File I/O Library

Microchip Libraries for Applications

Table of Contents

File I/O Library

1 File I/O Library

1.1 Introduction

Overview of this library's functionality and features.

Description

This File I/O library provides FAT file system (FAT12, FAT16, and FAT32) functionality for the Microchip family of microcontrollers with a convenient C language interface. There are two instances of this library- one that supports Long File Name functionality, and one that does not. The long file name version of the library offers additional functionality and produces (and accesses) files with more human-readable names, but it also uses more microcontroller resources.

This library can be used with multiple instances of one or more physical layers. These physical layers provide an interface into removable flash-based media that support the FAT file system.

1.2 Legal Information

Software License Agreement

Trademark Information

1.3 Release Notes

File I/O Library

This library has been moved to the following location for future updates/support:

https://github.com/MicrochipTech/mla_fileio

v1.05

- Speed improvement when searching for free cluster to create new file. By starting at the last successfully found cluster rather than the start of the drive.
- Fixes issue of writing after seeking causing trailing 0s in the file.
- Fixes error in DirectoryRemove failing to correctly delete folder when folder contains previously deleted folders/files.
- Fixes issue where a file marked with 8.3 designator not recognized as 8.3 file.
- Speed improvements for the seek function.
- Fixes issue where some drives may not mount successfully.
- Fixes issue where format request may not happen successfully.

v1.03

- Added required user defined callback for internal flash physical layer: DRV_FILEIO_INTERNAL_FLASH_CONFIG_UNLOCK_VERIFICATION_FUNCTION. This function should validate that the system parameters are valid for flash writes on the device on the board (Vdd, clock speeds, etc. are all in valid range for self writes).

v1.02

- Fixed issue with soft detect for SD card connectors that don't have card detect pin
- Licensed under Apache v2.0. License under MLA license available upon request.
- Relocated driver files from "<mla installation folder>/framework/drivers/fileio" to "<mla installation folder>/framework/fileio/drivers".

v1.01

- Added of support for devices with SPI with I2S module.
- Now allows I/O initialization function pointer to be NULL for hardware that has independent I/O initialization.
- Relocated the include files from the root directory of the library to the "/inc" folder. Projects must add "<install folder>/framework/fileio/inc" to their include path.

v1.00

This is the first release of the library.

Tested with MPLAB XC16 v1.24.

1.4 Using the Library

This topic describes the basic architecture of the File I/O Library and provides information and examples on how to use it.

Description

This topic describes the basic architecture of the File I/O Library and provides information and examples on how to use it.

Interface Header File: fileio.h or fileio_lfn.h

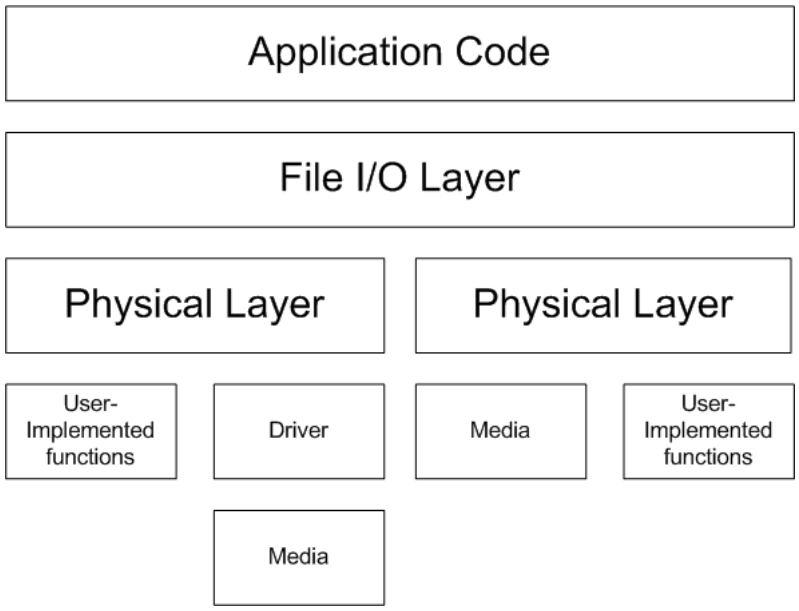
The interface to the File I/O library is defined by one of two header files. The "fileio.h" header file describes the API of the library version that supports short file names only. The "fileio_lfn.h" header file describes the API of the library version that supports long file names. The long file name library requires additional microcontroller resources. Any C language source (.c) file that uses the File I/O library should include "fileio.h" or "fileio_lfn.h."

1.4.1 Abstraction Model

This library provides the low-level abstraction of the File I/O module on the Microchip family of microcontrollers with a convenient C language interface. This topic describes how that abstraction is modeled in the software and introduces the library interface.

Description

File I/O Software Abstraction Block Diagram



The File I/O module model is relatively straightforward. The user will write application code that makes calls into the File I/O Layer. The File I/O Layer will then make calls into at least one Physical Layer (or one of multiple Physical Layers, depending on how the user has initialize and configured their device). The Physical Layer will either interface directly with the media, or use a separate driver to interface to the media. The Physical Layer may also call functions that are implemented by the user if necessary. For example, the SD-SPI Physical Layer will use the drv_spi SPI driver module to interface to an SD card, and it will also call user-implemented functions to set/clear the chip select pin and get the status of other I/O pins.

1.4.2 Library Overview

Describes the API sub-sections in the library.

Description

The library interface routines are divided into various sub-sections, each of sub-section addresses one of the blocks or the overall operation of the File I/O module.

File I/O Layer

This section describes API used for the File I/O layer.

Library Interface Section	Description
Short File Name Library API	Describes file I/O functions and types that are unique to the short file name version of this library.
Long File Name Library API	Describes file I/O functions and types that are unique to the long file name version of this library.
Common API	Describes file I/O functions and types that are common to both versions of this library.

Physical Layer

This section describes API used by the available physical layers.

Library Interface Section	Description
SD (SPI) Driver	Describes the physical layer and user-implemented functions and types for an SD/MMC Card Physical Layer that used SPI communications.

1.4.3 How the Library Works

Describes how the library works.

Description

General Information

Several functions in this library make use of path/name strings. In the short file name library, these are simply char strings; in the long file name library, they are uint16_t strings (unsigned short int). These pathnames can be specified as relative paths or as absolute paths. A relative path will perform the specified operation relative to a current working directory. An absolute path will perform the specified operation on the exact specified directory. You can use the FILEIO_CONFIG_DELIMITER configuration macro to specify the delimiter to use for path strings.

```
Relative path:
handle = FILEIO_Open ("DIR2/FILE1.TXT", ...

Absolute path:
handle = FILEIO_Open ("A:/DIR1/DIR2/FILE1.TXT", ...
```

Note that **Short File Names** can only use upper-case alphanumeric characters, the space character (0x20), and the following symbols:

```
! # $ % & ' ( ) - @ ^ _ ` { } ~
```

Each short file name can use between one and eight characters for the name, and up to three for the extension (e.g. "FILENAME.TXT", "FILE.TX", "F").

Alternatively,

Long File Names can support up to 255 UCS-2 characters, with the exception of the following characters:

```
\ / : * ? " < > |
```

Describing a Drive

Each media device you access will be described by an instance of the FILEIO_DRIVE_CONFIG structure. This structure contains function pointers and information that will be used to access that drive. You must maintain this structure in memory as long as the drive is mounted. For more information about this structure and the function pointer types it requires, please see the Physical Layer Functions topic.

Mounting a Drive

To begin using the File I/O library, you must first use the FILEIO_DriveMount function to mount a drive. This will initialize the drive and read all of the parameters that the File I/O library needs to access that drive. The first time that you mount a drive after power-up, that drive's root directory will be set as the current working directory. Each time you mount a drive, you will specify a single-character drive ID. You can use this drive ID in path strings to specify absolute paths. For an absolute path, the path must begin with a drive ID (char for short file name paths, uint16_t for long file name paths), followed by a colon, optionally followed by a delimiter character.

Some physical layers may require the user to specify additional parameters that define which instance of a drive should be used or how it should be accessed. This information will be passed into the mediaParameters argument in the FILEIO_DriveMount function. The format of this data will depend on the physical layer used.

When you are finished using a drive, you can unmount it using the FILEIO_DriveUnmount function. This will free the memory used to store drive information, and perform any media-specific de-initialization. You must close all open files on a drive before unmounting that drive, or they may become corrupted.

Opening and Closing Files

Before accessing any of the files on your device, you must open them with the FILEIO_Open function. Opening a file will read the file information from the drive and initialize variables to track the current read/write location in the file. If FILEIO_Open is successful, it will return true and populate the FILEIO_OBJECT structure that the user has specified. A pointer to this file object can then be passed into other library functions to perform operations on that file.

After you are finished accessing a file, you must close the file with FILEIO_Close. This will write any cached data to the file and update the file's information on the media.

User-Implemented Functionality

This library requires the user to implement a function to generate timestamps with the FILEIO_TIMESTAMP format. This function format must match the FILEIO_TimestampGet definition. Once this function is implemented, you can pass it to the library with the FILEIO_RegisterTimestampGet function. When modifying or creating files, the library will call this function to generate a timestamp for that file. The method used to generate the timestamps will be application-dependant (obtained from the RTCC, user-specified, obtained from an SNTP time server, etc).

Certain physical layers may also require you to implement application-specific functions that will be used by those physical layers.

1.5 Configuring the Library

Describes how to configure the File I/O library.

Modules

Name	Description
File I/O Configuration Options	Describes File I/O Layer configuration options.

Description

The configuration of the File I/O library is based on the files system_config.h and fileio_config.h.

These header files contain the configuration selection for the File I/O library. Based on the selections made, the File I/O library will support or not support selected features. These configuration settings will apply to all instances of the File I/O module.

These headers can be placed anywhere; however, the path of these headers needs to be present in the include search path for a successful build.

Each driver may require additional configuration files/options. For example, the SD-SPI physical layer requires the definitions given in sd_spi_config.h.

1.5.1 File I/O Configuration Options

Describes File I/O Layer configuration options.

Macros

Name	Description
FILEIO_CONFIG_MAX_DRIVES	Macro indicating how many drives can be mounted simultaneously.
FILEIO_CONFIG_DELIMITER	Defines a character to use as a delimiter for directories. Forward slash ('/') or backslash ('\') is recommended.
FILEIO_CONFIG_MEDIA_SECTOR_SIZE	Macro defining the maximum supported sector size for the FILEIO module. This value should always be 512 , 1024, 2048, or 4096 bytes. Most media uses 512-byte sector sizes.

Description

This section describes the configuration options used by the File I/O layer of this library. Typically, these options are defined in fileio_config.h, which is included in system_config.h. The system_config.h header is then included in all library files.

Some system-specific macros or functions used by the library (like the clock configuration macros/functions) are defined in system.c/h. The system.h file is also included in the library by files that use these functions.

1.5.1.1 Clock Configuration

Describes required clock configuration options for the File I/O library.

Macros

Name	Description
SYS_CLK_FrequencySystemGet	The File I/O library requires the user to define the system clock frequency (Hz)

SYS_CLK_FrequencyPeripheralGet	The File I/O library requires the user to define the peripheral clock frequency (Hz)
SYS_CLK_FrequencyInstructionGet	The File I/O library requires the user to define the instruction clock frequency (Hz)

Module

File I/O Configuration Options

Description

Several functions performed by the File I/O Library are timing-based. To facilitate these functions, the user must define several functions or macros to describe how the part is clocked.

1.5.1.1.1 SYS_CLK_FrequencySystemGet Macro

File

system_template.h

Syntax

```
#define SYS_CLK_FrequencySystemGet 32000000
```

Description

The File I/O library requires the user to define the system clock frequency (Hz)

1.5.1.1.2 SYS_CLK_FrequencyPeripheralGet Macro

File

system_template.h

Syntax

```
#define SYS_CLK_FrequencyPeripheralGet SYS_CLK_FrequencySystemGet()
```

Description

The File I/O library requires the user to define the peripheral clock frequency (Hz)

1.5.1.1.3 SYS_CLK_FrequencyInstructionGet Macro

File

system_template.h

Syntax

```
#define SYS_CLK_FrequencyInstructionGet (SYS_CLK_FrequencySystemGet() / 2)
```

Description

The File I/O library requires the user to define the instruction clock frequency (Hz)

1.5.1.2 Feature Disable

Describes macros that the user can define to disable File I/O library features.

Macros

Name	Description
FILEIO_CONFIG_DIRECTORY_DISABLE	Define FILEIO_CONFIG_FUNCTION_DIRECTORY to disable use of directories on your drive. Disabling this feature will limit you to performing all file operations in the root directory.

FILEIO_CONFIG_DRIVE_PROPERTIES_DISABLE	Define FILEIO_CONFIG_FUNCTION_DRIVE_PROPERTIES to disable the FILEIO_DrivePropertiesGet function. This function will determine the properties of your device, including unused memory.
FILEIO_CONFIG_FORMAT_DISABLE	Define FILEIO_CONFIG_FUNCTION_FORMAT to disable the function used to format drives.
FILEIO_CONFIG_MULTIPLE_BUFFER_MODE_DISABLE	Define FILEIO_CONFIG_MULTIPLE_BUFFER_MODE_DISABLE to disable multiple buffer mode. This will force the library to use a single instance of the FAT and Data buffer. Otherwise, it will use one FAT buffer and one data buffer per drive (defined by FILEIO_CONFIG_MAX_DRIVES). If you are only using one drive in your application, this option has no effect.
FILEIO_CONFIG_SEARCH_DISABLE	Define FILEIO_CONFIG_FUNCTION_SEARCH to disable the functions used to search for files.
FILEIO_CONFIG_WRITE_DISABLE	Define FILEIO_CONFIG_FUNCTION_WRITE to disable the functions that write to a drive. Disabling this feature will force the file system into read-only mode.

Module

File I/O Configuration Options

Description

At times the user may not want to use certain File I/O features. This section details macros that the user can define to disable certain features, which will cause the library to use fewer microcontroller resources.

1.5.1.2.1 FILEIO_CONFIG_DIRECTORY_DISABLE Macro

File

fileio_config_template.h

Syntax

```
#define FILEIO_CONFIG_DIRECTORY_DISABLE
```

Description

Define FILEIO_CONFIG_FUNCTION_DIRECTORY to disable use of directories on your drive. Disabling this feature will limit you to performing all file operations in the root directory.

1.5.1.2.2 FILEIO_CONFIG_DRIVE_PROPERTIES_DISABLE Macro

File

fileio_config_template.h

Syntax

```
#define FILEIO_CONFIG_DRIVE_PROPERTIES_DISABLE
```

Description

Define FILEIO_CONFIG_FUNCTION_DRIVE_PROPERTIES to disable the FILEIO_DrivePropertiesGet function. This function will determine the properties of your device, including unused memory.

1.5.1.2.3 FILEIO_CONFIG_FORMAT_DISABLE Macro

File

fileio_config_template.h

Syntax

```
#define FILEIO_CONFIG_FORMAT_DISABLE
```

Description

Define FILEIO_CONFIG_FUNCTION_FORMAT to disable the function used to format drives.

1.5.1.2.4 FILEIO_CONFIG_MULTIPLE_BUFFER_MODE_DISABLE Macro

File

fileio_config_template.h

Syntax

```
#define FILEIO_CONFIG_MULTIPLE_BUFFER_MODE_DISABLE
```

Description

Define FILEIO_CONFIG_MULTIPLE_BUFFER_MODE_DISABLE to disable multiple buffer mode. This will force the library to use a single instance of the FAT and Data buffer. Otherwise, it will use one FAT buffer and one data buffer per drive (defined by FILEIO_CONFIG_MAX_DRIVES). If you are only using one drive in your application, this option has no effect.

1.5.1.2.5 FILEIO_CONFIG_SEARCH_DISABLE Macro

File

fileio_config_template.h

Syntax

```
#define FILEIO_CONFIG_SEARCH_DISABLE
```

Description

Define FILEIO_CONFIG_FUNCTION_SEARCH to disable the functions used to search for files.

1.5.1.2.6 FILEIO_CONFIG_WRITE_DISABLE Macro

File

fileio_config_template.h

Syntax

```
#define FILEIO_CONFIG_WRITE_DISABLE
```

Description

Define FILEIO_CONFIG_FUNCTION_WRITE to disable the functions that write to a drive. Disabling this feature will force the file system into read-only mode.

1.5.1.3 FILEIO_CONFIG_MAX_DRIVES Macro

File

fileio_config_template.h

Syntax

```
#define FILEIO_CONFIG_MAX_DRIVES 1
```

Module

File I/O Configuration Options

Description

Macro indicating how many drives can be mounted simultaneously.

1.5.1.4 FILEIO_CONFIG_DELIMITER Macro

File

fileio_config_template.h

Syntax

```
#define FILEIO_CONFIG_DELIMITER '/'
```

Module

File I/O Configuration Options

Description

Defines a character to use as a delimiter for directories. Forward slash ('/') or backslash ('\') is recommended.

1.5.1.5 FILEIO_CONFIG_MEDIA_SECTOR_SIZE Macro

File

fileio_config_template.h

Syntax

```
#define FILEIO_CONFIG_MEDIA_SECTOR_SIZE 512
```

Module

File I/O Configuration Options

Description

Macro defining the maximum supported sector size for the FILEIO module. This value should always be 512 , 1024, 2048, or 4096 bytes. Most media uses 512-byte sector sizes.

1.5.2 Physical Layer Configuration Options

Modules

Name	Description
SD-SPI Configuration Options	Describes configuration options for the SD-SPI Physical Layer.
Internal Flash Configuration Options	Describes configuration options for the Internal Flash Physical Layer.

1.5.2.1 SD-SPI Configuration Options

Describes configuration options for the SD-SPI Physical Layer.

Macros

Name	Description
FILEIO_SD_CONFIG_MEDIA_SOFT_DETECT	Define FILEIO_SD_CONFIG_MEDIA_SOFT_DETECT to enable soft detect of an SD card. Some connectors do not have a card detect pin and must use software to detect the presence of a card.
FILEIO_SD_SendMediaCmd_Slow	Define the function to send a media command at a slow clock rate

FILEIO_SD_SPI_Get_Slow	Define the function to read an SPI byte at a slow clock rate
FILEIO_SD_SPI_Put_Slow	Define the function to write an SPI byte at a slow clock rate
FILEIO_SD_SPIInitialize_Slow	Define the function to initialize the SPI module for operation at a slow clock rate

Description

This section describes configuration options for the SD-SPI Physical Layer.

During the media initialization sequence for SD cards, it is necessary to clock the media at a frequency between 100 kHz and 400 kHz, since some media types power up in open drain output mode and cannot run fast initially. On PIC18 devices, when the CPU is running at full frequency, the standard SPI prescalars cannot reach a low enough SPI frequency. Therefore, we provide a number of function pointer configuration options to allow the user to remap the SPI functions called during the "slow" part of the initialization to user-implemented functions that can provide the correct functionality. For example, a bit-banged SPI module could be implemented to provide a clock between 100 and 400 kHz.

If the system clock can be scaled to provide an appropriate SPI clock frequency, these functions can simply be mapped to the fast SPI driver functions. Alternatively, you can decrease the PIC18's system clock frequency (by disabling the PLL, clock switching, etc) to provide a slow enough clock to allow SD Card initialization. If you choose this option, you must define the SYS_CLK_FrequencySystemGet function in a way that will return the correct clock frequency at both given clock frequencies.

Note: The SD-SPI physical layer makes use of the MLA's SPI driver (drv_spi.c/h). This driver requires additional configuration definitions to enable SPI channels or features (e.g. #define DRV_SPI_CONFIG_CHANNEL_1_ENABLE). For more information, please see the MLA Driver help file.

1.5.2.1.1 FILEIO_SD_CONFIG_MEDIA_SOFT_DETECT Macro

File

sd_spi_config_template.h

Syntax

```
#define FILEIO_SD_CONFIG_MEDIA_SOFT_DETECT
```

Module

SD-SPI Configuration Options

Description

Define FILEIO_SD_CONFIG_MEDIA_SOFT_DETECT to enable soft detect of an SD card. Some connectors do not have a card detect pin and must use software to detect the presence of a card.

1.5.2.1.2 FILEIO_SD_SendMediaCmd_Slow Macro

File

sd_spi_config_template.h

Syntax

```
#define FILEIO_SD_SendMediaCmd_Slow FILEIO_SD_SendCmd
```

Module

SD-SPI Configuration Options

Description

Define the function to send a media command at a slow clock rate

1.5.2.1.3 FILEIO_SD_SPI_Get_Slow Macro

File

sd_spi_config_template.h

Syntax

```
#define FILEIO_SD_SPI_Get_Slow DRV_SPI_Get
```

Module

SD-SPI Configuration Options

Description

Define the function to read an SPI byte at a slow clock rate

1.5.2.1.4 FILEIO_SD_SPI_Put_Slow Macro

File

sd_spi_config_template.h

Syntax

```
#define FILEIO_SD_SPI_Put_Slow DRV_SPI_Put
```

Module

SD-SPI Configuration Options

Description

Define the function to write an SPI byte at a slow clock rate

1.5.2.1.5 FILEIO_SD_SPIInitialize_Slow Macro

File

sd_spi_config_template.h

Syntax

```
#define FILEIO_SD_SPIInitialize_Slow FILEIO_SD_SPISlowInitialize
```

Module

SD-SPI Configuration Options

Description

Define the function to initialize the SPI module for operation at a slow clock rate

1.5.2.2 Internal Flash Configuration Options

Describes configuration options for the Internal Flash Physical Layer.

Macros

Name	Description
DRV_FILEIO_INTERNAL_FLASH_CONFIG_UNLOCK_VERIFICATION_FUNCTION	Macro maps to a user function that will determine if the system parameters are valid for a self write for the processor on the board. Returns true if the write can continue and false if it should not.

Description

This section describes configuration options for the Internal Flash Physical Layer.

1.5.2.2.1
DRV_FILEIO_INTERNAL_FLASH_CONFIG_UNLOCK_VERIFICATION_FUNCTION
Macro

File

internal_flash_config_template.h

Syntax

```
#define DRV_FILEIO_INTERNAL_FLASH_CONFIG_UNLOCK_VERIFICATION_FUNCTION true
```

Module

Internal Flash Configuration Options

Description

Macro maps to a user function that will determine if the system parameters are valid for a self write for the processor on the board. Returns true if the write can continue and false if it should not.

1.6 Building the Library

This section describes the source files that must be included when building the File I/O module.

Description

This section describes the source files that must be included when building the File I/O module.

File	Description	Condition
fileio.c	Source file for the short file name version of the library.	Must be included when using the short file name version of the library.
fileio_lfn.c	Source file for the long file name version of the library.	Must be included when using the long file name version of the library.
sd_spi.c	Source file for the SD-SPI driver.	Must be included when using the SD-SPI physical layer.
drv_spi.c	Source file for the MLA SPI driver.	Must be included when using the SD-SPI physical layer.

1.7 Library Interface

Describes the Application Programming Interface (API) functions of the File I/O library.

Description

This section describes the Application Programming Interface (API) functions of the File I/O library.

Refer to each section for a detailed description.

1.7.1 File I/O Layer

Describes the API of the File I/O functions used by the library.

Modules

Name	Description
Short File Name Library API	Describes APIs that are specific to the Short File Name version of the library defined by fileio.h.
Long File Name Library API	Describes APIs that are specific to the Long File Name version of the library defined by fileio_lfn.h.

Description

This section describes the API of the File I/O functions used by the library.

1.7.1.1 Short File Name Library API

Describes APIs that are specific to the Short File Name version of the library defined by fileio.h.

Functions

	Name	Description
	FILEIO_DriveMount	Initializes a drive and loads its configuration information.
	FILEIO_DriveUnmount	Unmounts a drive.
	FILEIO_Open	Opens a file for access.
	FILEIO_Remove	Deletes a file.
	FILEIO_Rename	Renames a file.
	FILEIO_Find	Searches for a file in the current working directory.
	FILEIO_DirectoryMake	Creates the directory/directories specified by 'path.'
	FILEIO_DirectoryChange	Changes the current working directory.
	FILEIO_DirectoryRemove	Deletes a directory.
	FILEIO_DirectoryGetCurrent	Gets the name of the current working directory.
	FILEIO_ErrorClear	Clears the last error on a drive.
	FILEIO_ErrorGet	Gets the last error condition of a drive.
	FILEIO_FileSystemTypeGet	Describes the file system type of a file system.

Description

This section describes APIs that are specific to the Short File Name version of the library defined by fileio.h. Most functions in this section have a corresponding function in the Long File Name version of the library that accepts Long File Name arguments.

1.7.1.1.1 FILEIO_DriveMount Function

Initializes a drive and loads its configuration information.

File

fileio.h

Syntax

```
FILEIO_ERROR_TYPE FILEIO_DriveMount(char driveId, const FILEIO_DRIVE_CONFIG * driveConfig,
void * mediaParameters);
```

Module

Short File Name Library API

Returns

- FILEIO_ERROR_NONE - Drive was mounted successfully
- FILEIO_ERROR_TOO_MANY_DRIVES_OPEN - You have already mounted the maximum number of drives. Change FILEIO_CONFIG_MAX_DRIVES in fileio_config.h to increase this.
- FILEIO_ERROR_WRITE - The library was not able to write cached data in the buffer to the device (can occur when using multiple drives and single buffer mode)
- FILEIO_ERROR_INIT_ERROR - The driver's Media Initialize function indicated that the media could not be initialized.
- FILEIO_ERROR_UNSUPPORTED_SECTOR_SIZE - The media's sector size exceeds the maximum sector size specified in fileio_config.h (FILEIO_CONFIG_MEDIA_SECTOR_SIZE macro)
- FILEIO_ERROR_BAD_SECTOR_READ - The stack could not read the boot sector of Master Boot Record from the media.
- FILEIO_ERROR_BAD_PARTITION - The boot signature in the MBR is bad on your media device.
- FILEIO_ERROR_UNSUPPORTED_FS - The partition is formatted with an unsupported file system.
- FILEIO_ERROR_NOT_FORMATTED - One of the parameters in the boot sector is bad in the partition being mounted.

Description

This function will initialize a drive and load the required information from it.

Preconditions

FILEIO_Initialize must have been called.

Parameters

Parameters	Description
char driveId	An alphanumeric character that will be used to identify the drive.
const FILEIO_DRIVE_CONFIG * driveConfig	Constant structure containing function pointers that the library will use to access the drive.
void * mediaParameters	Constant structure containing media-specific values that describe which instance of the media to use for this operation.

Function

FILEIO_ERROR_TYPE FILEIO_DriveMount (char driveId,

const FILEIO_DRIVE_CONFIG * driveConfig, void * mediaParameters);

1.7.1.1.2 FILEIO_DriveUnmount Function

Unmounts a drive.

File

fileio.h

Syntax

```
int FILEIO_DriveUnmount(const char driveId);
```

Module

Short File Name Library API

Returns

- If Success: FILEIO_RESULT_SUCCESS
- If Failure: FILEIO_RESULT_FAILURE

Description

Unmounts a drive from the file system and writes any pending data to the drive.

Preconditions

FILEIO_DriveMount must have been called.

Parameters

Parameters	Description
const char driveId	The character representation of the mounted drive.

Function

int FILEIO_DriveUnmount (const char driveID)

1.7.1.1.3 FILEIO_Open Function

Opens a file for access.

File

fileio.h

Syntax

```
int FILEIO_Open(FILEIO_OBJECT * filePtr, const char * pathName, uint16_t mode);
```

Module

Short File Name Library API

Returns

- If Success: FILEIO_RESULT_SUCCESS
- If Failure: FILEIO_RESULT_FAILURE

- Sets error code which can be retrieved with FILEIO_ErrorGet Note that if the path cannot be resolved, the error will be returned for the current working directory.
  - FILEIO_ERROR_INVALID_ARGUMENT - The path could not be resolved.
  - FILEIO_ERROR_WRITE_PROTECTED - The device is write protected or this function was called in a write/create mode when writes are disabled in configuration.
  - FILEIO_ERROR_INVALID_FILENAME - The file name is invalid.
  - FILEIO_ERROR_ERASE_FAIL - There was an error when trying to truncate the file.
  - FILEIO_ERROR_WRITE - Cached file data could not be written to the device.
  - FILEIO_ERROR_DONE - The directory entry could not be found.

- FILEIO_ERROR_BAD_SECTOR_READ - The directory entry could not be cached.
- FILEIO_ERROR_DRIVE_FULL - There are no more clusters available on this device that can be allocated to the file.
- FILEIO_ERROR_FILENAME_EXISTS - All of the possible alias values for this file are in use.
- FILEIO_ERROR_BAD_CACHE_READ - There was an error caching LFN entries.
- FILEIO_ERROR_INVALID_CLUSTER - The next cluster in the file is invalid (can occur in APPEND mode).
- FILEIO_ERROR_COULD_NOT_GET_CLUSTER - There was an error finding the cluster that contained the specified offset (can occur in APPEND mode).

Description

Opens a file for access using a combination of modes specified by the user.

Preconditions

The drive containing the file must be mounted.

Parameters

Parameters	Description
FILEIO_OBJECT * filePtr	Pointer to the file object to initialize
const char * pathName	The path/name of the file to open.
uint16_t mode	The mode in which the file should be opened. Specified by inclusive or'ing parameters from FILEIO_OPEN_ACCESS_MODES.

Function

int FILEIO_Open (FILEIO_OBJECT * filePtr, const char * pathName, uint16_t mode)

1.7.1.1.4 FILEIO_Remove Function

Deletes a file.

File

fileio.h

Syntax

```
int FILEIO_Remove(const char * pathName);
```

Module

Short File Name Library API

Returns

- If Success: FILEIO_RESULT_SUCCESS
- If Failure: FILEIO_RESULT_FAILURE

- Sets error code which can be retrieved with FILEIO_ErrorGet. Note that if the path cannot be resolved, the error will be returned for the current working directory.
  - FILEIO_ERROR_INVALID_ARGUMENT - The path could not be resolved.
  - FILEIO_ERROR_WRITE_PROTECTED - The device is write-protected.
  - FILEIO_ERROR_INVALID_FILENAME - The file name is invalid.
  - FILEIO_ERROR_DELETE_DIR - The file being deleted is actually a directory (use FILEIO_DirectoryRemove)
  - FILEIO_ERROR_ERASE_FAIL - The erase operation failed.
  - FILEIO_ERROR_FILE_NOT_FOUND - The file entries for this file are invalid or have already been erased.
  - FILEIO_ERROR_WRITE - The updated file data and entry could not be written to the device.

- FILEIO_ERROR_DONE - The directory entry could not be found.
- FILEIO_ERROR_BAD_SECTOR_READ - The directory entry could not be cached.

Description

Deletes the file specified by pathName.

Preconditions

The file's drive must be mounted and the file should exist.

Parameters

Parameters	Description
const char * pathName	The path/name of the file.

Function

int FILEIO_Remove (const char * pathName)

1.7.1.1.5 FILEIO_Rename Function

Renames a file.

File

fileio.h

Syntax

```
int FILEIO_Rename(const char * oldPathName, const char * newFileName);
```

Module

Short File Name Library API

Returns

- If Success: FILEIO_RESULT_SUCCESS
- If Failure: FILEIO_RESULT_FAILURE

- Sets error code which can be retrieved with FILEIO_ErrorGet Note that if the path cannot be resolved, the error will be returned for the current working directory.
  - FILEIO_ERROR_INVALID_ARGUMENT - The path could not be resolved.
  - FILEIO_ERROR_WRITE_PROTECTED - The device is write-protected.
  - FILEIO_ERROR_INVALID_FILENAME - One of the file names is invalid.
  - FILEIO_ERROR_FILENAME_EXISTS - The new file name already exists on this device.
  - FILEIO_ERROR_FILE_NOT_FOUND - The file could not be found.
  - FILEIO_ERROR_WRITE - The updated file data and entry could not be written to the device.
  - FILEIO_ERROR_DONE - The directory entry could not be found or the library could not find a sufficient number of empty entries in the dir to store the new file name.
  - FILEIO_ERROR_BAD_SECTOR_READ - The directory entry could not be cached.
  - FILEIO_ERROR_ERASE_FAIL - The file's entries could not be erased (applies when renaming a long file name)
  - FILEIO_ERROR_DIR_FULL - New file entries could not be created.
  - FILEIO_ERROR_BAD_CACHE_READ - The lfn entries could not be cached.

Description

Renames a file specifed by oldPathname to the name specified by newFilename.

Preconditions

The file's drive must be mounted and the file/path specified by oldPathname must exist.

Parameters

Parameters	Description
const char * oldPathName	The path/name of the file to rename.
const char * newFileName	The new name of the file.

Function

int FILEIO_Rename (const char * oldPathname, const char * newFilename)

1.7.1.1.6 FILEIO_Find Function

Searches for a file in the current working directory.

File

fileio.h

Syntax

```
int FILEIO_Find(const char * fileName, unsigned int attr, FILEIO_SEARCH_RECORD * record,
bool newSearch);
```

Module

Short File Name Library API

Returns

- If Success: FILEIO_RESULT_SUCCESS
- If Failure: FILEIO_RESULT_FAILURE
- Returns file information in the record parameter.

- Sets error code which can be retrieved with FILEIO_ErrorGet Note that if the path cannot be resolved, the error will be returned for the current working directory.
  - FILEIO_ERROR_INVALID_ARGUMENT - The path could not be resolved.
  - FILEIO_ERROR_INVALID_FILENAME - The file name is invalid.
  - FILEIO_ERROR_BAD_CACHE_READ - There was an error searching directory entries.
  - FILEIO_ERROR_DONE - File not found.

Description

Searches for a file in the current working directory.

Preconditions

A drive must have been mounted by the FILEIO library.

Parameters

Parameters	Description
const char * fileName	The file's name. May contain limited partial string search elements. '?' can be used as a single-character wild-card and '*' can be used as a multiple-character wild card (only at the end of the file's name or extension).
unsigned int attr	Inclusive OR of all of the attributes (FILEIO_ATTRIBUTES structure members) that a found file may have.

FILEIO_SEARCH_RECORD * record	Structure containing parameters about the found file. Also contains private information used for additional searches for files that match the given criteria in the same directory.
bool newSearch	true if this is the first search for the specified file parameters in the specified directory, false otherwise. This parameter must be specified as 'true' the first time this function is called with any given FILEIO_SEARCH_RECORD structure. The same FILEIO_SEARCH_RECORD structure should be used with subsequent calls of this function to search for additional files matching the given criteria.

Function

int FILEIO_Find (const char * fileName, unsigned int attr,

FILEIO_SEARCH_RECORD * record, bool newSearch)

1.7.1.1.7 FILEIO_DirectoryMake Function

Creates the directory/directories specified by 'path.'

File

fileio.h

Syntax

```
int FILEIO_DirectoryMake(const char * path);
```

Module

Short File Name Library API

Returns

- If Success: FILEIO_RESULT_SUCCESS
- If Failure: FILEIO_RESULT_FAILURE

Description

Creates the directory/directories specified by 'path.'

Preconditions

The specified drive must be mounted.

Parameters

Parameters	Description
const char * path	Path string containing all directories to create.

Function

int FILEIO_DirectoryMake (const char * path)

1.7.1.1.8 FILEIO_DirectoryChange Function

Changes the current working directory.

File

fileio.h

Syntax

```
int FILEIO_DirectoryChange(const char * path);
```

Module

Short File Name Library API

Returns

- If Success: FILEIO_RESULT_SUCCESS
- If Failure: FILEIO_RESULT_FAILURE

Description

Changes the current working directory to the directory specified by 'path.'

Preconditions

The specified drive must be mounted and the directory being changed to should exist.

Parameters

Parameters	Description
const char * path	The path of the directory to change to.

Function

int FILEIO_DirectoryChange (const char * path)

1.7.1.1.9 FILEIO_DirectoryRemove Function

Deletes a directory.

File

fileio.h

Syntax

```
int FILEIO_DirectoryRemove(const char * pathName);
```

Module

Short File Name Library API

Returns

- If Success: FILEIO_RESULT_SUCCESS
- If Failure: FILEIO_RESULT_FAILURE

Description

Deletes a directory. The specified directory must be empty.

Preconditions

The directory's drive must be mounted and the directory should exist.

Parameters

Parameters	Description
const char * pathName	The path/name of the directory to delete.

Function

int FILEIO_DirectoryRemove (const char * pathName)

1.7.1.1.10 FILEIO_DirectoryGetCurrent Function

Gets the name of the current working directory.

File

fileio.h

Syntax

```
uint16_t FILEIO_DirectoryGetCurrent(char * buffer, uint16_t size);
```

Module

Short File Name Library API

Returns

- uint16_t - The number of characters in the current working directory name. May exceed the size of the buffer. In this case, the name will be truncated to 'size' characters, but the full length of the path name will be returned.
- Sets error code which can be retrieved with FILEIO_ErrorGet
  - FILEIO_ERROR_INVALID_ARGUMENT - The arguments for the buffer or its size were invalid.
  - FILEIO_ERROR_DIR_NOT_FOUND - One of the directories in your current working directory could not be found in its parent directory.

Description

Gets the name of the current working directory and stores it in 'buffer.' The directory name will be null-terminated. If the buffer size is insufficient to contain the whole path name, as much as possible will be copied and null-terminated.

Preconditions

A drive must be mounted.

Parameters

Parameters	Description
char * buffer	The buffer to contain the current working directory name.
uint16_t size	Size of the buffer (bytes).

Function

uint16_t FILEIO_DirectoryGetCurrent (char * buffer, uint16_t size)

1.7.1.1.11 FILEIO_ErrorClear Function

Clears the last error on a drive.

File

fileio.h

Syntax

```
void FILEIO_ErrorClear(char driveId);
```

Module

Short File Name Library API

Returns

void

Description

Clears the last error of the specified drive.

Preconditions

The drive must have been mounted.

Parameters

Parameters	Description
char driveId	The character representation of the drive.

Function

void FILEIO_ErrorClear (char driveId)

1.7.1.1.12 FILEIO_ErrorGet Function

Gets the last error condition of a drive.

File

fileio.h

Syntax

```
FILEIO_ERROR_TYPE FILEIO_ErrorGet(char driveId);
```

Module

Short File Name Library API

Returns

FILEIO_ERROR_TYPE - The last error that occurred on the drive.

Description

Gets the last error condition of the specified drive.

Preconditions

The drive must have been mounted.

Parameters

Parameters	Description
char driveId	The character representation of the drive.

Function

FILEIO_ERROR_TYPE FILEIO_ErrorGet (char driveId)

1.7.1.1.13 FILEIO_FileSystemTypeGet Function

Describes the file system type of a file system.

File

fileio.h

Syntax

```
FILEIO_FILE_SYSTEM_TYPE FILEIO_FileSystemTypeGet(char driveId);
```

Module

Short File Name Library API

Returns

- If Success: FILEIO_FILE_SYSTEM_TYPE enumeration member
- If Failure: FILEIO_FILE_SYSTEM_NONE

Description

Describes the file system type of a file system.

Preconditions

A drive must have been mounted by the FILEIO library.

Parameters

Parameters	Description
char driveId	Character representation of the mounted device.

Function

FILEIO_FILE_SYSTEM_TYPE FILEIO_FileSystemTypeGet (char driveId)

1.7.1.2 Long File Name Library API

Describes APIs that are specific to the Long File Name version of the library defined by fileio_lfn.h.

Functions

	Name	Description
	FILEIO_DriveMount	Initializes a drive and loads its configuration information.
	FILEIO_DriveUnmount	Unmounts a drive.
	FILEIO_Open	Opens a file for access.
	FILEIO_Remove	Deletes a file.
	FILEIO_Rename	Renames a file.
	FILEIO_Find	Searches for a file in the current working directory.
	FILEIO_DirectoryMake	Creates the directory/directories specified by 'path.'
	FILEIO_DirectoryChange	Changes the current working directory.
	FILEIO_DirectoryRemove	Deletes a directory.
	FILEIO_DirectoryGetCurrent	Gets the name of the current working directory.
	FILEIO_ErrorClear	Clears the last error on a drive.
	FILEIO_ErrorGet	Gets the last error condition of a drive.
	FILEIO_FileSystemTypeGet	Describes the file system type of a file system.
	FILEIO_Format	Formats a drive.
	FILEIO_ShortFileNameGet	Obtains the short file name of an open file.

Description

This section describes APIs that are specific to the Long File Name version of the library defined by fileio_lfn.h. Most functions in this section have a corresponding function in the Short File Name version of the library that accepts Short File Name arguments.

1.7.1.2.1 FILEIO_DriveMount Function

Initializes a drive and loads its configuration information.

File

fileio_lfn.h

Syntax

```
FILEIO_ERROR_TYPE FILEIO_DriveMount(uint16_t driveId, const FILEIO_DRIVE_CONFIG *
driveConfig, void * mediaParameters);
```

Module

Long File Name Library API

Returns

- FILEIO_ERROR_NONE - Drive was mounted successfully
- FILEIO_ERROR_TOO_MANY_DRIVES_OPEN - You have already mounted the maximum number of drives. Change FILEIO_CONFIG_MAX_DRIVES in fileio_config.h to increase this.
- FILEIO_ERROR_WRITE - The library was not able to write cached data in the buffer to the device (can occur when using multiple drives and single buffer mode)
- FILEIO_ERROR_INIT_ERROR - The driver's Media Initialize function indicated that the media could not be initialized.
- FILEIO_ERROR_UNSUPPORTED_SECTOR_SIZE - The media's sector size exceeds the maximum sector size specified in fileio_config.h (FILEIO_CONFIG_MEDIA_SECTOR_SIZE macro)
- FILEIO_ERROR_BAD_SECTOR_READ - The stack could not read the boot sector of Master Boot Record from the media.
- FILEIO_ERROR_BAD_PARTITION - The boot signature in the MBR is bad on your media device.
- FILEIO_ERROR_UNSUPPORTED_FS - The partition is formatted with an unsupported file system.
- FILEIO_ERROR_NOT_FORMATTED - One of the parameters in the boot sector is bad in the partition being mounted.

Description

This function will initialize a drive and load the required information from it.

Preconditions

FILEIO_Initialize must have been called.

Parameters

Parameters	Description
uint16_t driveId	A Unicode character that will be used to identify the drive.
const FILEIO_DRIVE_CONFIG * driveConfig	Constant structure containing function pointers that the library will use to access the drive.
void * mediaParameters	Constant structure containing media-specific values that describe which instance of the media to use for this operation.

Function

FILEIO_ERROR_TYPE FILEIO_DriveMount (uint16_t driveId,

const FILEIO_DRIVE_CONFIG * driveConfig,

void * mediaParameters);

1.7.1.2.2 FILEIO_DriveUnmount Function

Unmounts a drive.

File

fileio_lfn.h

Syntax

```
int FILEIO_DriveUnmount(const uint16_t driveId);
```

Module

Long File Name Library API

Returns

- If Success: FILEIO_RESULT_SUCCESS
- If Failure: FILEIO_RESULT_FAILURE

Description

Unmounts a drive from the file system and writes any pending data to the drive.

Preconditions

FILEIO_DriveMount must have been called.

Parameters

Parameters	Description
const uint16_t driveId	The character representation of the mounted drive.

Function

int FILEIO_DriveUnmount (const uint16_t driveID)

1.7.1.2.3 FILEIO_Open Function

Opens a file for access.

File

fileio_lfn.h

Syntax

```
int FILEIO_Open(FILEIO_OBJECT * filePtr, const uint16_t * pathName, uint16_t mode);
```

Module

Long File Name Library API

Returns

- If Success: FILEIO_RESULT_SUCCESS
- If Failure: FILEIO_RESULT_FAILURE

- Sets error code which can be retrieved with FILEIO_ErrorGet Note that if the path cannot be resolved, the error will be returned for the current working directory.
  - FILEIO_ERROR_INVALID_ARGUMENT - The path could not be resolved.
  - FILEIO_ERROR_WRITE_PROTECTED - The device is write protected or this function was called in a write/create mode when writes are disabled in configuration.
  - FILEIO_ERROR_INVALID_FILENAME - The file name is invalid.
  - FILEIO_ERROR_ERASE_FAIL - There was an error when trying to truncate the file.
  - FILEIO_ERROR_WRITE - Cached file data could not be written to the device.
  - FILEIO_ERROR_DONE - The directory entry could not be found.
  - FILEIO_ERROR_BAD_SECTOR_READ - The directory entry could not be cached.
  - FILEIO_ERROR_DRIVE_FULL - There are no more clusters available on this device that can be allocated to the file.
  - FILEIO_ERROR_FILENAME_EXISTS - All of the possible alias values for this file are in use.
  - FILEIO_ERROR_BAD_CACHE_READ - There was an error caching LFN entries.
  - FILEIO_ERROR_INVALID_CLUSTER - The next cluster in the file is invalid (can occur in APPEND mode).
  - FILEIO_ERROR_COULD_NOT_GET_CLUSTER - There was an error finding the cluster that contained the specified offset (can occur in APPEND mode).

Description

Opens a file for access using a combination of modes specified by the user.

Preconditions

The drive containing the file must be mounted.

Parameters

Parameters	Description
FILEIO_OBJECT * filePtr	Pointer to the file object to initialize
const uint16_t * pathName	The path/name of the file to open.
uint16_t mode	The mode in which the file should be opened. Specified by inclusive or'ing parameters from FILEIO_OPEN_ACCESS_MODES.

Function

int FILEIO_Open (FILEIO_OBJECT * filePtr, const uint16_t * pathName, uint16_t mode)

1.7.1.2.4 FILEIO_Remove Function

Deletes a file.

File

fileio_lfn.h

Syntax

```
int FILEIO_Remove(const uint16_t * pathName);
```

Module

Long File Name Library API

Returns

- If Success: FILEIO_RESULT_SUCCESS
- If Failure: FILEIO_RESULT_FAILURE

- Sets error code which can be retrieved with FILEIO_ErrorGet. Note that if the path cannot be resolved, the error will be returned for the current working directory.
  - FILEIO_ERROR_INVALID_ARGUMENT - The path could not be resolved.
  - FILEIO_ERROR_WRITE_PROTECTED - The device is write-protected.
  - FILEIO_ERROR_INVALID_FILENAME - The file name is invalid.
  - FILEIO_ERROR_DELETE_DIR - The file being deleted is actually a directory (use FILEIO_DirectoryRemove)
  - FILEIO_ERROR_ERASE_FAIL - The erase operation failed.
  - FILEIO_ERROR_FILE_NOT_FOUND - The file entries for this file are invalid or have already been erased.
  - FILEIO_ERROR_WRITE - The updated file data and entry could not be written to the device.
  - FILEIO_ERROR_DONE - The directory entry could not be found.
  - FILEIO_ERROR_BAD_SECTOR_READ - The directory entry could not be cached.

Description

Deletes the file specified by pathName.

Preconditions

The file's drive must be mounted and the file should exist.

Parameters

Parameters	Description
const uint16_t * pathName	The path/name of the file.

Function

int FILEIO_Remove (const char * pathName)

1.7.1.2.5 FILEIO_Rename Function

Renames a file.

File

fileio_lfn.h

Syntax

```
int FILEIO_Rename(const uint16_t * oldPathName, const uint16_t * newFileName);
```

Module

Long File Name Library API

Returns

- If Success: FILEIO_RESULT_SUCCESS
- If Failure: FILEIO_RESULT_FAILURE

- Sets error code which can be retrieved with FILEIO_ErrorGet Note that if the path cannot be resolved, the error will be returned for the current working directory.
  - FILEIO_ERROR_INVALID_ARGUMENT - The path could not be resolved.
  - FILEIO_ERROR_WRITE_PROTECTED - The device is write-protected.
  - FILEIO_ERROR_INVALID_FILENAME - One of the file names is invalid.
  - FILEIO_ERROR_FILENAME_EXISTS - The new file name already exists on this device.
  - FILEIO_ERROR_FILE_NOT_FOUND - The file could not be found.
  - FILEIO_ERROR_WRITE - The updated file data and entry could not be written to the device.
  - FILEIO_ERROR_DONE - The directory entry could not be found or the library could not find a sufficient number of empty entries in the dir to store the new file name.
  - FILEIO_ERROR_BAD_SECTOR_READ - The directory entry could not be cached.
  - FILEIO_ERROR_ERASE_FAIL - The file's entries could not be erased (applies when renaming a long file name)
  - FILEIO_ERROR_DIR_FULL - New file entries could not be created.
  - FILEIO_ERROR_BAD_CACHE_READ - The lfn entries could not be cached.

Description

Renames a file specifed by oldPathname to the name specified by newFilename.

Preconditions

The file's drive must be mounted and the file/path specified by oldPathname must exist.

Parameters

Parameters	Description
const uint16_t * oldPathName	The path/name of the file to rename.
const uint16_t * newFileName	The new name of the file.

Function

int FILEIO_Rename (const uint16_t * oldPathname,

const uint16_t * newFilename)

1.7.1.2.6 FILEIO_Find Function

Searches for a file in the current working directory.

File

fileio_lfn.h

Syntax

```
int FILEIO_Find(const uint16_t * fileName, unsigned int attr, FILEIO_SEARCH_RECORD *
record, bool newSearch);
```

Module

Long File Name Library API

Returns

- If Success: FILEIO_RESULT_SUCCESS
- If Failure: FILEIO_RESULT_FAILURE
- Returns file information in the record parameter.

- Sets error code which can be retrieved with FILEIO_ErrorGet Note that if the path cannot be resolved, the error will be returned for the current working directory.
  - FILEIO_ERROR_INVALID_ARGUMENT - The path could not be resolved.
  - FILEIO_ERROR_INVALID_FILENAME - The file name is invalid.
  - FILEIO_ERROR_BAD_CACHE_READ - There was an error searching directory entries.
  - FILEIO_ERROR_DONE - File not found.

Description

Searches for a file in the current working directory.

Preconditions

A drive must have been mounted by the FILEIO library.

Parameters

Parameters	Description
const uint16_t * fileName	The file's name. May contain limited partial string search elements. '?' can be used as a single-character wild-card and '*' can be used as a multiple-character wild card (only at the end of the file's name or extension).
unsigned int attr	Inclusive OR of all of the attributes (FILEIO_ATTRIBUTES structure members) that a found file may have.
FILEIO_SEARCH_RECORD * record	Structure containing parameters about the found file. Also contains private information used for additional searches for files that match the given criteria in the same directory.

bool newSearch	true if this is the first search for the specified file parameters in the specified directory, false otherwise. This parameter must be specified as 'true' the first time this function is called with any given FILEIO_SEARCH_RECORD structure. The same FILEIO_SEARCH_RECORD structure should be used with subsequent calls of this function to search for additional files matching the given criteria.

Function

int FILEIO_Find (const char * fileName, unsigned int attr,

FILEIO_SEARCH_RECORD * record, bool newSearch)

1.7.1.2.7 FILEIO_DirectoryMake Function

Creates the directory/directories specified by 'path.'

File

fileio_lfn.h

Syntax

```
int FILEIO_DirectoryMake(const uint16_t * path);
```

Module

Long File Name Library API

Returns

- If Success: FILEIO_RESULT_SUCCESS
- If Failure: FILEIO_RESULT_FAILURE

Description

Creates the directory/directories specified by 'path.'

Preconditions

The specified drive must be mounted.

Parameters

Parameters	Description
const uint16_t * path	Path string containing all directories to create.

Function

int FILEIO_DirectoryMake (const uint16_t * path)

1.7.1.2.8 FILEIO_DirectoryChange Function

Changes the current working directory.

File

fileio_lfn.h

Syntax

```
int FILEIO_DirectoryChange(const uint16_t * path);
```

Module

Long File Name Library API

Returns

- If Success: FILEIO_RESULT_SUCCESS
- If Failure: FILEIO_RESULT_FAILURE

Description

Changes the current working directory to the directory specified by 'path.'

Preconditions

The specified drive must be mounted and the directory being changed to should exist.

Parameters

Parameters	Description
const uint16_t * path	The path of the directory to change to.

Function

int FILEIO_DirectoryChange (const uint16_t * path)

1.7.1.2.9 FILEIO_DirectoryRemove Function

Deletes a directory.

File

fileio_lfn.h

Syntax

```
int FILEIO_DirectoryRemove(const uint16_t * pathName);
```

Module

Long File Name Library API

Returns

- If Success: FILEIO_RESULT_SUCCESS
- If Failure: FILEIO_RESULT_FAILURE

Description

Deletes a directory. The specified directory must be empty.

Preconditions

The directory's drive must be mounted and the directory should exist.

Parameters

Parameters	Description
const uint16_t * pathName	The path/name of the directory to delete.

Function

int FILEIO_DirectoryRemove (const uint16_t * pathName)

1.7.1.2.10 FILEIO_DirectoryGetCurrent Function

Gets the name of the current working directory.

File

fileio_lfn.h

Syntax

```
uint16_t FILEIO_DirectoryGetCurrent(uint16_t * buffer, uint16_t size);
```

Module

Long File Name Library API

Returns

- uint16_t - The number of characters in the current working directory name. May exceed the size of the buffer. In this case, the name will be truncated to 'size' characters, but the full length of the path name will be returned.
- Sets error code which can be retrieved with FILEIO_ErrorGet
  - FILEIO_ERROR_INVALID_ARGUMENT - The arguments for the buffer or its size were invalid.
  - FILEIO_ERROR_DIR_NOT_FOUND - One of the directories in your current working directory could not be found in its parent directory.

Description

Gets the name of the current working directory and stores it in 'buffer.' The directory name will be null-terminated. If the buffer size is insufficient to contain the whole path name, as much as possible will be copied and null-terminated.

Preconditions

A drive must be mounted.

Parameters

Parameters	Description
uint16_t * buffer	The buffer to contain the current working directory name.
uint16_t size	Size of the buffer (16-bit words).

Function

uint16_t FILEIO_DirectoryGetCurrent (uint16_t * buffer, uint16_t size)

1.7.1.2.11 FILEIO_ErrorClear Function

Clears the last error on a drive.

File

fileio_lfn.h

Syntax

```
void FILEIO_ErrorClear(uint16_t driveId);
```

Module

Long File Name Library API

Returns

void

Description

Clears the last error of the specified drive.

Preconditions

The drive must have been mounted.

Parameters

Parameters	Description
uint16_t driveId	The character representation of the drive.

Function

void FILEIO_ErrorClear (uint16_t driveId)

1.7.1.2.12 FILEIO_ErrorGet Function

Gets the last error condition of a drive.

File

fileio_lfn.h

Syntax

```
FILEIO_ERROR_TYPE FILEIO_ErrorGet(uint16_t driveId);
```

Module

Long File Name Library API

Returns

FILEIO_ERROR_TYPE - The last error that occurred on the drive.

Description

Gets the last error condition of the specified drive.

Preconditions

The drive must have been mounted.

Parameters

Parameters	Description
uint16_t driveId	The character representation of the drive.

Function

FILEIO_ERROR_TYPE FILEIO_ErrorGet (uint16_t driveId)

1.7.1.2.13 FILEIO_FileSystemTypeGet Function

Describes the file system type of a file system.

File

fileio_lfn.h

Syntax

```
FILEIO_FILE_SYSTEM_TYPE FILEIO_FileSystemTypeGet(uint16_t driveId);
```

Module

Long File Name Library API

Returns

- If Success: FILEIO_FILE_SYSTEM_TYPE enumeration member
- If Failure: FILEIO_FILE_SYSTEM_NONE

Description

Describes the file system type of a file system.

Preconditions

A drive must have been mounted by the FILEIO library.

Parameters

Parameters	Description
uint16_t driveId	Character representation of the mounted device.

Function

FILEIO_FILE_SYSTEM_TYPE FILEIO_FileSystemTypeGet (uint16_t driveId)

1.7.1.2.14 FILEIO_Format Function

Formats a drive.

File

fileio_lfn.h

Syntax

```
int FILEIO_Format(FILEIO_DRIVE_CONFIG * config, void * mediaParameters, FILEIO_FORMAT_MODE
mode, uint32_t serialNumber, char * volumeId);
```

Module

Long File Name Library API

Returns

- If Success: FILEIO_RESULT_SUCCESS
- If Failure: FILEIO_RESULT_FAILURE

Description

Formats a drive.

Preconditions

FILEIO_Initialize must have been called.

Parameters

Parameters	Description
FILEIO_DRIVE_CONFIG * config	Drive configuration pointer
FILEIO_FORMAT_MODE mode	FILEIO_FORMAT_MODE specifier
uint32_t serialNumber	Serial number to write to the drive
char * volumeId	Name of the drive.

Function

int FILEIO_Format (FILEIO_DRIVE_CONFIG * config,

void * mediaParameters, char mode,

uint32_t serialNumber, char * volumeID)

1.7.1.2.15 FILEIO_ShortFileNameGet Function

Obtains the short file name of an open file.

File

fileio_lfn.h

Syntax

```
void FILEIO_ShortFileNameGet(FILEIO_OBJECT * filePtr, char * buffer);
```

Module

Long File Name Library API

Returns

None

Description

Obtains the short file name of an open file.

Preconditions

A drive must have been mounted by the FILEIO library and the file being specified my be open.

Parameters

Parameters	Description
FILEIO_OBJECT * filePtr	Pointer to an open file.
char * buffer	A buffer to store the null-terminated short file name. Must be large enough to contain at least 13 characters.

Function

void FILEIO_ShortFileNameGet (FILEIO_OBJECT * filePtr, char * buffer)

1.7.1.3 Common API

Describes APIs that are common to both versions of the File I/O library.

Enumerations

Name	Description
FILEIO_ATTRIBUTES	Enumeration defining standard attributes used by FAT file systems
FILEIO_DRIVE_ERRORS	Possible results of the FSGetDiskProperties() function.
FILEIO_ERROR_TYPE	Enumeration for specific return codes
FILEIO_FILE_SYSTEM_TYPE	Enumeration of macros defining possible file system types supported by a device
FILEIO_FORMAT_MODE	Enumeration for formatting modes
FILEIO_OPEN_ACCESS_MODES	Enumeration for file access modes
FILEIO_RESULT	Enumeration for general purpose return values
FILEIO_SEEK_BASE	Enumeration defining base locations for seeking

Functions

	Name	Description
	FILEIO_MediaDetect	Determines if the given media is accessible.
	FILEIO_Initialize	Initialized the FILEIO library.
	FILEIO_Reinitialize	Reinitialized the FILEIO library.
	FILEIO_Flush	Saves unwritten file data to the device without closing the file.
	FILEIO_Close	Closes a file.
	FILEIO_GetChar	Reads a character from a file.
	FILEIO_PutChar	Writes a character to a file.
	FILEIO_Read	Reads data from a file.
	FILEIO_Write	Writes data to a file.
	FILEIO_Eof	Determines if the file's current read/write position is at the end of the file.
	FILEIO_Seek	Changes the current read/write position in the file.
	FILEIO_Tell	Returns the current read/write position in the file.
	FILEIO_DrivePropertiesGet	Allows user to get the drive properties (size of drive, free space, etc)

	FILEIO_LongFileNameGet	Obtains the long file name of a file found by the FILEIO_Find function.
	FILEIO_RegisterTimestampGet	Registers a FILEIO_TimestampGet function with the library.

Structures

Name	Description
FILEIO_TIMESTAMP	Structure to describe the time fields of a file
FILEIO_DRIVE_PROPERTIES	Structure that contains the disk search information, intermediate values, and results
FILEIO_OBJECT	Contains file information and is used to indicate which file to access.
FILEIO_SEARCH_RECORD	Search structure

Types

Name	Description
FILEIO_TimestampGet	Describes the user-implemented function to provide the timestamp.

Unions

Name	Description
FILEIO_TIME	Function to describe the FAT file system time.
FILEIO_DATE	Structure to describe a FAT file system date

Description

This section describes APIs that are common to both versions of the File I/O library.

1.7.1.3.1 Physical Layer Functions

Describes function pointer types used to define a physical layer.

Structures

Name	Description
FILEIO_DRIVE_CONFIG	Function pointer table that describes a drive being configured by the user

Types

Name	Description
FILEIO_DRIVER_IOInitialize	Function pointer prototype for a driver function to initialize I/O pins and modules for a driver.
FILEIO_DRIVER_MediaInitialize	Function pointer prototype for a driver function to perform media- specific initialization tasks.
FILEIO_DRIVER_MediaDeinitialize	Function pointer prototype for a driver function to deinitialize a media device.
FILEIO_DRIVER_MediaDetect	Function pointer prototype for a driver function to detect if a media device is attached/available.
FILEIO_DRIVER_SectorRead	Function pointer prototype for a driver function to read a sector of data from the device.
FILEIO_DRIVER_SectorWrite	Function pointer prototype for a driver function to write a sector of data to the device.
FILEIO_DRIVER_WriteProtectStateGet	Function pointer prototype for a driver function to determine if the device is write-protected.

Description

This section describes the functions that a physical layer must define in order to allow the File I/O layer to interface with it. A FILEIO_DRIVE_CONFIG structure containing pointers to functions that match these prototypes will be passed into the FILEIO_DriveMount function to initialize a physical layer.

1.7.1.3.1.1 FILEIO_DRIVE_CONFIG Structure

File

fileio_lfn.h

Syntax

```
typedef struct {
  FILEIO_DRIVER_IOInitialize funcIOInit;
  FILEIO_DRIVER_MediaDetect funcMediaDetect;
  FILEIO_DRIVER_MediaInitialize funcMediaInit;
  FILEIO_DRIVER_MediaDeinitialize funcMediaDeinit;
  FILEIO_DRIVER_SectorRead funcSectorRead;
  FILEIO_DRIVER_SectorWrite funcSectorWrite;
  FILEIO_DRIVER_WriteProtectStateGet funcWriteProtectGet;
} FILEIO_DRIVE_CONFIG;
```

Members

Members	Description
FILEIO_DRIVER_IOInitialize funcIOInit;	I/O Initialization function
FILEIO_DRIVER_MediaDetect funcMediaDetect;	Media Detection function
FILEIO_DRIVER_MediaInitialize funcMediaInit;	Media Initialization function
FILEIO_DRIVER_MediaDeinitialize funcMediaDeinit;	Media Deinitialization function.
FILEIO_DRIVER_SectorRead funcSectorRead;	Function to read a sector of the media.
FILEIO_DRIVER_SectorWrite funcSectorWrite;	Function to write a sector of the media.
FILEIO_DRIVER_WriteProtectStateGet funcWriteProtectGet;	Function to determine if the media is write-protected.

Description

Function pointer table that describes a drive being configured by the user

1.7.1.3.1.2 FILEIO_DRIVER_IOInitialize Type

Function pointer prototype for a driver function to initialize I/O pins and modules for a driver.

File

fileio_lfn.h

Syntax

```
typedef void (* FILEIO_DRIVER_IOInitialize)(void * mediaConfig);
```

Returns

None

Description

Function pointer prototype for a driver function to initialize I/O pins and modules for a driver.

Preconditions

None

Parameters

Parameters	Description
mediaConfig	Pointer to a driver-defined config structure

Function

void (*FILEIO_DRIVER_IOInitialize)(void * mediaConfig);

1.7.1.3.1.3 FILEIO_DRIVER_MediaInitialize Type

Function pointer prototype for a driver function to perform media- specific initialization tasks.

File

fileio_lfn.h

Syntax

```
typedef FILEIO_MEDIA_INFORMATION * (* FILEIO_DRIVER_MediaInitialize)(void * mediaConfig);
```

Returns

FILEIO_MEDIA_INFORMATION * - Pointer to a media initialization structure that has been loaded with initialization values.

Description

Function pointer prototype for a driver function to perform media- specific initialization tasks.

Preconditions

FILEIO_DRIVE_IOInitialize will be called first.

Parameters

Parameters	Description
mediaConfig	Pointer to a driver-defined config structure

Function

FILEIO_MEDIA_INFORMATION * (*FILEIO_DRIVER_MediaInitialize)(void * mediaConfig);

1.7.1.3.1.4 FILEIO_DRIVER_MediaDeinitialize Type

Function pointer prototype for a driver function to deinitialize a media device.

File

fileio_lfn.h

Syntax

```
typedef bool (* FILEIO_DRIVER_MediaDeinitialize)(void * mediaConfig);
```

Returns

If Success: true If Failure: false

Description

Function pointer prototype for a driver function to deinitialize a media device.

Preconditions

None

Parameters

Parameters	Description
mediaConfig	Pointer to a driver-defined config structure

Function

bool (*FILEIO_DRIVER_MediaDeinitialize)(void * mediaConfig);

1.7.1.3.1.5 FILEIO_DRIVER_MediaDetect Type

Function pointer prototype for a driver function to detect if a media device is attached/available.

File

fileio_lfn.h

Syntax

```
typedef bool (* FILEIO_DRIVER_MediaDetect)(void * mediaConfig);
```

Returns

If media attached: true If media not atached: false

Description

Function pointer prototype for a driver function to detect if a media device is attached/available.

Preconditions

None

Parameters

Parameters	Description
mediaConfig	Pointer to a driver-defined config structure

Function

bool (*FILEIO_DRIVER_MediaDetect)(void * mediaConfig);

1.7.1.3.1.6 FILEIO_DRIVER_SectorRead Type

Function pointer prototype for a driver function to read a sector of data from the device.

File

fileio_lfn.h

Syntax

```
typedef bool (* FILEIO_DRIVER_SectorRead)(void * mediaConfig, uint32_t sector_addr,
uint8_t* buffer);
```

Returns

If Success: true If Failure: false

Description

Function pointer prototype for a driver function to read a sector of data from the device.

Preconditions

The device will be initialized.

Parameters

Parameters	Description
mediaConfig	Pointer to a driver-defined config structure
sectorAddress	The address of the sector to read. This address format depends on the media.
buffer	A buffer to store the copied data sector.

Function

bool (*FILEIO_DRIVER_SectorRead)(void * mediaConfig,

uint32_t sector_addr, uint8_t * buffer);

1.7.1.3.1.7 FILEIO_DRIVER_SectorWrite Type

Function pointer prototype for a driver function to write a sector of data to the device.

File

fileio_lfn.h

Syntax

```
typedef uint8_t (* FILEIO_DRIVER_SectorWrite)(void * mediaConfig, uint32_t sector_addr,
uint8_t* buffer, bool allowWriteToZero);
```

Returns

If Success: true If Failure: false

Description

Function pointer prototype for a driver function to write a sector of data to the device.

Preconditions

The device will be initialized.

Parameters

Parameters	Description
mediaConfig	Pointer to a driver-defined config structure
sectorAddress	The address of the sector to write. This address format depends on the media.
buffer	A buffer containing the data to write.
allowWriteToZero	Check to prevent writing to the master boot record. This will always be false on calls that write to files, which will prevent a device from accidentally overwriting its own MBR if its root or FAT are corrupted. This should only be true if the user specifically tries to construct a new MBR.

Function

bool (*FILEIO_DRIVER_SectorWrite)(void * mediaConfig,

uint32_t sectorAddress, uint8_t * buffer, bool allowWriteToZero);

1.7.1.3.1.8 FILEIO_DRIVER_WriteProtectStateGet Type

Function pointer prototype for a driver function to determine if the device is write-protected.

File

fileio_lfn.h

Syntax

```
typedef bool (* FILEIO_DRIVER_WriteProtectStateGet)(void * mediaConfig);
```

Returns

If write-protected: true If not write-protected: false

Description

Function pointer prototype for a driver function to determine if the device is write-protected.

Preconditions

None

Parameters

Parameters	Description
mediaConfig	Pointer to a driver-defined config structure

Function

bool (*FILEIO_DRIVER_WriteProtectStateGet)(void * mediaConfig);

1.7.1.3.2 FILEIO_TIME Union

File

fileio_lfn.h

Syntax

```
typedef union {
  struct {
    uint16_t secondsDiv2 : 5;
    uint16_t minutes : 6;
    uint16_t hours : 5;
  } bitfield;
  uint16_t value;
} FILEIO_TIME;
```

Members

Members	Description
uint16_t secondsDiv2 : 5;	(Seconds / 2) (1-30)
uint16_t minutes : 6;	Minutes (1-60)
uint16_t hours : 5;	Hours (1-24)

Description

Function to describe the FAT file system time.

1.7.1.3.3 FILEIO_DATE Union

File

fileio_lfn.h

Syntax

```
typedef union {
  struct {
    uint16_t day : 5;
    uint16_t month : 4;
    uint16_t year : 7;
  } bitfield;
  uint16_t value;
} FILEIO_DATE;
```

Members

Members	Description
uint16_t day : 5;	Day (1-31)
uint16_t month : 4;	Month (1-12)
uint16_t year : 7;	Year (number of years since 1980)

Description

Structure to describe a FAT file system date

1.7.1.3.4 FILEIO_TIMESTAMP Structure

File

fileio_lfn.h

Syntax

```
typedef struct {
  FILEIO_DATE date;
  FILEIO_TIME time;
  uint8_t timeMs;
} FILEIO_TIMESTAMP;
```

Members

Members	Description
FILEIO_DATE date;	The create or write date of the file/directory.

FILEIO_TIME time;	The create of write time of the file/directory.
uint8_t timeMs;	The millisecond portion of the time.

Description

Structure to describe the time fields of a file

1.7.1.3.5 FILEIO_ATTRIBUTES Enumeration

File

fileio_lfn.h

Syntax

```
typedef enum {
  FILEIO_ATTRIBUTE_READ_ONLY = 0x01,
  FILEIO_ATTRIBUTE_HIDDEN = 0x02,
  FILEIO_ATTRIBUTE_SYSTEM = 0x04,
  FILEIO_ATTRIBUTE_VOLUME = 0x08,
  FILEIO_ATTRIBUTE_LONG_NAME = 0x0F,
  FILEIO_ATTRIBUTE_DIRECTORY = 0x10,
  FILEIO_ATTRIBUTE_ARCHIVE = 0x20,
  FILEIO_ATTRIBUTE_MASK = 0x3F
} FILEIO_ATTRIBUTES;
```

Members

Members	Description
FILEIO_ATTRIBUTE_READ_ONLY = 0x01	Read-only attribute. A file with this attribute should not be written to.
FILEIO_ATTRIBUTE_HIDDEN = 0x02	Hidden attribute. A file with this attribute may be hidden from the user.
FILEIO_ATTRIBUTE_SYSTEM = 0x04	System attribute. A file with this attribute is used by the operating system and should not be modified.
FILEIO_ATTRIBUTE_VOLUME = 0x08	Volume attribute. If the first file in the root directory of a volume has this attribute, the entry name is the volume name.
FILEIO_ATTRIBUTE_LONG_NAME = 0x0F	A file entry with this attribute mask is used to store part of the file's Long File Name.
FILEIO_ATTRIBUTE_DIRECTORY = 0x10	A file entry with this attribute points to a directory.
FILEIO_ATTRIBUTE_ARCHIVE = 0x20	Archive attribute. A file with this attribute should be archived.
FILEIO_ATTRIBUTE_MASK = 0x3F	Mask for all attributes.

Description

Enumeration defining standard attributes used by FAT file systems

1.7.1.3.6 FILEIO_DRIVE_ERRORS Enumeration

File

fileio_lfn.h

Syntax

```
typedef enum {
  FILEIO_GET_PROPERTIES_NO_ERRORS = 0,
  FILEIO_GET_PROPERTIES_CACHE_ERROR,
  FILEIO_GET_PROPERTIES_DRIVE_NOT_MOUNTED,
  FILEIO_GET_PROPERTIES_CLUSTER_FAILURE,
  FILEIO_GET_PROPERTIES_STILL_WORKING = 0xFF
} FILEIO_DRIVE_ERRORS;
```

Description

Possible results of the FSGetDiskProperties() function.

1.7.1.3.7 FILEIO_DRIVE_PROPERTIES Structure

File

fileio_lfn.h

Syntax

```
typedef struct {
  char disk;
  bool new_request;
  FILEIO_DRIVE_ERRORS properties_status;
  struct {
    uint8_t disk_format;
    uint16_t sector_size;
    uint8_t sectors_per_cluster;
    uint32_t total_clusters;
    uint32_t free_clusters;
  } results;
  struct {
    uint32_t c;
    uint32_t curcls;
    uint32_t EndClusterLimit;
    uint32_t ClusterFailValue;
  } private;
} FILEIO_DRIVE_PROPERTIES;
```

Members

Members	Description
char disk;	pointer to the disk we are searching
bool new_request;	is this a new request or a continued request
FILEIO_DRIVE_ERRORS properties_status;	status of the last call of the function
struct { uint8_t disk_format; uint16_t sector_size; uint8_t sectors_per_cluster; uint32_t total_clusters; uint32_t free_clusters; } results;	the results of the current search
uint8_t disk_format;	disk format: FAT12, FAT16, FAT32
uint16_t sector_size;	sector size of the drive
uint8_t sectors_per_cluster;	number of sectors per cluster
uint32_t total_clusters;	the number of total clusters on the drive
uint32_t free_clusters;	the number of free (unused) clusters on drive
struct { uint32_t c; uint32_t curcls; uint32_t EndClusterLimit; uint32_t ClusterFailValue; } private;	intermediate values used to continue searches. This member should be used only by the FSGetDiskProperties() function

Description

Structure that contains the disk search information, intermediate values, and results

1.7.1.3.8 FILEIO_ERROR_TYPE Enumeration

File

fileio_lfn.h

Syntax

```
typedef enum {
  FILEIO_ERROR_NONE = 0,
  FILEIO_ERROR_ERASE_FAIL,
  FILEIO_ERROR_NOT_PRESENT,
  FILEIO_ERROR_NOT_FORMATTED,
  FILEIO_ERROR_BAD_PARTITION,
  FILEIO_ERROR_UNSUPPORTED_FS,
  FILEIO_ERROR_INIT_ERROR,
  FILEIO_ERROR_UNINITIALIZED,
  FILEIO_ERROR_BAD_SECTOR_READ,
  FILEIO_ERROR_WRITE,
  FILEIO_ERROR_INVALID_CLUSTER,
  FILEIO_ERROR_DRIVE_NOT_FOUND,
  FILEIO_ERROR_FILE_NOT_FOUND,
  FILEIO_ERROR_DIR_NOT_FOUND,
  FILEIO_ERROR_BAD_FILE,
  FILEIO_ERROR_DONE,
  FILEIO_ERROR_COULD_NOT_GET_CLUSTER,
  FILEIO_ERROR_FILENAME_TOO_LONG,
  FILEIO_ERROR_FILENAME_EXISTS,
  FILEIO_ERROR_INVALID_FILENAME,
  FILEIO_ERROR_DELETE_DIR,
  FILEIO_ERROR_DELETE_FILE,
  FILEIO_ERROR_DIR_FULL,
  FILEIO_ERROR_DRIVE_FULL,
  FILEIO_ERROR_DIR_NOT_EMPTY,
  FILEIO_ERROR_UNSUPPORTED_SIZE,
  FILEIO_ERROR_WRITE_PROTECTED,
  FILEIO_ERROR_FILE_UNOPENED,
  FILEIO_ERROR_SEEK_ERROR,
  FILEIO_ERROR_BAD_CACHE_READ,
  FILEIO_ERROR_FAT32_UNSUPPORTED,
  FILEIO_ERROR_READ_ONLY,
  FILEIO_ERROR_WRITE_ONLY,
  FILEIO_ERROR_INVALID_ARGUMENT,
  FILEIO_ERROR_TOO_MANY_FILES_OPEN,
  FILEIO_ERROR_TOO_MANY_DRIVES_OPEN,
  FILEIO_ERROR_UNSUPPORTED_SECTOR_SIZE,
  FILEIO_ERROR_NO_LONG_FILE_NAME,
  FILEIO_ERROR_EOF
} FILEIO_ERROR_TYPE;
```

Members

Members	Description
FILEIO_ERROR_NONE = 0	No error
FILEIO_ERROR_ERASE_FAIL	An erase failed
FILEIO_ERROR_NOT_PRESENT	No device was present
FILEIO_ERROR_NOT_FORMATTED	The disk is of an unsupported format
FILEIO_ERROR_BAD_PARTITION	The boot record is bad
FILEIO_ERROR_UNSUPPORTED_FS	The file system type is unsupported
FILEIO_ERROR_INIT_ERROR	An initialization error has occured
FILEIO_ERROR_UNINITIALIZED	An operation was performed on an uninitialized device
FILEIO_ERROR_BAD_SECTOR_READ	A bad read of a sector occured
FILEIO_ERROR_WRITE	Could not write to a sector
FILEIO_ERROR_INVALID_CLUSTER	Invalid cluster value > maxcls
FILEIO_ERROR_DRIVE_NOT_FOUND	The specified drive could not be found
FILEIO_ERROR_FILE_NOT_FOUND	Could not find the file on the device
FILEIO_ERROR_DIR_NOT_FOUND	Could not find the directory
FILEIO_ERROR_BAD_FILE	File is corrupted
FILEIO_ERROR_DONE	No more files in this directory
FILEIO_ERROR_COULD_NOT_GET_CLUSTER	Could not load/allocate next cluster in file

FILEIO_ERROR_FILENAME_TOO_LONG	A specified file name is too long to use
FILEIO_ERROR_FILENAME_EXISTS	A specified filename already exists on the device
FILEIO_ERROR_INVALID_FILENAME	Invalid file name
FILEIO_ERROR_DELETE_DIR	The user tried to delete a directory with FILEIO_Remove
FILEIO_ERROR_DELETE_FILE	The user tried to delete a file with FILEIO_DirectoryRemove
FILEIO_ERROR_DIR_FULL	All root dir entry are taken
FILEIO_ERROR_DRIVE_FULL	All clusters in partition are taken
FILEIO_ERROR_DIR_NOT_EMPTY	This directory is not empty yet, remove files before deleting
FILEIO_ERROR_UNSUPPORTED_SIZE	The disk is too big to format as FAT16
FILEIO_ERROR_WRITE_PROTECTED	Card is write protected
FILEIO_ERROR_FILE_UNOPENED	File not opened for the write
FILEIO_ERROR_SEEK_ERROR	File location could not be changed successfully
FILEIO_ERROR_BAD_CACHE_READ	Bad cache read
FILEIO_ERROR_FAT32_UNSUPPORTED	FAT 32 - card not supported
FILEIO_ERROR_READ_ONLY	The file is read-only
FILEIO_ERROR_WRITE_ONLY	The file is write-only
FILEIO_ERROR_INVALID_ARGUMENT	Invalid argument
FILEIO_ERROR_TOO_MANY_FILES_OPEN	Too many files are already open
FILEIO_ERROR_TOO_MANY_DRIVES_OPEN	Too many drives are already open
FILEIO_ERROR_UNSUPPORTED_SECTOR_SIZE	Unsupported sector size
FILEIO_ERROR_NO_LONG_FILE_NAME	Long file name was not found
FILEIO_ERROR_EOF	End of file reached

Description

Enumeration for specific return codes

1.7.1.3.9 FILEIO_FILE_SYSTEM_TYPE Enumeration

File

fileio_lfn.h

Syntax

```
typedef enum {
  FILEIO_FILE_SYSTEM_TYPE_NONE = 0,
  FILEIO_FILE_SYSTEM_TYPE_FAT12,
  FILEIO_FILE_SYSTEM_TYPE_FAT16,
  FILEIO_FILE_SYSTEM_TYPE_FAT32
} FILEIO_FILE_SYSTEM_TYPE;
```

Members

Members	Description
FILEIO_FILE_SYSTEM_TYPE_NONE = 0	No file system
FILEIO_FILE_SYSTEM_TYPE_FAT12	The device is formatted with FAT12
FILEIO_FILE_SYSTEM_TYPE_FAT16	The device is formatted with FAT16
FILEIO_FILE_SYSTEM_TYPE_FAT32	The device is formatted with FAT32

Description

Enumeration of macros defining possible file system types supported by a device

1.7.1.3.10 FILEIO_FORMAT_MODE Enumeration

File

fileio_lfn.h

Syntax

```
typedef enum {
  FILEIO_FORMAT_ERASE = 0,
  FILEIO_FORMAT_BOOT_SECTOR
} FILEIO_FORMAT_MODE;
```

Members

Members	Description
FILEIO_FORMAT_ERASE = 0	Erases the contents of the partition
FILEIO_FORMAT_BOOT_SECTOR	Creates a boot sector based on user-specified information and erases any existing information

Description

Enumeration for formatting modes

1.7.1.3.11 FILEIO_OBJECT Structure

Contains file information and is used to indicate which file to access.

File

fileio_lfn.h

Syntax

```
typedef struct {
  uint32_t baseClusterDir;
  uint32_t currentClusterDir;
  uint32_t firstCluster;
  uint32_t currentCluster;
  uint32_t size;
  uint32_t absoluteOffset;
  void * disk;
  uint16_t * lfnPtr;
  uint16_t lfnLen;
  uint16_t currentSector;
  uint16_t currentOffset;
  uint16_t entry;
  uint16_t attributes;
  uint16_t time;
  uint16_t date;
  uint8_t timeMs;
  char name[FILEIO_FILE_NAME_LENGTH_8P3_NO_RADIX];
  struct {
    unsigned writeEnabled : 1;
    unsigned readEnabled : 1;
  } flags;
} FILEIO_OBJECT;
```

Members

Members	Description
uint32_t baseClusterDir;	The base cluster of the file's directory
uint32_t currentClusterDir;	The current cluster of the file's directory
uint32_t firstCluster;	The first cluster of the file
uint32_t currentCluster;	The current cluster of the file
uint32_t size;	The size of the file
uint32_t absoluteOffset;	The absolute offset in the file

void * disk;	Pointer to a device structure
uint16_t * lfnPtr;	Pointer to a LFN buffer
uint16_t lfnLen;	Length of the long file name
uint16_t currentSector;	The current sector in the current cluster of the file
uint16_t currentOffset;	The position in the current sector
uint16_t entry;	The position of the file's directory entry in its directory
uint16_t attributes;	The file's attributes
uint16_t time;	The file's last update time
uint16_t date;	The file's last update date
uint8_t timeMs;	The file's last update time (ms portion)
char name[FILEIO_FILE_NAME_LENGTH_8P3_NO_RADIX];	The short name of the file
unsigned writeEnabled : 1;	Indicates a file was opened in a mode that allows writes
unsigned readEnabled : 1;	Indicates a file was opened in a mode that allows reads

Description

The FILEIO_OBJECT structure is used to hold file information for an open file as it's being modified or accessed. A pointer to an open file's FILEIO_OBJECT structure will be passed to any library function that will modify that file.

1.7.1.3.12 FILEIO_OPEN_ACCESS_MODES Enumeration

File

fileio_lfn.h

Syntax

```
typedef enum {
  FILEIO_OPEN_READ = 0x01,
  FILEIO_OPEN_WRITE = 0x02,
  FILEIO_OPEN_CREATE = 0x04,
  FILEIO_OPEN_TRUNCATE = 0x08,
  FILEIO_OPEN_APPEND = 0x10
} FILEIO_OPEN_ACCESS_MODES;
```

Members

Members	Description
FILEIO_OPEN_READ = 0x01	Open the file for reading.
FILEIO_OPEN_WRITE = 0x02	Open the file for writing.
FILEIO_OPEN_CREATE = 0x04	Create the file if it doesn't exist.
FILEIO_OPEN_TRUNCATE = 0x08	Truncate the file to 0-length.
FILEIO_OPEN_APPEND = 0x10	Set the current read/write location in the file to the end of the file.

Description

Enumeration for file access modes

1.7.1.3.13 FILEIO_RESULT Enumeration

File

fileio_lfn.h

Syntax

```
typedef enum {
  FILEIO_RESULT_SUCCESS = 0,
  FILEIO_RESULT_FAILURE = -1
} FILEIO_RESULT;
```

Members

Members	Description
FILEIO_RESULT_SUCCESS = 0	File operation was a success
FILEIO_RESULT_FAILURE = -1	File operation failed

Description

Enumeration for general purpose return values

1.7.1.3.14 FILEIO_SEARCH_RECORD Structure

File

fileio_lfn.h

Syntax

```
typedef struct {
  uint8_t shortFileName[13];
  uint8_t attributes;
  uint32_t fileSize;
  FILEIO_TIMESTAMP timeStamp;
  uint32_t baseDirCluster;
  uint32_t currentDirCluster;
  uint16_t currentClusterOffset;
  uint16_t currentEntryOffset;
  uint16_t pathOffset;
  uint16_t driveId;
} FILEIO_SEARCH_RECORD;
```

Members

Members	Description
uint8_t shortFileName[13];	The name of the file that has been found (NULL-terminated).
uint8_t attributes;	The attributes of the file that has been found.
uint32_t fileSize;	The size of the file that has been found (bytes).
FILEIO_TIMESTAMP timeStamp;	The create or write time of the file that has been found.
uint32_t baseDirCluster;	Private Parameters

Description

Search structure

1.7.1.3.15 FILEIO_SEEK_BASE Enumeration

File

fileio_lfn.h

Syntax

```
typedef enum {
  FILEIO_SEEK_SET = 0,
  FILEIO_SEEK_CUR,
  FILEIO_SEEK_END
} FILEIO_SEEK_BASE;
```

Members

Members	Description
FILEIO_SEEK_SET = 0	Change the position in the file to an offset relative to the beginning of the file.
FILEIO_SEEK_CUR	Change the position in the file to an offset relative to the current location in the file.

FILEIO_SEEK_END	Change the position in the file to an offset relative to the end of the file.

Description

Enumeration defining base locations for seeking

1.7.1.3.16 FILEIO_MediaDetect Function

Determines if the given media is accessible.

File

fileio_lfn.h

Syntax

```
bool FILEIO_MediaDetect(const FILEIO_DRIVE_CONFIG * driveConfig, void * mediaParameters);
```

Returns

- If media is available : true
- If media is not available : false

Description

This function determines if a specified media device is available for further access.

Preconditions

FILEIO_Initialize must have been called. The driveConfig struct must have been initialized with the media-specific parameters and the FILEIO_DRIVER_MediaDetect function.

Parameters

Parameters	Description
const FILEIO_DRIVE_CONFIG * driveConfig	Constant structure containing function pointers that the library will use to access the drive.
void * mediaParameters	Pointer to the media-specific parameter structure

Function

bool FILEIO_MediaDetect (const FILEIO_DRIVE_CONFIG * driveConfig,

void * mediaParameters)

1.7.1.3.17 FILEIO_Initialize Function

Initialized the FILEIO library.

File

fileio_lfn.h

Syntax

```
int FILEIO_Initialize();
```

Returns

- If Success: FILEIO_RESULT_SUCCESS
- If Failure: FILEIO_RESULT_FAILURE

Description

Initializes the structures used by the FILEIO library.

Preconditions

None.

Function

int FILEIO_Initialize (void)

1.7.1.3.18 FILEIO_Reinitialize Function

Reinitialized the FILEIO library.

File

fileio_lfn.h

Syntax

```
int FILEIO_Reinitialize();
```

Returns

- If Success: FILEIO_RESULT_SUCCESS
- If Failure: FILEIO_RESULT_FAILURE

Description

Reinitialized the structures used by the FILEIO library.

Preconditions

FILEIO_Initialize must have been called.

Function

int FILEIO_Reinitialize (void)

1.7.1.3.19 FILEIO_Flush Function

Saves unwritten file data to the device without closing the file.

File

fileio_lfn.h

Syntax

```
int FILEIO_Flush(FILEIO_OBJECT * handle);
```

Returns

- If Success: FILEIO_RESULT_SUCCESS
- If Failure: FILEIO_RESULT_FAILURE

- Sets error code which can be retrieved with FILEIO_ErrorGet
  - FILEIO_ERROR_WRITE - Data could not be written to the device.
  - FILEIO_ERROR_BAD_CACHE_READ - The file's directory entry could not be cached.

Description

Saves unwritten file data to the device without closing the file. This function is useful if the user needs to continue writing to a file but also wants to ensure that data isn't lost in the event of a reset or power loss condition.

Preconditions

The drive containing the file must be mounted and the file handle must represent a valid, opened file.

Parameters

Parameters	Description
FILEIO_OBJECT * handle	The handle of the file to flush.

Function

int FILEIO_Flush (FILEIO_OBJECT * handle)

1.7.1.3.20 FILEIO_Close Function

Closes a file.

File

fileio_lfn.h

Syntax

```
int FILEIO_Close(FILEIO_OBJECT * handle);
```

Returns

- If Success: FILEIO_RESULT_SUCCESS
- If Failure: FILEIO_RESULT_FAILURE

- Sets error code which can be retrieved with FILEIO_ErrorGet
  - FILEIO_ERROR_WRITE - Data could not be written to the device.
  - FILEIO_ERROR_BAD_CACHE_READ - The file's directory entry could not be cached.

Description

Closes a file. This will save the unwritten data to the file and make the memory used to allocate a file available to open other files.

Preconditions

The drive containing the file must be mounted and the file handle must represent a valid, opened file.

Parameters

Parameters	Description
FILEIO_OBJECT * handle	The handle of the file to close.

Function

int FILEIO_Close (FILEIO_OBJECT * handle)

1.7.1.3.21 FILEIO_GetChar Function

Reads a character from a file.

File

fileio_lfn.h

Syntax

```
int FILEIO_GetChar(FILEIO_OBJECT * handle);
```

Returns

- If Success: The character that was read (cast to an int).
- If Failure: FILEIO_RESULT_FAILURE

- Sets error code which can be retrieved with FILEIO_ErrorGet
  - FILEIO_ERROR_WRITE_ONLY - The file is not opened in read mode.
  - FILEIO_ERROR_BAD_SECTOR_READ - There was an error reading the FAT to determine the next cluster in the file, or an error reading the file data.

- FILEIO_ERROR_INVALID_CLUSTER - The next cluster in the file is invalid.
- FILEIO_ERROR_EOF - There is no next cluster in the file (EOF)
- FILEIO_ERROR_WRITE - Cached data could not be written to the device.

Description

Reads a character from a file.

Preconditions

The drive containing the file must be mounted and the file handle must represent a valid, opened file.

Parameters

Parameters	Description
FILEIO_OBJECT * handle	The handle of the file.

Function

int FILEIO_GetChar (FILEIO_OBJECT * handle)

1.7.1.3.22 FILEIO_PutChar Function

Writes a character to a file.

File

fileio_lfn.h

Syntax

```
int FILEIO_PutChar(char c, FILEIO_OBJECT * handle);
```

Returns

- If Success: FILEIO_RESULT_SUCCESS
- If Failure: FILEIO_RESULT_FAILURE

- Sets error code which can be retrieved with FILEIO_ErrorGet
  - FILEIO_ERROR_READ_ONLY - The file was not opened in write mode.
  - FILEIO_ERROR_WRITE_PROTECTED - The media is write-protected.
  - FILEIO_ERROR_BAD_SECTOR_READ - There was an error reading the FAT to determine the next cluster in the file, or an error reading the file data.
  - FILEIO_ERROR_INVALID_CLUSTER - The next cluster in the file is invalid.
  - FILEIO_ERROR_WRITE - Cached data could not be written to the device.
  - FILEIO_ERROR_BAD_SECTOR_READ - File data could not be cached.
  - FILEIO_ERROR_DRIVE_FULL - There are no more clusters on the media that can be allocated to the file.

Description

Writes a character to a file.

Preconditions

The drive containing the file must be mounted and the file handle must represent a valid, opened file.

Parameters

Parameters	Description
char c	The character to write.
FILEIO_OBJECT * handle	The handle of the file.

Function

int FILEIO_PutChar (char c, FILEIO_OBJECT * handle)

1.7.1.3.23 FILEIO_Read Function

Reads data from a file.

File

fileio_lfn.h

Syntax

```
size_t FILEIO_Read(void * buffer, size_t size, size_t count, FILEIO_OBJECT * handle);
```

Returns

The number of data objects that were read. This value will match 'count' if the read was successful, or be less than count if it was not.

Sets error code which can be retrieved with FILEIO_ErrorGet:

- FILEIO_ERROR_WRITE_ONLY - The file is not opened in read mode.
- FILEIO_ERROR_BAD_SECTOR_READ - There was an error reading the FAT to determine the next cluster in the file, or an error reading the file data.
- FILEIO_ERROR_INVALID_CLUSTER - The next cluster in the file is invalid.
- FILEIO_ERROR_EOF - There is no next cluster in the file (EOF)
- FILEIO_ERROR_WRITE - Cached data could not be written to the device.

Description

Reads data from a file and stores it in 'buffer.'

Preconditions

The drive containing the file must be mounted and the file handle must represent a valid, opened file.

Parameters

Parameters	Description
void * buffer	The buffer that the data will be written to.
size_t size	The size of data objects to read, in bytes
size_t count	The number of data objects to read
FILEIO_OBJECT * handle	The handle of the file.

Function

size_t FILEIO_Read (void * buffer, size_t size, size_t count,

FILEIO_OBJECT * handle)

1.7.1.3.24 FILEIO_Write Function

Writes data to a file.

File

fileio_lfn.h

Syntax

```
size_t FILEIO_Write(const void * buffer, size_t size, size_t count, FILEIO_OBJECT * handle);
```

Returns

The number of data objects that were written. This value will match 'count' if the write was successful, or be less than count if

it was not.

Sets error code which can be retrieved with FILEIO_ErrorGet:

- FILEIO_ERROR_READ_ONLY - The file was not opened in write mode.
- FILEIO_ERROR_WRITE_PROTECTED - The media is write-protected.
- FILEIO_ERROR_BAD_SECTOR_READ - There was an error reading the FAT to determine the next cluster in the file, or an error reading the file data.
- FILEIO_ERROR_INVALID_CLUSTER - The next cluster in the file is invalid.
- FILEIO_ERROR_WRITE - Cached data could not be written to the device.
- FILEIO_ERROR_BAD_SECTOR_READ - File data could not be cached.
- FILEIO_ERROR_DRIVE_FULL - There are no more clusters on the media that can be allocated to the file.

Description

Writes data from 'buffer' to a file.

Preconditions

The drive containing the file must be mounted and the file handle must represent a valid, opened file.

Parameters

Parameters	Description
const void * buffer	The buffer that contains the data to write.
size_t size	The size of data objects to write, in bytes
size_t count	The number of data objects to write
FILEIO_OBJECT * handle	The handle of the file.

Function

size_t FILEIO_Write (void * buffer, size_t size, size_t count,

FILEIO_OBJECT * handle)

1.7.1.3.25 FILEIO_Eof Function

Determines if the file's current read/write position is at the end of the file.

File

fileio_lfn.h

Syntax

```
bool FILEIO_Eof(FILEIO_OBJECT * handle);
```

Returns

- If EOF: true
- If Not EOF: false

Description

Determines if the file's current read/write position is at the end of the file.

Preconditions

The drive containing the file must be mounted and the file handle must represent a valid, opened file.

Parameters

Parameters	Description
FILEIO_OBJECT * handle	The handle of the file.

Function

bool FILEIO_Eof (FILEIO_OBJECT * handle)

1.7.1.3.26 FILEIO_Seek Function

Changes the current read/write position in the file.

File

fileio_lfn.h

Syntax

```
int FILEIO_Seek(FILEIO_OBJECT * handle, int32_t offset, int base);
```

Returns

- If Success: FILEIO_RESULT_SUCCESS
- If Failure: FILEIO_RESULT_FAILURE

- Sets error code which can be retrieved with FILEIO_ErrorGet
  - FILEIO_ERROR_WRITE - Cached data could not be written to the device.
  - FILEIO_ERROR_INVALID_ARGUMENT - The specified location exceeds the file's size.
  - FILEIO_ERROR_BAD_SECTOR_READ - There was an error reading the FAT to determine the next cluster in the file, or an error reading the file data.
  - FILEIO_ERROR_INVALID_CLUSTER - The next cluster in the file is invalid.
  - FILEIO_ERROR_DRIVE_FULL - There are no more clusters on the media that can be allocated to the file. Clusters will be allocated to the file if the file is opened in a write mode and the user seeks to the end of a file that ends on a cluster boundary.
  - FILEIO_ERROR_COULD_NOT_GET_CLUSTER - There was an error finding the cluster that contained the specified offset.

Description

Changes the current read/write position in the file.

Preconditions

The drive containing the file must be mounted and the file handle must represent a valid, opened file.

Parameters

Parameters	Description
FILEIO_OBJECT * handle	The handle of the file.
int32_t offset	The offset of the new read/write position (in bytes) from the base location. The offset will be added to FILEIO_SEEK_SET or FILEIO_SEEK_CUR, or subtracted from FILEIO_SEEK_END.
int base	The base location. Is of the FILEIO_SEEK_BASE type.

Function

int FILEIO_Seek (FILEIO_OBJECT * handle, int32_t offset, int base)

1.7.1.3.27 FILEIO_Tell Function

Returns the current read/write position in the file.

File

fileio_lfn.h

Syntax

```
long FILEIO_Tell(FILEIO_OBJECT * handle);
```

Description

Returns the current read/write position in the file.

Offset of the current read/write position from the beginning of the file, in bytes.

Preconditions

The drive containing the file must be mounted and the file handle must represent a valid, opened file.

Parameters

Parameters	Description
FILEIO_OBJECT * handle	THe handle of the file.

Function

long FILEIO_Tell (FILEIO_OBJECT * handle)

1.7.1.3.28 FILEIO_DrivePropertiesGet Function

Allows user to get the drive properties (size of drive, free space, etc)

File

fileio.h

Syntax

```
void FILEIO_DrivePropertiesGet(FILEIO_DRIVE_PROPERTIES* properties, char driveId);
```

Side Effects

Can cause errors if called when files are open. Close all files before calling this function.

Calling this function without setting the new_request member on the first call can result in undefined behavior and results.

Calling this function after a result is returned other than FILEIO_GET_PROPERTIES_STILL_WORKING can result in undefined behavior and results.

Description

This function returns the information about the mounted drive. The results member of the properties object passed into the function is populated with the information about the drive.

Before starting a new request, the new_request member of the properties input parameter should be set to true. This will initiate a new search request.

This function will return before the search is complete with partial results. All of the results except the free_clusters will be correct after the first call. The free_clusters will contain the number of free clusters found up until that point, thus the free_clusters result will continue to grow until the entire drive is searched. If an application only needs to know that a certain number of bytes is available and doesn't need to know the total free size, then this function can be called until the required free size is verified. To continue a search, pass a pointer to the same FILEIO_FILEIO_DRIVE_PROPERTIES object that was passed in to create the search.

A new search request should be made once this function has returned a value other than FILEIO_GET_PROPERTIES_STILL_WORKING. Continuing a completed search can result in undefined behavior or results.

Typical Usage:

```
FILEIO_DRIVE_PROPERTIES disk_properties;

disk_properties.new_request = true;

do
{
```

```
    FILEIO_DiskPropertiesGet(&disk_properties, 'A');
} while (disk_properties.properties_status == FILEIO_GET_PROPERTIES_STILL_WORKING);
```

results.disk_format - contains the format of the drive. Valid results are FAT12(1), FAT16(2), or FAT32(3).

results.sector_size - the sector size of the mounted drive. Valid values are 512, 1024, 2048, and 4096.

results.sectors_per_cluster - the number sectors per cluster.

results.total_clusters - the number of total clusters on the drive. This can be used to calculate the total disk size (total_clusters * sectors_per_cluster * sector_size = total size of drive in bytes)

results.free_clusters - the number of free (unallocated) clusters on the drive. This can be used to calculate the total free disk size (free_clusters * sectors_per_cluster * sector_size = total size of drive in bytes)

Remarks

PIC24F size estimates: Flash - 400 bytes (-Os setting)

PIC24F speed estimates: Search takes approximately 7 seconds per Gigabyte of drive space. Speed will vary based on the number of sectors per cluster and the sector size.

Preconditions

1) ALLOW_GET_FILEIO_DRIVE_PROPERTIES must be defined in FSconfig.h 2) a FS_FILEIO_DRIVE_PROPERTIES object must be created before the function is called 3) the new_request member of the FS_FILEIO_DRIVE_PROPERTIES object must be set before calling the function for the first time. This will start a new search. 4) this function should not be called while there is a file open. Close all files before calling this function.

Parameters

Parameters	Description
FILEIO_DRIVE_PROPERTIES* properties	a pointer to a FS_FILEIO_DRIVE_PROPERTIES object where the results should be stored.

Return Values

Return Values	Description
the following possible values	
FILEIO_GET_PROPERTIES_NO_ERRORS	operation completed without error. Results are in the properties object passed into the function.
FILEIO_GET_PROPERTIES_DRIVE_NOT_MOUNTED	there is no mounted disk. Results in properties object is not valid
FILEIO_GET_PROPERTIES_CLUSTER_FAILURE	there was a failure trying to read a cluster from the drive. The results in the properties object is a partial result up until the point of the failure.
FILEIO_GET_PROPERTIES_STILL_WORKING	the search for free sectors is still in process. Continue calling this function with the same properties pointer until either the function completes or until the partial results meets the application needs. The properties object contains the partial results of the search and can be used by the application.

Function

void FILEIO_DrivePropertiesGet()

1.7.1.3.29 FILEIO_LongFileNameGet Function

Obtains the long file name of a file found by the FILEIO_Find function.

File

fileio_lfn.h

Syntax

```
int FILEIO_LongFileNameGet(FILEIO_SEARCH_RECORD * record, uint16_t * buffer, uint16_t
```

```
length);
```

Returns

- If Success: FILEIO_RESULT_SUCCESS
- If Failure: FILEIO_RESULT_FAILURE

- Sets error code which can be retrieved with FILEIO_ErrorGet Note that if the path cannot be resolved, the error will be returned for the current working directory.
  - FILEIO_ERROR_INVALID_ARGUMENT - The path could not be resolved.
  - FILEIO_ERROR_NO_LONG_FILE_NAME - The short file name does not have an associated long file name.
  - FILEIO_ERROR_DONE - The directory entry could not be cached because the entryOffset contained in record was invalid.
  - FILEIO_ERROR_WRITE - Cached data could not be written to the device.
  - FILEIO_ERROR_BAD_SECTOR_READ - The directory entry could not be cached because there was an error reading from the device.

Description

This function will obtain the long file name of a file found by the FILEIO_Find function and copy it into a user-specified buffer. The name will be returned in unicode characters.

Preconditions

A drive must have been mounted by the FILEIO library. The FILEIO_SEARCH_RECORD structure must contain valid file information obtained from the FILEIO_Find function.

Parameters

Parameters	Description
FILEIO_SEARCH_RECORD * record	The file record obtained from a successful call of FILEIO_Find.
uint16_t * buffer	A buffer to contain the long file name of the file.
uint16_t length	The length of the buffer, in 16-bit words.

Function

int FILEIO_LongFileNameGet (FILEIO_SEARCH_RECORD * record, uint16_t * buffer, uint16_t length)

1.7.1.3.30 FILEIO_TimestampGet Type

Describes the user-implemented function to provide the timestamp.

File

fileio_lfn.h

Syntax

```
typedef void (* FILEIO_TimestampGet)(FILEIO_TIMESTAMP *);
```

Returns

void

Description

Files in a FAT files system use time values to track create time, access time, and last-modified time. In the FILEIO library, the user must implement a function that the library can call to obtain the current time. That function will have this format.

Preconditions

N/A.

Function

typedef void (*FILEIO_TimestampGet)(FILEIO_TIMESTAMP *)

1.7.1.3.31 FILEIO_RegisterTimestampGet Function

Registers a FILEIO_TimestampGet function with the library.

File

fileio_lfn.h

Syntax

```
void FILEIO_RegisterTimestampGet(FILEIO_TimestampGet timestampFunction);
```

Returns

void

Description

The user must call this function to specify which user-implemented function will be called by the library to generate timestamps.

Preconditions

FILEIO_Initialize must have been called.

Parameters

Parameters	Description
FILEIO_TimestampGet timestampFunction	A pointer to the user-implemented function that will provide timestamps to the library.

Function

void FILEIO_RegisterTimestampGet (FILEIO_TimestampGet timestampFunction)

1.7.2 Physical Layer

Describes the API of the physical layers used by the library.

Modules

Name	Description
SD (SPI) Driver	Describes the SD-SPI physical layer.

Description

This section describes the API of the physical layers used by the library.

1.7.2.1 SD (SPI) Driver

Describes the SD-SPI physical layer.

Functions

	Name	Description
	FILEIO_SD_AsyncReadTasks	This is function FILEIO_SD_AsyncReadTasks.
	FILEIO_SD_AsyncWriteTasks	This is function FILEIO_SD_AsyncWriteTasks.
	FILEIO_SD_IOInitialize	Initializes the I/O lines connected to the card

	FILEIO_SD_MediaDetect	Determines whether an SD card is present
	FILEIO_SD_MediaInitialize	Initializes the SD card.
	FILEIO_SD_MediaDeinitialize	Disables the SD card
	FILEIO_SD_CapacityRead	Determines the current capacity of the SD card
	FILEIO_SD_SectorSizeRead	Determines the current sector size on the SD card
	FILEIO_SD_SectorRead	Reads a sector of data from an SD card.
	FILEIO_SD_SectorWrite	Writes a sector of data to an SD card.
	FILEIO_SD_WriteProtectStateGet	Indicates whether the card is write-protected.

Description

This section describes the SD-SPI physical layer. This module allows access to SD and MMC cards via SPI.

A pointer to a FILEIO_SD_DRIVE_CONFIG structure should be used as the mediaParameters element in the FILEIO_DRIVE_CONFIG structure describing this type of media.

1.7.2.1.1 FILEIO_SD_AsyncReadTasks Function

File

sd_spi.h

Syntax

```
uint8_t FILEIO_SD_AsyncReadTasks(FILEIO_SD_DRIVE_CONFIG * config, FILEIO_SD_ASYNC_IO*);
```

Module

SD (SPI) Driver

Description

This is function FILEIO_SD_AsyncReadTasks.

1.7.2.1.2 User-Implemented Functions

Describes functions that must be implemented by the user.

Module

SD (SPI) Driver

Structures

Name	Description
FILEIO_SD_DRIVE_CONFIG	A configuration structure used by the SD-SPI driver functions to perform specific tasks.

Types

Name	Description
FILEIO_SD_CSSet	Prototype for a user-implemented function to set or clear the SPI's chip select pin.
FILEIO_SD_CDGet	Prototype for a user-implemented function to get the current state of the Card Detect pin, if one exists.
FILEIO_SD_WPGet	Prototype for a user-implemented function to get the current state of the Write Protect pin, if one exists.
FILEIO_SD_PinConfigure	Prototype for a user-implemented function to configure the pins used by the SD card.

Description

This section describes functions that must be implemented by the user for the FILEIO_SD_DRIVE_CONFIG structure used to initialize a FILEIO_DRIVE_CONFIG mediaParameters element.

1.7.2.1.2.1 FILEIO_SD_DRIVE_CONFIG Structure

File

sd_spi.h

Syntax

```
typedef struct {
  uint8_t index;
  FILEIO_SD_CSSet csFunc;
  FILEIO_SD_CDGet cdFunc;
  FILEIO_SD_WPGet wpFunc;
  FILEIO_SD_PinConfigure configurePins;
} FILEIO_SD_DRIVE_CONFIG;
```

Members

Members	Description
uint8_t index;	The numeric index of the SPI module to use (i.e. 1 for SPI1/SSP1, 2 for SPI2, SSP2,...)
FILEIO_SD_CSSet csFunc;	Pointer to a user-implemented function to set/clear the chip select pins
FILEIO_SD_CDGet cdFunc;	Pointer to a user-implemented function to get the status of the card detect pin
FILEIO_SD_WPGet wpFunc;	Pointer to a user-implemented function to get the status of the write protect pin
FILEIO_SD_PinConfigure configurePins;	Pointer to a user-implemented function to configure the pins used by the SD Card

Description

A configuration structure used by the SD-SPI driver functions to perform specific tasks.

1.7.2.1.2.2 FILEIO_SD_CSSet Type

Prototype for a user-implemented function to set or clear the SPI's chip select pin.

File

sd_spi.h

Syntax

```
typedef void (* FILEIO_SD_CSSet)(uint8_t value);
```

Description

Most functions in this driver require the user to implement the functions that comprise a FILEIO_SD_DRIVE_CONFIG structure. This function pointer definition describes a function in this structure that will set/clear the chip select pin.

Remarks

None

Parameters

Parameters	Description
value	The value of the chip select pin (1 or 0)

Function

typedef void (*FILEIO_SD_CSSet)(uint8_t value)

1.7.2.1.2.3 FILEIO_SD_CDGet Type

Prototype for a user-implemented function to get the current state of the Card Detect pin, if one exists.

File

sd_spi.h

Syntax

```
typedef bool (* FILEIO_SD_CDGet)(void);
```

Description

Most functions in this driver require the user to implement the functions that comprise a FILEIO_SD_DRIVE_CONFIG structure. This function pointer definition describes a function in this structure that will return the value of a card detect pin. These pins are a typical feature on the physical sockets manufactured for SD card (not on the SD cards themselves). On some types of SD card (i.e. micro SD) this pin will not be available.

Remarks

None

Function

typedef bool (*FILEIO_SD_CDGet)(void);

1.7.2.1.2.4 FILEIO_SD_WPGet Type

Prototype for a user-implemented function to get the current state of the Write Protect pin, if one exists.

File

sd_spi.h

Syntax

```
typedef bool (* FILEIO_SD_WPGet)(void);
```

Description

Most functions in this driver require the user to implement the functions that comprise a FILEIO_SD_DRIVE_CONFIG structure. This function pointer definition describes a function in this structure that will return the value of a write protect pin. These pins are a typical feature on the physical sockets manufactured for SD card (not on the SD cards themselves). On some types of SD card (i.e. micro SD) this pin will not be available.

Remarks

None

Function

typedef bool (*FILEIO_SD_WPGet)(void);

1.7.2.1.2.5 FILEIO_SD_PinConfigure Type

Prototype for a user-implemented function to configure the pins used by the SD card.

File

sd_spi.h

Syntax

```
typedef void (* FILEIO_SD_PinConfigure)(void);
```

Description

Most functions in this driver require the user to implement the functions that comprise a FILEIO_SD_DRIVE_CONFIG structure. This function pointer definition describes a function in this structure that will configure all of the pins used by the SD Card. The configuration may involve setting/clearing the TRIS bits, disabling the analog state of the pins, setting up peripheral pin select, or other operations (depending on the device). The user must configure the chip select, card detect,

and write protect pins. Optionally, configuration for the SPI pins (SDI, SDO, SCK) and SPI module may be performed in this function, though it may make more sense to configure those in another part of any given application.

Remarks

None

Function

typedef void (*FILEIO_SD_PinConfigure)(void);

1.7.2.1.3 FILEIO_SD_AsyncWriteTasks Function

File

sd_spi.h

Syntax

```
uint8_t FILEIO_SD_AsyncWriteTasks(FILEIO_SD_DRIVE_CONFIG * config, FILEIO_SD_ASYNC_IO*);
```

Module

SD (SPI) Driver

Description

This is function FILEIO_SD_AsyncWriteTasks.

1.7.2.1.4 FILEIO_SD_IOInitialize Function

Initializes the I/O lines connected to the card

File

sd_spi.h

Syntax

```
void FILEIO_SD_IOInitialize(FILEIO_SD_DRIVE_CONFIG * config);
```

Module

SD (SPI) Driver

Side Effects

None.

Returns

None

Description

The FILEIO_SD_IOInitialize function initializes the I/O pins connected to the SD card.

Remarks

None

Preconditions

FILEIO_SD_MediaInitialize() is complete. The MDD_InitIO function pointer is pointing to this function.

Parameters

Parameters	Description
FILEIO_SD_DRIVE_CONFIG * config	An SD Drive configuration structure pointer

Function

void FILEIO_SD_IOInitialize (

FILEIO_SD_DRIVE_CONFIG * config)

1.7.2.1.5 FILEIO_SD_MediaDetect Function

Determines whether an SD card is present

File

sd_spi.h

Syntax

```
bool FILEIO_SD_MediaDetect(FILEIO_SD_DRIVE_CONFIG * config);
```

Module

SD (SPI) Driver

Side Effects

None.

Description

The FILEIO_SD_MediaDetect function determine if an SD card is connected to the microcontroller. If the MEDIA_SOFT_DETECT is not defined, the detection is done by polling the SD card detect pin. The MicroSD connector does not have a card detect pin, and therefore a software mechanism must be used. To do this, the SEND_STATUS command is sent to the card. If the card is not answering with 0x00, the card is either not present, not configured, or in an error state. If this is the case, we try to reconfigure the card. If the configuration fails, we consider the card not present (it still may be present, but malfunctioning). In order to use the software card detect mechanism, the MEDIA_SOFT_DETECT macro must be defined.

Remarks

None

Preconditions

The FILEIO_SD_MediaDetect function pointer must be configured to point to this function in FSconfig.h

Parameters

Parameters	Description
FILEIO_SD_DRIVE_CONFIG * config	The given drive configuration

Return Values

Return Values	Description
true	Card detected
false	No card detected

Function

bool FILEIO_SD_MediaDetect (FILEIO_SD_DRIVE_CONFIG * config)

1.7.2.1.6 FILEIO_SD_MediaInitialize Function

Initializes the SD card.

File

sd_spi.h

Syntax

```
FILEIO_MEDIA_INFORMATION * FILEIO_SD_MediaInitialize(FILEIO_SD_DRIVE_CONFIG * config);
```

Module

SD (SPI) Driver

Side Effects

None.

Description

This function will send initialization commands to and SD card.

Remarks

Psuedo code flow for the media initialization process is as follows:

SD Card SPI Initialization Sequence (for physical layer v1.x or v2.0 device) is as follows:

---

0. Power up tasks a. Initialize microcontroller SPI module to no more than 400kbps rate so as to support MMC devices. b. Add delay for SD card power up, prior to sending it any commands. It wants the longer of: 1ms, the Vdd ramp time (time from 2.7V to Vdd stable), and 74+ clock pulses.
1. Send CMD0 (GO_IDLE_STATE) with CS = 0. This puts the media in SPI mode and software resets the SD/MMC card.
2. Send CMD8 (SEND_IF_COND). This requests what voltage the card wants to run at.

Some cards will not support this command. a. If illegal command response is received, this implies either a v1.x physical spec device, or not an SD card (ex: MMC). b. If normal response is received, then it must be a v2.0 or later SD memory card.

If v1.x device:

---

3. Send CMD1 repeatedly, until initialization complete (indicated by R1 response uint8_t/idle bit == 0)
4. Basic initialization is complete. May now switch to higher SPI frequencies.
5. Send CMD9 to read the CSD structure. This will tell us the total flash size and other info which will be useful later.
6. Parse CSD structure bits (based on v1.x structure format) and extract useful information about the media.
7. The card is now ready to perform application data transfers.

If v2.0+ device:

---

3. Verify the voltage range is feasible. If not, unusable card, should notify user that the card is incompatible with this host.
4. Send CMD58 (Read OCR).
5. Send CMD55, then ACMD41 (SD_SEND_OP_COND, with HCS = 1). a. Loop CMD55/ACMD41 until R1 response uint8_t == 0x00 (indicating the card is no longer busy/no longer in idle state).
6. Send CMD58 (Get CCS). a. If CCS = 1 --> SDHC card. b. If CCS = 0 --> Standard capacity SD card (which is v2.0+).
7. Basic initialization is complete. May now switch to higher SPI frequencies.
8. Send CMD9 to read the CSD structure. This will tell us the total flash size and other info which will be useful later.
9. Parse CSD structure bits (based on v2.0 structure format) and extract useful information about the media.
10. The card is now ready to perform application data transfers.

Preconditions

The FILEIO_SD_MediaInitialize function pointer must be pointing to this function.

Parameters

Parameters	Description
FILEIO_SD_DRIVE_CONFIG * config	An SD Drive configuration structure pointer

Return Values

Return Values	Description
errorCode member may contain the following values	• MEDIA_NO_ERROR - The media initialized successfully • MEDIA_CANNOT_INITIALIZE - Cannot initialize the media.

Function

FILEIO_MEDIA_INFORMATION * FILEIO_SD_MediaInitialize (void)

1.7.2.1.7 FILEIO_SD_MediaDeinitialize Function

Disables the SD card

File

sd_spi.h

Syntax

```
bool FILEIO_SD_MediaDeinitialize(FILEIO_SD_DRIVE_CONFIG * config);
```

Module

SD (SPI) Driver

Side Effects

None.

Returns

true if successful, false otherwise

Description

This function will disable the SPI port and deselect the SD card.

Remarks

None

Preconditions

The FILEIO_SD_MediaDeinitialize function pointer is pointing towards this function.

Parameters

Parameters	Description
FILEIO_SD_DRIVE_CONFIG * config	An SD Drive configuration structure pointer

Function

bool FILEIO_SD_MediaDeinitialize(

FILEIO_SD_DRIVE_CONFIG * config)

1.7.2.1.8 FILEIO_SD_CapacityRead Function

Determines the current capacity of the SD card

File

sd_spi.h

Syntax

```
uint32_t FILEIO_SD_CapacityRead(FILEIO_SD_DRIVE_CONFIG * config);
```

Module

SD (SPI) Driver

Side Effects

None.

Returns

The capacity of the device

Description

The FILEIO_SD_CapacityRead function is used by the USB mass storage class to return the total number of sectors on the card.

Remarks

None

Preconditions

FILEIO_SD_MediaInitialize() is complete

Parameters

Parameters	Description
FILEIO_SD_DRIVE_CONFIG * config	An SD Drive configuration structure pointer

Function

uint32_t FILEIO_SD_CapacityRead(

FILEIO_SD_DRIVE_CONFIG * config)

1.7.2.1.9 FILEIO_SD_SectorSizeRead Function

Determines the current sector size on the SD card

File

sd_spi.h

Syntax

```
uint16_t FILEIO_SD_SectorSizeRead(FILEIO_SD_DRIVE_CONFIG * config);
```

Module

SD (SPI) Driver

Side Effects

None.

Returns

The size of the sectors for the physical media

Description

The FILEIO_SD_SectorSizeRead function is used by the USB mass storage class to return the card's sector size to the PC on request.

Remarks

None

Preconditions

FILEIO_SD_MediaInitialize() is complete

Parameters

Parameters	Description
FILEIO_SD_DRIVE_CONFIG * config	An SD Drive configuration structure pointer

Function

uint16_t FILEIO_SD_SectorSizeRead(

FILEIO_SD_DRIVE_CONFIG * config)

1.7.2.1.10 FILEIO_SD_SectorRead Function

Reads a sector of data from an SD card.

File

sd_spi.h

Syntax

```
bool FILEIO_SD_SectorRead(FILEIO_SD_DRIVE_CONFIG * config, uint32_t sector_addr, uint8_t * buffer);
```

Module

SD (SPI) Driver

Side Effects

None

Description

The FILEIO_SD_SectorRead function reads a sector of data uint8_ts (512 uint8_ts) of data from the SD card starting at the sector address and stores them in the location pointed to by 'buffer.'

Remarks

The card expects the address field in the command packet to be a uint8_t address. The sector_addr value is converted to a uint8_t address by shifting it left nine times (multiplying by 512).

This function performs a synchronous read operation. In other uint16_ts, this function is a blocking function, and will not return until either the data has fully been read, or, a timeout or other error occurred.

Preconditions

The FILEIO_SD_SectorRead function pointer must be pointing towards this function.

Parameters

Parameters	Description
FILEIO_SD_DRIVE_CONFIG * config	An SD Drive configuration structure pointer
uint8_t * buffer	The buffer where the retrieved data will be stored. If buffer is NULL, do not store the data anywhere.
sectorAddress	The address of the sector on the card.

Return Values

Return Values	Description
true	The sector was read successfully
false	The sector could not be read

Function

uint8_t FILEIO_SD_SectorRead (uint32_t sector_addr, uint8_t * buffer)

1.7.2.1.11 FILEIO_SD_SectorWrite Function

Writes a sector of data to an SD card.

File

sd_spi.h

Syntax

```
bool FILEIO_SD_SectorWrite(FILEIO_SD_DRIVE_CONFIG * config, uint32_t sector_addr, uint8_t *
buffer, bool allowWriteToZero);
```

Module

SD (SPI) Driver

Side Effects

None.

Description

The FILEIO_SD_SectorWrite function writes one sector of data (512 uint8_ts) of data from the location pointed to by 'buffer' to the specified sector of the SD card.

Remarks

The card expects the address field in the command packet to be a uint8_t address. The sector_addr value is converted to a uint8_t address by shifting it left nine times (multiplying by 512).

Preconditions

The FILEIO_SD_SectorWrite function pointer must be pointing to this function.

Parameters

Parameters	Description
FILEIO_SD_DRIVE_CONFIG * config	An SD Drive configuration structure pointer
uint8_t * buffer	The buffer with the data to write.
bool allowWriteToZero	• true - Writes to the 0 sector (MBR) are allowed • false - Any write to the 0 sector will fail.
sectorAddress	The address of the sector on the card.

Return Values

Return Values	Description
true	The sector was written successfully.
false	The sector could not be written.

Function

bool FILEIO_SD_SectorWrite (FILEIO_SD_DRIVE_CONFIG * config,

uint32_t sector_addr, uint8_t * buffer, uint8_t allowWriteToZero)

1.7.2.1.12 FILEIO_SD_WriteProtectStateGet Function

Indicates whether the card is write-protected.

File

sd_spi.h

Syntax

```
bool FILEIO_SD_WriteProtectStateGet(FILEIO_SD_DRIVE_CONFIG * config);
```

Module

SD (SPI) Driver

Side Effects

None.

Description

The FILEIO_SD_WriteProtectStateGet function will determine if the SD card is write protected by checking the electrical signal that corresponds to the physical write-protect switch.

Remarks

None

Preconditions

The FILEIO_SD_WriteProtectStateGet function pointer must be pointing to this function.

Parameters

Parameters	Description
FILEIO_SD_DRIVE_CONFIG * config	An SD Drive configuration structure pointer

Return Values

Return Values	Description
true	The card is write-protected
false	The card is not write-protected

Function

uint8_t FILEIO_SD_WriteProtectStateGet

1.8 Migration

Describes migration from the MDD File System Interface Library.

Description

Older versions of Microchip's software releases have included a FAT file system library called the MDD File System Interface Library. For various reasons (functionality, code size, execution speed) you may wish to migrate from the MDDFS library to this library. This topic will provide information to make this transition easier.

1.8.1 Initialization

Describes changes in initialization routines between the File I/O library and the MDD library.

Description

Because the File I/O library supports multiple drives, the method for initializing it has changed. To begin initializing the File I/O library, the user must first call FILEIO_Initialize. This will initialize the library's structures in the same way that FSInit did for the MDD library. Unlike FSInit, FILEIO_Initialize will not initialize the media accessed by the library,

In the MDD library, physical media access functions were tied to the library by definitions in a header file. In the File I/O library, this information is provided to the library at run time to allow the library to access multiple devices dynamically. To specify how to access a media device, the user will pass a pointer to a FILEIO_DRIVE_CONFIG structure and a pointer to a structure containing media-specific parameters into the FILEIO_DriveMount function. These structures contain function pointers to the functions that will allow the File I/O library to access the media. In most cases, the functions in the FILEIO_DRIVE_CONFIG structure functions will be implemented in the media layer and the media-specific parameter functions must be implemented by the user, if they are required. For more information, see the How the Library Works topic.

1.8.2 API Differences

Describes differences in the API between libraries.

Description

There are several differences between the File I/O and MDD API. The following table describes these differences.

File I/O Library API	Nearest MDD API	Notable Differences
FILEIO_MediaDetect	-	This API provides a middleware-level interface to the media detect function.
FILEIO_Initialize, FILEIO_Reinitialize, FILEIO_DriveMount	FSInit	Since the File I/O library supports multiple physical layers, the drive mounting functionality was separated from the library initialization functionality.
FILEIO_DriveUnmount	-	
FILEIO_Open	FSfopen	FILEIO_Open accepts full paths as arguments. Instead of an ASCII mode string, it now accepts a logical OR of mode parameters. File objects are now allocated by the user instead of the library and are passed in as arguments. This function will now return FILEIO_RESULT_SUCCESS/FAILURE instead of a file pointer or NULL.

FILEIO_Flush	-	
FILEIO_Close	FSfclose	This function now returns FILEIO_RESULT_SUCCESS/FAILURE instead of 0/EOF.
FILEIO_GetChar	-	
FILEIO_PutChar	-	
FILEIO_Read	FSfread	
FILEIO_Write	FSfwrite	
FILEIO_Eof	FSfeof	FILEIO_Eof returns 'true' and 'false' instead of 0 and !0.
FILEIO_Seek	FSfseek	This function returns FILEIO_RESULT_SUCCESS/FAILURE instead of 0/-1.
FILEIO_Tell	FSftell	
FILEIO_DrivePropertiesGet	FSGetDiskProperties	The name of the drive properties structure has changes to FILEIO_DRIVE_PROPERTIES. This function accepts the drive ID as a second argument.
FILEIO_LongFileNameGet	-	
FILEIO_Remove	FSremove	This function now accepts full path strings as an argument. The return value of this function is FILEIO_RESULT_SUCCESS/FAILURE instead of 0/EOF.
FILEIO_Rename	FSrename	This function now accepts a file path and a file name instead of a pointer to an open file and a file name. The return values are FILEIO_RESULT_SUCCESS/FAILURE instead of 0/EOF.
FILEIO_Find	FindFirst, FindNext	The MDD find functions are now represented by a single function. The name of the SearchRec structure has changed to FILEIO_SEARCH_RECORD. The user now specifies whether a new search should be conducted with a boolean function argument. FILEIO_Find now accepts full path names instead of simple file names. The return values have changed to FILEIO_RESULT_SUCCESS/FAILURE.
FILEIO_DirectoryMake	FSmkdir	The return values have changed to FILEIO_RESULT_SUCCESS/FAILURE.
FILEIO_DirectoryChange	FSchdir	The return values have changed to FILEIO_RESULT_SUCCESS/FAILURE.
FILEIO_DirectoryRemove	FSrmdir	The return values have changed to FILEIO_RESULT_SUCCESS/FAILURE. This function can no longer remove subdirectories and files within the deleted directory automatically.
FILEIO_DirectoryGetCurrent	FSgetcwd	This function will no longer return a pointer to a 10-byte buffer if the user-specified buffer is NULL.
FILEIO_ErrorClear	-	
FILEIO_ErrorGet	FSerror	Several error types have changed. See the FILEIO_ERROR_TYPE enumeration for more information.
FILEIO_FileSystemTypeGet	-	
FILEIO_RegisterTimestampGet	-	

Index