

# Release Notes for Microchip Memory Disk Drive File System

Version 1.2.4

November 18th, 2009

## 1. Description

This library is intended to provide an interface to file systems compatible with ISO/IEC specification 9293 (commonly referred to as FAT12 and FAT16). The FAT32 file system is also supported, although long file names are not implemented. This library includes four different physical interface files: one for SecureDigital card interface using the SPI module, one for CompactFlash card interface using manual bit toggling, one for CompactFlash card interface using the Parallel Master Port module included on several PIC24/PIC32 microcontrollers, and one template interface file that can be modified by the user to create a custom interface layer to an unsupported device.. In addition, Microchip's USB Host stack (available from [www.microchip.com/usb](http://www.microchip.com/usb) can be used as a physical layer.

## 2. Changes In This Release

### From version 1.2.3

- a. Add a software mechanism for the card detection to be used when the connectors does not provide a card detect signal (e.g. MicroSD connectors).
- b. Optimize some code by collapsing duplicated code into a single for loop.
- c. Update to use the CSD register data to calculate the SD card capacity.
- d. Fixed compatibility problems in SD-SPI.c that was preventing it from working with some of the cards.
- e. Make modifications to allow dynamic sector size giving the possibility to use thumb drives with different sector sizes.
- f. Corrected a bug that prevented the last two clusters of a drive to be able to be written to.
- g. Add the GetDiskProperties function to allow the users to get the disk properties (size of disk, free space, etc)

### From version 1.2.2

- a. Added FSGetDiskProperties() function. This function gets the remaining disk space as well as other properties such as sector size, clusters per sector, partition format, etc.
- b. Fixed bug that prevented the last two clusters in a partition from being allocated.
- c. Added SDHC support for the MDD file system.
- d. Fixed the problem where on an append to a file that is smaller than a sector.
- e. Fixed TODO that limited internal flash to PSV page boundaries in on PIC24 and only PSVPAG = 1.
- f. Fixed an issue with the FSattrib function that was changing the wrong cached file object.

### From version 1.2.1

- g. Improved the SPI code operation and prescaler calculation.

- h. Cast the root directory size value determination to a double word for devices in which the root directory begins after 0xFFFF clusters.
- i. Changed the size of the index variable in the FSformat function to allow calculation in devices with larger FATs.
- j. Replaced several hard-coded values with references to MEDIA\_SECTOR\_SIZE.
- k. Changed the return mechanism of the MediaInitialize function to provide compatibility with the USB stack. This function will now return a pointer to a MEDIA\_INFORMATION structure, which contains an error code and (for the USB stack) the size of a sector (in bytes) and the number of LUNs on the device. This structure's definition is located in FSDefs.h.
- l. Added a new error code: CE\_UNSUPPORTED\_SECTOR\_SIZE indicates that the sector size of the device is not supported by the file system.

#### From version 1.2.0

- a. Fixed an issue with the calculation of the SPI prescaler value for 16-bit microcontrollers.
- b. Changed the 'cwdclus' type in the SearchRec structure to 'unsigned long'
- c. Fixed a potential null-pointer reference in the Cache\_File\_Entry() function
- d. Improved PIC32 reliability.
- e. Changed PIC32 code to allow the user to select which SPI module to use. Selection is performed using #define macros in HardwareProfile.h.
- f. Changed PIC32 code to allow the user to define the desired SPI clock frequency. Selection is performed by setting #define SPI\_FREQUENCY to the desired value in HardwareProfiles.h.
- g. Changed the return type of MDD\_SDSPI\_ShutdownMedia to match the USB Host ShutdownMedia function.
- h. Replaced instances of \_\_C18XX with \_\_18CXX
- i. Added the packed attribute to several structures in FSDefs.h.

#### From version 1.1.2

- a. Fixed a bug that prevented the library from correctly loading the boot sector on devices with no Master Boot Record.
- b. Added support for 8.3 format directory names (up to 8 name characters and 3 extension characters.) To create or access directories with extensions, use path strings with radix characters (e.g. FSmkdir ("EXAMPLE.DIR")).
- c. Added checks to the FSmkdir function to prevent the user from creating files with too many radix characters ('.'). Radixes at the beginning of the directory name will cause the FSmkdir function to fail.
- d. Added a check to the FSrmdir function to prevent the user from using it to delete non-directory files or the current working directory.
- e. Added the question mark ('?') partial string search operator to the FindFirst and FindNext functions. Now when calling FindFirst or FindNext, you can skip checks of individual characters by replacing them with question marks in the search string. For example, calling FindFirst ("F?L?.TX?", ATTR\_ARCHIVE, &rec); would let you find the files "FILE.TXT," "FOLD.TXT," "FILM.TXM," etc.

- f. Modified the FindFirst and FindNext functions to correctly output directory names with extensions.
- g. Modified the FSgetcwd function to correctly insert directory names with extensions in a path string.
- h. Merged the functions to validate file/directory name characters together.
- i. Added three new methods of opening files. To use these methods, just specify the new strings as the mode argument in the FSfopen function. The new modes are:
  - a. “r+”: File will be opened for reading or writing
  - b. “w+”: File will be opened for reading or writing. If the file exists, its length will be truncated to 0.
  - c. “a+”: File will be opened for reading or writing. If the file exists, the current location within the file will be set to the end of the file.
- j. Modified the FSfopen function to allow the user to open directories in the read mode.
- k. Modified the FSrename function. Now, to rename a directory, open the directory in read mode with FSfopen and pass the pointer to that open directory into FSrename.
- l. Added a new function. The FSattrib function will allow the user to change the attributes of files and directories. The API is:

Function:           int FSattrib (FSFILE \* file, unsigned char attributes)  
 PreCondition:     The file or directory pointed to by ‘file’ has been opened successfully.  
 Input:            file           - The file or directory to modify  
                   attributes - The new attributes, including:

Attribute	Value	Indication
ATTR_READ_ONLY	0x01	The read-only attribute
ATTR_HIDDEN	0x02	The hidden attribute
ATTR_SYSTEM	0x04	The system attribute
ATTR_ARCHIVE	0x20	The archive attribute

Output:            int            - Returns 0 if success, -1 otherwise  
 Side Effects:     None  
 Overview:         Change the attributes of a file or directory.  
 Note:             None  
 Example:

```
FSFILE * pointer;
pointer = FSfopen ("FILE.TXT", "r");
if (pointer == NULL)
    // Error
if (FSattrib (pointer, ATTR_READ_ONLY | ATTR_HIDDEN) != 0)
    // Error
FSfclose (pointer);
```

- m. Modified the SD Data Logger project to include a new shell command; the ‘ATTRIB’ command will let the user change or display the attributes of a file.

Example 1: ATTRIB +R +S -H -A FILE.TXT

This command will give the file FILE.TXT the read-only and system attributes, and remove the hidden and archive attributes, if they’re set.

Example 2: ATTRIB FILE.TXT

This example will display the attributes of FILE.TXT.

- n. Added a new function. The FSferror function will provide information about why a previously called function failed. The API is:

```

Function:          int FSError (void)
PreCondition:     None
Input:            None
Output:           int    - Value depends on the last failing
function
    FSInit
        CE_GOOD - No Error
        CE_INIT_ERROR - The physical media could not be
            initialized
        CE_BAD_SECTOR_READ - The MBR or the boot sector could
            not be read correctly.
        CE_BAD_PARTITION - The MBR signature code was
            incorrect.
        CE_NOT_FORMATTED - The boot sector signature code was
            incorrect or indicates an invalid number of
            bytes per sector.
        CE_CARDFAT32 - The physical media is FAT32 type (only
            an error when FAT32 support is disabled).
        CE_UNSUPPORTED_FS - The device is formatted with an
            unsupported file system (not FAT12 or 16).

    FSfopen
        CE_GOOD - No Error
        CE_NOT_INIT - The device has not been initialized.
        CE_TOO_MANY_FILES_OPEN - The function could not
            allocate any additional file information to the
            array of FSFILE structures or the heap.
        CE_INVALID_FILENAME - The file name argument was
            invalid.
        CE_INVALID_ARGUMENT - The user attempted to open a
            directory in a write mode or specified an
            invalid mode argument.
        CE_FILE_NOT_FOUND - The specified file (which was to
            be opened in read mode) does not exist on the
            device.
        CE_BADCACHEREAD - A read from the device failed.
        CE_ERASE_FAIL - The existing file could not be erased
            (when opening a file in WRITE mode).
        CE_DIR_FULL - The directory is full.
        CE_DISK_FULL - The data memory section is full.
        CE_WRITE_ERROR - A write to the device failed.
        CE_SEEK_ERROR - The current position in the file could
            not be set to the end (when the file was opened
            in APPEND mode).

    FSfclose
        CE_GOOD - No Error
        CE_WRITE_ERROR - The existing data in the data buffer
            or the new file entry information could not be
            written to the device.
        CE_BADCACHEREAD - The file entry information could not
            be cached

    FSfread
        CE_GOOD - No Error
        CE_WRITEONLY - The file was opened in a write-only
            mode.
        CE_WRITE_ERROR - The existing data in the data buffer
            could not be written to the device.
        CE_BAD_SECTOR_READ - The data sector could not be
            read.
        CE_EOF - The end of the file was reached.
        CE_COULD_NOT_GET_CLUSTER - Additional clusters in the
            file could not be loaded.

```

#### FSfwrite

CE\_GOOD - No Error  
CE\_READONLY - The file was opened in a read-only mode.  
CE\_WRITE\_PROTECTED - The device write-protect check function indicated that the device has been write-protected.  
CE\_WRITE\_ERROR - There was an error writing data to the device.  
CE\_BADCACHEREAD - The data sector to be modified could not be read from the device.  
CE\_DISK\_FULL - All data clusters on the device are in use.

#### FSfseek

CE\_GOOD - No Error  
CE\_WRITE\_ERROR - The existing data in the data buffer could not be written to the device.  
CE\_INVALID\_ARGUMENT - The specified offset exceeds the size of the file.  
CE\_BADCACHEREAD - The sector that contains the new current position could not be loaded.  
CE\_COULD\_NOT\_GET\_CLUSTER - Additional clusters in the file could not be loaded/allocated.

#### FSftell

CE\_GOOD - No Error

#### FSattrib

CE\_GOOD - No Error  
CE\_INVALID\_ARGUMENT - The attribute argument was invalid.  
CE\_BADCACHEREAD - The existing file entry information could not be loaded.  
CE\_WRITE\_ERROR - The file entry information could not be written to the device.

#### FSrename

CE\_GOOD - No Error  
CE\_FILENOTOPENED - A null file pointer was passed into the function.  
CE\_INVALID\_FILENAME - The file name passed into the function was invalid.  
CE\_BADCACHEREAD - A read from the device failed.  
CE\_FILENAME\_EXISTS - A file with the specified name already exists.  
CE\_WRITE\_ERROR - The new file entry data could not be written to the device.

#### FSfeof

CE\_GOOD - No Error

#### FSformat

CE\_GOOD - No Error  
CE\_INIT\_ERROR - The device could not be initialized.  
CE\_BADCACHEREAD - The master boot record or boot sector could not be loaded successfully.  
CE\_INVALID\_ARGUMENT - The user selected to create their own boot sector on a device that has no master boot record, or the mode argument was invalid.  
CE\_WRITE\_ERROR - The updated MBR/Boot sector could not be written to the device.

CE\_BAD\_PARTITION - The calculated number of sectors per clusters was invalid.  
CE\_NONSUPPORTED\_SIZE - The card has too many sectors to be formatted as FAT12 or FAT16.

#### FSremove

CE\_GOOD - No Error  
CE\_WRITE\_PROTECTED - The device write-protect check function indicated that the device has been write-protected.  
CE\_INVALID\_FILENAME - The specified filename was invalid.  
CE\_FILE\_NOT\_FOUND - The specified file could not be found.  
CE\_ERASE\_FAIL - The file could not be erased.

#### FSchdir

CE\_GOOD - No Error  
CE\_INVALID\_ARGUMENT - The path string was mis-formed or the user tried to change to a non-directory file.  
CE\_BADCACHEREAD - A directory entry could not be cached.  
CE\_DIR\_NOT\_FOUND - Could not find a directory in the path.

#### FSgetcwd

CE\_GOOD - No Error  
CE\_INVALID\_ARGUMENT - The user passed a 0-length buffer into the function.  
CE\_BADCACHEREAD - A directory entry could not be cached.  
CE\_BAD\_SECTOR\_READ - The function could not determine a previous directory of the CWD.

#### FSmkdir

CE\_GOOD - No Error  
CE\_WRITE\_PROTECTED - The device write-protect check function indicated that the device has been write-protected.  
CE\_INVALID\_ARGUMENT - The path string was mis-formed.  
CE\_BADCACHEREAD - Could not successfully change to a recently created directory to store its dir entry information, or could not cache directory entry information.  
CE\_INVALID\_FILENAME - One or more of the directory names has an invalid format.  
CE\_WRITE\_ERROR - The existing data in the data buffer could not be written to the device or the dot/dotdot entries could not be written to a newly created directory.  
CE\_DIR\_FULL - There are no available dir entries in the CWD.  
CE\_DISK\_FULL - There are no available clusters in the data region of the device.

#### FSrmdir

CE\_GOOD - No Error  
CE\_DIR\_NOT\_FOUND - The directory specified could not be found or the function could not change to a subdirectory within the directory to be deleted (when recursive delete is enabled).

CE\_INVALID\_ARGUMENT - The user tried to remove the CWD or root directory.  
CE\_BADCACHEREAD - A directory entry could not be cached.  
CE\_DIR\_NOT\_EMPTY - The directory to be deleted was not empty and recursive subdirectory removal was disabled.  
CE\_ERASE\_FAIL - The directory or one of the directories or files within it could not be deleted.  
CE\_BAD\_SECTOR\_READ - The function could not determine a previous directory of the CWD.

#### SetClockVars

CE\_GOOD - No Error  
CE\_INVALID\_ARGUMENT - The time values passed into the function were invalid.

#### FindFirst

CE\_GOOD - No Error  
CE\_INVALID\_FILENAME - The specified filename was invalid.  
CE\_FILE\_NOT\_FOUND - No file matching the specified criteria was found.  
CE\_BADCACHEREAD - The file information for the file that was found could not be cached.

#### FindNext

CE\_GOOD - No Error  
CE\_NOT\_INIT - The SearchRec object was not initialized by a call to FindFirst.  
CE\_INVALID\_ARGUMENT - The SearchRec object was initialized in a different directory from the CWD.  
CE\_INVALID\_FILENAME - The filename is invalid.  
CE\_FILE\_NOT\_FOUND - No file matching the specified criteria was found.

#### FSfprintf

CE\_GOOD - No Error  
CE\_WRITE\_ERROR - Characters could not be written to the file.

Side Effects: None  
Overview: Returns an error value for the last function called.

Note: None

#### Example:

```
int error;
FSFILE * pointer;
pointer = FSfopen ("FILE.TXT", "r");
if (pointer == NULL)
    error = FSferror();
switch (error)
{
    // Error handling
}
```

- o. Revised most of the comment headers in the library.
- p. Generated a CHM help file for the library. This file can be found in the (default) directory "...Microchip Solutions\Microchip\Help"
- q. Removed extraneous macros and definitions.

- r. Added a new Microchip standard header file (Compiler.h) to the library.
- s. Removed the architecture-type configuration from the sample HardwareProfile.h files. This will now be taken care of automatically within the source files.

#### From version 1.1.1

- a. Fixed a bug that prevented the allocation of new clusters to the root directory in FAT32 implementations.
- b. Fixed a bug that prevented writing more than one cluster's worth of file entries to the root directory in FAT16/FAT12 implementations.
- c. Fixed a bug that returned an incorrect date for directory entries located in the first directory entry after a cluster boundary of a FAT32 root directory.
- d. Fixed a bug with FSrename that would cause the function to improperly fail if the directory entries in the current working directory (or previous directory, when renaming the CWD) completely filled a cluster (and no data clusters were allocated to the directory after that).

#### From version 1.1.0

- a. Fixed a bug with the PIC24 clock divider that was causing the interface to run more slowly than intended.
- b. Added support for PIC32 microcontrollers.

#### From version 1.01

- a. Added support for FAT32. To enable this functionality, make sure the SUPPORT\_FAT32 macro is uncommented in FSconfig.h.
- b. Added functions to provide support for the USB Mass Storage Host code.
- c. Moved pin and hardware definitions from physical interface files to HardwareProfiles.h.
- d. Created function pointers for functions that vary between interface files. These are located in FSconfig.h.
- e. Moved macros to select the correct physical layer to HardwareProfiles.h.
- f. Modified the SD-SPI physical layer to ensure that communication speed during startup falls between 100 kHz and 400 kHz
- g. Created a new example project: MDD File System-PIC24-SD Data Logger. This project contains code for a shell-style program based on the USB Thumb-drive shell demonstrated in Application Note 1145.
- h. Decreased the delay in the SD-SPI media initialization from 100 ms to 1 ms.
- i. Added the ability to change directories when writes are disabled.

#### From version 1.0

- a. FindFirst and FindNext will now return the create time/data in the timestamp field of a SearchRec object when they return values for a directory.
- b. Corrects a bug in the FindEmptyCluster function when searching for files beyond the end of a storage device.
- c. Automatically aligns buffers for 16-bit architectures.



- d. For the SPI interface, prescaler divides will now be determined dynamically based on the system clock speed defined in FSconfig.h.
- e. The DiskMount, LoadMBR, LoadBootSector, and FSFormat functions, as well as the gDiskData, gFATBuffer, and gDataBuffer structures are now located in FSIO.c instead of in the interface files.
- f. The SectorRead function will now do a dummy read of the sector and discard the data if it is called with NULL as the data pointer.
- g. Replaced the device initialization code in the FSFormat function with calls to InitIO and MediaInitialize.
- h. The MediaDetect function is not de-bounced. In order to determine that a device is available, you must call MediaDetect, wait for an appropriate amount of time, and then call it again.
- i. The sample linker script in the MDD File System-PIC18-CF-DynMem-UserDefClock project has been modified. Previously, several databanks were merged together; this caused an issue accessing variables that spanned multiple data banks. C18 only allows users to access variables like these using pointers.
- j. Added a new user function. The FSrename function will allow the user to rename files and directories. A version that accepts a ROM filename is available for PIC18 (FSrenamepgm). The API is:

```

Function:      int FSrename (const char *fileName, FSFILE * fo)
PreCondition: None
Input:        fileName  - The new name of the file
              fo        - The file to rename
Output:       int       - Returns 0 if success, -1 otherwise
Side Effects: None
Overview:     Change the name of a file or directory
Note:        This function will change the name of the current
              working directory if 'fo' equals NULL.

```

### **3. Known Issues**

- a. This implementation does not support long file names. When using the FSremove or FSremovepgm functions on a file with long file names, the file's FAT entries and short name directory entry will be deleted successfully, but any long file name entries will not be removed.

### **4. Compiler Version Used**

This library was compiled using MPLAB C18 v.3.21, MPLAB C30 v.3.11b, and MPLAB C32 v1.3 C compilers.

### **5. Memory Size**

Unoptimized memory usage for the file interface library using the SD-SPI physical layer is given in Table 1. 512 bytes of data memory are used for the data buffer, and an additional 512 are used for the file allocation table buffer. Additional data memory will be needed based on the number of files opened by the user at once. The default data memory values provided include space for two files opened in static allocation mode. The

C18 data memory value includes a 512 byte stack. The first row of the table indicates the smallest amount of memory that the library will use (for read-only mode), and each subsequent row indicates the increase in memory caused by enabling other functionality. Optimized and unoptimized totals for program and data memory with all functions enabled are listed after the table. This data was compiled while allowing two file objects to be opened simultaneously.

**Table 1: Memory Usage (Unoptimized)**

Functions Included	Program Memory (C18)	Data Memory (C18)	Program Memory (C30)	Data Memory (C30)	Program Memory (C32)	Data Memory (C32)
All extra functions disabled (read only mode)	25203 bytes	1886 bytes	14460 bytes	1324 bytes	20196 bytes	2904 bytes
Read only mode with directory support	+8092 bytes	+77 bytes	+4182 bytes	+80 bytes	+5484 bytes	+92 bytes
File Search enabled	+3556 bytes	+0 bytes	+1497 bytes	+0 bytes	+1968 bytes	+0 bytes
Write enabled	+17366 bytes	+0 bytes	+9396 bytes	+0 bytes	+11632 bytes	+0 bytes
Format enabled (Write must be enabled)	+7394 bytes	+0 bytes	+4839 bytes	+0 bytes	+5088 bytes	+0 bytes
Directories enabled (With writes enabled)	+16293 bytes	+90 bytes	+8751 bytes	+80 bytes	+11704 bytes	+92 bytes
FSprintf enabled	Not testable under specified conditions		+4827 bytes	+0 bytes	+8536 bytes	+0 bytes
File Search and Directories enabled	+250 bytes	+0 bytes	+57 bytes	+0 bytes	+68 bytes	+0 bytes
Pgm functions enabled	+2640 bytes	+0 bytes	N/A	N/A	N/A	N/A

Total memory usage\*

C18:

Unoptimized Program memory- 72702 bytes

Unoptimized Data memory- 1976 bytes

**Optimized Program memory- 38134 bytes**

**Optimized Data Memory- 1976 bytes**

C30:

Unoptimized Program memory- 43827 bytes

Unoptimized Data memory- 1404 bytes

**Optimized Program memory- 24903 bytes**  
**Optimized Data memory- 1404 bytes**

C32:

Unoptimized Program memory- 59192 bytes  
Unoptimized Data memory- 2996 bytes  
**Optimized Program memory- 35236 bytes**  
**Optimized Data memory- 2996 bytes**

\*Note: C18 total memory usage does not include FSprintf functionality. Since FSprintf requires integer promotion to be enabled, using it greatly increases the code size of all functions.

## ***6. More Information***

More detailed information about the operation of this library is available in Application Note 1045, available from [www.microchip.com](http://www.microchip.com).