Arduino SdFat Library

Introduction

The Arduino SdFat Library is a minimal implementation of FAT16 and FAT32 file systems on SD flash memory cards. Standard SD and high capacity SDHC cards are supported.

The SdFat only supports short 8.3 names.

The main classes in SdFat are **Sd2Card**, **SdVolume**, and **SdFile**.

The **Sd2Card** class supports access to standard SD cards and SDHC cards. Most applications will only need to call the **Sd2Card::init()** member function.

The **SdVolume** class supports FAT16 and FAT32 partitions. Most applications will only need to call the **SdVolume::init()** member function.

The **SdFile** class provides file access functions such as open(), read(), remove(), write(), close() and sync(). This class supports access to the root directory and subdirectories.

A number of example are provided in the SdFat/examples folder. These were developed to test SdFat and illustrate its use.

SdFat was developed for high speed data recording. SdFat was used to implement an audio record/play class, WaveRP, for the Adafruit Wave Shield. This application uses special **Sd2Card** calls to write to contiguous files in raw mode. These functions reduce write latency so that audio can be recorded with the small amount of RAM in the Arduino.

SD\SDHC Cards

Arduinos access SD cards using the cards SPI protocol. PCs, Macs, and most consumer devices use the 4-bit parallel SD protocol. A card that functions well on A PC or Mac may not work well on the Arduino.

Most cards have good SPI read performance but cards vary widely in SPI write performance. Write performance is limited by how efficiently the card manages internal erase/remapping operations. The Arduino cannot optimize writes to reduce erase operations because of its limit RAM.

SanDisk cards generally have good write performance. They seem to have more internal RAM buffering than other cards and therefore can limit the number of flash erase operations that the Arduino forces due to its limited RAM.

Hardware Configuration

SdFat was developed using an `Adafruit Industries Wave Shield`.

The hardware interface to the SD card should not use a resistor based level shifter. SdFat sets the SPI bus frequency to 8 MHz which results in signal rise times that are too slow for the edge detectors in many newer SD card controllers when resistor voltage dividers are used.

The 5 to 3.3 V level shifter for 5 V Arduinos should be IC based like the 74HC4050N based circuit shown in the file SdLevel.png. The Adafruit Wave Shield uses a 74AHC125N. Gravitech sells SD and MicroSD Card Adapters based on the 74LCX245.

If you are using a resistor based level shifter and are having problems try setting the SPI bus frequency to 4 MHz. This can be done by using card.init(SPI_HALF_SPEED) to initialize the SD card.

Bugs and Comments

If you wish to report bugs or have comments, send email to `fat16lib@sbcglobal.net`.

SdFat Usage

SdFat uses a slightly restricted form of short names. Only printable ASCII characters are supported. No characters with code point values greater than 127 are allowed. Space is not allowed even though space was allowed in the API of early versions of DOS.

Short names are limited to 8 characters followed by an optional period (.) and extension of up to 3 characters. The characters may be any combination of letters and digits. The following special characters are also allowed:

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

Short names are always converted to upper case and their original case value is lost.

Note:

The Arduino **Print** class uses character at a time writes so it was necessary to use a **sync()** function to control when data is written to the SD card.

An application which writes to a file using **print()**, **println()** or **write()** must call **sync()** at the appropriate time to force data and directory information to be written to the SD Card. Data and directory information are also written to the SD card when **close()** is called.

Applications must use care calling **sync()** since 2048 bytes of I/O is required to update file and directory information. This includes writing the current data block, reading the block that contains the directory entry for update, writing the directory block back and reading back the current data block.

It is possible to open a file with two or more instances of **SdFile**. A file may be corrupted if data is written to the file by more than one instance of **SdFile**.

How to format SD Cards as FAT Volumes

You should use a freshly formatted SD card for best performance. FAT file systems become slower if many files have been created and deleted. This is because the directory entry for a deleted file is marked as deleted, but is not deleted. When a new file is created, these entries must be scanned before creating the file, a flaw in the FAT design. Also files can become fragmented which causes reads and writes to be slower.

Microsoft operating systems support removable media formatted with a Master Boot Record, MBR, or formatted as a super floppy with a FAT Boot Sector in block zero.

Microsoft operating systems expect MBR formatted removable media to have only one partition. The first partition should be used.

Microsoft operating systems do not support partitioning SD flash cards. If you erase an SD card with a program like KillDisk, Most versions of Windows will format the card as a super floppy.

The best way to restore an SD card's format is to use SDFormatter which can be downloaded from:

`http://www.sdcard.org/consumers/formatter/`

SDFormatter aligns flash erase boundaries with file system structures which reduces write latency and file system overhead.

SDFormatter does not have an option for FAT type so it may format small cards as FAT12.

After the MBR is restored by SDFormatter you may need to reformat small cards that have been formatted FAT12 to force the volume type to be FAT16.

If you reformat the SD card with an OS utility, choose a cluster size that will result in:

4084 < CountOfClusters && CountOfClusters < 65525

The volume will then be FAT16.

If you are formatting an SD card on OS X or Linux, be sure to use the first partition. Format this partition with a cluster count in above range.

References

Adafruit Industries:

`http://www.adafruit.com/`

`http://www.ladyada.net/make/waveshield/`

The Arduino site:

`http://www.arduino.cc/`

For more information about FAT file systems see:

`http://www.microsoft.com/whdc/system/platform/firmware/fatgen.mspx`

For information about using SD cards as SPI devices see:

`http://www.sdcard.org/developers/tech/sdcard/pls/Simplified_Physical_Layer_Spec.pdf`

The ATmega328 datasheet:

`http://www.atmel.com/dyn/resources/prod_documents/doc8161.pdf`

Deprecated List

Member SdFile::contiguousRange (uint32_t &bgnBlock, uint32_t &endBlock)
Use: uint8_t **SdFile::contiguousRange(uint32_t* bgnBlock, uint32_t* endBlock)**;

Member SdFile::createContiguous (SdFile &dirFile, const char *fileName, uint32_t size)
Use: uint8_t **SdFile::createContiguous**(SdFile* dirFile, const char* fileName, uint32_t size)

Member SdFile::dateTimeCallback (void(*dateTime)(uint16_t &date, uint16_t &time))
Use: static void **SdFile::dateTimeCallback**(void (*dateTime)(uint16_t* date, uint16_t* time));

Member SdFile::dirEntry (dir_t &dir)
Use: uint8_t **SdFile::dirEntry(dir_t* dir)**;

Member SdFile::makeDir (SdFile &dir, const char *dirName)
Use: uint8_t **SdFile::makeDir(SdFile* dir, const char* dirName)**;

Member SdFile::open (SdFile &dirFile, uint16_t index, uint8_t oflag)
Use: uint8_t **SdFile::open(SdFile* dirFile, uint16_t index, uint8_t oflag)**;
Do not use in new apps
Use: uint8_t **SdFile::open(SdFile* dirFile, const char* fileName, uint8_t oflag)**;

Member SdFile::openRoot (SdVolume &vol)
Use: uint8_t **SdFile::openRoot(SdVolume* vol)**;

Member SdFile::readDir (dir_t &dir)
Use: int8_t **SdFile::readDir(dir_t* dir)**;

Member SdFile::remove (SdFile &dirFile, const char *fileName)
Use: static uint8_t **SdFile::remove(SdFile* dirFile, const char* fileName)**;

Member SdVolume::init (Sd2Card &dev, uint8_t part)
Use: uint8_t **SdVolume::init(Sd2Card* dev, uint8_t vol)**;
Use: uint8_t **SdVolume::init(Sd2Card* dev)**

Sd2Card Class Reference

Raw access to SD and SDHC flash memory cards.
`#include <Sd2Card.h>`

Public Member Functions

- uint32_t **cardSize** (void)
- uint8_t **erase** (uint32_t firstBlock, uint32_t lastBlock)
- uint8_t **eraseSingleBlockEnable** (void)
- uint8_t **errorCode** (void) const
- uint8_t **errorData** (void) const
- uint8_t **init** (uint8_t sckRateID, uint8_t chipSelectPin)
- uint8_t **init** (uint8_t sckRateID)
- uint8_t **init** (void)
- uint8_t **partialBlockRead** (void) const
- void **partialBlockRead** (uint8_t value)
- uint8_t **readBlock** (uint32_t block, uint8_t *dst)
- uint8_t **readCID** (cid_t *cid)
- uint8_t **readCSD** (csd_t *csd)
- uint8_t **readData** (uint32_t block, uint16_t offset, uint16_t count, uint8_t *dst)
- void **readEnd** (void)
- **Sd2Card** (void)
- uint8_t **setSckRate** (uint8_t sckRateID)
- uint8_t **type** (void) const
- uint8_t **writeBlock** (uint32_t blockNumber, const uint8_t *src)
- uint8_t **writeData** (const uint8_t *src)
- uint8_t **writeStart** (uint32_t blockNumber, uint32_t eraseCount)
- uint8_t **writeStop** (void)

Detailed Description

Raw access to SD and SDHC flash memory cards.

Constructor & Destructor Documentation

Sd2Card::Sd2Card (void) `[inline]`

Construct an instance of **Sd2Card**.

Member Function Documentation

uint32_t Sd2Card::cardSize (void)

Determine the size of an SD flash memory card.

Returns:

The number of 512 byte data blocks in the card or zero if an error occurs.

uint8_t Sd2Card::erase (uint32_t *firstBlock*, uint32_t *lastBlock*)

Erase a range of blocks.

Parameters:

[in] *firstBlock* The address of the first block in the range.
[in] *lastBlock* The address of the last block in the range.

Note:

This function requests the SD card to do a flash erase for a range of blocks. The data on the card after an erase operation is either 0 or 1, depends on the card vendor. The card must support single block erase.

Returns:

The value one, true, is returned for success and the value zero, false, is returned for failure.

uint8_t Sd2Card::eraseSingleBlockEnable (void)

Determine if card supports single block erase.

Returns:

The value one, true, is returned if single block erase is supported. The value zero, false, is returned if single block erase is not supported.

uint8_t Sd2Card::errorCode (void) const `[inline]`

Returns:

error code for last error. See **Sd2Card.h** for a list of error codes.

uint8_t Sd2Card::errorData (void) const `[inline]`

Returns:

error data for last error.

uint8_t Sd2Card::init (uint8_t *sckRateID*, uint8_t *chipSelectPin*)

Initialize an SD flash memory card.

Parameters:

[in] *sckRateID* SPI clock rate selector. See **setSckRate()**.
[in] *chipSelectPin* SD chip select pin number.

Returns:

The value one, true, is returned for success and the value zero, false, is returned for failure. The reason for failure can be determined by calling **errorCode()** and **errorData()**.

uint8_t Sd2Card::init (uint8_t *sckRateID*) `[inline]`

Initialize an SD flash memory card with the selected SPI clock rate and the default SD chip select pin. See sd2Card::init(uint8_t sckRateID, uint8_t chipSelectPin).

uint8_t Sd2Card::init (void) `[inline]`

Initialize an SD flash memory card with default clock rate and chip select pin. See sd2Card::init(uint8_t sckRateID, uint8_t chipSelectPin).

uint8_t Sd2Card::partialBlockRead (void) const [inline]

Returns the current value, true or false, for partial block read.

void Sd2Card::partialBlockRead (uint8_t *value*)

Enable or disable partial block reads.

Enabling partial block reads improves performance by allowing a block to be read over the SPI bus as several sub-blocks. Errors may occur if the time between reads is too long since the SD card may timeout. The SPI SS line will be held low until the entire block is read or **readEnd()** is called.

Use this for applications like the Adafruit Wave Shield.

Parameters:

[in] *value* The value TRUE (non-zero) or FALSE (zero).)

uint8_t Sd2Card::readBlock (uint32_t *block*, uint8_t * *dst*)

Read a 512 byte block from an SD card device.

Parameters:

[in] *block* Logical block to be read.
[out] *dst* Pointer to the location that will receive the data.

Returns:

The value one, true, is returned for success and the value zero, false, is returned for failure.

uint8_t Sd2Card::readCID (cid_t * *cid*) [inline]

Read a cards CID register. The CID contains card identification information such as Manufacturer ID, Product name, Product serial number and Manufacturing date.

uint8_t Sd2Card::readCSD (csd_t * *csd*) [inline]

Read a cards CSD register. The CSD contains Card-Specific Data that provides information regarding access to the card's contents.

uint8_t Sd2Card::readData (uint32_t *block*, uint16_t *offset*, uint16_t *count*, uint8_t * *dst*)

Read part of a 512 byte block from an SD card.

Parameters:

[in] *block* Logical block to be read.
[in] *offset* Number of bytes to skip at start of block
[out] *dst* Pointer to the location that will receive the data.
[in] *count* Number of bytes to read

Returns:

The value one, true, is returned for success and the value zero, false, is returned for failure.

void Sd2Card::readEnd (void)

Skip remaining data in a block when in partial block read mode.

uint8_t Sd2Card::setSckRate (uint8_t *sckRateID*)

Set the SPI clock rate.

Parameters:

[in] *sckRateID* A value in the range [0, 6].

The SPI clock will be set to F_CPU/pow(2, 1 + sckRateID). The maximum SPI rate is F_CPU/2 for *sckRateID* = 0 and the minimum rate is F_CPU/128 for *scsRateID* = 6.

Returns:

The value one, true, is returned for success and the value zero, false, is returned for an invalid value of *sckRateID* .

uint8_t Sd2Card::type (void) const `[inline]`

Return the card type: SD V1, SD V2 or SDHC

uint8_t Sd2Card::writeBlock (uint32_t *blockNumber*, const uint8_t * *src*)

Writes a 512 byte block to an SD card.

Parameters:

[in] *blockNumber* Logical block to be written.
[in] *src* Pointer to the location of the data to be written.

Returns:

The value one, true, is returned for success and the value zero, false, is returned for failure.

uint8_t Sd2Card::writeData (const uint8_t * *src*)

Write one data block in a multiple block write sequence

uint8_t Sd2Card::writeStart (uint32_t *blockNumber*, uint32_t *eraseCount*)

Start a write multiple blocks sequence.

Parameters:

[in] *blockNumber* Address of first block in sequence.
[in] *eraseCount* The number of blocks to be pre-erased.

Note:

This function is used with **writeData()** and **writeStop()** for optimized multiple block writes.

Returns:

The value one, true, is returned for success and the value zero, false, is returned for failure.

uint8_t Sd2Card::writeStop (void)

End a write multiple blocks sequence.

Returns:

The value one, true, is returned for success and the value zero, false, is returned for failure.

The documentation for this class was generated from the following files:

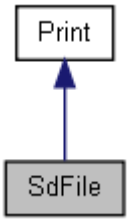
- Arduino/libraries/SdFat/**Sd2Card.h**
- Arduino/libraries/SdFat/Sd2Card.cpp

SdFile Class Reference

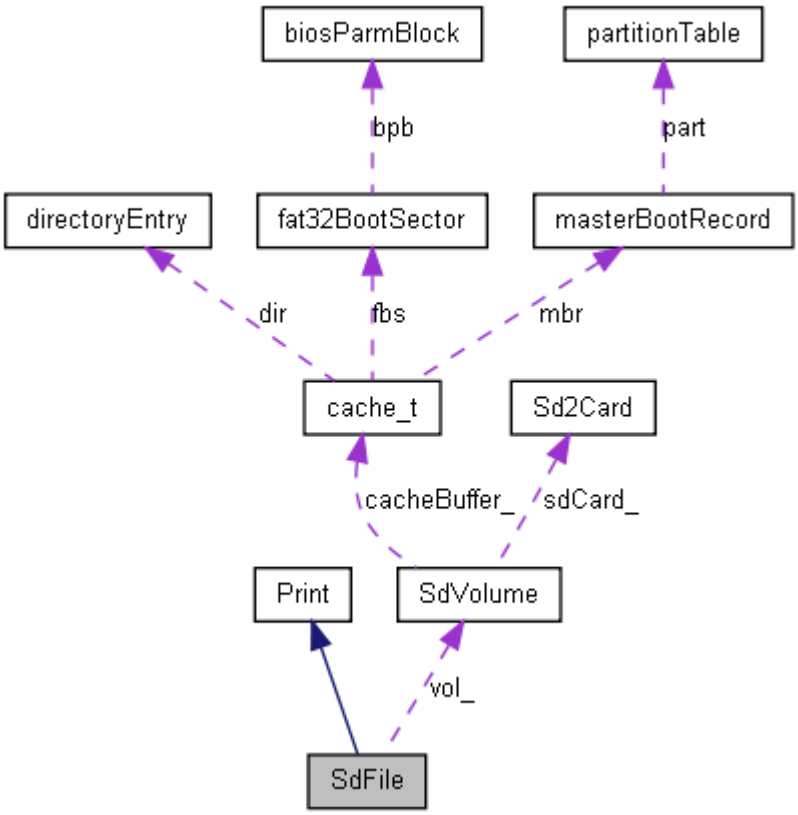
Access FAT16 and FAT32 files on SD and SDHC cards.

`#include <SdFat.h>`

Inheritance diagram for SdFile:



Collaboration diagram for SdFile:



Public Member Functions

- void **clearUnbufferedRead** (void)
- uint8_t **close** (void)
- uint8_t **contiguousRange** (uint32_t &bgnBlock, uint32_t &endBlock)
- uint8_t **contiguousRange** (uint32_t *bgnBlock, uint32_t *endBlock)
- uint8_t **createContiguous** (**SdFile** &dirFile, const char *fileName, uint32_t size)
- uint8_t **createContiguous** (**SdFile** *dirFile, const char *fileName, uint32_t size)
- uint32_t **curCluster** (void) const

- uint32_t **curPosition** (void) const
- uint32_t **dirBlock** (void) const
- uint8_t **dirEntry** (**dir_t** &dir)
- uint8_t **dirEntry** (**dir_t** *dir)
- uint8_t **dirIndex** (void) const
- uint32_t **fileSize** (void) const
- uint32_t **firstCluster** (void) const
- uint8_t **isDir** (void) const
- uint8_t **isFile** (void) const
- uint8_t **isOpen** (void) const
- uint8_t **isRoot** (void) const
- uint8_t **isSubDir** (void) const
- void **ls** (uint8_t flags=0, uint8_t indent=0)
- uint8_t **makeDir** (**SdFile** &dir, const char *dirName)
- uint8_t **makeDir** (**SdFile** *dir, const char *dirName)
- uint8_t **open** (**SdFile** &dirFile, uint16_t index, uint8_t oflag)
- uint8_t **open** (**SdFile** &dirFile, const char *fileName)
- uint8_t **open** (**SdFile** &dirFile, const char *fileName, uint8_t oflag)
- uint8_t **open** (**SdFile** *dirFile, const char *fileName, uint8_t oflag)
- uint8_t **open** (**SdFile** *dirFile, uint16_t index, uint8_t oflag)
- uint8_t **openRoot** (**SdVolume** &vol)
- uint8_t **openRoot** (**SdVolume** *vol)
- int16_t **read** (void *buf, uint16_t nbyte)
- int16_t **read** (void)
- int8_t **readDir** (**dir_t** &dir)
- int8_t **readDir** (**dir_t** *dir)
- uint8_t **remove** (void)
- void **rewind** (void)
- uint8_t **rmDir** (void)
- uint8_t **rmRfStar** (void)
- **SdFile** (void)
- uint8_t **seekCur** (uint32_t pos)
- uint8_t **seekEnd** (void)
- uint8_t **seekSet** (uint32_t pos)
- void **setUnbufferedRead** (void)
- uint8_t **sync** (void)
- uint8_t **timestamp** (uint8_t flag, uint16_t year, uint8_t month, uint8_t day, uint8_t hour, uint8_t minute, uint8_t second)
- uint8_t **truncate** (uint32_t size)
- uint8_t **type** (void) const
- uint8_t **unbufferedRead** (void) const
- **SdVolume** * **volume** (void) const
- void **write** (const char *str)
- int16_t **write** (const void *buf, uint16_t nbyte)
- void **write** (uint8_t b)
- void **write_P** (PGM_P str)
- void **writeln_P** (PGM_P str)

Static Public Member Functions

- static void **dateTimeCallback** (void(*dateTime)(uint16_t &date, uint16_t &time))
- static void **dateTimeCallback** (void(*dateTime)(uint16_t *date, uint16_t *time))
- static void **dateTimeCallbackCancel** (void)

- static void **dirName** (const **dir_t** &dir, char *name)
- static void **printDirName** (const **dir_t** &dir, uint8_t width)
- static void **printFatDate** (uint16_t fatDate)
- static void **printFatTime** (uint16_t fatTime)
- static void **printTwoDigits** (uint8_t v)
- static uint8_t **remove** (**SdFile** &dirFile, const char *fileName)
- static uint8_t **remove** (**SdFile** *dirFile, const char *fileName)

Public Attributes

- bool **writeError**

Detailed Description

Access FAT16 and FAT32 files on SD and SDHC cards.

Constructor & Destructor Documentation

SdFile::SdFile (void) `[inline]`

Create an instance of **SdFile**.

Member Function Documentation

void SdFile::clearUnbufferedRead (void) `[inline]`

Cancel unbuffered reads for this file. See **setUnbufferedRead()**

uint8_t SdFile::close (void)

Close a file and force cached data and directory information to be written to the storage device.

Returns:

The value one, true, is returned for success and the value zero, false, is returned for failure. Reasons for failure include no file is open or an I/O error.

uint8_t SdFile::contiguousRange (uint32_t & *bgnBlock*, uint32_t & *endBlock*) `[inline]`

Deprecated:

Use: uint8_t **SdFile::contiguousRange(uint32_t* bgnBlock, uint32_t* endBlock)**;

uint8_t SdFile::contiguousRange (uint32_t * *bgnBlock*, uint32_t * *endBlock*)

Check for contiguous file and return its raw block range.

Parameters:

[out] *bgnBlock* the first block address for the file.
[out] *endBlock* the last block address for the file.

Returns:

The value one, true, is returned for success and the value zero, false, is returned for failure. Reasons for failure include file is not contiguous, file has zero length or an I/O error occurred.

uint8_t SdFile::createContiguous (SdFile & *dirFile*, const char * *fileName*, uint32_t *size*) [inline]

Deprecated:

Use: uint8_t **SdFile::createContiguous**(SdFile* dirFile, const char* fileName, uint32_t size)

uint8_t SdFile::createContiguous (SdFile * *dirFile*, const char * *fileName*, uint32_t *size*)

Create and open a new contiguous file of a specified size.

Note:

This function only supports short DOS 8.3 names. See **open()** for more information.

Parameters:

[in] *dirFile* The directory where the file will be created.
[in] *fileName* A valid DOS 8.3 file name.
[in] *size* The desired file size.

Returns:

The value one, true, is returned for success and the value zero, false, is returned for failure. Reasons for failure include *fileName* contains an invalid DOS 8.3 file name, the FAT volume has not been initialized, a file is already open, the file already exists, the root directory is full or an I/O error.

uint32_t SdFile::curCluster (void) const [inline]

Returns:

The current cluster number for a file or directory.

uint32_t SdFile::curPosition (void) const [inline]

Returns:

The current position for a file or directory.

static void SdFile::dateTimeCallback (void(*)(uint16_t &date, uint16_t &time) *dateTime*) [inline, static]

Deprecated:

Use: static void **SdFile::dateTimeCallback**(void (*dateTime)(uint16_t* date, uint16_t* time));

static void SdFile::dateTimeCallback (void(*)(uint16_t *date, uint16_t *time) *dateTime*) [inline, static]

Set the date/time callback function

Parameters:

[in] *dateTime* The user's call back function. The callback function is of the form:

```
void dateTime(uint16 t* date, uint16 t* time) {
  uint16_t year;
  uint8 t month, day, hour, minute, second;

  // User gets date and time from GPS or real-time clock here

  // return date using FAT DATE macro to format fields
  *date = FAT_DATE(year, month, day);

  // return time using FAT_TIME macro to format fields
  *time = FAT_TIME(hour, minute, second);
```

```
}
```

Sets the function that is called when a file is created or when a file's directory entry is modified by **sync()**. All timestamps, access, creation, and modify, are set when a file is created. **sync()** maintains the last access date and last modify date/time.

See the **timestamp()** function.

static void SdFile::dateTimeCallbackCancel (void) [inline, static]

Cancel the date/time callback function.

uint32_t SdFile::dirBlock (void) const [inline]

Returns:

Address of the block that contains this file's directory.

uint8_t SdFile::dirEntry (dir_t & *dir*) [inline]

Deprecated:

Use: uint8_t **SdFile::dirEntry(dir_t* dir)**;

uint8_t SdFile::dirEntry (dir_t * *dir*)

Return a files directory entry

Parameters:

[out] *dir* Location for return of the files directory entry.

Returns:

The value one, true, is returned for success and the value zero, false, is returned for failure.

uint8_t SdFile::dirIndex (void) const [inline]

Returns:

Index of this file's directory in the block dirBlock.

void SdFile::dirName (const dir_t & *dir*, char * *name*) [static]

Format the name field of *dir* into the 13 byte array *name* in standard 8.3 short name format.

Parameters:

[in] *dir* The directory structure containing the name.
[out] *name* A 13 byte char array for the formatted name.

uint32_t SdFile::fileSize (void) const [inline]

Returns:

The total number of bytes in a file or directory.

uint32_t SdFile::firstCluster (void) const [inline]

Returns:

The first cluster number for a file or directory.

uint8_t SdFile::isDir (void) const [inline]

Returns:

True if this is a **SdFile** for a directory else false.

uint8_t SdFile::isFile (void) const [inline]

Returns:

True if this is a **SdFile** for a file else false.

uint8_t SdFile::isOpen (void) const [inline]

Returns:

True if this is a **SdFile** for an open file/directory else false.

uint8_t SdFile::isRoot (void) const [inline]

Returns:

True if this is a **SdFile** for the root directory.

uint8_t SdFile::isSubDir (void) const [inline]

Returns:

True if this is a **SdFile** for a subdirectory else false.

void SdFile::ls (uint8_t *flags* = 0, uint8_t *indent* = 0)

List directory contents to Serial.

Parameters:

[in] *flags* The inclusive OR of

LS_DATE - Print file modification date

LS_SIZE - Print file size.

LS_R - Recursive list of subdirectories.

Parameters:

[in] *indent* Amount of space before file name. Used for recursive list to indicate subdirectory level.

uint8_t SdFile::makeDir (SdFile & *dir*, const char * *dirName*) [inline]

Deprecated:

Use: uint8_t **SdFile::makeDir(SdFile* dir, const char* dirName)**;

uint8_t SdFile::makeDir (SdFile * *dir*, const char * *dirName*)

Make a new directory.

Parameters:

[in] *dir* An open SdFat instance for the directory that will containing the new directory.
[in] *dirName* A valid 8.3 DOS name for the new directory.

Returns:

The value one, true, is returned for success and the value zero, false, is returned for failure. Reasons for failure include this **SdFile** is already open, *dir* is not a directory, *dirName* is invalid or already exists in *dir* .

uint8_t SdFile::open (SdFile & *dirFile*, uint16_t *index*, uint8_t *oflag*) `[inline]`

Deprecated:

Use: uint8_t **SdFile::open(SdFile* dirFile, uint16_t index, uint8_t oflag)**;

uint8_t SdFile::open (SdFile & *dirFile*, const char * *fileName*) `[inline]`

Deprecated:

Do not use in new apps

uint8_t SdFile::open (SdFile & *dirFile*, const char * *fileName*, uint8_t *oflag*) `[inline]`

Deprecated:

Use: uint8_t **SdFile::open(SdFile* dirFile, const char* fileName, uint8_t oflag)**;

uint8_t SdFile::open (SdFile * *dirFile*, const char * *fileName*, uint8_t *oflag*)

Open a file or directory by name.

Parameters:

[in] *dirFile* An open SdFat instance for the directory containing the file to be opened.
[in] *fileName* A valid 8.3 DOS name for a file to be opened.
[in] *oflag* Values for *oflag* are constructed by a bitwise-inclusive OR of flags from the following list

O_READ - Open for reading.

O_RDONLY - Same as O_READ.

O_WRITE - Open for writing.

O_WRONLY - Same as O_WRITE.

O_RDWR - Open for reading and writing.

O_APPEND - If set, the file offset shall be set to the end of the file prior to each write.

O_CREAT - If the file exists, this flag has no effect except as noted under O_EXCL below. Otherwise, the file shall be created

O_EXCL - If O_CREAT and O_EXCL are set, **open()** shall fail if the file exists.

O_SYNC - Call **sync()** after each write. This flag should not be used with **write(uint8_t)**, **write_P(PGM_P)**, **writeln_P(PGM_P)**, or the Arduino **Print** class. These functions do character at a time writes so **sync()** will be called after each byte.

O_TRUNC - If the file exists and is a regular file, and the file is successfully opened and is not read only, its length shall be truncated to 0.

Note:

Directory files must be opened read only. Write and truncation is not allowed for directory files.

Returns:

The value one, true, is returned for success and the value zero, false, is returned for failure. Reasons for failure include this **SdFile** is already open, *difFile* is not a directory, *fileName* is invalid, the file does not exist or can't be opened in the access mode specified by oflag.

uint8_t SdFile::open (SdFile * *dirFile*, uint16_t *index*, uint8_t *oflag*)

Open a file by index.

Parameters:

[in] *dirFile* An open SdFat instance for the directory.
[in] *index* The *index* of the directory entry for the file to be opened. The value for *index* is (directory file position)/32.
[in] *oflag* Values for *oflag* are constructed by a bitwise-inclusive OR of flags O_READ, O_WRITE, O_TRUNC, and O_SYNC.

See **open()** by fileName for definition of flags and return values.

uint8_t SdFile::openRoot (SdVolume & *vol*) `[inline]`

Deprecated:

Use: uint8_t **SdFile::openRoot(SdVolume* vol)**;

uint8_t SdFile::openRoot (SdVolume * *vol*)

Open a volume's root directory.

Parameters:

[in] *vol* The FAT volume containing the root directory to be opened.

Returns:

The value one, true, is returned for success and the value zero, false, is returned for failure. Reasons for failure include the FAT volume has not been initialized or it a FAT12 volume.

void SdFile::printDirName (const dir_t & *dir*, uint8_t *width*) `[static]`

Print the name field of a directory entry in 8.3 format to Serial.

Parameters:

[in] *dir* The directory structure containing the name.
[in] *width* Blank fill name if length is less than *width* .

void SdFile::printFatDate (uint16_t *fatDate*) `[static]`

Print a directory date field to Serial.

Format is yyyy-mm-dd.

Parameters:

[in] *fatDate* The date field from a directory entry.

void SdFile::printFatTime (uint16_t *fatTime*) `[static]`

Print a directory time field to Serial.

Format is hh:mm:ss.

Parameters:

[in] *fatTime* The time field from a directory entry.

void SdFile::printTwoDigits (uint8_t *v*) [static]

Print a value as two digits to Serial.

Parameters:

[in] *v* Value to be printed, 0 <= *v* <= 99

int16_t SdFile::read (void * *buf*, uint16_t *nbyte*)

Read data from a file starting at the current position.

Parameters:

[out] *buf* Pointer to the location that will receive the data.
[in] *nbyte* Maximum number of bytes to read.

Returns:

For success **read()** returns the number of bytes read. A value less than *nbyte* , including zero, will be returned if end of file is reached. If an error occurs, **read()** returns -1. Possible errors include **read()** called before a file has been opened, corrupt file system or an I/O error occurred.

int16_t SdFile::read (void) [inline]

Read the next byte from a file.

Returns:

For success read returns the next byte in the file as an int. If an error occurs or end of file is reached -1 is returned.

int8_t SdFile::readDir (dir_t & *dir*) [inline]

Deprecated:

Use: int8_t **SdFile::readDir(dir_t* dir)**;

int8_t SdFile::readDir (dir_t * *dir*)

Read the next directory entry from a directory file.

Parameters:

[out] *dir* The dir_t struct that will receive the data.

Returns:

For success **readDir()** returns the number of bytes read. A value of zero will be returned if end of file is reached. If an error occurs, **readDir()** returns -1. Possible errors include **readDir()** called before a directory has been opened, this is not a directory file or an I/O error occurred.

static uint8_t SdFile::remove (SdFile & *dirFile*, const char * *fileName*) [inline, static]

Deprecated:

Use: static uint8_t **SdFile::remove(SdFile* dirFile, const char* fileName)**;

uint8_t SdFile::remove (void)

Remove a file.

The directory entry and all data for the file are deleted.

Note:

This function should not be used to delete the 8.3 version of a file that has a long name. For example if a file has the long name "New Text Document.txt" you should not delete the 8.3 name "NEWTEX~1.TXT".

Returns:

The value one, true, is returned for success and the value zero, false, is returned for failure. Reasons for failure include the file read-only, is a directory, or an I/O error occurred.

uint8_t SdFile::remove (SdFile * *dirFile*, const char * *fileName*) [static]

Remove a file.

The directory entry and all data for the file are deleted.

Parameters:

[in] *dirFile* The directory that contains the file.
[in] *fileName* The name of the file to be removed.

Note:

This function should not be used to delete the 8.3 version of a file that has a long name. For example if a file has the long name "New Text Document.txt" you should not delete the 8.3 name "NEWTEX~1.TXT".

Returns:

The value one, true, is returned for success and the value zero, false, is returned for failure. Reasons for failure include the file is a directory, is read only, *dirFile* is not a directory, *fileName* is not found or an I/O error occurred.

void SdFile::rewind (void) [inline]

Set the file's current position to zero.

uint8_t SdFile::rmDir (void)

Remove a directory file.

The directory file will be removed only if it is empty and is not the root directory. **rmDir()** follows DOS and Windows and ignores the read-only attribute for the directory.

Note:

This function should not be used to delete the 8.3 version of a directory that has a long name. For example if a directory has the long name "New folder" you should not delete the 8.3 name "NEWFOL~1".

Returns:

The value one, true, is returned for success and the value zero, false, is returned for failure. Reasons for failure include the file is not a directory, is the root directory, is not empty, or an I/O error occurred.

uint8_t SdFile::rmRfStar (void)

Recursively delete a directory and all contained files.

This is like the Unix/Linux 'rm -rf *' if called with the root directory hence the name.

Warning - This will remove all contents of the directory including subdirectories. The directory will then be removed if it is not root. The read-only attribute for files will be ignored.

Note:

This function should not be used to delete the 8.3 version of a directory that has a long name. See **remove()** and **rmDir()**.

Returns:

The value one, true, is returned for success and the value zero, false, is returned for failure.

uint8_t SdFile::seekCur (uint32_t *pos*) [inline]

Set the files position to current position + *pos* . See **seekSet()**.

uint8_t SdFile::seekEnd (void) `[inline]`

Set the files current position to end of file. Useful to position a file for append. See **seekSet()**.

uint8_t SdFile::seekSet (uint32_t *pos*)

Sets a file's position.

Parameters:

[in] *pos* The new position in bytes from the beginning of the file.

Returns:

The value one, true, is returned for success and the value zero, false, is returned for failure.

void SdFile::setUnbufferedRead (void) `[inline]`

Use unbuffered reads to access this file. Used with Wave Shield ISR. Used with **Sd2Card::partialBlockRead()** in WaveRP.

Not recommended for normal applications.

uint8_t SdFile::sync (void)

The **sync()** call causes all modified data and directory fields to be written to the storage device.

Returns:

The value one, true, is returned for success and the value zero, false, is returned for failure. Reasons for failure include a call to **sync()** before a file has been opened or an I/O error.

uint8_t SdFile::timestamp (uint8_t *flags*, uint16_t *year*, uint8_t *month*, uint8_t *day*, uint8_t *hour*, uint8_t *minute*, uint8_t *second*)

Set a file's timestamps in its directory entry.

Parameters:

[in] *flags* Values for *flags* are constructed by a bitwise-inclusive OR of flags from the following list

T_ACCESS - Set the file's last access date.

T_CREATE - Set the file's creation date and time.

T_WRITE - Set the file's last write/modification date and time.

Parameters:

[in] *year* Valid range 1980 - 2107 inclusive.
[in] *month* Valid range 1 - 12 inclusive.
[in] *day* Valid range 1 - 31 inclusive.
[in] *hour* Valid range 0 - 23 inclusive.
[in] *minute* Valid range 0 - 59 inclusive.
[in] *second* Valid range 0 - 59 inclusive

Note:

It is possible to set an invalid date since there is no check for the number of days in a month.
Modify and access timestamps may be overwritten if a date time callback function has been set by **dateTimeCallback()**.

Returns:

The value one, true, is returned for success and the value zero, false, is returned for failure.

uint8_t SdFile::truncate (uint32_t *length*)

Truncate a file to a specified length. The current file position will be maintained if it is less than or equal to *length* otherwise it will be set to end of file.

Parameters:

[in] *length* The desired length for the file.

Returns:

The value one, true, is returned for success and the value zero, false, is returned for failure. Reasons for failure include file is read only, file is a directory, *length* is greater than the current file size or an I/O error occurs.

uint8_t SdFile::type (void) const `[inline]`

Type of this **SdFile**. You should use **isFile()** or **isDir()** instead of **type()** if possible.

Returns:

The file or directory type.

uint8_t SdFile::unbufferedRead (void) const `[inline]`

Returns:

Unbuffered read flag.

SdVolume* SdFile::volume (void) const `[inline]`

Returns:

SdVolume that contains this file.

void SdFile::write (const char * *str*) `[virtual]`

Write a string to a file. Used by the Arduino **Print** class.

Use **SdFile::writeError** to check for errors.

Reimplemented from **Print** (*p.**Error! Bookmark not defined.***).

int16_t SdFile::write (const void * *buf*, uint16_t *nbyte*)

Write data to an open file.

Note:

Data is moved to the cache but may not be written to the storage device until **sync()** is called.

Parameters:

[in] *buf* Pointer to the location of the data to be written.
[in] *nbyte* Number of bytes to write.

Returns:

For success **write()** returns the number of bytes written, always *nbyte* . If an error occurs, **write()** returns -1. Possible errors include **write()** is called before a file has been opened, write is called for a read-only file, device is full, a corrupt file system or an I/O error.

void SdFile::write (uint8_t *b*) `[virtual]`

Write a byte to a file. Required by the Arduino **Print** class.

Use **SdFile::writeError** to check for errors.

Reimplemented from **Print** (*p.**Error! Bookmark not defined.***).

void SdFile::write_P (PGM_P *str*)

Write a PROGMEM string to a file.

Use **SdFile::writeError** to check for errors.

void SdFile::writeln_P (PGM_P *str*)

Write a PROGMEM string followed by CR/LF to a file.

Use **SdFile::writeError** to check for errors.

Member Data Documentation

bool SdFile::writeError

writeError is set to true if an error occurs during a **write()**. Set writeError to false before calling **print()** and/or **write()** and check for true after calls to **print()** and/or **write()**.

The documentation for this class was generated from the following files:

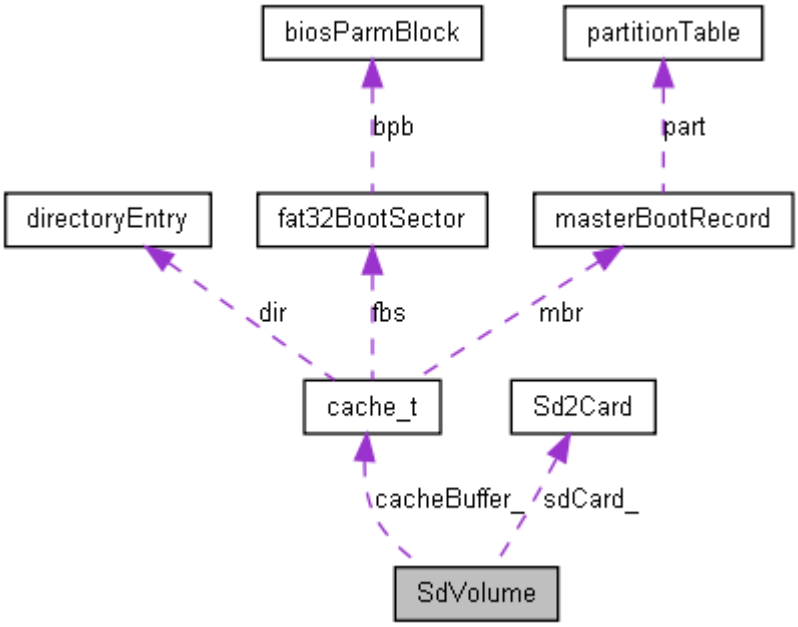
- Arduino/libraries/SdFat/**SdFat.h**
- Arduino/libraries/SdFat/SdFile.cpp

SdVolume Class Reference

Access FAT16 and FAT32 volumes on SD and SDHC cards.

`#include <SdFat.h>`

Collaboration diagram for SdVolume:



Public Member Functions

- uint8_t **blocksPerCluster** (void) const
- uint32_t **blocksPerFat** (void) const
- uint32_t **clusterCount** (void) const
- uint8_t **clusterSizeShift** (void) const
- uint32_t **dataStartBlock** (void) const
- uint8_t **fatCount** (void) const
- uint32_t **fatStartBlock** (void) const
- uint8_t **fatType** (void) const
- uint8_t **init** (**Sd2Card** &dev, uint8_t part)
- uint8_t **init** (**Sd2Card** &dev)
- uint8_t **init** (**Sd2Card** *dev, uint8_t part)
- uint8_t **init** (**Sd2Card** *dev)
- uint32_t **rootDirEntryCount** (void) const
- uint32_t **rootDirStart** (void) const
- **SdVolume** (void)

Static Public Member Functions

- static uint8_t * **cacheClear** (void)
- static **Sd2Card** * **sdCard** (void)

Friends

- class **SdFile**

Detailed Description

Access FAT16 and FAT32 volumes on SD and SDHC cards.

Constructor & Destructor Documentation

SdVolume::SdVolume (void) [inline]

Create an instance of **SdVolume**

Member Function Documentation

uint8_t SdVolume::blocksPerCluster (void) const [inline]

Returns:

The volume's cluster size in blocks.

uint32_t SdVolume::blocksPerFat (void) const [inline]

Returns:

The number of blocks in one FAT.

static uint8_t* SdVolume::cacheClear (void) [inline, static]

Clear the cache and returns a pointer to the cache. Used by the WaveRP recorder to do raw write to the SD card. Not for normal apps.

uint32_t SdVolume::clusterCount (void) const [inline]

Returns:

The total number of clusters in the volume.

uint8_t SdVolume::clusterSizeShift (void) const [inline]

Returns:

The shift count required to multiply by blocksPerCluster.

uint32_t SdVolume::dataStartBlock (void) const [inline]

Returns:

The logical block number for the start of file data.

uint8_t SdVolume::fatCount (void) const `[inline]`

Returns:

The number of FAT structures on the volume.

uint32_t SdVolume::fatStartBlock (void) const `[inline]`

Returns:

The logical block number for the start of the first FAT.

uint8_t SdVolume::fatType (void) const `[inline]`

Returns:

The FAT type of the volume. Values are 12, 16 or 32.

uint8_t SdVolume::init (Sd2Card & *dev*, uint8_t *part*) `[inline]`

Deprecated:

Use: uint8_t **SdVolume::init(Sd2Card* dev, uint8_t vol)**;

uint8_t SdVolume::init (Sd2Card & *dev*) `[inline]`

Deprecated:

Use: uint8_t **SdVolume::init(Sd2Card* dev)**;

uint8_t SdVolume::init (Sd2Card * *dev*, uint8_t *part*)

Initialize a FAT volume.

Parameters:

[in] *dev* The SD card where the volume is located.
[in] *part* The partition to be used. Legal values for *part* are 1-4 to use the corresponding partition on a device formatted with a MBR, Master Boot Record, or zero if the device is formatted as a super floppy with the FAT boot sector in block zero.

Returns:

The value one, true, is returned for success and the value zero, false, is returned for failure. Reasons for failure include not finding a valid partition, not finding a valid FAT file system in the specified partition or an I/O error.

uint8_t SdVolume::init (Sd2Card * *dev*) `[inline]`

Initialize a FAT volume. Try partition one first then try super floppy format.

Parameters:

[in] *dev* The **Sd2Card** where the volume is located.

Returns:

The value one, true, is returned for success and the value zero, false, is returned for failure. Reasons for failure include not finding a valid partition, not finding a valid FAT file system or an I/O error.

uint32_t SdVolume::rootDirEntryCount (void) const `[inline]`

Returns:

The number of entries in the root directory for FAT16 volumes.

uint32_t SdVolume::rootDirStart (void) const [inline]

Returns:

The logical block number for the start of the root directory on FAT16 volumes or the first cluster number on FAT32 volumes.

static Sd2Card* SdVolume::sdCard (void) [inline, static]

return a pointer to the **Sd2Card** object for this volume

The documentation for this class was generated from the following files:

- Arduino/libraries/SdFat/**SdFat.h**
- Arduino/libraries/SdFat/SdVolume.cpp