# Microchip MDD File System Interface Library

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

The MDDFS Interface Library will provide an easy way to create and manipulate files on removable flash-based media devices.

This help file served two purposes. The first is to be a guide for first-time users of the MDDFS Interface Library. The Getting Started section begins a series of pages to help you become familiar with the library and configure it for use on a Microchip development board.

The second purpose is to serve as a programmer's reference guide. Many significant changes have been made to the stack since AN1045 was first published, and this document will serve as a more up-to-date guide to the features and APIs available in the MDDFS Interface Library.

## 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 http://www.microchip.com. New features are constantly being added, so check there periodically for updates and bug fixes.

## Thank You!

We appreciate your interest in the Microchip MDD File System

Interface Library, and thank you for choosing Microchip products!

## Topics

| Name | Description |
| --- | --- |
| Getting Help | Where to go for more help |

Microchip MDD File System Interface Library

Microchip MDD File System Interface 1.2.0 - [Aug 18, 2008]

# 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.

Microchip MDD File System Interface Library > Getting Help

# Getting Started

This section will walk through the initial configuration of the stack and compatible Microchip development hardware.

To begin, start by familiarizing yourself with the Directory Structure.

## Topics

| Name | Description |
|---|---|
| Terminology | A list of terms that may appear in this help file. |
| Directory Structure | Describes where to find files in the MDDFS Interface Library. |
| Configuring Hardware | Walks through hardware configuration for supported development platforms. |
| Software Configuration | Information about software configuration options for the library. |
| The SD Card Demo | Information about the default SD Card demo. |
| The SD Data Logger Demo | Information about the SD card file explorer demo. |

# Terminology

A list of terms that may appear in this help file.

## Topics

| Name | Description |
|---|---|
| Boot sector | The boot sector is the first sector of a partition. It contains information about how the partition is organized. |
| 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. |
| 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. |
| Directory | A directory is a type of file that contains pointers to other files or directories. |
| 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. |
| 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. |
| 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. |
| Sector | A sector is a group of bytes in the FAT file system. Sectors are most commonly 512 bytes. |

# Boot sector

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

Getting Started > Terminology > Boot sector

# 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.

Getting Started > Terminology > Cluster

# 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.

Getting Started > Terminology > Current Working Directory

# Directory

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

Getting Started > Terminology > Directory

# 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.

Getting Started > Terminology > FAT

# 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.

Getting Started > Terminology > Master Boot Record

# 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.

Getting Started > Terminology > Root directory

# Sector

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

Getting Started > Terminology > Sector

# Directory Structure

The MDDFS Library comes with many files, documents, and project examples. Before getting started, take a moment to familiarize yourself with the 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. Insize that folder, several subfolders are created, as documented in the table below.

| | |
|---|---|
| C:\Microchip\Solutions | Root folder for all library files |
| - Microchip | Internal stack files. These files rarely need modification. |
| -- Include | |
| --- MDD File System | Header (*.h) files for the MDDFS Library |
| --- PIC18 salloc | Header (*.h) files for dynamic memory allocation for PIC18 |
| -- MDD File System | Source (*.c) files for the MDDFS Library |
| --- Documentation | Readme files, schematics, and AN1045. |
| -- PIC18 salloc | Source (*.c) files for dynamic memory allocation for PIC18 |
| -- Help | The location of this help file. |
| - MDD File System-SD Card | Main file I/O demo application. |
| | Configuration files for the PIC18F project |

| | |
|---|---|
| -- PIC18F | in the demo. |
| -- PIC24F | Configuration files for the PIC24F project in the demo. |
| -- PIC32 | Configuration files for the PIC32 project in the demo. |
| - MDD File System-SD Data Logger | Demo application that functions as a shell program using the UART module and an SD card. |

# Configuring Hardware

The first step to use the stack is to make sure an appropriate development board is configured. To get started, select a platform from the topics presented below.

## Topics

| Name | Description |
|---|---|
| Explorer 16 with PICtail for SD and MMC | Information about how to configure the Explorer 16 board to use the demo projects. |
| HPC Explorer with PICtail for SD and MMC | Information about how to configure the HPC Explorer board to use the demo projects. |

Getting Started > Configuring Hardware

# Explorer 16 with PICtail for SD and MMC

Visit the Microchip web site to view the Explorer 16 Product Page and User's Guide, and the PICtail Board for SD and MMC,

The Explorer 16 board can be expanded for SD card support using the PICtail™ Board for SD and MMC. The daughterboard should be inserted into the top-most socket of the J5 header. The orientation of the daughterboard should be such that the card socket faces towards the Processor Interface Module.

Check that:

1. Switch S2 selects PIM.
2. Jumper J7 selects PIC24.

The development board is now ready to use.

## Project Setup

A few configuration settings are required to ensure that the MDDFS Interface Library will run on your hardware:

1. Start with an appropriate MPLAB IDE project:
   - 16-bit parts: MDD File System-SD Card\MDDFS-SD-PIC24.mcp
   - 32-bit parts: MDD File System-SD Card\MDDFS-SD-PIC32.mcp
2. Change the MPLAB IDE processor target selection to match the part installed on the Explorer 16 (e.x. PIC24FJ128GA010, PIC32MX360F256L).

Getting Started > Configuring Hardware > Explorer 16 with PICtail for SD and MMC

# HPC Explorer with PICtail for SD and MMC

Visit the Microchip web site to view the HPC Explorer Product Page and User's Guide, and the PICtail Board for SD and MMC,

The HPC Explorer board can be expanded for SD card support using the PICtail™ Board for SD and MMC. The daughterboard should be inserted into the PICtail connector.

The development board is now ready to use.

## Project Setup

A few configuration settings are required to ensure that the MDDFS Interface Library will run on your hardware:

1. Start with an appropriate MPLAB IDE project: MDD File System-SD Card\MDDFS-SD-PIC18.mcp
2. Change the MPLAB IDE processor target selection to match the part installed on the HPC Explorer Board (e.x. PIC18F8722).

## Troubleshooting

Because of the level translator on the PICtail™ board for SD and MMC, some versions of the HPC Explorer may experience communication errors between the microcontroller and SD card. This disruption is not limited to a specific range of clock frequencies. To improve the chances of a successful communication, some options are:

1. Change the SPI module speed so the communication will proceed more slowly.
2. Cut the traces that correspond to the SPI pins near the PICtail connector. Bridge the trace cuts with 100-400 Ohm terminating resistors. Note that this solution will cause damage to the HPC Explorer board.

# Software Configuration

Once your hardware is configured, the next step is to configure the MDDFS Library firmware. Library configuration is stored as a set of configuration macros in FSconfig.h and HardwareProfiles.h in the demo application folders. To begin, open FSconfig.h in a text editor program like MPLAB.

## 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 maximu number of files that can/will be opened at once. |
| 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 definitio out to disable the file search functions (FindFirst and FindNext). This will reduce code size. |
| ALLOW_WRITES | Feature toggle | Comment this definitio out to disable all write functionality. This will reduce code size. |
| | | Comment this definitio |

| | | |
|---|---|---|
| ALLOW_FORMATS | Feature toggle | out to disable the form function. This will reduce code size. |
| ALLOW_DIRS | Feature toggle | Comment this definitio out to disable all directory functionality. This will reduce code size. |
| ALLOW_PGMFUNCTIONS | Feature toggle | Comment this definitio out to disable -pgm functions. The library requires -pgm function to be disabled when no using PIC18. This will reduce code size. |
| ALLOW_FSFPRINTF | Feature toggle | Comment this definitio out to disable the FSfprintf function. This will reduce code size. |
| SUPPORT_FAT32 | Feature toggle | Comment this definitio out to disable FAT32 support. FAT12 and FAT16 will still be supported. |
| USEREALTIMECLOCK | Create/last modified timestamp generator | Uncomment this macro to generate timestamp automatically with the RTCC module. You must configure the RTCC for this method to work correctly. Only one timestamp generation method ma be enabled at one time |

| | | |
|---|---|---|
| USERDEFINEDCLOCK | Create/last modified timestamp generator | Uncomment this macro to generate timestamp based on global variabled that are set manually by the user using the SetClockVars() functio Only one timestamp generation method ma 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 ma be enabled at one time |
| FS_DYNAMIC_MEM | Static/dynamic FSFILE object allocation. | Set the #if preprocesso definition to 1 to allocate FSFILE object dynamically. You will b 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 w |

| | | |
|---|---|---|
| | | be allocated in a static array, with the maximum number of FSFILE objects determined by the FS_MAX_FILES_OPE 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. |

# The SD Card Demo

The basic SD Card demo application is located in the MDD File System-SD Card folder. It contains projects for PIC18, PIC24, and PIC32 architectures. This project will give a basic demonstration of how most of the file I/O functions can be used.

Getting Started > The SD Card Demo

# The SD Data Logger Demo

The SD Data Logger project is based on the USB Data logger project found in AN1145.

## Additional Hardware Setup

This demonstration is set up to use the Explorer 16 board with a PIC24 or PIC32 part. It will require a serial connection from the D-sub (DB-9) connector on the Explorer 16 to a PC running a terminal program (e.g. HyperTerminal) with the following settings: 57600 BPS, 8 data bits, no parity, 1 stop bit, no flow control.

## Using the Demo

Upon programming and running the demo code, a command prompt will appear on the PC's terminal screen. By typing "HELP" or "?" and pressing enter, the user can display a list of commands that can be used to access and modify files and directories on the card. Note that the terminal program is only transmitting and receiving information from the microcontroller; all functionality (including echoing characters) is handled by the microcontroller. Available shell commands in this demo include:

| COMMAND | SYNTAX | EXAMPLE | FUNCTION |
|---|---|---|---|
| ATTRIB | ATTRIB <+\|->R <+\|->S <+\|->H <+\|->A <name> | ATTRIB +S +A -H FILE.TXT | Clears or sets read-only, system, hidden, or archive attributes from a file or directory. |
| | CD | | Changes the directory to the |

| | | | |
|---|---|---|---|
| CD | <name> | CD DIR1\DIR2 | path specified by <name>. |
| COPY | COPY <file1> <file2> | COPY ONE.TXT TWO.TXT | Copies a file. |
| COPY | COPY CON <file1> | COPY CON EXAMPLE.TXT | Copies data from the console into a file as the user types it. |
| DATE | DATE [yyyy-mm-dd] | DATE 2008-08-19 | Sets the date. If no date is specified, this command will display the currently set date. |
| DEL | DEL <file> | DEL FILE.TXT | Deletes a file. |
| DIR | DIR [file] | DIR EXAMPLE.* | Displays files or directories in the current working directory that match the specified naming criteria. If no argument is specified, all files in the current working directory will be displayed. |
| HELP/? | HELP | HELP | Displays a list of available commands. |

| | | | |
|---|---|---|---|
| LOG | LOG <POT\|TMP> <file> | LOG TMP DATA.CSV | Logs data from the temperature sensor to potentiometer on the Explorer 16 to the specified file. |
| MD | MD <name> | MD ONE\TWO\THREE | Create one or more directories. |
| RD | RD <name> | RD ONE\TWO\THREE | Remove a directory. |
| REN | REN <file1> <file2> | REN ONE.TXT TWO.TXT | Rename <file1> to <file2> |
| TIME | TIME [hh:mm:ss] | TIME [10:52:03] | Set the time to the specified value. If no value is specified, the current time will be output. |
| TYPE | TYPE <file> | TYPE EXAMPLE.TXT | Display the contents of a file in ASCII text |

# APIs

The Microchip MDDFS Interface Library is implemented as an application layer (written by the user), a file manipulation layer, which actually performs operations on files, and one of several physical interface layers, including an SD card interface and two methods for interfacing with CF cards. The APIs for the file manipulation layer and each physical layer are described in this section.

## Topics

| Name | Description |
|---|---|
| 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. |
| SD-SPI Physical Layer | The SD-SPI physical layer offers the ability to interface to SD cards using the SPI protocol. SPI modules can be found on many Microchip microcontrollers. |
| CF Physical Layer | The CF physical layers offer two methods for interfacing with CF cards. The manual interface method will bit-bang the parallel interface protocol used by CF cards. The CF-PMP files will interface to the cards using the parallel master port on 16-bit PIC devices. At this time, 8-bit architecture PMP interface is not supported. |

# 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.

## Topics

| Name | Description |
|---|---|
| Public Members | The following functions, variables, structures, and macros are available for use by the user application. |
| Library Members | The following functions, variables, structures, and macros are public, but are intended only to be accessed by the library itself. Applications should generally not call these functions or modify these variables. |
| Internal Members | The following functions, variables, structures, and macros are designated as internal to the library. |

APIs > File Manipulation Layer (FSIO)

# Public Members

The following functions, variables, structures, and macros are available for use by the user application.

## Functions

| | Name | Description |
|---|---|---|
| | FindFirst | Initial search function |
| | FindFirstpgm | Find a file named with a ROM string on PIC18 |
| | FindNext | Sequential search function |
| | FSattrib | Change the attributes of a file |
| | FSchdir | Change the current working directory |
| | FSchdirpgm | Changed the CWD with a path in ROM on PIC18 |
| | FSCreateMBR | Creates a master boot record |
| | FSerror | Return an error code for the last function call |
| | FSfclose | Update file information and free FSFILE objects |
| | FSfeof | Indicate whether the current file position is at the end |
| | FSfopen | Open a file |
| | | Open a file named with a ROM string on |

| | | |
|---|---|---|
| | FSfopenpgm | PIC18 |
| | FSformat | Formats a device |
| | FSfprintf | Function to write formatted strings 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 |
| | FSfwrite | Write data to a file |
| | FSgetcwd | Get the current working directory name |
| | FSInit | Function to initialize the device. |
| | FSmkdir | Create a directory |
| | FSmkdirpgm | Create a directory with a path in ROM on PIC18 |
| | FSremove | Delete a file |
| | FSremovepgm | Delete a file named with a ROM string on PIC18 |
| | FSrename | Change the name of a file or directory |
| | FSrenamepgm | Rename a file named with a ROM string on PIC18 |
| | FSrewind | Set the current position in a file to the beginning |
| | FSrmdir | Delete a directory |
| | | Delete a directory with a path in ROM on |

| | | |
|---|---|---|
| | FSrmdirpgm | PIC18 |
| | SetClockVars | Manually set timestamp variables |

## Macros

| | Name | Description |
|---|---|---|
| | ALLOW_DIRS | A macro to enable/disable directory operations. |
| | ALLOW_FILESEARCH | A macro to enable/disable file search functions. |
| | ALLOW_FSFPRINTF | A macro to enable/disable the FSfprintf function. |
| | ALLOW_FORMATS | A macro to enable/disable format functionality |
| | ALLOW_PGMFUNCTIONS | A macro to enable/disable PIC18 ROM functions. |
| | ALLOW_WRITES | A macro to enable/disable write functionality |
| | APPEND | Macro for the FSfopen APPEND mode |
| | APPENDPLUS | Macro for the FSfopen APPEND+ mode |

| | | |
|---|---|---|
| | ATTR_ARCHIVE | An archive attribute macro |
| | ATTR_DIRECTORY | A directory attribute macro |
| | ATTR_HIDDEN | A hidden attribute macro |
| | ATTR_MASK | A macro for all attributes |
| | ATTR_READ_ONLY | A read-only attribute macro |
| | ATTR_SYSTEM | A system attribute macro |
| | ATTR_VOLUME | A volume attribute macro |
| | EOF | Indicates error conditions or end-of-file conditions |
| | FALSE | False value |
| | 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 |
| | INCREMENTTIMESTAMP | A macro to enable don't-care timestamp generation |
| | intmax_t | A data type indicating the maximum integer size in an architecture |
| | | |

| | | |
|---|---|---|
| | MAX_HEAP_SIZE | A macro used to define the heap size for PIC18 |
| | MDD_MediaDetect | Function pointer to the Media Detect Physical Layer function |
| | MEDIA_SECTOR_SIZE | A macro defining the size of a sector |
| | NEAR_MODEL | A macro used to enable nead-model RAM addressing |
| | READ | Macro for the FSfopen READ mode |
| | READPLUS | Macro for the FSfopen READ+ mode |
| | SEEK_CUR | Macro for the FSfseek SEEK_CUR base location. |
| | SEEK_END | Macro for the FSfseek SEEK_END base location |
| | SEEK_SET | Macro for the FSfseek SEEK_SET base location. |
| | SUPPORT_FAT32 | A macro to enable/disable FAT32 support. |
| | TRUE | True value |
| | | |

| | | |
|---|---|---|
| | USE_CF_INTERFACE_WITH_PMP | Macro used to enable the CF-PMP physical layer (CF-PMP.c and .h) |
| | USE_MANUAL_CF_INTERFACE | Macro used to enable the CF-Manual physical layer (CF-Bit transaction.c and .h) |
| | USE_SD_INTERFACE_WITH_SPI | Macro used to enable the SD-SPI physical layer (SD-SPI.c and .h) |
| | USE_USB_INTERFACE | Macro used to enable the USB Host physical layer (USB host MSD library) |
| | USERDEFINEDCLOCK | A macro to enable manual timestamp generation |
| | USEREALTIMECLOCK | A macro to enable RTCC based timestamp generation |
| | WRITE | Macro for the FSfopen WRITE mode |
| | WRITEPLUS | Macro for the FSfopen WRITE+ mode |

## 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. |

# FindFirst Function

C

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

## Description

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.

## Preconditions

None

## Parameters

| Parameters | Description |
|---|---|
| fileName | The name to search for<br>• Parital string search characters<br>• * - Indicates the rest of the filename or extension can vary (e.g. FILE.*)<br>• ? - Indicates that one character in a filename can vary (e.g. F?LE.T?T) |

| | |
|---|---|
| | |
| attr | The attributes that a found file may have<br>• ATTR_READ_ONLY - File may be read only<br>• ATTR_HIDDEN - File may be a hidden file<br>• ATTR_SYSTEM - File may be a system file<br>• ATTR_VOLUME - Entry may be a volume label<br>• ATTR_DIRECTORY - File may be a directory<br>• ATTR_ARCHIVE - File may have archive attribute<br>• 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 |

## Side Effects

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

## Remarks

Call FindFirst or FindFirstpgm before calling FindNext

# FindFirstpgm Function

C

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

## Description

The FindFirstpgm function will copy a PIC18 ROM fileName argument into a RAM array, and then pass that array to the FindFirst function.

## 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 |

## Side Effects

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

## Remarks

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

# FindNext Function

C

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

## 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.

## 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 |

## Side Effects

The FSerrno variable will be changed.

## Remarks

Call FindFirst or FindFirstpgm before calling this function

# FSattrib Function

C

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

## 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.

## Preconditions

File opened

## Parameters

| Parameters | Description |
|---|---|
| file | Pointer to file structure |
| attributes | The attributes to set for the file<br>• Attribute - Value - Indications<br>• ATTR_READ_ONLY - 0x01 - The read-only attribute<br>• ATTR_HIDDEN - 0x02 - The hidden attribute<br>• 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 |

## Side Effects

The FSerrno variable will be changed.

## Remarks

None

# FSchdir Function

C

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

## Description

The FSchdir function passes a RAM pointer to the path to the chdirhelper function.

## 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 |

## Side Effects

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

## Remarks

None

# FSchdirpgm Function

C

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

## Description

The FSchdirpgm function passes a PIC18 ROM path pointer to the chdirhelper function.

## 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 |

## Side Effects

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

## Remarks

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

# FSCreateMBR Function

C

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

## 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.

## Preconditions

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

## Parameters

| Parameters | Description |
|---|---|
| | The first sector of the partition on the device |

| | |
|---|---|
| firstSector | (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 |

## Side Effects

None

## 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.

# FSerror Function

C

```c
int FSerror();
```

## 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.

## Preconditions

The return value depends on the last function called.

## Return Values

| Return Values | Description |
|---|---|
| FSInit | • CE_GOOD – No Error<br>• CE_INIT_ERROR – The physical media could not be initialized<br>• CE_BAD_SECTOR_READ – The MBR or the boot sector could not be read correctly<br>• CE_BAD_PARITION – The MBR signature code was incorrect.<br>• CE_NOT_FORMATTED – The boot sector signature code was incorrect or indicates an invalid number of bytes per sector.<br>• 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<br>• CE_NOT_INIT – The device has not been initialized.<br>• CE_TOO_MANY_FILES_OPEN – The function could not allocate any additional file information to the array of FSFILE structures or the heap.<br>• CE_INVALID_FILENAME – The file name argument was invalid.<br>• CE_INVALID_ARGUMENT – The user attempted to open a directory in a write mode or specified an invalid mode argument.<br>• CE_FILE_NOT_FOUND – The specified file (which was to be opened in read mode) does not exist on the device.<br>• CE_BADCACHEREAD – A read from the device failed.<br>• CE_ERASE_FAIL – The existing file could not be erased (when opening a file in WRITE mode).<br>• CE_DIR_FULL – The directory is full.<br>• CE_DISK_FULL– The data memory section is full.<br>• CE_WRITE_ERROR – A write to the device failed.<br>• CE_SEEK_ERROR – The current position in the file could not be set to the end (when the file was opened in APPEND mode). |

| | |
|---|---|
| | |
| FSfclose | • CE_GOOD – No Error<br>• CE_WRITE_ERROR – The existing data in the data buffer or the new file entry information could not be written to the device.<br>• CE_BADCACHEREAD – The file entry information could not be cached |
| FSfread | • CE_GOOD – No Error<br>• CE_WRITEONLY – The file was opened in a write-only mode.<br>• CE_WRITE_ERROR – The existing data in the data buffer could not be written to the device.<br>• CE_BAD_SECTOR_READ – The data sector could not be read.<br>• CE_EOF – The end of the file was reached.<br>• CE_COULD_NOT_GET_CLUSTER – Additional clusters in the file could not be loaded. |
| FSfwrite | • CE_GOOD – No Error<br>• CE_READONLY – The file was opened in a read-only mode.<br>• CE_WRITE_PROTECTED – The device write-protect check function indicated that the device has been write-protected.<br>• CE_WRITE_ERROR – There was an error writing data to the device.<br>• CE_BADCACHEREAD – The data |

| | |
|---|---|
| | sector to be modified could not be read from the device.<br>• CE_DISK_FULL – All data clusters on the device are in use. |
| FSfseek | • CE_GOOD – No Error<br>• CE_WRITE_ERROR – The existing data in the data buffer could not be written to the device.<br>• CE_INVALID_ARGUMENT – The specified offset exceeds the size of the file.<br>• CE_BADCACHEREAD – The sector that contains the new current position could not be loaded.<br>• 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<br>• CE_INVALID_ARGUMENT – The attribute argument was invalid.<br>• CE_BADCACHEREAD – The existing file entry information could not be loaded.<br>• CE_WRITE_ERROR – The file entry information could not be written to the device. |
| | |

| | |
|---|---|
| FSrename | • CE_GOOD – No Error<br>• CE_FILENOTOPENED – A null file pointer was passed into the function.<br>• CE_INVALID_FILENAME – The file name passed into the function was invalid.<br>• CE_BADCACHEREAD – A read from the device failed.<br>• CE_FILENAME_EXISTS – A file with the specified name already exists.<br>• 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<br>• CE_INIT_ERROR – The device could not be initialized.<br>• CE_BADCACHEREAD – The master boot record or boot sector could not be loaded successfully.<br>• 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.<br>• CE_WRITE_ERROR – The updated MBR/Boot sector could not be written to the device.<br>• 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<br>• CE_WRITE_PROTECTED – The device write-protect check function indicated that the device has been write-protected.<br>• CE_INVALID_FILENAME – The specified filename was invalid.<br>• CE_FILE_NOT_FOUND – The specified file could not be found.<br>• CE_ERASE_FAIL – The file could not be erased. |
| FSchdir | • CE_GOOD – No Error<br>• CE_INVALID_ARGUMENT – The path string was mis-formed or the user tried to change to a non-directory file.<br>• CE_BADCACHEREAD – A directory entry could not be cached.<br>• CE_DIR_NOT_FOUND – Could not find a directory in the path. |
| FSgetcwd | • CE_GOOD – No Error<br>• CE_INVALID_ARGUMENT – The user passed a 0-length buffer into the function.<br>• CE_BADCACHEREAD – A directory entry could not be cached.<br>• CE_BAD_SECTOR_READ – The |

| | |
|---|---|
| | function could not determine a previous directory of the current working directory. |
| FSmkdir | • CE_GOOD – No Error<br>• CE_WRITE_PROTECTED – The device write-protect check function indicated that the device has been write-protected.<br>• CE_INVALID_ARGUMENT – The path string was mis-formed.<br>• CE_BADCACHEREAD – Could not successfully change to a recently created directory to store its dir entry information, or could not cache directory entry information.<br>• CE_INVALID_FILENAME – One or more of the directory names has an invalid format.<br>• 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.<br>• CE_DIR_FULL – There are no available dir entries in the CWD.<br>• CE_DISK_FULL – There are no available clusters in the data region of the device. |
| | • CE_GOOD – No Error<br>• CE_DIR_NOT_FOUND – The directory specified could not be found or the function could not change to a |

| | |
|---|---|
| FSrmdir | subdirectory within the directory to be deleted (when recursive delete is enabled).<br>• CE_INVALID_ARGUMENT – The user tried to remove the CWD or root directory.<br>• CE_BADCACHEREAD – A directory entry could not be cached.<br>• CE_DIR_NOT_EMPTY – The directory to be deleted was not empty and recursive subdirectory removal was disabled.<br>• CE_ERASE_FAIL – The directory or one of the directories or files within it could not be deleted.<br>• CE_BAD_SECTOR_READ – The function could not determine a previous directory of the CWD. |
| SetClockVars | • CE_GOOD – No Error<br>• CE_INVALID_ARGUMENT – The time values passed into the function were invalid. |
| FindFirst | • CE_GOOD – No Error<br>• CE_INVALID_FILENAME – The specified filename was invalid.<br>• CE_FILE_NOT_FOUND – No file matching the specified criteria was found.<br>• CE_BADCACHEREAD – The file information for the file that was found could not be cached. |

| | |
|---|---|
| | |
| FindNext | • CE_GOOD – No Error<br>• CE_NOT_INIT – The SearchRec object was not initialized by a call to FindFirst.<br>• CE_INVALID_ARGUMENT – The SearchRec object was initialized in a different directory from the CWD.<br>• CE_INVALID_FILENAME – The filename is invalid.<br>• CE_FILE_NOT_FOUND – No file matching the specified criteria was found. |
| FSfprintf | • CE_GOOD – No Error<br>• CE_WRITE_ERROR – Characters could not be written to the file. |

## Side Effects

None.

## Remarks

None

APIs > File Manipulation Layer (FSIO) > Public Members > FSerror Function

# FSfclose Function

C

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

## 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.

## 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 |

## Side Effects

The FSerrno variable will be changed.

## 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.

# FSfeof Function

C

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

## 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.

## 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 |

## Side Effects

The FSerrno variable will be changed.

## Remarks

None.

# FSfopen Function

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

## Description

This function will open a file or directory. 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. 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, it 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.

## Preconditions

For read modes, file exists; FSInit performed

## Parameters

| Parameters | Description |
| --- | --- |
| fileName | The name of the file to open |
| mode | • WRITE - Create a new file or replace an existing file<br>• READ - Read data from an existing file<br>• APPEND - Append data to an existing file<br>• WRITEPLUS - Create a new file or replace an existing file (reads also enabled)<br>• READPLUS - Read data from an existing file (writes also enabled)<br>• 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 |

## Side Effects

The FSerrno variable will be changed.

## Remarks

None.

# FSfopenpgm Function

C

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

## Description

The FSfopenpgm function will copy a PIC18 ROM fileName and mode argument into RAM arrays, and then pass those arrays to the FSfopen function.

## 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 |

## Side Effects

The FSerrno variable will be changed.

## Remarks

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

# FSformat Function

C

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

## 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.

## Preconditions

The device must possess a valid master boot record.

## Parameters

| Parameters | Description |
|---|---|
| | • 0 - Just erase the FAT and root |

| | |
|---|---|
| mode | • 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 |

## Side Effects

The FSerrno variable will be changed.

## Remarks

FAT12 and FAT16 formatting is supported.

# FSfprintf Function

C

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

## Description

Writes a specially formatted string to a file.

## 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 |

## Returns

The number of characters written to the file

## Side Effects

The FSerrno variable will be changed.

## Remarks

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

# FSfread Function

C

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

## 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.

## 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 |

## Returns

size_t - number of units read

## Side Effects

The FSerrno variable will be changed.

## Remarks

None.

# FSfseek Function

C

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

## 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.

## Preconditions

File opened

## Parameters

| Parameters | Description |
|---|---|
| stream | Pointer to file structure |
| offset | Offset from base location |
| | • SEEK_SET - Seek from start of file<br>• SEEK_CUR - Seek from current |

| | |
|---|---|
| whence | location<br>• SEEK_END - Seek from end of file (subtract offset) |

## Return Values

| Return Values | Description |
|---|---|
| 0 | Operation successful |
| -1 | Operation unsuccesful |

## Side Effects

The FSerrno variable will be changed.

## Remarks

None

# FSftell Function

C

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

## 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.

## Preconditions

File opened

## Parameters

| Parameters | Description |
|---|---|
| fo | Pointer to file structure |

## Returns

Current location in the file

## Side Effects

The FSerrno variable will be changed

## Remarks

None

# FSfwrite Function

C

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

## 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 filezize 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.

## Preconditions

File opened in WRITE, APPEND, WRITE+, APPEND+, READ+ mode

## Parameters

| | |
|---|---|

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

## Returns

size_t - number of units written

## Side Effects

The FSerrno variable will be changed.

## Remarks

None.

# FSgetcwd Function

C

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

## Description

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.

## 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. |

## Side Effects

The FSerrno variable will be changed

## Remarks

None

# FSInit Function

C

```c
int FSInit();
```

## Description

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.

## Preconditions

The physical device should be connected to the microcontroller.

## Return Values

| Return Values | Description |
|---|---|
| TRUE | Initialization successful |
| FALSE | Initialization unsuccessful |

## Side Effects

The FSerrno variable will be changed.

## Remarks

None

# Function

# FSmkdir Function

C

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

## Description

The FSmkdir function passes a RAM pointer to the path to the mkdirhelper function.

## 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 |

## Side Effects

Will create all non-existent directories in the path. The FSerrno

variable will be changed.

## Remarks

None

# FSmkdirpgm Function

C

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

## Description

The FSmkdirpgm function passes a PIC18 ROM path pointer to the mkdirhelper function.

## 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 |

## Side Effects

Will create all non-existent directories in the path. The FSerrno

variable will be changed.

## Remarks

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

# FSremove Function

C

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

## Description

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.

## 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 |

## Side Effects

The FSerrno variable will be changed.

## Remarks

None

# FSremovepgm Function

C

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

## Description

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

## 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 |

## Side Effects

The FSerrno variable will be changed.

## Remarks

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

# FSrename Function

C

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

## Description

The FSrename function will rename a file. 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.'

## 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 |

## Side Effects

The FSerrno variable will be changed.

## Remarks

None

# FSrenamepgm Function

C

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

## Description

The Fsrenamepgm function will copy the rom fileName specified by the user into a RAM array and pass that array into the FSrename function.

## 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 |

## Side Effects

The FSerrno variable will be changed.

## Remarks

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

# FSrewind Function

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

## 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.

## Preconditions

File opened.

## Parameters

| Parameters | Description |
|---|---|
| fo | Pointer to file structure |

## Side Effects

None.

## Remarks

None.

# FSrmdir Function

C

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

## Description

The FSrmdir function passes a RAM pointer to the path to the rmdirhelper function.

## 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<br>• 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 |

## Side Effects

The FSerrno variable will be changed.

## Remarks

None.

# FSrmdirpgm Function

C

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

## Description

The FSrmdirpgm function passes a PIC18 ROM path pointer to the rmdirhelper function.

## 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<br>• 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 |

## Side Effects

The FSerrno variable will be changed.

## Remarks

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

# SetClockVars Function

C

```c
int SetClockVars(
    unsigned int year,
    unsigned char month,
    unsigned char day,
    unsigned char hour,
    unsigned char minute,
    unsigned char second
);
```

## Description

Lets the user manually set the timing variables. The values passed in will be converted to the format used by the FAT timestamps.

## Preconditions

USERDEFINEDCLOCK macro defined in FSconfig.h.

## Parameters

| Parameters | Description |
|---|---|
| year | The year (1980-2107) |
| month | The month (1-12) |
| day | The day of the month (1-31) |
| hour | The hour (0-23) |
| minute | The minute (0-59) |

| | |
|---|---|
| second | The second (0-59) |

## Side Effects

Modifies global timing variables

## Remarks

Call this before creating a file or directory (set create time) and before closing a file (set last access time, last modified time)

# ALLOW_DIRS Macro

C

```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.

APIs > File Manipulation Layer (FSIO) > Public Members > ALLOW_DIRS Macro

# ALLOW_FILESEARCH Macro

C

```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.

APIs > File Manipulation Layer (FSIO) > Public Members > ALLOW_FILESEARCH Macro

# ALLOW_FSFPRINTF Macro

C

```c
#define ALLOW_FSFPRINTF
```

## Description

The ALLOW_FSFPRINTF definition can be commented out to disable the FSfprintf function. This will save code space. Note that if FSfprintf is enabled and the PIC18 architecture is used, integer promotions must be enabled in the Project->Build Options menu. Write operations must be enabled to use FSfprintf.

APIs > File Manipulation Layer (FSIO) > Public Members > ALLOW_FSFPRINTF Macro

# ALLOW_FORMATS Macro

C

```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.

APIs > File Manipulation Layer (FSIO) > Public Members > ALLOW_FORMATS Macro

# ALLOW_PGMFUNCTIONS Macro

C

```
#define ALLOW_PGMFUNCTIONS
```

## Description

The ALLOW_PGMFUNCTIONS definition can be commented out to disable all PIC18 functions that allow the user to pass string arguments in ROM (denoted by the suffix -pgm). Note that this functionality must be disabled when not using PIC18.

APIs > File Manipulation Layer (FSIO) > Public Members > ALLOW_PGMFUNCTIONS Macro

# ALLOW_WRITES Macro

C

```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.

APIs > File Manipulation Layer (FSIO) > Public Members > ALLOW_WRITES Macro

# APPEND Macro

C

```c
#define APPEND "a"
```

## Description

If this macro is specified as the mode argument in a call of FSfopen, the file being opened will be created if it doesn't exist. If it does exist, it's file information will be loaded and the current location in the file will be set to the end. The user will then be able to write to the file.

APIs > File Manipulation Layer (FSIO) > Public Members > APPEND Macro

# APPENDPLUS Macro

C

```c
#define APPENDPLUS "a+"
```

## Description

If this macro is specified as the mode argument in a call of FSfopen, the file being opened will be created if it doesn't exist. If it does exist, it's file information will be loaded and the current location in the file will be set to the end. The user will then be able to write to the file or read from the file.

APIs > File Manipulation Layer (FSIO) > Public Members > APPENDPLUS Macro

# ATTR_ARCHIVE Macro

C

```c
#define ATTR_ARCHIVE 0x20
```

## Description

A macro for the archive attribute. This attribute will indicate to some archiving programs that the file with this attribute needs to be backed up. Most operating systems create files with the archive attribute set.

APIs > File Manipulation Layer (FSIO) > Public Members > ATTR_ARCHIVE Macro

# ATTR_DIRECTORY Macro

C

```c
#define ATTR_DIRECTORY 0x10
```

## Description

A macro for the directory attribute. If a directory entry has this attribute set, the file it points to is a directory- type file, and will contain directory entries that point to additional directories or files.

APIs > File Manipulation Layer (FSIO) > Public Members > ATTR_DIRECTORY Macro

# ATTR_HIDDEN Macro

C

```c
#define ATTR_HIDDEN 0x02
```

## Description

A macro for the hidden attribute. A file with this attribute may be hidden from the user, depending on the implementation of the operating system.

APIs > File Manipulation Layer (FSIO) > Public Members > ATTR_HIDDEN Macro

# ATTR_MASK Macro

C

```c
#define ATTR_MASK 0x3f
```

## Description

A macro for all attributes. The search functions in this library require an argument that determines which attributes a file is allowed to have in order to be found. If ATTR_MASK is specified as this argument, any file may be found, regardless of its attributes.

APIs > File Manipulation Layer (FSIO) > Public Members > ATTR_MASK Macro

# ATTR_READ_ONLY Macro

C

```c
#define ATTR_READ_ONLY 0x01
```

## Description

A macro for the read-only attribute. A file with this attribute should not be written to. Note that this attribute will not actually prevent a write to the file; that functionality is operating-system dependant. The user should take care not to write to a read-only file.

APIs > File Manipulation Layer (FSIO) > Public Members > ATTR_READ_ONLY Macro

# ATTR_SYSTEM Macro

C

```c
#define ATTR_SYSTEM 0x04
```

## Description

A macro for the system attribute. A file with this attribute is used by the operating system, and should not be modified. Note that this attribute will not actually prevent a write to the file.

APIs > File Manipulation Layer (FSIO) > Public Members > ATTR_SYSTEM Macro

# ATTR_VOLUME Macro

C

```c
#define ATTR_VOLUME 0x08
```

## Description

A macro for the volume attribute. If the first directory entry in the root directory has the volume attribute set, the device will use the name in that directory entry as the volume name.

APIs > File Manipulation Layer (FSIO) > Public Members > ATTR_VOLUME Macro

# EOF Macro

C

```c
#define EOF ((int)-1)
```

## Description

The EOF macro is used to indicate error conditions in some function calls. It is also used to indicate that the end-of-file has been reached.

APIs > File Manipulation Layer (FSIO) > Public Members > EOF Macro

# FALSE Macro

C

```c
#define FALSE 0
```

## Description

This macro will indicate that a condition is false.

APIs > File Manipulation Layer (FSIO) > Public Members > FALSE Macro

# FS_DYNAMIC_MEM Macro

C

```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.

APIs > File Manipulation Layer (FSIO) > Public Members > FS_DYNAMIC_MEM Macro

# FS_MAX_FILES_OPEN Macro

C

```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.

# INCREMENTTIMESTAMP Macro

C

```c
#define INCREMENTTIMESTAMP
```

## Description

The INCREMENTTIMESTAMP macro will set the create time of a file to a static value and increment it when a file is updated. This timestamp generation method should only be used in applications where file times are not necessary.

APIs > File Manipulation Layer (FSIO) > Public Members > INCREMENTTIMESTAMP Macro

# intmax_t Macro

C

```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.

APIs > File Manipulation Layer (FSIO) > Public Members > intmax_t Macro

# MDD_MediaDetect Macro

C

```c
#define MDD_MediaDetect USBHostMSDSCSIMediaDetect
```

## Description

Function pointer to the Media Detect Physical Layer function

APIs > File Manipulation Layer (FSIO) > Public Members > MDD_MediaDetect Macro

# MEDIA_SECTOR_SIZE Macro

C

```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.

APIs > File Manipulation Layer (FSIO) > Public Members > MEDIA_SECTOR_SIZE Macro

# NEAR_MODEL Macro

C

```
#define NEAR_MODEL
```

## Description

By uncommenting the NEAR_MODEL macro, the user can enable near-model RAM addressing when using dynamic FSFILE object allocation with PIC18

APIs > File Manipulation Layer (FSIO) > Public Members > NEAR_MODEL Macro

# READ Macro

C

```c
#define READ "r"
```

## Description

If this macro is specified as the mode argument in a call of FSfopen, the file information for the specified file will be loaded. If the file does not exist, the FSfopen function will fail. The user will then be able to read from the file.

# READPLUS Macro

C

```c
#define READPLUS "r+"
```

## Description

If this macro is specified as the mode argument in a call of FSfopen, the file information for the specified file will be loaded. If the file does not exist, the FSfopen function will fail. The user will then be able to read from the file or write to the file.

APIs > File Manipulation Layer (FSIO) > Public Members > READPLUS Macro

# SEEK_CUR Macro

**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

APIs > File Manipulation Layer (FSIO) > Public Members > SEEK_CUR Macro

# SEEK_END Macro

C

```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.

APIs > File Manipulation Layer (FSIO) > Public Members > SEEK_END Macro

# SEEK_SET Macro

C

```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.

APIs > File Manipulation Layer (FSIO) > Public Members > SEEK_SET Macro

# SUPPORT_FAT32 Macro

C

```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.

APIs > File Manipulation Layer (FSIO) > Public Members > SUPPORT_FAT32 Macro

# TRUE Macro

**C**

```
#define TRUE !FALSE
```

## Description

True value

APIs > File Manipulation Layer (FSIO) > Public Members > TRUE Macro

# USE_CF_INTERFACE_WITH_PMP Macro

C

```c
#define USE_CF_INTERFACE_WITH_PMP
```

## Description

Macro used to enable the CF-PMP physical layer (CF-PMP.c and .h)

APIs > File Manipulation Layer (FSIO) > Public Members > USE_CF_INTERFACE_WITH_PMP Macro

# USE_MANUAL_CF_INTERFACE Macro

C

```
#define USE_MANUAL_CF_INTERFACE
```

## Description

Macro used to enable the CF-Manual physical layer (CF-Bit transaction.c and .h)

APIs > File Manipulation Layer (FSIO) > Public Members > USE_MANUAL_CF_INTERFACE Macro

# USE_SD_INTERFACE_WITH_SPI Macro

C

```c
#define USE_SD_INTERFACE_WITH_SPI
```

## Description

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

APIs > File Manipulation Layer (FSIO) > Public Members > USE_SD_INTERFACE_WITH_SPI Macro

# USE_USB_INTERFACE Macro

C

```
#define USE_USB_INTERFACE
```

## Description

Macro used to enable the USB Host physical layer (USB host MSD library)

APIs > File Manipulation Layer (FSIO) > Public Members > USE_USB_INTERFACE Macro

# USERDEFINEDCLOCK Macro

C

```c
#define USERDEFINEDCLOCK
```

## Description

The USERDEFINEDCLOCK macro will allow the user to manually set timestamp information using the SetClockVars function. The user will need to set the time variables immediately before creating or closing a file or directory.

APIs > File Manipulation Layer (FSIO) > Public Members > USERDEFINEDCLOCK Macro

# USEREALTIMECLOCK Macro

C

```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.

APIs > File Manipulation Layer (FSIO) > Public Members > USEREALTIMECLOCK Macro

# WRITE Macro

C

```c
#define WRITE "w"
```

## Description

If this macro is specified as the mode argument in a call of FSfopen, the file being opened will be created if it doesn't exist. If it does exist, it will be erased and replaced by an empty file of the same name. The user will then be able to write to the file.

APIs > File Manipulation Layer (FSIO) > Public Members > WRITE Macro

# WRITEPLUS Macro

C

```c
#define WRITEPLUS "w+"
```

## Description

If this macro is specified as the mode argument in a call of FSfopen, the file being opened will be created if it doesn't exist. If it does exist, it will be erased and replaced by an empty file of the same name. The user will then be able to write to the file or read from the file.

APIs > File Manipulation Layer (FSIO) > Public Members > WRITEPLUS Macro

# FSFILE Structure

C

```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];
  WORD entry;
  WORD chk;
  WORD attributes;
  DWORD dirclus;
  DWORD dirccls;
} FSFILE;
```

## 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.

## 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]; | The name of the file |
| 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 |

# SearchRec Structure

C

```c
typedef struct {
  char filename[FILE_NAME_SIZE + 2];
  unsigned char attributes;
  unsigned long filesize;
  unsigned long timestamp;
  unsigned int entry;
  char searchname[FILE_NAME_SIZE + 2];
  unsigned char searchattr;
  unsigned int cwdclus;
  unsigned char initialized;
} SearchRec;
```

## 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.

## Members

| Members | Description |
|---|---|
| char filename[FILE_NAME_SIZE + 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) |
| unsigned int entry; | The directory entry of the last file found that matches the specified attributes. (Internal use only) |
| char searchname[FILE_NAME_SIZE + 2]; | The 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 int 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) |

# Library Members

The following functions, variables, structures, and macros are public, but are intended only to be accessed by the library itself. Applications should generally not call these functions or modify these variables.

## Enumerations

| | Name | Description |
|---|---|---|
| | CETYPE | An enumeration used for various error codes. |
| | _CETYPE | An enumeration used for various error codes. |
| | SEARCH_TYPE | Indicates how to search for file entries in the FILEfind function |

## Functions

| | Name | Description |
|---|---|---|
| | ReadByte | Read a byte from a buffer |
| | ReadDWord | Read a 32-bit double word from a buffer |
| | ReadWord | Read a 16-bit word from a buffer |
| | MDD_WriteProtectState | Function pointer that points to a physical layer's MDD_xxxxx_WriteProtectState function |

## Macros

| | Name | Descriptior |
|---|---|---|
| | ATTR_LONG_NAME | A macro for the attribute for a long-fil name entry |
| | BSI_BOOTSIG | A macro for the boot sector boot signature offset |
| | BSI_BPS | A macro for the boot sector bytes per sector value offset |
| | BSI_FAT32_BOOTSIG | A macro for the FAT32 boot sector boot signature offset |
| | BSI_FAT32_FSTYPE | A macro for the FAT32 boot sector file system type string offset |
| | BSI_FATCOUNT | A macro for the boot sector FAT |

| | | |
|---|---|---|
| | | count value offset |
| | BSI_FATSZ32 | A macro for the boot sector 32-bi sector per FAT value offset |
| | BSI_FSTYPE | A macro for the boot sector file system type string offset |
| | BSI_RESRVSEC | A macro for the boot sector reserved sector coun value offset |
| | BSI_ROOTCLUS | A macro for the boot sector start cluster of root directory value offset |
| | BSI_ROOTDIRENTS | A macro for the boot sector root directory entry count value offset |
| | | |

| | | |
|---|---|---|
| | BSI_SPC | A macro for the boot sector secto per cluster value offset |
| | BSI_SPF | A macro for the boot sector sectors per FAT value offset |
| | BSI_TOTSEC16 | A macro for the boot sector 16-bi total sector count value offset |
| | BSI_TOTSEC32 | A macro for the boot sector 32-bi total sector count value offset |
| | CE_EOF | Error that indicates that the end of the file has been reached |
| | | Error that indicates an attempt to read FAT |

| | | |
|---|---|---|
| | CE_FAT_EOF | entries beyond the end of the file |
| | CLUSTER_EMPTY | A macro to indicate an empty FAT entry |
| | CLUSTER_FAIL_FAT16 | A macro to indicate the failure of the ReadFAT function |
| | CLUSTER_FAIL_FAT32 | A macro to indicate the failure of the ReadFAT function |
| | DIR_DEL | A macro for a deleted di entry marker. |
| | DIR_EMPTY | A macro for the last dir entry marker. |
| | DIR_EXTENSION | A macro used to indicate the length of an 8.3 file |

| | | |
|---|---|---|
| | | extension |
| | DIR_NAMECOMP | A macro used to indicate the length of an 8.3 file nam and extension |
| | DIR_NAMESIZE | A macro used to indicate the length of an 8.3 file name |
| | END_CLUSTER_FAT12 | A macro to indicate the last allocatable cluster for FAT12 |
| | END_CLUSTER_FAT16 | A macro to indicate the last allocatable cluster for FAT16 |
| | END_CLUSTER_FAT32 | A macro to indicate the last allocatable cluster for FAT32 |
| | | |

| | | |
|---|---|---|
| | FAT_GOOD_SIGN_0 | A macro for the first boo sector/MBR signature byte |
| | FAT_GOOD_SIGN_1 | A macro for the second boot sector/MBR signature byte |
| | 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 | MAcro indicating the length o an 8.3 file name in a directory entry |

| | | |
|---|---|---|
| | FO_MBR | A macro indicating the offset fo the master boot record |
| | FOUND | A macro indicating a dir entry wa found |
| | GetInstructionClock | Instruction clock frequency |
| | GetPeripheralClock | Peripheral clock frequency |
| | GetSystemClock | System clock frequency (Hz) |
| | INPUT | A macro used to set TRIS register bits to input |
| | LAST_CLUSTER_FAT12 | A macro to indicate the last cluster value for FAT12 |
| | | A macro to |

| | | |
|---|---|---|
| | LAST_CLUSTER_FAT16 | indicate the last cluster value for FAT16 |
| | LAST_CLUSTER_FAT32 | A macro to indicate the last cluster value for FAT32 |
| | MASK_MAX_FILE_ENTRY_LIMIT_BITS | A mask that indicates the limit of directory entries in a sector |
| | 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 |
| | | A macro indicating |

| | | |
|---|---|---|
| | NO_MORE | that no more files were found |
| | NOT_FOUND | A macro indicating no dir entry was found |
| | NUMBER_OF_BYTES_IN_DIR_ENTRY | A macro indicating the number of bytes in a directory entry. |
| | OUTPUT | A macro used to set TRIS register bits to output |
| | RAMread | A macro to read a byte from RAM |
| | RAMreadD | A macro to read a 32-bit word from RAM |
| | RAMreadW | A macro to read a 16-bit word from RAM |
| | RAMwrite | A macro to write a byte |

| | | |
|---|---|---|
| | | to RAM |
| | TOTAL_FILE_SIZE | Macro indicating the length o a 8.3 file name |
| | VALUE_BASED_ON_ENTRIES_PER_CLUSTER | Value used for shift operations t calculate th sector offse in a directory |
| | VALUE_DOTDOT_CLUSTER_VALUE_FOR_ROOT | A value that will indicate that a dotdo directory entry points to the root. |

## Structures

| | Name | Description |
|---|---|---|
| | _BootSec | A structure of the organization of a boot sector. |
| | _BPB_FAT12 | A structure containing the bios parameter block for a FAT12 file system (in the boot sector) |
| | _BPB_FAT16 | A structure containing the bios parameter block for a FAT16 file system (in the boot sector) |

| | | |
|---|---|---|
| | _BPB_FAT32 | A structure containing the bios parameter block for a FAT32 file system (in the boot sector) |
| | _PT_MBR | A structure of the organization of a master boot record. |
| | DISK | A structure containing information about the device. |
| | FILEFLAGS | Indicates flag conditions for a file object |
| | PTE_MBR | A partition table entry structure. |
| | SWORD | A 24-bit data type |

## Types

| | Name | Description |
|---|---|---|
| | BootSec | A pointer to a _BootSec structure |
| | PT_MBR | A pointer to a _PT_MBR structure |
| | SALLOC | The segment header data type |

# ReadByte Function

C

```
BYTE ReadByte(
    BYTE* pBuffer,
    WORD index
);
```

## Description

Reads a byte from a buffer

## Preconditions

This function should not be called by the user.

## Parameters

| Parameters | Description |
|---|---|
| pBuffer | pointer to a buffer to read from |
| index | index in the buffer to read to |

## Returns

BYTE - the byte read

## Side Effects

None

## Remarks

None.

# ReadDWord Function

C

```
DWORD ReadDWord(
    BYTE* pBuffer,
    WORD index
);
```

## Description

Reads a 32-bit double word from a buffer

## Preconditions

This function should not be called by the user.

## Parameters

| Parameters | Description |
|---|---|
| pBuffer | pointer to a buffer to read from |
| index | index in the buffer to read to |

## Returns

DWORD - the double word read

## Side Effects

None

## Remarks

None.

# ReadWord Function

C

```
WORD ReadWord(
    BYTE* pBuffer,
    WORD index
);
```

## Description

Reads a 16-bit word from a buffer

## Preconditions

This function should not be called by the user.

## Parameters

| Parameters | Description |
|---|---|
| pBuffer | pointer to a buffer to read from |
| index | index in the buffer to read to |

## Returns

WORD - the word read

## Side Effects

None

## Remarks

None.

# ATTR_LONG_NAME Macro

C

```c
#define ATTR_LONG_NAME 0x0f
```

## Description

A macro for the long-name attributes. If a directory entry is used in a long-file name implementation, it will have all four lower bits set. This indicates that any software that does not support long file names should ignore that entry.

APIs > File Manipulation Layer (FSIO) > Library Members > ATTR_LONG_NAME Macro

# BSI_BOOTSIG Macro

C

```c
#define BSI_BOOTSIG 38
```

## Description

A macro for the boot sector boot signature offset

APIs > File Manipulation Layer (FSIO) > Library Members > BSI_BOOTSIG Macro

# BSI_BPS Macro

C

```
#define BSI_BPS 11
```

## Description

A macro for the boot sector bytes per sector value offset

APIs > File Manipulation Layer (FSIO) > Library Members > BSI_BPS Macro

# BSI_FAT32_BOOTSIG Macro

C

```c
#define BSI_FAT32_BOOTSIG 66
```

## Description

A macro for the FAT32 boot sector boot signature offset

APIs > File Manipulation Layer (FSIO) > Library Members > BSI_FAT32_BOOTSIG Macro

# BSI_FAT32_FSTYPE Macro

**C**

```
#define BSI_FAT32_FSTYPE 82
```

## Description

A macro for the FAT32 boot sector file system type string offset

APIs > File Manipulation Layer (FSIO) > Library Members > BSI_FAT32_FSTYPE Macro

# BSI_FATCOUNT Macro

C

```c
#define BSI_FATCOUNT 16
```

## Description

A macro for the boot sector FAT count value offset

APIs > File Manipulation Layer (FSIO) > Library Members > BSI_FATCOUNT Macro

# BSI_FATSZ32 Macro

C

```
#define BSI_FATSZ32 36
```

## Description

A macro for the boot sector 32-bit sector per FAT value offset

APIs > File Manipulation Layer (FSIO) > Library Members > BSI_FATSZ32 Macro

# BSI_FSTYPE Macro

C

```c
#define BSI_FSTYPE 54
```

## Description

A macro for the boot sector file system type string offset

APIs > File Manipulation Layer (FSIO) > Library Members > BSI_FSTYPE Macro

# BSI_RESRVSEC Macro

C

```c
#define BSI_RESRVSEC 14
```

## Description

A macro for the boot sector reserved sector count value offset

APIs > File Manipulation Layer (FSIO) > Library Members > BSI_RESRVSEC Macro

# BSI_ROOTCLUS Macro

C

```c
#define BSI_ROOTCLUS 44
```

## Description

A macro for the boot sector start cluster of root directory value offset

APIs > File Manipulation Layer (FSIO) > Library Members > BSI_ROOTCLUS Macro

# BSI_ROOTDIRENTS Macro

C

```
#define BSI_ROOTDIRENTS 17
```

## Description

A macro for the boot sector root directory entry count value offset

APIs > File Manipulation Layer (FSIO) > Library Members > BSI_ROOTDIRENTS Macro

# BSI_SPC Macro

C

```
#define BSI_SPC 13
```

## Description

A macro for the boot sector sector per cluster value offset

APIs > File Manipulation Layer (FSIO) > Library Members > BSI_SPC Macro

# BSI_SPF Macro

C

```c
#define BSI_SPF 22
```

## Description

A macro for the boot sector sectors per FAT value offset

APIs > File Manipulation Layer (FSIO) > Library Members > BSI_SPF Macro

# BSI_TOTSEC16 Macro

C

```c
#define BSI_TOTSEC16 19
```

## Description

A macro for the boot sector 16-bit total sector count value offset

APIs > File Manipulation Layer (FSIO) > Library Members > BSI_TOTSEC16 Macro

# BSI_TOTSEC32 Macro

C

```c
#define BSI_TOTSEC32 32
```

## Description

A macro for the boot sector 32-bit total sector count value offset

APIs > File Manipulation Layer (FSIO) > Library Members > BSI_TOTSEC32 Macro

# CE_EOF Macro

C

```c
#define CE_EOF 61   // Error that indicates that the
```

## Description

Error that indicates that the end of the file has been reached

APIs > File Manipulation Layer (FSIO) > Library Members > CE_EOF Macro

# CE_FAT_EOF Macro

C

```c
#define CE_FAT_EOF 60   // Error that indicates an a
```

## Description

Error that indicates an attempt to read FAT entries beyond the end of the file

APIs > File Manipulation Layer (FSIO) > Library Members > CE_FAT_EOF Macro

# CLUSTER_EMPTY Macro

C

```c
#define CLUSTER_EMPTY 0x0000
```

## Description

The CLUSTER_EMPTY value is used to indicate that a FAT entry and it's corresponding cluster are available.

APIs > File Manipulation Layer (FSIO) > Library Members > CLUSTER_EMPTY Macro

# CLUSTER_FAIL_FAT16 Macro

C

```c
#define CLUSTER_FAIL_FAT16 0xFFFF
```

## Description

The CLUSTER_FAIL_FAT16 macro is used by the ReadFAT function to indicate that an error occured reading a FAT12 or FAT16 file allocation table. Note that since '0xFFF8' is used for the last cluster return value in the FAT16 implementation the end-of-file value '0xFFFF' can be used to indicate an error condition.

APIs > File Manipulation Layer (FSIO) > Library Members > CLUSTER_FAIL_FAT16 Macro

# CLUSTER_FAIL_FAT32 Macro

C

```c
#define CLUSTER_FAIL_FAT32 0x0FFFFFFF
```

## Description

The CLUSTER_FAIL_FAT32 macro is used by the ReadFAT function to indicate that an error occured reading a FAT32 file allocation able.

APIs > File Manipulation Layer (FSIO) > Library Members > CLUSTER_FAIL_FAT32 Macro

# DIR_DEL Macro

C

```c
#define DIR_DEL 0xE5
```

## Description

The DIR_DEL macro is used to mark a directory entry as deleted. When a file is deleted, this value will replace the first character in the file name, and will indicate that the file the entry points to was deleted.

APIs > File Manipulation Layer (FSIO) > Library Members > DIR_DEL Macro

# DIR_EMPTY Macro

C

```c
#define DIR_EMPTY 0
```

## Description

The DIR_EMPTY macro is used to indicate the last entry in a directory. Since entries in use cannot start with a 0 and deleted entries start with the DIR_DEL character, a 0 will mark the end of the in-use or previously used group of entries in a directory

APIs > File Manipulation Layer (FSIO) > Library Members > DIR_EMPTY Macro

# DIR_EXTENSION Macro

C

```c
#define DIR_EXTENSION 3
```

## Description

The DIR_EXTENSION macro is used when validating the extension portion of 8.3 filenames

APIs > File Manipulation Layer (FSIO) > Library Members > DIR_EXTENSION Macro

# DIR_NAMECOMP Macro

C

```c
#define DIR_NAMECOMP (DIR_NAMESIZE+DIR_EXTENSION)
```

## Description

The DIR_NAMECOMP macro is used when validating 8.3 filenames

APIs > File Manipulation Layer (FSIO) > Library Members > DIR_NAMECOMP Macro

# DIR_NAMESIZE Macro

C

```c
#define DIR_NAMESIZE 8
```

## Description

The DIR_NAMESIZE macro is used when validing the name portion of 8.3 filenames

APIs > File Manipulation Layer (FSIO) > Library Members > DIR_NAMESIZE Macro

# END_CLUSTER_FAT12 Macro

C

```c
#define END_CLUSTER_FAT12 0xFF7
```

## Description

The END_CLUSTER_FAT12 value is used as a comparison in FAT12 to determine that the firmware has reached the end of the range of allowed allocatable clusters.

APIs > File Manipulation Layer (FSIO) > Library Members > END_CLUSTER_FAT12 Macro

# END_CLUSTER_FAT16 Macro

C

```c
#define END_CLUSTER_FAT16 0xFFF7
```

## Description

The END_CLUSTER_FAT16 value is used as a comparison in FAT16 to determine that the firmware has reached the end of the range of allowed allocatable clusters.

APIs > File Manipulation Layer (FSIO) > Library Members > END_CLUSTER_FAT16 Macro

# END_CLUSTER_FAT32 Macro

C

```c
#define END_CLUSTER_FAT32 0x0FFFFFF7
```

## Description

The END_CLUSTER_FAT32 value is used as a comparison in FAT32 to determine that the firmware has reached the end of the range of allowed allocatable clusters.

APIs > File Manipulation Layer (FSIO) > Library Members > END_CLUSTER_FAT32 Macro

# FAT_GOOD_SIGN_0 Macro

C

```c
#define FAT_GOOD_SIGN_0 0x55
```

## Description

The FAT_GOOD_SIGN_0 macro is used to determine that the first byte of the MBR or boot sector signature code is correct

APIs > File Manipulation Layer (FSIO) > Library Members > FAT_GOOD_SIGN_0 Macro

# FAT_GOOD_SIGN_1 Macro

C

```c
#define FAT_GOOD_SIGN_1 0xAA
```

## Description

The FAT_GOOD_SIGN_1 macro is used to determine that the second byte of the MBR or boot sector signature code is correct

APIs > File Manipulation Layer (FSIO) > Library Members > FAT_GOOD_SIGN_1 Macro

# FAT12 Macro

C

```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.

APIs > File Manipulation Layer (FSIO) > Library Members > FAT12 Macro

# FAT16 Macro

C

```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.

APIs > File Manipulation Layer (FSIO) > Library Members > FAT16 Macro

# FAT32 Macro

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.

APIs > File Manipulation Layer (FSIO) > Library Members > FAT32 Macro

# FILE_NAME_SIZE Macro

C

```c
#define FILE_NAME_SIZE 11
```

## Description

The FILE_NAME_SIZE 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.

APIs > File Manipulation Layer (FSIO) > Library Members > FILE_NAME_SIZE Macro

# FO_MBR Macro

C

```c
#define FO_MBR 0L
```

## Description

FO_MBR is a macro that indicates the addresss of the master boot record on the device. When the device is initialized this sector will be read

APIs > File Manipulation Layer (FSIO) > Library Members > FO_MBR Macro

# FOUND Macro

C

```c
#define FOUND 0
```

## Description

The FOUND macro indicates that a directory entry was found in the specified position

APIs > File Manipulation Layer (FSIO) > Library Members > FOUND Macro

# GetInstructionClock Macro

C

```c
#define GetInstructionClock (GetSystemClock())
```

## Description

Instruction clock frequency

APIs > File Manipulation Layer (FSIO) > Library Members > GetInstructionClock Macro

# GetPeripheralClock Macro

C

```c
#define GetPeripheralClock (GetSystemClock())
```

## Description

Peripheral clock frequency

APIs > File Manipulation Layer (FSIO) > Library Members > GetPeripheralClock Macro

# GetSystemClock Macro

C

```c
#define GetSystemClock (60000000ul)
```

## Description

System clock frequency (Hz)

APIs > File Manipulation Layer (FSIO) > Library Members > GetSystemClock Macro

# INPUT Macro

C

```c
#define INPUT 1
```

## Description

A macro used to set TRIS register bits to input

APIs > File Manipulation Layer (FSIO) > Library Members > INPUT Macro

# LAST_CLUSTER_FAT12 Macro

C

```c
#define LAST_CLUSTER_FAT12 0xff8
```

## Description

The LAST_CLUSTER_FAT12 macro is used when reading the FAT to indicate that the next FAT12 entry for a file contains the end-of-file value.

APIs > File Manipulation Layer (FSIO) > Library Members > LAST_CLUSTER_FAT12 Macro

# LAST_CLUSTER_FAT16 Macro

C

```c
#define LAST_CLUSTER_FAT16 0xfff8
```

## Description

The LAST_CLUSTER_FAT16 macro is used when reading the FAT to indicate that the next FAT16 entry for a file contains the end-of-file value.

APIs > File Manipulation Layer (FSIO) > Library Members > LAST_CLUSTER_FAT16 Macro

# LAST_CLUSTER_FAT32 Macro

C

```c
#define LAST_CLUSTER_FAT32 0x0FFFFFF8
```

## Description

The LAST_CLUSTER_FAT32 macro is used when reading the FAT to indicate that the next FAT32 entry for a file contains the end-of-file value.

APIs > File Manipulation Layer (FSIO) > Library Members > LAST_CLUSTER_FAT32 Macro

# MASK_MAX_FILE_ENTRY_LIMIT_BITS Macro

C

```c
#define MASK_MAX_FILE_ENTRY_LIMIT_BITS 0x0f
```

## Description

The MASK_MAX_FILE_ENTRY_LIMIT_BITS is used to indicate to the Cache_File_Entry function that a new sector needs to be loaded.

APIs > File Manipulation Layer (FSIO) > Library Members > MASK_MAX_FILE_ENTRY_LIMIT_BITS Macro

# MDD_InitIO Macro

C

```c
#define MDD_InitIO ;
```

## Description

Function pointer to the I/O Initialization Physical Layer function

APIs > File Manipulation Layer (FSIO) > Library Members > MDD_InitIO Macro

# MDD_MediaInitialize Macro

C

```
#define MDD_MediaInitialize USBHostMSDSCSIMediaInitia
```

## Description

Function pointer to the Media Initialize Physical Layer function

APIs > File Manipulation Layer (FSIO) > Library Members > MDD_MediaInitialize Macro

# MDD_ReadCapacity Macro

C

```
#define MDD_ReadCapacity MDD_SDSPI_ReadCapacity
```

## Description

Function pointer to the Read Capacity Physical Layer function

APIs > File Manipulation Layer (FSIO) > Library Members > MDD_ReadCapacity Macro

# MDD_ReadSectorSize Macro

C

```
#define MDD_ReadSectorSize MDD_SDSPI_ReadSectorSize
```

## Description

Function pointer to the Read Sector Size Physical Layer Function

APIs > File Manipulation Layer (FSIO) > Library Members > MDD_ReadSectorSize Macro

# MDD_SectorRead Macro

C

```
#define MDD_SectorRead USBHostMSDSCSISectorRead
```

## Description

Function pointer to the Sector Read Physical Layer function

APIs > File Manipulation Layer (FSIO) > Library Members > MDD_SectorRead Macro

# MDD_SectorWrite Macro

C

```c
#define MDD_SectorWrite USBHostMSDSCSISectorWrite
```

## Description

Function pointer to the Sector Write Physical Layer function

APIs > File Manipulation Layer (FSIO) > Library Members > MDD_SectorWrite Macro

# MDD_ShutdownMedia Macro

C

```
#define MDD_ShutdownMedia USBHostMSDSCSIMediaReset
```

## Description

Function pointer to the Media Shutdown Physical Layer function

APIs > File Manipulation Layer (FSIO) > Library Members > MDD_ShutdownMedia Macro

# MDD_WriteProtectState Function

**C**

```
BYTE MDD_WriteProtectState();
```

## Description

Function pointer that points to a physical layer's MDD_xxxxx_WriteProtectState function

APIs > File Manipulation Layer (FSIO) > Library Members > MDD_WriteProtectState Function

# NO_MORE Macro

C

```c
#define NO_MORE 2
```

## Description

The NO_MORE macro indicates that there are no more directory entries to search for

APIs > File Manipulation Layer (FSIO) > Library Members > NO_MORE Macro

# NOT_FOUND Macro

C

```c
#define NOT_FOUND 1
```

## Description

The NOT_FOUND macro indicates that the specified directory entry to load was deleted

APIs > File Manipulation Layer (FSIO) > Library Members > NOT_FOUND Macro

# NUMBER_OF_BYTES_IN_DIR_ENTRY Macro

C

```c
#define NUMBER_OF_BYTES_IN_DIR_ENTRY 32
```

## Description

The NUMBER_OF_BYTES_IN_DIR_ENTRY macro represents the number of bytes in one directory entry. It is used to calculate the number of sectors in the root directory based on information in the boot sector.

APIs > File Manipulation Layer (FSIO) > Library Members > NUMBER_OF_BYTES_IN_DIR_ENTRY Macro

# OUTPUT Macro

C

```c
#define OUTPUT 0
```

## Description

A macro used to set TRIS register bits to output

APIs > File Manipulation Layer (FSIO) > Library Members > OUTPUT Macro

# RAMread Macro

C

```
#define RAMread( a, f) *(a+f)
```

## Description

The RAMread macro is used to read a byte of data from a RAM array

APIs > File Manipulation Layer (FSIO) > Library Members > RAMread Macro

# RAMreadD Macro

C

```c
#define RAMreadD( a, f) *(DWORD *)(a+f)
```

## Description

The RAMreadD macro is used to read four bytes of data from a RAM array

APIs > File Manipulation Layer (FSIO) > Library Members > RAMreadD Macro

# RAMreadW Macro

C

```c
#define RAMreadW( a, f) *(WORD *)(a+f)
```

## Description

The RAMreadW macro is used to read two bytes of data from a RAM array

APIs > File Manipulation Layer (FSIO) > Library Members > RAMreadW Macro

# RAMwrite Macro

C

```c
#define RAMwrite( a, f, d) *(a+f) = d
```

## Description

The RAMwrite macro is used to write a byte of data to a RAM array

APIs > File Manipulation Layer (FSIO) > Library Members > RAMwrite Macro

# TOTAL_FILE_SIZE Macro

C

```c
#define TOTAL_FILE_SIZE 8+3+1
```

## Description

The TOTAL_FILE_SIZE macro indicates the maximum number of characters in an 8.3 file name. This value includes 8 characters for the name, three for the extentsion, and 1 for the radix ('.')

APIs > File Manipulation Layer (FSIO) > Library Members > TOTAL_FILE_SIZE Macro

# VALUE_BASED_ON_ENTRIES_PER_CLUSTER Macro

C

```c
#define VALUE_BASED_ON_ENTRIES_PER_CLUSTER 4
```

## Description

The VALUE_BASED_ON_ENTRIES_PER_CLUSTER macro is used to calculate sector offsets for directories. The position of the entry is shifted by 4 bits (divided by 16, since there are 16 entries in a sector) to calculate how many sectors a specified entry is offset from the beginning of the directory.

APIs > File Manipulation Layer (FSIO) > Library Members > VALUE_BASED_ON_ENTRIES_PER_CLUSTER Macro

# VALUE_DOTDOT_CLUSTER_VALUE_FOR_ROOT Macro

C

```c
#define VALUE_DOTDOT_CLUSTER_VALUE_FOR_ROOT 0
```

## Description

The VALUE_DOTDOT_CLUSTER_VALUE_FOR_ROOT macro is used as an absolute address when writing information to a dotdot entry in a newly created directory. If a dotdot entry points to the root directory, it's cluster value must be set to 0, regardless of the actual cluster number of the root directory.

APIs > File Manipulation Layer (FSIO) > Library Members > VALUE_DOTDOT_CLUSTER_VALUE_FOR_ROOT Macro

# _BootSec Structure

C

```c
typedef struct {
  union {
    _BPB_FAT32 FAT_32;
    _BPB_FAT16 FAT_16;
    _BPB_FAT12 FAT_12;
  } FAT;
  BYTE Reserved[MEDIA_SECTOR_SIZE-sizeof(_BPB_FAT32)
  BYTE Signature0;
  BYTE Signature1;
} _BootSec;
```

## Description

The _BootSec structure has the same form as a boot sector. When the boot sector is loaded from the device, it will be cast as a _BootSec structure so the boot sector elements can be accessed.

## Members

| Members | Description |
|---|---|
| union {<br>_BPB_FAT32 FAT_32;<br>_BPB_FAT16 FAT_16;<br>_BPB_FAT12 FAT_12;<br>} FAT; | A union of different bios parameter blocks |
| BYTE Reserved[MEDIA_SECTOR_SIZE-sizeof(_BPB_FAT32)-2]; | Reserved space |
| | |

| | |
|---|---|
| BYTE Signature0; | Boot sector signature code - equal to 0x55 |
| BYTE Signature1; | Boot sector signature code - equal to 0xAA |

# _BPB_FAT12 Structure

C

```c
typedef struct {
  SWORD BootSec_JumpCmd;
  BYTE BootSec_OEMName[8];
  WORD BootSec_BPS;
  BYTE BootSec_SPC;
  WORD BootSec_ResrvSec;
  BYTE BootSec_FATCount;
  WORD BootSec_RootDirEnts;
  WORD BootSec_TotSec16;
  BYTE BootSec_MDesc;
  WORD BootSec_SPF;
  WORD BootSec_SPT;
  WORD BootSec_HeadCnt;
  DWORD BootSec_HiddenSecCnt;
  DWORD BootSec_Reserved;
  BYTE BootSec_DriveNum;
  BYTE BootSec_Reserved2;
  BYTE BootSec_BootSig;
  BYTE BootSec_VolID[4];
  BYTE BootSec_VolLabel[11];
  BYTE BootSec_FSType[8];
} _BPB_FAT12;
```

## Description

The _BPB_FAT12 structure provides a layout of the "bios parameter block" in the boot sector of a FAT12 partition.

## Members

| Members | Description |
|---|---|
| | |

| | |
|---|---|
| SWORD BootSec_JumpCmd; | Jump Command |
| BYTE BootSec_OEMName[8]; | OEM name |
| WORD BootSec_BPS; | Number of bytes per sector |
| BYTE BootSec_SPC; | Number of sectors per cluster |
| WORD BootSec_ResrvSec; | Number of reserved sectors at the beginning of the partition |
| BYTE BootSec_FATCount; | Number of FATs on the partition |
| WORD BootSec_RootDirEnts; | Number of root directory entries |
| WORD BootSec_TotSec16; | Total number of sectors |
| BYTE BootSec_MDesc; | Media descriptor |
| WORD BootSec_SPF; | Number of sectors per FAT |
| WORD BootSec_SPT; | Number of sectors per track |
| WORD BootSec_HeadCnt; | Number of heads |
| DWORD BootSec_HiddenSecCnt; | Number of hidden sectors |
| DWORD BootSec_Reserved; | Reserved space |
| BYTE BootSec_DriveNum; | Drive number |

| | |
|---|---|
| BYTE BootSec_Reserved2; | Reserved space |
| BYTE BootSec_BootSig; | Boot signature - equal to 0x29 |
| BYTE BootSec_VolID[4]; | Volume ID |
| BYTE BootSec_VolLabel[11]; | Volume Label |
| BYTE BootSec_FSType[8]; | File system type in ASCII. Not used for determination |

# _BPB_FAT16 Structure

C

```c
typedef struct {
  SWORD BootSec_JumpCmd;
  BYTE BootSec_OEMName[8];
  WORD BootSec_BPS;
  BYTE BootSec_SPC;
  WORD BootSec_ResrvSec;
  BYTE BootSec_FATCount;
  WORD BootSec_RootDirEnts;
  WORD BootSec_TotSec16;
  BYTE BootSec_MDesc;
  WORD BootSec_SPF;
  WORD BootSec_SPT;
  WORD BootSec_HeadCnt;
  DWORD BootSec_HiddenSecCnt;
  DWORD BootSec_TotSec32;
  BYTE BootSec_DriveNum;
  BYTE BootSec_Reserved;
  BYTE BootSec_BootSig;
  BYTE BootSec_VolID[4];
  BYTE BootSec_VolLabel[11];
  BYTE BootSec_FSType[8];
} _BPB_FAT16;
```

## Description

The _BPB_FAT16 structure provides a layout of the "bios parameter block" in the boot sector of a FAT16 partition.

## Members

| Members | Description |
|---|---|
| | |

| | |
|---|---|
| SWORD BootSec_JumpCmd; | Jump Command |
| BYTE BootSec_OEMName[8]; | OEM name |
| WORD BootSec_BPS; | Number of bytes per sector |
| BYTE BootSec_SPC; | Number of sectors per cluster |
| WORD BootSec_ResrvSec; | Number of reserved sectors at the beginning of the partition |
| BYTE BootSec_FATCount; | Number of FATs on the partition |
| WORD BootSec_RootDirEnts; | Number of root directory entries |
| WORD BootSec_TotSec16; | Total number of sectors |
| BYTE BootSec_MDesc; | Media descriptor |
| WORD BootSec_SPF; | Number of sectors per FAT |
| WORD BootSec_SPT; | Number of sectors per track |
| WORD BootSec_HeadCnt; | Number of heads |
| DWORD BootSec_HiddenSecCnt; | Number of hidden sectors |
| DWORD BootSec_TotSec32; | Total sector count (32 bits) |
| BYTE BootSec_DriveNum; | Drive number |

| | |
|---|---|
| BYTE BootSec_Reserved; | Reserved space |
| BYTE BootSec_BootSig; | Boot signature - equal to 0x29 |
| BYTE BootSec_VolID[4]; | Volume ID |
| BYTE BootSec_VolLabel[11]; | Volume Label |
| BYTE BootSec_FSType[8]; | File system type in ASCII. Not used for determination |

# _BPB_FAT32 Structure

C

```c
typedef struct {
  SWORD BootSec_jmpBoot;
  BYTE BootSec_OEMName[8];
  WORD BootSec_BytsPerSec;
  BYTE BootSec_SecPerClus;
  WORD BootSec_RsvdSecCnt;
  BYTE BootSec_NumFATs;
  WORD BootSec_RootEntCnt;
  WORD BootSec_TotSec16;
  BYTE BootSec_Media;
  WORD BootSec_FATSz16;
  WORD BootSec_SecPerTrk;
  WORD BootSec_NumHeads;
  DWORD BootSec_HiddSec;
  DWORD BootSec_TotSec32;
  DWORD BootSec_FATSz32;
  WORD BootSec_ExtFlags;
  WORD BootSec_FSVers;
  DWORD BootSec_RootClus;
  WORD BootSec_FSInfo;
  WORD BootSec_BkBootSec;
  BYTE BootSec_Reserved[12];
  BYTE BootSec_DrvNum;
  BYTE BootSec_Reserved1;
  BYTE BootSec_BootSig;
  BYTE BootSec_VolID[4];
  BYTE BootSec_VolLab[11];
  BYTE BootSec_FilSysType[8];
} _BPB_FAT32;
```

## Description

The _BPB_FAT32 structure provides a layout of the "bios parameter block" in the boot sector of a FAT32 partition.

## Members

| Members | Description |
|---|---|
| SWORD BootSec_jmpBoot; | Jump Command |
| BYTE BootSec_OEMName[8]; | OEM name |
| WORD BootSec_BytsPerSec; | Number of bytes per sector |
| BYTE BootSec_SecPerClus; | Number of sectors per cluster |
| WORD BootSec_RsvdSecCnt; | Number of reserved sectors at the beginning of the partition |
| BYTE BootSec_NumFATs; | Number of FATs on the partition |
| WORD BootSec_RootEntCnt; | Number of root directory entries |
| WORD BootSec_TotSec16; | Total number of sectors |
| BYTE BootSec_Media; | Media descriptor |
| WORD BootSec_FATSz16; | Number of sectors per FAT |
| WORD BootSec_SecPerTrk; | Number of sectors per track |

| | |
|---|---|
| WORD BootSec_NumHeads; | Number of heads |
| DWORD BootSec_HiddSec; | Number of hidden sectors |
| DWORD BootSec_TotSec32; | Total sector count (32 bits) |
| DWORD BootSec_FATSz32; | Sectors per FAT (32 bits) |
| WORD BootSec_ExtFlags; | Presently active FAT. Defined by bits 0-3 if bit 7 is 1. |
| WORD BootSec_FSVers; | FAT32 filesystem version. Should be 0:0 |
| DWORD BootSec_RootClus; | Start cluster of the root directory (should be 2) |
| WORD BootSec_FSInfo; | File system information |
| WORD BootSec_BkBootSec; | Backup boot sector address. |
| BYTE BootSec_Reserved[12]; | Reserved space |
| BYTE BootSec_DrvNum; | Drive number |
| BYTE BootSec_Reserved1; | Reserved space |
| BYTE BootSec_BootSig; | Boot signature - 0x29 |

| | |
|---|---|
| BYTE BootSec_VolID[4]; | Volume ID |
| BYTE BootSec_VolLab[11]; | Volume Label |
| BYTE BootSec_FilSysType[8]; | File system type in ASCII. Not used for determination |

# _PT_MBR Structure

C

```c
typedef struct {
  BYTE ConsChkRtn[446];
  PTE_MBR Partition0;
  PTE_MBR Partition1;
  PTE_MBR Partition2;
  PTE_MBR Partition3;
  BYTE Signature0;
  BYTE Signature1;
} _PT_MBR;
```

## Description

The _PT_MBR structure has the same form as a master boot record. When the MBR is loaded from the device, it will be cast as a _PT_MBR structure so the MBR elements can be accessed.

## Members

| Members | Description |
|---|---|
| BYTE ConsChkRtn[446]; | Boot code |
| PTE_MBR Partition0; | The first partition table entry |
| PTE_MBR Partition1; | The second partition table entry |
| PTE_MBR Partition2; | The third partition table entry |
| PTE_MBR Partition3; | The fourth partition table entry |
| BYTE Signature0; | MBR signature code - equal to 0x55 |

| | |
|---|---|
| BYTE Signature1; | MBR signature code - equal to 0xAA |

# BootSec Type

C

```
typedef _BootSec * BootSec;
```

## Description

The BootSec pointer points to a _BootSec structure.

APIs > File Manipulation Layer (FSIO) > Library Members > BootSec Type

# CETYPE Enumeration

C

```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
} CETYPE;
```

## Description

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

## 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 |

# DISK Structure

C

```c
typedef struct {
  BYTE * buffer;
  DWORD firsts;
  DWORD fat;
  DWORD root;
  DWORD data;
  WORD maxroot;
  DWORD maxcls;
  WORD fatsize;
  BYTE fatcopy;
  BYTE SecPerClus;
  BYTE type;
  BYTE mount;
} DISK;
```

## Description

The DISK structure contains information about the device being accessed.

## Members

| Members | Description |
|---|---|
| BYTE * buffer; | Address of the global data buffer used to read and write file information |
| DWORD firsts; | Logical block address of the first sector of the FAT partition on the device |
| DWORD fat; | Logical block address of the FAT |
| | |

| | |
|---|---|
| DWORD root; | Logical block address of the root directory |
| DWORD data; | Logical block address of the data section of the device. |
| WORD maxroot; | The maximum number of entries in the root directory. |
| DWORD maxcls; | The maximum number of clusters in the partition. |
| WORD fatsize; | The number of sectors in the FAT |
| BYTE fatcopy; | The number of copies of the FAT in the partition |
| BYTE SecPerClus; | The number of sectors per cluster in the data region |
| BYTE type; | The file system type of the partition (FAT12, FAT16 or FAT32) |
| BYTE mount; | Device mount flag (TRUE if disk was mounted successfully, FALSE otherwise) |

# FILEFLAGS Structure

C

```c
typedef struct {
  unsigned write : 1;
  unsigned read : 1;
  unsigned FileWriteEOF : 1;
} FILEFLAGS;
```

## Description

The FILEFLAGS structure is used to indicate conditions in a file. It contains three flags: 'write' indicates that the file was opened in a mode that allows writes, 'read' indicates that the file was opened in a mode that allows reads, and 'FileWriteEOF' indicates that additional data that is written to the file will increase the file size.

## Members

| Members | Description |
|---|---|
| unsigned write : 1; | Indicates a file was opened in a mode that allows writes |
| unsigned read : 1; | Indicates a file was opened in a mode that allows reads |
| unsigned FileWriteEOF : 1; | Indicates the current position in a file is at the end of the file |

APIs > File Manipulation Layer (FSIO) > Library Members > FILEFLAGS Structure

# PT_MBR Type

C

```
typedef _PT_MBR * PT_MBR;
```

## Description

The PT_MBR pointer points to a _PT_MBR structure.

APIs > File Manipulation Layer (FSIO) > Library Members > PT_MBR Type

# PTE_MBR Structure

C

```c
typedef struct {
  BYTE PTE_BootDes;
  SWORD PTE_FrstPartSect;
  BYTE PTE_FSDesc;
  SWORD PTE_LstPartSect;
  DWORD PTE_FrstSect;
  DWORD PTE_NumSect;
} PTE_MBR;
```

## Description

The PTE_MBR structure contains values found in a partition table entry in the MBR of a device.

## Members

| Members | Description |
|---|---|
| BYTE PTE_BootDes; | The boot descriptor (should be 0x00 in a non-bootable device) |
| SWORD PTE_FrstPartSect; | The cylinder-head-sector address of the first sector of the partition |
| BYTE PTE_FSDesc; | The file system descriptor |
| SWORD PTE_LstPartSect; | The cylinder-head-sector address of the last sector of the partition |
| DWORD PTE_FrstSect; | The logical block address of the first sector of the partition |
| | |

| | |
|---|---|
| DWORD PTE_NumSect; | The number of sectors in a partition |

# SALLOC Type

C

```
typedef union _SALLOC SALLOC;
```

## Description

The SALLOC union allows the PIC18 dynamic memory allocation algorithm to perform bitwise accesses on segment headers.

APIs > File Manipulation Layer (FSIO) > Library Members > SALLOC Type

# SEARCH_TYPE Enumeration

C

```c
typedef enum {
  LOOK_FOR_EMPTY_ENTRY = 0,
  LOOK_FOR_MATCHING_ENTRY
} SEARCH_TYPE;
```

## Description

The values in the SEARCH_TYPE enumeration are used internally by the library to indicate how the FILEfind function how to perform a search. The 'LOOK_FOR_EMPTY_ENTRY' value indicates that FILEfind should search for an empty file entry. The 'LOOK_FOR_MATCHING_ENTRY' value indicates that FILEfind should search for an entry that matches the FSFILE object that was passed into the FILEfind function.

APIs > File Manipulation Layer (FSIO) > Library Members > SEARCH_TYPE Enumeration

# Internal Members

The following functions, variables, structures, and macros are designated as internal to the library.

## Functions

| | Name | Description |
|---|---|---|
| | _SRAMmerge | This function tries to merge adjacent segments that have not been allocated. The largest possible segment is merged if possible. |
| | Cache_File_Entry | Load a file entry |
| | CacheTime | Automatically store timestamp information from the RTCC |
| | chdirhelper | Helper function for FSchdir |
| | Cluster2Sector | Convert a cluster number to the corresponding sector |
| | CreateDIR | FSmkdir helper function to create a directory |
| | CreateFileEntry | Create a new file entry |
| | CreateFirstCluster | Create the first cluster for a file |
| | DISKmount | Initialies the device and loads MBR and boot sector information |
| | EraseCluster | Erase a cluster |

| | | |
|---|---|---|
| | FAT_erase_cluster_chain | Erase a chain of clusters |
| | FATfindEmptyCluster | Find the next available cluster on the device |
| | FILEallocate_new_cluster | This function will find an empty cluster on the device using the FATfindEmptyCluster function. It will then mark it as the last cluster in the file in the FAT chain, and link the current last cluster of the passed file to the new cluster. If the new cluster is a directory cluster, it will be erased (so there are no extraneous directory entries). If it's allocated to a non-directory file, it doesn't need to be erased; extraneous data in the cluster will be unviewable because of the file size parameter. |
| | FILECreateHeadCluster | Create the first cluster of a file |
| | FILEerase | Erase a file |
| | FILEfind | Finds a file on the device |
| | FILEget_next_cluster | Step through a chain of clusters |
| | FileObjectCopy | Copy a file object |
| | FILEopen | Loads file information from the device |
| | eraseDir | FSrmdir helper function to erase dirs |
| | Fill_File_Object | Fill a file object with specified dir |

| | | |
|---|---|---|
| | | entry data |
| | FindEmptyEntries | Find an empty dir entry |
| | flushData | Flush unwritten data to a file |
| | FormatDirName | Format a dir name into dir entry format |
| | FormatFileName | Format a file name into dir entry format |
| | FSputc | FSfprintf helper function to write a char |
| | FSvfprintf | Helper function for FSfprintf |
| | GetFullClusterNumber | Gets the cluster number from a directory entry |
| | GetPreviousEntry | Get the file entry info for the parent dir of the specified dir |
| | IncrementTimeStamp | Automatically set the timestamp to "don't care" data |
| | LoadBootSector | Load the boot sector and extract the necessary information |
| | LoadDirAttrib | Load file information from a directory entry and cache the entry |
| | LoadMBR | Loads the MBR and extracts necessary information |
| | mkdirhelper | Helper function for FSmkdir |
| | | |

| | | |
|---|---|---|
| | PopulateEntries | Populate a dir entry with data |
| | ReadFAT | Read the next entry from the FAT |
| | rmdirhelper | Helper function for FSrmdir |
| | SRAMInitHeap | This function initializes the dynamic heap. It inserts segment headers to maximize segment space. |
| | str_put_n_chars | FSfprintf helper function to write a char multiple times |
| | ValidateChars | Validate the characters in a given file name |
| | Write_File_Entry | Write dir entry info into a specified entry |
| | writeDotEntries | Create dot and dotdot entries in a non-root directory |

## Macros

| | Name | Description |
|---|---|---|
| | _FLAG_MINUS | FSfprintf minus flag indicator |
| | _FLAG_OCTO | FSfprintf octothorpe (hash mark) flag indicator |
| | _FLAG_PLUS | FSfprintf plus flag indicator |
| | _FLAG_SIGNED | FSfprintf signed flag indicator |
| | _FLAG_SPACE | FSfprintf space flag indicator |

| | | |
|---|---|---|
| | _FLAG_ZERO | FSfprintf zero flag indicator |
| | _FMT_BYTE | FSfprintf 8-bit argument size flag |
| | _FMT_LONG | FSfprintf 32-bit argument size flag |
| | _FMT_LONGLONG | FSfprintf 64-bit argument size flag |
| | _FMT_SHRTLONG | FSfprintf 24-bit argument size flag |
| | _FMT_UNSPECIFIED | FSfprintf unspecified argument size flag |
| | _MAX_HEAP_SIZE | A macro used to determine the heap initialization size. |
| | _MAX_SEGMENT_SIZE | A macro used to determine the maximum size of a dynamic memory segment. |
| | DIRECTORY | Value indicating that the CreateFileEntry function will be creating a directory |
| | DIRENTRIES_PER_SECTOR | The number of directory entries in a sector |
| | NEAR | A macro used to specify the near-model action |

## Structures

| | Name | Description |
|---|---|---|

| | | |
|---|---|---|
| | _DIRENTRY | Directory entry structure |

## Types

| | Name | Description |
|---|---|---|
| | DIRENTRY | A pointer to a directory entry structure |
| | FILEOBJ | Pointer to an FSFILE object |

## Variables

| | Name | Description |
|---|---|---|
| | _uDynamicHeap | The PIC18 dynamic memory heap |
| | cwd | Global current working directory |
| | cwdptr | Pointer to the current working directory |
| | defaultArray | This string is used by FSgetcwd to return the cwd name if the path passed into the function is NULL |
| | defaultString | This string is used by dir functions to hold dir names temporarily |
| | dirCleared | Global variable used by the "recursive" FSrmdir function to indicate that all subdirectories and files have been deleted from the target directory. |
| | FatRootDirClusterValue | Global variable containing the cluster number of the root dir (0 for |

| | | |
|---|---|---|
| | | FAT12/16) |
| | FSerrno | Global error variable. Set to one of many error codes after each function call. |
| | gBufferOwner | Global variable indicating which file is using the data buffer |
| | gBufferZeroed | Global variable indicating that the data buffer contains all zeros |
| | gDataBuffer | The global data sector buffer |
| | gDiskData | Global structure containing device information. |
| | gFATBuffer | The global FAT sector buffer |
| | gFileArray | Array that contains file information (static allocation) |
| | gFileSlotOpen | Array that indicates which elements of gFileArray are available for use |
| | gFileTemp | Global variable used for file operations. |
| | gLastDataSectorRead | Global variable indicating which data sector was read last |
| | gLastFATSectorRead | Global variable indicating which FAT sector was read last |
| | gNeedDataWrite | Global variable indicating that there is information that needs to be written to the data section |
| | | Global variable indicating that there |

| | | |
|---|---|---|
| | gNeedFATWrite | is information that needs to be written to the FAT |
| | gTimeAccDate | Global time variable (for timestamps) used to indicate last access date |
| | gTimeCrtDate | Global time variable (for timestamps) used to indicate create date |
| | gTimeCrtMS | Global time variable (for timestamps) used to indicate create time (milliseconds) |
| | gTimeCrtTime | Global time variable (for timestamps) used to indicate create time |
| | gTimeWrtDate | Global time variable (for timestamps) used to indicate last update date |
| | gTimeWrtTime | Global time variable (for timestamps) used to indicate last update time |
| | nextClusterIsLast | Global variable indicating that the entries in a directory align with a cluster boundary |
| | recache | Global variable used by the "recursive" FSrmdir function to indicate that additional cache reads are needed. |
| | s_digits | FSfprintf table of conversion digits |
| | tempArray | This array is used to prevent a stack frame error |
| | TempClusterCalc | Global variable used to store the calculated value of the cluster of a |

| | | |
|---|---|---|
| | | specified sector. |
| | tempCWDobj | Global variable used to preserve the current working directory information. |

# _SRAMmerge Function

C

```
NEAR unsigned char _SRAMmerge(
    SALLOC * NEAR pSegA
);
```

## Description

This function tries to merge adjacent segments that have not been allocated. The largest possible segment is merged if possible.

## Parameters

| Parameters | Description |
|---|---|
| SALLOC * NEAR pSegA | pointer to the first segment. |

## Returns

usnigned char - returns the length of the merged segment or zero if failed to merge.

# Cache_File_Entry Function

C

```
DIRENTRY Cache_File_Entry(
    FILEOBJ fo,
    WORD * curEntry,
    BYTE ForceRead
);
```

## Description

Load the sector containing the file entry pointed to by 'curEntry' from the directory pointed to by the variables in 'fo.'

## Preconditions

This function should not be called by the user.

## Parameters

| Parameters | Description |
|---|---|
| fo | File information |
| curEntry | Offset of the directory entry to load. |
| ForceRead | Forces loading of a new sector of the directory. |

## Returns

DIRENTRY - Pointer to the directory entry that was loaded.

## Side Effects

Any unwritten data in the data buffer will be written to the device.

## Remarks

Any modification of this function is extremely likely to break something.

# CacheTime Function

C

```
void CacheTime();
```

## Description

This function will automatically load information from an RTCC module and use it to update the global timing variables. These can then be used to update file timestamps.

## Preconditions

RTCC module enabled. Should not be called by the user.

## Side Effects

Modifies global timing variables

## Remarks

None.

APIs > File Manipulation Layer (FSIO) > Internal Members > CacheTime Function

# chdirhelper Function

C

```c
int chdirhelper(
    BYTE mode,
    char * ramptr,
    const rom char * romptr
);
```

## Description

This helper function is used by the FSchdir function. If the path argument is specified in ROM for PIC18 this function will be able to parse it correctly. The function will loop through a switch statement to process the tokens in the path string. Dot or dotdot entries are handled in the first case statement. A backslash character is handled in the second case statement (note that this case statement will only be used if backslash is the first character in the path; backslash token delimiters will automatically be skipped after each token in the path is processed). The third case statement will handle actual directory name strings.

## Preconditions

None

## Parameters

| Parameters | Description |
|---|---|
| mode | Indicates which path pointer to use |
| ramptr | Pointer to the path specified in RAM |

| | |
|---|---|
| romptr | Pointer to the path specified in ROM |

## Return Values

| Return Values | Description |
|---|---|
| 0 | Directory was changed successfully. |
| EOF | Directory could not be changed. |

## Side Effects

The current working directory will be changed. The FSerrno variable will be changed. Any unwritten data in the data buffer will be written to the device.

## Remarks

None.

# Cluster2Sector Function

C

```
DWORD Cluster2Sector(
    DISK * disk,
    DWORD cluster
);
```

## Description

The Cluster2Sector function will calculate the sector number that corresponds to the first sector of the cluster whose value was passed into the function.

## Preconditions

This function should not be called by the user.

## Parameters

| Parameters | Description |
|---|---|
| disk | Disk structure |
| cluster | Cluster to be converted |

## Returns

sector - Sector that corresponds to given cluster

## Side Effects

None

## Remarks

None.

# CreateDIR Function

C

```c
int CreateDIR(
    char * path
);
```

## Description

The CreateDIR function is a helper function for the mkdirhelper function. The CreateDIR function will create a new file entry for a directory and assign a cluster to it. It will erase the cluster and write a dot and dotdot entry to it.

## Preconditions

This function should not be called by the user.

## Parameters

| Parameters | Description |
|---|---|
| path | The name of the dir to create |

## Return Values

| Return Values | Description |
|---|---|
| TRUE | Directory was created successfully |
| FALSE | Directory could not be created. |

## Side Effects

Any unwritten data in the data buffer or the FAT buffer will be written to the device.

## Remarks

None.

# CreateFileEntry Function

C

```
CETYPE CreateFileEntry(
    FILEOBJ fo,
    WORD * fHandle,
    BYTE mode
);
```

## Description

With the data passed within fo, create a new file entry in the current directory. This function will first search for empty file entries. Once an empty entry is found, the entry will be populated with data for a file or directory entry. Finally, the first cluster of the new file will be located and allocated, and its value will be written into the file entry.

## Preconditions

Should not be called by the user.

## Parameters

| Parameters | Description |
|---|---|
| fo | Pointer to file structure |
| fHandle | Location to create file |

## Return Values

| Return Values | Description |
|---|---|
| | |

| | |
|---|---|
| CE_GOOD | File Creation successful |
| CE_DIR_FULL | All root directory entries are taken |
| CE_WRITE_ERROR | The head cluster of the file could not be created. |

## Side Effects

Modifies the FSerrno variable.

## Remarks

None

# CreateFirstCluster Function

C

```
CETYPE CreateFirstCluster(
    FILEOBJ fo
);
```

## Description

This function will find an unused cluster, link it to a file's directory entry, and write the entry back to the device.

## Preconditions

This function should not be called by the user.

## Parameters

| Parameters | Description |
|---|---|
| fo | The file that contains the first cluster |

## Return Values

| Return Values | Description |
|---|---|
| CE_GOOD | First cluster created successfully |
| CE_WRITE_ERROR | Cluster creation failed |

## Side Effects

None

## Remarks

None.

# DISKmount Function

C

```
BYTE DISKmount(
    DISK * dsk
);
```

## Description

This function will use the function pointed to by the MDD_MediaInitialize function pointer to initialize the device (if any initialization is required). It then attempts to load the master boot record with the LoadMBR function and the boot sector with the LoadBootSector function. These two functions will be used to initialize a global DISK structure that will be used when accessing file information in the future.

## Preconditions

This function should not be called by the user.

## Parameters

| Parameters | Description |
|---|---|
| dsk | The disk structure to be initialized. |

## Return Values

| Return Values | Description |
|---|---|
| CE_GOOD | Disk mounted |
| CE_INIT_ERROR | Initialization error has occured |

## Side Effects

None

## Remarks

None

# EraseCluster Function

C

```
BYTE EraseCluster(
    DISK * disk,
    DWORD cluster
);
```

## Description

The EraseCluster function will write a 0 value into every byte of the specified cluster.

## Preconditions

This function should not be called by the user.

## Parameters

| Parameters | Description |
|---|---|
| dsk | Disk structure |
| cluster | Cluster to be erased |

## Return Values

| Return Values | Description |
|---|---|
| CE_GOOD | File closed successfully |
| CE_WRITE_ERROR | Could not write to the sector |

## Side Effects

None

## Remarks

None.

# FAT_erase_cluster_chain Function

C

```
BYTE FAT_erase_cluster_chain(
    DWORD cluster,
    DISK * dsk
);
```

## Description

This function will parse through a cluster chain starting with the cluster pointed to by 'cluster' and mark all of the FAT entries as empty until the end of the chain has been reached or an error occurs.

## Preconditions

This function should not be called by the user.

## Parameters

| Parameters | Description |
|---|---|
| cluster | The cluster number |
| dsk | The disk structure |

## Return Values

| Return Values | Description |
|---|---|
| TRUE | Operation successful |
| FALSE | Operation failed |

## Side Effects

None

## Remarks

None

# FATfindEmptyCluster Function

C

```
DWORD FATfindEmptyCluster(
    FILEOBJ fo
);
```

## Description

This function will search through the FAT to find the next available cluster on the device.

## Preconditions

This function should not be called by the user.

## Parameters

| Parameters | Description |
|---|---|
| fo | Pointer to file structure |

## Return Values

| Return Values | Description |
|---|---|
| DWORD | Address of empty cluster |
| 0 | Could not find empty cluster |

## Side Effects

None

## Remarks

Should not be called by user

# FILEallocate_new_cluster Function

C

```
BYTE FILEallocate_new_cluster(
    FILEOBJ fo,
    BYTE mode
);
```

## Description

This function will find an empty cluster on the device using the FATfindEmptyCluster function. It will then mark it as the last cluster in the file in the FAT chain, and link the current last cluster of the passed file to the new cluster. If the new cluster is a directory cluster, it will be erased (so there are no extraneous directory entries). If it's allocated to a non-directory file, it doesn't need to be erased; extraneous data in the cluster will be unviewable because of the file size parameter.

## Preconditions

Should not be called by the user.

## Parameters

| Parameters | Description |
|---|---|
| fo | Pointer to file structure |
| mode | • 0 - Allocate a cluster to a file<br>• 1 - Allocate a cluster to a directory |

## Return Values

| Return Values | Description |
| --- | --- |
| CE_GOOD | Cluster allocated |
| CE_DISK_FULL | No clusters available |

## Side Effects

None

## Remarks

None.

# FILECreateHeadCluster Function

C

```
CETYPE FILECreateHeadCluster(
    FILEOBJ fo,
    DWORD * cluster
);
```

## Description

The FILECreateHeadCluster function will create the first cluster of a file. First, it will find an empty cluster with the FATfindEmptyCluster function and mark it as the last cluster in the file. It will then erase the cluster using the EraseCluster function.

## Preconditions

This function should not be called by the user.

## Parameters

| Parameters | Description |
|---|---|
| fo | Pointer to file structure |
| cluster | Cluster location |

## Return Values

| Return Values | Description |
|---|---|
| CE_GOOD | File closed successfully |
| | |

| | |
|---|---|
| CE_WRITE_ERROR | Could not write to the sector |
| CE_DISK_FULL | All clusters in partition are taken |

## Side Effects

None

## Remarks

None.

# FILEerase Function

C

```
CETYPE FILEerase(
    FILEOBJ fo,
    WORD * fHandle,
    BYTE EraseClusters
);
```

## Description

This function will cache the sector of directory entries in the directory pointed to by the dirclus value in the FSFILE object 'fo' that contains the entry that corresponds to the fHandle offset. It will then mark that entry as deleted. If the EraseClusters argument is TRUE, the chain of clusters for that file will be marked as unused in the FAT by the FAT_erase_cluster_chain function.

## Preconditions

This function should not be called by the user.

## Parameters

| Parameters | Description |
|---|---|
| fo | Pointer to file structure |
| fHandle | Location of file information |
| EraseClusters | Remove cluster allocation from FAT? |

## Return Values

| Return Values | Description |
| --- | --- |
| CE_GOOD | File erased successfully |
| CE_FILE_NOT_FOUND | Could not find the file on the card |
| CE_ERASE_FAIL | Internal Card erase failed |

## Side Effects

None

## Remarks

None.

# FILEfind Function

C

```
CETYPE FILEfind(
    FILEOBJ foDest,
    FILEOBJ foCompareTo,
    BYTE cmd,
    BYTE mode
);
```

## Description

The FILEfind function will sequentially cache directory entries within the current working directory into the foDest FSFILE object. If the cmd parameter is specified as LOOK_FOR_EMPTY_ENTRY the search will continue until an empty directory entry is found. If the cmd parameter is specified as LOOK_FOR_MATCHING_ENTRY these entries will be compared to the foCompareTo object until a match is found or there are no more entries in the current working directory. If the mode is specified a '0' the attributes of the FSFILE entries are irrelevant. If the mode is specified as '1' the attributes of the foDest entry must match the attributes specified in the foCompareTo file and partial string search characters may bypass portions of the comparison.

## Preconditions

This function should not be called by the user.

## Parameters

| Parameters | Description |
|---|---|
| | |

| | |
|---|---|
| foDest | FSFILE object containing information of the file found |
| foCompareTo | FSFILE object containing the name/attr of the file to be found |
| cmd | • LOOK_FOR_EMPTY_ENTRY: Search for empty entry.<br>• LOOK_FOR_MATCHING_ENTRY: Search for matching entry. |
| mode | • 0: Match file exactly with default attributes.<br>• 1: Match file to user-specified attributes. |

## Return Values

| Return Values | Description |
|---|---|
| CE_GOOD | File found. |
| CE_FILE_NOT_FOUND | File not found. |

## Side Effects

None.

## Remarks

None

# FILEget_next_cluster Function

C

```
BYTE FILEget_next_cluster(
    FILEOBJ fo,
    DWORD n
);
```

## Description

This function will load 'n' proximate clusters for a file from the FAT on the device. It will stop checking for clusters if the ReadFAT function returns an error, if it reaches the last cluster in a file, or if the device tries to read beyond the last cluster used by the device.

## Preconditions

This function should not be called by the user.

## Parameters

| Parameters | Description |
|---|---|
| fo | The file to get the next cluster of |
| n | Number of links in the FAT cluster chain to jump through |

## Return Values

| Return Values | Description |
|---|---|
| CE_GOOD | Operation successful |

| | |
|---|---|
| CE_BAD_SECTOR_READ | A bad read occured of a sector |
| CE_INVALID_CLUSTER | Invalid cluster value > maxcls |
| CE_FAT_EOF | Fat attempt to read beyond EOF |

## Side Effects

None

## Remarks

None

# FileObjectCopy Function

C

```c
void FileObjectCopy(
    FILEOBJ foDest,
    FILEOBJ foSource
);
```

## Description

The FileObjectCopy function will make an exacy copy of a specified FSFILE object.

## Preconditions

This function should not be called by the user.

## Parameters

| Parameters | Description |
|---|---|
| foDest | The destination |
| foSource | the source |

## Returns

None

## Side Effects

None

## Remarks

None

# FILEopen Function

C

```c
CETYPE FILEopen(
    FILEOBJ fo,
    WORD * fHandle,
    char type
);
```

## Description

This function will cache a directory entry in the directory specified by the dirclus parameter of hte FSFILE object 'fo.' The offset of the entry in the directory is specified by fHandle. Once the directory entry has been loaded, the first sector of the file can be loaded using the cluster value specified in the directory entry. The type argument will specify the mode the files will be opened in. This will allow this function to set the correct read/write flags for the file.

## Preconditions

This function should not be called by the user.

## Parameters

| Parameters | Description |
|---|---|
| fo | File to be opened |
| fHandle | Location of file |
| | • WRITE - Create a new file or replace an existing file |

| | |
|---|---|
| type | • READ - Read data from an existing file<br>• APPEND - Append data to an existing file |

## Return Values

| Return Values | Description |
|---|---|
| CE_GOOD | FILEopen successful |
| CE_NOT_INIT | Device is not yet initialized |
| CE_FILE_NOT_FOUND | Could not find the file on the device |
| CE_BAD_SECTOR_READ | A bad read of a sector occured |

## Side Effects

None

## Remarks

If the mode the file is being opened in is a plus mode (e.g. READ+) the flags will be modified further in the FSfopen function.

# Fill_File_Object Function

C

```
BYTE Fill_File_Object(
    FILEOBJ fo,
    WORD * fHandle
);
```

## Description

This function will cache the sector of directory entries in the directory pointed to by the dirclus value in the FSFILE object 'fo' that contains the entry that corresponds to the fHandle offset. It will then copy the file information for that entry into the 'fo' FSFILE object.

## Preconditions

This function should not be called by the user.

## Parameters

| Parameters | Description |
|---|---|
| fo | Pointer to file structure |
| fHandle | Passed member's location |

## Return Values

| Return Values | Description |
|---|---|
| FOUND | Operation successful |
| | |

| | |
|---|---|
| NOT_FOUND | Operation failed |

## Side Effects

None

## Remarks

None.

# FindEmptyEntries Function

C

```
BYTE FindEmptyEntries(
    FILEOBJ fo,
    WORD * fHandle
);
```

## Description

This function will cache directory entries, starting with the one pointed to by the fHandle argument. It will then search through the entries until an unused one is found. If the end of the cluster chain for the directory is reached, a new cluster will be allocated to the directory (unless it's a FAT12 or FAT16 root) and the first entry of the new cluster will be used.

## Preconditions

This function should not be called by the user.

## Parameters

| Parameters | Description |
|---|---|
| fo | Pointer to file structure |
| fHandle | Start of entries |

## Return Values

| Return Values | Description |
|---|---|
| TRUE | One found |

| | |
|---|---|
| FALSE | None found |

## Side Effects

None

## Remarks

None.

# flushData Function

**C**

```
BYTE flushData();
```

## Description

The flushData function is called when it is necessary to read new data into the global data buffer and the gNeedDataWrite variable indicates that there is data in the buffer that hasn't been written to the device. The flushData function will write the data from the buffer into the current cluster of the FSFILE object that is stored in the gBufferOwner global variable.

## Preconditions

File opened in a write mode, data needs to be written

## Return Values

| Return Values | Description |
|---|---|
| CE_GOOD | Data was updated successfully |
| CE_WRITE_ERROR | Data could not be updated |

## Side Effects

None

## Remarks

None

# FormatDirName Function

C

```c
BYTE FormatDirName(
    char * string,
    BYTE mode
);
```

## Description

Format an 8.3 filename into directory structure format. If the name is less than 8 chars, then it will be padded with spaces. If the extension name is fewer than 3 chars, then it will also be oadded with spaces. The ValidateChars function is used to ensure the characters in the specified directory name are valid in this filesystem.

## Preconditions

This function should not be called by the user.

## Parameters

| Parameters | Description |
|---|---|
| string | The name to be formatted |
| mode | • TRUE - Partial string search characters are allowed<br>• FALSE - Partial string search characters are forbidden |

## Return Values

| Return Values | Description |
| --- | --- |
| TRUE | The name was formatted correctly |
| FALSE | The name contained invalid characters |

## Side Effects

None

## Remarks

None.

# FormatFileName Function

C

```c
BYTE FormatFileName(
    const char* fileName,
    char* fN2,
    BYTE mode
);
```

## Description

Format an 8.3 filename into FSFILE structure format. If filename is less than 8 chars, then it will be padded with spaces. If the extension name is fewer than 3 chars, then it will also be oadded with spaces. The ValidateChars function is used to ensure the characters in the specified filename are valid in this filesystem.

## Preconditions

This function should not be called by the user.

## Parameters

| Parameters | Description |
|---|---|
| fileName | The name to be formatted |
| fN2 | The location the formatted name will be stored |
| mode | Non-zero if parital string search chars are allowed |

## Return Values

| Return Values | Description |
| --- | --- |
| TRUE | Name formatted successfully |
| FALSE | File name could not be formatted |

## Side Effects

None

## Remarks

None.

# FSputc Function

C

```c
int FSputc(
    char c,
    FSFILE * file
);
```

## Description

This is a helper function for FSfprintf. It will write one character to a file.

## Preconditions

This function should not be called by the user.

## Parameters

| Parameters | Description |
|---|---|
| c | The character to write to the file. |
| file | The file to write to. |

## Return Values

| Return Values | Description |
|---|---|
| 0 | The character was written successfully |
| EOF | The character was not written to the file. |

## Side Effects

None

## Remarks

None

# FSvfprintf Function

C

```c
int FSvfprintf(
    auto FSFILE * handle,
    auto const rom char * formatString,
    auto va_list ap
);
```

## Description

This helper function will access the elements passed to FSfprintf

## Preconditions

This function should not be called by the user.

## Parameters

| Parameters | Description |
|---|---|
| handle | A pointer to the file to write to. |
| formatString | A string of characters and format specifiers to write to the file |
| ap | A structure pointing to the arguments on the stack |

## Returns

The number of characters written to the file

## Side Effects

The FSerrno variable will be changed.

## Remarks

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

# GetFullClusterNumber Function

C

```c
DWORD GetFullClusterNumber(
    DIRENTRY entry
);
```

## Description

This function will load both the high and low 16-bit first cluster values of a file from a directory entry and copy them into a 32-bit cluster number variable, which will be returned.

## Preconditions

This function should not be called by the user.

## Parameters

| Parameters | Description |
|---|---|
| entry | The cached directory entry to get the cluster number from |

## Returns

The cluster value from the passed directory entry

## Side Effects

None.

## Remarks

None

# GetPreviousEntry Function

C

```
BYTE GetPreviousEntry(
    FSFILE * fo
);
```

## Description

The GetPreviousEntry function is used by the FSgetcwd function to load the previous (parent) directory. This function will load the parent directory and then search through the file entries in that directory for one that matches the cluster number of the original directory. When the matching entry is found, the name of the original directory is copied into the 'fo' FSFILE object.

## Preconditions

Should not be called by the user.

## Parameters

| Parameters | Description |
|---|---|
| fo | The file to get the previous entry of |

## Return Values

| Return Values | Description |
|---|---|
| 0 | The previous entry was successfully retrieved |
| -1 | The previous entry could not be retrieved |

## Side Effects

None

## Remarks

None.

# IncrementTimeStamp Function

```
C
void IncrementTimeStamp(
    DIRENTRY dir
);
```

## Description

This function will increment the timestamp variable in the 'dir' directory entry. This is used for the don't-care timing method.

## Preconditions

Should not be called by the user.

## Parameters

| Parameters | Description |
|---|---|
| dir | Pointer to directory structure |

## Side Effects

None

## Remarks

None

APIs > File Manipulation Layer (FSIO) > Internal Members > IncrementTimeStamp Function

# LoadBootSector Function

**C**

```
BYTE LoadBootSector(
    DISK * dsk
);
```

## Description

LoadBootSector will use the function pointed to by the MDD_SectorWrite function pointer to load the boot sector, whose location was obtained by a previous call of LoadMBR. If the boot sector is loaded successfully, partition information will be calcualted from it and copied into the DISK structure pointed to by 'dsk.'

## Preconditions

This function should not be called by the user.

## Parameters

| Parameters | Description |
|---|---|
| dsk | The disk containing the boot sector |

## Return Values

| Return Values | Description |
|---|---|
| CE_GOOD | Boot sector loaded |
| CE_BAD_SECTOR_READ | A bad read occured of a sector |
| | |

| | |
|---|---|
| CE_NOT_FORMATTED | The disk is of an unsupported format |
| CE_CARDFAT32 | FAT 32 device not supported |

## Side Effects

None

## Remarks

None

# LoadDirAttrib Function

C

```
DIRENTRY LoadDirAttrib(
    FILEOBJ fo,
    WORD * fHandle
);
```

## Description

This function will cache the sector of directory entries in the directory pointed to by the dirclus value in the FSFILE object 'fo' that contains the entry that corresponds to the fHandle offset. It will then return a pointer to the directory entry in the global data buffer.

## Preconditions

This function should not be called by the user.

## Parameters

| Parameters | Description |
|---|---|
| fo | Pointer to file structure |
| fHandle | Information location |

## Return Values

| Return Values | Description |
|---|---|
| DIRENTRY | Pointer to the directory entry |
| | |

| | |
|---|---|
| NULL | Directory entry could not be loaded |

## Side Effects

None

## Remarks

None.

# LoadMBR Function

C

```
BYTE LoadMBR(
    DISK * dsk
);
```

## Description

The LoadMBR function will use the function pointed to by the MDD_SectorRead function pointer to read the 0 sector from the device. If a valid boot signature is obtained, this function will compare fields in that cached sector to the values that would be present if that sector was a boot sector. If all of those values match, it will be assumed that the device does not have a master boot record and the 0 sector is actually the boot sector. Otherwise, data about the partition and the actual location of the boot sector will be loaded from the MBR into the DISK structure pointed to by 'dsk.'

## Preconditions

This function should not be called by the user.

## Parameters

| Parameters | Description |
|---|---|
| dsk | The disk containing the master boot record to be loaded |

## Return Values

| | |
|---|---|

| Return Values | Description |
| --- | --- |
| CE_GOOD | MBR loaded successfully |
| CE_BAD_SECTOR_READ | A bad read occured of a sector |
| CE_BAD_PARTITION | The boot record is bad |

## Side Effects

None

## Remarks

None

# mkdirhelper Function

C

```c
int mkdirhelper(
    BYTE mode,
    char * ramptr,
    const rom char * romptr
);
```

## Description

This helper function is used by the FSchdir function. If the path argument is specified in ROM for PIC18 this function will be able to parse it correctly. This function will first scan through the path to ensure that any DIR names don't exceed 11 characters. It will then backup the current working directory and begin changing directories through the path until it reaches a directory than can't be changed to. It will then create the specified directory and change directories to the new directory. The function will continue creating and changing to directories until the end of the path is reached. The function will then restore the original current working directory.

## Preconditions

None

## Parameters

| Parameters | Description |
|---|---|
| mode | Indicates which path pointer to use |
| ramptr | Pointer to the path specified in RAM |

| | |
|---|---|
| romptr | Pointer to the path specified in ROM |

## Return Values

| Return Values | Description |
|---|---|
| 0 | Directory was created |
| -1 | Directory could not be created |

## Side Effects

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

## Remarks

None

# PopulateEntries Function

C

```c
BYTE PopulateEntries(
    FILEOBJ fo,
    char * name,
    WORD * fHandle,
    BYTE mode
);
```

## Description

This function will write data into a new file entry. It will also load timestamp data (based on the method selected by the user) and update the timestamp variables.

## Preconditions

Should not be called by the user.

## Parameters

| Parameters | Description |
|---|---|
| fo | Pointer to file structure |
| name | Name of the file |
| fHandle | Location of the file |

## Return Values

| Return Values | Description |
|---|---|
| CE_GOOD | Population successful |

## Side Effects

None

## Remarks

None.

# ReadFAT Function

C

```c
DWORD ReadFAT(
    DISK * dsk,
    DWORD ccls
);
```

## Description

The ReadFAT function will read the FAT and determine the next cluster value after the cluster specified by 'ccls.' Note that the FAT sector that is read is stored in the global FAT cache buffer.

## Preconditions

This function should not be called by the user.

## Parameters

| Parameters | Description |
|---|---|
| dsk | The disk structure |
| ccls | The current cluster |

## Returns

DWORD - The next cluster in a file chain

## Side Effects

None

## Remarks

None.

# rmdirhelper Function

```c
int rmdirhelper(
    BYTE mode,
    char * ramptr,
    const rom char * romptr,
    unsigned char rmsubdirs
);
```

## Description

This helper function is used by the FSmkdir function. If the path argument is specified in ROM for PIC18 this function will be able to parse it correctly. This function will first change to the specified directory. If the rmsubdirs argument is FALSE the function will search through the directory to ensure that it is empty and then remove it. If the rmsubdirs argument is TRUE the function will also search through the directory for subdirectories or files. When the function finds a file, the file will be erased. When the function finds a subdirectory, it will switch to the subdirectory and begin removing all of the files in that subdirectory. Once the subdirectory is empty, the function will switch back to the original directory. return to the original position in that directory, and continue removing files. Once the specified directory is empty, the function will change to the parent directory, search through it for the directory to remove, and then erase that directory.

## Preconditions

This function should not be called by the user.

## Parameters

| Parameters | Description |
| --- | --- |
| path | The path of the dir to delete |
| rmsubdirs | • TRUE - Remove all sub-directories and files in the directory<br>• FALSE - Non-empty directories can not be removed |

## Return Values

| Return Values | Description |
| --- | --- |
| 0 | The specified directory was successfully removed. |
| EOF | The specified directory could not be removed. |

## Side Effects

The FSerrno variable will be changed.

## Remarks

None.

# SRAMInitHeap Function

C

```
void SRAMInitHeap();
```

## Description

This function initializes the dynamic heap. It inserts segment headers to maximize segment space.

## Returns

void

## Remarks

This function must be called at least one time. And it could be called more times to reset the heap.

APIs > File Manipulation Layer (FSIO) > Internal Members > SRAMInitHeap Function

# str_put_n_chars Function

C

```c
unsigned char str_put_n_chars(
    FSFILE * handle,
    unsigned char n,
    char c
);
```

## Description

This funciton is used by the FSfprintf function to write multiple instances of a single character to a file (for example, when padding a format specifier with leading spacez or zeros).

## Preconditions

This function should not be called by the user.

## Parameters

| Parameters | Description |
|---|---|
| handle | The file to write to. |
| n | The number of times to write that character to a file. |
| c | The character to write to the file. |

## Return Values

| Return Values | Description |
|---|---|
| 0 | The characters were written successfully |

| | |
|---|---|
| EOF | The characters were not written to the file. |

## Side Effects

None

## Remarks

None.

# ValidateChars Function

C

```c
BYTE ValidateChars(
    char * FileName,
    BYTE mode
);
```

## Description

The ValidateChars function will compare characters in a specified filename to determine if they're permissable in the FAT file system. Lower-case characters will be converted to upper-case. If the mode argument is specifed to be 'TRUE,' partial string search characters are allowed.

## Preconditions

This function should not be called by the user.

## Parameters

| Parameters | Description |
|---|---|
| fileName | The name to be validated |
| mode | Determines if partial string search is allowed |

## Return Values

| Return Values | Description |
|---|---|
| TRUE | Name was validated |

| | |
|---|---|
| FALSE | File name was not valid |

## Side Effects

None

## Remarks

None.

# Write_File_Entry Function

C

```
BYTE Write_File_Entry(
    FILEOBJ fo,
    WORD * curEntry
);
```

## Description

This function will calculate the sector of the directory (whose base sector is pointed to by the dirccls value in the FSFILE object 'fo') that contains a directory entry whose offset is indicated by the curEntry parameter. It will then write the data in the global data buffer (which should already contain the entries for that sector) to the device.

## Preconditions

This function should not be called by the user.

## Parameters

| Parameters | Description |
|---|---|
| fo | File structure |
| curEntry | Write destination |

## Return Values

| Return Values | Description |
|---|---|
| TRUE | Operation successful |

| | |
|---|---|
| FALSE | Operation failed |

## Side Effects

None

## Remarks

None

# writeDotEntries Function

C

```
BYTE writeDotEntries(
    DISK * dsk,
    DWORD dotAddress,
    DWORD dotdotAddress
);
```

## Description

The writeDotEntries function will create and write dot and dotdot entries to a newly created directory.

## Preconditions

This function should not be called by the user.

## Parameters

| Parameters | Description |
|---|---|
| disk | The global disk structure |
| dotAddress | The cluster the current dir is in |
| dotdotAddress | The cluster the previous directory was in |

## Return Values

| Return Values | Description |
|---|---|
| TRUE | The dot and dotdot entries were created |
| | The dot and dotdot entries could not be |

| | |
|---|---|
| FALSE | created in the new directory |

## Side Effects

None

## Remarks

None.

# _FLAG_MINUS Macro

C

```c
#define _FLAG_MINUS 0x1                // FSfprintf minu
```

## Description

FSfprintf minus flag indicator

APIs > File Manipulation Layer (FSIO) > Internal Members > _FLAG_MINUS Macro

# _FLAG_OCTO Macro

C

```c
#define _FLAG_OCTO 0x8                    // FSfprintf octo
```

## Description

FSfprintf octothorpe (hash mark) flag indicator

APIs > File Manipulation Layer (FSIO) > Internal Members > _FLAG_OCTO Macro

# _FLAG_PLUS Macro

C

```c
#define _FLAG_PLUS 0x2                    // FSfprintf plus
```

## Description

FSfprintf plus flag indicator

APIs > File Manipulation Layer (FSIO) > Internal Members > _FLAG_PLUS Macro

# _FLAG_SIGNED Macro

C

```c
#define _FLAG_SIGNED 0x80          // FSfprintf sig
```

## Description

FSfprintf signed flag indicator

APIs > File Manipulation Layer (FSIO) > Internal Members > _FLAG_SIGNED Macro

# _FLAG_SPACE Macro

C

```c
#define _FLAG_SPACE 0x4                // FSfprintf spa
```

## Description

FSfprintf space flag indicator

APIs > File Manipulation Layer (FSIO) > Internal Members > _FLAG_SPACE Macro

# _FLAG_ZERO Macro

C

```c
#define _FLAG_ZERO 0x10            // FSfprintf zero
```

## Description

FSfprintf zero flag indicator

APIs > File Manipulation Layer (FSIO) > Internal Members > _FLAG_ZERO Macro

# _FMT_BYTE Macro

C

```c
#define _FMT_BYTE 3                    // FSfprintf 8-bit a
```

## Description

FSfprintf 8-bit argument size flag

APIs > File Manipulation Layer (FSIO) > Internal Members > _FMT_BYTE Macro

# _FMT_LONG Macro

C

```c
#define _FMT_LONG 2                    // FSfprintf 32-bit a
```

## Description

FSfprintf 32-bit argument size flag

APIs > File Manipulation Layer (FSIO) > Internal Members > _FMT_LONG Macro

# _FMT_LONGLONG Macro

C

```c
#define _FMT_LONGLONG 1          // FSfprintf 64-bit a
```

## Description

FSfprintf 64-bit argument size flag

APIs > File Manipulation Layer (FSIO) > Internal Members > _FMT_LONGLONG Macro

# _FMT_SHRTLONG Macro

C

```c
#define _FMT_SHRTLONG 2          // FSfprintf 24-bit a
```

## Description

FSfprintf 24-bit argument size flag

APIs > File Manipulation Layer (FSIO) > Internal Members > _FMT_SHRTLONG Macro

# _FMT_UNSPECIFIED Macro

C

```c
#define _FMT_UNSPECIFIED 0      // FSfprintf unspeci
```

## Description

FSfprintf unspecified argument size flag

APIs > File Manipulation Layer (FSIO) > Internal Members > _FMT_UNSPECIFIED Macro

# _MAX_HEAP_SIZE Macro

C

```c
#define _MAX_HEAP_SIZE MAX_HEAP_SIZE-1
```

## Description

A macro used to determine the heap initialization size.

APIs > File Manipulation Layer (FSIO) > Internal Members > _MAX_HEAP_SIZE Macro

# _MAX_SEGMENT_SIZE Macro

C

```c
#define _MAX_SEGMENT_SIZE 0x7F
```

## Description

A macro used to determine the maximum size of a dynamic memory segment.

APIs > File Manipulation Layer (FSIO) > Internal Members > _MAX_SEGMENT_SIZE Macro

# DIRECTORY Macro

C

```c
#define DIRECTORY 0x12          // Value indicating
```

## Description

Value indicating that the CreateFileEntry function will be creating a directory

APIs > File Manipulation Layer (FSIO) > Internal Members > DIRECTORY Macro

# DIRENTRIES_PER_SECTOR Macro

C

```
#define DIRENTRIES_PER_SECTOR (MEDIA_SECTOR_SIZE / 3
```

## Description

The number of directory entries in a sector

APIs > File Manipulation Layer (FSIO) > Internal Members > DIRENTRIES_PER_SECTOR Macro

# NEAR Macro

C

```
#define NEAR
```

## Description

Functions can be declared using the NEAR macro. If the NEAR_MODEL macro is uncommented, the NEAR macro will be ignored.

APIs > File Manipulation Layer (FSIO) > Internal Members > NEAR Macro

# _DIRENTRY Structure

C

```c
typedef struct {
  char DIR_Name[DIR_NAMESIZE];
  char DIR_Extension[DIR_EXTENSION];
  BYTE DIR_Attr;
  BYTE DIR_NTRes;
  BYTE DIR_CrtTimeTenth;
  WORD DIR_CrtTime;
  WORD DIR_CrtDate;
  WORD DIR_LstAccDate;
  WORD DIR_FstClusHI;
  WORD DIR_WrtTime;
  WORD DIR_WrtDate;
  WORD DIR_FstClusLO;
  DWORD DIR_FileSize;
} _DIRENTRY;
```

## Description

Directory entry structure

## Members

| Members | Description |
|---|---|
| char DIR_Name[DIR_NAMESIZE]; | File name |
| char DIR_Extension[DIR_EXTENSION]; | File extension |
| BYTE DIR_Attr; | File attributes |
| BYTE DIR_NTRes; | Reserved byte |

| | |
|---|---|
| BYTE DIR_CrtTimeTenth; | Create time (millisecond field) |
| WORD DIR_CrtTime; | Create time (second, minute, hour field) |
| WORD DIR_CrtDate; | Create date |
| WORD DIR_LstAccDate; | Last access date |
| WORD DIR_FstClusHI; | High word of the entry's first cluster number |
| WORD DIR_WrtTime; | Last update time |
| WORD DIR_WrtDate; | Last update date |
| WORD DIR_FstClusLO; | Low word of the entry's first cluster number |
| DWORD DIR_FileSize; | The 32-bit file size |

# DIRENTRY Type

C

```c
typedef _DIRENTRY * DIRENTRY;
```

## Description

A pointer to a directory entry structure

APIs > File Manipulation Layer (FSIO) > Internal Members > DIRENTRY Type

# FILEOBJ Type

C

```
typedef FSFILE * FILEOBJ;
```

## Description

Pointer to an FSFILE object

APIs > File Manipulation Layer (FSIO) > Internal Members > FILEOBJ Type

# _uDynamicHeap Variable

C

```c
unsigned char _uDynamicHeap[MAX_HEAP_SIZE];
```

## Description

The _uDynamicHeap array is used as a heap for PIC18 dynamic memory allocation.

APIs > File Manipulation Layer (FSIO) > Internal Members > _uDynamicHeap Variable

# cwd Variable

**C**

```
FSFILE cwd;
```

## Description

Global current working directory

APIs > File Manipulation Layer (FSIO) > Internal Members > cwd Variable

# cwdptr Variable

C

```
FSFILE * cwdptr = &cwd;
```

## Description

Pointer to the current working directory

APIs > File Manipulation Layer (FSIO) > Internal Members > cwdptr Variable

# defaultArray Variable

C

```
char defaultArray[10];
```

## Description

This string is used by FSgetcwd to return the cwd name if the path passed into the function is NULL

APIs > File Manipulation Layer (FSIO) > Internal Members > defaultArray Variable

# defaultString Variable

C

```c
char defaultString[13];
```

## Description

This string is used by dir functions to hold dir names temporarily

APIs > File Manipulation Layer (FSIO) > Internal Members > defaultString Variable

# dirCleared Variable

**C**

```
BYTE dirCleared;
```

## Description

Global variable used by the "recursive" FSrmdir function to indicate that all subdirectories and files have been deleted from the target directory.

APIs > File Manipulation Layer (FSIO) > Internal Members > dirCleared Variable

# FatRootDirClusterValue Variable

**C**

```
DWORD FatRootDirClusterValue;
```

## Description

Global variable containing the cluster number of the root dir (0 for FAT12/16)

APIs > File Manipulation Layer (FSIO) > Internal Members > FatRootDirClusterValue Variable

# FSerrno Variable

**C**

```
BYTE FSerrno;
```

## Description

Global error variable. Set to one of many error codes after each function call.

APIs > File Manipulation Layer (FSIO) > Internal Members > FSerrno Variable

# gBufferOwner Variable

C

```
FSFILE * gBufferOwner = NULL;
```

## Description

Global variable indicating which file is using the data buffer

APIs > File Manipulation Layer (FSIO) > Internal Members > gBufferOwner Variable

# gBufferZeroed Variable

C

```
BYTE gBufferZeroed = FALSE;
```

## Description

Global variable indicating that the data buffer contains all zeros

APIs > File Manipulation Layer (FSIO) > Internal Members > gBufferZeroed Variable

# gDataBuffer Variable

**C**

```
BYTE gDataBuffer[MEDIA_SECTOR_SIZE];
```

## Description

The global data sector buffer

APIs > File Manipulation Layer (FSIO) > Internal Members > gDataBuffer Variable

# gDiskData Variable

C

```
DISK gDiskData;
```

## Description

Global structure containing device information.

APIs > File Manipulation Layer (FSIO) > Internal Members > gDiskData Variable

# gFATBuffer Variable

**C**

```
BYTE gFATBuffer[MEDIA_SECTOR_SIZE];
```

## Description

The global FAT sector buffer

APIs > File Manipulation Layer (FSIO) > Internal Members > gFATBuffer Variable

# gFileArray Variable

C

```
FSFILE gFileArray[FS_MAX_FILES_OPEN];
```

## Description

Array that contains file information (static allocation)

APIs > File Manipulation Layer (FSIO) > Internal Members > gFileArray Variable

# gFileSlotOpen Variable

C

```
BYTE gFileSlotOpen[FS_MAX_FILES_OPEN];
```

## Description

Array that indicates which elements of gFileArray are available for use

APIs > File Manipulation Layer (FSIO) > Internal Members > gFileSlotOpen Variable

# gFileTemp Variable

C

```
FSFILE gFileTemp;
```

## Description

Global variable used for file operations.

APIs > File Manipulation Layer (FSIO) > Internal Members > gFileTemp Variable

# gLastDataSectorRead Variable

**C**

```
DWORD gLastDataSectorRead = 0xFFFFFFFF;
```

## Description

Global variable indicating which data sector was read last

APIs > File Manipulation Layer (FSIO) > Internal Members > gLastDataSectorRead Variable

# gLastFATSectorRead Variable

**C**

```
DWORD gLastFATSectorRead = 0xFFFF;
```

## Description

Global variable indicating which FAT sector was read last

APIs > File Manipulation Layer (FSIO) > Internal Members > gLastFATSectorRead Variable

# gNeedDataWrite Variable

**C**

```
BYTE gNeedDataWrite = FALSE;
```

## Description

Global variable indicating that there is information that needs to be written to the data section

APIs > File Manipulation Layer (FSIO) > Internal Members > gNeedDataWrite Variable

# gNeedFATWrite Variable

**C**

```
BYTE gNeedFATWrite = FALSE;
```

## Description

Global variable indicating that there is information that needs to be written to the FAT

APIs > File Manipulation Layer (FSIO) > Internal Members > gNeedFATWrite Variable

# gTimeAccDate Variable

**C**

```
WORD gTimeAccDate;
```

## Description

Global time variable (for timestamps) used to indicate last access date

APIs > File Manipulation Layer (FSIO) > Internal Members > gTimeAccDate Variable

# gTimeCrtDate Variable

**C**

```
WORD gTimeCrtDate;
```

## Description

Global time variable (for timestamps) used to indicate create date

APIs > File Manipulation Layer (FSIO) > Internal Members > gTimeCrtDate Variable

# gTimeCrtMS Variable

**C**

```
BYTE gTimeCrtMS;
```

## Description

Global time variable (for timestamps) used to indicate create time (milliseconds)

APIs > File Manipulation Layer (FSIO) > Internal Members > gTimeCrtMS Variable

# gTimeCrtTime Variable

**C**

```
WORD gTimeCrtTime;
```

## Description

Global time variable (for timestamps) used to indicate create time

APIs > File Manipulation Layer (FSIO) > Internal Members > gTimeCrtTime Variable

# gTimeWrtDate Variable

**C**

```
WORD gTimeWrtDate;
```

## Description

Global time variable (for timestamps) used to indicate last update date

APIs > File Manipulation Layer (FSIO) > Internal Members > gTimeWrtDate Variable

# gTimeWrtTime Variable

**C**

```
WORD gTimeWrtTime;
```

## Description

Global time variable (for timestamps) used to indicate last update time

APIs > File Manipulation Layer (FSIO) > Internal Members > gTimeWrtTime Variable

# nextClusterIsLast Variable

**C**

```
BYTE nextClusterIsLast = FALSE;
```

## Description

Global variable indicating that the entries in a directory align with a cluster boundary

APIs > File Manipulation Layer (FSIO) > Internal Members > nextClusterIsLast Variable

# recache Variable

C

```
BYTE recache = FALSE;
```

## Description

Global variable used by the "recursive" FSrmdir function to indicate that additional cache reads are needed.

# s_digits Variable

C

```c
const char s_digits[] = "0123456789abcdef";
```

## Description

FSfprintf table of conversion digits

APIs > File Manipulation Layer (FSIO) > Internal Members > s_digits Variable

# tempArray Variable

C

```c
char tempArray[13] = "            ";
```

## Description

This array is used to prevent a stack frame error

APIs > File Manipulation Layer (FSIO) > Internal Members > tempArray Variable

# TempClusterCalc Variable

**C**

```
DWORD TempClusterCalc;
```

## Description

Global variable used to store the calculated value of the cluster of a specified sector.

APIs > File Manipulation Layer (FSIO) > Internal Members > TempClusterCalc Variable

# tempCWDobj Variable

**C**

```
FSFILE tempCWDobj;
```

## Description

Global variable used to preserve the current working directory information.

APIs > File Manipulation Layer (FSIO) > Internal Members > tempCWDobj Variable

# SD-SPI Physical Layer

The SD-SPI physical layer offers the ability to interface to SD cards using the SPI protocol. SPI modules can be found on many Microchip microcontrollers.

## Topics

| Name | Description |
|---|---|
| Public Members | The following functions, variables, structures, and macros are available for use by the user application. |
| Library Members | The following functions, variables, structures, and macros are public, but are intended only to be accessed by the library itself. Applications should generally not call these functions or modify these variables. |
| Internal Members | The following functions, variables, structures, and macros are designated as internal to the library. |

APIs > SD-SPI Physical Layer

# Public Members

The following functions, variables, structures, and macros are available for use by the user application.

## Macros

| | Name | Description |
|---|---|---|
| | SD_CD | SD-SPI Card Detect Input bit |
| | SD_CD_TRIS | SD-SPI Card Detect TRIS bit |
| | SD_CS | SD-SPI Chip Select Output bit |
| | SD_CS_TRIS | SD-SPI Chip Select TRIS bit |
| | SD_WE | SD-SPI Write Protect Check Input bit |
| | SD_WE_TRIS | SD-SPI Write Protect Check TRIS bit |
| | SPI_INTERRUPT_FLAG | The interrupt flag for the SPI module |
| | SPIBRG | The definition for the SPI baud rate generator register (PIC32) |
| | SPIBUF | The SPI Buffer |
| | SPICLOCK | The TRIS bit for the SCK pin |
| | SPICLOCKPORT | The port for the SCK pin |
| | SPICLOCKLAT | The output latch for the SCK pin |

| | | |
|---|---|---|
| | SPICON1 | The main SPI control register |
| | SPICON1bits | The bitwise define for the SPI control register (i.e. _____bits) |
| | SPIENABLE | The enable bit for the SPI module |
| | SPIIN | The TRIS bit for the SDI pin |
| | SPIINPORT | The port for the SDI pin |
| | SPIINLAT | The output latch for the SDI pin |
| | SPIOUT | The TRIS bit for the SDO pin |
| | SPIOUTPORT | The port for the SDO pin |
| | SPIOUTLAT | The output latch for the SDO pin |
| | SPISTAT | The SPI status register |
| | SPISTAT_RBF | The receive buffer full bit in the SPI status register |
| | SPISTATbits | The bitwise define for the SPI status register (i.e. _____bits) |

# SD_CD Macro

C

```c
#define SD_CD PORTFbits.RF0
```

## Description

SD-SPI Card Detect Input bit

APIs > SD-SPI Physical Layer > Public Members > SD_CD Macro

# SD_CD_TRIS Macro

C

```c
#define SD_CD_TRIS TRISFbits.TRISF0
```

## Description

SD-SPI Card Detect TRIS bit

APIs > SD-SPI Physical Layer > Public Members > SD_CD_TRIS Macro

# SD_CS Macro

C

```c
#define SD_CS PORTBbits.RB1
```

## Description

SD-SPI Chip Select Output bit

APIs > SD-SPI Physical Layer > Public Members > SD_CS Macro

# SD_CS_TRIS Macro

C

```
#define SD_CS_TRIS TRISBbits.TRISB1
```

## Description

SD-SPI Chip Select TRIS bit

APIs > SD-SPI Physical Layer > Public Members > SD_CS_TRIS Macro

# SD_WE Macro

C

```c
#define SD_WE PORTFbits.RF1
```

## Description

SD-SPI Write Protect Check Input bit

APIs > SD-SPI Physical Layer > Public Members > SD_WE Macro

# SD_WE_TRIS Macro

C

```c
#define SD_WE_TRIS TRISFbits.TRISF1
```

## Description

SD-SPI Write Protect Check TRIS bit

APIs > SD-SPI Physical Layer > Public Members > SD_WE_TRIS Macro

# SPI_INTERRUPT_FLAG Macro

C

```c
#define SPI_INTERRUPT_FLAG PIR1bits.SSPIF
```

## Description

The interrupt flag for the SPI module

APIs > SD-SPI Physical Layer > Public Members > SPI_INTERRUPT_FLAG Macro

# SPIBRG Macro

C

```c
#define SPIBRG SPI1BRG
```

## Description

The definition for the SPI baud rate generator register (PIC32)

APIs > SD-SPI Physical Layer > Public Members > SPIBRG Macro

# SPIBUF Macro

C

```c
#define SPIBUF SPI1BUF
```

## Description

The SPI Buffer

APIs > SD-SPI Physical Layer > Public Members > SPIBUF Macro

# SPICLOCK Macro

C

```c
#define SPICLOCK TRISFbits.TRISF6
```

## Description

The TRIS bit for the SCK pin

APIs > SD-SPI Physical Layer > Public Members > SPICLOCK Macro

# SPICLOCKPORT Macro

C

```c
#define SPICLOCKPORT PORTCbits.RC3
```

## Description

The port for the SCK pin

APIs > SD-SPI Physical Layer > Public Members > SPICLOCKPORT Macro

# SPICLOCKLAT Macro

C

```c
#define SPICLOCKLAT LATCbits.LATC3
```

## Description

The output latch for the SCK pin

APIs > SD-SPI Physical Layer > Public Members > SPICLOCKLAT Macro

# SPICON1 Macro

C

```c
#define SPICON1 SPI1CON
```

## Description

The main SPI control register

APIs > SD-SPI Physical Layer > Public Members > SPICON1 Macro

# SPICON1bits Macro

C

```c
#define SPICON1bits SPI1CONbits
```

## Description

The bitwise define for the SPI control register (i.e. _____bits)

APIs > SD-SPI Physical Layer > Public Members > SPICON1bits Macro

# SPIENABLE Macro

C

```c
#define SPIENABLE SPICON1bits.ON
```

## Description

The enable bit for the SPI module

APIs > SD-SPI Physical Layer > Public Members > SPIENABLE Macro

# SPIIN Macro

C

```c
#define SPIIN TRISFbits.TRISF7
```

## Description

The TRIS bit for the SDI pin

APIs > SD-SPI Physical Layer > Public Members > SPIIN Macro

# SPIINPORT Macro

C

```c
#define SPIINPORT PORTCbits.RC4
```

## Description

The port for the SDI pin

APIs > SD-SPI Physical Layer > Public Members > SPIINPORT Macro

# SPIINLAT Macro

C

```c
#define SPIINLAT LATCbits.LATC4
```

## Description

The output latch for the SDI pin

APIs > SD-SPI Physical Layer > Public Members > SPIINLAT Macro

# SPIOUT Macro

C

```c
#define SPIOUT TRISFbits.TRISF8
```

## Description

The TRIS bit for the SDO pin

APIs > SD-SPI Physical Layer > Public Members > SPIOUT Macro

# SPIOUTPORT Macro

C

```c
#define SPIOUTPORT PORTCbits.RC5
```

## Description

The port for the SDO pin

APIs > SD-SPI Physical Layer > Public Members > SPIOUTPORT Macro

# SPIOUTLAT Macro

C

```c
#define SPIOUTLAT LATCbits.LATC5
```

## Description

The output latch for the SDO pin

APIs > SD-SPI Physical Layer > Public Members > SPIOUTLAT Macro

# SPISTAT Macro

C

```
#define SPISTAT SPI1STAT
```

## Description

The SPI status register

APIs > SD-SPI Physical Layer > Public Members > SPISTAT Macro

# SPISTAT_RBF Macro

C

```c
#define SPISTAT_RBF SPI1STATbits.SPIRBF
```

## Description

The receive buffer full bit in the SPI status register

APIs > SD-SPI Physical Layer > Public Members > SPISTAT_RBF Macro

# SPISTATbits Macro

C

```c
#define SPISTATbits SPI1STATbits
```

## Description

The bitwise define for the SPI status register (i.e. _____bits)

APIs > SD-SPI Physical Layer > Public Members > SPISTATbits Macro

# Library Members

The following functions, variables, structures, and macros are public, but are intended only to be accessed by the library itself. Applications should generally not call these functions or modify these variables.

## Enumerations

| | Name | Description |
|---|---|---|
| | RESP | Enumeration of different SD response types |
| | sdmmc_cmd | An enumeration of SD commands |

## Functions

| | Name | Description |
|---|---|---|
| | MDD_SDSPI_InitIO | Initializes the I/O lines connected to the card |
| | MDD_SDSPI_MediaDetect | Determines whether an SD card is present |
| | MDD_SDSPI_MediaInitialize | Initializes the SD card. |
| | MDD_SDSPI_ReadCapacity | Determines the current capacity of the SD card |
| | MDD_SDSPI_ReadMedia | Reads a byte of data from the SD card. |
| | MDD_SDSPI_ReadSectorSize | Determines the current sector size on the SD card |

| | | |
|---|---|---|
| | MDD_SDSPI_SectorWrite | Writes a sector of data to an SD card. |
| | MDD_SDSPI_SectorRead | Reads a sector of data from an SD card. |
| | MDD_SDSPI_ShutdownMedia | Disables the SD card |
| | MDD_SDSPI_WriteProtectState | Indicates whether the card is write-protected. |

## Macros

| | Name | Description |
|---|---|---|
| | cmdAPP_CMD | This macro defines the command code to begin application specific command inputs |
| | cmdCRC_ON_OFF | This macro defines the command code to disable CRC checking |
| | cmdERASE | This macro defines the command code to erase all previously selected blocks |
| | cmdGO_IDLE_STATE | This macro defines the command code to reset the SD card |
| | cmdREAD_MULTI_BLOCK | This macro defines the command code to read multiple blocks from the card |
| | | This macro defines the command code to get the OCR |

| | | |
|---|---|---|
| | cmdREAD_OCR | register information from the card |
| | cmdREAD_SINGLE_BLOCK | This macro defines the command code to read one block from the card |
| | cmdSEND_CID | This macro defines the command code to get the Card Information |
| | cmdSEND_CSD | This macro defines the command code to get the Card Specific Data |
| | cmdSEND_OP_COND | This macro defines the command code to initialize the SD card |
| | cmdSEND_STATUS | This macro defines the command code to get the card status information |
| | cmdSET_BLOCKLEN | This macro defines the command code to set the block length of the card |
| | cmdSTOP_TRANSMISSION | This macro defines the command code to stop transmission during a multi-block read |
| | cmdTAG_SECTOR_END | This macro defines the command code to set the address of the end of an erase operation |
| | | This macro defines the |

| | | |
|---|---|---|
| | cmdTAG_SECTOR_START | command code to set the address of the start of an erase operation |
| | cmdWRITE_MULTI_BLOCK | This macro defines the command code to write multiple blocks to the card |
| | cmdWRITE_SINGLE_BLOCK | This macro defines the command code to write one block to the card |
| | DATA_ACCEPTED | This macro represents an SD card data accepted token |
| | DATA_START_TOKEN | This macro represents an SD card start token |
| | DELAY_OVERHEAD | An approximation of the number of cycles per delay loop of overhead |
| | DELAY_PRESCALER | A delay prescaler |
| | MASTER_ENABLE_ON | This macro indicates the SPI enable bit for 16-bit PICs |
| | MILLISECDELAY | An approximate calculation of how many times to loop to delay 1 ms in the Delayms function |
| | MMC_BAD_RESPONSE | This macro represents a bad SD card response byte |
| | MMC_FLOATING_BUS | This macro represents a floating SPI bus condition |
| | | |

| | | |
|---|---|---|
| | MOREDATA | This macro indicates that the SD card expects to transmit or receive more data |
| | mReadCRC | A macro to send clock cycles to dummy-read the CRC |
| | mSend8ClkCycles | A macro to send 8 clock cycles for SD timing requirements |
| | mSendCRC | A macro to send clock cycles to dummy-write the CRC |
| | NODATA | This macro indicates that the SD card does not expect to transmit or receive more data |
| | PRI_PRESCAL_1_1 | This macro is used to initialize a 16-bit PIC SPI module primary prescaler |
| | SEC_PRESCAL_1_1 | This macro is used to initialize a 16-bit PIC SPI module secondary prescaler |
| | SYNC_MODE_FAST | This macro is used to initialize a 16-bit PIC SPI module |
| | SYNC_MODE_MED | This macro is used to initialize a PIC18 SPI module with a 16x prescale divider |
| | SYNC_MODE_SLOW | This macro is used to initialize a 16-bit PIC SPI module |

## Structures

| | | |
|---|---|---|

| | Name | Description |
|---|---|---|
| | typMMC_CMD | SD card command data structure |

## Unions

| | Name | Description |
|---|---|---|
| | CID | A description of the card information register |
| | CMD_PACKET | An SD command packet |
| | CSD | A description of the card specific data register |
| | RESPONSE_1 | The format of an R1 type response |
| | RESPONSE_2 | The format of an R2 type response |
| | MMC_RESPONSE | A union of responses from an SD card |

# MDD_SDSPI_InitIO Function

```
C
void MDD_SDSPI_InitIO();
```

## Description

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

## Preconditions

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

## Returns

None

## Side Effects

None.

## Remarks

None

APIs > SD-SPI Physical Layer > Library Members > MDD_SDSPI_InitIO Function

# MDD_SDSPI_MediaDetect Function

**C**

```
BYTE MDD_SDSPI_MediaDetect();
```

## Description

The MDD_SDSPI_MediaDetect function will determine if an SD card is connected to the microcontroller by polling the SD card detect pin.

## 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 |

## Side Effects

None.

## Remarks

None

# MDD_SDSPI_MediaInitialize Function

**C**

```
BYTE MDD_SDSPI_MediaInitialize();
```

## Description

This function will send initialization commands to and SD card.

## Preconditions

The MDD_MediaInitialize function pointer must be pointing to this function.

## Return Values

| Return Values | Description |
|---|---|
| TRUE | The card was successfully initialized |
| FALSE | Communication could not be established. |

## Side Effects

None.

## Remarks

None.

# MDD_SDSPI_ReadCapacity Function

C

```
DWORD MDD_SDSPI_ReadCapacity();
```

## Description

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

## Preconditions

MDD_MediaInitialize() is complete

## Returns

The capacity of the device

## Side Effects

None.

## Remarks

None

APIs > SD-SPI Physical Layer > Library Members > MDD_SDSPI_ReadCapacity Function

# MDD_SDSPI_ReadMedia Function

**C**

```
BYTE MDD_SDSPI_ReadMedia();
```

## Description

The MDD_SDSPI_ReadMedia function will read one byte from the SPI port.

## Preconditions

None.

## Returns

The byte read.

## Side Effects

None.

## Remarks

This function replaces ReadSPI, since some implementations of that function will initialize SSPBUF/SPIBUF to 0x00 when reading. The card expects 0xFF.

APIs > SD-SPI Physical Layer > Library Members > MDD_SDSPI_ReadMedia Function

# MDD_SDSPI_ReadSectorSize Function

C

```
WORD MDD_SDSPI_ReadSectorSize();
```

## 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.

## Preconditions

MDD_MediaInitialize() is complete

## Returns

The size of the sectors for the physical media

## Side Effects

None.

## Remarks

None

APIs > SD-SPI Physical Layer > Library Members > MDD_SDSPI_ReadSectorSize Function

# MDD_SDSPI_SectorWrite Function

C

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

## Description

The MDD_SDSPI_SectorWrite function writes 512 bytes of data from the location pointed to by 'buffer' to the specified sector of the SD card.

## 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<br>• 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. |

## Side Effects

None.

## 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).

# MDD_SDSPI_SectorRead Function

C

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

## Description

The MDD_SDSPI_SectorRead function reads 512 bytes of data from the SD card starting at the sector address and stores them in the location pointed to by 'buffer.'

## 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. |
| byffer | 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 |

## Side Effects

None

## 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).

# MDD_SDSPI_ShutdownMedia Function

C

```c
void MDD_SDSPI_ShutdownMedia();
```

## Description

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

## Preconditions

The MDD_ShutdownMedia function pointer is pointing towards this function.

## Returns

None

## Side Effects

None.

## Remarks

None

APIs > SD-SPI Physical Layer > Library Members > MDD_SDSPI_ShutdownMedia Function

# cmdAPP_CMD Macro

C

```c
#define cmdAPP_CMD 55
```

## Description

This macro defines the command code to begin application specific command inputs

APIs > SD-SPI Physical Layer > Library Members > cmdAPP_CMD Macro

# cmdCRC_ON_OFF Macro

C

```c
#define cmdCRC_ON_OFF 59
```

## Description

This macro defines the command code to disable CRC checking

APIs > SD-SPI Physical Layer > Library Members > cmdCRC_ON_OFF Macro

# cmdERASE Macro

C

```
#define cmdERASE 38
```

## Description

This macro defines the command code to erase all previously selected blocks

APIs > SD-SPI Physical Layer > Library Members > cmdERASE Macro

# cmdGO_IDLE_STATE Macro

C

```c
#define cmdGO_IDLE_STATE 0
```

## Description

This macro defines the command code to reset the SD card

APIs > SD-SPI Physical Layer > Library Members > cmdGO_IDLE_STATE Macro

# cmdREAD_MULTI_BLOCK Macro

C

```c
#define cmdREAD_MULTI_BLOCK 18
```

## Description

This macro defines the command code to read multiple blocks from the card

APIs > SD-SPI Physical Layer > Library Members > cmdREAD_MULTI_BLOCK Macro

# cmdREAD_OCR Macro

C

```c
#define cmdREAD_OCR 58
```

## Description

This macro defines the command code to get the OCR register information from the card

APIs > SD-SPI Physical Layer > Library Members > cmdREAD_OCR Macro

# cmdREAD_SINGLE_BLOCK Macro

C

```c
#define cmdREAD_SINGLE_BLOCK 17
```

## Description

This macro defines the command code to read one block from the card

APIs > SD-SPI Physical Layer > Library Members > cmdREAD_SINGLE_BLOCK Macro

# cmdSEND_CID Macro

C

```c
#define cmdSEND_CID 10
```

## Description

This macro defines the command code to get the Card Information

APIs > SD-SPI Physical Layer > Library Members > cmdSEND_CID Macro

# cmdSEND_CSD Macro

C

```c
#define cmdSEND_CSD 9
```

## Description

This macro defines the command code to get the Card Specific Data

APIs > SD-SPI Physical Layer > Library Members > cmdSEND_CSD Macro

# cmdSEND_OP_COND Macro

C

```c
#define cmdSEND_OP_COND 1
```

## Description

This macro defines the command code to initialize the SD card

APIs > SD-SPI Physical Layer > Library Members > cmdSEND_OP_COND Macro

# cmdSEND_STATUS Macro

C

```c
#define cmdSEND_STATUS 13
```

## Description

This macro defines the command code to get the card status information

APIs > SD-SPI Physical Layer > Library Members > cmdSEND_STATUS Macro

# cmdSET_BLOCKLEN Macro

C

```c
#define cmdSET_BLOCKLEN 16
```

## Description

This macro defines the command code to set the block length of the card

APIs > SD-SPI Physical Layer > Library Members > cmdSET_BLOCKLEN Macro

# cmdSTOP_TRANSMISSION Macro

C

```c
#define cmdSTOP_TRANSMISSION 12
```

## Description

This macro defines the command code to stop transmission during a multi-block read

APIs > SD-SPI Physical Layer > Library Members > cmdSTOP_TRANSMISSION Macro

# cmdTAG_SECTOR_END Macro

C

```c
#define cmdTAG_SECTOR_END 33
```

## Description

This macro defines the command code to set the address of the end of an erase operation

APIs > SD-SPI Physical Layer > Library Members > cmdTAG_SECTOR_END Macro

# cmdTAG_SECTOR_START Macro

C

```c
#define cmdTAG_SECTOR_START 32
```

## Description

This macro defines the command code to set the address of the start of an erase operation

APIs > SD-SPI Physical Layer > Library Members > cmdTAG_SECTOR_START Macro

# cmdWRITE_MULTI_BLOCK Macro

C

```c
#define cmdWRITE_MULTI_BLOCK 25
```

## Description

This macro defines the command code to write multiple blocks to the card

APIs > SD-SPI Physical Layer > Library Members > cmdWRITE_MULTI_BLOCK Macro

# cmdWRITE_SINGLE_BLOCK Macro

C

```c
#define cmdWRITE_SINGLE_BLOCK 24
```

## Description

This macro defines the command code to write one block to the card

APIs > SD-SPI Physical Layer > Library Members > cmdWRITE_SINGLE_BLOCK Macro

# DATA_ACCEPTED Macro

C

```c
#define DATA_ACCEPTED 0x05
```

## Description

This macro represents an SD card data accepted token

APIs > SD-SPI Physical Layer > Library Members > DATA_ACCEPTED Macro

# DATA_START_TOKEN Macro

C

```
#define DATA_START_TOKEN 0xFE
```

## Description

This macro represents an SD card start token

APIs > SD-SPI Physical Layer > Library Members > DATA_START_TOKEN Macro

# DELAY_OVERHEAD Macro

C

```
#define DELAY_OVERHEAD (BYTE)        5
```

## Description

An approximation of the number of cycles per delay loop of overhead

# DELAY_PRESCALER Macro

**C**

```
#define DELAY_PRESCALER (BYTE)        8
```

## Description

A delay prescaler

APIs > SD-SPI Physical Layer > Library Members > DELAY_PRESCALER Macro

# MASTER_ENABLE_ON Macro

C

```c
#define MASTER_ENABLE_ON 0x0020
```

## Description

This macro indicates the SPI enable bit for 16-bit PICs

APIs > SD-SPI Physical Layer > Library Members > MASTER_ENABLE_ON Macro

# MILLISECDELAY Macro

C

```
#define MILLISECDELAY (WORD)      ((GetInstructionCl
```

## Description

An approximate calculation of how many times to loop to delay 1 ms in the Delayms function

APIs > SD-SPI Physical Layer > Library Members > MILLISECDELAY Macro

# MMC_BAD_RESPONSE Macro

C

```
#define MMC_BAD_RESPONSE MMC_FLOATING_BUS
```

## Description

This macro represents a bad SD card response byte

APIs > SD-SPI Physical Layer > Library Members > MMC_BAD_RESPONSE Macro

# MMC_FLOATING_BUS Macro

C

```c
#define MMC_FLOATING_BUS 0xFF
```

## Description

This macro represents a floating SPI bus condition

APIs > SD-SPI Physical Layer > Library Members > MMC_FLOATING_BUS Macro

# MOREDATA Macro

C

```c
#define MOREDATA !0
```

## Description

This macro indicates that the SD card expects to transmit or receive more data

APIs > SD-SPI Physical Layer > Library Members > MOREDATA Macro

# mReadCRC Macro

C

```c
#define mReadCRC WriteSPIM(0xFF);WriteSPIM(0xFF);
```

## Description

A macro to send clock cycles to dummy-read the CRC

APIs > SD-SPI Physical Layer > Library Members > mReadCRC Macro

# mSend8ClkCycles Macro

C

```c
#define mSend8ClkCycles WriteSPIM(0xFF);
```

## Description

A macro to send 8 clock cycles for SD timing requirements

APIs > SD-SPI Physical Layer > Library Members > mSend8ClkCycles Macro

# mSendCRC Macro

C

```c
#define mSendCRC WriteSPIM(0xFF);WriteSPIM(0xFF);
```

## Description

A macro to send clock cycles to dummy-write the CRC

APIs > SD-SPI Physical Layer > Library Members > mSendCRC Macro

# NODATA Macro

C

```c
#define NODATA 0
```

## Description

This macro indicates that the SD card does not expect to transmit or receive more data

APIs > SD-SPI Physical Layer > Library Members > NODATA Macro

# PRI_PRESCAL_1_1 Macro

C

```c
#define PRI_PRESCAL_1_1 0x0003
```

## Description

This macro is used to initialize a 16-bit PIC SPI module primary prescaler

APIs > SD-SPI Physical Layer > Library Members > PRI_PRESCAL_1_1 Macro

# SEC_PRESCAL_1_1 Macro

C

```c
#define SEC_PRESCAL_1_1 0x001c
```

## Description

This macro is used to initialize a 16-bit PIC SPI module secondary prescaler

APIs > SD-SPI Physical Layer > Library Members > SEC_PRESCAL_1_1 Macro

# SYNC_MODE_FAST Macro

C

```c
#define SYNC_MODE_FAST 0x3E
```

## Description

This macro is used to initialize a 16-bit PIC SPI module

APIs > SD-SPI Physical Layer > Library Members > SYNC_MODE_FAST Macro

# SYNC_MODE_MED Macro

C

```c
#define SYNC_MODE_MED 0x01
```

## Description

This macro is used to initialize a PIC18 SPI module with a 16x prescale divider

APIs > SD-SPI Physical Layer > Library Members > SYNC_MODE_MED Macro

# SYNC_MODE_SLOW Macro

C

```c
#define SYNC_MODE_SLOW 0x3C
```

## Description

This macro is used to initialize a 16-bit PIC SPI module

APIs > SD-SPI Physical Layer > Library Members > SYNC_MODE_SLOW Macro

# CID Union

C

```c
typedef union {
  struct {
    DWORD _u320;
    DWORD _u321;
    DWORD _u322;
    DWORD _u323;
  }
  struct {
    BYTE _byte[16];
  }
  struct {
    unsigned NOT_USED : 1;
    unsigned CRC : 7;
    unsigned MDT : 8;
    DWORD PSN;
    unsigned PRV : 8;
    char PNM[6];
    WORD OID;
    unsigned MID : 8;
  }
} CID;
```

## Description

This union represents different ways to access information in a packet with SD card CID register informaiton. For more information on the CID register, consult an SD card user's manual.

# CMD_PACKET Union

C

```c
typedef union {
  struct {
    BYTE field[7];
  }
  struct {
    BYTE crc;
    BYTE c30filler;
    BYTE c32filler[3];
    BYTE addr0;
    BYTE addr1;
    BYTE addr2;
    BYTE addr3;
    BYTE cmd;
  }
  struct {
    BYTE END_BIT : 1;
    BYTE CRC7 : 7;
    DWORD address;
    BYTE CMD_INDEX : 6;
    BYTE TRANSMIT_BIT : 1;
    BYTE START_BIT : 1;
  }
} CMD_PACKET;
```

## Description

This union represents different ways to access an SD card command packet

## Members

| | |
|---|---|

| Members | Description |
| --- | --- |
| BYTE field[7]; | BYTE array |
| BYTE crc; | The CRC byte |
| BYTE c30filler; | Filler space (since bitwise declarations can't cross a WORD boundary) |
| BYTE c32filler[3]; | Filler space (since bitwise declarations can't cross a DWORD boundary) |
| BYTE addr0; | Address byte 0 |
| BYTE addr1; | Address byte 1 |
| BYTE addr2; | Address byte 2 |
| BYTE addr3; | Address byte 3 |
| BYTE cmd; | Command code byte |
| BYTE END_BIT : 1; | Packet end bit |
| BYTE CRC7 : 7; | CRC value |
| DWORD address; | Address |
| BYTE CMD_INDEX : 6; | Command code |
| BYTE TRANSMIT_BIT : 1; | Transmit bit |
| BYTE START_BIT : 1; | Packet start bit |

# CSD Union

C

```c
typedef union {
  struct {
    DWORD _u320;
    DWORD _u321;
    DWORD _u322;
    DWORD _u323;
  }
  struct {
    BYTE _byte[16];
  }
  struct {
    unsigned NOT_USED : 1;
    unsigned CRC : 7;
    unsigned ECC : 2;
    unsigned FILE_FORMAT : 2;
    unsigned TMP_WRITE_PROTECT : 1;
    unsigned PERM_WRITE_PROTECT : 1;
    unsigned COPY : 1;
    unsigned FILE_FORMAT_GRP : 1;
    unsigned RESERVED_1 : 5;
    unsigned WRITE_BL_PARTIAL : 1;
    unsigned WRITE_BL_LEN_L : 2;
    unsigned WRITE_BL_LEN_H : 2;
    unsigned R2W_FACTOR : 3;
    unsigned DEFAULT_ECC : 2;
    unsigned WP_GRP_ENABLE : 1;
    unsigned WP_GRP_SIZE : 5;
    unsigned ERASE_GRP_SIZE_L : 3;
    unsigned ERASE_GRP_SIZE_H : 2;
    unsigned SECTOR_SIZE : 5;
    unsigned C_SIZE_MULT_L : 1;
    unsigned C_SIZE_MULT_H : 2;
```

```
    unsigned VDD_W_CURR_MAX : 3;
    unsigned VDD_W_CUR_MIN : 3;
    unsigned VDD_R_CURR_MAX : 3;
    unsigned VDD_R_CURR_MIN : 3;
    unsigned C_SIZE_L : 2;
    unsigned C_SIZE_H : 8;
    unsigned C_SIZE_U : 2;
    unsigned RESERVED_2 : 2;
    unsigned DSR_IMP : 1;
    unsigned READ_BLK_MISALIGN : 1;
    unsigned WRITE_BLK_MISALIGN : 1;
    unsigned READ_BL_PARTIAL : 1;
    unsigned READ_BL_LEN : 4;
    unsigned CCC_L : 4;
    unsigned CCC_H : 8;
    unsigned TRAN_SPEED : 8;
    unsigned NSAC : 8;
    unsigned TAAC : 8;
    unsigned RESERVED_3 : 2;
    unsigned SPEC_VERS : 4;
    unsigned CSD_STRUCTURE : 2;
  }
} CSD;
```

## Description

This union represents different ways to access information in a packet with SD card CSD informaiton. For more information on the CSD register, consult an SD card user's manual.

# RESPONSE_1 Union

C

```c
typedef union {
  BYTE _byte;
  struct {
    unsigned IN_IDLE_STATE : 1;
    unsigned ERASE_RESET : 1;
    unsigned ILLEGAL_CMD : 1;
    unsigned CRC_ERR : 1;
    unsigned ERASE_SEQ_ERR : 1;
    unsigned ADDRESS_ERR : 1;
    unsigned PARAM_ERR : 1;
    unsigned B7 : 1;
  }
} RESPONSE_1;
```

## Description

This union represents different ways to access an SD card R1 type response packet.

## Members

| Members | Description |
|---|---|
| BYTE _byte; | Byte-wise access This structure allows bitwise access of the response |
| unsigned IN_IDLE_STATE : 1; | Card is in idle state |
| unsigned ERASE_RESET : 1; | Erase reset flag |
| | |

| | |
|---|---|
| unsigned ILLEGAL_CMD : 1; | Illegal command flag |
| unsigned CRC_ERR : 1; | CRC error flag |
| unsigned ERASE_SEQ_ERR : 1; | Erase sequence error flag |
| unsigned ADDRESS_ERR : 1; | Address error flag |
| unsigned PARAM_ERR : 1; | Parameter flag |
| unsigned B7 : 1; | Unused bit 7 |

# RESPONSE_2 Union

C

```c
typedef union {
  WORD _word;
  struct {
    BYTE _byte0;
    BYTE _byte1;
  }
  struct {
    unsigned IN_IDLE_STATE : 1;
    unsigned ERASE_RESET : 1;
    unsigned ILLEGAL_CMD : 1;
    unsigned CRC_ERR : 1;
    unsigned ERASE_SEQ_ERR : 1;
    unsigned ADDRESS_ERR : 1;
    unsigned PARAM_ERR : 1;
    unsigned B7 : 1;
    unsigned CARD_IS_LOCKED : 1;
    unsigned WP_ERASE_SKIP_LK_FAIL : 1;
    unsigned ERROR : 1;
    unsigned CC_ERROR : 1;
    unsigned CARD_ECC_FAIL : 1;
    unsigned WP_VIOLATION : 1;
    unsigned ERASE_PARAM : 1;
    unsigned OUTRANGE_CSD_OVERWRITE : 1;
  }
} RESPONSE_2;
```

## Description

This union represents different ways to access an SD card R2 type response packet

# Union

# MMC_RESPONSE Union

C

```c
typedef union {
  RESPONSE_1 r1;
  RESPONSE_2 r2;
} MMC_RESPONSE;
```

## Description

The MMC_RESPONSE union represents any of the possible responses that an SD card can return after being issued a command.

APIs > SD-SPI Physical Layer > Library Members > MMC_RESPONSE Union

# RESP Enumeration

C

```c
typedef enum {
  R1,
  R1b,
  R2,
  R3
} RESP;
```

## Description

Enumeration of different SD response types

## Members

| Members | Description |
|---|---|
| R1 | R1 type response |
| R1b | R1b type response |
| R2 | R2 type response |
| R3 | R3 type response |

# sdmmc_cmd Enumeration

C

```c
typedef enum {
  GO_IDLE_STATE,
  SEND_OP_COND,
  SEND_CSD,
  SEND_CID,
  STOP_TRANSMISSION,
  SEND_STATUS,
  SET_BLOCKLEN,
  READ_SINGLE_BLOCK,
  READ_MULTI_BLOCK,
  WRITE_SINGLE_BLOCK,
  WRITE_MULTI_BLOCK,
  TAG_SECTOR_START,
  TAG_SECTOR_END,
  ERASE,
  APP_CMD,
  READ_OCR,
  CRC_ON_OFF
} sdmmc_cmd;
```

## Description

This enumeration corresponds to the position of each command in the sdmmc_cmdtable array These macros indicate to the SendMMCCmd function which element of the sdmmc_cmdtable array to retrieve command code information from.

APIs > SD-SPI Physical Layer > Library Members > sdmmc_cmd Enumeration

# typMMC_CMD Structure

C

```c
typedef struct {
  BYTE CmdCode;
  BYTE CRC;
  RESP responsetype;
  BYTE moredataexpected;
} typMMC_CMD;
```

## Description

The typMMC_CMD structure is used to create a command table of information needed for each relevant SD command

## Members

| Members | Description |
|---|---|
| BYTE CmdCode; | The command code |
| BYTE CRC; | The CRC value for that command |
| RESP responsetype; | The response type |
| BYTE moredataexpected; | Set to MOREDATA or NODATA, depending on whether more data is expected or not |

APIs > SD-SPI Physical Layer > Library Members > typMMC_CMD Structure

# Internal Members

The following functions, variables, structures, and macros are designated as internal to the library.

## Functions

| | Name | Description |
|---|---|---|
| | Delayms | Delay. |
| | CloseSPIM | Disables the SPI module. |
| | OpenSPIM | This is function OpenSPIM. |
| | ReadMediaManual | Reads a byte of data from the SD card. |
| | SendMMCCmd | Sends a command packet to the SD card. |
| | SendMMCCmdManual | Sends a command packet to the SD card with bit-bang SPI. |
| | WriteSPIM | Writes data to the SD card. |
| | WriteSPIManual | Write a character to the SD card with bit-bang SPI. |

## Macros

| | Name | Description |
|---|---|---|
| | MANUAL_SPI_CLOCK_VALUE | Delay value for the manual SPI clock |

## Variables

| | Name | Description |
|---|---|---|
| | sdmmc_cmdtable | Table of SD card commands and parameters |
| | MDD_SDSPI_finalLBA | Used for the mass-storage library to determine capacity |

# Delayms Function

C

```c
void Delayms(
    BYTE milliseconds
);
```

## Description

The Delayms function will delay a specified number of milliseconds. Used for SPI timing.

## Preconditions

None.

## Parameters

| Parameters | Description |
|---|---|
| BYTE milliseconds | Number of ms to delay |

## Returns

None.

## Side Effects

None.

## Remarks

Depending on compiler revisions, this function may delay for the exact time specified. This shouldn't create a significant problem.

# CloseSPIM Function

```
C
void CloseSPIM();
```

## Description

Disables the SPI module.

## Preconditions

None.

## Returns

None.

## Side Effects

None.

## Remarks

None.

APIs > SD-SPI Physical Layer > Internal Members > CloseSPIM Function

# OpenSPIM Function

C

```c
void OpenSPIM(
    unsigned int sync_mode
);
```

## Description

This is function OpenSPIM.

APIs > SD-SPI Physical Layer > Internal Members > OpenSPIM Function

# ReadMediaManual Function

**C**

```
BYTE ReadMediaManual();
```

## Description

The MDD_SDSPI_ReadMedia function will read one byte from the SPI port.

## Preconditions

None.

## Returns

The byte read.

## Side Effects

None.

## Remarks

This function replaces ReadSPI, since some implementations of that function will initialize SSPBUF/SPIBUF to 0x00 when reading. The card expects 0xFF. This function is for use on a PIC18 when the clock speed is so high that the maximum SPI clock prescaler cannot reduce the SPI clock below the maximum SD card initialization speed.

APIs > SD-SPI Physical Layer > Internal Members > ReadMediaManual Function

# SendMMCCmd Function

```c
MMC_RESPONSE SendMMCCmd(
    BYTE cmd,
    DWORD address
);
```

## Description

SendMMCCmd prepares a command packet and sends it out over the SPI interface. Response data of type 'R1' (as indicated by the SD/MMC product manual is returned.

## Preconditions

None.

## Return Values

| Return Values | Description |
|---|---|
| MMC_RESPONSE | The response from the card<br>• Bit 0 - Idle state<br>• Bit 1 - Erase Reset<br>• Bit 2 - Illegal Command<br>• Bit 3 - Command CRC Error<br>• Bit 4 - Erase Sequence Error<br>• Bit 5 - Address Error<br>• Bit 6 - Parameter Error<br>• Bit 7 - Unused. Always 0. |

## Side Effects

None.

## Remarks

None.

# SendMMCCmdManual Function

C

```c
MMC_RESPONSE SendMMCCmdManual(
    BYTE cmd,
    DWORD address
);
```

## Description

SendMMCCmd prepares a command packet and sends it out over the SPI interface. Response data of type 'R1' (as indicated by the SD/MMC product manual is returned. This function is intended to be used when the clock speed of a PIC18 device is so high that the maximum SPI divider can't reduce the clock below the maximum SD card initialization sequence speed.

## Preconditions

None.

## Return Values

| Return Values | Description |
|---|---|
| MMC_RESPONSE | The response from the card<br>• Bit 0 - Idle state<br>• Bit 1 - Erase Reset<br>• Bit 2 - Illegal Command<br>• Bit 3 - Command CRC Error<br>• Bit 4 - Erase Sequence Error<br>• Bit 5 - Address Error<br>• Bit 6 - Parameter Error<br>• Bit 7 - Unused. Always 0. |

| | |
|---|---|

## Side Effects

None.

## Remarks

None.

# WriteSPIM Function

C

```c
unsigned char WriteSPIM(
    unsigned char data_out
);
```

## Description

The WriteSPIM function will write a byte of data from the microcontroller to the SD card.

## Preconditions

None.

## Parameters

| Parameters | Description |
|---|---|
| data_out | The data to write. |

## Returns

0.

## Side Effects

None.

## Remarks

None.

# WriteSPIManual Function

C

```c
unsigned char WriteSPIManual(
    unsigned char data_out
);
```

## Description

Writes a character to the SD card.

## Preconditions

None.

## Parameters

| Parameters | Description |
| --- | --- |
| data_out | Data to send. |

## Returns

0.

## Side Effects

None.

## Remarks

The WriteSPIManual function is for use on a PIC18 when the clock speed is so high that the maximum SPI clock divider cannot reduce the SPI clock speed below the maximum SD card

initialization speed.

# MANUAL_SPI_CLOCK_VALUE Macro

C

```c
#define MANUAL_SPI_CLOCK_VALUE 1
```

## Description

Delay value for the manual SPI clock

APIs > SD-SPI Physical Layer > Internal Members > MANUAL_SPI_CLOCK_VALUE Macro

# sdmmc_cmdtable Variable

C

```
const rom typMMC_CMD sdmmc_cmdtable[] = const typMMC
```

## Description

The sdmmc_cmdtable contains an array of SD card commands, the corresponding CRC code, the response type that the card will return, and a parameter indicating whether to expect additional data from the card.

# MDD_SDSPI_finalLBA Variable

**C**

```
DWORD MDD_SDSPI_finalLBA;
```

## Description

Used for the mass-storage library to determine capacity

APIs > SD-SPI Physical Layer > Internal Members > MDD_SDSPI_finalLBA Variable

# CF Physical Layer

The CF physical layers offer two methods for interfacing with CF cards. The manual interface method will bit-bang the parallel interface protocol used by CF cards. The CF-PMP files will interface to the cards using the parallel master port on 16-bit PIC devices. At this time, 8-bit architecture PMP interface is not supported.

## Topics

| Name | Description |
|---|---|
| Public Members | The following functions, variables, structures, and macros are available for use by the user application. |
| Library Members | The following functions, variables, structures, and macros are public, but are intended only to be accessed by the library itself. Applications should generally not call these functions or modify these variables. |

APIs > CF Physical Layer

# Public Members

The following functions, variables, structures, and macros are available for use by the user application.

## Functions

| | Name | Description |
|---|---|---|
| | MDD_CFBT_MediaDetect | Determines if a card is inserted |
| | MDD_CFPMP_MediaDetect | Determines if a card is inserted |

## Macros

| | Name | Description |
|---|---|---|
| | MDD_CFBT_DATABIN | The Manual CF data bus port register |
| | MDD_CFBT_DATABOUT | The Manual CF data bus output latch register |
| | MDD_CFBT_DATADIR | The Manual CF data bus TRIS register |
| | MDD_CFPMP_DATADIR | Defines the PMP data bus direction register |
| | MDD_CFread | Function pointer to the CompactFlash Read Physical Layer function |
| | MDD_CFwait | Function pointer to the CompactFlash Wait Physical Layer function |

| | | |
|---|---|---|
| | MDD_CFwrite | Function pointer to the CompactFlash Write Physical Layer function |
| | ADDBL | The CF address bus output latch register (for PIC18) |
| | ADDDIR | The CF address bus TRIS register (for PIC18) |
| | ADDR0 | The CF address bus bit 0 output latch definition (for PIC24/30/33/32) |
| | ADDR1 | The CF address bus bit 1 output latch definition (for PIC24/30/33/32) |
| | ADDR2 | The CF address bus bit 2 output latch definition (for PIC24/30/33/32) |
| | ADDR3 | The CF address bus bit 3 output latch definition (for PIC24/30/33/32) |
| | ADRTRIS0 | The CF address bus bit 0 TRIS definition (for PIC24/30/33/32) |
| | ADRTRIS1 | The CF address bus bit 1 TRIS definition (for PIC24/30/33/32) |
| | ADRTRIS2 | The CF address bus bit 2 TRIS definition (for PIC24/30/33/32) |
| | ADRTRIS3 | The CF address bus bit 3 TRIS definition (for PIC24/30/33/32) |

| | | |
|---|---|---|
| | CF_BT_CD1 | The CF card detect signal port bit |
| | CF_BT_CD1DIR | The CF card detect signal TRIS bit |
| | CF_BT_RDY | The CF card ready signal port bit |
| | CF_BT_READYDIR | The CF card ready signal TRIS bit |
| | CF_BT_RESETDIR | The CF card reset signal TRIS bit |
| | CF_BT_RST | The CF card reset signal latch bit |
| | CF_CE | The CF card chip select output latch bit |
| | CF_CEDIR | The CF card chip select TRIS bit |
| | CF_OE | The CF card output enable strobe latch bit |
| | CF_OEDIR | The CF card output enable strobe TRIS bit |
| | CF_PMP_CD1 | The input port for the CF card detect signal |
| | CF_PMP_CD1DIR | The TRIS bit for the CF card detect signal |
| | CF_PMP_RDY | The input port for the CF Ready signal |
| | CF_PMP_READYDIR | The TRIS bit for the CF Ready signal |
| | CF_PMP_RESETDIR | The TRIS bit for the CF Reset signal |
| | | The output latch for the CF Reset |

| | | |
|---|---|---|
| | CF_PMP_RST | signal |
| | CF_WE | The CF card write enable strobe latch bit |
| | CF_WEDIR | The CF card write enable strobe TRIS bit |

# MDD_CFBT_DATABIN Macro

C

```c
#define MDD_CFBT_DATABIN PORTE
```

## Description

The Manual CF data bus port register

APIs > CF Physical Layer > Public Members > MDD_CFBT_DATABIN Macro

# MDD_CFBT_DATABOUT Macro

C

```
#define MDD_CFBT_DATABOUT PORTE
```

## Description

The Manual CF data bus output latch register

APIs > CF Physical Layer > Public Members > MDD_CFBT_DATABOUT Macro

# MDD_CFBT_DATADIR Macro

C

```c
#define MDD_CFBT_DATADIR TRISE
```

## Description

The Manual CF data bus TRIS register

APIs > CF Physical Layer > Public Members > MDD_CFBT_DATADIR Macro

# MDD_CFBT_MediaDetect Function

**C**

```
BYTE MDD_CFBT_MediaDetect();
```

## Description

Determines if a card is inserted

## Preconditions

None

## Returns

TRUE - Card present FALSE - Card absent

## Side Effects

None

## Remarks

None

# MDD_CFPMP_DATADIR Macro

C

```c
#define MDD_CFPMP_DATADIR TRISE
```

## Description

Defines the PMP data bus direction register

APIs > CF Physical Layer > Public Members > MDD_CFPMP_DATADIR Macro

# MDD_CFPMP_MediaDetect Function

**C**

```
BYTE MDD_CFPMP_MediaDetect();
```

## Description

Determines if a card is inserted

## Preconditions

None

## Returns

TRUE - Card present FALSE - Card absent

## Side Effects

None

## Remarks

None

APIs > CF Physical Layer > Public Members > MDD_CFPMP_MediaDetect Function

# MDD_CFread Macro

C

```
#define MDD_CFread MDD_CFBT_CFread
```

## Description

Function pointer to the CompactFlash Read Physical Layer function

APIs > CF Physical Layer > Public Members > MDD_CFread Macro

# MDD_CFwait Macro

C

```c
#define MDD_CFwait MDD_CFBT_CFwait
```

## Description

Function pointer to the CompactFlash Wait Physical Layer function

APIs > CF Physical Layer > Public Members > MDD_CFwait Macro

# MDD_CFwrite Macro

C

```
#define MDD_CFwrite MDD_CFBT_CFwrite
```

## Description

Function pointer to the CompactFlash Write Physical Layer function

APIs > CF Physical Layer > Public Members > MDD_CFwrite Macro

# ADDBL Macro

**C**

```
#define ADDBL LATA
```

## Description

The CF address bus output latch register (for PIC18)

APIs > CF Physical Layer > Public Members > ADDBL Macro

# ADDDIR Macro

C

```c
#define ADDDIR TRISA
```

## Description

The CF address bus TRIS register (for PIC18)

APIs > CF Physical Layer > Public Members > ADDDIR Macro

# ADDR0 Macro

C

```c
#define ADDR0 LATBbits.LATB15
```

## Description

The CF address bus bit 0 output latch definition (for PIC24/30/33/32)

APIs > CF Physical Layer > Public Members > ADDR0 Macro

# ADDR1 Macro

C

```c
#define ADDR1 LATBbits.LATB14
```

## Description

The CF address bus bit 1 output latch definition (for PIC24/30/33/32)

APIs > CF Physical Layer > Public Members > ADDR1 Macro

# ADDR2 Macro

C

```c
#define ADDR2 LATGbits.LATG9
```

## Description

The CF address bus bit 2 output latch definition (for PIC24/30/33/32)

APIs > CF Physical Layer > Public Members > ADDR2 Macro

# ADDR3 Macro

C

```c
#define ADDR3 LATGbits.LATG8
```

## Description

The CF address bus bit 3 output latch definition (for PIC24/30/33/32)

APIs > CF Physical Layer > Public Members > ADDR3 Macro

# ADRTRIS0 Macro

C

```
#define ADRTRIS0 TRISBbits.TRISB15
```

## Description

The CF address bus bit 0 TRIS definition (for PIC24/30/33/32)

APIs > CF Physical Layer > Public Members > ADRTRIS0 Macro

# ADRTRIS1 Macro

C

```
#define ADRTRIS1 TRISBbits.TRISB14
```

## Description

The CF address bus bit 1 TRIS definition (for PIC24/30/33/32)

APIs > CF Physical Layer > Public Members > ADRTRIS1 Macro

# ADRTRIS2 Macro

C

```c
#define ADRTRIS2 TRISGbits.TRISG9
```

## Description

The CF address bus bit 2 TRIS definition (for PIC24/30/33/32)

APIs > CF Physical Layer > Public Members > ADRTRIS2 Macro

# ADRTRIS3 Macro

C

```c
#define ADRTRIS3 TRISGbits.TRISG8
```

## Description

The CF address bus bit 3 TRIS definition (for PIC24/30/33/32)

APIs > CF Physical Layer > Public Members > ADRTRIS3 Macro

# CF_BT_CD1 Macro

C

```c
#define CF_BT_CD1 PORTCbits.RC4
```

## Description

The CF card detect signal port bit

APIs > CF Physical Layer > Public Members > CF_BT_CD1 Macro

# CF_BT_CD1DIR Macro

C

```c
#define CF_BT_CD1DIR TRISCbits.TRISC4
```

## Description

The CF card detect signal TRIS bit

APIs > CF Physical Layer > Public Members > CF_BT_CD1DIR Macro

# CF_BT_RDY Macro

C

```c
#define CF_BT_RDY PORTDbits.RD12
```

## Description

The CF card ready signal port bit

APIs > CF Physical Layer > Public Members > CF_BT_RDY Macro

# CF_BT_READYDIR Macro

C

```c
#define CF_BT_READYDIR TRISDbits.TRISD12
```

## Description

The CF card ready signal TRIS bit

APIs > CF Physical Layer > Public Members > CF_BT_READYDIR Macro

# CF_BT_RESETDIR Macro

C

```c
#define CF_BT_RESETDIR TRISDbits.TRISD0
```

## Description

The CF card reset signal TRIS bit

APIs > CF Physical Layer > Public Members > CF_BT_RESETDIR Macro

# CF_BT_RST Macro

C

```c
#define CF_BT_RST PORTDbits.RD0
```

## Description

The CF card reset signal latch bit

APIs > CF Physical Layer > Public Members > CF_BT_RST Macro

# CF_CE Macro

C

```c
#define CF_CE PORTDbits.RD11
```

## Description

The CF card chip select output latch bit

APIs > CF Physical Layer > Public Members > CF_CE Macro

# CF_CEDIR Macro

C

```
#define CF_CEDIR TRISDbits.TRISD11
```

## Description

The CF card chip select TRIS bit

APIs > CF Physical Layer > Public Members > CF_CEDIR Macro

# CF_OE Macro

C

```c
#define CF_OE PORTDbits.RD5
```

## Description

The CF card output enable strobe latch bit

APIs > CF Physical Layer > Public Members > CF_OE Macro

# CF_OEDIR Macro

C

```c
#define CF_OEDIR TRISDbits.TRISD5
```

## Description

The CF card output enable strobe TRIS bit

APIs > CF Physical Layer > Public Members > CF_OEDIR Macro

# CF_PMP_CD1 Macro

C

```c
#define CF_PMP_CD1 PORTCbits.RC4
```

## Description

The input port for the CF card detect signal

APIs > CF Physical Layer > Public Members > CF_PMP_CD1 Macro

# CF_PMP_CD1DIR Macro

C

```
#define CF_PMP_CD1DIR TRISCbits.TRISC4
```

## Description

The TRIS bit for the CF card detect signal

APIs > CF Physical Layer > Public Members > CF_PMP_CD1DIR Macro

# CF_PMP_RDY Macro

C

```c
#define CF_PMP_RDY PORTDbits.RD12
```

## Description

The input port for the CF Ready signal

APIs > CF Physical Layer > Public Members > CF_PMP_RDY Macro

# CF_PMP_READYDIR Macro

C

```
#define CF_PMP_READYDIR TRISDbits.TRISD12
```

## Description

The TRIS bit for the CF Ready signal

APIs > CF Physical Layer > Public Members > CF_PMP_READYDIR Macro

# CF_PMP_RESETDIR Macro

C

```c
#define CF_PMP_RESETDIR TRISDbits.TRISD0
```

## Description

The TRIS bit for the CF Reset signal

APIs > CF Physical Layer > Public Members > CF_PMP_RESETDIR Macro

# CF_PMP_RST Macro

C

```c
#define CF_PMP_RST PORTDbits.RD0
```

## Description

The output latch for the CF Reset signal

APIs > CF Physical Layer > Public Members > CF_PMP_RST Macro

# CF_WE Macro

C

```c
#define CF_WE PORTDbits.RD4
```

## Description

The CF card write enable strobe latch bit

APIs > CF Physical Layer > Public Members > CF_WE Macro

# CF_WEDIR Macro

C

```
#define CF_WEDIR TRISDbits.TRISD4
```

## Description

The CF card write enable strobe TRIS bit

APIs > CF Physical Layer > Public Members > CF_WEDIR Macro

# Library Members

The following functions, variables, structures, and macros are public, but are intended only to be accessed by the library itself. Applications should generally not call these functions or modify these variables.

## Functions

| | Name | Description |
|---|---|---|
| | MDD_CFBT_CFread | Reads a byte from the CF card |
| | MDD_CFBT_CFwait | Wait until the card is ready |
| | MDD_CFBT_CFwrite | Writes a byte to the CF card |
| | MDD_CFBT_InitIO | None |
| | 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_CFPMP_CFread | Reads a byte from the CF card |
| | MDD_CFPMP_CFwait | Wait until the card and PMP are ready |
| | MDD_CFPMP_CFwrite | Writes a byte to the CF card |
| | MDD_CFPMP_InitIO | None |
| | 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 |

## Macros

| | Name | Description |
|---|---|---|
| | MDD_CFBT_DATABinput | A macro to set the CF data bus TRIS register to inputs |

| | | |
|---|---|---|
| | MDD_CFBT_DATABoutput | A macro to set the CF data bus TRIS register to outputs |
| | MDD_CFBT_MediaInitialize | Prototypes |
| | MDD_CFPMP_DATABinput | A macro to set the CF data bus TRIS register to inputs |
| | MDD_CFPMP_DATABoutput | A macro to set the CF data bus TRIS register to outputs |
| | MDD_CFPMP_MediaInitialize | The initialization function for CF cards (no initialization required) |
| | R_CMD | A macro for the command register offset for CF cards |
| | R_COUNT | A macro for the count register offset for CF cards |
| | R_CYHI | A macro for the cylinder-high register offset for CF cards |
| | R_CYLO | A macro for the cylinder-low register offset for CF cards |
| | R_DATA | A macro for the data register offset for CF cards |
| | R_DRIVE | A macro for the drive register offset for CF cards |
| | R_ERROR | A macro for the error register offset for CF cards |
| | R_SECT | A macro for the sector register offset for CF cards |

| | | |
|---|---|---|
| | R_STATUS | A macro for the status offset for CF cards |
| | C_DRIVE_DIAG | A macro for the CF drive diagnostic command |
| | C_DRIVE_IDENT | A macro for the CF drive identify command |
| | C_SECTOR_READ | A macro for the CF read comment |
| | C_SECTOR_WRITE | A macro for the CF write command |
| | S_ERROR | A macro indicating that the CF status register reports an error condition |
| | S_READY | A macro indicating that the CF status register reports a ready condition |

# MDD_CFBT_CFread Function

C

```
BYTE MDD_CFBT_CFread(
    BYTE add
);
```

## Description

Reads a byte from the CF card

## Preconditions

None

## Parameters

| Parameters | Description |
|---|---|
| BYTE add | address to read from |

## Returns

BYTE - the byte read

## Side Effects

None

## Remarks

None

# MDD_CFBT_CFwait Function

C

```c
void MDD_CFBT_CFwait();
```

## Description

Wait until the card is ready

## Preconditions

None

## Returns

None

## Side Effects

None

## Remarks

None

APIs > CF Physical Layer > Library Members > MDD_CFBT_CFwait Function

# MDD_CFBT_CFwrite Function

C

```c
void MDD_CFBT_CFwrite(
    BYTE add,
    BYTE d
);
```

## Description

Writes a byte to the CF card

## Preconditions

None

## Parameters

| Parameters | Description |
|---|---|
| BYTE add | the address to write to |
| BYTE d | the byte to write |

## Returns

None

## Side Effects

None

## Remarks

None

# MDD_CFBT_DATABinput Macro

C

```c
#define MDD_CFBT_DATABinput MDD_CFBT_DATADIR = 0xff;
```

## Description

A macro to set the CF data bus TRIS register to inputs

APIs > CF Physical Layer > Library Members > MDD_CFBT_DATABinput Macro

# MDD_CFBT_DATABoutput Macro

C

```c
#define MDD_CFBT_DATABoutput MDD_CFBT_DATADIR = 0;
```

## Description

A macro to set the CF data bus TRIS register to outputs

APIs > CF Physical Layer > Library Members > MDD_CFBT_DATABoutput Macro

# MDD_CFBT_InitIO Function

```
C
void MDD_CFBT_InitIO();
```

## Description

None

## Preconditions

None

## Returns

void

## Side Effects

None

## Remarks

None

APIs > CF Physical Layer > Library Members > MDD_CFBT_InitIO Function

# MDD_CFBT_MediaInitialize Macro

**C**

```
#define MDD_CFBT_MediaInitialize TRUE
```

## Description

Prototypes

APIs > CF Physical Layer > Library Members > MDD_CFBT_MediaInitialize Macro

# MDD_CFBT_SectorRead Function

C

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

## 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'.

## 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 |

## Returns

TRUE - Sector read FALSE - Sector could not be read

## Side Effects

None

## Remarks

None

# MDD_CFBT_SectorWrite Function

C

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

## 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.

## 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 |

## Returns

TRUE - Sector written FALSE - Sector could not be written

## Side Effects

None

## Remarks

None

# MDD_CFBT_WriteProtectState Function

**C**

```
BYTE MDD_CFBT_WriteProtectState();
```

## Description

Added for compatibility- no write protect feature

## Preconditions

None

## Returns

0

## Side Effects

None

## Remarks

None

APIs > CF Physical Layer > Library Members > MDD_CFBT_WriteProtectState Function

# MDD_CFPMP_CFread Function

C

```
BYTE MDD_CFPMP_CFread(
    BYTE add
);
```

## Description

Reads a byte from the CF card

## Preconditions

None

## Parameters

| Parameters | Description |
|---|---|
| BYTE add | address to read from |

## Returns

BYTE - the byte read

## Side Effects

None

## Remarks

None

# MDD_CFPMP_CFwait Function

```
C
void MDD_CFPMP_CFwait();
```

## Description

Wait until the card and PMP are ready

## Preconditions

None

## Returns

None

## Side Effects

None

## Remarks

None

APIs > CF Physical Layer > Library Members > MDD_CFPMP_CFwait Function

# MDD_CFPMP_CFwrite Function

C

```c
void MDD_CFPMP_CFwrite(
    BYTE add,
    BYTE d
);
```

## Description

Writes a byte to the CF card

## Preconditions

None

## Parameters

| Parameters | Description |
|---|---|
| BYTE add | the address to write to |
| BYTE d | the byte to write |

## Returns

None

## Side Effects

None

## Remarks

None

# MDD_CFPMP_DATABinput Macro

C

```
#define MDD_CFPMP_DATABinput MDD_CFPMP_DATADIR = 0xf
```

## Description

A macro to set the CF data bus TRIS register to inputs

APIs > CF Physical Layer > Library Members > MDD_CFPMP_DATABinput Macro

# MDD_CFPMP_DATABoutput Macro

C

```c
#define MDD_CFPMP_DATABoutput MDD_CFPMP_DATADIR = 0;
```

## Description

A macro to set the CF data bus TRIS register to outputs

APIs > CF Physical Layer > Library Members > MDD_CFPMP_DATABoutput Macro

# MDD_CFPMP_InitIO Function

C

```c
void MDD_CFPMP_InitIO();
```

## Description

None

## Preconditions

None

## Returns

TRUE - Card initialized FALSE - Card not initialized

## Side Effects

None

## Remarks

None

APIs > CF Physical Layer > Library Members > MDD_CFPMP_InitIO Function

# MDD_CFPMP_MediaInitialize Macro

C

```
#define MDD_CFPMP_MediaInitialize TRUE
```

## Description

The initialization function for CF cards (no initialization required)

APIs > CF Physical Layer > Library Members > MDD_CFPMP_MediaInitialize Macro

# MDD_CFPMP_SectorRead Function

C

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

## 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'.

## Preconditions

None

## Parameters

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

## Returns

TRUE - Sector read FALSE - Sector could not be read

## Side Effects

None

## Remarks

None

# MDD_CFPMP_SectorWrite Function

C

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

## 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.

## 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 |

## Returns

TRUE - Sector written FALSE - Sector could not be written

## Side Effects

None

## Remarks

None

# MDD_CFPMP_WriteProtectState Function

**C**

```
BYTE MDD_CFPMP_WriteProtectState();
```

## Description

Added for compatibility- no write protect feature

## Preconditions

None

## Returns

0

## Side Effects

None

## Remarks

None

APIs > CF Physical Layer > Library Members > MDD_CFPMP_WriteProtectState Function

# R_CMD Macro

C

```c
#define R_CMD 7
```

## Description

A macro for the command register offset for CF cards

APIs > CF Physical Layer > Library Members > R_CMD Macro

# R_COUNT Macro

C

```c
#define R_COUNT 2
```

## Description

A macro for the count register offset for CF cards

APIs > CF Physical Layer > Library Members > R_COUNT Macro

# R_CYHI Macro

C

```c
#define R_CYHI 5
```

## Description

A macro for the cylinder-high register offset for CF cards

APIs > CF Physical Layer > Library Members > R_CYHI Macro

# R_CYLO Macro

C

```c
#define R_CYLO 4
```

## Description

A macro for the cylinder-low register offset for CF cards

APIs > CF Physical Layer > Library Members > R_CYLO Macro

# R_DATA Macro

C

```c
#define R_DATA 0
```

## Description

A macro for the data register offset for CF cards

APIs > CF Physical Layer > Library Members > R_DATA Macro

# R_DRIVE Macro

C

```c
#define R_DRIVE 6
```

## Description

A macro for the drive register offset for CF cards

APIs > CF Physical Layer > Library Members > R_DRIVE Macro

# R_ERROR Macro

C

```c
#define R_ERROR 1
```

## Description

A macro for the error register offset for CF cards

APIs > CF Physical Layer > Library Members > R_ERROR Macro

# R_SECT Macro

C

```
#define R_SECT 3
```

## Description

A macro for the sector register offset for CF cards

APIs > CF Physical Layer > Library Members > R_SECT Macro

# R_STATUS Macro

C

```
#define R_STATUS 7
```

## Description

A macro for the status offset for CF cards

APIs > CF Physical Layer > Library Members > R_STATUS Macro

# C_DRIVE_DIAG Macro

C

```
#define C_DRIVE_DIAG 0x90
```

## Description

A macro for the CF drive diagnostic command

APIs > CF Physical Layer > Library Members > C_DRIVE_DIAG Macro

# C_DRIVE_IDENT Macro

C

```
#define C_DRIVE_IDENT 0xEC
```

## Description

A macro for the CF drive identify command

APIs > CF Physical Layer > Library Members > C_DRIVE_IDENT Macro

# C_SECTOR_READ Macro

C

```
#define C_SECTOR_READ 0x20
```

## Description

A macro for the CF read comment

APIs > CF Physical Layer > Library Members > C_SECTOR_READ Macro

# C_SECTOR_WRITE Macro

C

```
#define C_SECTOR_WRITE 0x30
```

## Description

A macro for the CF write command

APIs > CF Physical Layer > Library Members > C_SECTOR_WRITE Macro

# S_ERROR Macro

C

```c
#define S_ERROR 0x51
```

## Description

A macro indicating that the CF status register reports an error condition

APIs > CF Physical Layer > Library Members > S_ERROR Macro

# S_READY Macro

C

```
#define S_READY 0x58
```

## Description

A macro indicating that the CF status register reports a ready condition

APIs > CF Physical Layer > Library Members > S_READY Macro

# Contents

FSerror Function
FSfclose Function
FSfeof Function
FSfopen Function
FSfopenpgm Function
FSformat Function
FSfprintf Function
FSfread Function
FSfseek Function
FSftell Function
FSfwrite Function
FSgetcwd Function
FSInit Function
FSmkdir Function
FSmkdirpgm Function
FSremove Function
FSremovepgm Function
FSrename Function
FSrenamepgm Function
FSrewind Function
FSrmdir Function
FSrmdirpgm Function
SetClockVars Function
ALLOW_DIRS Macro
ALLOW_FILESEARCH Macro
ALLOW_FSFPRINTF Macro
ALLOW_FORMATS Macro
ALLOW_PGMFUNCTIONS Macro
ALLOW_WRITES Macro
APPEND Macro
APPENDPLUS Macro
ATTR_ARCHIVE Macro
ATTR_DIRECTORY Macro

ATTR_HIDDEN Macro
ATTR_MASK Macro
ATTR_READ_ONLY Macro
ATTR_SYSTEM Macro
ATTR_VOLUME Macro
EOF Macro
FALSE Macro
FS_DYNAMIC_MEM Macro
FS_MAX_FILES_OPEN Macro
INCREMENTTIMESTAMP Macro
intmax_t Macro
MDD_MediaDetect Macro
MEDIA_SECTOR_SIZE Macro
NEAR_MODEL Macro
READ Macro
READPLUS Macro
SEEK_CUR Macro
SEEK_END Macro
SEEK_SET Macro
SUPPORT_FAT32 Macro
TRUE Macro
USE_CF_INTERFACE_WITH_PMP Macro
USE_MANUAL_CF_INTERFACE Macro
USE_SD_INTERFACE_WITH_SPI Macro
USE_USB_INTERFACE Macro
USERDEFINEDCLOCK Macro
USEREALTIMECLOCK Macro
WRITE Macro
WRITEPLUS Macro
FSFILE Structure
SearchRec Structure
**Library Members**
ReadByte Function

ReadDWord Function
ReadWord Function
ATTR_LONG_NAME Macro
BSI_BOOTSIG Macro
BSI_BPS Macro
BSI_FAT32_BOOTSIG Macro
BSI_FAT32_FSTYPE Macro
BSI_FATCOUNT Macro
BSI_FATSZ32 Macro
BSI_FSTYPE Macro
BSI_RESRVSEC Macro
BSI_ROOTCLUS Macro
BSI_ROOTDIRENTS Macro
BSI_SPC Macro
BSI_SPF Macro
BSI_TOTSEC16 Macro
BSI_TOTSEC32 Macro
CE_EOF Macro
CE_FAT_EOF Macro
CLUSTER_EMPTY Macro
CLUSTER_FAIL_FAT16 Macro
CLUSTER_FAIL_FAT32 Macro
DIR_DEL Macro
DIR_EMPTY Macro
DIR_EXTENSION Macro
DIR_NAMECOMP Macro
DIR_NAMESIZE Macro
END_CLUSTER_FAT12 Macro
END_CLUSTER_FAT16 Macro
END_CLUSTER_FAT32 Macro
FAT_GOOD_SIGN_0 Macro
FAT_GOOD_SIGN_1 Macro
FAT12 Macro

FAT16 Macro
FAT32 Macro
FILE_NAME_SIZE Macro
FO_MBR Macro
FOUND Macro
GetInstructionClock Macro
GetPeripheralClock Macro
GetSystemClock Macro
INPUT Macro
LAST_CLUSTER_FAT12 Macro
LAST_CLUSTER_FAT16 Macro
LAST_CLUSTER_FAT32 Macro
MASK_MAX_FILE_ENTRY_LIMIT_BITS Macro
MDD_InitIO Macro
MDD_MediaInitialize Macro
MDD_ReadCapacity Macro
MDD_ReadSectorSize Macro
MDD_SectorRead Macro
MDD_SectorWrite Macro
MDD_ShutdownMedia Macro
MDD_WriteProtectState Function
NO_MORE Macro
NOT_FOUND Macro
NUMBER_OF_BYTES_IN_DIR_ENTRY Macro
OUTPUT Macro
RAMread Macro
RAMreadD Macro
RAMreadW Macro
RAMwrite Macro
TOTAL_FILE_SIZE Macro
VALUE_BASED_ON_ENTRIES_PER_CLUSTER Macro
VALUE_DOTDOT_CLUSTER_VALUE_FOR_ROOT Macro
_BootSec Structure

_BPB_FAT12 Structure
_BPB_FAT16 Structure
_BPB_FAT32 Structure
_PT_MBR Structure
BootSec Type
CETYPE Enumeration
DISK Structure
FILEFLAGS Structure
PT_MBR Type
PTE_MBR Structure
SALLOC Type
SEARCH_TYPE Enumeration

**Internal Members**

_SRAMmerge Function
Cache_File_Entry Function
CacheTime Function
chdirhelper Function
Cluster2Sector Function
CreateDIR Function
CreateFileEntry Function
CreateFirstCluster Function
DISKmount Function
EraseCluster Function
FAT_erase_cluster_chain Function
FATfindEmptyCluster Function
FILEallocate_new_cluster Function
FILECreateHeadCluster Function
FILEerase Function
FILEfind Function
FILEget_next_cluster Function
FileObjectCopy Function
FILEopen Function
Fill_File_Object Function

FindEmptyEntries Function
flushData Function
FormatDirName Function
FormatFileName Function
FSputc Function
FSvfprintf Function
GetFullClusterNumber Function
GetPreviousEntry Function
IncrementTimeStamp Function
LoadBootSector Function
LoadDirAttrib Function
LoadMBR Function
mkdirhelper Function
PopulateEntries Function
ReadFAT Function
rmdirhelper Function
SRAMInitHeap Function
str_put_n_chars Function
ValidateChars Function
Write_File_Entry Function
writeDotEntries Function
WriteFAT Function
_FLAG_MINUS Macro
_FLAG_OCTO Macro
_FLAG_PLUS Macro
_FLAG_SIGNED Macro
_FLAG_SPACE Macro
_FLAG_ZERO Macro
_FMT_BYTE Macro
_FMT_LONG Macro
_FMT_LONGLONG Macro
_FMT_SHRTLONG Macro
_FMT_UNSPECIFIED Macro

_MAX_HEAP_SIZE Macro
_MAX_SEGMENT_SIZE Macro
DIRECTORY Macro
DIRENTRIES_PER_SECTOR Macro
NEAR Macro
_DIRENTRY Structure
DIRENTRY Type
FILEOBJ Type
_uDynamicHeap Variable
cwd Variable
cwdptr Variable
defaultArray Variable
defaultString Variable
dirCleared Variable
FatRootDirClusterValue Variable
FSerrno Variable
gBufferOwner Variable
gBufferZeroed Variable
gDataBuffer Variable
gDiskData Variable
gFATBuffer Variable
gFileArray Variable
gFileSlotOpen Variable
gFileTemp Variable
gLastDataSectorRead Variable
gLastFATSectorRead Variable
gNeedDataWrite Variable
gNeedFATWrite Variable
gTimeAccDate Variable
gTimeCrtDate Variable
gTimeCrtMS Variable
gTimeCrtTime Variable
gTimeWrtDate Variable

gTimeWrtTime Variable
nextClusterIsLast Variable
recache Variable
s_digits Variable
tempArray Variable
TempClusterCalc Variable
tempCWDobj Variable

**SD-SPI Physical Layer**

**Public Members**

SD_CD Macro
SD_CD_TRIS Macro
SD_CS Macro
SD_CS_TRIS Macro
SD_WE Macro
SD_WE_TRIS Macro
SPI_INTERRUPT_FLAG Macro
SPIBRG Macro
SPIBUF Macro
SPICLOCK Macro
SPICLOCKPORT Macro
SPICLOCKLAT Macro
SPICON1 Macro
SPICON1bits Macro
SPIENABLE Macro
SPIIN Macro
SPIINPORT Macro
SPIINLAT Macro
SPIOUT Macro
SPIOUTPORT Macro
SPIOUTLAT Macro
SPISTAT Macro
SPISTAT_RBF Macro
SPISTATbits Macro

**Library Members**

MDD_SDSPI_InitIO Function
MDD_SDSPI_MediaDetect Function
MDD_SDSPI_MediaInitialize Function
MDD_SDSPI_ReadCapacity Function
MDD_SDSPI_ReadMedia Function
MDD_SDSPI_ReadSectorSize Function
MDD_SDSPI_SectorWrite Function
MDD_SDSPI_SectorRead Function
MDD_SDSPI_ShutdownMedia Function
cmdAPP_CMD Macro
cmdCRC_ON_OFF Macro
cmdERASE Macro
cmdGO_IDLE_STATE Macro
cmdREAD_MULTI_BLOCK Macro
cmdREAD_OCR Macro
cmdREAD_SINGLE_BLOCK Macro
cmdSEND_CID Macro
cmdSEND_CSD Macro
cmdSEND_OP_COND Macro
cmdSEND_STATUS Macro
cmdSET_BLOCKLEN Macro
cmdSTOP_TRANSMISSION Macro
cmdTAG_SECTOR_END Macro
cmdTAG_SECTOR_START Macro
cmdWRITE_MULTI_BLOCK Macro
cmdWRITE_SINGLE_BLOCK Macro
DATA_ACCEPTED Macro
DATA_START_TOKEN Macro
DELAY_OVERHEAD Macro
DELAY_PRESCALER Macro
MASTER_ENABLE_ON Macro
MILLISECDELAY Macro

MMC_BAD_RESPONSE Macro
MMC_FLOATING_BUS Macro
MOREDATA Macro
mReadCRC Macro
mSend8ClkCycles Macro
mSendCRC Macro
NODATA Macro
PRI_PRESCAL_1_1 Macro
SEC_PRESCAL_1_1 Macro
SYNC_MODE_FAST Macro
SYNC_MODE_MED Macro
SYNC_MODE_SLOW Macro
CID Union
CMD_PACKET Union
CSD Union
RESPONSE_1 Union
RESPONSE_2 Union
MMC_RESPONSE Union
RESP Enumeration
sdmmc_cmd Enumeration
typMMC_CMD Structure

**Internal Members**

Delayms Function
CloseSPIM Function
OpenSPIM Function
ReadMediaManual Function
SendMMCCmd Function
SendMMCCmdManual Function
WriteSPIM Function
WriteSPIManual Function
MANUAL_SPI_CLOCK_VALUE Macro
sdmmc_cmdtable Variable
MDD_SDSPI_finalLBA Variable

**CF Physical Layer**

**Public Members**

MDD_CFBT_DATABIN Macro
MDD_CFBT_DATABOUT Macro
MDD_CFBT_DATADIR Macro
MDD_CFBT_MediaDetect Function
MDD_CFPMP_DATADIR Macro
MDD_CFPMP_MediaDetect Function
MDD_CFread Macro
MDD_CFwait Macro
MDD_CFwrite Macro
ADDBL Macro
ADDDIR Macro
ADDR0 Macro
ADDR1 Macro
ADDR2 Macro
ADDR3 Macro
ADRTRIS0 Macro
ADRTRIS1 Macro
ADRTRIS2 Macro
ADRTRIS3 Macro
CF_BT_CD1 Macro
CF_BT_CD1DIR Macro
CF_BT_RDY Macro
CF_BT_READYDIR Macro
CF_BT_RESETDIR Macro
CF_BT_RST Macro
CF_CE Macro
CF_CEDIR Macro
CF_OE Macro
CF_OEDIR Macro
CF_PMP_CD1 Macro
CF_PMP_CD1DIR Macro

CF_PMP_RDY Macro
CF_PMP_READYDIR Macro
CF_PMP_RESETDIR Macro
CF_PMP_RST Macro
CF_WE Macro
CF_WEDIR Macro

**Library Members**

MDD_CFBT_CFread Function
MDD_CFBT_CFwait Function
MDD_CFBT_CFwrite Function
MDD_CFBT_DATABinput Macro
MDD_CFBT_DATABoutput Macro
MDD_CFBT_InitIO Function
MDD_CFBT_MediaInitialize Macro
MDD_CFBT_SectorRead Function
MDD_CFBT_SectorWrite Function
MDD_CFBT_WriteProtectState Function
MDD_CFPMP_CFread Function
MDD_CFPMP_CFwait Function
MDD_CFPMP_CFwrite Function
MDD_CFPMP_DATABinput Macro
MDD_CFPMP_DATABoutput Macro
MDD_CFPMP_InitIO Function
MDD_CFPMP_MediaInitialize Macro
MDD_CFPMP_SectorRead Function
MDD_CFPMP_SectorWrite Function
MDD_CFPMP_WriteProtectState Function
R_CMD Macro
R_COUNT Macro
R_CYHI Macro
R_CYLO Macro
R_DATA Macro
R_DRIVE Macro

R_ERROR Macro
R_SECT Macro
R_STATUS Macro
C_DRIVE_DIAG Macro
C_DRIVE_IDENT Macro
C_SECTOR_READ Macro
C_SECTOR_WRITE Macro
S_ERROR Macro
S_READY Macro

---

# Index

## B



## H



## I



## L

## C



## M

## N



## O



## P



## R

## D



## S

## E



## F

## T



## U



## V



## W

# MAX_HEAP_SIZE Macro

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

# SWORD Structure

C

```c
typedef struct {
  unsigned char array[3];
} SWORD;
```

## Description

The SWORD macro is used to defined a 24-bit data type. For 16+ bit architectures, this must be represented as an array of three bytes.

APIs > File Manipulation Layer (FSIO) > Library Members > SWORD Structure

# eraseDir Function

C

```c
int eraseDir(
    char * path
);
```

## Description

The eraseDir function is a helper function for the rmdirhelper function. The eraseDir function will search for the directory that matches the specified path name and then erase it with the FILEerase function.

## Preconditions

This function should not be called by the user.

## Parameters

| Parameters | Description |
|---|---|
| path | The name of the directory to delete |

## Return Values

| Return Values | Description |
|---|---|
| 0 | Dir was deleted successfully |
| -1 | Dir could not be deleted. |

## Side Effects

None

## Remarks

None.

# MDD_SDSPI_WriteProtectState Function

**C**

```
BYTE MDD_SDSPI_WriteProtectState();
```

## Description

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

## Preconditions

The MDD_WriteProtectState function pointer must be pointing to this function.

## Return Values

| Return Values | Description |
|---|---|
| TRUE | The card is write-protected |
| FALSE | The card is not write-protected |

## Side Effects

None.

## Remarks

None