Arduino Fat16 Library

Introduction

The Arduino **Fat16** Library is a minimal implementation of the FAT16 file system on standard SD flash memory cards. **Fat16** supports read, write and file creation.

The **Fat16** class only supports access to files in the root directory and only supports short 8.3 names. Directory time and date fields for creation, access and write are not maintained.

Fat16 was designed to use the Arduino Version 12 **Print** class which allows files to be written with **print()** and **println()**.

It should be possible to use **Fat16** with storage devices other than SD flash cards. Hardware access can be through any class derived from the **BlockDevice** class.

Hardware Configuration

Fat16 was developed using an `Adafruit Industries GPS Shield`. See the `Schematic` for details.

Warning

Fat16 has been tested with several SD Cards but is bound to contain many bugs. In most companies this would be called pre-alpha software. I hope people will try it and send me comments.

Bugs and Comments

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

Fat16 Usage

The class **Fat16** is a minimal implementation of FAT16 on standard SD cards. High Capacity SD cards, SDHC, are not supported. It should work on all standard cards from 8MB to 2GB formatted with a FAT16 file system.

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.

Fat16 only supports access to files in the root directory and only supports short 8.3 names.

It is possible to open a file with two or more instances of **Fat16**. A file may be corrupted if data is written to the file by more than one instance of **Fat16**.

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.

Fat16 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.

Fat16 has been optimized for The Arduino ATmega168. Minimizing RAM use is the highest priority goal followed by flash use and finally performance. Most SD cards only support 512 byte block write operations so a 512 byte cache buffer is used by **Fat16**. This is the main use of RAM. A small amount of RAM is used to store key volume and file information. Flash memory usage can be controlled by selecting options in **Fat16Config.h**.

How to format SD Cards as FAT16 Volumes

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 MBR is to use SDFormatter which can be downloaded from:

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

SDFormatter does not have an option for FAT type so it may format small cards as FAT12 and larger cards as FAT32.

After the MBR is restored by SDFormatter you can reformat it and force the volume type to be FAT16 by selecting the FAT option on XP or specifying the "Allocation unit size" on Vista.

The FAT type, FAT12, FAT16, or FAT32, is determined by the count of clusters on the volume and nothing else.

Microsoft published the following code for determining FAT type:

```
if (CountOfClusters < 4085) {
  // Volume is FAT12
}
else if (CountOfClusters < 65525) {
  // Volume is FAT16
}
else {
  // Volume is FAT32
}
```

When you format a FAT volume, 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

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 ATmega168 datasheet:

http://www.atmel.com/dyn/resources/prod_documents/doc8025.pdf

Fat16/Fat16Config.h File Reference

Define Documentation

#define FAT16_DEBUG_SUPPORT 1

Set non-zero to allow access to **Fat16** internals by cardInfo debug sketch

#define FAT16_PRINT_SUPPORT 1

Set non-zero to enable Arduino V12 **Print** support

#define FAT16_READ_SUPPORT 1

Set non-zero to enable read() support

#define FAT16_SAVE_RAM 1

Set non-zero to store strings in PROGMEM. Will cause bogus warnings from gcc.

#define FAT16_SEEK_SUPPORT 1

Set non-zero to enable seek() support

#define FAT16_WRITE_SUPPORT 1

Set non-zero to enable write() and create() support

#define SD_CARD_READ_REG_SUPPORT 0

Should be zero - only useful for debug

#define SD_CARD_SIZE_SUPPORT 1

Set non-zero to enable **SdCard::cardSize()**

#define SD_PROTECT_BLOCK_ZERO 1

SdCard::writeBlock will protect block zero if set non-zero

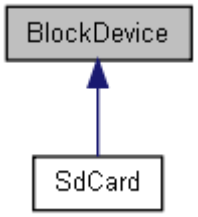
Class Documentation

BlockDevice Class Reference

BlockDevice is the base class for devices supported by **Fat16**.

`#include <BlockDevice.h>`

Inheritance diagram for BlockDevice:



Public Member Functions

- virtual uint8_t **readBlock** (uint32_t blockNumber, uint8_t *dst)
- virtual uint8_t **writeBlock** (uint32_t blockNumber, uint8_t *src)

Detailed Description

BlockDevice is the base class for devices supported by **Fat16**.

Member Function Documentation

virtual uint8_t BlockDevice::readBlock (uint32_t *blockNumber*, uint8_t * *dst*) `[virtual]`

Read a 512 byte block from a storage device.

Parameters:

blockNumber Logical block to be read.
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.

Reimplemented in **SdCard** (*p.12*).

virtual uint8_t BlockDevice::writeBlock (uint32_t *blockNumber*, uint8_t * *src*) `[virtual]`

Write a 512 byte block to a storage device.

Parameters:

blockNumber Logical block to be written.
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.

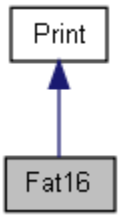
Reimplemented in **SdCard** (*p.12*).

Fat16 Class Reference

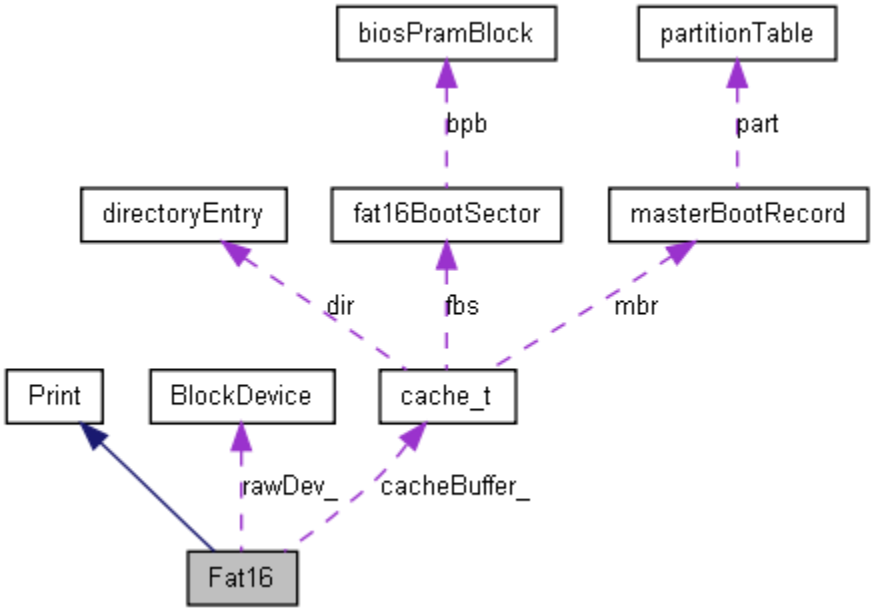
Fat16 implements a minimal Arduino FAT16 Library.

`#include <Fat16.h>`

Inheritance diagram for Fat16:



Collaboration diagram for Fat16:



Public Member Functions

- uint8_t **close** (void)
- uint8_t **create** (char *fileName)
- uint32_t **fileSize** (void)
- uint8_t **isOpen** (void)
- uint8_t **open** (char *fileName)
- uint8_t **open** (uint16_t entry)
- int16_t **read** (void)
- int16_t **read** (uint8_t *dst, uint16_t count)
- uint32_t **readPos** (void)
- uint8_t **seek** (uint32_t pos)
- uint8_t **sync** (void)
- void **write** (uint8_t b)
- int16_t **write** (uint8_t *src, uint16_t count)

Static Public Member Functions

- static uint8_t **init** (**BlockDevice** &dev, uint8_t part)
- static **dir_t** * **readDir** (uint16_t &entry, uint8_t skip=(DIR_ATT_VOLUME_ID|DIR_ATT_DIRECTORY))
- static uint16_t **rootDirEntryCount** (void)
- static void **dbgSetDev** (**BlockDevice** &dev)
- static uint8_t * **dbgCacheBlock** (uint32_t blockNumber)
- static **dir_t** * **dbgCacheDir** (uint16_t index)
- static uint16_t * **dbgCacheFat** (uint16_t cluster)

Detailed Description

Fat16 implements a minimal Arduino FAT16 Library.

Fat16 does not support subdirectories or long file names.

Member Function Documentation

uint8_t Fat16::init (BlockDevice & *dev*, uint8_t *part*) `[static]`

Initialize a FAT16 volume.

Parameters:

dev The **BlockDevice** where the volume is located.
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 FAT16 file system in the specified partition, a call to **init()** after a volume has been successful initialized or an I/O error.

static dir_t* Fat16::readDir (uint16_t & *entry*, uint8_t *skip* = `(DIR_ATT_VOLUME_ID | DIR_ATT_DIRECTORY)`) `[inline, static]`

Read the next short, 8.3, directory entry into the cache buffer.

Unused entries and entries for long names are skipped.

The directory entry must not be modified since the cached block containing the entry may be written back to the storage device.

Parameters:

entry The search starts at *entry* and *entry* is updated with the root directory index of the found directory entry. If the entry is a file, it may be opened by calling **open(entry)**.
skip Skip entries that have these attributes. If *skip* is not specified, the default is to skip the volume label and directories.

Returns:

A pointer to a dir_t structure for the found directory entry, or NULL if an error occurs or the end of the root directory is reached. On success, *entry* is set to the index of the found directory entry.

static uint16_t Fat16::rootDirEntryCount (void) `[inline, static]`

Returns:

The number of entries in the root directory.

uint8_t Fat16::close (void)

Closes a file and forces write of cached data and directory information 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 Fat16::create (char * *fileName*)

Create and open a new file.

Note:

This function only creates files in the root directory and only supports short DOS 8.3 names. See **open()** for more information.

Parameters:

fileName a valid DOS 8.3 file name.

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 Fat16::fileSize (void) `[inline]`

Returns:

The file's size in bytes. This is also the write position for the file.

uint8_t Fat16::isOpen (void) `[inline]`

Checks the file's open/closed status for this instance of **Fat16**.

Returns:

The value true if a file is open otherwise false;

uint8_t Fat16::open (char * *fileName*)

Open a file for read and write by file name. Two file positions are maintained. The write position is at the end of the file. Data is appended to the file. The read position starts at the beginning of the file.

Note:

The file must be in the root directory and must have a DOS 8.3 name.

Parameters:

fileName A valid 8.3 DOS name for a file in the root directory.

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, a file is already open, *fileName* is invalid, the file does not exist or it is a directory.

uint8_t Fat16::open (uint16_t *index*)

Open a file for read and write by file index. Two file positions are maintained. The write position is at the end of the file. Data is appended to the file. The read position starts at the beginning of the file.

Parameters:

index The root directory index of the file to be opened. See **readDir()**.

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, a file is already open, *index* is invalid or is not the index of a file.

int16_t Fat16::read (void)

Read one 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.

int16_t Fat16::read (uint8_t * *dst*, uint16_t *count*)

Read data from a file at the current read position.

Parameters:

dst Pointer to the location that will receive the data.
count Maximum number of bytes to read.

Returns:

For success read returns the number of bytes read. A value less than *count* , including zero, may 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 I/O error.

uint32_t Fat16::readPos (void) `[inline]`

Returns:

The read position in bytes.

uint8_t Fat16::seek (uint32_t *pos*)

Sets the file's read position.

Returns:

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

uint8_t Fat16::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.

void Fat16::write (uint8_t *b*) `[virtual]`

Append one byte to a file. This function is called by Arduino's **Print** class.

Note:

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

Parameters:

b The byte to be written.

Reimplemented from **Print** .

int16_t Fat16::write (uint8_t * *src*, uint16_t *count*)

Write data at the end of an open file.

Note:

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

Parameters:

src Pointer to the location of the data to be written.
count Number of bytes to write.

Returns:

For success **write()** returns the number of bytes written, always *count* . 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.

static void Fat16::dbgSetDev (BlockDevice & *dev*) [inline, static]

For debug only. Do not use in applications.

static uint8_t* Fat16::dbgCacheBlock (uint32_t *blockNumber*) [inline, static]

For debug only. Do not use in applications.

static dir_t* Fat16::dbgCacheDir (uint16_t *index*) [inline, static]

For debug only. Do not use in applications.

static uint16_t* Fat16::dbgCacheFat (uint16_t *cluster*) [inline, static]

For debug only. Do not use in applications.

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

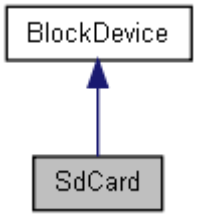
- Fat16/Fat16.h
- Fat16/Fat16.cpp

SdCard Class Reference

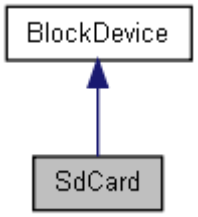
Hardware access class for SD flash cards.

`#include <SdCard.h>`

Inheritance diagram for SdCard:



Collaboration diagram for SdCard:



Public Member Functions

- uint32_t **cardSize** (void)
- uint8_t **init** (void)
- uint8_t **readBlock** (uint32_t block, uint8_t *dst)
- uint8_t **writeBlock** (uint32_t block, uint8_t *src)

Detailed Description

Hardware access class for SD flash cards.

Supports raw access to a standard SD flash memory card.

Member Function Documentation

uint32_t SdCard::cardSize (void)

Determine the size of a standard SD flash memory card

Returns:

The number of 512 byte data blocks in the card

uint8_t SdCard::init (void)

Initialize a SD flash memory card.

Returns:

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

uint8_t SdCard::readBlock (uint32_t *blockNumber*, uint8_t * *dst*) [virtual]

Reads a 512 byte block from a storage device.

Parameters:

blockNumber Logical block to be read.
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.

Reimplemented from **BlockDevice** (*p.5*).

uint8_t SdCard::writeBlock (uint32_t *blockNumber*, uint8_t * *src*) [virtual]

Writes a 512 byte block to a storage device.

Parameters:

blockNumber Logical block to be written.
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.

Reimplemented from **BlockDevice** (*p.5*).

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

- Fat16/SdCard.h
- Fat16/SdCard.cpp