Arduino FatReader Library

Introduction

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

The **FatReader** class only supports short 8.3 names.

Short 8.3 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 uppercase letters and digits. The following special characters are also allowed:

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

Hardware Configuration

FatReader was developed using an `Adafruit Industries Wave Shield`. See the `Schematic` for details.

Warning

FatReader 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`.

References

Adafruit Industries:

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

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

Schematic - `http://www.ladyada.net/make/waveshield/download.html`

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

Class Index

Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

File Index

File List

Here is a list of all files with brief descriptions:

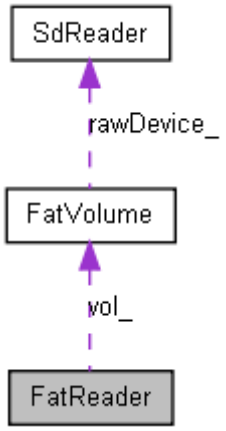
Class Documentation

FatReader Class Reference

FatReader implements a minimal FAT16/FAT32 file reader class.

```
#include <FatReader.h>
```

Collaboration diagram for FatReader:



Public Member Functions

- void **close** (void)
- **FatReader** (void)
- uint32_t **fileSize** (void)
- uint32_t **firstCluster** (void)
- uint8_t **isDir** (void)
- uint8_t **isFile** (void)
- uint8_t **isOpen** (void)
- uint8_t **open** (**FatReader** &dir, char *name)
- uint8_t **open** (**FatVolume** &vol, dir_t &dir)
- uint8_t **openRoot** (**FatVolume** &vol)
- int16_t **read** (uint8_t *dst, uint16_t count)
- uint32_t **readCluster** (void)
- int8_t **readDir** (dir_t &dir)
- uint32_t **readPosition** (void)
- void **rewind** (void)
- uint8_t **seekCur** (uint32_t pos)
- uint8_t **seekSet** (uint32_t pos)
- uint8_t **type** (void)
- **FatVolume** * **volume** (void)

Detailed Description

FatReader implements a minimal FAT16/FAT32 file reader class.

Constructor & Destructor Documentation

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

Create an instance of **FatReader**.

Member Function Documentation

void FatReader::close (void) [inline]

Close this instance of **FatReader**.

uint32_t FatReader::fileSize (void) [inline]

Returns:

The total number of bytes in a file or directory.

uint32_t FatReader::firstCluster (void) [inline]

Returns:

The first cluster number for a file or directory.

uint8_t FatReader::isDir (void) [inline]

Returns:

True if this is a **FatReader** for a directory else false

uint8_t FatReader::isFile (void) [inline]

Returns:

True if this is a **FatReader** for a file else false

uint8_t FatReader::isOpen (void) [inline]

Returns:

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

uint8_t FatReader::open (FatReader & *dir*, char * *name*)

Open a file or subdirectory by name.

Note:

The file or subdirectory, *name* , must be in the specified directory, *dir* , and must have a DOS 8.3 name.

Parameters:

dir An open **FatReader** instance for the directory.
name A valid 8.3 DOS name for a file or subdirectory in the directory *dir* .

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, *dir* is not a directory, *name* is invalid, the file or subdirectory does not exist, or an I/O error occurred.

uint8_t FatReader::open (FatVolume & *vol*, dir_t & *dir*)

Open a file or subdirectory by directory structure.

Parameters:

vol The FAT volume that contains the file or subdirectory.
dir The directory structure describing the file or subdirectory.

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, *vol* , has not been initialized, *vol* is a FAT12 volume or *dir* is not a valid directory entry.

uint8_t FatReader::openRoot (FatVolume & *vol*)

Open a volume's root directory.

Parameters:

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.

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

Read data from a file at starting 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, 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.

uint32_t FatReader::readCluster (void) `[inline]`

Returns:

The current cluster number for a file or directory.

int8_t FatReader::readDir (dir_t & *dir*)

Read the next directory entry from a directory file.

Parameters:

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.

uint32_t FatReader::readPosition (void) `[inline]`

Returns:

The read position for a file or directory.

void FatReader::rewind (void)

Set read position to start of file

uint8_t FatReader::seekCur (uint32_t *offset*)

Set the read position for a file or directory to the current position plus *offset* .

Parameters:

offset The amount to advance the read position.

Returns:

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

uint8_t FatReader::seekSet (uint32_t *pos*) `[inline]`

Set the read position for a file or directory to *pos* .

Parameters:

pos The new read 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.

uint8_t FatReader::type (void) `[inline]`

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

Returns:

The file or directory type.

FatVolume* FatReader::volume (void) `[inline]`

Parent volume

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

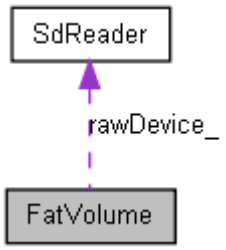
- C:/Arduino/arduino15/hardware/libraries/WaveHC/**FatReader.h**
- C:/Arduino/arduino15/hardware/libraries/WaveHC/**FatReader.cpp**

FatVolume Class Reference

FatVolume provides access to FAT volumes.

```
#include <FatReader.h>
```

Collaboration diagram for FatVolume:



Public Member Functions

- uint8_t **blocksPerCluster** (void)
- uint32_t **blocksPerFat** (void)
- uint32_t **clusterCount** (void)
- uint32_t **dataStartBlock** (void)
- uint8_t **fatCount** (void)
- uint32_t **fatStartBlock** (void)
- uint8_t **fatType** (void)
- **FatVolume** (void)
- uint8_t **init** (**SdReader** &dev, uint8_t part)
- uint8_t **init** (**SdReader** &dev)
- **SdReader** * **rawDevice** (void)
- uint32_t **rootDirEntryCount** (void)
- uint32_t **rootDirStart** (void)
- uint32_t **totalBlocks** (void)

Friends

- class **FatReader**

Detailed Description

FatVolume provides access to FAT volumes.

Constructor & Destructor Documentation

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

Create an instance of **FatVolume**

Member Function Documentation

uint8_t FatVolume::blocksPerCluster (void) `[inline]`

Returns:

The volume's cluster size in blocks.

uint32_t FatVolume::blocksPerFat (void) `[inline]`

Returns:

The number of blocks in one FAT.

uint32_t FatVolume::clusterCount (void) `[inline]`

Returns:

The total number of clusters in the volume.

uint32_t FatVolume::dataStartBlock (void) [inline]

Returns:

The logical block number for the start of file data.

uint8_t FatVolume::fatCount (void) [inline]

Returns:

The number of FAT structures on the volume.

uint32_t FatVolume::fatStartBlock (void) [inline]

Returns:

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

uint8_t FatVolume::fatType (void) [inline]

Returns:

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

uint8_t FatVolume::init (SdReader & *dev*, uint8_t *part*)

Initialize a FAT volume.

Parameters:

dev The SD card 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 partition, not finding a valid FAT file system in the specified partition or an I/O error.

uint8_t FatVolume::init (SdReader & *dev*) [inline]

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

Parameters:

dev The **SdReader** 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.

SdReader* FatVolume::rawDevice (void) [inline]

Raw device for this volume

uint32_t FatVolume::rootDirEntryCount (void) `[inline]`

Returns:

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

uint32_t FatVolume::rootDirStart (void) `[inline]`

Returns:

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

uint32_t FatVolume::totalBlocks (void) `[inline]`

Returns:

The total number of blocks in the volume.

Friends And Related Function Documentation

friend class FatReader `[friend]`

Allow **FatReader** access to **FatVolume** private data.

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

- C:/Arduino/arduino15/hardware/libraries/WaveHC/**FatReader.h**
- C:/Arduino/arduino15/hardware/libraries/WaveHC/**FatReader.cpp**

SdReader Class Reference

Hardware access class for SD flash cards.

`#include <SdReader.h>`

Public Member Functions

- uint32_t **cardSize** (void)
- uint8_t **errorCode** (void)
- uint8_t **errorData** (void)
- uint8_t **init** (uint8_t slow=0)
- 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, uint8_t *dst, uint16_t count)
- void **readEnd** (void)

- **SdReader** (void)
- uint8_t **type** ()

Detailed Description

Hardware access class for SD flash cards.

Supports raw access to SD and SDHC flash memory cards.

Constructor & Destructor Documentation

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

Construct an instance of **SdReader**.

Member Function Documentation

uint32_t SdReader::cardSize (void)

Determine the size of a SD flash memory card.

Returns:

The number of 512 byte data blocks in the card

uint8_t SdReader::errorCode (void) `[inline]`

Returns:

error code for last error

uint8_t SdReader::errorData (void) `[inline]`

Returns:

error data for last error

uint8_t SdReader::init (uint8_t *slow* = 0)

Initialize a SD flash memory card.

Returns:

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

void SdReader::partialBlockRead (uint8_t *value*) `[inline]`

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 will occur if the time between reads is too long since the SD card will timeout.

Use this for applications like the Adafruit Wave Shield.

Parameters:

value The value TRUE (non-zero) or FALSE (zero).)

uint8_t SdReader::readBlock (uint32_t *block*, uint8_t * *dst*) [inline]

Read a 512 byte block from a SD card device.

Parameters:

block 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.

uint8_t SdReader::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 SdReader::readCSD (csd_t & *csd*) [inline]

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

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

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

Parameters:

block Logical block to be read.
offset Number of bytes to skip at start of block
dst Pointer to the location that will receive the data.
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 SdReader::readEnd (void)

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

uint8_t SdReader::type (void) [inline]

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

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

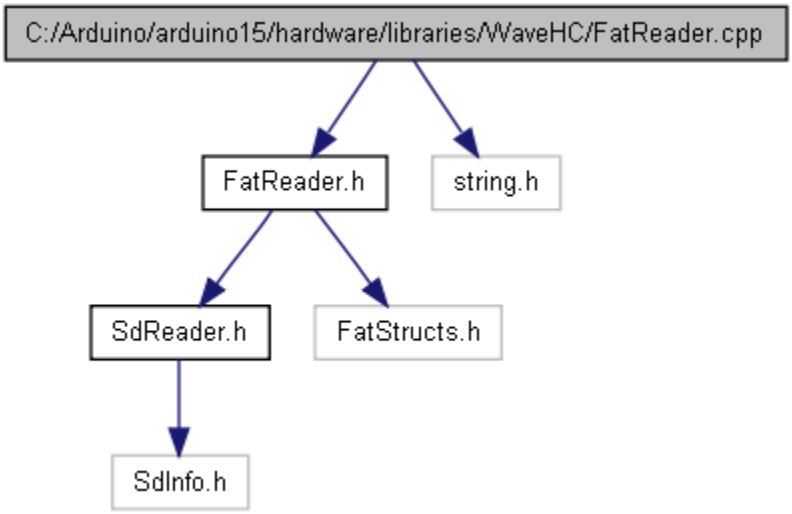
- C:/Arduino/arduino15/hardware/libraries/WaveHC/**SdReader.h**
- C:/Arduino/arduino15/hardware/libraries/WaveHC/**SdReader.cpp**

File Documentation

C:/Arduino/arduino15/hardware/libraries/WaveHC/FatReader.cpp File Reference

```
#include "FatReader.h"
#include <string.h>
```

Include dependency graph for FatReader.cpp:



Functions

- void **dirName** (dir_t &dir, char name[])

Function Documentation

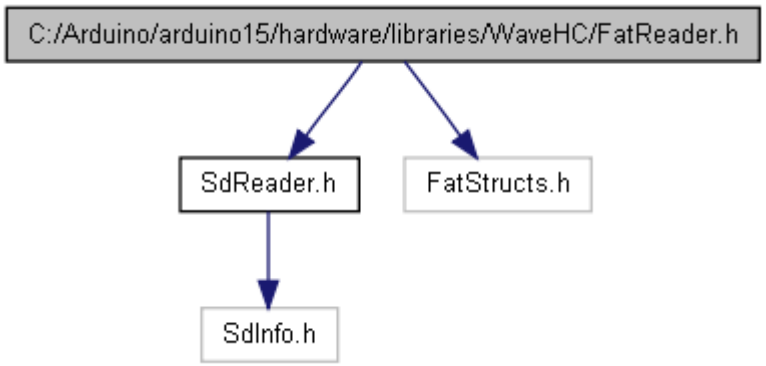
void dirName (dir_t & *dir*, char *name*[])

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

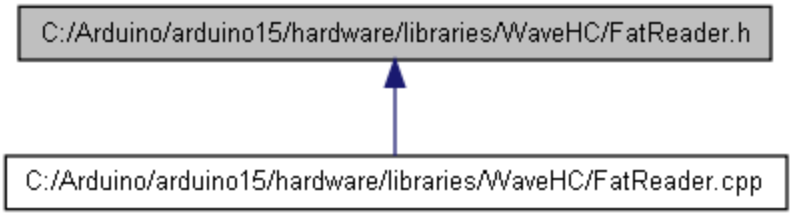
C:/Arduino/arduino15/hardware/libraries/WaveHC/FatReader.h File Reference

```
#include "SdReader.h"
#include "FatStructs.h"
```

Include dependency graph for FatReader.h:



This graph shows which files directly or indirectly include this file:



Classes

- class **FatReader**
  FatReader *implements a minimal FAT16/FAT32 file reader class.*
- class **FatVolume**
  FatVolume *provides access to FAT volumes.*

Defines

- #define **BPB_COUNT** 37
- #define **BPB_OFFSET** 11
- #define **DIR_ATT_FILE_TYPE_MASK** (DIR_ATT_VOLUME_ID | DIR_ATT_DIRECTORY)
- #define **DIR_IS_FILE**(dir) (((dir).attributes & DIR_ATT_FILE_TYPE_MASK) == 0)
- #define **DIR_IS_LONG_NAME**(dir) (((dir).attributes & DIR_ATT_LONG_NAME_MASK) == DIR_ATT_LONG_NAME)
- #define **DIR_IS_SUBDIR**(dir) (((dir).attributes & DIR_ATT_FILE_TYPE_MASK) == DIR_ATT_DIRECTORY)
- #define **FAT_READER_TYPE_CLOSED** 0
- #define **FAT_READER_TYPE_MIN_DIR** FAT_READER_TYPE_ROOT16
- #define **FAT_READER_TYPE_NORMAL** 1
- #define **FAT_READER_TYPE_ROOT16** 2
- #define **FAT_READER_TYPE_ROOT32** 3
- #define **FAT_READER_TYPE_SUBDIR** 4
- #define **PART_OFFSET** (512-64-2)

Functions

- void **dirName** (dir_t &dir, char name[])

Define Documentation

#define BPB_COUNT 37

Byte count for part of BIOS Parameter Block to be read by init()

#define BPB_OFFSET 11

Offset to BIOS Parameter Block in FAT Boot Sector

#define DIR_ATT_FILE_TYPE_MASK (DIR_ATT_VOLUME_ID | DIR_ATT_DIRECTORY)

Mask for file/subdirectory tests

#define DIR_IS_FILE(dir) (((dir).attributes & DIR_ATT_FILE_TYPE_MASK) == 0)

Directory entry is for a file

#define DIR_IS_LONG_NAME(dir) (((dir).attributes & DIR_ATT_LONG_NAME_MASK) == DIR_ATT_LONG_NAME)

Directory entry is part of a long name

#define DIR_IS_SUBDIR(dir) (((dir).attributes & DIR_ATT_FILE_TYPE_MASK) == DIR_ATT_DIRECTORY)

Directory entry is for a subdirectory

#define FAT_READER_TYPE_CLOSED 0

This **FatReader** has not been opened.

#define FAT_READER_TYPE_MIN_DIR FAT_READER_TYPE_ROOT16

Test value for directory type

#define FAT_READER_TYPE_NORMAL 1

FatReader for a file

#define FAT_READER_TYPE_ROOT16 2

FatReader for a FAT16 root directory

#define FAT_READER_TYPE_ROOT32 3

FatReader for a FAT32 root directory

#define FAT_READER_TYPE_SUBDIR 4

FatReader for a subdirectory

#define PART_OFFSET (512-64-2)

offset to partition table in mbr

Function Documentation

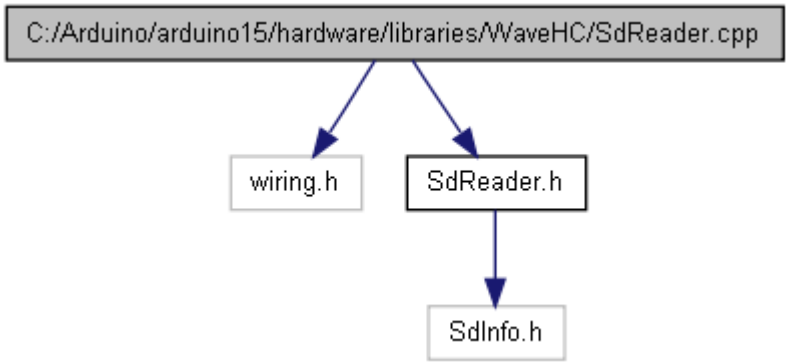
void dirName (dir_t & *dir*, char *name*[])

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

C:/Arduino/arduino15/hardware/libraries/WaveHC/SdReader.cpp File Reference

```
#include "wiring.h"
#include "SdReader.h"
```

Include dependency graph for SdReader.cpp:



Defines

- #define **DATA_RES_ACCEPTED** 0X05
- #define **DATA_RES_CRC_ERROR** 0X0B
- #define **DATA_RES_MASK** 0X1F
- #define **DATA_RES_WRITE_ERROR** 0X0D
- #define **DATA_START_BLOCK** 0XFE
- #define **MISO** 12
- #define **MOSI** 11
- #define **R1_IDLE_STATE** 1
- #define **R1_READY_STATE** 0
- #define **SCK** 13
- #define **SS** 10

Functions

- uint8_t **spiRec** (void)
- void **spiSend** (uint8_t b)
- void **spiSSHigh** (void)
- void **spiSSLow** (void)

Define Documentation

#define DATA_RES_ACCEPTED 0X05

write data accepted token

#define DATA_RES_CRC_ERROR 0X0B

write data crc error token

#define DATA_RES_MASK 0X1F

mask for data response tokens after a write block operation

#define DATA_RES_WRITE_ERROR 0X0D

write data programming error token

#define DATA_START_BLOCK 0XFE

start data token for read or write

#define MISO 12

spi master input, slave output pin

#define MOSI 11

spi master output, slave input pin

#define R1_IDLE_STATE 1

status for card in the idle state

#define R1_READY_STATE 0

status for card in the ready state

#define SCK 13

spi serial clock pin

#define SS 10

Slave Select pin for card

Function Documentation

uint8_t spiRec (void) `[inline]`

Receive a byte from the card

void spiSend (uint8_t *b*) `[inline]`

Send a byte to the card

void spiSSHigh (void) `[inline]`

Set Slave Select high

void spiSSLow (void) `[inline]`

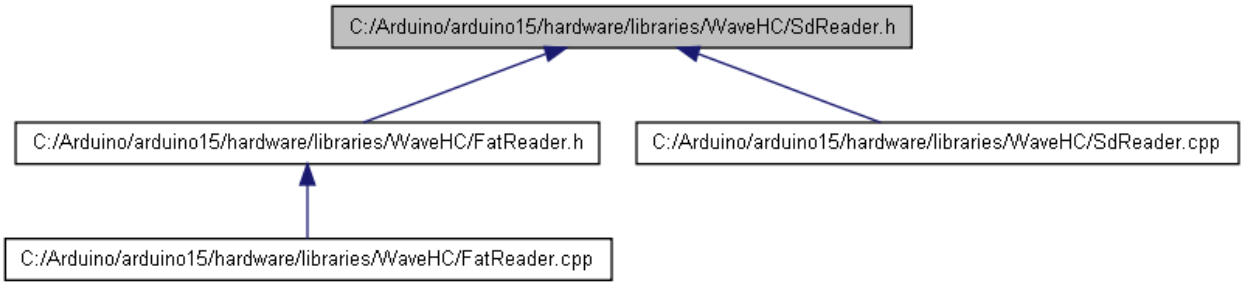
Set Slave Select low

C:/Arduino/arduino15/hardware/libraries/WaveHC/SdReader.h File Reference

```
#include "SdInfo.h"
```

Include dependency graph for SdReader.h:

This graph shows which files directly or indirectly include this file:



Classes

- class **SdReader**
  Hardware access class for SD flash cards.

Defines

- #define **ACMD41** 0X29
- #define **CMD0** 0X00
- #define **CMD10** 0X0A
- #define **CMD17** 0X11
- #define **CMD55** 0X37
- #define **CMD58** 0X3A
- #define **CMD8** 0X08

- #define **CMD9** 0X09
- #define **SD_CARD_ERROR_ACMD41** 0X6
- #define **SD_CARD_ERROR_BAD_CSD** 0X7
- #define **SD_CARD_ERROR_CMD0** 0X1
- #define **SD_CARD_ERROR_CMD17** 0X3
- #define **SD_CARD_ERROR_CMD24** 0X4
- #define **SD_CARD_ERROR_CMD58** 0X5
- #define **SD_CARD_ERROR_CMD8** 0X2
- #define **SD_CARD_ERROR_READ** 0X10
- #define **SD_CARD_ERROR_READ_REG** 0X8
- #define **SD_CARD_ERROR_WRITE** 0X20
- #define **SD_CARD_ERROR_WRITE_BLOCK_ZERO** 0XA
- #define **SD_CARD_ERROR_WRITE_TIMEOUT** 0X9
- #define **SD_CARD_INFO_SUPPORT** 1
- #define **SD_CARD_TYPE_SD1** 1
- #define **SD_CARD_TYPE_SD2** 2
- #define **SD_CARD_TYPE_SDHC** 3

Define Documentation

#define ACMD41 0X29

SD_SEND_OP_COMD - Sends host capacity support information and activates the card's initialization process

#define CMD0 0X00

GO_IDLE_STATE - init card in spi mode if CS low

#define CMD10 0X0A

SEND_CID - read the card identification information (CID register)

#define CMD17 0X11

READ_BLOCK - read a single data block from the card

#define CMD55 0X37

APP_CMD - escape for application specific command

#define CMD58 0X3A

READ_OCR - read the OCR register of a card

#define CMD8 0X08

SEND_IF_COND - verify SD Memory Card interface operating condition.

#define CMD9 0X09

SEND_CSD - read the Card Specific Data (CSD register)

#define SD_CARD_ERROR_ACMD41 0X6

card's ACMD41 initialization process timeout

#define SD_CARD_ERROR_BAD_CSD 0X7

card returned a bad CSR version field

#define SD_CARD_ERROR_CMD0 0X1

timeout error for command CMD0

#define SD_CARD_ERROR_CMD17 0X3

card returned an error response for CMD17 (read block)

#define SD_CARD_ERROR_CMD24 0X4

card returned an error response for CMD24 (write block)

#define SD_CARD_ERROR_CMD58 0X5

card returned an error response for CMD58 (read OCR)

#define SD_CARD_ERROR_CMD8 0X2

CMD8 was not accepted - not a valid SD card

#define SD_CARD_ERROR_READ 0X10

card returned an error token instead of read data

#define SD_CARD_ERROR_READ_REG 0X8

read CID or CSD failed

#define SD_CARD_ERROR_WRITE 0X20

card returned an error token as a response to a write operation

#define SD_CARD_ERROR_WRITE_BLOCK_ZERO 0XA

attempt to write protected block zero

#define SD_CARD_ERROR_WRITE_TIMEOUT 0X9

timeout occurred during write programming

#define SD_CARD_INFO_SUPPORT 1

Optional readCID(), readCSD() and cardSize() if nonzero

#define SD_CARD_TYPE_SD1 1

Standard capacity V1 SD card

#define SD_CARD_TYPE_SD2 2

Standard capacity V2 SD card

#define SD_CARD_TYPE_SDHC 3

High Capacity SD card