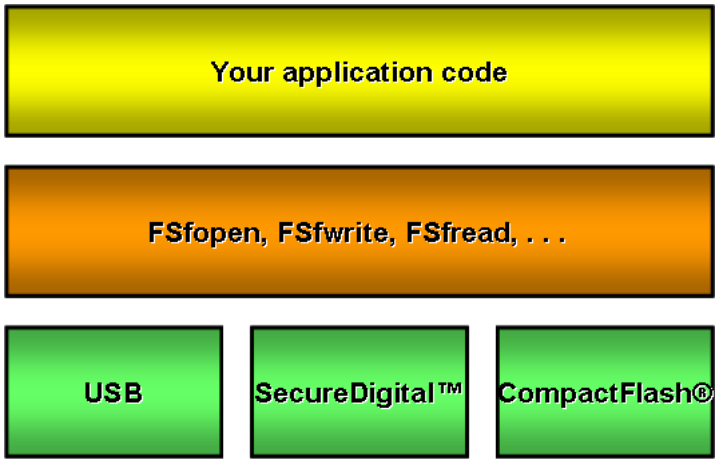
MDDFS Interface Library Help

Table of Contents

1 Microchip MDD File System Interface Library

Welcome to the Microchip Memory Disk Drive File System Interface Library!

Microchip's Memory Disk Drive File System(MDDFS) supports FAT12, FAT16 and FAT32 format for all 8, 16 and 32 bit PIC® MCU's. MDDFS software is independent of the physical layer used and can be easily integrated to any of the physical layers like USB, SD card, compact flash...etc...



MDDFS supports all file and directory operations(like read,write,remove,rename...etc...). The maximum length of any file or directory name is restricted to 255 characters. The Long File Name(LFN) support is available for 16 and 32 bit PIC® microcontrollers. 8 bit PIC® MCU's doesn't support LFN feature due to comparatively lesser RAM size. Whereas the basic 8.3 format filename is supported by all 8, 16 & 32 bit PIC® MCU's.The MDD File System Interface Library will provide an easy way to create and manipulate files on removable flash-based media devices.

USB Functionality

Note that the source code package and help file for this library do not include USB physical layer information. For more information about using the USB Host stack as a physical layer, please visit Microchip's USB Development Page or the AN1145: Using a USB Flash Drive with an Embedded Host page.

Updates

The latest version of the Microchip MDD File System Interface library is always available at Microchip MDD File System page.

Getting Help

The MDDFS Interface Library is supported through Microchip's standard support channels. If you encounter difficulties, you may submit ticket requests at http://support.microchip.com.

Thank You!

We appreciate your interest in the Microchip MDD File System Interface Library, and thank you for choosing Microchip products!

2 SW License Agreement

MICROCHIP IS WILLING TO LICENSE THE ACCOMPANYING SOFTWARE AND DOCUMENTATION TO YOU ONLY ON THE CONDITION THAT YOU ACCEPT ALL OF THE FOLLOWING TERMS. TO ACCEPT THE TERMS OF THIS LICENSE, CLICK "I ACCEPT" AND PROCEED WITH THE DOWNLOAD OR INSTALL. IF YOU DO NOT ACCEPT THESE LICENSE TERMS, CLICK "I DO NOT ACCEPT," AND DO NOT DOWNLOAD OR INSTALL THIS SOFTWARE.

NON-EXCLUSIVE SOFTWARE LICENSE AGREEMENT

This Nonexclusive Software License Agreement ("Agreement") is a contract between you, your heirs, successors and assigns ("Licensee") and Microchip Technology Incorporated, a Delaware corporation, with a principal place of business at 2355 W. Chandler Blvd., Chandler, AZ 85224-6199, and its subsidiary, Microchip Technology (Barbados) II Incorporated (collectively, "Microchip") for the accompanying Microchip software including, but not limited to, Graphics Library Software, IrDA Stack Software, MCHPFSUSB Stack Software, Memory Disk Drive File System Software, mTouch(TM) Capacitive Library Software, Smart Card Library Software, TCP/IP Stack Software, MiWi(TM) DE Software, Security Package Software, and/or any PC programs and any updates thereto (collectively, the "Software"), and accompanying documentation, including images and any other graphic resources provided by Microchip ("Documentation").

1. Definitions. As used in this Agreement, the following capitalized terms will have the meanings defined below:

a. "Microchip Products" means Microchip microcontrollers and Microchip digital signal controllers.

b. "Licensee Products" means Licensee products that use or incorporate Microchip Products.

c. "Object Code" means the Software computer programming code that is in binary form (including related documentation, if any), and error corrections, improvements, modifications, and updates.

d. "Source Code" means the Software computer programming code that may be printed out or displayed in human readable form (including related programmer comments and documentation, if any), and error corrections, improvements, modifications, and updates.

e. "Third Party" means Licensee’s agents, representatives, consultants, clients, customers, or contract manufacturers.

f. "Third Party Products" means Third Party products that use or incorporate Microchip Products.

2. Software License Grant. Microchip grants strictly to Licensee a non-exclusive, non-transferable, worldwide license to:

a. use the Software in connection with Licensee Products and/or Third Party Products;

b. if Source Code is provided, modify the Software; provided that Licensee clearly notifies Third Parties regarding the source of such modifications;

c. distribute the Software to Third Parties for use in Third Party Products, so long as such Third Party agrees to be bound by this Agreement (in writing or by "click to accept") and this Agreement accompanies such distribution;

d. sublicense to a Third Party to use the Software, so long as such Third Party agrees to be bound by this Agreement (in writing or by "click to accept");

e. with respect to the TCP/IP Stack Software, Licensee may port the ENC28J60.c, ENC28J60.h, ENCX24J600.c, and ENCX24J600.h driver source files to a non-Microchip Product used in conjunction with a Microchip ethernet controller;

f. with respect to the MiWi (TM) DE Software, Licensee may only exercise its rights when the Software is embedded on a Microchip Product and used with a Microchip radio frequency transceiver or UBEC UZ2400 radio frequency transceiver which are integrated into Licensee Products or Third Party Products.

For purposes of clarity, Licensee may NOT embed the Software on a non-Microchip Product, except as described in this

Section.

3. Documentation License Grant. Microchip grants strictly to Licensee a non-exclusive, non-transferable, worldwide license to use the Documentation in support of Licensee's authorized use of the Software

4. Third Party Requirements. Licensee acknowledges that it is Licensee’s responsibility to comply with any third party license terms or requirements applicable to the use of such third party software, specifications, systems, or tools. This includes, by way of example but not as a limitation, any standards setting organizations requirements and, particularly with respect to the Security Package Software, local encryption laws and requirements. Microchip is not responsible and will not be held responsible in any manner for Licensee’s failure to comply with such applicable terms or requirements.

5. Open Source Components. Notwithstanding the license grant in Section 1 above, Licensee further acknowledges that certain components of the Software may be covered by so-called "open source" software licenses ("Open Source Components"). Open Source Components means any software licenses approved as open source licenses by the Open Source Initiative or any substantially similar licenses, including without limitation any license that, as a condition of distribution of the software licensed under such license, requires that the distributor make the software available in source code format. To the extent required by the licenses covering Open Source Components, the terms of such license will apply in lieu of the terms of this Agreement. To the extent the terms of the licenses applicable to Open Source Components prohibit any of the restrictions in this Agreement with respect to such Open Source Components, such restrictions will not apply to such Open Source Component.

6. Licensee Obligations. Licensee will not: (a) engage in unauthorized use, modification, disclosure or distribution of Software or Documentation, or its derivatives; (b) use all or any portion of the Software, Documentation, or its derivatives except in conjunction with Microchip Products, Licensee Products or Third Party Products; or (c) reverse engineer (by disassembly, decompilation or otherwise) Software or any portion thereof. Licensee may not remove or alter any Microchip copyright or other proprietary rights notice posted in any portion of the Software or Documentation. Licensee will defend, indemnify and hold Microchip and its subsidiaries harmless from and against any and all claims, costs, damages, expenses (including reasonable attorney's fees), liabilities, and losses, including without limitation: (x) any claims directly or indirectly arising from or related to the use, modification, disclosure or distribution of the Software, Documentation, or any intellectual property rights related thereto; (y) the use, sale and distribution of Licensee Products or Third Party Products; and (z) breach of this Agreement.

7. Confidentiality. Licensee agrees that the Software (including but not limited to the Source Code, Object Code and library files) and its derivatives, Documentation and underlying inventions, algorithms, know-how and ideas relating to the Software and the Documentation are proprietary information belonging to Microchip and its licensors ("Proprietary Information"). Except as expressly and unambiguously allowed herein, Licensee will hold in confidence and not use or disclose any Proprietary Information and will similarly bind its employees and Third Party(ies) in writing. Proprietary Information will not include information that: (i) is in or enters the public domain without breach of this Agreement and through no fault of the receiving party; (ii) the receiving party was legally in possession of prior to receiving it; (iii) the receiving party can demonstrate was developed by the receiving party independently and without use of or reference to the disclosing party's Proprietary Information; or (iv) the receiving party receives from a third party without restriction on disclosure. If Licensee is required to disclose Proprietary Information by law, court order, or government agency, License will give Microchip prompt notice of such requirement in order to allow Microchip to object or limit such disclosure. Licensee agrees that the provisions of this Agreement regarding unauthorized use and nondisclosure of the Software, Documentation and related Proprietary Rights are necessary to protect the legitimate business interests of Microchip and its licensors and that monetary damage alone cannot adequately compensate Microchip or its licensors if such provisions are violated. Licensee, therefore, agrees that if Microchip alleges that Licensee or Third Party has breached or violated such provision then Microchip will have the right to injunctive relief, without the requirement for the posting of a bond, in addition to all other remedies at law or in equity.

8. Ownership of Proprietary Rights. Microchip and its licensors retain all right, title and interest in and to the Software and Documentation including, but not limited to all patent, copyright, trade secret and other intellectual property rights in the Software, Documentation, and underlying technology and all copies and derivative works thereof (by whomever produced). Licensee and Third Party use of such modifications and derivatives is limited to the license rights described in this Agreement.

9. Termination of Agreement. Without prejudice to any other rights, this Agreement terminates immediately, without notice by Microchip, upon a failure by Licensee or Third Party to comply with any provision of this Agreement. Upon termination, Licensee and Third Party will immediately stop using the Software, Documentation, and derivatives thereof, and immediately

destroy all such copies.

10. Warranty Disclaimers. THE SOFTWARE AND DOCUMENTATION ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING WITHOUT LIMITATION, ANY WARRANTY OF MERCHANTABILITY, TITLE, NON-INFRINGEMENT AND FITNESS FOR A PARTICULAR PURPOSE. MICROCHIP AND ITS LICENSORS ASSUME NO RESPONSIBILITY FOR THE ACCURACY, RELIABILITY OR APPLICATION OF THE SOFTWARE OR DOCUMENTATION. MICROCHIP AND ITS LICENSORS DO NOT WARRANT THAT THE SOFTWARE WILL MEET REQUIREMENTS OF LICENSEE OR THIRD PARTY, BE UNINTERRUPTED OR ERROR-FREE. MICROCHIP AND ITS LICENSORS HAVE NO OBLIGATION TO CORRECT ANY DEFECTS IN THE SOFTWARE.

11. Limited Liability. IN NO EVENT WILL MICROCHIP OR ITS LICENSORS BE LIABLE OR OBLIGATED UNDER ANY LEGAL OR EQUITABLE THEORY FOR ANY DIRECT OR INDIRECT DAMAGES OR EXPENSES INCLUDING BUT NOT LIMITED TO INCIDENTAL, SPECIAL, INDIRECT, PUNITIVE OR CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, COST OF PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY, SERVICES, OR ANY CLAIMS BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF), OR OTHER SIMILAR COSTS. The aggregate and cumulative liability of Microchip and its licensors for damages hereunder will in no event exceed $1000 or the amount Licensee paid Microchip for the Software and Documentation, whichever is greater. Licensee acknowledges that the foregoing limitations are reasonable and an essential part of this Agreement.

12. General. THIS AGREEMENT WILL BE GOVERNED BY AND CONSTRUED UNDER THE LAWS OF THE STATE OF ARIZONA AND THE UNITED STATES WITHOUT REGARD TO CONFLICTS OF LAWS PROVISIONS. Licensee agrees that any disputes arising out of or related to this Agreement, Software or Documentation will be brought exclusively in either the U.S. District Court for the District of Arizona, Phoenix Division, or the Superior Court of Arizona located in Maricopa County, Arizona. This Agreement will constitute the entire agreement between the parties with respect to the subject matter hereof. It will not be modified except by a written agreement signed by an authorized representative of Microchip. If any provision of this Agreement will be held by a court of competent jurisdiction to be illegal, invalid or unenforceable, that provision will be limited or eliminated to the minimum extent necessary so that this Agreement will otherwise remain in full force and effect and enforceable. No waiver of any breach of any provision of this Agreement will constitute a waiver of any prior, concurrent or subsequent breach of the same or any other provisions hereof, and no waiver will be effective unless made in writing and signed by an authorized representative of the waiving party. Licensee agrees to comply with all import and export laws and restrictions and regulations of the Department of Commerce or other United States or foreign agency or authority. The indemnities, obligations of confidentiality, and limitations on liability described herein, and any right of action for breach of this Agreement prior to termination, will survive any termination of this Agreement. Any prohibited assignment will be null and void. Use, duplication or disclosure by the United States Government is subject to restrictions set forth in subparagraphs (a) through (d) of the Commercial Computer-Restricted Rights clause of FAR 52.227-19 when applicable, or in subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227-7013, and in similar clauses in the NASA FAR Supplement. Contractor/manufacturer is Microchip Technology Inc., 2355 W. Chandler Blvd., Chandler, AZ 85224-6199.

If Licensee has any questions about this Agreement, please write to Microchip Technology Inc., 2355 W. Chandler Blvd., Chandler, AZ 85224-6199 USA. ATTN: Marketing.

License Rev. No. 05-012412

3 Release Notes

Microchip Memory Disk Drive File System Release Notes

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, along with Long File Name(LFN) support for files and directories. This library includes four different physical interface files:

- SecureDigital card interface using the SPI module
- CompactFlash card interface using manual bit toggling
- 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 can be used as a physical layer.

Latest version - 1.3.8 Log

1. Internal Flash.c Modifications:-
   - Modified "MDD_IntFlash_SectorRead" to write the correct word data in 'buffer' pointer.

Compiler Version Used

This library was compiled using MPLAB C18 v.3.40, MPLAB C30 v.3.31, and MPLAB C32 v2.02a complier.

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
FSfprintf 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 (without LFN support):

Unoptimized Program memory- 73702 bytes Unoptimized Data memory- 1976 bytes Optimized Program memory- 38834 bytes Optimized Data Memory- 1976 bytes

- C30 (with LFN support):

Unoptimized Program memory- 60606 bytes Unoptimized Data memory- 5104 bytes Optimized Program memory- 32903 bytes Optimized Data memory- 5104 bytes

- C32 (with LFN support):

Unoptimized Program memory- 80192 bytes Unoptimized Data memory- 6096 bytes Optimized Program memory- 41236 bytes Optimized Data memory- 6096 bytes

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

More Information

More detailed information about the operation of this library is available in Application Note 1045, available from www.microchip.com.

Previous Versions Log

version 1.3.6

1. SD-SPI.c Modifications:-
   - Modified "FSConfig.h" to "FSconfig.h" in '#include' directive.
   - Moved 'spiconvalue' variable definition to only C30 usage, as C32 is not using it.

- Modified 'MDD_SDSPI_MediaDetect' function to ensure that CMD0 is sent freshly after CS is asserted low. This minimizes the risk of SPI clock pulse master/slave syncronization problems.

2. FSIO.c Modifications:-
   - The function "FILEget_next_cluster" is made public.
   - Modified "FILEfind" function such that when using 8.3 format the file searches are not considered as case sensitive
   - In function 'CacheTime', the variables 'ptr1' & 'ptr0' are not used when compiled for PIC32. So there definitions were removed for PIC32.
   - Modified "rmdirhelper", "FormatDirName" & "writeDotEntries" functions to remove non-critical warnings during compilation.
   - Updated comments in most of the function header blocks.
3. FSIO.h Modifications:-
   - The function "FILEget_next_cluster" is made public.

version 1.3.4

1. SD-SPI.c Modifications:-
   - Added support for dsPIC33E & PIC24E controllers.
   - #include "HardwareProfile.h" is moved up in the order.
   - "#define SPI_INTERRUPT_FLAG_ASM PIR1, 3" is removed from SD-SPI.c. "SPI_INTERRUPT_FLAG_ASM" macro has to be defined in "HardwareProfile.h" file for PIC18 microcontrollers.
   - Replaced "__C30" usage with "__C30__"
2. FSIO.c Modifications:-
   - Initialized some of the local variables to default values to remove non-critical compiler warnings for code sanitation.
     - In function "FILEfind", local variables "fileFoundLfnIndex", "fileFoundMaxLfnIndex", "fileFoundDotPosition", "lfnMaxSequenceNum" & "reminder" initialized to '0'.
     - In function "FindEmptyEntries", local variable "a" initialized to '0'.
     - In function "FILEerase", local variable "clus" initialized to '0'.
     - In function "FormatFileName", local pointer variable "localFileName" initialized to 'NULL'.
   - The sector size of the media device is obtained from the MBR of media. So, instead of using the hard coded macro "DIRENTRIES_PER_SECTOR", the variables "dirEntriesPerSector" & "disk->sectorSize" are used in the code. The above mentioned variables use the sector size from the media rather than depending upon predefined macro "MEDIA_SECTOR_SIZE". Refer "Cache_File_Entry", "EraseCluster" & "writeDotEntries" fucntions to see the change.

version 1.3.2

1. Modified SD-SPI.c, MDD_SDSPI_AsyncWriteTasks() so pre-erase command only gets used for multi-block write scenarios. .

version 1.3.0

1. Supported LFN entries for files and directories.
2. Implemented new functions to improve the previous performance/flexibility issues in SD- SPI.c and .h files.
3. Fixed SD Card Initialization Issues, especially with SDHC.

version 1.2.4

1. 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).
2. Optimize some code by collapsing duplicated code into a single for loop.
3. Update to use the CSD register data to calculate the SD card capacity.
4. Fixed compatibility problems in SD-SPI.c that was preventing it from working with some of the cards.
5. Make modifications to allow dynamic sector size giving the possibility to use thumb drives with different sector sizes.
6. Corrected a bug that prevented the last two clusters of a drive to be able to be written to.

7. Add the GetDiskProperties function to allow the users to get the disk properties (size of disk, free space, etc)

version 1.2.3

1. 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.
2. Fixed bug that prevented the last two clusters in a partition from being allocated.
3. Added SDHC support for the MDD file system.
4. Fixed the problem where on an append to a file that is smaller than a sector.
5. Fixed TODO that limited internal flash to PSV page boundaries in on PIC24 and only PSVPAG = 1.
6. Fixed an issue with the FSattrib function that was changing the wrong cached file object.

version 1.2.2

1. Improved the SPI code operation and prescaler calculation.
2. Cast the root directory size value determination to a double word for devices in which the root directory begins after 0xFFFF clusters.
3. Changed the size of the index variable in the FSformat function to allow calculation in devices with larger FATs.
4. Replaced several hard-coded values with references to MEDIA_SECTOR_SIZE.
5. 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.
6. Added a new error code: CE_UNSUPPORTED_SECTOR_SIZE indicates that the sector size of the device is not supported by the file system.

version 1.2.1

1. Fixed an issue with the calculation of the SPI prescaler value for 16-bit microcontrollers.
2. Changed the 'cwdclus' type in the SearchRec structure to 'unsigned long'
3. Fixed a potential null-pointer reference in the Cache_File_Entry() function
4. Improved PIC32 reliability.
5. Changed PIC32 code to allow the user to select which SPI module to use. Selection is performed using #define macros in HardwareProfile.h.
6. 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.
7. Changed the return type of MDD_SDSPI_ShutdownMedia to match the USB Host ShutdownMedia function.
8. Replaced instances of C18XX with 18CXX
9. Added the packed attribute to several structures in FSDefs.h.

version 1.2.0

1. Fixed a bug that prevented the library from correctly loading the boot sector on devices with no Master Boot Record.
2. 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")).
3. 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.
4. Added a check to the FSrmdir function to prevent the user from using it to delete non- directory files or the current working directory.
5. 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.
6. Modified the FindFirst and FindNext functions to correctly output directory names with extensions.
7. Modified the FSgetcwd function to correctly insert directory names with extensions in a path string.

8. Merged the functions to validate file/directory name characters together.
9. 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:
   - “r+”: File will be opened for reading or writing
   - “w+”: File will be opened for reading or writing. If the file exists, its length will be truncated to 0.
   - “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.
10. Modified the FSfopen function to allow the user to open directories in the read mode.
11. 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.
12. Added a new function. The FSattrib function will allow the user to change the attributes of files and directories.
13. 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.
14. Added a new function. The FSferror function will provide information about why a previously called function failed.
15. Revised most of the comment headers in the library.
16. Generated a CHM help file for the library. This file can be found in the (default) directory “…\Microchip Solutions\Microchip\Help”
17. Removed extraneous macros and definitions.
18. Added a new Microchip standard header file (Compiler.h) to the library.
19. Removed the architecture-type configuration from the sample HardwareProfile.h files. This will now be taken care of automatically within the source files.

version 1.1.2

1. Fixed a bug that prevented the allocation of new clusters to the root directory in FAT32 implementations.
2. Fixed a bug that prevented writing more than one cluster’s worth of file entries to the root directory in FAT16/FAT12 implementations.
3. 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.
4. 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).

version 1.1.1

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

version 1.1.0

1. Added support for FAT32. To enable this functionality, make sure the SUPPORT_FAT32 macro is uncommented in FSconfig.h.
2. Added functions to provide support for the USB Mass Storage Host code.
3. Moved pin and hardware definitions from physical interface files to HardwareProfiles.h.
4. Created function pointers for functions that vary between interface files. These are located in FSconfig.h.
5. Moved macros to select the correct physical layer to HardwareProfiles.h.
6. Modified the SD-SPI physical layer to ensure that communication speed during startup falls between 100 kHz and 400 kHz
7. 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.
8. 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.

version 1.01

1. 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.
2. Corrects a bug in the FindEmptyCluster function when searching for files beyond the end of a storage device.
3. Automatically aligns buffers for 16-bit architectures.
4. For the SPI interface, prescaler divides will now be determined dynamically based on the system clock speed defined in FSconfig.h.
5. 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.
6. 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.
7. Replaced the device initialization code in the FSFormat function with calls to InitIO and MediaInitialize.
8. 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.
9. 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.
10. 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).

4 Getting Started

Information about how to easily get started with the MDDFS library.

Description

This section will walk through the terminologies used in FAT file systems, initial configuration of the stack and compatible Microchip development hardware.

4.1 Terminology

Below are the terms which are frequently referred in the File Systems.

4.1.1 Boot sector

The boot sector is the first sector of a partition. It contains information about how the partition is organized.

4.1.2 Cluster

A cluster is a group of sectors in the data region of a FAT partition. The number of sectors per cluster can be any positive, power-of-two signed 8-bit value (1, 2, 4, 8, 16, 32, or 64) and is set when the partition is formatted.

4.1.3 Current Working Directory

All file I/O operations (except those that accept a path variable) take place within the current working directory. When FSInit completes successfully the CWD will be set the the root directory. It can be changed using the FSchdir or FSchdirpgm function.

4.1.4 Directory

A directory is a type of file that contains pointers to other files or directories.

4.1.5 FAT

The File Allocation Table. The FAT is an array-based linked list with one entry for each data cluster on the device. Each entry either points to the next cluster of a file or contains a special value. FAT12 has 12-bit entries, FAT16 has 16-bit entries, and FAT32 has 32-bit entries.

FAT can also refer to the FAT file system itself.

4.1.6 Master Boot Record

The first cluster of a device. The master boot record contains pointers to different partitions on the device and information about how they're organized.

4.1.7 Root directory

The root directory is a directory that is the base of the directory tree. For FAT12 and FAT16 the root directory is located after the FAT; for FAT32 the root directory is make up of clusters (like a regular directory) and is located in the data region of the device.

4.1.8 Sector

A sector is a group of bytes in the FAT file system. Sectors are most commonly 512 bytes.

4.1.9 LFN

LFN refers to the Long File Name entries of the files and directories present in the memory. As per the LFN specifications, the file or directory name can be maximum of 255 characters.

4.2 Required Hardware

To run this project, you will need one of the following sets of hardware:

4.2.1 Configuration 1: PIC18 Explorer Board

1. PIC18 Explorer Board (Microchip part number DM183032)
2. SD Card PICTail™ Plus Daughter Card (Microchip part number AC164122)
3. PIC18F87J50 Plug-In-Module (PIM)(Microchip part number MA180021)

4.2.2 Configuration 2: Explorer 16 Board

1. Explorer 16 (Microchip part number DM240001)
2. SD Card PICTail™ Plus Daughter Card (Microchip part number AC164122)

3. And one of the following PIMs
    1. PIC24FJ128GA010 Plug-In-Module (PIM)(Microchip part number MA240011)
    2. PIC24FJ256GB110 Plug-In-Module (PIM)(Microchip part number MA240014)
    3. PIC24EP512GU810 Plug-In-Module (PIM)(Microchip part number MA240025-1)
    4. dsPIC33EP512MU810 Plug-In-Module (PIM)
    5. PIC32MX360F512L Plug-In-Module (PIM)(Microchip part number MA320001)
    6. PIC32MX460F512L Plug-In-Module (PIM)(Microchip part number MA320002)

4.2.3 Configuration 3: PIC24FJ256DA210 Development Board

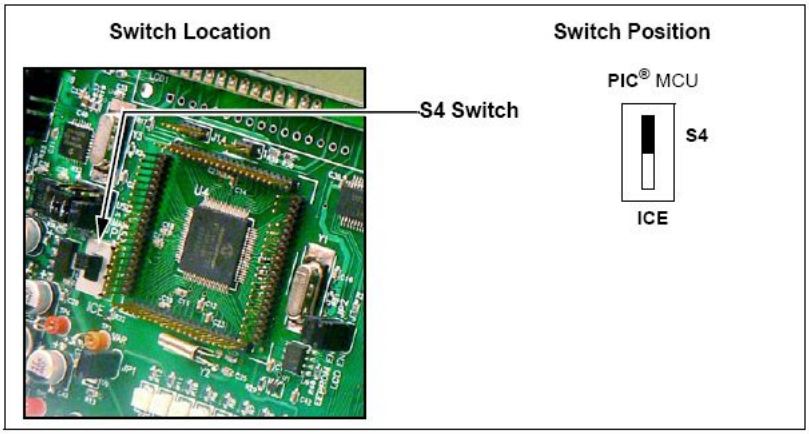
1. PIC24FJ256DA210 Development Board (Microchip part number DM240312)
2. SD Card PICTail™ Plus Daughter Card (Microchip part number AC164122)

4.3 Configuring Hardware

This section describes how to set up the various configurations of hardware to run this demo.

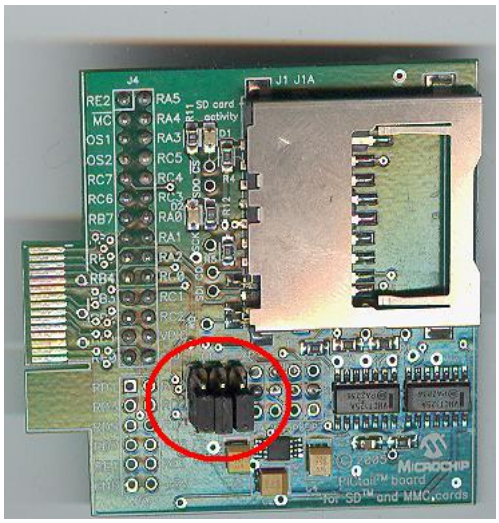
4.3.1 Configuration using PIC18 Explorer Board

1. Before inserting PIC18F87J50 PIM PIM in the PIC18 Explorer board, insure that the processor selector switch (S4) is in the “ICE” position as seen in the image below. Failure to so will result in difficulties in getting the PIC18F87J50 PIM to sit properly on the PIC18 Explorer.



- **Note:** The processor selector switch (S4) should be in “PIC® MCU” position when on board PIC18F8722 chip is used for the demo application.

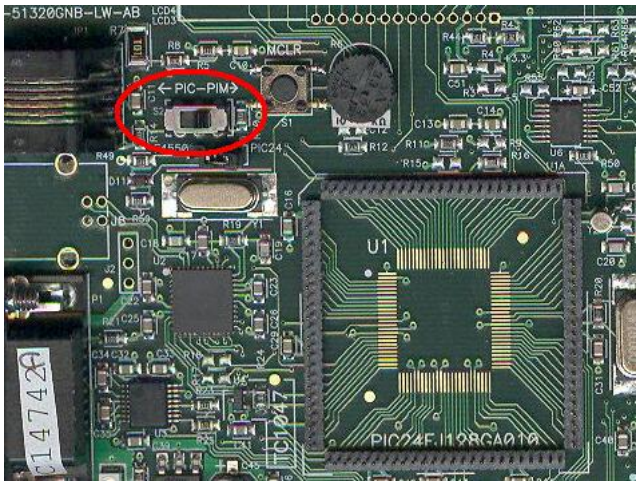
2. Be careful while inserting the PIC18F87J50 PIM into PIC18 board. Insure that no pins are bent or damaged during the process. Also insure that the PIM is not shifted in any direction and that all of the headers are properly aligned.
3. On the SD Card PICTail™ Plus board, short JP1, JP2, and JP3 on the side farthest from the SD Card holder. Depending on the revision of the board you have the silk-screen on the board may incorrectly label the top as the “HPC-EXP” setting. Please ignore this silk screen and place the jumpers as described above and seen below.



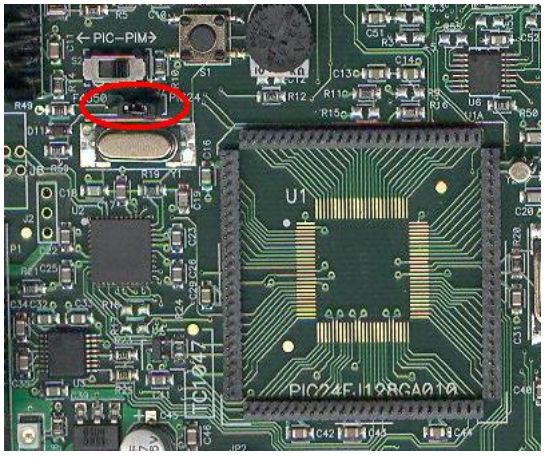
4. Insert the J4 port pins of SD Card PICTail™ Plus Daughter Card in the J3 port of PIC18 Explorer board with correct pin to pin mapping. Insert the SD Card in SD Card PICTail™ Plus Daughter board.

4.3.2 Configuration using Explorer 16 Board

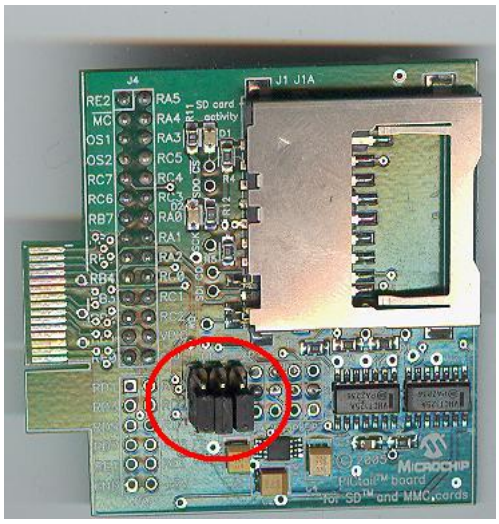
1. Before attaching the PIM to the Explorer 16 board, insure that the processor selector switch (S2) is in the “PIM” position as seen in the image below.



2. Short the J7 jumper to the “PIC24” setting:



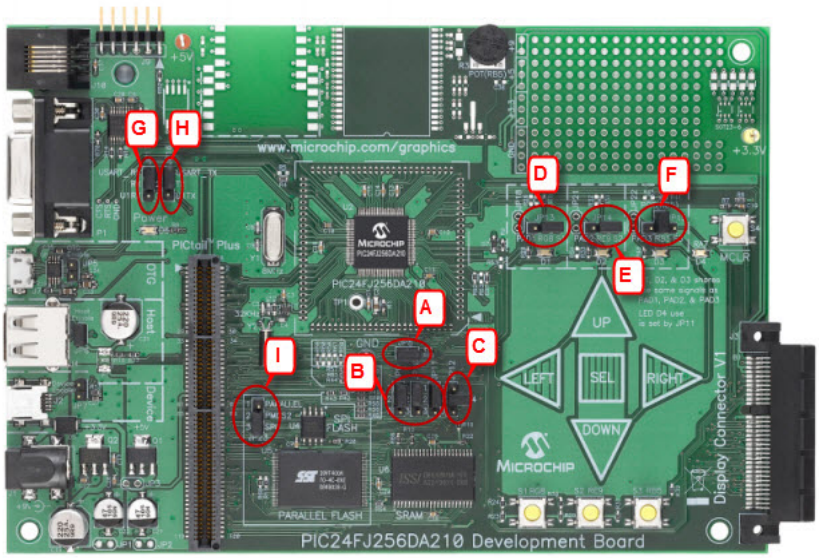
3. Be careful while inserting the appropriate PIM into Exp 16 board. Insure that no pins are bent or damaged during the process. Also insure that the PIM is not shifted in any direction and that all of the headers are properly aligned.
4. On the SD Card PICTail™ Plus board, short JP1, JP2, and JP3 on the side farthest from the SD Card holder. Depending on the revision of the board you have the silk-screen on the board may incorrectly label the top as the “HPC-EXP” setting. Please ignore this silk screen and place the jumpers as described above and seen below.



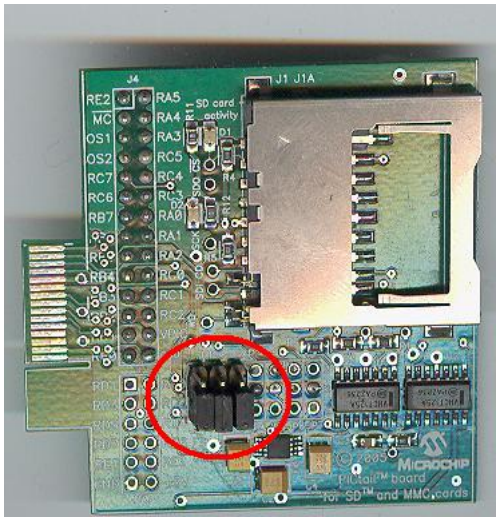
5. Insert the J2 slot of SD Card PICTail™ Plus Daughter Card into J5 port of Explorer 16 board. Make sure that the SD Card Connector is facing towards the Explorer 16 board. Insert the SD Card in SD Card PICTail™ Plus Daughter board.

4.3.3 Configuration using PIC24FJ256DA210 Development Board

1. Before attaching the SD Card PICTail™ Plus Daughter Card to the PIC24FJ256DA210 Development Board, make sure that the jumpers on the development board are set to the default positions as seen in the image below.
    1. JP8 – Install jumper
    2. JP9, JP10, JP11 – Install jumper to pins 1-2
    3. JP12 – Install jumper to pins 2-4
    4. JP13 – Install jumper to pins RG8-S1
    5. JP14 – Install jumper to pins RE9-S2
    6. JP15 – Install jumper to pins RB5-POT
    7. JP16 – Install jumper to TX-USART_TX
    8. JP17 – Install jumper to RX-USART_RX
    9. JP23 – Install jumper to PMCS2-SPI



2. On the SD Card PICTail™ Plus board, short JP1, JP2, and JP3 on the side farthest from the SD Card holder. Depending on the revision of the board you have the silk- screen on the board may incorrectly label the top as the “HPC-EXP” setting. Please ignore this silk screen and place the jumpers as described above and seen below.



3. Insert the J2 slot of SD Card PICTail™ Plus Daughter Card into PICTail™ Plus (J8) port of PIC24FJ256DA210 development board with correct pin to pin mapping. Make sure that the SD Card Connector is facing away from the PIC24FJ256DA210 chip of the development board. Insert the SD Card in SD Card PICTail™ Plus Daughter board.

4.4 Firmware Directory Structure

The MDDFS Library comes with many files, documents, and project examples. Before getting started, take a moment to familiarize yourself with the firmware directory structure so that you may find what you need quickly.

Directory Structure

By default, the MDDFS Library installs into C:\Microchip Solutions along with any other Microchip software stacks you may be using. Inside that folder, several subdirectories are created, as documented in the table below.

C:\Microchip Solutions	Root folder for all library files
\MDD File System-SD Card	Main demo application for file system interfacing with SD card.

\MDD File System-SD Card\PIC18F	Configuration, linker & demo file for the PIC18F project.
\MDD File System-SD Card\PIC24F	Configuration & demo files for the PIC24F project.
\MDD File System-SD Card\dsPIC33E_PIC24E	Configuration & demo files for the dsPIC33E and PIC24E project.
\MDD File System-SD Card\PIC32	Configuration & demo files for the PIC32 project.
\Microchip	Internal stack files. These files rarely need modification.
\Microchip\MDD File System	Source (*.c) files of the MDD File System Library
\Microchip\PIC18 salloc	Source (*.c) files for dynamic memory allocation for PIC18
\Microchip\MDD File System\Documentation	Readme files, schematics, and AN1045 application note.
\Microchip\Help	The location of this help file.
\Microchip\Include	Internal stack header files. These files rarely need modification
\Microchip\Include\MDD File System	Header (*.h) files for the MDDFS Library
\Microchip\Include\PIC18 salloc	Header (*.h) files for dynamic memory allocation for PIC18

4.5 Firmware

To run this project, you will need to load the corresponding firmware into the devices.

The source code for this demo is available in the "\Microchip Solutions\MDD File System-SD Card" directory. In this directory you will find all of the user level source and header files, linker file as well as project file for each of the hardware platforms. Find the project (*.mcp) file that corresponds to the hardware platform you wish to test. Compile and program the demo code into the hardware platform. For more help on how to compile and program projects, please refer to the MPLAB® IDE help available through the help menu of MPLAB (Help->Topics…->MPLAB IDE).

4.6 Running the SD Card Demo

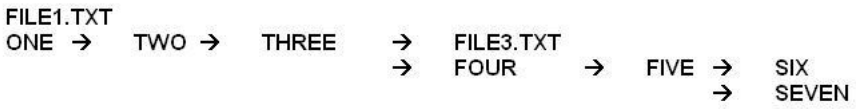
The “MDD File System - SD Card” demo application supports PIC18, PIC24F, PIC24E, dsPIC33E and PIC32 architectures. The demo uses the selected hardware platform for data I/O operations with SD card through SPI channel. This demo shows how to create a file, write into the file, close the file, read from the file, rename the file, delete a file, create a directory, change the current working directory, delete the directory and it's contents, search a file in the directory...etc...etc.. in the SD card memory. The maximum length of any file or directory name is restricted to 255 characters (LFN format). The Long File Name (LFN) format is supported for only 16 and 32 bit PIC® microcontrollers. PIC18 microcontrollers doesn’t support LFN feature due to comparatively lesser RAM size. Whereas the basic 8.3 format filename is supported by all 8, 16 & 32 bit PIC® microcontrollers. All the project demos of “MDD File System - SD Card” have to be programmed & verified in debug mode of MPLAB IDE.

After programming the appropriate firmware on the appropriate hardware platform, run the demo application till the last while(1) loop in the main() function. Remove the SD card from the SD Card PICTail™ Plus Daughter board and verify the final contents of the SD card in your Laptop or PC. The contents of the SD card should match the below file and directory structure.

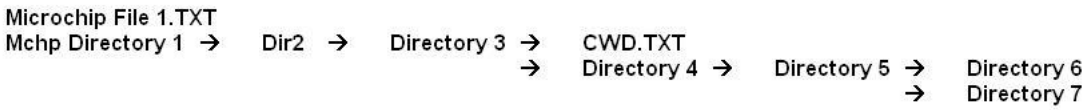
1. PIC18F: (using Demonstration.c)

2. PIC24F,PIC32 & dsPIC33E/PIC24E:

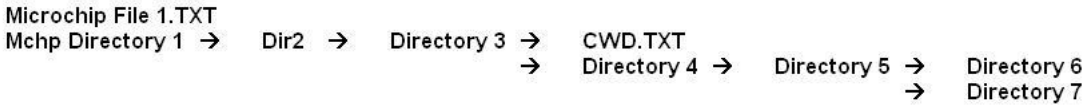
- Using "Demonstration1.c" file:-



- Using "Demonstration2.c" file:-



- Using "Demonstration3.c" file:-



Note:

1. Please open the corresponding "Demonstration .c" file to understand the flow of the source code thoroughly.
2. "Demonstration.c" & "Demonstration1.c" files show the usage of File System API's when the file & directory names are in 8.3 Format.
3. "Demonstration2.c" file shows the usage of File System API's when the file & directory names are in LFN Format.
4. "Demonstration3.c" file shows the usage of File System API's when the file & directory names are in UTF-16 bit Format.

For more details about SD card communication using MDD – File System Library, please refer AN1045

Troubleshooting Tips:

Issue 1: How to increase the speed of SD card read/write operation?

Solution: The main bottleneck to increase the speed for SD card read/write operation is SPI clock frequency. In the released stack the SPI clock rate is set as 4 MHz. Please search for "OpenSPIM(SYNC_MODE_FAST)" in the stack. This function sets the SPI clock speed during the data transfers with the SD card. This function is called in "MDD_SDSPI_MediaInitialize()" function. If you want to increase the SPI clock rate for PIC18/PIC24F, modify the SYNC_MODE_FAST macro value. The maximum value of SPI clock frequency that can be set for PIC18/PIC24F microcontrollers is 8 MHz.

For PIC32, modify the macro "SPI_FREQUENCY" to change the SPI clock rate for the data transfers. The maximum value of SPI clock frequency that can be set for PIC32 microcontrollers is 25 MHz.

Please verify peripheral bus frequency, system clock frequency, SPI Baud Rate register values (SPI Baud Rate Calculation formula) and the corresponding PIC® microcontroller datasheet before modifying the macro "SYNC_MODE_FAST" or "SPI_FREQUENCY" in the source code.

4.7 Example Code

- Below is an example on PIC24/PIC32/dsPIC devices which shows how to create a file, write a file, read a file, close a file...etc...

```
FSFILE * pointer;
char sendBuffer[] = "This is test string 1";
char receiveBuffer[50];

// Wait in while loop until the physical media device like SD card, CF card or
// USB memory device is detected in the software...
while (!MDD_MediaDetect());

// Initialize the file system library & the physical media device
while (!FSInit());

// Create a file
pointer = FSfopen ("FILE1.TXT", "w");
if (pointer == NULL)
   while(1);

// Write 21 1-byte objects from sendBuffer into the file
if (FSfwrite (sendBuffer, 1, 21, pointer) != 21)
   while(1);

// Close the file
if (FSfclose (pointer))
   while(1);

// Open file 1 in read mode
pointer = FSfopen ("FILE1.TXT", "r");
if (pointer == NULL)
   while(1);

// Renames the file FILE1.TXT to Microchip File 2.TXT
if (FSrename ("Microchip File 2.TXT", pointer))
   while(1);

// Read one four-byte object
if (FSfread (receiveBuffer, 4, 1, pointer) != 1)
   while(1);

// Close the file
if (FSfclose (pointer))
   while(1);
```

- Below is an example on PIC24/PIC32/dsPIC devices which shows how to create a directory, change the current working directory, delete the directory...etc...

```
// Wait in while loop until the physical media device like SD card, CF card or
// USB memory device is detected in the software...
while (!MDD_MediaDetect());

// Initialize the file system library & the physical media device
while (!FSInit());

// Create a small directory tree
// Beginning the path string with a '.' will create the directory tree in the
// current directory.
if (FSmkdir (".\\Mchp Directory 1\\Dir2\\Directory 3"))
   while(1);

// Change to current working directory to 'Directory 3'
if (FSchdir ("Mchp Directory 1\\Dir2\\Directory 3"))
   while(1);
```

```
// Create another tree in 'Directory 3'
if (FSmkdir ("Directory 4\\Directory 5\\Directory 6"))
   while(1);

// This will delete Directory 5 and all three of its sub-directories
if (FSrmdir ("Directory 4\\Directory 5", TRUE))
   while(1);

// Change directory to the root dir
if (FSchdir ("\\"))
   while(1);
```

- Below is an example on PIC24/PIC32/dsPIC devices which shows how to create files,search files & delete specific file

```
FSFILE * pointer;
char sendBuffer[] = "This is test string 1";
unsigned char attributes;
unsigned char size = 0, i;

// Wait in while loop until the physical media device like SD card, CF card or
// USB memory device is detected in the software...
while (!MDD_MediaDetect());

// Initialize the file system library & the physical media device
while (!FSInit());

// Create a file FILE1.TXT
pointer = FSfopen ("FILE1.TXT", "w");
if (pointer == NULL)
   while(1);

// Write 21 1-byte objects from sendBuffer into the file
if (FSfwrite (sendBuffer, 1, 21, pointer) != 21)
   while(1);

// Close the file
if (FSfclose (pointer))
   while(1);

// Create a file FILE2.TXT
pointer = FSfopen ("FILE2.TXT", "w");
if (pointer == NULL)
   while(1);

// Write 21 1-byte objects from sendBuffer into the file
if (FSfwrite (sendBuffer, 1, 21, pointer) != 21)
   while(1);

// Close the file
if (FSfclose (pointer))
   while(1);

// Create a file FILE3.TXT
pointer = FSfopen ("FILE3.TXT", "w");
if (pointer == NULL)
   while(1);

// Write 21 1-byte objects from sendBuffer into the file
if (FSfwrite (sendBuffer, 1, 21, pointer) != 21)
   while(1);

// Close the file
if (FSfclose (pointer))
   while(1);

// Set attributes
attributes = ATTR_ARCHIVE | ATTR_READ_ONLY | ATTR_HIDDEN;

// Functions "FindFirst" & "FindNext" can be used to find files
```

```
// and directories with required attributes in the current working directory.

// Find the first .TXT file with any (or none) of those attributes that
// has a name beginning with the letters "FILE" in your current working
// directory
if (FindFirst ("FILE*.TXT", attributes, &rec))
   while(1);

// Keep finding files until we get FILE2.TXT
while(rec.filename[4] != '2')
{
   if (FindNext (&rec))
      while(1);
}

// Delete FILE2.TXT
if (FSremove (rec.filename))
   while(1);
```

Note:

- For UTF16 file names 'wFSfopen','wFSrename', 'wFSmkdir', wFSchdir','wFSrmdir'...etc...has to be used instead of 'FSfopen','FSrename', 'FSmkdir', FSchdir','FSrmdir'...
- The LFN support can be enabled by defining the "SUPPORT_LFN" macro in 'FSconfig.h' file. If "SUPPORT_LFN" macro is not defined, then only 8.3 format filename is supported.
- For further clarifications, please refer "MDD File System-SD Card" demo source code. By default it is installed on "C:\Microchip Solutions" folder.

5 Configuring the library

Library configuration is stored as a set of configuration macros in FSconfig.h and HardwareProfiles.h in the demo application folders. These macros has to be modified as per the application requirement.

FSconfig.h

This file contains options to configure the library firmware. The configuration macros include:

Macro/Option	Category	Indication
FS_MAX_FILES_OPEN	Definition	Describes the maximum number of files that can/will be opened at once.
SUPPORT_LFN	Definition	When enabled, the file system supports Long File Name entries for files and directories.
MEDIA_SECTOR_SIZE	Definition	Describes the size of a sector on the device. This will almost always equal 512.
ALLOW_FILESEARCH	Feature toggle	Comment this definition out to disable the file search functions (FindFirst and FindNext). This will reduce code size.
ALLOW_WRITES	Feature toggle	Comment this definition out to disable all write functionality. This will reduce code size.
ALLOW_FORMATS	Feature toggle	Comment this definition out to disable the format function. This will reduce code size.
ALLOW_DIRS	Feature toggle	Comment this definition out to disable all directory functionality. This will reduce code size.
ALLOW_PGMFUNCTIONS	Feature toggle	Comment this definition out to disable -pgm functions. The library requires -pgm functions to be disabled when not using PIC18. This will reduce code size.
ALLOW_FSFPRINTF	Feature toggle	Comment this definition out to disable the FSfprintf function. This will reduce code size.
SUPPORT_FAT32	Feature toggle	Comment this definition out to disable FAT32 support. FAT12 and FAT16 will still be supported.
USEREALTIMECLOCK	Create/last modified timestamp generator	Uncomment this macro to generate timestamps automatically with the RTCC module. You must configure the RTCC for this method to work correctly. Only one timestamp generation method may be enabled at one time.
USERDEFINEDCLOCK	Create/last modified timestamp generator	Uncomment this macro to generate timestamps based on global variabled that are set manually by the user using the SetClockVars() function. Only one timestamp generation method may be enabled at one time.
INCREMENTTIMESTAMP	Create/last modified timestamp generator	Uncomment this macro to generate static timestamps. These timestamps will be incremented by 1 whenever the file is accessed. This should only be used in applications when create.modified times are not required. Only one timestamp generation method may be enabled at one time.
FS_DYNAMIC_MEM	Static/dynamic FSFILE object allocation.	Set the #if preprocessor definition to 1 to allocate FSFILE objects dynamically. You will be required to allocate a heap to do this. For PIC18, you will be required to include the salloc.c and salloc.h files in your project. If the #if statement is set to 0, FSFILE objects will be allocated in a static array, with the maximum number of FSFILE objects determined by the FS_MAX_FILES_OPEN macro.

HardwareProfiles.h

The HardwareProfiles.h header file reflects the state of the hardware. It contains the following macros:

Macro	Indication
GetSystemClock()	Returns the value of the system clock.
GetPeripheralClock()	Returns the value of the microcontroller's peripheral clock
GetInstructionClock()	Returns the value of the microcontroller's instruction clock
USE_SD_INTERFACE_WITH_SPI	Uncomment this definition to use the SD-SPI physical layer. Only one physical layer may be enabled at one time.
USE_CF_INTERFACE_WITH_PMP	Uncomment this definition to use the CF-PMP physical layer. Only one physical layer may be enabled at one time.
USE_MANUAL_CF_INTERFACE	Uncomment this definition to use the CF-Manual physical layer. Only one physical layer may be enabled at one time.
USE_USB_INTERFACE	Uncomment this definition to use the USB host physical layer. This physical layer is described in greater detail at http://www.microchip.com/usb. Only one physical layer may be enabled at one time.
SD_CS, SD_CD, SD_WE	Used for the SD-SPI physical layer. Set these to the I/O port register locations for the chip select, card detect, and write protect signals (e.g. PORTBbits.RB3).
SD_CS_TRIS, SD_CD_TRIS, SD_WE_TRIS	Used for the SD-SPI physical layer. Set these to the I/O tris register locations that correspond to the pins used for each signal (e.g. TRISBbits.TRISB3).
SPICON1, SPISTAT, SPIBUF, SPISTAT_RBF, SPICON1bits, SPISTATbits, SPI_INTERRUPT_FLAG, SPIENABLE	Used for the SD-SPI physical layer. Set these to the SPI registers or bits that correspond to the module you're using (e.g. SSP1CON1, SSP1STAT, SSP1BUF, SSP1STATbits.BF, SSP1CON1bits, SSP1STATbits, PIR1bits.SSPIF).
SPICLOCK, SPIIN, SPIOUT, SPICLOCKLAT, SPIINLAT, SPIOUTLAT, SPICLOCKPORT, SPIINPORT, SPIOUTPORT	Used for the SD-SPI physical layer. Set these to the SPI tris/lat/port register bits for the module you're using.
CF_PMP_RST, CF_PMP_RDY, CF_PMP_CD1	Used with the CF-PMP physical layer. Set these to the I/O port register locations for the reset, ready, and card detect signals for your card.
CF_PMP_RESETDIR, CF_PMP_READYDIR, CF_PMP_CD1DIR	Used with the CF-PMP physical layer. Set these to the I/O tris register that corresponds to the reset, ready, and card detect signals.
MDD_CFPMP_DATADIR	Used with the CF-PMP physical layer. Set this to the tris register that corresponds to the PMP data bus.
ADDBL, ADDDIR	Used with the CF-Manual physical layer. Set these to the lat and tris registers that correspond to the address bus (PIC18).
ADDR0, ADDR1, ADDR2, ADDR3	Used with the CF-Manual physical layer. Set these to the 4 lat pins used for your address bus.
ADRTRIS0, ADRTRIS1, ADRTRIS2, ADRTRIS3	Used with the CF-Manual physical layer. Set these to the corresponding tris bits for your data bus.
MDD_CFBT_DATABIN, MDD_CFBT_DATABOUT, MDD_CFBT_DATADIR	Used with the CF-Manual physical layer. Set these to the port, lat, and tris registers that correspond to your data bus.
CF_CE, CF_OE, CF_WE, CF_BT_RST, CF_BT_RDY, CF_BT_CD1	Used with the CF-Manual physical layer. Set these to the I/O lat and port bits that correspond to the chip select, output enable strobe, write enable strobe, reset, ready, and card detect signals, respectively.
CF_CEDIR, CF_OEDIR, CF_WEDIR, CF_BT_RESETDIR, CF_BT_READYDIR, CF_BT_CD1DIR	Used with the CF-Manual physical layer. Set these to tris bits that correspond to the control signals for the card.

6 APIs

The file system performs file operations on one of the physical interface device like SD card ,CF cards or USB. The APIs of file manipulation layer,SD card layer & CF cards layer is described in this section. The file system operations on USB as host is provided in AN1145: Using a USB Flash Drive with an Embedded Host page.

6.1 File Manipulation Layer (FSIO)

The File Manipulation Layer contains functions for manipulating files or functions to access the device that are common across all physical layers.

6.1.1 Functions

Functions

	Name	Description
	FSInit	Function to initialize the device.
	FSfopen	Opens a file with ascii input 'fileName' on PIC24/PIC32/dsPIC MCU's.
	FSfopenpgm	Opens a file on PIC18 Microcontrollers where 'fileName' ROM string is given in Ascii format.
	wFSfopen	Opens a file with UTF16 input 'fileName' on PIC24/PIC32/dsPIC MCU's.
	FSrename	Renames the Ascii name of the file or directory on PIC24/PIC32/dsPIC devices
	FSrenamepgm	Renames the file with the ascii ROM string(PIC18)
	wFSrename	Renames the name of the file or directory to the UTF16 input fileName on PIC24/PIC32/dsPIC devices
	FSremove	Deletes the file on PIC24/PIC32/dsPIC device.The 'fileName' is in ascii format.
	FSremovepgm	Deletes the file on PIC18 device
	wFSremove	Deletes the file on PIC24/PIC32/dsPIC device.The 'fileName' is in UTF16 format.
	FindFirst	Initial search function for the input Ascii fileName on PIC24/PIC32/dsPIC devices.
	FindFirstpgm	Find a file named with a ROM string on PIC18
	wFindFirst	Initial search function for the 'fileName' in UTF16 format on PIC24/PIC32/dsPIC devices.
	FindNext	Sequential search function
	FSfwrite	Write data to a file
	FSfread	Read data from a file
	FSfseek	Change the current position in a file
	FSftell	Determine the current location in a file
	FSfclose	Update file information and free FSFILE objects
	FSfeof	Indicate whether the current file position is at the end
	FSerror	Return an error code for the last function call
	FSattrib	Change the attributes of a file
	FSfprintf	Function to write formatted strings to a file

	FSrewind	Set the current position in a file to the beginning
	FSformat	Formats a device
	FSCreateMBR	Creates a master boot record
	FSGetDiskProperties	Allows user to get the disk properties (size of disk, free space, etc)
	FSgetcwd	Get the current working directory path in Ascii format
	wFSgetcwd	Get the current working directory path in UTF16 format
	FSmkdir	Creates a directory as per the ascii input path (PIC24/PIC32/dsPIC)
	FSmkdirpgm	Creates a directory as per the path mentioned in the input string on PIC18 devices.
	wFSmkdir	Creates a directory as per the UTF16 input path (PIC24/PIC32/dsPIC)
	FSchdir	Changes the current working directory to the ascii input path(PIC24/PIC32/dsPIC)
	FSchdirpgm	Changes the CWD to the input path on PIC18
	wFSchdir	Change the current working directory as per the path specified in UTF16 format (PIC24/PIC32/dsPIC)
	FSrmdir	Deletes the directory as per the ascii input path (PIC24/PIC32/dsPIC).
	FSrmdirpgm	Deletes the directory as per the ascii input path (PIC18).
	wFSrmdir	Deletes the directory as per the UTF16 input path (PIC24/PIC32/dsPIC).

Macros

	Name	Description
	intmax_t	A data type indicating the maximum integer size in an architecture

Description

The following functions are available for the user application.

6.1.1.1 FSInit

Function to initialize the device.

File

FSIO.h

C

```
int FSInit();
```

Side Effects

The FSerrno variable will be changed.

Description

This function initializes the file system stack & the interfacing device. Initializes the static or dynamic memory slots for holding file structures. Initializes the device with the DISKmount function. Loads MBR and boot sector information. Initializes the current working directory to the root directory for the device if directory support is enabled.

Remarks

None

Preconditions

The physical device should be connected to the microcontroller.

Return Values

Return Values	Description
TRUE	Initialization successful
FALSE	Initialization unsuccessful

Function

int FSInit(void)

6.1.1.2 FSfopen

Opens a file with ascii input 'fileName' on PIC24/PIC32/dsPIC MCU's.

File

FSIO.h

C

```
FSFILE * FSfopen(
    const char * fileName,
    const char * mode
);
```

Side Effects

The FSerrno variable will be changed.

Description

This function will open a ascii name file or directory on PIC24/PIC32/dsPIC MCU's. First, RAM in the dynamic heap or static array will be allocated to a new FSFILE object. Then, the specified file name will be formatted to ensure that it's in 8.3 format or LFN format. Next, the FILEfind function will be used to search for the specified file name. If the name is found, one of three things will happen: if the file was opened in read mode, its file info will be loaded using the FILEopen function; if it was opened in write mode, it will be erased, and a new file will be constructed in its place; if it was opened in append mode, its file info will be loaded with FILEopen and the current location will be moved to the end of the file using the FSfseek function. If the file was not found by FILEfind, a new file will be created if the mode was specified as a write or append mode. In these cases, a pointer to the heap or static FSFILE object array will be returned. If the file was not found and the mode was specified as a read mode, the memory allocated to the file will be freed and the NULL pointer value will be returned.

Remarks

None.

Preconditions

For read modes, file exists; FSInit performed

Parameters

Parameters	Description
fileName	The name of the file to open
mode	• FS_WRITE - Create a new file or replace an existing file • FS_READ - Read data from an existing file • FS_APPEND - Append data to an existing file • FS_WRITEPLUS - Create a new file or replace an existing file (reads also enabled) • FS_READPLUS - Read data from an existing file (writes also enabled) • FS_APPENDPLUS - Append data to an existing file (reads also enabled)

Return Values

Return Values	Description
FSFILE *	The pointer to the file object
NULL	The file could not be opened

Function

FSFILE * FSfopen (const char * fileName, const char *mode)

6.1.1.3 FSfopenpgm

Opens a file on PIC18 Microcontrollers where 'fileName' ROM string is given in Ascii format.

File

FSIO.h

C

```
FSFILE * FSfopenpgm(
    const rom char * fileName,
    const rom char * mode
);
```

Side Effects

The FSerrno variable will be changed.

Description

This function opens a file on PIC18 Microcontrollers where 'fileName' ROM string is given in Ascii format.The FSfopenpgm function will copy a PIC18 ROM fileName and mode argument into RAM arrays, and then pass those arrays to the FSfopen function.

Remarks

This function is for use with PIC18 when passing arguments in ROM.

Preconditions

For read modes, file exists; FSInit performed

Parameters

Parameters	Description
fileName	The name of the file to be opened (ROM)
mode	The mode the file will be opened in (ROM)

Return Values

Return Values	Description
FSFILE *	A pointer to the file object
NULL	File could not be opened

Function

FSFILE * FSfopenpgm(const rom char * fileName, const rom char *mode)

6.1.1.4 wFSfopen

Opens a file with UTF16 input 'fileName' on PIC24/PIC32/dsPIC MCU's.

File

FSIO.h

C

```
FSFILE * wFSfopen(
    const unsigned short int * fileName,
    const char * mode
);
```

Side Effects

The FSerrno variable will be changed.

Description

This function opens a file with UTF16 input 'fileName' on PIC24/PIC32/dsPIC MCU's. First, RAM in the dynamic heap or static array will be allocated to a new FSFILE object. Then, the specified file name will be formatted to ensure that it's in 8.3 format or LFN format. Next, the FILEfind function will be used to search for the specified file name. If the name is found, one of three things will happen: if the file was opened in read mode, its file info will be loaded using the FILEopen function; if it was opened in write mode, it will be erased, and a new file will be constructed in its place; if it was opened in append mode, its file info will be loaded with FILEopen and the current location will be moved to the end of the file using the FSfseek function. If the file was not found by FILEfind, a new file will be created if the mode was specified as a write or append mode. In these cases, a pointer to the heap or static FSFILE object array will be returned. If the file was not found and the mode was specified as a read mode, the memory allocated to the file will be freed and the NULL pointer value will be returned.

Remarks

None.

Preconditions

For read modes, file exists; FSInit performed

Parameters

Parameters	Description
fileName	The name of the file to open
mode	• FS_WRITE - Create a new file or replace an existing file • FS_READ - Read data from an existing file • FS_APPEND - Append data to an existing file • FS_WRITEPLUS - Create a new file or replace an existing file (reads also enabled) • FS_READPLUS - Read data from an existing file (writes also enabled) • FS_APPENDPLUS - Append data to an existing file (reads also enabled)

Return Values

Return Values	Description
FSFILE *	The pointer to the file object
NULL	The file could not be opened

Function

FSFILE * wFSfopen (const unsigned short int * fileName, const char *mode)

6

6.1.1.5 FSrename

Renames the Ascii name of the file or directory on PIC24/PIC32/dsPIC devices

File

FSIO.h

C

```
int FSrename(
    const char * fileName,
    FSFILE * fo
);
```

Side Effects

The FSerrno variable will be changed.

Description

Renames the Ascii name of the file or directory on PIC24/PIC32/dsPIC devices. First, it will search through the current working directory to ensure the specified new filename is not already in use. If it isn't, the new filename will be written to the file entry of the file pointed to by 'fo.'

Remarks

None

Preconditions

File opened.

Parameters

Parameters	Description
fileName	The new name of the file
fo	The file to rename

Return Values

Return Values	Description
0	File was renamed successfully
EOF	File was not renamed

Function

int FSrename (const rom char * fileName, FSFILE * fo)

6.1.1.6 FSrenamepgm

Renames the file with the ascii ROM string(PIC18)

File

FSIO.h

C

```
int FSrenamepgm(
    const rom char * fileName,
    FSFILE * fo
);
```

Side Effects

The FSerrno variable will be changed.

Description

Renames the file with the ascii ROM string(PIC18).The Fsrenamepgm function will copy the rom fileName specified by the user into a RAM array and pass that array into the FSrename function.

Remarks

This function is for use with PIC18 when passing arguments in ROM.

Preconditions

File opened.

Parameters

Parameters	Description
fileName	The new name of the file (in ROM)

fo	The file to rename

Return Values

Return Values	Description
0	File renamed successfully
-1	File could not be renamed

Function

int FSrenamepgm(const rom char * fileName, FSFILE * fo)

6.1.1.7 wFSrename

Renames the name of the file or directory to the UTF16 input fileName on PIC24/PIC32/dsPIC devices

File

FSIO.h

C

```
int wFSrename(
    const unsigned short int * fileName,
    FSFILE * fo
);
```

Side Effects

The FSerrno variable will be changed.

Description

Renames the name of the file or directory to the UTF16 input fileName on PIC24/PIC32/dsPIC devices. First, it will search through the current working directory to ensure the specified new UTF16 filename is not already in use. If it isn't, the new filename will be written to the file entry of the file pointed to by 'fo.'

Remarks

None

Preconditions

File opened.

Parameters

Parameters	Description
fileName	The new name of the file
fo	The file to rename

Return Values

Return Values	Description
0	File was renamed successfully
EOF	File was not renamed

Function

int wFSrename (const rom unsigned short int * fileName, FSFILE * fo)

6.1.1.8 FSremove

Deletes the file on PIC24/PIC32/dsPIC device.The 'fileName' is in ascii format.

File

FSIO.h

C

```
int FSremove(
    const char * fileName
);
```

Side Effects

The FSerrno variable will be changed.

Description

Deletes the file on PIC24/PIC32/dsPIC device.The 'fileName' is in ascii format. The FSremove function will attempt to find the specified file with the FILEfind function. If the file is found, it will be erased using the FILEerase function. The user can also provide ascii alias name of the ascii long file name as the input to this function to get it erased from the memory.

Remarks

None

Preconditions

File not opened, file exists

Parameters

Parameters	Description
fileName	Name of the file to erase

Return Values

Return Values	Description
0	File removed
EOF	File was not removed

Function

int FSremove (const char * fileName)

6.1.1.9 FSremovepgm

Deletes the file on PIC18 device

File

FSIO.h

C

```
int FSremovepgm(
    const rom char * fileName
);
```

Side Effects

The FSerrno variable will be changed.

Description

Deletes the file on PIC18 device.The FSremovepgm function will copy a PIC18 ROM fileName argument into a RAM array, and then pass that array to the FSremove function.

Remarks

This function is for use with PIC18 when passing arguments in ROM.

Preconditions

File not opened; file exists

Parameters

Parameters	Description
fileName	The name of the file to be deleted (ROM)

Return Values

Return Values	Description
0	File was removed successfully
-1	File could not be removed

Function

int FSremovepgm (const rom char * fileName)

6.1.1.10 wFSremove

Deletes the file on PIC24/PIC32/dsPIC device.The 'fileName' is in UTF16 format.

File

FSIO.h

C

```
int wFSremove(
    const unsigned short int * fileName
);
```

Side Effects

The FSerrno variable will be changed.

Description

Deletes the file on PIC24/PIC32/dsPIC device.The 'fileName' is in UTF16 format. The wFSremove function will attempt to find the specified UTF16 file name with the FILEfind function. If the file is found, it will be erased using the FILEerase function.

Remarks

None

Preconditions

File not opened, file exists

Parameters

Parameters	Description
fileName	Name of the file to erase

Return Values

Return Values	Description
0	File removed
EOF	File was not removed

Function

int wFSremove (const unsigned short int * fileName)

6.1.1.11 FindFirst

Initial search function for the input Ascii fileName on PIC24/PIC32/dsPIC devices.

File

FSIO.h

C

```
int FindFirst(
    const char * fileName,
    unsigned int attr,
    SearchRec * rec
);
```

Side Effects

Search criteria from previous FindFirst call on passed SearchRec object will be lost. "utf16LFNfound" is overwritten after subsequent FindFirst/FindNext operations.It is the responsibility of the application to read the "utf16LFNfound" before it is lost.The FSerrno variable will be changed.

Description

Initial search function for the input Ascii fileName on PIC24/PIC32/dsPIC devices. The FindFirst function will search for a file based on parameters passed in by the user. This function will use the FILEfind function to parse through the current working directory searching for entries that match the specified parameters. If a file is found, its parameters are copied into the SearchRec structure, as are the initial parameters passed in by the user and the position of the file entry in the current working directory.If the return value of the function is 0 then "utf16LFNfoundLength" indicates whether the file found was long file name or short file name(8P3 format). The "utf16LFNfoundLength" is non-zero for long file name and is zero for 8P3 format."utf16LFNfound" points to the address of long file name if found during the operation.

Remarks

Call FindFirst or FindFirstpgm before calling FindNext

Preconditions

None

Parameters

Parameters	Description
fileName	The name to search for • Parital string search characters • * - Indicates the rest of the filename or extension can vary (e.g. FILE.*) • ? - Indicates that one character in a filename can vary (e.g. F?LE.T?T)
attr	The attributes that a found file may have • ATTR_READ_ONLY - File may be read only • ATTR_HIDDEN - File may be a hidden file • ATTR_SYSTEM - File may be a system file • ATTR_VOLUME - Entry may be a volume label • ATTR_DIRECTORY - File may be a directory • ATTR_ARCHIVE - File may have archive attribute • ATTR_MASK - All attributes
rec	pointer to a structure to put the file information in

Return Values

Return Values	Description
0	File was found
-1	No file matching the specified criteria was found

Function

int FindFirst (const char * fileName, unsigned int attr, SearchRec * rec)

6.1.1.12 FindFirstpgm

Find a file named with a ROM string on PIC18

File

FSIO.h

C

```
int FindFirstpgm(
    const rom char * fileName,
    unsigned int attr,
    SearchRec * rec
);
```

Side Effects

Search criteria from previous FindFirstpgm call on passed SearchRec object will be lost.The FSerrno variable will be changed.

Description

This function finds a file named with 'fileName' on PIC18. The FindFirstpgm function will copy a PIC18 ROM fileName argument into a RAM array, and then pass that array to the FindFirst function.

Remarks

Call FindFirstpgm or FindFirst before calling FindNext. This function is for use with PIC18 when passing arguments in ROM.

Preconditions

None

Parameters

Parameters	Description
fileName	The name of the file to be found (ROM)
attr	The attributes of the file to be found
rec	Pointer to a search record to store the file info in

Return Values

Return Values	Description
0	File was found
-1	No file matching the given parameters was found

Function

int FindFirstpgm (const char * fileName, unsigned int attr, SearchRec * rec)

6.1.1.13 wFindFirst

Initial search function for the 'fileName' in UTF16 format on PIC24/PIC32/dsPIC devices.

File

FSIO.h

C

```
int wFindFirst(
    const unsigned short int * fileName,
    unsigned int attr,
    SearchRec * rec
);
```

Side Effects

Search criteria from previous wFindFirst call on passed SearchRec object will be lost. "utf16LFNfound" is overwritten after subsequent wFindFirst/FindNext operations.It is the responsibility of the application to read the "utf16LFNfound" before it is lost.The FSerrno variable will be changed.

Description

Initial search function for the 'fileName' in UTF16 format on PIC24/PIC32/dsPIC devices. The wFindFirst function will search for a file based on parameters passed in by the user. This function will use the FILEfind function to parse through the current working directory searching for entries that match the specified parameters. If a file is found, its parameters are copied into the SearchRec structure, as are the initial parameters passed in by the user and the position of the file entry in the current working directory.If the return value of the function is 0 then "utf16LFNfoundLength" indicates whether the file found was long file name or short file name(8P3 format). The "utf16LFNfoundLength" is non-zero for long file name and is zero for 8P3 format."utf16LFNfound" points to the address of long file name if found during the operation.

Remarks

Call FindFirst or FindFirstpgm before calling FindNext

Preconditions

None

Parameters

Parameters	Description
fileName	The name to search for • Parital string search characters • * - Indicates the rest of the filename or extension can vary (e.g. FILE.*) • ? - Indicates that one character in a filename can vary (e.g. F?LE.T?T)
attr	The attributes that a found file may have • ATTR_READ_ONLY - File may be read only • ATTR_HIDDEN - File may be a hidden file • ATTR_SYSTEM - File may be a system file • ATTR_VOLUME - Entry may be a volume label • ATTR_DIRECTORY - File may be a directory • ATTR_ARCHIVE - File may have archive attribute • ATTR_MASK - All attributes
rec	pointer to a structure to put the file information in

Return Values

Return Values	Description
0	File was found
-1	No file matching the specified criteria was found

Function

int wFindFirst (const unsigned short int * fileName, unsigned int attr, SearchRec * rec)

6.1.1.14 FindNext

Sequential search function

File

FSIO.h

C

```
int FindNext(
    SearchRec * rec
);
```

Side Effects

Search criteria from previous FindNext call on passed SearchRec object will be lost. "utf16LFNfound" is overwritten after subsequent FindFirst/FindNext operations.It is the responsibility of the application to read the "utf16LFNfound" before it is lost.The FSerrno variable will be changed.

Description

The FindNext function performs the same function as the FindFirst funciton, except it does not copy any search parameters into the SearchRec structure (only info about found files) and it begins searching at the last directory entry offset at which a file was found, rather than at the beginning of the current working directory.If the return value of the function is 0 then "utf16LFNfoundLength" indicates whether the file found was long file name or short file name(8P3 format). The "utf16LFNfoundLength" is non-zero for long file name and is zero for 8P3 format."utf16LFNfound" points to the address of long file name if found during the operation.

Remarks

Call FindFirst or FindFirstpgm before calling this function

Preconditions

None

Parameters

Parameters	Description
rec	The structure to store the file information in

Return Values

Return Values	Description
0	File was found
-1	No additional files matching the specified criteria were found

Function

int FindNext (SearchRec * rec)

6.1.1.15 FSfwrite

Write data to a file

File

FSIO.h

C

```
size_t FSfwrite(
    const void * data_to_write,
    size_t size,
    size_t n,
    FSFILE * stream
);
```

Side Effects

The FSerrno variable will be changed.

Returns

size_t - number of units written

Description

The FSfwrite function will write data to a file. First, the sector that corresponds to the current position in the file will be loaded (if it hasn't already been cached in the global data buffer). Data will then be written to the device from the specified buffer until the specified amount has been written. If the end of a cluster is reached, the next cluster will be loaded, unless the end-of-file flag for the specified file has been set. If it has, a new cluster will be allocated to the file. Finally, the new position and filesize will be stored in the FSFILE object. The parameters 'size' and 'n' indicate how much data to write. 'Size' refers to the size of one object to write (in bytes), and 'n' will refer to the number of these objects to write. The value returned will be equal to 'n' unless an error occured.

Remarks

None.

Preconditions

File opened in FS_WRITE, FS_APPEND, FS_WRITE+, FS_APPEND+, FS_READ+ mode

Parameters

Parameters	Description
data_to_write	Pointer to source buffer
size	Size of units in bytes
n	Number of units to transfer
stream	Pointer to file structure

Function

size_t FSfwrite(const void *data_to_write, size_t size, size_t n, FSFILE *stream)

6.1.1.16 FSfread

Read data from a file

File

FSIO.h

C

```
size_t FSfread(
    void * ptr,
    size_t size,
    size_t n,
    FSFILE * stream
);
```

Side Effects

The FSerrno variable will be changed.

Returns

size_t - number of units read

Description

The FSfread function will read data from the specified file. First, the appropriate sector of the file is loaded. Then, data is read into the specified buffer until the specified number of bytes have been read. When a cluster boundary is reached, a new cluster will be loaded. The parameters 'size' and 'n' indicate how much data to read. 'Size' refers to the size of one object to read (in bytes), and 'n' will refer to the number of these objects to read. The value returned will be equal to 'n' unless an error occured or the user tried to read beyond the end of the file.

Remarks

None.

Preconditions

File is opened in a read mode

Parameters

Parameters	Description
ptr	Destination buffer for read bytes
size	Size of units in bytes
n	Number of units to be read
stream	File to be read from

Function

size_t FSfread(void *ptr, size_t size, size_t n, FSFILE *stream)

6.1.1.17 FSfseek

Change the current position in a file

File

FSIO.h

C

```
int FSfseek(
    FSFILE * stream,
    long offset,
    int whence
);
```

Side Effects

The FSerrno variable will be changed.

Description

The FSfseek function will change the current position in the file to one specified by the user. First, an absolute offset is calculated using the offset and base location passed in by the user. Then, the position variables are updated, and the sector number that corresponds to the new location. That sector is then loaded. If the offset falls exactly on a cluster boundary, a new cluster will be allocated to the file and the position will be set to the first byte of that cluster.

Remarks

None

Preconditions

File opened

Parameters

Parameters	Description
stream	Pointer to file structure
offset	Offset from base location
whence	• SEEK_SET - Seek from start of file • SEEK_CUR - Seek from current location • SEEK_END - Seek from end of file (subtract offset)

Return Values

Return Values	Description
0	Operation successful
-1	Operation unsuccesful

Function

int FSfseek(FSFILE *stream, long offset, int whence)

6.1.1.18 FSftell

Determine the current location in a file

File

FSIO.h

C

```
long FSftell(
    FSFILE * fo
);
```

Side Effects

The FSerrno variable will be changed

Returns

Current location in the file

Description

The FSftell function will return the current position in the file pointed to by 'fo' by returning the 'seek' variable in the FSFILE object, which is used to keep track of the absolute location of the current position in the file.

Remarks

None

Preconditions

File opened

Parameters

Parameters	Description
fo	Pointer to file structure

Function

long FSftell (FSFILE * fo)

6.1.1.19 FSfclose

Update file information and free FSFILE objects

File

FSIO.h

C

```
int FSfclose(
    FSFILE * fo
);
```

Side Effects

The FSerrno variable will be changed.

Description

This function will update the directory entry for the file pointed to by 'fo' with the information contained in 'fo,' including the new file size and attributes. Timestamp information will also be loaded based on the method selected by the user and written to the entry as the last modified time and date. The file entry will then be written to the device. Finally, the memory used for the specified file object will be freed from the dynamic heap or the array of FSFILE objects.

Remarks

A function to flush data to the device without closing the file can be created by removing the portion of this function that frees the memory and the line that clears the write flag.

Preconditions

File opened

Parameters

Parameters	Description
fo	Pointer to the file to close

Return Values

Return Values	Description
0	File closed successfully
EOF	Error closing the file

Function

int FSfclose(FSFILE *fo)

6.1.1.20 FSfeof

Indicate whether the current file position is at the end

File

FSIO.h

C

```
int FSfeof(
    FSFILE * stream
);
```

Side Effects

The FSerrno variable will be changed.

Description

The FSfeof function will indicate that the end-of- file has been reached for the specified file by comparing the absolute location in the file to the size of the file.

Remarks

None.

Preconditions

File is open in a read mode

Parameters

Parameters	Description
stream	Pointer to the target file

Return Values

Return Values	Description
Non-Zero	EOF reached
0	Not at end of File

Function

int FSfeof(FSFILE * stream)

6.1.1.21 FSerror

Return an error code for the last function call

File

FSIO.h

C

```
int FSerror();
```

Side Effects

None.

Description

The FSerror function will return the FSerrno variable. This global variable will have been set to an error value during the last call of a library function.

Remarks

None

Preconditions

The return value depends on the last function called.

Return Values

Return Values	Description
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_PARITION – 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_UNSUPPORTED_SECTOR_SIZE - The number of bytes per sector is unsupported • 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 FS_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 FS_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 current working directory.
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.

Function

int FSerror (void)

6.1.1.22 FSattrib

Change the attributes of a file

File

FSIO.h

C

```
int FSattrib(
    FSFILE * file,
    unsigned char attributes
);
```

Side Effects

The FSerrno variable will be changed.

Description

The FSattrib funciton will set the attributes of the specified file to the attributes passed in by the user. This function will load the file entry, replace the attributes with the ones specified, and write the attributes back. If the specified file is a directory, the directory attribute will be preserved.

Remarks

None

Preconditions

File opened

Parameters

Parameters	Description
file	Pointer to file structure
attributes	The attributes to set for the file • Attribute - Value - Indications • 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

Return Values

Return Values	Description
0	Attribute change was successful
-1	Attribute change was unsuccessful

Function

int FSattrib (FSFILE * file, unsigned char attributes)

6.1.1.23 FSfprintf

Function to write formatted strings to a file

File

FSIO.h

C

```
int FSfprintf(
    FSFILE * fptr,
    const rom char * fmt,
    ...
);
```

Side Effects

The FSerrno variable will be changed.

Returns

The number of characters written to the file

Description

Writes a specially formatted string to a file.

Remarks

Consult AN1045 for a full description of how to use format specifiers.

Preconditions

For PIC18, integer promotion must be enabled in the project build options menu. File opened in a write mode.

Parameters

Parameters	Description
fptr	A pointer to the file to write to.
fmt	A string of characters and format specifiers to write to the file
...	Additional arguments inserted in the string by format specifiers

Function

// PIC24/30/33/32

int FSfprintf (FSFILE * fptr, const char * fmt, ...)

// PIC18

int FSfpritnf (FSFILE * fptr, const rom char * fmt, ...)

6.1.1.24 FSrewind

Set the current position in a file to the beginning

File

FSIO.h

C

```
void FSrewind(
    FSFILE * fo
);
```

Side Effects

None.

Description

The FSrewind funciton will reset the position of the specified file to the beginning of the file. This functionality is faster than using FSfseek to reset the position in the file.

Remarks

None.

Preconditions

File opened.

Parameters

Parameters	Description
fo	Pointer to file structure

Function

void FSrewind (FSFILE * fo)

6.1.1.25 FSformat

Formats a device

File

FSIO.h

C

```
int FSformat(
    char mode,
    long int serialNumber,
    char * volumeID
);
```

Side Effects

The FSerrno variable will be changed.

Description

The FSformat function can be used to create a new boot sector on a device, based on the information in the master boot record. This function will first initialize the I/O pins and the device, and then attempts to read the master boot record. If the MBR cannot be loaded successfully, the function will fail. Next, if the 'mode' argument is specified as '0' the existing boot sector information will be loaded. If the 'mode' argument is '1' an entirely new boot sector will be constructed using the disk values from the master boot record. Once the boot sector has been successfully loaded/created, the locations of the FAT and root will be loaded from it, and they will be completely erased. If the user has specified a volumeID parameter, a VOLUME attribute entry will be created in the root directory to name the device.

FAT12, FAT16 and FAT32 formatting are supported.

Based on the number of sectors, the format function automatically compute the smallest possible value for the cluster size in order to accommodate the physical size of the media. In this case, if a media with a big capacity is formatted, the format function may take a very long time to write all the FAT tables.

Therefore, the FORMAT_SECTORS_PER_CLUSTER macro may be used to specify the exact cluster size (in multiples of sector size). This macro can be defined in FSconfig.h

Remarks

Only devices with a sector size of 512 bytes are supported by the format function

Preconditions

The device must possess a valid master boot record.

Parameters

Parameters	Description
mode	• 0 - Just erase the FAT and root • 1 - Create a new boot sector
serialNumber	Serial number to write to the card
volumeID	Name of the card

Return Values

Return Values	Description
0	Format was successful
EOF	Format was unsuccessful

Function

int FSformat (char mode, long int serialNumber, char * volumeID)

6.1.1.26 FSCreateMBR

Creates a master boot record

File

FSIO.h

C

```
int FSCreateMBR(
    unsigned long firstSector,
    unsigned long numSectors
);
```

Side Effects

None

Description

This function can be used to create a master boot record for a device. Note that this function should not be used on a device that is already formatted with a master boot record (i.e. most SD cards, CF cards, USB keys). This function will fill the global data buffer with appropriate partition information for a FAT partition with a type determined by the number of sectors available to the partition. It will then write the MBR information to the first sector on the device. This function should be followed by a call to FSformat, which will create a boot sector, root dir, and FAT appropriate the the information contained in the new master boot record. Note that FSformat only supports FAT12 and FAT16 formatting at this time, and so cannot be used to format a device with more than 0x3FFD5F sectors.

Remarks

This function can damage the device being used, and should not be called unless the user is sure about the size of the device and the first sector value.

Preconditions

The I/O pins for the device have been initialized by the InitIO function.

Parameters

Parameters	Description
firstSector	The first sector of the partition on the device (cannot be 0; that's the MBR)
numSectors	The number of sectors available in memory (including the MBR)

Return Values

Return Values	Description
0	MBR was created successfully
EOF	MBR could not be created

Function

int FSCreateMBR (unsigned long firstSector, unsigned long numSectors)

6.1.1.27 FSGetDiskProperties

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

File

FSIO.h

C

```
void FSGetDiskProperties(
    FS_DISK_PROPERTIES* properties
);
```

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 FS_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 FS_DISK_PROPERTIES object that was passed in to create the search.

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

Typical Usage:

```
FS_DISK_PROPERTIES disk_properties;

disk_properties.new_request = TRUE;

do
{
    FSGetDiskProperties(&disk_properties);
} while (disk_properties.properties_status == FS_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_DISK_PROPERTIES must be defined in FSconfig.h 2) a FS_DISK_PROPERTIES object must be created before the function is called 3) the new_request member of the FS_DISK_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
properties	a pointer to a FS_DISK_PROPERTIES object where the results should be stored.

Return Values

Return Values	Description
following possible values	
FS_GET_PROPERTIES_NO_ERRORS	operation completed without error. Results are in the properties object passed into the function.

FS_GET_PROPERTIES_DISK_NOT_MOUNTED	there is no mounted disk. Results in properties object is not valid
FS_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.
FS_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 FSGetDiskProperties(FS_DISK_PROPERTIES* properties)

6.1.1.28 FSgetcwd

Get the current working directory path in Ascii format

File

FSIO.h

C

```
char * FSgetcwd(
    char * path,
    int numbchars
);
```

Side Effects

The FSerrno variable will be changed

Description

Get the current working directory path in Ascii format. The FSgetcwd function will get the name of the current working directory and return it to the user. The name will be copied into the buffer pointed to by 'path,' starting at the root directory and copying as many chars as possible before the end of the buffer. The buffer size is indicated by the 'numchars' argument. The first thing this function will do is load the name of the current working directory, if it isn't already present. This could occur if the user switched to the dotdot entry of a subdirectory immediately before calling this function. The function will then copy the current working directory name into the buffer backwards, and insert a backslash character. Next, the function will continuously switch to the previous directories and copy their names backwards into the buffer until it reaches the root. If the buffer overflows, it will be treated as a circular buffer, and data will be copied over existing characters, starting at the beginning. Once the root directory is reached, the text in the buffer will be swapped, so that the buffer contains as much of the current working directory name as possible, starting at the root.

Remarks

None

Preconditions

None

Parameters

Parameters	Description
path	Pointer to the array to return the cwd name in
numchars	Number of chars in the path

Return Values

Return Values	Description
char *	The cwd name string pointer (path or defaultArray)
NULL	The current working directory name could not be loaded.

Function

char * FSgetcwd (char * path, int numchars)

6.1.1.29 wFSgetcwd

Get the current working directory path in UTF16 format

File

FSIO.h

C

```
char * wFSgetcwd(
    unsigned short int * path,
    int numbchars
);
```

Side Effects

The FSerrno variable will be changed

Description

Get the current working directory path in UTF16 format. The FSgetcwd function will get the name of the current working directory and return it to the user. The name will be copied into the buffer pointed to by 'path,' starting at the root directory and copying as many chars as possible before the end of the buffer. The buffer size is indicated by the 'numchars' argument. The first thing this function will do is load the name of the current working directory, if it isn't already present. This could occur if the user switched to the dotdot entry of a subdirectory immediately before calling this function. The function will then copy the current working directory name into the buffer backwards, and insert a backslash character. Next, the function will continuously switch to the previous directories and copy their names backwards into the buffer until it reaches the root. If the buffer overflows, it will be treated as a circular buffer, and data will be copied over existing characters, starting at the beginning. Once the root directory is reached, the text in the buffer will be swapped, so that the buffer contains as much of the current working directory name as possible, starting at the root.

Remarks

None

Preconditions

None

Parameters

Parameters	Description
path	Pointer to the array to return the cwd name in
numchars	Number of chars in the path

Return Values

Return Values	Description
char *	The cwd name string pointer (path or defaultArray)
NULL	The current working directory name could not be loaded.

Function

char * wFSgetcwd (unsigned short int * path, int numchars)

6.1.1.30 FSmkdir

Creates a directory as per the ascii input path (PIC24/PIC32/dsPIC)

File

FSIO.h

C

```
int FSmkdir(
    char * path
);
```

Side Effects

Will create all non-existent directories in the path. The FSerrno variable will be changed.

Description

Creates a directory as per the ascii input path (PIC24/PIC32/dsPIC). This function doesn't move the current working directory setting.

Remarks

None

Preconditions

None

Parameters

Parameters	Description
path	The path of directories to create.

Return Values

Return Values	Description
0	The specified directory was created successfully
EOF	The specified directory could not be created

Function

int FSmkdir (char * path)

6.1.1.31 FSmkdirpgm

Creates a directory as per the path mentioned in the input string on PIC18 devices.

File

FSIO.h

C

```
int FSmkdirpgm(
    const rom char * path
);
```

Side Effects

Will create all non-existent directories in the path. The FSerrno variable will be changed.

Description

Creates a directory as per the path mentioned in the input string on PIC18 devices.'FSmkdirpgm' creates the directories as per the input string path.This function doesn't move the current working directory setting.

Remarks

This function is for use with PIC18 when passing arugments in ROM

Preconditions

None

Parameters

Parameters	Description
path	The path of directories to create (ROM)

Return Values

Return Values	Description
0	The specified directory was created successfully
EOF	The specified directory could not be created

Function

int FSmkdirpgm (const rom char * path)

6.1.1.32 wFSmkdir

Creates a directory as per the UTF16 input path (PIC24/PIC32/dsPIC)

File

FSIO.h

C

```
int wFSmkdir(
    unsigned short int * path
);
```

Side Effects

Will create all non-existent directories in the path. The FSerrno variable will be changed.

Description

Creates a directory as per the UTF16 input path (PIC24/PIC32/dsPIC). This function doesn't move the current working directory setting.

Remarks

None

Preconditions

None

Parameters

Parameters	Description
path	The path of directories to create.

Return Values

Return Values	Description
0	The specified directory was created successfully
EOF	The specified directory could not be created

Function

int wFSmkdir (unsigned short int * path)

6.1.1.33 FSchdir

Changes the current working directory to the ascii input path(PIC24/PIC32/dsPIC)

File

FSIO.h

C

```
int FSchdir(
    char * path
);
```

Side Effects

The current working directory may be changed. The FSerrno variable will be changed.

Description

Changes the current working directory to the ascii input path(PIC24/PIC32/dsPIC). The FSchdir function passes a RAM pointer to the path to the chdirhelper function.

Remarks

None

Preconditions

None

Parameters

Parameters	Description
path	The path of the directory to change to.

Return Values

Return Values	Description
0	The current working directory was changed successfully
EOF	The current working directory could not be changed

Function

int FSchdir (char * path)

6.1.1.34 FSchdirpgm

Changes the CWD to the input path on PIC18

File

FSIO.h

C

```
int FSchdirpgm(
    const rom char * path
);
```

Side Effects

The current working directory may be changed. The FSerrno variable will be changed.

Description

Changes the CWD to the input path on PIC18.The FSchdirpgm function passes a PIC18 ROM path pointer to the chdirhelper function.

Remarks

This function is for use with PIC18 when passing arguments in ROM

Preconditions

None

Parameters

Parameters	Description
path	The path of the directory to change to (ROM)

Return Values

Return Values	Description
0	The current working directory was changed successfully
EOF	The current working directory could not be changed

Function

int FSchdirpgm (const rom char * path)

6.1.1.35 wFSchdir

Change the current working directory as per the path specified in UTF16 format (PIC24/PIC32/dsPIC)

File

FSIO.h

C

```
int wFSchdir(
    unsigned short int * path
);
```

Side Effects

The current working directory may be changed. The FSerrno variable will be changed.

Description

Change the current working directory as per the path specified in UTF16 format (PIC24/PIC32/dsPIC).The FSchdir function passes a RAM pointer to the path to the chdirhelper function.

Remarks

None

Preconditions

None

Parameters

Parameters	Description
path	The path of the directory to change to.

Return Values

Return Values	Description
0	The current working directory was changed successfully
EOF	The current working directory could not be changed

Function

int wFSchdir (unsigned short int * path)

6.1.1.36 FSrmdir

Deletes the directory as per the ascii input path (PIC24/PIC32/dsPIC).

File

FSIO.h

C

```
int FSrmdir(
    char * path,
    unsigned char rmsubdirs
);
```

Side Effects

The FSerrno variable will be changed.

Description

Deletes the directory as per the ascii input path (PIC24/PIC32/dsPIC). This function wont delete the current working directory.

Remarks

None.

Preconditions

None

Parameters

Parameters	Description
path	The path of the directory to remove
rmsubdirs	• TRUE - All sub-dirs and files in the target dir will be removed • FALSE - FSrmdir will not remove non-empty directories

Return Values

Return Values	Description
0	The specified directory was deleted successfully
EOF	The specified directory could not be deleted

Function

int FSrmdir (char * path)

6.1.1.37 FSrmdirpgm

Deletes the directory as per the ascii input path (PIC18).

File

FSIO.h

C

```
int FSrmdirpgm(
    const rom char * path,
    unsigned char rmsubdirs
);
```

Side Effects

The FSerrno variable will be changed.

Description

Deletes the directory as per the ascii input path (PIC18). This function deletes the directory as specified in the path. This function wont delete the current working directory.

Remarks

This function is for use with PIC18 when passing arguments in ROM.

Preconditions

None.

Parameters

Parameters	Description
path	The path of the directory to remove (ROM)
rmsubdirs	• TRUE - All sub-dirs and files in the target dir will be removed • FALSE - FSrmdir will not remove non-empty directories

Return Values

Return Values	Description
0	The specified directory was deleted successfully
EOF	The specified directory could not be deleted

Function

int FSrmdirpgm (const rom char * path)

6.1.1.38 wFSrmdir

Deletes the directory as per the UTF16 input path (PIC24/PIC32/dsPIC).

File

FSIO.h

C

```
int wFSrmdir(
    unsigned short int * path,
    unsigned char rmsubdirs
);
```

Side Effects

The FSerrno variable will be changed.

Description

Deletes the directory as per the UTF16 input path (PIC24/PIC32/dsPIC). This function wont delete the current working directory.

Remarks

None.

Preconditions

None

Parameters

Parameters	Description
path	The path of the directory to remove
rmsubdirs	• TRUE - All sub-dirs and files in the target dir will be removed • FALSE - FSrmdir will not remove non-empty directories

Return Values

Return Values	Description
0	The specified directory was deleted successfully
EOF	The specified directory could not be deleted

Function

int wFSrmdir (unsigned short int * path, unsigned char rmsubdirs)

6.1.1.39 intmax_t

A data type indicating the maximum integer size in an architecture

File

FSIO.h

C

```
#define intmax_t long long
```

Description

The intmax_t data type refers to the maximum-sized data type on any given architecture. This data type can be specified as a format specifier size specification for the FSfprintf function.

6.1.2 Types

Enumerations

	Name	Description
	CETYPE	An enumeration used for various error codes.
	_CETYPE	An enumeration used for various error codes.

Structures

	Name	Description
	FSFILE	Contains file information and is used to indicate which file to access.
	SearchRec	A structure used for searching for files on a device.

Description

The following enum & structure API types are necessary to be known by the user application.

6.1.2.1 CETYPE

An enumeration used for various error codes.

File

FSDefs.h

C

```
typedef enum _CETYPE {
  CE_GOOD = 0,
  CE_ERASE_FAIL,
  CE_NOT_PRESENT,
  CE_NOT_FORMATTED,
  CE_BAD_PARTITION,
  CE_UNSUPPORTED_FS,
```

```
  CE_INIT_ERROR,
  CE_NOT_INIT,
  CE_BAD_SECTOR_READ,
  CE_WRITE_ERROR,
  CE_INVALID_CLUSTER,
  CE_FILE_NOT_FOUND,
  CE_DIR_NOT_FOUND,
  CE_BAD_FILE,
  CE_DONE,
  CE_COULD_NOT_GET_CLUSTER,
  CE_FILENAME_2_LONG,
  CE_FILENAME_EXISTS,
  CE_INVALID_FILENAME,
  CE_DELETE_DIR,
  CE_DIR_FULL,
  CE_DISK_FULL,
  CE_DIR_NOT_EMPTY,
  CE_NONSUPPORTED_SIZE,
  CE_WRITE_PROTECTED,
  CE_FILENOTOPENED,
  CE_SEEK_ERROR,
  CE_BADCACHEREAD,
  CE_CARDFAT32,
  CE_READONLY,
  CE_WRITEONLY,
  CE_INVALID_ARGUMENT,
  CE_TOO_MANY_FILES_OPEN,
  CE_UNSUPPORTED_SECTOR_SIZE
} CETYPE;
```

Members

Members	Description
CE_GOOD = 0	No error
CE_ERASE_FAIL	An erase failed
CE_NOT_PRESENT	No device was present
CE_NOT_FORMATTED	The disk is of an unsupported format
CE_BAD_PARTITION	The boot record is bad
CE_UNSUPPORTED_FS	The file system type is unsupported
CE_INIT_ERROR	An initialization error has occured
CE_NOT_INIT	An operation was performed on an uninitialized device
CE_BAD_SECTOR_READ	A bad read of a sector occured
CE_WRITE_ERROR	Could not write to a sector
CE_INVALID_CLUSTER	Invalid cluster value > maxcls
CE_FILE_NOT_FOUND	Could not find the file on the device
CE_DIR_NOT_FOUND	Could not find the directory
CE_BAD_FILE	File is corrupted
CE_DONE	No more files in this directory
CE_COULD_NOT_GET_CLUSTER	Could not load/allocate next cluster in file
CE_FILENAME_2_LONG	A specified file name is too long to use
CE_FILENAME_EXISTS	A specified filename already exists on the device
CE_INVALID_FILENAME	Invalid file name
CE_DELETE_DIR	The user tried to delete a directory with FSremove
CE_DIR_FULL	All root dir entry are taken
CE_DISK_FULL	All clusters in partition are taken
CE_DIR_NOT_EMPTY	This directory is not empty yet, remove files before deleting
CE_NONSUPPORTED_SIZE	The disk is too big to format as FAT16
CE_WRITE_PROTECTED	Card is write protected
CE_FILENOTOPENED	File not opened for the write

CE_SEEK_ERROR	File location could not be changed successfully
CE_BADCACHEREAD	Bad cache read
CE_CARDFAT32	FAT 32 - card not supported
CE_READONLY	The file is read-only
CE_WRITEONLY	The file is write-only
CE_INVALID_ARGUMENT	Invalid argument
CE_TOO_MANY_FILES_OPEN	Too many files are already open
CE_UNSUPPORTED_SECTOR_SIZE	Unsupported sector size

Description

The CETYPE enumeration is used to indicate different error conditions during device operation.

6.1.2.2 FSFILE

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

File

FSIO.h

C

```
typedef struct {
  DISK * dsk;
  DWORD cluster;
  DWORD ccls;
  WORD sec;
  WORD pos;
  DWORD seek;
  DWORD size;
  FILEFLAGS flags;
  WORD time;
  WORD date;
  char name[FILE_NAME_SIZE_8P3];
  BOOL AsciiEncodingType;
  unsigned short int * utf16LFNptr;
  unsigned short int utf16LFNlength;
  WORD entry;
  WORD chk;
  WORD attributes;
  DWORD dirclus;
  DWORD dirccls;
} FSFILE;
```

Members

Members	Description
DISK * dsk;	Pointer to a DISK structure
DWORD cluster;	The first cluster of the file
DWORD ccls;	The current cluster of the file
WORD sec;	The current sector in the current cluster of the file
WORD pos;	The position in the current sector
DWORD seek;	The absolute position in the file
DWORD size;	The size of the file
FILEFLAGS flags;	A structure containing file flags
WORD time;	The file's last update time
WORD date;	The file's last update date
char name[FILE_NAME_SIZE_8P3];	The short name of the file
BOOL AsciiEncodingType;	Ascii file name or Non-Ascii file name indicator
unsigned short int * utf16LFNptr;	Pointer to long file name in UTF16 format

unsigned short int utf16LFNlength;	LFN length in terms of words including the NULL word at the last.
WORD entry;	The position of the file's directory entry in it's directory
WORD chk;	File structure checksum
WORD attributes;	The file attributes
DWORD dirclus;	The base cluster of the file's directory
DWORD dirccls;	The current cluster of the file's directory

Description

The FSFILE 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 FSFILE structure will be passeed to any library function that will modify that file.

6.1.2.3 SearchRec

A structure used for searching for files on a device.

File

FSIO.h

C

```
typedef struct {
  char filename[FILE_NAME_SIZE_8P3 + 2];
  unsigned char attributes;
  unsigned long filesize;
  unsigned long timestamp;
  BOOL AsciiEncodingType;
  unsigned short int * utf16LFNfound;
  unsigned short int utf16LFNfoundLength;
  unsigned int entry;
  char searchname[FILE_NAME_SIZE_8P3 + 2];
  unsigned char searchattr;
  unsigned long cwdclus;
  unsigned char initialized;
} SearchRec;
```

Members

Members	Description
char filename[FILE_NAME_SIZE_8P3 + 2];	The name of the file that has been found
unsigned char attributes;	The attributes of the file that has been found
unsigned long filesize;	The size of the file that has been found
unsigned long timestamp;	The last modified time of the file that has been found (create time for directories)
BOOL AsciiEncodingType;	Ascii file name or Non-Ascii file name indicator
unsigned short int * utf16LFNfound;	Pointer to long file name found in UTF16 format
unsigned short int utf16LFNfoundLength;	LFN Found length in terms of words including the NULL word at the last.
unsigned int entry;	The directory entry of the last file found that matches the specified attributes. (Internal use only)
char searchname[FILE_NAME_SIZE_8P3 + 2];	The 8.3 format name specified when the user began the search. (Internal use only)
unsigned char searchattr;	The attributes specified when the user began the search. (Internal use only)
unsigned long cwdclus;	The directory that this search was performed in. (Internal use only)
unsigned char initialized;	Check to determine if the structure was initialized by FindFirst (Internal use only)

Description

The SearchRec structure is used when searching for file on a device. It contains parameters that will be loaded with file information when a file is found. It also contains the parameters that the user searched for, allowing further searches to be perfomed in the same directory for additional files that meet the specified criteria.

6.1.3 Macros

Macros

	Name	Description
	MDD_InitIO	Function pointer to the I/O Initialization Physical Layer function
	MDD_MediaInitialize	Function pointer to the Media Initialize Physical Layer function
	MDD_ReadCapacity	Function pointer to the Read Capacity Physical Layer function
	MDD_ReadSectorSize	Function pointer to the Read Sector Size Physical Layer Function
	MDD_SectorRead	Function pointer to the Sector Read Physical Layer function
	MDD_SectorWrite	Function pointer to the Sector Write Physical Layer function
	MDD_ShutdownMedia	Function pointer to the Media Shutdown Physical Layer function
	MDD_WriteProtectState	Function pointer to the Write Protect Check Physical Layer function
	ALLOW_DIRS	A macro to enable/disable directory operations.
	ALLOW_FILESEARCH	A macro to enable/disable file search functions.
	ALLOW_FORMATS	A macro to enable/disable format functionality
	ALLOW_WRITES	A macro to enable/disable write functionality
	FAT12	A macro indicating the device is formatted with FAT12
	FAT16	A macro indicating the device is formatted with FAT16
	FAT32	A macro indicating the device is formatted with FAT32
	FILE_NAME_SIZE_8P3	MAcro indicating the length of an 8.3 file name in a directory entry
	FS_DYNAMIC_MEM	A macro indicating that FSFILE objects will be allocated dynamically
	FS_MAX_FILES_OPEN	A macro indicating the maximum number of concurrently open files
	MAX_FILE_NAME_LENGTH_LFN	Macro indicating the max length of a LFN file name
	MAX_HEAP_SIZE	A macro used to define the heap size for PIC18
	MEDIA_SECTOR_SIZE	A macro defining the size of a sector
	SEEK_SET	Macro for the FSfseek SEEK_SET base location.
	SEEK_CUR	Macro for the FSfseek SEEK_CUR base location.
	SEEK_END	Macro for the FSfseek SEEK_END base location
	SUPPORT_FAT32	A macro to enable/disable FAT32 support.
	SUPPORT_LFN	A macro indicating whether Long File Name is supported
	USE_SD_INTERFACE_WITH_SPI	Macro used to enable the SD-SPI physical layer (SD-SPI.c and .h)
	USEREALTIMECLOCK	A macro to enable RTCC based timestamp generation

Description

The following macros are available for the user application.

6

6.1.3.1 MDD_InitIO

File

FSconfig.h

C

```
#define MDD_InitIO ;
```

Description

Function pointer to the I/O Initialization Physical Layer function

6.1.3.2 MDD_MediaInitialize

File

FSconfig.h

C

```
#define MDD_MediaInitialize USBHostMSDSCSIMediaInitialize
```

Description

Function pointer to the Media Initialize Physical Layer function

6.1.3.3 MDD_ReadCapacity

File

FSconfig.h

C

```
#define MDD_ReadCapacity MDD_SDSPI_ReadCapacity
```

Description

Function pointer to the Read Capacity Physical Layer function

6.1.3.4 MDD_ReadSectorSize

File

FSconfig.h

C

```
#define MDD_ReadSectorSize MDD_SDSPI_ReadSectorSize
```

Description

Function pointer to the Read Sector Size Physical Layer Function

6.1.3.5 MDD_SectorRead

File

FSconfig.h

C

```
#define MDD_SectorRead USBHostMSDSCSISectorRead
```

Description

Function pointer to the Sector Read Physical Layer function

6.1.3.6 MDD_SectorWrite

File

FSconfig.h

C

```
#define MDD_SectorWrite USBHostMSDSCSISectorWrite
```

Description

Function pointer to the Sector Write Physical Layer function

6.1.3.7 MDD_ShutdownMedia

File

FSconfig.h

C

```
#define MDD_ShutdownMedia USBHostMSDSCSIMediaReset
```

Description

Function pointer to the Media Shutdown Physical Layer function

6.1.3.8 MDD_WriteProtectState

File

FSconfig.h

C

```
#define MDD_WriteProtectState USBHostMSDSCSIWriteProtectState
```

Description

Function pointer to the Write Protect Check Physical Layer function

6.1.3.9 ALLOW_DIRS

A macro to enable/disable directory operations.

File

FSconfig.h

C

```
#define ALLOW_DIRS
```

Description

The ALLOW_DIRS definition can be commented out to disable all directory functionality. This will reduce code size. If directories are enabled, write operations must also be enabled by uncommenting ALLOW_WRITES in order to use the FSmkdir or FSrmdir functions.

6.1.3.10 ALLOW_FILESEARCH

A macro to enable/disable file search functions.

File

FSconfig.h

C

```
#define ALLOW_FILESEARCH
```

Description

The ALLOW_FILESEARCH definition can be commented out to disable file search functions in the library. This will prevent the use of the FindFirst and FindNext functions and reduce code size.

6.1.3.11 ALLOW_FORMATS

A macro to enable/disable format functionality

File

FSconfig.h

C

```
#define ALLOW_FORMATS
```

Description

The ALLOW_FORMATS definition can be commented out to disable formatting functionality. This will prevent the use of the FSformat function. If formats are enabled, write operations must also be enabled by uncommenting ALLOW_WRITES.

6.1.3.12 ALLOW_WRITES

A macro to enable/disable write functionality

File

FSconfig.h

C

```
#define ALLOW_WRITES
```

Description

The ALLOW_WRITES definition can be commented out to disable all operations that write to the device. This will greatly reduce code size.

6.1.3.13 FAT12

A macro indicating the device is formatted with FAT12

File

FSDefs.h

C

```
#define FAT12 1
```

Description

The FAT12 macro is used to indicate that the file system on the device being accessed is a FAT12 file system.

6.1.3.14 FAT16

A macro indicating the device is formatted with FAT16

File

FSDefs.h

C

```
#define FAT16 2
```

Description

The FAT16 macro is used to indicate that the file system on the device being accessed is a FAT16 file system.

6.1.3.15 FAT32

A macro indicating the device is formatted with FAT32

File

FSDefs.h

C

```
#define FAT32 3
```

Description

The FAT32 macro is used to indicate that the file system on the device being accessed is a FAT32 file system.

6.1.3.16 FILE_NAME_SIZE_8P3

MAcro indicating the length of an 8.3 file name in a directory entry

File

FSIO.h

C

```
#define FILE_NAME_SIZE_8P3 11
```

Description

The FILE_NAME_SIZE_8P3 macro indicates the number of characters that an 8.3 file name will take up when packed in a directory entry. This value includes 8 characters for the name and 3 for the extension. Note that the radix is not stored in the directory entry.

6.1.3.17 FS_DYNAMIC_MEM

A macro indicating that FSFILE objects will be allocated dynamically

File

FSconfig.h

C

```
#define FS_DYNAMIC_MEM
```

Description

The FS_DYNAMIC_MEM macro will cause FSFILE objects to be allocated from a dynamic heap. If it is undefined, the file objects will be allocated using a static array.

6.1.3.18 FS_MAX_FILES_OPEN

A macro indicating the maximum number of concurrently open files

File

FSconfig.h

C

```
#define FS_MAX_FILES_OPEN 3
```

Description

The FS_MAX_FILES_OPEN #define is only applicable when dynamic memory allocation is not used (FS_DYNAMIC_MEM is not defined). This macro defines the maximum number of open files at any given time. The amount of RAM used by FSFILE objects will be equal to the size of an FSFILE object multipled by this macro value. This value should be kept as small as possible as dictated by the application. This will reduce memory usage.

6.1.3.19 MAX_FILE_NAME_LENGTH_LFN

Macro indicating the max length of a LFN file name

File

FSIO.h

C

```
#define MAX_FILE_NAME_LENGTH_LFN 256
```

Description

The MAX_FILE_NAME_LENGTH_LFN macro indicates the maximum number of characters in an LFN file name.

6.1.3.20 MAX_HEAP_SIZE

A macro used to define the heap size for PIC18

File

salloc.c

C

```
#define MAX_HEAP_SIZE 0x100
```

Description

When using dynamic FSFILE object allocation with PIC18, the MAX_HEAP_SIZE will allow the user to specify the size of the dynamic heap to use

6.1.3.21 MEDIA_SECTOR_SIZE

A macro defining the size of a sector

File

FSconfig.h

C

```
#define MEDIA_SECTOR_SIZE 512
```

Description

The MEDIA_SECTOR_SIZE macro will define the size of a sector on the FAT file system. This value must equal 512 bytes, 1024 bytes, 2048 bytes, or 4096 bytes. The value of a sector will usually be 512 bytes.

6.1.3.22 SEEK_SET

Macro for the FSfseek SEEK_SET base location.

File

FSIO.h

C

```
#define SEEK_SET 0
```

Description

Functions as an input for FSfseek that specifies that the position in the file will be changed relative to the beginning of the file.

6.1.3.23 SEEK_CUR

Macro for the FSfseek SEEK_CUR base location.

File

FSIO.h

C

```
#define SEEK_CUR 1
```

Description

Functions as an input for FSfseek that specifies that the position in the file will be changed relative to the current location of the file

6.1.3.24 SEEK_END

Macro for the FSfseek SEEK_END base location

File

FSIO.h

C

```
#define SEEK_END 2
```

Description

Functions as an input for FSfseek that specifies that the position in the file will be changed relative to the end of the file. For this macro, the offset value will be subtracted from the end location of the file by default.

6.1.3.25 SUPPORT_FAT32

A macro to enable/disable FAT32 support.

File

FSconfig.h

C

```
#define SUPPORT_FAT32
```

Description

The SUPPORT_FAT32 definition can be commented out to disable support for FAT32 functionality. This will save a small amount of code space.

6.1.3.26 SUPPORT_LFN

A macro indicating whether Long File Name is supported

File

FSconfig.h

C

```
#define SUPPORT_LFN
```

Description

If this macro is disabled then only 8.3 format file name is enabled. If this macro is enabled then long file names upto 256 characters are supported.

6.1.3.27 USE_SD_INTERFACE_WITH_SPI

File

HardwareProfile.h

C

```
#define USE_SD_INTERFACE_WITH_SPI
```

Description

Macro used to enable the SD-SPI physical layer (SD-SPI.c and .h)

6.1.3.28 USEREALTIMECLOCK

A macro to enable RTCC based timestamp generation

File

FSconfig.h

C

```
#define USEREALTIMECLOCK
```

Description

The USEREALTIMECLOCK macro will configure the code to automatically generate timestamp information for files from the RTCC module. The user must enable and configure the RTCC module before creating or modifying files.

6.2 SD-SPI Physical Layer

The SD-SPI physical layer offers the ability to interface SD cards using the SPI protocol.

6.2.1 Functions

Functions

	Name	Description
	MDD_SDSPI_MediaDetect	Determines whether an SD card is present
	MDD_SDSPI_InitIO	Initializes the I/O lines connected to the card
	MDD_SDSPI_MediaInitialize	Initializes the SD card.
	MDD_SDSPI_SectorRead	Reads a sector of data from an SD card.
	MDD_SDSPI_SectorWrite	Writes a sector of data to an SD card.
	MDD_SDSPI_ReadSectorSize	Determines the current sector size on the SD card
	MDD_SDSPI_ReadCapacity	Determines the current capacity of the SD card
	MDD_SDSPI_ShutdownMedia	Disables the SD card

Description

The following driver functions are API's for FSIO layer.

6.2.1.1 MDD_SDSPI_MediaDetect

Determines whether an SD card is present

File

SD-SPI.h

C

```
BYTE MDD_SDSPI_MediaDetect();
```

Side Effects

None.

Description

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

Return Values

Return Values	Description
TRUE	Card detected
FALSE	No card detected

Function

BYTE MDD_SDSPI_MediaDetect

6.2.1.2 MDD_SDSPI_InitIO

Initializes the I/O lines connected to the card

File

SD-SPI.h

C

```
void MDD_SDSPI_InitIO();
```

Side Effects

None.

Returns

None

Description

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

Remarks

None

Preconditions

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

Function

WORD MDD_SDSPI_InitIO (void)

6.2.1.3 MDD_SDSPI_MediaInitialize

Initializes the SD card.

File

SD-SPI.c

C

```
MEDIA_INFORMATION * MDD_SDSPI_MediaInitialize();
```

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 byte/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 byte == 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 MDD_MediaInitialize function pointer must be pointing to this function.

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

MEDIA_INFORMATION * MDD_SDSPI_MediaInitialize (void)

6

6.2.1.4 MDD_SDSPI_SectorRead

Reads a sector of data from an SD card.

File

SD-SPI.h

C

```
BYTE MDD_SDSPI_SectorRead(
    DWORD sector_addr,
    BYTE* buffer
);
```

Side Effects

None

Description

The MDD_SDSPI_SectorRead function reads a sector of data bytes (512 bytes) 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 byte address. The sector_addr value is converted to a byte address by shifting it left nine times (multiplying by 512).

This function performs a synchronous read operation. In other words, 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 MDD_SectorRead function pointer must be pointing towards this function.

Parameters

Parameters	Description
sector_addr	The address of the sector on the card.
buffer	The buffer where the retrieved data will be stored. If buffer is NULL, do not store the data anywhere.

Return Values

Return Values	Description
TRUE	The sector was read successfully
FALSE	The sector could not be read

Function

BYTE MDD_SDSPI_SectorRead (DWORD sector_addr, BYTE * buffer)

6.2.1.5 MDD_SDSPI_SectorWrite

Writes a sector of data to an SD card.

File

SD-SPI.h

C

```
BYTE MDD_SDSPI_SectorWrite(
    DWORD sector_addr,
    BYTE* buffer,
    BYTE allowWriteToZero
);
```

Side Effects

None.

Description

The MDD_SDSPI_SectorWrite function writes one sector of data (512 bytes) 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 byte address. The sector_addr value is ocnverted to a byte address by shifting it left nine times (multiplying by 512).

Preconditions

The MDD_SectorWrite function pointer must be pointing to this function.

Parameters

Parameters	Description
sector_addr	The address of the sector on the card.
buffer	The buffer with the data to write.
allowWriteToZero	• TRUE - Writes to the 0 sector (MBR) are allowed • FALSE - Any write to the 0 sector will fail.

Return Values

Return Values	Description
TRUE	The sector was written successfully.
FALSE	The sector could not be written.

Function

BYTE MDD_SDSPI_SectorWrite (DWORD sector_addr, BYTE * buffer, BYTE allowWriteToZero)

6.2.1.6 MDD_SDSPI_ReadSectorSize

Determines the current sector size on the SD card

File

SD-SPI.h

C

```
WORD MDD_SDSPI_ReadSectorSize();
```

Side Effects

None.

Returns

The size of the sectors for the physical media

Description

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

Remarks

None

Preconditions

MDD_MediaInitialize() is complete

Function

WORD MDD_SDSPI_ReadSectorSize (void)

6.2.1.7 MDD_SDSPI_ReadCapacity

Determines the current capacity of the SD card

File

SD-SPI.h

C

```
DWORD MDD_SDSPI_ReadCapacity();
```

Side Effects

None.

Returns

The capacity of the device

Description

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

Remarks

None

Preconditions

MDD_MediaInitialize() is complete

Function

DWORD MDD_SDSPI_ReadCapacity (void)

6.2.1.8 MDD_SDSPI_ShutdownMedia

Disables the SD card

File

SD-SPI.h

C

```
BYTE MDD_SDSPI_ShutdownMedia();
```

Side Effects

None.

Returns

None

Description

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

Remarks

None

Preconditions

The MDD_ShutdownMedia function pointer is pointing towards this function.

Function

BYTE MDD_SDSPI_ShutdownMedia (void)

6.3 CF Physical Layer

The CF physical layer files offer two methods to interface the CF cards.

- The manual interface method bit-bangs the parallel interface protocol to the CF cards.
- The CF-PMP files interface the cards using the parallel master port on 16-bit PIC devices.

6.3.1 Functions

Functions

	Name	Description
	MDD_CFBT_MediaDetect	Determines if a card is inserted
	MDD_CFBT_InitIO	None
	MDD_CFBT_MediaInitialize	Return a MEDIA_INFORMATION structure to FSIO.c
	MDD_CFBT_SectorRead	SectorRead reads 512 bytes of data from the card starting at the sector address specified by sector_addr and stores them in the location pointed to by 'buffer'.
	MDD_CFBT_SectorWrite	SectorWrite sends 512 bytes of data from the location pointed to by 'buffer' to the card starting at the sector address specified by sector_addr.
	MDD_CFBT_WriteProtectState	Added for compatibility- no write protect feature
	MDD_CFBT_CFwait	Wait until the card is ready
	MDD_CFPMP_MediaDetect	Determines if a card is inserted
	MDD_CFPMP_InitIO	None
	MDD_CFPMP_MediaInitialize	Return a MEDIA_INFORMATION structure to FSIO.c
	MDD_CFPMP_SectorRead	SectorRead reads 512 bytes of data from the card starting at the sector address specified by sector_addr and stores them in the location pointed to by 'buffer'.
	MDD_CFPMP_SectorWrite	SectorWrite sends 512 bytes of data from the location pointed to by 'buffer' to the card starting at the sector address specified by sector_addr.
	MDD_CFPMP_WriteProtectState	Added for compatibility- no write protect feature
	MDD_CFPMP_CFwait	Wait until the card and PMP are ready

Description

The following driver functions are API's for FSIO layer.

6.3.1.1 MDD_CFBT_MediaDetect

File

CF- Bit transaction.h

C

```
BYTE MDD_CFBT_MediaDetect();
```

Side Effects

None

Returns

TRUE - Card present FALSE - Card absent

Description

Determines if a card is inserted

Remarks

None

Preconditions

None

Function

BYTE MDD_CFBT_MediaDetect(void)

6.3.1.2 MDD_CFBT_InitIO

File

CF- Bit transaction.h

C

```
void MDD_CFBT_InitIO();
```

Side Effects

None

Returns

void

Description

None

Remarks

None

Preconditions

None

Function

void MDD_CFBT_InitIO(void)

6.3.1.3 MDD_CFBT_MediaInitialize

File

CF- Bit transaction.h

C

```
MEDIA_INFORMATION * MDD_CFBT_MediaInitialize();
```

Side Effects

None

Returns

MEDIA_NO_ERROR - The media initialized successfully

Description

Return a MEDIA_INFORMATION structure to FSIO.c

Remarks

None

Preconditions

None

Function

BYTE MDD_CFBT_MediaInitialize(void)

6.3.1.4 MDD_CFBT_SectorRead

File

CF- Bit transaction.h

C

```
BYTE MDD_CFBT_SectorRead(
    DWORD lda,
    BYTE * buf
);
```

Side Effects

None

Returns

TRUE - Sector read FALSE - Sector could not be read

Description

SectorRead reads 512 bytes of data from the card starting at the sector address specified by sector_addr and stores them in the location pointed to by 'buffer'.

Remarks

None

Preconditions

None

Parameters

Parameters	Description
sector_addr	Sector address, each sector contains 512-byte
buffer	Buffer where data will be stored, see 'ram_acs.h' for 'block' definition. 'Block' is dependent on whether internal or external memory is used

Function

BYTE MDD_CFBT_SectorRead(DWORD sector_addr, BYTE *buffer)

6.3.1.5 MDD_CFBT_SectorWrite

File

CF- Bit transaction.h

C

```
BYTE MDD_CFBT_SectorWrite(
    DWORD lda,
    BYTE * buf,
    BYTE allowWriteToZero
);
```

Side Effects

None

Returns

TRUE - Sector written FALSE - Sector could not be written

Description

SectorWrite sends 512 bytes of data from the location pointed to by 'buffer' to the card starting at the sector address specified by sector_addr.

Remarks

None

Preconditions

None

Parameters

Parameters	Description
sector_addr	Sector address, each sector contains 512 bytes
buffer	Buffer where data will be read from
allowWriteToZero	allows write to the MBR sector

Function

BYTE MDD_CFBT_SectorWrite(DWORD sector_addr, BYTE *buffer, BYTE allowWriteToZero)

6.3.1.6 MDD_CFBT_WriteProtectState

File

CF- Bit transaction.h

C

```
BYTE MDD_CFBT_WriteProtectState();
```

Side Effects

None

Returns

0

Description

Added for compatibility- no write protect feature

Remarks

None

Preconditions

None

Function

BYTE MDD_CFBT_WriteProtectState(void)

6.3.1.7 MDD_CFBT_CFwait

File

CF- Bit transaction.h

C

```
void MDD_CFBT_CFwait();
```

Side Effects

None

Returns

None

Description

Wait until the card is ready

Remarks

None

Preconditions

None

Function

BYTE MDD_CFBT_CFwait(void)

6.3.1.8 MDD_CFPMP_MediaDetect

File

CF-PMP.h

C

```
BYTE MDD_CFPMP_MediaDetect();
```

Side Effects

None

Returns

TRUE - Card present FALSE - Card absent

Description

Determines if a card is inserted

Remarks

None

Preconditions

None

Function

BYTE MDD_CFPMP_MediaDetect(void)

6.3.1.9 MDD_CFPMP_InitIO

File

CF-PMP.h

C

```
void MDD_CFPMP_InitIO();
```

Side Effects

None

Returns

TRUE - Card initialized FALSE - Card not initialized

Description

None

Remarks

None

Preconditions

None

Function

BYTE MDD_CFPMP_InitIO(void)

6.3.1.10 MDD_CFPMP_MediaInitialize

File

CF-PMP.h

C

```
MEDIA_INFORMATION * MDD_CFPMP_MediaInitialize();
```

Side Effects

None

Returns

MEDIA_NO_ERROR - The media initialized successfully

Description

Return a MEDIA_INFORMATION structure to FSIO.c

Remarks

None

Preconditions

None

Function

BYTE MDD_CFPMP_MediaInitialize(void)

6.3.1.11 MDD_CFPMP_SectorRead

File

CF-PMP.h

C

```
BYTE MDD_CFPMP_SectorRead(
    DWORD lda,
    BYTE * buf
);
```

Side Effects

None

Returns

TRUE - Sector read FALSE - Sector could not be read

Description

SectorRead reads 512 bytes of data from the card starting at the sector address specified by sector_addr and stores them in the location pointed to by 'buffer'.

Remarks

None

Preconditions

None

Parameters

Parameters	Description
sector_addr	Sector address, each sector contains 512-byte
buffer	Buffer where data will be stored

Function

BYTE MDD_CFPMP_SectorRead(DWORD sector_addr, BYTE *buffer)

6.3.1.12 MDD_CFPMP_SectorWrite

File

CF-PMP.h

C

```
BYTE MDD_CFPMP_SectorWrite(
    DWORD lda,
    BYTE * buf,
    BYTE allowWriteToZero
);
```

Side Effects

None

Returns

TRUE - Sector written FALSE - Sector could not be written

Description

SectorWrite sends 512 bytes of data from the location pointed to by 'buffer' to the card starting at the sector address specified by sector_addr.

Remarks

None

Preconditions

None

Parameters

Parameters	Description
sector_addr	Sector address, each sector contains 512 bytes
buffer	Buffer where data will be read from
allowWriteToZero	allows write to the MBR sector

Function

BYTE MDD_CFPMP_SectorWrite(DWORD sector_addr, BYTE *buffer, BYTE allowWriteToZero)

6.3.1.13 MDD_CFPMP_WriteProtectState

File

CF-PMP.h

C

```
BYTE MDD_CFPMP_WriteProtectState();
```

Side Effects

None

Returns

0

Description

Added for compatibility- no write protect feature

Remarks

None

Preconditions

None

Function

BYTE MDD_CFPMP_WriteProtectState(void)

6.3.1.14 MDD_CFPMP_CFwait

File

CF-PMP.h

C

```
void MDD_CFPMP_CFwait();
```

Side Effects

None

Returns

None

Description

Wait until the card and PMP are ready

Remarks

None

Preconditions

None

Function

void MDD_CFPMP_CFwait(void)

Index

G

I

L

M

R

S

T

U

W