tinyFAT

Arduino SD card library

Manual

Introduction:

This library has been made to provide basic functionallity for reading from and writing to SD/MMC cards using Arduino boards.

As this library originally was made because I wanted to learn more about filesystems and how they work, and the fact that getting hold of SD/microSD cards that are supported by the library (2GB or smaller) is getting harder and harder this library will not get any further updates.
This also means that there will not be any support for more microcontrollers or development boards than there currently is. So no support for Arduino Due, Teensy or chipKit will be added.

You can always find the latest version of the library at **http://www.RinkyDinkElectronics.com/**

For version information, please refer to **version.txt**.

REQUIREMENTS:

The library require the following connections:

Signal	SD card pin	Arduino pin[1]	Arduino Mega pin
SCK	5	D13	D52
MISO	7	D12	D50
MOSI	2	D11	D51
SS	1	Selectable	Selectable

[1] All boards with pinout like the Arduino Duemilanove / Arduino UNO

STRUCTURES:

file.buffer[];

Buffer used for reading and writing SD-card sectors.

Variables:

```
file.buffer[0-511]:  Byte-array to hold one sector of data.
```

file.MBR;

Master Boot Record of the SD card.
The data is available, but you should never have to use it.

Variables:

```
part1Type:  Partition1 Type. Only types 0x04, 0x06 and 0x86 can be used.
part1Start: First sector of Partition1.
part1Size:  Number of sectors in Partition1.
```

file.BS;

Boot Sector of Partition1.
The data is available, but you should never have to use it.

Variables:

```
sectorsPerCluster:      Number of sectors per cluster.
reservedSectors:        Number of reserved sectors.
fatCopies:              Number of File Allocation Tables in partition. Almost always 2.
rootDirectoryEntries:   Maximum number of root directory entries.
totalFilesystemSectors: Total number of sectors available to the file system.
sectorsPerFAT:          Sectors per File Allocation Table.
hiddenSectors:          Number of hidden sectors preceding the partition that contains this FAT
volume.
partitionSerialNum:     Partition serial number.
fat1Start:              First sector of primary File Allocation Table.
fat2Start:              First sector of secondary File Allocation Table.
partitionSize:          Size of partition in MB.
```

file.DE;

Directory Entry structure. Used by findFirstFile() and findNextFile().

Variables:

```
filename:     Char array containing the file's name.
fileext:      Char array containing the file's extension.
attributes:   File attributes.
time:         File creation time (encoded).
date:         File creation date (encoded).
startCluster: First cluster of file data.
fileSize:     File size in bytes.
```

DEFINED LITERALS:

Errors	
NO_ERROR:	0x00
ERROR_NO_MORE_FILES:	0x01
ERROR_FILE_NOT_FOUND:	0x10
ERROR_ANOTHER_FILE_OPEN:	0x11
ERROR_NO_FILE_OPEN:	0x12
ERROR_MBR_READ_ERROR:	0xF0
ERROR_MBR_SIGNATURE:	0xF1
ERROR_MBR_INVALID_FS:	0xF2
ERROR_BOOTSEC_READ_ERROR:	0xE0
ERROR_BOOTSEC_SIGNATURE:	0xE1
ERROR_NO_FILE_OPEN:	0xFFF0
ERROR_WRONG_FILEMODE:	0xFFF1
FILE_IS_EMPTY:	0xFFFD
BUFFER_OVERFLOW:	0xFFFE
EOF:	0xFFFF

Filemode	
FILEMODE_BINARY:	0x01
FILEMODE_TEXT_READ:	0x02
FILEMODE_TEXT_WRITE:	0x03

SPI Speed	
SPISPEED_LOW:	0x03
SPISPEED_MEDIUM:	0x02
SPISPEED_HIGH:	0x01
SPISPEED_VERYHIGH:	0x00

FUNCTIONS:

file.initFAT(spispeed);

Initialize the interface, and connect to the SD card.

Parameters: spispeed: <*optional*>
SPISPEED_LOW
SPISPEED_MEDIUM
SPISPEED_HIGH (Default)
SPISPEED_VERYHIGH

Returns: Result as a byte.

Usage: res = file.initFAT(); // Try to connect to the SD card.

Notes: If you experience strange behaviour, or are having problems accessing the SD card you should try lowering the spi-speed.
I could never get SPISPEED_VERYHIGH to work, but it might be because all my SD card interfaces use resistor levelshifters.

file.findFirstFile(DEstruct);

Find information about the first file in the root directory.

Parameters: DEstruct: Directory Entry structure to fill

Returns: Result as a byte.

Usage: res = file.findFirstFile(&file.DE); // Get information.

file.findNextFile(DEstruct);

Find information about the next file in the root directory.

Parameters: DEstruct: Directory Entry structure to fill

Returns: Result as a byte.

Usage: res = file.findNextFile(&file.DE); // Get information.

Notes: Use findFirstFile() before using findNextFile().

file.openFile(filename, filemode);

Open a file for reading.

Parameters: filename: Name of the file to open.
filemode: <*optional*>
FILEMODE_BINARY – For reading binary files (Default)
FILEMODE_TEXT_READ - For reading text-files
FILEMODE_TEXT_WRITE – For writing text-files

Returns: Result as a byte.

Usage: res = file.openFile(”DATA.DAT”); // Attempt to open DATA.DAT for binary reading

Notes: There can only be one file open at any time.

file.readBinary();

Read the next sector of an open binary file.

Parameters: None

Returns: Result as a word

Usage: res = file.readBinary(); // Attempt to read the next sector of an already opened file

Notes: If read is successful the data will be available through file.buffer[]
The result will contain the number of bytes returned in the buffer. It will be 512 if a full sector was read, and less if the end of the file was encountered during the read.
Result can also be FILE_IS_EMPTY, ERROR_NO_FILE_OPEN or ERROR_WRONG_FILEMODE.

file.readLn(buffer, bufSize);

Read the next line of text from an open text-file.

Parameters: buffer: charArray to put the next line of text into
bufSize: size of buffer in bytes

Returns: Result as a word.
The result will be the length of the textline that are returned.
If the buffer was to small it will be filled with all the text it could contain, and result will be BUFFER_OVERFLOW. It the end of the file was reached during the read result will be EOF.
Result can also be FILE_IS_EMPTY, ERROR_NO_FILE_OPEN or ERROR_WRONG_FILEMODE.

Usage: res = file.readLn(st, 80); // Attempt to read the line of text and return it in st

file.writeLn(text);

Append a line of text to a text-file.

Parameters: text: Char array of text to append to the open file.
Returns: Result as a word.
Result can be NO_ERROR, ERROR_NO_FILE_OPEN or ERROR_WRONG_FILEMODE.
Usage: res = file.writeLn(”Some Text”); // Append text to the end of a file
Notes: CR + LF will be added to the text written to the file.
The line of text will always be added to the end of the existing text.

file.closeFile();

Close the currently open file.

Parameters: None
Returns: Nothing
Usage: file.closeFile(); // Close the open file

file.exists(filename);

Check if a file exists.

Parameters: filename: Name of file to check if exists
Returns: TRUE if file exists, else FALSE.
Usage: Res = file.exists(”SOMEFILE.DAT”); // Check if ”SOMEFILE.DAT” exists

file.rename(from-name, to-name);

Rename a file.

Parameters: from-name: Name of existing file to rename
to-name: New name for the file
Returns: TRUE if successful, else FALSE
Usage: file.rename(”OLDNAME.DAT”, ”NEWNAME.DAT”); // Rename a file from ”OLDNAME.DAT” to ”NEWNAME.DAT”

file.delFile(filename);

Delete a file.

Parameters: filename: Name of file to delete
Returns: TRUE if successful, else FALSE
Usage: file.delFile(”OLDFILE.BIN”); // Delete ”OLDFILE.BIN”

file.create(filename);

Create a new, empty file.

Parameters: filename: Name of file to create
Returns: TRUE if successful, else FALSE
Usage: file.create(”NEWFILE.TXT”); // Create ”NEWFILE.TXT”

file.setSSpin(pin);

Select which pin to use for the SS signal.

Parameters: pin: Arduino pin number
Returns: Nothing
Usage: file.setSSpin(10); // Use Arduino pin D10 as SPI SS signal pin
Notes: This must be set before calling file.initFAT()
The SS pin will default to the hardware SS pin if this function is not called.