# Table of Contents

1 System Overview ........................................................................................................ 4  
   1.1 Introduction ........................................................................................................... 5  
      Standard File System Functions ................................................................. 6  
      Fail-safe functions ...................................................................................... 7  
   1.2 How Testing Works .............................................................................................. 8  
      Testing the Standard Functions ...................................................................... 8  
      Testing SafeexFAT Functions ......................................................................... 8  
      The Unit Test Driver ..................................................................................... 8  
      How the Unit Test Driver Works ................................................................. 8  
   1.3 Feature Check ........................................................................................................ 10  
   1.4 Packages and Documents .................................................................................. 11  
      Packages ........................................................................................................... 11  
      Documents ...................................................................................................... 11  
   1.5 Change History ................................................................................................. 12  

2 Source File List ........................................................................................................ 13  
   2.1 API Header File .................................................................................................. 13  
   2.2 Configuration File .............................................................................................. 13  
   2.3 Source Code ....................................................................................................... 13  
   2.4 Version File ......................................................................................................... 14  
   2.5 Platform Support Package (PSP) Files ............................................................. 14  

3 Configuration Options .............................................................................................. 15  

4 Running Tests .......................................................................................................... 25  
   4.1 exfat_do_test ..................................................................................................... 25  
   4.2 exfat_do_test2 ................................................................................................. 26  
   4.3 exfat_test_readsector ..................................................................................... 27  
   4.4 exfat_test_readmultiplesector ................................................................ 28  
   4.5 Test Return Codes ........................................................................................... 29  

5 Integration ................................................................................................................ 31  
   5.1 OS Abstraction Layer ....................................................................................... 31  
   5.2 PSP Porting ....................................................................................................... 32  
      Unicode string literals ................................................................................. 33
t_exfat_test_cb .................................................................34
psp_exfat_test_set_error ..................................................35
psp_exfat_test_clear_error ...............................................36
psp_exfat_test_reset_driver ..............................................37
psp_exfat_test_set_write_error_counter ..........................38
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.
- **How Testing Works** – describes the role of the unit test driver.
- **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 want to test an HCC Embedded exFAT/SafeexFAT file system. SafeexFat was designed with integration and target verification in mind and you can use this comprehensive test suite to prove that the product behaves correctly when integrated with your product.

This diagram shows the system architecture.

The **Fail-safety and Unit Test Driver** is used with the target MCU to test that the file system is fail-safe. This driver is used to inject errors at the media driver level. Once testing has shown that the file system is fail-safe, this module can be removed before the file system goes live.
This test suite allows you to test two types of function, described below.

**Standard File System Functions**

The standard file system functions include the following:

- File and directory creation and deletion.
- File reading and writing.
- Checking of file content.
- Handling of "a", "a+", "w", and "w+" files.
- Concurrent access with "r" files.
- Volume formatting.
- Recovery from power failure.
- Finding and seeking.
- File listing.
- File and directory renaming and moving.
- File and directory attributes.
- File positions.
- File truncation.
- File ranges.
- Files with no FAT chain.
- Timestamps.
- Appending to files.
- Write-protection.
- Upcase tables.
- Error handling.
- SafeexFAT repair function.
Fail-safe functions

Safety is achieved by using internal logs, which are also written to the media. These log files contain all the information required to recover the last good state of the file system after reset or power failure occurs.

SafeexFAT has been implemented for the following functions and all of these can be tested as shown below.

<table>
<thead>
<tr>
<th>Function</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>exfat_open()</td>
<td>In modes a, a+, w, w+, and r+.</td>
</tr>
<tr>
<td>exfat_write()</td>
<td>In mode &quot;r+&quot; exfat_write() copies the whole cluster when overwriting existing data. This way existing data is kept until exfat_close()/exfat_flush() is called. This copy can be optimized to reduce the amount of copied data.</td>
</tr>
<tr>
<td>exfat_seek()</td>
<td>When seeking beyond file size in a file opened in all the modes above.</td>
</tr>
<tr>
<td>exfat_flush()</td>
<td>Tested in a file opened in all the modes above.</td>
</tr>
<tr>
<td>exfat_close()</td>
<td>Tested in a file opened in all the modes above.</td>
</tr>
<tr>
<td>exfat_ftruncate()</td>
<td>Tested in a file opened in all the modes above.</td>
</tr>
<tr>
<td>exfat_mkdir()</td>
<td></td>
</tr>
<tr>
<td>exfat_rmdir()</td>
<td></td>
</tr>
<tr>
<td>exfat_remove()</td>
<td></td>
</tr>
<tr>
<td>exfat_remove_content()</td>
<td>If this call is interrupted, the file is removed, but some clusters may not be cleared.</td>
</tr>
<tr>
<td>exfat_rename()</td>
<td></td>
</tr>
<tr>
<td>exfat_move()</td>
<td></td>
</tr>
</tbody>
</table>
1.2 How Testing Works

Testing the Standard Functions

The standard test suite of functions exercises the file system using the target media driver and media. This is the setup shown in the left-hand side of this diagram:

Testing SafeexFAT Functions

This is the setup shown in the right-hand side of the above diagram.

The Unit Test Driver

To test the fail-safe features of the system, the unit test driver is installed between the real media driver and the SafeexFAT file system. It is used to inject faults into the media operations for writing and reading. Normally write requests are forwarded to the real media driver but, to simulate a power failure, a reset is issued by the test media driver at a specific moment.

In short, the test driver helps simulate error conditions so that the correct behavior can be checked. The same test media driver is used for Windows and Linux systems.

How the Unit Test Driver Works

To test the safe functions listed earlier, a simulated reset is implemented. The file system uses the unit test driver to write and read from the media.

A counter in the unit test driver counts sector writes. After a pre-defined sector write count is reached, the test driver jumps from the sector write function to a previously specified reset point and leaves the media in
an inconsistent state. This reset can happen at any time: during log file writing, log file extending, writing the user’s file content, updating directory entries, etc.

After a reset the test function:

1. Detects the reset event, initializes the file system using `exfat_init()`, and initializes the volume using `exfat_initvolume()`, as though this was a power-up.
2. It repairs the file system and checks that files are in the expected condition: exists, does not exist, having the correct FAT chain or content.

The same test is repeated with increasing write counts until the simulated software reset does not occur. If the test finally runs without resetting, it means that all possible write errors have been tried.

Note that:

- The test runs longer when the cluster size is larger, because more sector writes are needed. Therefore more repeats are necessary.
- Resources are deliberately not released during reset, therefore these tests need many more mutexes than the other ones.
- To run the SafeexFAT tests, Visual Studio and the Windows 32 or Linux OS Abstraction Layer (OAL) are needed.
1.3 Feature Check

The main features of the module are the following:

- Provides a full test capability for HCC Embedded's exFAT and SafeexFAT file systems.
- Conforms to the HCC Advanced Embedded Framework.
- Designed for integration with both RTOS and non-RTOS based systems.
- Conforms to the HCC Coding Standard.
1.4 Packages and Documents

Packages

This table lists the packages that need to be used with this module:

<table>
<thead>
<tr>
<th>Package</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hcc_base_doc</td>
<td>This contains the two guides that will help you get started.</td>
</tr>
<tr>
<td>fs_exfat_test</td>
<td>The exFAT and SafeexFAT test suite package.</td>
</tr>
<tr>
<td>psp_template_base</td>
<td>The Platform Support Package (PSP) base package.</td>
</tr>
<tr>
<td>media_drv_test</td>
<td>This is needed to run media failure tests.</td>
</tr>
</tbody>
</table>

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 document describes the main packages.

HCC exFAT and SafeexFAT Test Suite User Guide

This is this document.
1.5 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_test.

The current version of this manual is 1.20.

<table>
<thead>
<tr>
<th>Manual version</th>
<th>Date</th>
<th>Software version</th>
<th>Reason for change</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.20</td>
<td>2019-10-02</td>
<td>1.12</td>
<td>Added configuration options EXFAT_TEST_SAFE_LOG_WRITE_NEG, EXFAT_TEST_SAFE_LOG_FTRUNCATE_NEG and EXFAT_TEST_SAFE_LOG_FTRUNC_BELOW_ACTUALFSIZE</td>
</tr>
<tr>
<td>1.10</td>
<td>2019-09-16</td>
<td>1.11</td>
<td>Added many configuration options. Added functions to API: exfat_do_test2(), exfat_test_readsector(), and exfat_test_readmultiplesector(). Added EXFAT_TEST_ERR_INVALID_DRIVE_INDEX to Return Codes.</td>
</tr>
<tr>
<td>1.00</td>
<td>2019-08-29</td>
<td>1.09</td>
<td>First version.</td>
</tr>
</tbody>
</table>
2 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 file.

2.1 API Header File

The file src/api/api_exfat_test.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 Running Tests. This file should only be modified by HCC.

2.2 Configuration File

The file src/config/config_exfat_test.h contains all the configurable parameters of the system. Configure these as required. For details of these options, see Configuration Options.

2.3 Source Code

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

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>exfat_test.c and .h</td>
<td>Basic test functions.</td>
</tr>
<tr>
<td>exfat_test_dev.c and .h</td>
<td>Development test functions.</td>
</tr>
<tr>
<td>exfat_test_dev_safe.c and .h</td>
<td>SafeexFAT development test functions.</td>
</tr>
<tr>
<td>exfat_test_dev_safe_log.c and .h</td>
<td>SafeexFAT log test functions.</td>
</tr>
<tr>
<td>exfat_test_dir.c and .h</td>
<td>Directory test functions.</td>
</tr>
<tr>
<td>exfat_test_file.c and .h</td>
<td>File test functions.</td>
</tr>
<tr>
<td>exfat_test_media.c and .h</td>
<td>Media driver test functions.</td>
</tr>
<tr>
<td>exfat_test_task.c and .h</td>
<td>Task test functions.</td>
</tr>
<tr>
<td>exfat_test_upcase.c and .h</td>
<td>Upcase table test functions.</td>
</tr>
</tbody>
</table>
2.4 Version File

The file src/version/ver_exfat_test.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.

2.5 Platform Support Package (PSP) Files

These files provide functions and elements the core code may need to use, depending on the hardware. Modify these files as required for your hardware.

**Note:**
- These are PSP implementations for the specific microcontroller and board; you may need to modify these to work with a different microcontroller and/or development board; see PSP Porting for details.
- In the package these files are offset to avoid overwriting an existing implementation. Copy them to the root hcc directory for use.

The following files are in the directory src/psp/target/exfat:

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>psp_exfat_test.c</td>
<td>PSP source code.</td>
</tr>
<tr>
<td>psp_exfat_test.h</td>
<td>PSP header file.</td>
</tr>
</tbody>
</table>

The PSP also has a version file, ver_psp_exfat_test.h.
3 Configuration Options

Set the following exFAT Test Suite configuration options in the file `config_exfat_test.h`. To disable a test whose default is 1, just set its option to 0.

**EXFAT_TEST_VERBOSE**

Set this to 1 for fully detailed test output. The default is 0.

**EXFAT_TEST_PRINTF_CAPITAL_S_SUPPORTED**

There are two options:

- 0: only `psp_printf("%s", ascii_str)` is supported (ASCII).
- 1: `psp_printf("%S", utf32_str)` UTF-32 and ASCII are supported. This is the default.

**EXFAT_TEST_PRINTF_LLI_SUPPORTED**

There are two options:

- 0: only `psp_printf("%i", int32_t)` is supported.
- 1: `psp_printf("%lli", int64_t)` is supported as well as the above `int32_t` option. This is the default.

**EXFAT_TEST_ALWAYS_FORMAT**

There are two options:

- 0: only format the media if it is not formatted, or the file system is not exFAT. This is the default.
- 1: always format the media at the start of a test. **WARNING: all data will be lost!**

**EXFAT_TEST_VOLUME_LABEL**

Keep the default of 1 to test `exfat_setlabel()` and `exfat_getlabel()`.

**EXFAT_TEST_CREATE_TEST_FILES**

Keep the default of 1 to create the test files that the following two options need.

**EXFAT_TEST_CHDIR_GETCWD**

Keep the default of 1 to test `exfat_chdir()` with absolute and relative paths.

**EXFAT_TEST_READ_FILES**

Keep the default of 1 to test reading of files.
**EXFAT_TEST_WRITE_FILES**
Keep the default of 1 to enable the file writing (mode "w") test.

**EXFAT_TEST_DIR**
Keep the default of 1 to test `exfat_mkdir()` and `exfat_rmdir()`.

**EXFAT_TEST_REMOVE**
Keep the default of 1 to test `exfat_remove()`.

**Note:** The following option only applies if `EXFAT_ENABLE_REMOVE_CONTENT` (in the main file system package) is set.

**EXFAT_TEST_REMOVE_CONTENT**
Keep the default of 1 to test `exfat_remove_content()`.

**EXFAT_TEST_TRUNCATE**
Keep the default of 1 to test `exfat_truncate()` and `exfat_ftruncate/exfat_seteof()`.

**EXFAT_TEST_APPEND**
Keep the default of 1 to enable the file append test (mode "a"/"a+”).

**EXFAT_TEST_ATTR**
Keep the default of 1 to test `exfat_setattr()` and `exfat_getattr()`.

**EXFAT_TEST_RANGE**
Keep the default of 1 to test file read, write, and seek with special range values.

**EXFAT_TEST_SEEK_BEYOND**
Keep the default of 1 to test `exfat_seek()` seeking beyond the file size.

**EXFAT_TEST_SEEK_BEYOND_NOWRITE**
Keep the default of 1 to test `exfat_seek()` seeking beyond the file size with NOWRITE.

**EXFAT_TEST_DIR_OPEN**
Keep the default of 1 to test directory opening.

**EXFAT_TEST_LONG_FILENAME**
Keep the default of 1 to test long filenames.
EXFAT_TEST_LONG_PATH
Keep the default of 1 to test long pathnames.

EXFAT_TEST_OPEN_MODE
Keep the default of 1 to enable the open mode test.

EXFAT_TEST_MULTI_FILE
Keep the default of 1 to enable the multiple file test.

EXFAT_TEST_INIT_VOLUME
Keep the default of 1 to test exfat_initvolume().

**Note:** The following two options only apply if EXFAT_MAX_TASK_COUNT (in the main file system package) > 1.

EXFAT_TEST_ENTER_TASK
Keep the default of 1 to enable the task entry test.

EXFAT_TEST_TASK
Keep the default of 1 to enable the task test.

EXFAT_TEST_INVALID_FILE_NAME
Keep the default of 1 to enable the test of invalid file names.

EXFAT_TEST_INVALID_DIR_NAME
Keep the default of 1 to enable the test of invalid directory names.

EXFAT_TEST_OPEN_NEG
Keep the default of 1 to test file handling functions with invalid arguments.

EXFAT_TEST_DIR_NEG
Keep the default of 1 to test directory handling functions with invalid arguments.

EXFAT_TEST_REWIND_DIR
Keep the default of 1 to test the directory directory rewind function.

EXFAT_TEST_DIR_DEPTH
Keep the default of 1 to test exfat_chdir() to the maximum directory depth.
EXFAT_TEST_WRITE_PROTECTED
Keep the default of 1 to test the write-protected state of media. The function `media_drv_test()` is needed to run this test.

EXFAT_TEST_MEDIA_ERROR
Keep the default of 1 to test media error of media. The function `media_drv_test()` is needed to run this test.

EXFAT_TEST_FORMAT
Keep the default of 1 to test the format when the media has zero or low sectors.

EXFAT_TEST_PUTC_GETC
Keep the default of 1 to test the `exfat_getc()` and `exfat_putchar()` functions.

EXFAT_TEST_DIR_ENTRY
Keep the default of 1 to test the creation and deletion of directory entries.

EXFAT_TEST_REWIND
Keep the default of 1 to test the rewind function.

EXFAT_TEST_MEDIA_STATUS
Keep the default of 1 to test the missing and changed media states (the `media_drv_test` package is needed to run this test).

EXFAT_TEST_MEDIA_FLUSH
Keep the default of 1 to test the IOCTL flush (the `media_drv_test` package is needed to run this test).

EXFAT_TEST_INVALID_VOLUME
Keep the default of 1 to test functions with a non-existing volume index.

EXFAT_TEST SEEK NEG
Keep the default of 1 to test the error handling of `exfat_seek()`.

EXFAT_TEST_BOOTSECTOR_ERRORS
Keep the default of 1 to test the boot sector (the `media_drv_test` package is needed to run this test).

EXFAT_TEST_BITMAP_DIR_ENTRY_ERROR
Keep the default of 1 to test the allocation bitmap's entry (the `media_drv_test` package is needed to run this test).
EXFAT_TEST_UPCASE_DIR_ENTRY_ERROR
Keep the default of 1 to test errors in the upcase table's entry (the media_drv_test package is needed to run this test).

EXFAT_TEST_FILE_DIR_ENTRY_ERROR
Keep the default of 1 to test errors in a file's entry (the media_drv_test package is needed to run this test).

EXFAT_TEST_PARTITION_TABLE_ERROR
Keep the default of 1 to test error of FAT. The media_drv_test package and a disk image with a partition table (MBR) are needed to run this test.

EXFAT_TEST_FAT_ERROR
Keep the default of 1 to test FAT errors (the media_drv_test package is needed to run this test).

EXFAT_TEST_LABEL_DIR_ENTRY_ERROR
Keep the default of 1 to test errors in a label's entry (the media_drv_test package is needed to run this test).

EXFAT_TEST_INVALID_FILE_CUR_POS
Keep the default of 1 to test handling of an invalid current position.

EXFAT_TEST_DIR_CLUSTER
Keep the default of 1 to test exfat_mkdir() when the directory's cluster is full.

EXFAT_TEST_DIRENTRY_UPDATE_ERROR
Keep the default of 1 to test handling of directory entry errors (the media_drv_test package is needed to run this test).

EXFAT_TEST_STREAM_EXT_DIR_ENTRY_ERROR
Keep the default of 1 to test errors in the stream ext. directory entry (the media_drv_test package is needed to run this test).

EXFAT_TEST_FILEDLAETH_ERROR_WITH_NOFATCHAIN
Keep the default of 1 to test a file's DataLength error when NoFatChain=1.

EXFAT_TEST_DIR_WITH_BAD_ENTRY_FILE
Keep the default of 1 to test a file with a bad directory entry. This needs a special file to be present on the media.

EXFAT_TEST_INIT_VOLUME_WITH_INV_CACHE
Keep the default of 1 to test exfat_initvolume() with an invalid cache configuration.
EXFAT_TEST_INIT_NEG
Keep the default of 1 to test mutex resource errors (needs OAL with fault injection).

EXFAT_TEST_STREAM_ENTRY_UPDATE_ERROR
Keep the default of 1 to test an erroneous stream ext. directory entry (media_drv_test is needed to run this test).

EXFAT_TEST_DIR_SIZE_NEG
Keep the default of 1 to test a directory full of files (needs directories and files on media).

EXFAT_TEST_FREEENTRY_AT_CLUSTERBEGINING
Keep the default of 1 to test a directory which has a free entry at the cluster boundary (needs directories and files on media).

EXFAT_TEST_DIRENTRY_DISABLE_ERROR
Keep the default of 1 to test error of stream ext. directory entry (media_drv_test is needed to run this test).

EXFAT_TEST_NOFATCHAIN
Set this to 1 to enable the No FAT Chain test. The /nofatchain/ directory contains prepared files that are used to run this test.

EXFAT_TEST_TIMESTAMP
Keep the default of 1 to test setting/getting timestamps. This tests exfat_gettimestamp() and exfat_settimestamp().

EXFAT_TEST_UPCASE_RND
Keep the default of 1 to test a randomly-generated UpCase table.

EXFAT_TEST_UPCASE
Keep the default of 1 to test non case-sensitive behavior.

EXFAT_TEST_ZERO_LENGTH
Set this to 1 to test opening a zero length file in "r+" mode. The default is 0.

EXFAT_TEST_REOPEN
Keep the default of 1 to test opening the same file several times simultaneously.

EXFAT_TEST_RENAME
Keep the default of 1 to test file and directory renaming. This tests exfat_rename().
EXFAT_TEST_MOVE

Keep the default of 1 to test file and directory moving. This tests `exfat_move()`.

**Note:**
- The following option only applies if both EXFAT_TEST_CREATE_TEST_FILES and EXFAT_TEST_READ_FILES are set.
- Setting EXFAT_TEST_WRITE_FULL can make the test run very slowly.

EXFAT_TEST_WRITE_FULL

Keep the default of 1 to test writing to full media.

EXFAT_TEST_LIST_FILES

This only applies if EXFAT_TEST_CREATE_TEST_FILES (see above) is set. Set this to 1 to test `exfat_opendir()`, `exfat_readdir()`, and `exfat_closedir()`. The default is 0.

**Note:**
- The following SAFE tests (options starting `EXFAT_TEST_SAFE_`) are only available if EXFAT_ENABLE_SAFE is set to 1.
- These tests need a lot of mutex resources for reset simulation.
- These can only be enabled when Visual Studio and Win32 OAL or Linux OAL are used.

EXFAT_TEST_SAFE_FILE_APPEND

Keep the default of 1 to test file handling of append mode ("a"). This writes to the media until it is full, so can be very slow.

EXFAT_TEST_SAFE_FILE_OVERWRITE

Keep the default of 1 to test file handling of overwrite mode ("r+").

EXFAT_TEST_SAFE_FILE_OVERWRITE_NO_WERR

Keep the default of 1 to test file handling of overwrite mode ("r+") without write error simulation.

EXFAT_TEST_SAFE_REPAIR

Keep the default of 1 to test `exfat_repair()` and the repair-needed state.

EXFAT_TEST_SAFE_DIR

Keep the default of 1 to test directory handling.
**EXFAT_TEST_SAFE_LOG_RANGE**
Keep the default of 1 to test log file writing/reading.

**EXFAT_TEST_SAFE_LOG_ERROR**
Keep the default of 1 to enable the write error test of log file writing.

**EXFAT_TEST_SAFE_LOG_ENTRY**
Keep the default of 1 to test log entry reading/writing.

**EXFAT_TEST_SAFE_REMOVE**
Keep the default of 1 to test fail-safe `exfat_remove()`.

**EXFAT_TEST_SAFE_MKDIR**
Keep the default of 1 to test fail-safe `exfat_mkdir()` and `exfat_rmdir()`.

**EXFAT_TEST_SAFE_RMDIR**
Keep the default of 1 to test fail-safe `exfat_rmdir()`.

**EXFAT_TEST_SAFE_RENAME**
Keep the default of 1 to test fail-safe `exfat_rename()`.

**EXFAT_TEST_SAFE_MOVE**
Keep the default of 1 to test fail-safe `exfat_move()`.

**EXFAT_TEST_SAFE_TRUNCATE**
Keep the default of 1 to test `exfat_truncate()` and `exfat_ftruncate/exfat_seteof()`.

**EXFAT_TEST_SAFE_TRUNCATE_EXT**
Keep the default of 1 to test safe `exfat_truncate()` when extending file size.

**EXFAT_TEST_SAFE_LOG_INIT_ERROR**
Keep the default of 1 to test log initialization.

**EXFAT_TEST_SAFE_DIR_FAT_CHAIN**
Keep the default of 1 to test `exfat_check_fat_chain`.

**EXFAT_TEST_MKDIR_WRITE_ERROR**
Keep the default of 1 to test mkdir with different write errors.
EXFAT_TEST_SAFE_SETLABEL
Keep the default of 1 to test safe exfat_setlabel().

EXFAT_TEST_SAFE_SETTIMESTAMP
Keep the default of 1 to test safe exfat_settimestamp().

EXFAT_TEST_SAFE_SETATTR
Keep the default of 1 to test safe exfat_setattr().

EXFAT_TEST_SAFE_LOG_INIT_WITH_WPROTECT
Keep the default of 1 to test $$SAFE$$ creation when the media is write-protected.

EXFAT_TEST_SAFE_LOG_OPEN_WITH_DIR
Keep the default of 1 to test log opening with a test directory existing in $$SAFE$$.

EXFAT_TEST_SAFE_LOG_ENTRY_PART_MISSING
Keep the default of 1 to test log read with log directory entry second part missing.

EXFAT_TEST_SAFE_LOG_SEEK_NEG
Keep the default of 1 to test log seek with error values.

EXFAT_TEST_SAFE_LOG_REWIND
Keep the default of 1 to test log rewinding.

EXFAT_TEST_SAFE_LOG.Seek_BEYOND
Keep the default of 1 to test log seek by seeking beyond its file size.

EXFAT_TEST_SAFE_LOG_REMOVE_NEG
Keep the default of 1 to test log removal with a directory entry error.

EXFAT_TEST_SAFE_LOG_WRITE_NEG
Keep the default of 1 to test log write with an invalid open mode.

EXFAT_TEST_SAFE_LOG_FTRUNCATE_NEG
Keep the default of 1 to test log ftruncate() with an invalid open mode.

EXFAT_TEST_SAFE_LOG_FTRUNC_BELOW_ACTUAL_FSIZE
Keep the default of 1 to test log ftruncate() with a setting smaller than the real file size.
**EXFAT_TEST_BUFFER_SIZE**

The size of the statically allocated buffer for reading and writing files. The default is \((2 \times 1024 \times 1024)\).

**EXFAT_TEST_FILE_COUNT**

The number of test files used. The default is 256 and the minimum is 16.

**EXFAT_TEST_WRITE_FILE_COUNT**

The number of test files written. This only applies if EXFAT_TEST_WRITE_FILES (see above) is set. The default is 256 and the minimum is 1.

**EXFAT_TEST_TASK1_STACK_SIZE**

The stack size for the test task. The default is \((2 \times 1024 \times 1024)\).
4 Running Tests

This section shows how to run a test by using the API function `exfat_do_test()`. It also explains the return codes that running the test suite may produce.

4.1 exfat_do_test

Use this function to run the full test suite on a volume.

Format

```c
_t_exfat_ret exfat_do_test ( 
    _t_exfat_drive drivenum, 
    F_DRIVERINIT driver_init, 
    uint32_t driver_param )
```

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>drivenum</td>
<td>_t_exfat_drive</td>
<td>The ID of the volume to test.</td>
</tr>
<tr>
<td>driver_init</td>
<td>F_DRIVERINIT</td>
<td>The media driver's <code>F_DRIVERINIT()</code> function.</td>
</tr>
<tr>
<td>driver_param</td>
<td>uint32_t</td>
<td>This can optionally be used to pass information to the low level driver. Its use is driver-dependent. When the <code>xxx_initfunc()</code> 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.</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Return value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXFAT_NO_ERROR</td>
<td>Successful execution; all tests worked as expected.</td>
</tr>
<tr>
<td>Else</td>
<td>See Test Return Codes.</td>
</tr>
</tbody>
</table>
4.2 exfat_do_test2

Use this function to run the full test suite on two volumes.

Format

```c
void exfat_do_test ( 
    t_exfat_drive drivenum, 
    F_DRIVERINIT driver_init, 
    uint32_t driver_param, 
    t_exfat_drive drivenum2, 
    F_DRIVERINIT driver_init2, 
    uint32_t driver_param2 )
```

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>drivenum</td>
<td>t_exfat_drive</td>
<td>The ID of the volume to test.</td>
</tr>
<tr>
<td>driver_init</td>
<td>F_DRIVERINIT</td>
<td>The media driver's <code>F_DRIVERINIT()</code> function, used to allocate and initialize a new driver.</td>
</tr>
<tr>
<td>driver_param</td>
<td>uint32_t</td>
<td>This can optionally be used to pass information to the low level driver. Its use is driver-dependent. When the <code>xxx_initfunc()</code> 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.</td>
</tr>
<tr>
<td>drivenum2</td>
<td>t_exfat_drive</td>
<td>The ID of the second volume to test.</td>
</tr>
<tr>
<td>driver_init2</td>
<td>F_DRIVERINIT</td>
<td>The media driver's <code>F_DRIVERINIT()</code> function, used to allocate and initialize a second driver.</td>
</tr>
<tr>
<td>driver_param2</td>
<td>uint32_t</td>
<td>Use this is to specify which device associated with the second driver to initialize.</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Return value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXFAT_NO_ERROR</td>
<td>Successful execution; all tests worked as expected.</td>
</tr>
<tr>
<td>Else</td>
<td>See Test Return Codes.</td>
</tr>
</tbody>
</table>
4.3 exfat_test_readsector

This function alters the content of `p_buffer` according to fault injection.

**Note:** Do not call this function. It is called by `media_drv_test`.
Registered this function by using `drvtest_register_read_sector_cb()`.

**Format**

```c
int exfat_test_readsector ( 
    uint8_t         drvidx, 
    uint8_t *       p_buffer, 
    unsigned long   sector )
```

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>drvidx</td>
<td>uint8_t</td>
<td>The drive's index.</td>
</tr>
<tr>
<td>p_buffer</td>
<td>uint8_t *</td>
<td>The buffer to alter.</td>
</tr>
<tr>
<td>sector</td>
<td>unsigned long</td>
<td>The sector's index.</td>
</tr>
</tbody>
</table>

**Return Values**

<table>
<thead>
<tr>
<th>Return value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MDRIVER_TEST_NO_ERROR</td>
<td>Successful execution; the function changed the buffer's content.</td>
</tr>
<tr>
<td>MDRIVER_TEST_ERROR</td>
<td>The function did not change the buffer's content.</td>
</tr>
</tbody>
</table>
4.4 `exfat_test_readmultiplesector`

This function alters the content of `p_buffer` according to fault injection.

**Note:** Do not call this function. It is called by `media_drvm_test`.

Registered this function by using `drvtest_register_read_multiple_sector_cb()`.

**Format**

```c
int exfat_test_readmultiplesector (  
    uint8_t drvidx,  
    uint8_t * p_buffer,  
    unsigned long sector,  
    int cnt )
```

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>drvidx</td>
<td>uint8_t</td>
<td>The drive's index.</td>
</tr>
<tr>
<td>p_buffer</td>
<td>uint8_t *</td>
<td>The buffer to alter.</td>
</tr>
<tr>
<td>sector</td>
<td>unsigned long</td>
<td>The sector's index.</td>
</tr>
<tr>
<td>cnt</td>
<td>int</td>
<td>The number of sectors in <code>p_buffer</code>.</td>
</tr>
</tbody>
</table>

**Return Values**

<table>
<thead>
<tr>
<th>Return value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MDRIVER_TEST_NO_ERROR</td>
<td>Successful execution; the function changed the buffer's content.</td>
</tr>
<tr>
<td>MDRIVER_TEST_ERROR</td>
<td>The function did not change the buffer's content.</td>
</tr>
</tbody>
</table>
# 4.5 Test Return Codes

The table below lists all the error codes that may be generated by running the test suite.

Errors are reported with the relevant file name and the line number in that file.

<table>
<thead>
<tr>
<th>Error</th>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXFAT_TEST_ERR_NEGATIVE</td>
<td>1000</td>
<td>EXFAT_ERR_NO_ERROR was returned, but it is a faulty error code.</td>
</tr>
<tr>
<td>EXFAT_TEST_ERR_WRONG_OFFSET</td>
<td>1001</td>
<td>The offset is incorrect.</td>
</tr>
<tr>
<td>EXFAT_TEST_ERR_WRONG_STATUS</td>
<td>1002</td>
<td>The file status is incorrect.</td>
</tr>
<tr>
<td>EXFAT_TEST_ERR_INTERNAL_ERROR</td>
<td>1003</td>
<td>Internal error.</td>
</tr>
<tr>
<td>EXFAT_TEST_ERR_READ_ERROR</td>
<td>1004</td>
<td>Read error.</td>
</tr>
<tr>
<td>EXFAT_TEST_ERR_WRITE_ERROR</td>
<td>1005</td>
<td>Write error.</td>
</tr>
<tr>
<td>EXFAT_TEST_ERR_FILE_CONTENT_MISMATCH</td>
<td>1006</td>
<td>File content is incorrect.</td>
</tr>
<tr>
<td>EXFAT_TEST_ERR_FILE_SIZE_MISMATCH</td>
<td>1007</td>
<td>File size is incorrect.</td>
</tr>
<tr>
<td>EXFAT_TEST_ERR_CHECKSUM_MISMATCH</td>
<td>1008</td>
<td>Checksum is incorrect.</td>
</tr>
<tr>
<td>EXFAT_TEST_ERR_CWD_MISMATCH</td>
<td>1009</td>
<td>Current working directory is incorrect.</td>
</tr>
<tr>
<td>EXFAT_TEST_ERR_BUFFER_SIZE_TOO_SMALL</td>
<td>1010</td>
<td>Buffer size is insufficient.</td>
</tr>
<tr>
<td>EXFAT_TEST_ERR_LABEL_MISMATCH</td>
<td>1011</td>
<td>Label is incorrect.</td>
</tr>
<tr>
<td>EXFAT_TEST_ERR_WRONG_ATTRIBUTE</td>
<td>1012</td>
<td>File or directory’s attribute is incorrect.</td>
</tr>
<tr>
<td>EXFAT_TEST_ERR_TIMESTAMP_MISMATCH</td>
<td>1013</td>
<td>Timestamp is incorrect.</td>
</tr>
<tr>
<td>EXFAT_TEST_ERR_LONG_FILENAME_RW_MISMATCH</td>
<td>1014</td>
<td>Error reading/writing long filename.</td>
</tr>
<tr>
<td>EXFAT_TEST_ERR_LONG_FILEPATH_RW_MISMATCH</td>
<td>1015</td>
<td>Error reading/writing long file pathname.</td>
</tr>
<tr>
<td>EXFAT_TEST_ERR_TASK</td>
<td>1016</td>
<td>Task error.</td>
</tr>
<tr>
<td>EXFAT_TEST_ERR_EXCEEDS_PATH_LENGTH_LIMIT</td>
<td>1017</td>
<td>Pathname is too long.</td>
</tr>
<tr>
<td>EXFAT_TEST_ERR_SPACE_MISMATCH</td>
<td>1018</td>
<td>The used space obtained is not the expected amount.</td>
</tr>
<tr>
<td>EXFAT_TEST_ERR_DIR_READ_FILENAME_MISMATCH</td>
<td>1019</td>
<td>Directory contains unexpected file or directory.</td>
</tr>
<tr>
<td>Error</td>
<td>Value</td>
<td>Meaning</td>
</tr>
<tr>
<td>------------------------------------------------------------</td>
<td>-------</td>
<td>--------------------------------------------------------------------------</td>
</tr>
<tr>
<td>EXFAT_TEST_ERR_MULTIFILE_RW_MISMATCH</td>
<td>1020</td>
<td>The amount of data read does not match the amount of data written.</td>
</tr>
<tr>
<td>EXFAT_TEST_ERR_INITVOLUME_RW_MISMATCH</td>
<td>1021</td>
<td>After initializing the volume, the amount of data read does not match the amount of data written.</td>
</tr>
<tr>
<td>EXFAT_TEST_ERR_DIRECTORY_ERROR</td>
<td>1022</td>
<td>Directory does not have 'DIR' attribute.</td>
</tr>
<tr>
<td>EXFAT_TEST_ERRFAULT_INJECTION</td>
<td>1023</td>
<td>Cannot inject error using test media driver.</td>
</tr>
<tr>
<td>EXFAT_TEST_ERR_PUTC_GETC_RW_MISMATCH</td>
<td>1024</td>
<td><code>exfat_getc()/exfat_putc()</code> mismatch.</td>
</tr>
<tr>
<td>EXFAT_TEST_ERR_VOLUME_INFO</td>
<td>1025</td>
<td>Volume information error.</td>
</tr>
<tr>
<td>EXFAT_TEST_ERR_FILE_POS_MISMATCH</td>
<td>1026</td>
<td>File position error.</td>
</tr>
<tr>
<td>EXFAT_TEST_ERR_CLUSTERNAL_CHAIN_MISMATCH</td>
<td>1027</td>
<td>Cluster chain mismatch.</td>
</tr>
<tr>
<td>EXFAT_TEST_ERR_INVALID_DRIVE_INDEX</td>
<td>1028</td>
<td>Drive index is invalid.</td>
</tr>
</tbody>
</table>
5 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.

5.1 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 test suite uses the following OAL components:

<table>
<thead>
<tr>
<th>OAL Resource</th>
<th>Number required</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tasks</td>
<td>1</td>
</tr>
<tr>
<td>Mutexes</td>
<td>0 (but the Safe tests do require mutexes)</td>
</tr>
<tr>
<td>Events</td>
<td>2</td>
</tr>
</tbody>
</table>
5.2 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:

<table>
<thead>
<tr>
<th>Function</th>
<th>Package</th>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>psp_getrand()</td>
<td>psp_base</td>
<td>psp_rand</td>
<td>Generates a random number. This is used for the volume serial number.</td>
</tr>
<tr>
<td>psp_memcmp()</td>
<td>psp_base</td>
<td>psp_string</td>
<td>Compares two blocks of memory.</td>
</tr>
<tr>
<td>psp_memset()</td>
<td>psp_base</td>
<td>psp_string</td>
<td>Sets the specified area of memory to the defined value.</td>
</tr>
<tr>
<td>psp_printf()</td>
<td>psp_base</td>
<td>psp_stdio</td>
<td>Prints a string.</td>
</tr>
<tr>
<td>psp_snprintf()</td>
<td>psp_base</td>
<td>psp_stdio</td>
<td>Sends formatted output to compose a string holding the same text that would be printed if <code>fmt</code> was used on <code>psp_printf()</code>. Instead of being printed, the content is stored as a C string in a buffer.</td>
</tr>
<tr>
<td>psp_strncmp()</td>
<td>psp_base</td>
<td>psp_string</td>
<td>Compares two strings of defined length.</td>
</tr>
<tr>
<td>psp_strncpy()</td>
<td>psp_base</td>
<td>psp_string</td>
<td>Copies one string of defined length to another.</td>
</tr>
<tr>
<td>psp_w16csncat()</td>
<td>psp_base</td>
<td>psp_string</td>
<td>Concatenates a source string to the end of a destination string.</td>
</tr>
<tr>
<td>psp_w16csnchr()</td>
<td>psp_base</td>
<td>psp_string</td>
<td>Counts characters in a UTF-16 string buffer.</td>
</tr>
<tr>
<td>psp_w16csncmp()</td>
<td>psp_base</td>
<td>psp_string</td>
<td>Compares two strings, returning 0 when they match, otherwise -1 or 1.</td>
</tr>
<tr>
<td>psp_w16csncpy()</td>
<td>psp_base</td>
<td>psp_string</td>
<td>Copies a source string to the destination string, overwriting existing content.</td>
</tr>
<tr>
<td>psp_w16csnlen()</td>
<td>psp_base</td>
<td>psp_string</td>
<td>Returns the length of a UTF-16 string.</td>
</tr>
</tbody>
</table>
The module makes use of the following PSP functions. These are described in the following sections, as is the callback they use:

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>psp_exfat_test_set_error()</td>
<td>Sets error flags. This enables error injecting at the media driver level.</td>
</tr>
<tr>
<td>psp_exfat_test_clear_error()</td>
<td>Clears error flags; disables error injecting at the media driver level.</td>
</tr>
<tr>
<td>psp_exfat_test_set_write_error_counter()</td>
<td>Sets the write error counter which is decreased after every write. If this is zero, write operations are not executed, but no error is reported.</td>
</tr>
<tr>
<td>psp_exfat_test_reset_driver()</td>
<td>Resets the driver to the state it would be in after a power-on reset.</td>
</tr>
</tbody>
</table>

The module makes use of the following standard PSP macros:

<table>
<thead>
<tr>
<th>Macro</th>
<th>Package</th>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PSP_RD_BE16</td>
<td>psp_base</td>
<td>psp_endianness</td>
<td>Reads a 16 bit value stored as big-endian from a memory location.</td>
</tr>
<tr>
<td>PSP_RD_LE16</td>
<td>psp_base</td>
<td>psp_endianness</td>
<td>Reads a 16 bit value stored as little-endian from a memory location.</td>
</tr>
<tr>
<td>PSP_RD_BE32</td>
<td>psp_base</td>
<td>psp_endianness</td>
<td>Reads a 32 bit value stored as big-endian from a memory location.</td>
</tr>
<tr>
<td>PSP_RD_LE32</td>
<td>psp_base</td>
<td>psp_endianness</td>
<td>Reads a 32 bit value stored as little-endian from a memory location.</td>
</tr>
<tr>
<td>PSP_WR_BE16</td>
<td>psp_base</td>
<td>psp_endianness</td>
<td>Writes a 16 bit value to be stored as big-endian to a memory location.</td>
</tr>
<tr>
<td>PSP_WR_LE16</td>
<td>psp_base</td>
<td>psp_endianness</td>
<td>Writes a 16 bit value to be stored as little-endian to a memory location.</td>
</tr>
<tr>
<td>PSP_WR_BE32</td>
<td>psp_base</td>
<td>psp_endianness</td>
<td>Writes a 32 bit value to be stored as big-endian to a memory location.</td>
</tr>
</tbody>
</table>

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".
t_exfat_test_cb

The PSP provides this callback which is used by the PSP functions in this section.

**Format**

```
typedef void (* t_exfat_test_cb ) ( void )
```
**psp_exfat_test_set_error**

The PSP provides this function to set error flags. This enables error injecting at the media driver level.

**Format**

```c
t_exfat_ret psp_exfat_test_set_error ( 
    t_exfat_drive   drivenum, 
    uint64_t        error_flags )
```

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>drivenum</td>
<td>The drive (0='A', 1='B', and so on).</td>
<td>t_exfat_drive</td>
</tr>
<tr>
<td>error_flags</td>
<td>The error flags; the errors to inject.</td>
<td>uint64_t</td>
</tr>
</tbody>
</table>

**Return values**

<table>
<thead>
<tr>
<th>Return value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXFAT_NO_ERROR</td>
<td>Successful execution.</td>
</tr>
<tr>
<td>EXFAT_TEST_ERR_FAULT_INJECTION</td>
<td>Operation failed.</td>
</tr>
</tbody>
</table>
psp_exfat_test_clear_error

The PSP provides this function to clear the error flags. This disables error injecting at the media driver level.

Format

```
t_exfat_ret psp_exfat_test_clear_error ( 
    t_exfat_drive   drivenum, 
    uint64_t        error_flags )
```

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>drivenum</td>
<td>The drive (0='A', 1='B', and so on).</td>
<td>t_exfat_drive</td>
</tr>
<tr>
<td>error_flags</td>
<td>The error flags; the errors to clear.</td>
<td>uint64_t</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Return value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXFAT_NO_ERROR</td>
<td>Successful execution.</td>
</tr>
<tr>
<td>EXFAT_TEST_ERR_FAULT_INJECTION</td>
<td>Operation failed.</td>
</tr>
</tbody>
</table>
 PSP provides this function to reset the driver to the state it would be in after a power-on reset.

**Format**

```
t_exfat_ret psp_exfat_test_reset_driver ( t_exfat_drive drivenum )
```

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>drivenum</td>
<td>The drive (0='A', 1='B', and so on).</td>
<td>t_exfat_drive</td>
</tr>
</tbody>
</table>

**Return values**

<table>
<thead>
<tr>
<th>Return value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXFAT_NO_ERROR</td>
<td>Successful execution.</td>
</tr>
<tr>
<td>EXFAT_TEST_ERR_FAULT_INJECTION</td>
<td>Operation failed.</td>
</tr>
</tbody>
</table>
psp_exfat_test_set_write_error_counter

The PSP provides this function to set the write error counter that is decreased after every write. (If this is zero, write operations are not executed, but no error is reported.)

Format

```c
void psp_exfat_test_set_write_error_counter (
    t_exfat_drive drivenum,
    uint32_t write_error_counter,
    t_exfat_test_cb p_write_error_cb )
```

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>drivenum</td>
<td>The drive (0='A', 1='B', and so on).</td>
<td>t_exfat_drive</td>
</tr>
<tr>
<td>write_error_counter</td>
<td>The number of sector writes that is allowed. This is decreased after every write.</td>
<td>uint32_t</td>
</tr>
<tr>
<td>p_write_errorcb</td>
<td>A pointer to the callback.</td>
<td>t_exfat_test_cb</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Return value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXFAT_NO_ERROR</td>
<td>Successful execution.</td>
</tr>
<tr>
<td>EXFAT_TEST_ERR_FAULT_INJECTION</td>
<td>Operation failed.</td>
</tr>
</tbody>
</table>