driver is called, this parameter is passed to the driver. One use for this is to specify which device associated with the specified driver will be initialized.	uint32_t

Return values

Return value	Description
EXFAT_NO_ERROR	Successful execution.
EXFAT_ERR_REPAIR_NEEDED	The volume needs to be repaired. See exfat_repair().
EXFAT_ERR_NOT_FORMATTED	The volume is not formatted. See exfat_format().

Example

```
void myinitfs( void )
{
    int ret;
    exfat_init();   /* Initialize the file system */
    exfat_start();  /* Start the file system */

    /* Create a RAM volume on Drive A */
    exfat_initvolume( 0, ram_initfunc, 0 );

    /* Create a Compact Flash Volume on Drive B */
    exfat_initvolume( 1, cfc_initfunc, 0 );

    /* Create an MMC Volume on Drive C */
    exfat_initvolume( 2, mmc_initfunc, 0 );

    /* Create a Mass Storage Volume on Drive D */
    exfat_initvolume( 3, mst_initfunc, 0 );

    /* Create a second Mass Storage Volume on Drive E */
    exfat_initvolume( 4, mst_initfunc, 1 );
    .
    .
    .
}
```

exfat_delvolume

Use this function to delete an existing volume.

Note that:

- The link between the file system and the driver is broken; that is, an **xxx_release()** call is made to the driver.
- Any open files on the media are marked as closed, so that subsequent API calls that access a previously opened file handle return an error.

This function works independently of the status of the hardware; that is, it does not matter whether a card is inserted or not.

Format

```
t_exfat_ret exfat_delvolume ( t_exfat_drive const drivenum )
```

Arguments

Argument	Description	Type
drivenum	The drive to delete (0='A', 1='B', and so on).	t_exfat_drive

Return values

Return value	Description
EXFAT_NO_ERROR	Successful execution.
Else	See Error Codes.

Example

```
void mydelfs( int num )
{
    int ret;

    /* Delete volume */
    if (exfat_delvolume( num ))
        printf( "Unable to delete volume %c", &#39;A&#39; + num );
        .
        .
}
```

exfat_repair

Use this function to repair an existing volume.

Note:

- **This is only available with SafeexFAT.**
- If the drive is not repaired, its content can be read but cannot be changed.
- If a previous safe operation was interrupted, safe logs will exist. This function processes these log files and restores the last good state.

Format

```
t_exfat_ret exfat_repair ( t_exfat_drive const drivenum )
```

Arguments

Argument	Description	Type
drivenum	The drive to repair (0='A', 1='B', and so on).	t_exfat_drive

Return values

Return value	Description
EXFAT_NO_ERROR	Successfully repaired volume.
Else	See Error Codes.

Example

```
t_exfat_ret ret;

ret = exfat_initvolume( 0, mmcsd_initfunc, 0 );

if ( ret == EXFAT_ERR_REPAIR_NEEDED )
{
    ret = exfat_repair( 0 );
}
```

exfat_get_volume_count

Use this function to get the number of active volumes.

Format

```
t_exfat_ret exfat_get_volume_count ( uint8_t * const p_volume_count )
```

Arguments

Argument	Description	Type
p_volume_count	On return, a pointer to the number of active volumes.	uint8_t *

Return values

Return value	Description
EXFAT_NO_ERROR	Successful execution.
Else	See Error Codes.

Example

```
void mygetvols( void )
{
    uint8_t volume_count;
    if ( exfat_get_volume_count( &volume_count ) == EXFAT_NO_ERROR )
    {
        printf( "There are %d active volumes\n", volume_count );
        .
        .
    }
}
```

exfat_get_volume_list

Use this function to obtain a list of all the active (initialized) volumes.

Format

```
t_exfat_ret exfat_get_volume_list (
    uint8_t * const   p_volume_list,
    uint8_t           volume_list_size )
```

Arguments

Argument	Description	Type
p_volume_list	A pointer to the volume list to fill. This array should be of size EXFAT_MAX_VOLUME_COUNT.	uint8_t *
volume_list_size	The size of the list.	

Return values

Return value	Description
number	The number of active volumes.
Else	See Error Codes.

Example

```
void mygetvols( void )
{
    t_exfat_ret ret;
    int j;
    uint8_t buffer[EXFAT_MAX_VOLUME_COUNT];
    ret = exfat_get_volume_list( buffer, EXFAT_MAX_VOLUME_COUNT );
    if (ret != EXFAT_NO_ERROR)
    {
        printf( "No active volume found\n" );
    }
    else
    {
        for (j = 0; j< EXFAT_MAX_VOLUME_COUNT; j++)
        {
            if (buffer[j] < EXFAT_MAX_VOLUME_COUNT)
            {
                printf( "Volume %d is active\n", buffer[j] );
            }
        }
    }
}
```

exfat_get_volume_info

Use this function to get a volume's cluster size and sector size.

Format

```
t_exfat_ret exfat_get_volume_info (
    t_exfat_drive const           drivenum,
    t_exfat_volume_info * const   p_volume_info )
```

Arguments

Argument	Description	Type
drivenum	The drive number (0='A', 1='B', and so on).	t_exfat_drive
p_volume_info	Where to write the cluster size.	t_exfat_volume_info *

Return values

Return value	Description
EXFAT_NO_ERROR	Successful execution.
EXFAT_ERR_INVALID_PARAMETER	A parameter is invalid.

Example

```
void myvolumeinfo( void )
{
    t_exfat_ret ret;
    t_exfat_volume_info volume_info;

    /* Get cluster size of Drive A */
    ret = exfat_get_volume_info ( 0, &volume_info );

    if ( ret == EXFAT_NO_ERROR )
    {
        printf( "The cluster_size is %d \n", volume_info.cluster_size_byte );
    }
}
```

exfat_format

Use this function to format the specified drive for exFAT.

Cluster sizes can be configured for **exfat_format()** in the file **config_exfat.c**.

Note:

- If the media is not present this function fails. **If it succeeds, all data on the specified volume are destroyed and any open files are closed.**
- Any existing Master Boot Record is unaffected by this command. The boot sector information is re-created from the information provided by **exfat_getphy()**.

Format

```
t_exfat_ret exfat_format ( t_exfat_drive const drivenum )
```

Arguments

Argument	Description	Type
drivenum	The drive to format (0='A', 1='B', and so on).	t_exfat_drive

Return values

Return value	Description
EXFAT_NO_ERROR	Successful execution.
EXFAT_ERR_MEDIA_DRIVER	Media driver error.

Example

```
void myinitfs( void )
{
    int ret;

    exfat_init();   /* Initialize the file system */
    exfat_start();  /* Start the file system */

    exfat_initvolume( 0, cfc_initfunc, 0 );

    ret = exfat_format( 0 );

    if (ret)
    {
        printf( "Unable to format CFC: Error %d", ret );
    }
    else
    {
        printf( "CFC formatted" );
    }
    .
    .
}
```

exfat_chdrive

Use this function to change to a new current drive.

In non-multitasking and multitasking systems, call **exfat_chdrive()** if you need relative path access. In a multitasking system, and in a non-multitasking system after **exfat_initvolume()**, every **exfat_enter_task()** must be followed by an **exfat_chdrive()** function call. In a multitasking system every task has its own current drive.

Format

```
t_exfat_ret exfat_chdrive ( t_exfat_drive const drivenum )
```

Arguments

Argument	Description	Type
drivenum	The number of the drive to change to (0='A', 1='B', and so on).	t_exfat_drive

Return values

Return value	Description
EXFAT_NO_ERROR	Successful execution.
EXFAT_ERR_INVALID_VOLUME	Drive number is invalid.

Example

```
void myfunc( void )
{
    .
    .
    exfat_chdrive( 0 );    /* Change to drive A */
    .
    .
}
```

exfat_getdrive

Use this function to get the current drive number of a task.

Format

```
t_exfat_ret exfat_getdrive ( t_exfat_drive * const p_drivenum )
```

Arguments

Argument	Description	Type
p_drivenum	On return, a pointer to the drive number (0='A', 1='B', and so on).	t_exfat_drive *

Return values

Return value	Description
EXFAT_NO_ERROR	Successful execution.

Example

```
void myfunc( void )
{
    t_exfat_drive currentdrive;
    if ( exfat_getdrive( &currentdrive ) == EXFAT_NO_ERROR )
    {
        printf( "The current drive is %d \n", currentdrive );
    }
}
```

exfat_getfreespace

Use this function to fill a structure with information about the drive space usage: total space, free space, used space, and bad (damaged) size.

Note:

- The space is stored in 64 bit variables.
- The first call to this function after a drive is mounted may take some time, depending on the size and format of the medium being used. After the initial call, changes to the volume are counted; the function then returns immediately with the data.

Format

```
t_exfat_ret exfat_getfreespace (
    t_exfat_drive const     drivenum,
    t_exfat_space * const   p_space )
```

Arguments

Argument	Description	Type
drivenum	The drive number (0='A', 1='B', and so on).	t_exfat_drive
p_space	A pointer to the *t_exfat_space* structure.	t_exfat_space *

Return values

Return value	Description
EXFAT_NO_ERROR	Successful execution.
Else	See Error Codes.

Example

```
void info( void )
{
    t_exfat_space space;
    t_exfat_ret ret;
    t_exfat_drive drive;

    /* Get free space on current drive */
    ret = exfat_getdrive( &drive );
    if ( ret == EXFAT_NO_ERROR )
    {
        ret = exfat_getfreespace( drive, &space );
        if ( ret == EXFAT_NO_ERROR )
        {
            printf( "There are:\
              %d bytes total,\
              %d bytes free,\
              %d bytes used,\
              %d bytes bad.",\
              space.total, space.free_, space.used, space.bad );
        }
        else
        {
            printf( "\nError %d reading drive\n", ret );
        }
    }
}
```

exfat_getlabel

Use this function to get the drive's volume label, a Unicode string.

Format

```
t_exfat_ret exfat_getlabel (
    t_exfat_drive const   drivenum,
    wchar16_t *           p_label,
    uint32_t const        label_size )
```

Arguments

Argument	Description	Type
drivenum	The drive number (0='A', 1='B', and so on).	t_exfat_drive
p_label	On return, a pointer to the Unicode string of the volume label field.	wchar16_t *
label_size	The size of the input buffer.	uint32_t

Return values

Return value	Description
EXFAT_NO_ERROR	Successful execution.
Else	See Error Codes.

Example

```
void getlabel( void )
{
    wchar16_t label[12];
    t_exfat_ret result;
    t_exfat_drive drive;

    result = exfat_getdrive( &drive );
    if ( result == EXFAT_NO_ERROR )
    {
        result = exfat_getlabel( drive, &label, 12 );
    }
    if ( result != EXFAT_NO_ERROR )
    {
        printf( "Error on drive!\n" );
    }
    else
    {
        printf( "Drive is %ls\n", label );
    }
}
```

exfat_setlabel

Use this function to set a volume label.

This changes the label of the volume and this label is written to the media. If the media is inserted in a PC, the host OS can display it.

Format

```
t_exfat_ret exfat_setlabel (
    t_exfat_drive const   drivenum,
    const wchar16_t       p_label[] )
```

Arguments

Argument	Description	Type
drivenum	The drive number (0='A', 1='B', and so on).	t_exfat_drive
p_label[]	The null-terminated Unicode string to use to overwrite the drive's volume label field. The volume label must be a Unicode string with a maximum length of 11 characters. Non-printable characters are padded out as space characters.	wchar16_t

Return values

Return value	Description
EXFAT_NO_ERROR	Successful execution.
EXFAT_ERR_INVALID_LABEL_LENGTH	Label length is invalid.

Example

```
void setlabel( void )
{
    t_exfat_ret result;
    t_exfat_drive drive;

    result = exfat_getdrive( &drive );
    if ( result == EXFAT_NO_ERROR )
    {
        /* Note that UTF-16 string literal starts with u */
        result = exfat_setlabel( drive, u"DRIVE 1" );
    }
    if ( result != EXFAT_NO_ERROR )
    {
        printf( "Error on drive!\n" );
    }
}
```

Directory Management

The functions are the following:

Function	Description
exfat_mkdir()	Creates a new directory.
exfat_chdir()	Changes the current working directory.
exfat_rmdir()	Removes a directory.
exfat_is_directory()	Checks whether a directory exists.
exfat_getcwd()	Gets the current working directory.
exfat_getdcwd()	Gets the current working directory on the selected drive.
exfat_opendir()	Opens a directory. After this call directory entries can be read by using **exfat_readdir()**.
exfat_closedir()	Closes a directory that was opened by using **exfat_opendir()**.
exfat_readdir()	Reads a file or subdirectory from a specified open directory.
exfat_rewinddir()	Resets an open directory's read position.

exfat_mkdir

Use this function to create a new directory.

Format

```
t_exfat_ret exfat_mkdir ( wchar16_t const * const p_dirname )
```

Arguments

Argument	Description	Type
p_dirname	A pointer to the name of the directory to create.	wchar16_t *

Return values

Return value	Description
EXFAT_NO_ERROR	Successful execution.
EXFAT_ERR_ALREADY_EXISTS	There is a directory with this name already.
EXFAT_ERR_MEDIA_WRITE_PROTECTED	The media is in write-protected state.

Example

```
void myfunc( void )
{
    .
    .
    exfat_mkdir( u"subfolder" );  /* Create directories */
    exfat_mkdir( u"subfolder/sub1" );
    exfat_mkdir( u"subfolder/sub2" );
    exfat_mkdir( u"a:/subfolder/sub3" );
    .
    .
}
```

exfat_chdir

Use this function to change the current working directory.

Every relative path starts from this directory. In a multitasking system every task has its own current working directory.

Format

```
t_exfat_ret exfat_chdir ( wchar16_t const * const p_dirname )
```

Arguments

Argument	Description	Type
p_dirname	A null-terminated string containing the name of the directory to change to.	wchar16_t *

Return values

Return value	Description
EXFAT_NO_ERROR	Successful execution.
Else	See Error Codes.

Example

```
void myfunc( void )
{
    .
    .
    exfat_mkdir( u"subfolder" );
    exfat_chdir( u"subfolder" );       /* Change directory */
    exfat_mkdir( u"sub2" );
    exfat_chdir( u".." );              /* Go up one directory level */
    exfat_chdir( u"subfolder/sub2" );  /* Go into directory sub2 */
    .
    .
}
```

exfat_rmdir

Use this function to remove a directory.

Format

```
t_exfat_ret exfat_rmdir ( wchar16_t const * const p_dirname )
```

Arguments

Argument	Description	Type
p_dirname	A pointer to the name of the directory to remove.	wchar16_t *

Return values

Return value	Description
EXFAT_NO_ERROR	Successful execution.
EXFAT_ERR_DIRECTORY_NOT_EMPTY	The directory has contents so cannot be deleted.
EXFAT_ERR_FILE_NOT_FOUND	The directory does not exist.
EXFAT_ERR_ACCESS_DENIED	No permission to access this directory; it is read-only.

Example

```
void myfunc( void )
{
    .
    .
    exfat_mkdir( u"subfolder" );       /* Create directories */
    exfat_mkdir( u"subfolder/sub1" );
    .
    . /* Do some work */
    .
    exfat_rmdir( u"subfolder/sub1" );  /* Remove directories */
    exfat_rmdir( u"subfolder" );
    .
    .
}
```

exfat_is_directory

Use this function to check whether a directory exists.

Format

```
t_exfat_ret exfat_is_directory ( wchar16_t const * const p_dir )
```

Arguments

Argument	Description	Type
p_dir	A pointer to the directory pathname.	wchar16_t *

Return values

Return value	Description
EXFAT_NO_ERROR	The directory exists.
EXFAT_ERR_FILE_NOT_FOUND	The directory was not found.

Example

```
void isdir( void )
{
    t_exfat_ret result;
    result = exfat_is_directory( u"DIR12" );
    if ( result != EXFAT_NO_ERROR )
    {
        printf( "Directory not found!\n" );
    }
}
```

exfat_getcwd

Use this function to get the current working directory on the current drive.

Format

```
t_exfat_ret exfat_getcwd (
    wchar16_t * const   p_buffer,
    uint32_t const      buffer_size )
```

Arguments

Argument	Description	Type
p_buffer	Where to store the current working directory string.	wchar16_t *
buffer_size	The length of the buffer.	uint32_t

Return values

Return value	Description
EXFAT_NO_ERROR	Successful execution.
EXFAT_ERR_INVALID_PARAMETER	Either *p_buffer* is null or *buffer_size* is not set.

Example

```
#define BUFFLEN EXFAT_MAX_PATH_LENGTH + 1 /* +1 for trailing zero */

void myfunc( void )
{
    wchar16_t buffer[BUFFLEN];

    if (exfat_getcwd( &buffer, BUFFLEN ) == EXFAT_NO_ERROR )
    {
        printf( "Current directory is %ls\n", &buffer );
    }
    else
    {
        printf( "Drive error!\n" )
    }
}
```

exfat_getdcwd

Use this function to get the current working directory on the selected drive.

Format

```
t_exfat_ret exfat_getdcwd (
    t_exfat_drive const   drivenum,
    wchar16_t * const     p_buffer,
    uint32_t const        buffer_size )
```

Arguments

Argument	Description	Type
drivenum	The drive number (0='A', 1='B', and so on).	t_exfat_drive
p_buffer	Where to store the current working directory string.	wchar16_t *
buffer_size	The length of the buffer.	uint32_t

Return values

Return value	Description
EXFAT_NO_ERROR	Successful execution.
EXFAT_ERR_INVALID_PARAMETER	Either *p_buffer* is null or *buffer_size* is not set.

Example

```
#define BUFFLEN EXFAT_MAX_PATH_LENGTH + 1  /* +1 for trailing zero */
void myfunc( t_exfat_drive drivenum )
{
    wchar16_t buffer[BUFFLEN];
    if ( exfat_getdcwd( drivenum, &buffer, BUFFLEN ) == EXFAT_NO_ERROR )
    {
        printf( "Current directory is %ls", &buffer );
        printf( "on drive %c\n", drivenum+&#39;A&#39; );
    }
    else
    {
        printf( "Drive error!\n" )
    }
}
```

exfat_opendir

Use this function to open a directory.

After this call you can read directory entries by using **exfat_readdir()**.

Format

```
t_exfat_ret exfat_opendir (
    wchar16_t const * const      p_dirname,
    t_exfat_dir_handle * const   p_dir_handle )
```

Arguments

Argument	Description	Type
p_dirname	The path of the directory to open.	wchar16_t *
p_dir_handle	On return, a pointer to the directory handle.	t_exfat_dir_handle *

Return values

Return value	Description
EXFAT_NO_ERROR	Successful execution.
EXFAT_ERR_NOT_ENOUGH_RESOURCE	No free directory handle available.

Example

```
void mydir( void )
{
    t_exfat_dir_handle dir_handle;

    if ( exfat_opendir( u"subdir2", &dir_handle ) == EXFAT_NO_ERROR )
    {
        printf( "Opened directory\n" );
    }
    else
    {
        printf( "Error! Directory not opened\n" )
    }
}
```

exfat_closedir

Use this function to close a directory that was opened by using **exfat_opendir()**.

Format

```
t_exfat_ret exfat_closedir ( t_exfat_dir_handle dir_handle )
```

Arguments

Argument	Description	Type
dir_handle	The directory handle.	t_exfat_dir_handle

Return values

Return value	Description
EXFAT_NO_ERROR	Successful execution.
EXFAT_ERR_INVALID_HANDLE	The handle is invalid.

Example

```
void mydir( void )
{
    t_exfat_dir_handle dir_handle;
    exfat_opendir( u"subdir2", &dir_handle )
        .
        . / * Do some work */
        .
    if ( exfat_closedir( dir_handle ) == EXFAT_NO_ERROR )
    {
        printf( "Closed directory\n" );
    }
    else
    {
        printf( "Error! Directory not closed\n" )
    }
}
```

exfat_readdir

Use this function to read a file or subdirectory from a specified open directory.

The directory must be opened first by using **exfat_opendir()**.

Format

```
t_exfat_ret exfat_readdir (
    t_exfat_dir_handle            dir_handle,
    t_exfat_dir_entry * * const   pp_dir_entry )
```

Arguments

Argument	Description	Type
dir_handle	The handle of an open directory.	t_exfat_dir_handle
pp_dir_entry	On return, a pointer to the directory entry.	t_exfat_dir_entry **

Return values

Return value	Description
EXFAT_NO_ERROR	Successful execution.
EXFAT_ERRINVALID_PARAMETER	Invalid directory handle.

Example

```
void mydir( void )
{
    t_exfat_dir_handle dir_handle;
    t_exfat_dir_entry * p_dir_entry;
    if ( exfat_opendir( u"subdir2" , &dir_handle ) == EXFAT_NO_ERROR )
    {
        if ( exfat_readdir ( dir_handle, &p_dir_entry ) == EXFAT_NO_ERROR )
        {
            printf( "%ls size: %lld\n"
                    , p_dir_entry->filename
                    , p_dir_entry->filesize );
        }
    }
}
```

exfat_rewinddir

Use this function to reset an open directory's read position.

The directory must be opened with **exfat_opendir()** before using this function.

Format

```
t_exfat_ret exfat_rewinddir ( t_exfat_dir_handle dir_handle )
```

Arguments

Argument	Description	Type
dir_handle	The directory handle.	t_exfat_dir_handle

Return values

Return value	Description
EXFAT_NO_ERROR	Successful execution.
EXFAT_ERR_INVALID_HANDLE	The handle is invalid.

Example

```
void mydir( void )
{
    t_exfat_dir_handle dir_handle;
    exfat_opendir( u"subdir2" , &dir_handle )
    .
    . / * Do some work */
    .
    if ( exfat_rewinddir( dir_handle ) == EXFAT_NO_ERROR )
    {
        printf( "Reset directory read position\n" );
    }
    else
    {
        printf( "Error! Read position not reset\n" )
    }
}
```

File Access

The functions are the following:

Function	Description
exfat_open()	Opens a file.
exfat_close()	Closes a file.
exfat_flush()	Flushes an opened file to a storage medium.
exfat_read()	Reads bytes from a file at the current file position.
exfat_write()	Writes data into a file at the current file position.
exfat_getc()	Reads a character from the current position in an open file.
exfat_putc()	Writes a character to an open file at the current file position.
exfat_eof()	Checks whether the current position in an open file is the end of file (EOF).
exfat_seteof()	Moves the end of file (EOF) to the current file pointer.
exfat_tell()	Obtains the current read/write position in an open file.
exfat_seek()	Moves the stream position in a file.
exfat_rewind()	Sets the file position in an open file to the start of the file.
exfat_truncate()	Opens a file for writing and truncates it to the specified length.
exfat_ftruncate()	Truncates a file that is open for writing to a specified length.

exfat_open

Use this function to open a file for reading/writing/appending. If safe mode is enabled (EXFAT_ENABLE_SAFE is 1) and open mode is "w", "a", "r+", "w+" or "a+", this call opens the file in safe mode.

The possible opening modes are as follows:

Mode	Description
"r"	Open existing file for reading. The stream is positioned at the beginning of the file.
"r+"	Open existing file for reading and writing. The stream is positioned at the beginning of the file.
"w"	Truncate file to zero length or create file for writing. The stream is positioned at the beginning of the file.
"w+"	Open a file for reading and writing. The file is created if it does not exist; otherwise it is truncated. The stream is positioned at the beginning of the file.
"a"	Open for appending (writing to end of file). The file is created if it does not exist. The stream is positioned at the end of the file.
"a+"	Open for reading and appending (writing to end of file). The file is created if it does not exist. The stream is positioned at the end of the file.

Note the following:

- The same file can be opened multiple times in “r” mode.
- A file can only be opened once at a time in a mode which gives write access (that is, in “r+, “w”, “w+” , “a” or “a+” mode).
- The same file can be opened multiple times in “r” mode and at the same time once in one of the “r+, “a” or “a+” modes which give write access.
- If a file is opened in “w” or “w+” mode, a lock mechanism prevents it being opened in any other mode. This prevents opening of the file for reading and writing at the same time.

Note: There is no text mode. The system assumes that all files are in binary mode only.

Format

```
t_exfat_ret exfat_open (
    wchar16_t const * const       p_path,
    char_t const * const          p_mode,
    t_exfat_file_handle * const   p_file_handle )
```

Arguments

Argument	Description	Type
p_path	A pointer to the file to open.	wchar16_t *
p_mode	A pointer to the opening mode (see above).	char_t *
p_file_handle	On return, a pointer to the file handle.	t_exfat_file_handle *

Return values

Return value	Description
EXFAT_NO_ERROR	Successful execution.
EXFAT_ERR_NOT_ENOUGH_RESOURCE	No free handle was found.
EXFAT_ERR_MEDIA_WRITE_PROTECTED	The media is write protected and tried to open for writing.
EXFAT_ERR_FILE_NOT_FOUND	The file was not found.
EXFAT_ERR_IS_DIRECTORY	The "file" is a directory.

Example

```
void myfunc( void )
{
    t_exfat_file_handle file;
    t_exfat_ret ret;
    uint32_t bytes_read;
    char c;

    ret = exfat_open( HCC_UTF( "myfile.bin" ), "r", &file );
    if (ret != EXFAT_NO_ERROR )
    {
        printf( "File cannot be opened!\n" );
    }
    else
    {
        ret = exfat_read( &c, 1, file, &bytes_read );  /* Read one byte */
        if ( ret == EXFAT_NO_ERROR && bytes_read == 1 )
        {
            printf( "&#39;%c&#39; is read from file\n", c );
        }
        exfat_close( file );
    }
}
```

exfat_open_nonsafe

Use this function to open a file for reading/writing/appending in non-safe mode.

> **Note:** This is only available if EXFAT_ENABLE_SAFE is set.

The possible opening modes are as follows:

Mode	Description
"r"	Open existing file for reading. The stream is positioned at the beginning of the file.
"r+"	Open existing file for reading and writing. The stream is positioned at the beginning of the file.
"w"	Truncate file to zero length or create file for writing. The stream is positioned at the beginning of the file.
"w+"	Open a file for reading and writing. The file is created if it does not exist; otherwise it is truncated. The stream is positioned at the beginning of the file.
"a"	Open for appending (writing to end of file). The file is created if it does not exist. The stream is positioned at the end of the file.
"a+"	Open for reading and appending (writing to end of file). The file is created if it does not exist. The stream is positioned at the end of the file.

Note the following:

- The same file can be opened multiple times in “r” mode.
- A file can only be opened once at a time in a mode which gives write access (that is, in “r+, “w”, “w+” , “a” or “a+” mode).
- The same file can be opened multiple times in “r” mode and at the same time once in one of the “r+, “a” or “a+” modes which give write access.
- If a file is opened in “w” or “w+” mode, a lock mechanism prevents it being opened in any other mode. This prevents opening of the file for reading and writing at the same time.

> **Note:** There is no text mode. The system assumes that all files are in binary mode only.

Format

```
t_exfat_ret exfat_open_nonsafe (
    wchar16_t const * const       p_path,
    char_t const * const          p_mode,
    t_exfat_file_handle * const   p_file_handle )
```

Arguments

Argument	Description	Type
p_path	A pointer to the file to open.	wchar16_t *
p_mode	A pointer to the opening mode (see above).	char_t *
p_file_handle	On return, a pointer to the file handle.	t_exfat_file_handle *

Return values

Return value	Description
EXFAT_NO_ERROR	Successful execution.
EXFAT_ERR_NOT_ENOUGH_RESOURCE	No free handle was found.
EXFAT_ERR_MEDIA_WRITE_PROTECTED	The media is write protected and tried to open for writing.
EXFAT_ERR_FILE_NOT_FOUND	The file was not found.
EXFAT_ERR_IS_DIRECTORY	The "file" is a directory.

Example

```
void myfunc( void )
{
    t_exfat_file_handle file;
    t_exfat_ret ret;
    uint32_t bytes_read;
    char c;

    ret = exfat_open_nonsafe( HCC_UTF( "myfile.bin" ), "r", &file );
    if (ret != EXFAT_NO_ERROR )
    {
        printf( "File cannot be opened!\n" );
    }
    else
    {
        ret = exfat_read( &c, 1, file, &bytes_read );  /* Read one byte */
        if ( ret == EXFAT_NO_ERROR && bytes_read == 1 )
        {
            printf( "&#39;%c&#39; is read from file\n", c );
        }
        exfat_close( file );
    }
}
```

exfat_close

Use this function to close a previously opened file.

Format

```
t_exfat_ret exfat_close ( t_exfat_file_handle const file_handle )
```

Arguments

Argument	Description	Type
file_handle	The handle of the file.	t_exfat_file_handle

Return values

Return value	Description
EXFAT_NO_ERROR	Successful execution.
EXFAT_ERR_INVALID_HANDLE	Invalid file handle.

Example

```
void myfunc( void )
{
    t_exfat_file_handle *file;
    t_exfat_ret ret;
    uint32_t bytes_written;
    char *string = "ABC";

    ret = exfat_open( HCC_UTF( "myfile.bin" ), "w", &file );
    if ( ret != EXFAT_NO_ERROR )
    {
        printf( "File cannot be opened!\n" );
        return;
    }

    exfat_write( string, 3, file, &bytes_written );  /* Write 3 bytes */
    if (exfat_close( file ) == EXFAT_NO_ERROR )
    {
        printf( "File stored\n" );
    }
    else
    {
        printf( "File close error!\n" );
    }
}
```

exfat_flush

Use this function to flush an opened file to a storage medium. This is logically equivalent to performing a close and open on a file to ensure the data changed before the flush is committed to the medium.

Format

```
t_exfat_ret exfat_flush ( t_exfat_file_handle const file_handle )
```

Arguments

Argument	Description	Type
file_handle	The handle of the file.	t_exfat_file_handle

Return values

Return value	Description
EXFAT_NO_ERROR	Successful execution.
EXFAT_ERR_INVALID_HANDLE	Invalid file handle.

Example

```
void myfunc( void )
{
    t_exfat_file_handle file;
    t_exfat_ret ret;
    uint32_t bytes_written;
    char *string = "ABC";

    ret = exfat_open( HCC_UTF( "myfile.bin" ), "w", &file );
    if ( ret != EXFAT_NO_ERROR )
    {
        printf( "File cannot be opened!\n" );
        return;
    }
    exfat_write( string, 3, file, &bytes_written );  /* Write 3 bytes */
    exfat_flush( file );  /* Commit data written */
    .
    .
}
```

exfat_read

Use this function to read bytes from the current position in the specified file.

The file must be opened in “r”, "r+", "w+", or "a+" mode. (See exfat_open() for details of modes).

Format

```
t_exfat_ret exfat_read (
    void * const                p_buffer,
    uint32_t const              size,
    t_exfat_file_handle const   file_handle,
    uint32_t * const            p_size_read )
```

Arguments

Argument	Description	Type
p_buffer	On return, a pointer to the buffer containing the data.	void *
size	The size of the buffer in bytes.	uint32_t
file_handle	The handle of the file.	t_exfat_file_handle
p_size_read	On return, a pointer to the number of bytes successfully read.	uint32_t *

Return values

Return value	Description
EXFAT_NO_ERROR	Successful execution.
EXFAT_ERR_INVALID_HANDLE	Invalid file handle.

Example

```
int myreadfunc( wchar16_t *filename, char *buffer, long buffsize )
{
    t_exfat_file_handle file;
    t_exfat_ret ret;
    uint32_t bytes_read;
    t_exfat_size file_size;

    ret = exfat_filelength( filename, &file_size );
    if ( ret == EXFAT_NO_ERROR )
    {
        ret = exfat_open( filename, "r", &file );
    }
    if ( ret != EXFAT_NO_ERROR )
    {
        printf( "%ls cannot be opened!\n", filename );
        return 1;
    }
    if ( file_size > buffsize )
    {
        file_size = buffsize;
    }
    exfat_read( buffer, file_size, file, &bytes_read );
    if ( file_size != bytes_read )
    {
        printf( "Some items not read!\n" );
    }
    exfat_close( file );
    return 0;
}
```

exfat_write

Use this function to write data into a file at the current position.

The file must be opened in "r+", “w”, “w+”, "a+", or “a” mode (see exfat_open() for details of modes). The file pointer is moved forward by the number of bytes successfully written.

Note: Data is not permanently stored to the media until an **exfat_flush()** or **exfat_close()** has been executed on the file.

Format

```
t_exfat_ret exfat_write (
    void const *                p_buffer,
    uint32_t const              size,
    t_exfat_file_handle const   file_handle,
    uint32_t * const            p_size_written )
```

Arguments

Argument	Description	Type
p_buffer	A pointer to the data to write.	void *
size	The size of the buffer in bytes.	uint32_t
file_handle	The handle of the file.	t_exfat_file_handle
p_size_written	On return, the number of items written.	uint32_t *

Return values

Return value	Description
EXFAT_NO_ERROR	Successful execution.
EXFAT_ERR_INVALID_HANDLE	Invalid file handle.

Example

```
void myfunc( void )
{
    t_exfat_file_handle file;
    t_exfat_ret ret;
    uint32_t chars;
    char *string = "ABC";
    ret = exfat_open( HCC_UTF( "myfile.bin" ), "w", &file );
    if (ret == EXFAT_NO_ERROR)
    {
        printf( "File cannot be opened!\n" );
        return;
    }
    /* Write 3 bytes */
    if ( exfat_write( string, 3, file, &chars ) != EXFAT_NO_ERROR )
    {
        printf( "Write error!\n" );
    }
    else if ( chars != 3 )
    {
        printf( "Some items not written!\n" );
    }
    exfat_close( file );
}
```

exfat_getc

Use this function to read a character from the current position in the specified open file.

Format

```
t_exfat_ret exfat_getc (
    t_exfat_file_handle const   file_handle,
    uint8_t * const             p_ch )
```

Arguments

Argument	Description	Type
file_handle	The handle of the open file.	t_exfat_file_handle
p_ch	A pointer to the character to read.	uint8_t *

Return values

Return value	Description
EXFAT_NO_ERROR	Successful execution.
EXFAT_ERR_END_OF_FILE	Cannot read from the file.

Example

```
int myreadfunc( wchar16_t *filename, char *buffer, long buffsize )
{
    t_exfat_ret ret;
    t_exfat_file_handle file;
    ret = exfat_open( filename, "r", &file );
    while ( buffsize-- && ret == EXFAT_NO_ERROR )
    {
        uint8_t ch;
        ret = exfat_getc( file, &ch );
        if ( ret == EXFAT_NO_ERROR || ret == EXFAT_ERR_END_OF_FILE )
        {
            *buffer++ = ch;
            buffsize--;
        }
    }
    exfat_close( file );
    return 0;
}
```

exfat_putc

Use this function to write a character to the specified open file at the current file position. The current file position is then incremented.

Format

```
t_exfat_ret exfat_putc (
    uint8_t                     ch,
    t_exfat_file_handle const   file_handle )
```

Arguments

Argument	Description	Type
ch	The character to write.	uint8_t
file_handle	The handle of the open file.	t_exfat_file_handle

Return values

Return value	Description
EXFAT_NO_ERROR	Successful execution.
EXFAT_ERR_INVALID_HANDLE	Invalid file handle.

Example

```
void myfunc (wchar16_t *filename, long num)
{
    t_exfat_ret ret;
    t_exfat_file_handle file;
    ret = exfat_open( filename, "w", &file );
    while ( num-- && ret == EXFAT_NO_ERROR )
    {
        uint8_t ch = &#39;A&#39;;
        ret = exfat_putc( ch, file );
        if (ret != EXFAT_NO_ERROR)
        {
            printf( "exfat_putc error!\n" );
        }
    }
    exfat_close( file );
}
```

exfat_eof

Use this function to check whether the current position in the specified open file is the end of file (EOF).

Format

```
t_exfat_ret exfat_eof (
    t_exfat_file_handle const   file_handle,
    uint8_t * const             pb_is_eof )
```

Arguments

Argument	Description	Type
file_handle	The handle of the open file.	t_exfat_file_handle
pb_is_eof	On return, a pointer to a Boolean value: TRUE = end of file, FALSE = another position.	uint8_t *

Return values

Return value	Description
EXFAT_NO_ERROR	Successful execution.
EXFAT_ERR_INVALID_HANDLE	Invalid file handle.

Example

```
int myreadfunc( wchar16_t *filename, char *buffer, long buffsize )
{
    t_exfat_ret ret;
    t_exfat_file_handle file;
    uint32_t chars;
    uint8_t is_eof = FALSE;
    ret = exfat_open( filename, "r", &file );
    while ( ret == EXFAT_NO_ERROR && is_eof == FALSE )
    {
        if ( buffsize == 0 )
        {
            break;
        }
        buffsize--;
        ret = exfat_read( buffer++, 1, file, &chars );
        if ( ret == EXFAT_NO_ERROR )
        {
            ret = exfat_eof( file, &is_eof );
        }
    }
    exfat_close( file );
    return 0;
}
```

exfat_seteof

Use this function to move the end of file (EOF) to the current file pointer.

Note: All data after the new EOF position are lost.

Format

```
t_exfat_ret exfat_seteof ( t_exfat_file_handle const file_handle )
```

Arguments

Argument	Description	Type
file_handle	The handle of the open file.	t_exfat_file_handle

Return values

Return value	Description
EXFAT_NO_ERROR	Successful execution.
EXFAT_ERR_INVALID_HANDLE	Invalid file handle.

Example

```
int mytruncatefunc( wchar16_t *filename, int position )
{
    t_exfat_ret ret;
    t_exfat_file_handle file;
    ret = exfat_open( filename, "r+", &file );
    if ( ret == EXFAT_NO_ERROR )
    {
        exfat_seek( file, position, EXFAT_SEEK_SET );
        if ( exfat_seteof( file ) != EXFAT_NO_ERROR )
        {
            printf( "Truncate failed!\n" );
        }
        exfat_close( file );
    }
    return 0;
}
```

exfat_tell

Use this function to get the current read/write position in the specified open file.

Format

```
t_exfat_ret exfat_tell (
    t_exfat_file_handle const   file_handle,
    t_exfat_size * const        p_offset )
```

Arguments

Argument	Description	Type
file_handle	The handle of the open file.	t_exfat_file_handle
p_offset	On return, a pointer to the actual position in the file, counted from the start of the file.	t_exfat_size *

Return values

Return value	Description
EXFAT_NO_ERROR	Successful execution.
EXFAT_ERR_INVALID_HANDLE	Invalid file handle.

Example

```
int myreadfunc( wchar16_t *filename, char *buffer, long buffsize )
{
    t_exfat_ret ret;
    t_exfat_file_handle file;
    t_exfat_size position;
    uint32_t chars;
    ret = exfat_open( filename, "r", &file );
    if ( ret == EXFAT_NO_ERROR )
    {
        exfat_tell( file, &position );
        printf( "Current position %lld\n", position ); /* Position 0 */
        exfat_read( buffer, 1, file, &chars );         /* Read one byte */
        exfat_tell( file, &position );
        printf( "Current position %lld\n", position ); /* Position 1 */
        exfat_read( buffer, 1, file, &chars );         /* Read one byte */
        exfat_tell( file, &position );
        printf( "Current position %lld\n", position ); /* Position 2 */
        exfat_close( file );
    }
    return 0;
}
```

exfat_seek

Use this function to move the stream position in the specified open file.

An optional additional non-standard flag is provided, EXFAT_SEEK_NOWRITE. Set this if you want to seek past the end of file without filling the file with zeroes. This is useful for creating large files quickly, without the normal overhead of having to write to every sector.

Note: If EXFAT_SEEK_NOWRITE is used, the contents of the extended area are undefined.

The offset position is relative to *whence.*

Format

```
t_exfat_ret exfat_seek (
    t_exfat_file_handle const   file_handle,
    int64_t const               offset,
    uint8_t const               origin )
```

Arguments

Argument	Description	Type
file_handle	The handle of the open file.	t_exfat_file_handle
offset	The byte position relative to *whence.*	int64_t
origin	Where to calculate the offset from, one of the following: • EXFAT_SEEK_CUR – current position of the file pointer. • EXFAT_SEEK_END – end of file. • EXFAT_SEEK_SET – start of file. • EXFAT_SEEK_NOWRITE – see above.	uint8_t

Return values

Return value	Description
EXFAT_NO_ERROR	Successful execution.
EXFAT_ERR_INVALID_HANDLE	Invalid file handle.

Example

```
int myreadfunc( wchar16_t *filename, char *buffer, long buffsize )
{
    t_exfat_ret ret;
    t_exfat_file_handle file;
    uint32_t chars;
    ret = exfat_open( filename, "r", &file );
    if ( ret == EXFAT_NO_ERROR )
    {
        exfat_read( buffer, 1, file, &chars );  /* Read the first byte */
        exfat_seek( file, 0, EXFAT_SEEK_SET );
        exfat_read( buffer, 1, file, &chars );  /* Read the same byte */
        exfat_seek( file, -1, EXFAT_SEEK_END );
        exfat_read( buffer, 1, file, &chars );  /* Read the last byte */
        exfat_close( file );
    }
    return 0;
}
```

exfat_rewind

Use this function to set the file position in the specified open file to the start of the file.

Format

```
t_exfat_ret exfat_rewind ( t_exfat_file_handle const file_handle )
```

Arguments

Argument	Description	Type
file_handle	The handle of the open file.	t_exfat_file_handle

Return values

Return value	Description
EXFAT_NO_ERROR	Successful execution.
EXFAT_ERR_INVALID_HANDLE	Invalid file handle.

Example

```
void myfunc( void )
{
    t_exfat_ret ret;
    t_exfat_file_handle file;
    uint32_t bytes;
    char buffer[4];
    char buffer2[4];

    ret = exfat_open( HCC_UTF( "myfile.bin" ), "r", &file );
    if (ret == EXFAT_NO_ERROR)
    {
        exfat_read( buffer, 4, file, &bytes );
        /* Rewind file pointer */
        exfat_rewind( file );

        /* Read from the beginning */
        exfat_read( buffer2, 4, file, &bytes );
        exfat_close( file );
    }

}
```

exfat_truncate

Use this function to open a file for writing and truncate it to the specified length.

If the length is greater than the length of the existing file, the file is padded with zeroes to the truncated length.

Format

```
t_exfat_ret exfat_truncate (
    wchar16_t const * const       p_filename,
    t_exfat_size const            length,
    t_exfat_file_handle * const   p_file_handle )
```

Arguments

Argument	Description	Type
p_filename	A pointer to the file to open.	wchar16_t *
length	The length to truncate the file to.	t_exfat_size
p_file_handle	A pointer to the file handle.	t_exfat_file_handle *

Return values

Return value	Description
EXFAT_NO_ERROR	Successful execution.
EXFAT_ERR_NOT_ENOUGH_RESOURCE	No free handle was found.
EXFAT_ERR_MEDIA_WRITE_PROTECTED	The media is write-protected but the call tried to open it for writing.
EXFAT_ERR_FILE_NOT_FOUND	The file was not found.
EXFAT_ERR_IS_DIRECTORY	The "file" is a directory.

Example

```
int mytruncatefunc( wchar16_t *filename, unsigned long length )
{
    t_exfat_ret ret;
    t_exfat_file_handle file;

    ret = exfat_truncate( filename, length, &file );
    if ( ret != EXFAT_NO_ERROR )
    {
        printf( "File opening error!\n" );
    }
    else
    {
        printf( "File %s truncated to %d bytes\n", filename, length );
        exfat_close( file );
    }
    return 0;
}
```

exfat_ftruncate

Use this function to truncate a file which is open for writing to a specified length.

If *length* is greater than the length of the existing file, the file is padded with zeroes to the new length.

Format

```
t_exfat_ret exfat_ftruncate (
    t_exfat_file_handle   file_handle,
    t_exfat_size const    length )
```

Arguments

Argument	Description	Type
file_handle	The file handle of the open file.	t_exfat_file_handle
length	The new length of the file.	t_exfat_size

Return values

Return value	Description
EXFAT_NO_ERROR	Successful execution.
Else	See Error Codes.

Example

```
int mytruncatefunc( t_exfat_file_handle * p_file, unsigned long length )
{
    t_exfat_ret ret;
    ret = exfat_open( HCC_UTF( "myfile.bin" ), "r", p_file );
    if ( ret == EXFAT_NO_ERROR )
    {
        ret = exfat_ftruncate( *p_file, length );
        if ( ret != EXFAT_NO_ERROR )
        {
            printf( "Error:%d\n", ret );
        }
        else
        {
            printf( "File is truncated to %d bytes\n", length );
        }
    }
    return ret;
}
```

File Management

The functions are the following:

Function	Description
exfat_remove()	Deletes a file.
exfat_remove_content()	Deletes a file. This erases all data and removes the file's clusters before removing the file.
exfat_move()	Moves a file or directory.The original file or directory is lost.
exfat_rename()	Renames a file or directory.
exfat_getattr()	Gets the attributes of a file.
exfat_setattr()	Sets the attributes of a file.
exfat_gettimestamp()	Gets time and date information from a file or directory.
exfat_settimestamp()	Sets time and date information for a file or directory.
exfat_fstat()	Gets information about a file by using the file handle.
exfat_stat()	Gets information about a file or directory.
exfat_filelength()	Gets the length of a file.
exfat_is_file()	Checks that a file exists.
exfat_exists()	Checks that a file or directory exists.

exfat_remove

Use this function to remove an existing file.

Format

```
t_exfat_ret exfat_remove ( wchar16_t const * const p_path )
```

Arguments

Argument	Description	Type
p_path	A pointer to the name of the file to remove.	wchar16_t *

Return values

Return value	Description
EXFAT_NO_ERROR	Successful execution.
Else	See Error Codes.

Example

```
void mydir( void )
{
    exfat_remove( HCC_UTF( "oldfile.txt );             /* Delete file */
    exfat_remove( HCC_UTF( "A:/subdir/oldfile.txt" );  /* Delete file from
subdirectory */
}
```

exfat_remove_content

Use this function to delete an existing file and all its content. This erases all data and removes the file's clusters before removing the file.

Note: This is only available if configuration option EXFAT_ENABLE_REMOVE_CONTENT is set to 1.

Format

```
t_exfat_ret exfat_remove_content ( wchar16_t const * const p_path )
```

Arguments

Argument	Description	Type
p_path	A pointer to the name of the file to remove.	wchar16_t *

Return values

Return value	Description
EXFAT_NO_ERROR	Successful execution.
EXFAT_ERR_INVALID_HANDLE	Invalid file handle.

Example

```
void mydir( void )
{
    exfat_remove_content ( HCC_UTF( "oldfile.txt );             /* Delete file */
    exfat_remove_content ( HCC_UTF( "A:/subdir/oldfile.txt" );  /* Delete file from
subdirectory */
}
```

exfat_move

Use this function to move a file or directory within the volume. The original file or directory is lost.

The source and target must be in the same volume. A file can be moved only if it is not open. A directory can be moved only if there are no open files in it.

A file or directory can be moved, irrespective of its attribute settings. The attribute settings are moved with it.

Format

```
t_exfat_ret exfat_move (
    wchar16_t const *   p_old_path,
    wchar16_t const *   p_new_path )
```

Arguments

Argument	Description	Type
p_old_path	A pointer to the old file or directory name, with or without its path.	wchar16_t *
p_new_path	A pointer to the new name of the file or directory, with or without the path.	wchar16_t *

Return values

Return value	Description
EXFAT_NO_ERROR	Successful execution.
Else	See Error Codes.

Example

```
void myfunc( void )
{
    .
    exfat_move( HCC_UTF( "oldfile.txt" ), HCC_UTF( "newfile.txt" );
    exfat_move( HCC_UTF( "A:/subdir/oldfile.txt" ), HCC_UTF( "A:/newdir/oldfile.txt"
);
    .
    .
}
```

exfat_rename

Use this function to rename a file or directory.

Note: The file or directory must not be read-only. If it is a file, it must not be open.

Format

```
t_exfat_ret exfat_rename (
    wchar16_t const *   p_old_filename,
    wchar16_t const *   p_new_filename )
```

Arguments

Argument	Description	Type
p_old_filename	A pointer to the file or directory name, with or without its path.	wchar16_t *
p_new_filename	A pointer to the new name of the file or directory.	wchar16_t *

Return values

Return value	Description
EXFAT_NO_ERROR	Successful execution.
EXFAT_ERR_ALREADY_EXISTS	A file or directory with the new name already exists.
EXFAT_ERR_MEDIA_WRITE_PROTECTED	The media is write-protected.
Else	See Error Codes.

Example

```
void myfunc( void )
{
    .
    exfat_rename( HCC_UTF( "oldfile.txt" ), HCC_UTF( "newfile.txt" );
    exfat_rename( HCC_UTF( "A:/subdir/oldfile.txt" ), HCC_UTF( "newfile.txt" );
    .
    .
}
```

exfat_getattr

Use this function to get the attributes of a specified file.

Format

```
t_exfat_ret exfat_getattr (
    wchar16_t const * const   p_filename,
    t_exfat_attr * const      p_attr )
```

Arguments

Argument	Description	Type
p_filename	A pointer to the name of the file.	wchar16_t *
p_attr	Where to write the file attribute settings (EXFAT_ATTR_XXX).	t_exfat_attr *

Return values

Return value	Description
EXFAT_NO_ERROR	Successful execution.
EXFAT_ERR_INVALID_HANDLE	Invalid file handle.
Else	See Error Codes.

Example

```
void myfunc( void )
{
    t_exfat_attr attr;

    /* Find whether myfile.txt is read-only */
    if ( exfat_getattr( HCC_UTF( "myfile.txt" ), &attr ) == EXFAT_NO_ERROR )
    {
        if (attr & EXFAT_ATTR_READONLY)
        {
            printf( "myfile.txt is read-only\n" );
        }
        else
        {
            printf( "myfile.txt is writable\n" );
        }
    }
    else
    {
        printf( "File not found!\n" );
    }
}
```

exfat_setattr

Use this function to set the attributes of a file.

Note: The directory and volume attributes cannot be set by using this function.

Format

```
t_exfat_ret exfat_setattr (
    wchar16_t const * const   p_filename,
    t_exfat_attr const        attr )
```

Arguments

Argument	Description	Type
p_filename	A pointer to the name of the file.	wchar16_t *
attr	The new file attribute settings (EXFAT_ATTR_XXX).	t_exfat_attr

Return values

Return value	Description
EXFAT_NO_ERROR	Successful execution.
EXFAT_ERR_INVALID_HANDLE	Invalid file handle.
Else	See Error Codes.

Example

```
void myfunc( void )
{
    /* Make myfile.txt read-only and hidden */

    exfat_setattr( HCC_UTF("myfile.txt"), EXFAT_ATTR_READONLY | EXFAT_ATTR_HIDDEN );
}
```

exfat_gettimestamp

Use this function to get the timestamp of a file or directory.

The timestamp is automatically set by the system when a file or directory is created or modified, and when a file is closed.

Format

```
t_exfat_ret exfat_gettimestamp (
    wchar16_t const * const     p_filename,
    t_exfat_timestamp * const   p_timestamp )
```

Arguments

Argument	Description	Type
p_filename	A pointer to the name of the file or directory.	wchar16_t *
p_timestamp	Where to store the timestamp.	t_exfat_timestamp *

Return values

Return value	Description
EXFAT_NO_ERROR	Successful execution.
EXFAT_ERR_INVALID_PARAMETER	A parameter is invalid.

Example

```
void myfunc( void )
{
    t_exfat_timestamp buffer;

    if (exfat_gettimestamp( HCC_UTF( "subfolder" ), &buffer ) == EXFAT_NO_ERROR)
    {
        printf( "Creation date and time: %04i-%02i-%02i %2i:%02i:%02i\n"
               , buffer.create.year
               , buffer.create.month
               , buffer.create.day
               , buffer.create.hour
               , buffer.create.min
               , buffer.create.sec
              );
        printf( "Last modified: %04i-%02i-%02i %2i:%02i:%02i\n"
               , buffer.last_modified.year
               , buffer.last_modified.month
               , buffer.last_modified.day
               , buffer.last_modified.hour
               , buffer.last_modified.min
               , buffer.last_modified.sec
              );
        printf( "Last accessed: %04i-%02i-%02i %2i:%02i:%02i\n"
               , buffer.last_accessed.year
               , buffer.last_accessed.month
               , buffer.last_accessed.day
               , buffer.last_accessed.hour
               , buffer.last_accessed.min
               , buffer.last_accessed.sec
              );
    else
    {
        printf( "Timestamp cannot be retrieved!\n" );
    }
}
```

exfat_settimestamp

Use this function to set the timestamp of a file or directory.

Format

```
t_exfat_ret exfat_settimestamp (
    wchar16_t const * const           p_filename,
    t_exfat_timestamp const * const   p_timestamp )
```

Arguments

Argument	Description	Type
p_filename	A pointer to the name of the file or directory.	wchar16_t *
p_timestamp	A pointer to the timestamp to set.	t_exfat_timestamp *

Return values

Return value	Description
EXFAT_NO_ERROR	Successful execution.
EXFAT_ERR_INVALID_PARAMETER	A parameter is invalid.

Example

```
void myfunc( void )
{
    t_exfat_timestamp buffer;
    exfat_mkdir( HCC_UTF( "subfolder" ) ); /* Create directory */

    /* 2019, 4th of February */
    buffer.create.year = 2019;
    buffer.create.month = 2;
    buffer.create.day = 4;

    /* 10:14:33 */
    buffer.create.hour = 10;
    buffer.create.min = 14;
    buffer.create.sec = 33;

    /* Timezone +00:00 */
    buffer.zhour = 0;
    buffer.zmin = 0;
    buffer.last_modified = buffer.create;
    buffer.last_accessed = buffer.create;

    exfat_settimestamp( HCC_UTF( "subfolder" ), &buffer );
}
```

exfat_stat

Use this function to get information about a file or directory.

This function retrieves information by filling the t_exfat_stat structure passed to it. It sets the file/directory size, creation time/date, last access date, modified time/date, and the drive number where the file or directory is located.

Format

```
t_exfat_ret exfat_stat (
    wchar16_t const *       p_path,
    t_exfat_stat * const    p_stat )
```

Arguments

Argument	Description	Type
p_path	A pointer to the file or directory name.	wchar16_t *
p_stat	A pointer to the *t_exfat_stat* structure to fill.	t_exfat_stat *

Return values

Return value	Description
EXFAT_NO_ERROR	Successful execution.

Example

```
void myfunc( void )
{
    t_exfat_stat stat;
    if ( exfat_stat( HCC_UTF( "myfile.txt" ), &stat ) != EXFAT_NO_ERROR )
    {
        printf( "Error!\n" );
        return;
    }
    printf( "filesize:%lld\n", stat.filesize );
}
```

exfat_fstat

Use this function to get information about a file by using its file handle.

This function retrieves information by filling the t_exfat_stat structure passed to it.

Format

```
t_exfat_ret exfat_fstat (
    t_exfat_file_handle const   file_handle,
    t_exfat_stat * const        p_stat )
```

Arguments

Argument	Description	Type
file_handle	The file handle.	t_exfat_file_handle
p_stat	A pointer to the *t_exfat_stat* structure to fill.	t_exfat_stat *

Return values

Return value	Description
EXFAT_NO_ERROR	Successful execution.
EXFAT_ERR_INVALID_PARAMETER	A parameter is invalid.

Example

```
void myfunc ( void )
{
    t_exfat_file_handle file;
    t_exfat_stat stat;
    t_exfat_ret ret;

    ret = exfat_open( filename, "r", &file );
    if ( ret == EXFAT_NO_ERROR )
    {
        ret = exfat_fstat( file, &stat );
        if ( ret == EXFAT_NO_ERROR )
        {
            printf( "filesize:%lld\r\n", stat.filesize );
        }
        else
        {
            printf( "exfat_fstat error: %d.\r\n", ret );
        }
        exfat_close( file );
    }
    else
    {
        printf( "%ls Cannot open!\r\n", filename );
    }
}
```

exfat_filelength

Use this function to get the length of a file.

Format

```
t_exfat_ret exfat_filelength (
    wchar16_t const *       p_filename,
    t_exfat_size * const    p_filelength )
```

Arguments

Argument	Description	Type
p_filename	A pointer to the file name, with or without the path.	wchar16_t *
p_filelength	On return, a pointer to the size of the file in bytes.	t_exfat_size *

Return values

Return value	Description
EXFAT_NO_ERROR	Successful execution.
EXFAT_ERR_INVALID_PARAMETER	A parameter is invalid.

Example

```
int myreadfunc( wchar16_t *filename, char *buffer, long buffsize )
{
    t_exfat_file_handle file;
    t_exfat_ret ret;
    t_exfat_size length;
    uint32_t bytes_read;

    ret = exfat_open( filename, "r", &file );
    exfat_filelength( filename, &length );
    if ( ret != EXFAT_NO_ERROR )
    {
        printf( "%ls Cannot be opened!\n", filename );
        return 1;
    }

    if (length > buffsize)
    {
        printf( "Not enough memory!\n" );
        exfat_close( file );
        return 2;
    }

    exfat_read( buffer, length, file, &bytes_read );
    exfat_close( file );
    return 0;
}
```

exfat_is_file

Use this function to check that a file exists.

Format

```
t_exfat_ret exfat_is_file ( wchar16_t const * const p_filename )
```

Arguments

Argument	Description	Type
p_filename	A pointer to the file pathname.	wchar16_t *

Return values

Return value	Description
EXFAT_NO_ERROR	The file exists.
EXFAT_ERR_FILE_NOT_FOUND	The file was not found.

Example

```
void myfunc( void )
{
    t_exfat_ret result;
    result = exfat_is_file( HCC_UTF( "myfile" ));
    if ( result != EXFAT_NO_ERROR )
    {
        printf( "File not found!\n" );
    }
}
```

exfat_exists

Use this function to check that a file or directory exists.

Format

```
t_exfat_ret exfat_exists ( wchar16_t const * const p_path )
```

Arguments

Argument	Description	Type
p_path	A pointer to the file or directory pathname.	wchar16_t *

Return values

Return value	Description
EXFAT_NO_ERROR	The file or directory exists.
EXFAT_ERR_INVALID_PARAMETER	The path is invalid
EXFAT_ERR_FILE_NOT_FOUND	The file or directory was not found.

Example

```
void myfunc( void )
{
    t_exfat_ret ret;

    ret = exfat_exists( HCC_UTF( "subfolder" ));
    if ( ret != EXFAT_NO_ERROR )
    {
        printf( "Folder does not exist!\n" );
    }
    else
    {
        printf( "Folder exists\n" );
    }
}
```

6.3. Error Codes

The table below lists all the error codes that may be generated by API calls to HCC's file systems. Please note that some error codes are not used by every file system.

The header file to include for this list is: **/src/api/api_fs_err.h**

Error	Value	Meaning
EXFAT_NO_ERROR	0	Successful execution.
EXFAT_ERR_NOT_FORMATTED	2	The exFAT boot sector is invalid.
EXFAT_ERR_MEDIA_DRIVER	3	Media driver error.
EXFAT_ERR_INITFUNC	4	No init function is available for a driver, or the function generates an error.
EXFAT_ERR_BUSY	5	The caller could not obtain the semaphore within the expiry time.
EXFAT_ERR_END_OF_FILE	6	End of FAT chain was reached.
EXFAT_ERR_INVALID_VOLUME	7	The volume specified is invalid.
EXFAT_ERR_MEDIA_NOT_READY	8	The media is not ready for use.
EXFAT_ERR_NOT_ENOUGH_RESOURCE	9	Insufficient resources are available for the operation.
EXFAT_ERR_TOO_LARGE_SECTOR_SIZE	10	Increase configuration option EXFAT_MAX_SECTOR_SIZE.
EXFAT_ERR_INVALID_CLUSTER	11	The cluster is invalid.
EXFAT_ERR_INVALID_SECTOR	12	The sector is invalid.
EXFAT_ERR_NO_MORE_DIR_ENTRY	13	No more directory entries in the current directory.
EXFAT_ERR_INVALID_FORMAT	14	The format is invalid.
EXFAT_ERR_INVALID_PARAMETER	15	A parameter is invalid.
EXFAT_ERR_CHECKSUM	16	Checksum error.
EXFAT_ERR_TIMESTAMP_CONVERSION	17	Timestamp conversion failed.
EXFAT_ERR_TIMEDATE_CONVERSION	18	Time and date conversion failed.
EXFAT_ERR_IS_DIRECTORY	19	The "file" provided is actually a directory.
EXFAT_ERR_IS_FILE	20	The "directory" provided is actually a file.
EXFAT_ERR_FILE_NOT_FOUND	21	The file was not found.
EXFAT_ERR_INVALID_HANDLE	22	The handle is invalid.
EXFAT_ERR_SEEK	23	Invalid parameters for **exfat_seek()**.

Error	Value	Meaning
EXFAT_ERR_NO_MORE_SPACE	24	No more space is available on the media.
EXFAT_ERR_INVALID_MODE	25	The mode is invalid.
EXFAT_ERR_NO_ALLOCATION_BITMAP	26	The allocation bitmap does not exist.
EXFAT_ERR_NO_UPCASE_TABLE	27	The Upcase table does not exist.
EXFAT_ERR_MEDIA_WRITE_PROTECTED	28	The physical medium is write-protected.
EXFAT_ERR_TASK_NOT_FOUND	29	**exfat_enter_task()** was not called in the actual context.
EXFAT_ERR_ACCESS_DENIED	30	Tried to open read-only file for writing.
EXFAT_ERR_UPCASE_CONVERSION	31	Not for read.
EXFAT_ERR_ALREADY_EXISTS	32	The file or directory already exists.
EXFAT_ERR_INVALID_NAME	33	The path or filename contains invalid characters.
EXFAT_ERR_MEDIA_CHANGED	34	Media was removed and inserted; the volume must be initialized again.
EXFAT_ERR_DIRECTORY_TOO_DEEP	35	Directory depth is greater than configuration option EXFAT_MAX_DIR_DEPTH.
EXFAT_ERR_INVALID_LABEL_LENGTH	36	The label length is not valid.
EXFAT_ERR_DIRECTORY_NOT_EMPTY	37	Only an empty directory can be removed.
EXFAT_ERR_FILENAME_OR_PATH_TOO_LONG	38	Length of filename exceeds EXFAT_MAX_FILE_NAME_LENGTH or length of path exceeds EXFAT_MAX_PATH_LENGTH.
EXFAT_ERR_ALREADY_OPENED	39	The file is already open for writing.
EXFAT_ERR_PATH_NOT_FOUND	40	The path given was not found.
EXFAT_ERR_CANNOT_MOVE	41	The new path is on a different drive.
EXFAT_ERR_MEDIA_TOO_SMALL	42	At least 1 MiB (mebibyte) media is required for an exFAT file system.
EXFAT_ERR_ALREADY_INITIALIZED	43	The media is already initialized.
EXFAT_ERR_INVALID_PATH	44	The path contains invalid characters.
EXFAT_ERR_INVALID_CACHE_CFG	45	The cache configuration is invalid.
EXFAT_ERR_PARTITION_TABLE	46	The partition table is corrupt.
EXFAT_ERR_NO_PARTITION_TABLE	47	The partition table does not exist.
EXFAT_ERR_DIRECTORY_TOO_LARGE	48	The maximum size of a directory is 256 MB.
EXFAT_ERR_REPAIR_NEEDED	49	The media needs to be repaired using **exfat_repair()**. (Only applies in SafeexFAT.)

6.4. Types and Definitions

This section describes the main elements that are defined in the API Header file.

t_exfat_file_handle

The file handle, used as a reference for accessing files.

The handle is obtained when a file is opened and released when it is closed.

Name Lengths

The following file, path, and volume lengths are defined in the file **api_exfat.h**:

Element	Value	Description
EXFAT_FILE_NAME_LENGTH	(EXFAT_MAX_FILE_NAME_LENGTH + 1)	+1 is for trailing zero character.
EXFAT_PATH_LENGTH	(EXFAT_MAX_PATH_LENGTH + 1)	+1 is for trailing zero character.
EXFAT_MAX_VOLUME_LABEL_LENGTH	11	11 characters of type *wchar16_t*.
EXFAT_VOLUME_LABEL_LENGTH	(EXFAT_MAX_VOLUME_LABEL_LENGTH + 1)	+1 is for trailing zero character.

t_exfat_space

The *t_exfat_space* structure takes this form:

Element	Type	Description
total	t_exfat_size	The total size of the disk in bytes.
free_	t_exfat_size	The number of free bytes on the disk.
used	t_exfat_size	The number of used bytes on the disk.
bad	t_exfat_size	The number of bad bytes on the disk. These are not used.

t_exfat_stat

The *t_exfat_stat* structure takes this form:

Element	Type	Description
filesize	t_exfat_size	The size of the file.
timestamp	t_exfat_timestamp	The creation date and time.
attr	t_exfat_attr	The file's attributes.
drivenum	t_exfat_drive	The volume's index.

File and Directory Attributes

Directory entries, meta-description elements for files and directories, can have attributes assigned to them. These are backward-compatible with FAT12/16/32.

These are detailed in the table below.

Attribute	Value	Description
EXFAT_ATTR_ARC	0x20	An archived file or directory.
EXFAT_ATTR_DIR	0x10	A directory.
EXFAT_ATTR_SYSTEM	0x04	A system file or directory.
EXFAT_ATTR_HIDDEN	0x02	A hidden file or directory.
EXFAT_ATTR_READONLY	0x01	A read-only file or directory.

t_exfat_dir_entry

The *t_exfat_dir_entry* structure defines a directory:

Element	Type	Description
filename [EXFAT_FILE_NAME_LENGTH]	wchar16_t	The file or directory name.
filesize	t_exfat_size	The length of the file or directory.
timestamp	t_exfat_timestamp	The date and time of creation, modification, and last access.
attr	t_exfat_attr	The attributes of the file or directory.

t_exfat_timestamp

The *t_exfat_timestamp* structure takes this form:

Element	Type	Description
create	t_psp_timedate	The creation date and time.
last_modified	t_psp_timedate	The date and time when the file was last modified.
last_accessed	t_psp_timedate	The date and time when the file was last accessed.

t_exfat_volume_info

The *t_exfat_volume_info* structure takes this form:

Element	Type	Description
sector_size_byte	uint32_t	The sector size in bytes.
cluster_size_byte	uint32_t	The cluster size in bytes.
cluster_count;	uint32_t	The number of clusters.
exfat_boot_sector_idx	uint64_t	The boot sector index.

t_exfat_format_param

The *t_exfat_format_param* structure holds the parameters for formatting media. It takes this form:

Element	Type	Description
media_size_mb	uint32_t	The media size in MB.
sectors_per_cluster	uint32_t	The number of sectors in a cluster.
boundary_unit	uint32_t	The sector size unit.

Cache Definitions

The cache definitions are as follows:

Attribute	Value	Description
EXFAT_CACHE_TYPE_FAT	0	File Allocation Table.
EXFAT_CACHE_TYPE_ALLOCATION_BITMAP	1	Allocation bitmap.
EXFAT_CACHE_TYPE_UPCASE_TABLE	2	Upcase table.
EXFAT_CACHE_TYPE_DIRECTORY	3	A directory entry.
EXFAT_CACHE_TYPE_FILE_CONTENT	4	A file's content.
EXFAT_CACHE_TYPE_BOOT	5	Boot sector and so on.
EXFAT_CACHE_TYPE_COUNT	6	Not a real type, just used to count types.

t_exfat_cache_config

The *t_exfat_cache_config* structure takes this form:

Element	Type	Description
ech_cache_type	uint8_t	The type of cache: directory entry, allocation bitmap, and so on.
ech_sector_count	uint16_t	The count of sectors for each cache buffer. This must be at least 1. It also defines the read-ahead in the sector count.
ech_buffer_count	uint16_t	The count of buffers to allocate.

7. Integration

This section describes all aspects of the file system that require integration with your target project.

This includes porting and configuration of external resources.

7.1. OS Abstraction Layer

The module uses the OS Abstraction Layer (OAL) that allows it to run seamlessly with a wide variety of RTOSes, or without an RTOS.

The file system uses the following OAL components:

OAL Resource	Number required if FN_MAXTASK is 1	Number required if FN_MAXTASK > 1
Tasks	0	0
Mutexes	1	1 + EXFAT_MAX_VOLUME_COUNT
Events	0	0

Configuring the OAL

Configure the OAL as follows:

1. In **config_oal.h** keep the defaults of 1 for OAL_TASK_GET_ID_SUPPORTED and OAL_MUTEX_SUPPORTED.
2. In **config_oal_os.h** set the OAL_MUTEX_COUNT in **config_oal_os.h** to EXFAT_MAX_VOLUME_COUNT + 1.

Multiple Tasks, Mutexes and Reentrancy

Note: If your system has multiple tasks that access the file system, you must implement this section.

Each volume should be protected by a mutex mechanism to ensure that file access is safe.

If reentrancy is required, the following functions from the OAL are used:

- **oal_mutex_create()** – called on volume initialization/deletion and also on file system initialization/deletion.
- **oal_mutex_delete()** – called on volume initialization/deletion and also on file system initialization/deletion.
- **oal_mutex_get()** – called when a mutex is required.
- **oal_mutex_put()** – called when the mutex is released.

Within the standard API there is no support for the current working directory (**cwd**) to be maintained on a per-caller basis. By default the system provides a single **cwd** that can be changed by any user. The **cwd** is maintained on a per-volume basis, or on a per-task basis if reentrancy is implemented.

For a multitasking system, you must do the following:

1. Set EXFAT_MAX_TASK_COUNT to the maximum number of tasks that can simultaneously maintain access to the file system. This effectively creates a table of **cwd**s for each task.
2. Modify the function **oal_task_get_id()** to get a unique identifier for the calling task.
3. Ensure that any task using the file system calls **exfat_enter_task()** before using any other API calls; this ensures that the calling task is registered and the current working directory can be maintained for it.
4. Ensure that any application using the file system calls **exfat_exit_task()** with its unique identifier to free that table entry for use by other applications.

Once this is done, each caller is logged as it acquires the mutex, and a current working directory is associated with it. The caller must release this when it has finished using the file system; that is, when the calling task is terminated. This frees the entry for use by other tasks.

7.2. PSP Porting

The Platform Support Package (PSP) is designed to hold all platform-specific functionality, either because it relies on specific features of a target system, or because this provides the most efficient or flexible solution for the developer.

The module makes use of the following standard PSP functions:

Function	Package	Element	Description
psp_getcurrenttimedate()	psp_base	psp_rtc	Returns the current time and date. This is used for date and time-stamping files.
psp_getrand()	psp_base	psp_rand	Generates a random number. This is used for the volume serial number.
psp_memcpy()	psp_base	psp_string	Copies a block of memory. The result is a binary copy of the data.
psp_memset()	psp_base	psp_string	Sets the specified area of memory to the defined value.
psp_printf()	psp_base	psp_stdio	Prints a string.
psp_w16csncat()	psp_base	psp_string	Concatenates a source string to the end of a destination string.
psp_w16csnchr()	psp_base	psp_string	Counts characters in a UTF-16 string buffer.
psp_w16csncmp()	psp_base	psp_string	Compares two strings, returning 0 when they match, otherwise -1 or 1.
psp_w16csncpy()	psp_base	psp_string	Copies a source string to the destination string, overwriting existing content.
psp_w16csnlen()	psp_base	psp_string	Returns the length of a UTF-16 string.

The module makes use of the following standard PSP macros:

Macro	Package	Element	Description
PSP_RD_LE16	psp_base	psp_endianness	Reads a 16 bit value stored as little-endian from a memory location.
PSP_RD_LE32	psp_base	psp_endianness	Reads a 32 bit value stored as little-endian from a memory location.
PSP_RD_LE64	psp_base	psp_endianness	Reads a 64 bit value stored as little-endian from a memory location.
PSP_WR_LE16	psp_base	psp_endianness	Writes a 16 bit value to be stored as little-endian to a memory location.
PSP_WR_LE32	psp_base	psp_endianness	Writes a 32 bit value to be stored as little-endian to a memory location.
PSP_WR_LE64	psp_base	psp_endianness	Writes a 64 bit value to be stored as little-endian to a memory location.

Unicode string literals

The HCC_UTF macro is used to create UTF-16 string literals. This macro is used in the code examples.

- ISO C99 or older compilers generate a UTF-16 string when the capital letter 'L' prefix is used, for example `L"myfile.bin"`.
- ISO C11 or newer compilers generate a 16-bit Unicode string when the lower case letter 'u' is used, example `u"myfile.bin"`.

Get Time and Date

For compatibility with other systems, you must provide a real-time function so that files can be time-stamped and date-stamped.

A pseudo time/date function, **psp_getcurrenttimedate()**, is provided in **target/rtc/psp_rtc.c.** Modify this to provide the time in standard format from a Real-Time Clock source (RTC).

Random Number

The **target/psp_rand.c** file contains a function **psp_getrand()** that the file system uses to obtain a pseudo-random number to use as the volume serial number. This function is required only if a hard-format of a device is required.

It is recommended that you replace this routine with a random function from the base system, or alternatively generate a random number based on a combination of the system time and date and a system constant such as a MAC address.

8. Version

Version 1.60

For use with exFAT and SafeexFAT File Systems versions 1.17 and above