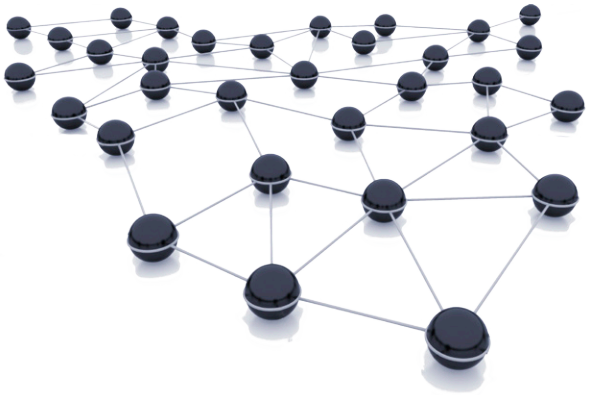
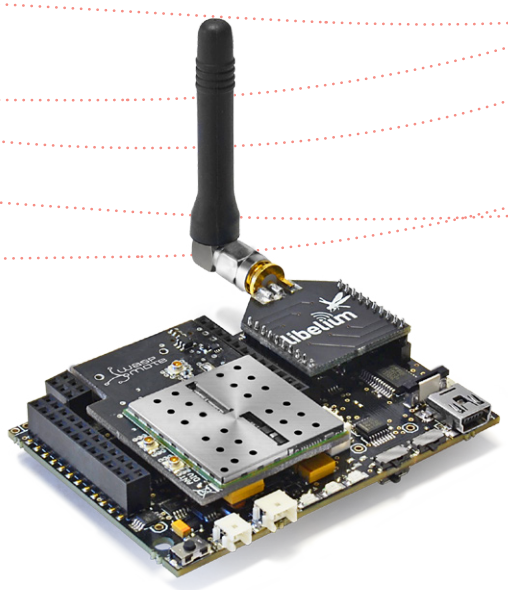
Waspmote SD CARD

Programming Guide





libelium

wasp mote

Document Version: v7.1 - 02/2018

INDEX

Note: *There are many SD card models. Any of them has defective blocks, which are ignored when using the Waspmote's SD library. However, when using OTA, those SD blocks cannot be avoided, so that the execution could crash. Libelium implements a special process to ensure the SD cards we provide will work fine with OTA. The only SD cards that Libelium can assure that work correctly with Waspmote are the SD cards we distribute officially.*

Note: *Make sure Waspmote is switched off before inserting or removing the SD card. Otherwise, the SD card could be damaged.*

Note: *Waspmote must not be switched off or reseted while there are ongoing read or write operations in the SD card. Otherwise, the SD card could be damaged and data could be lost. If you suspect that there may be some ongoing SD operations, wait a while until they are completed.*

Tip: *you can use one programmable LED to signal when the SD card is being processed.*

1. Introduction

This guide explains the SD card features and functions. There are no great variations in this library for our new product lines Waspmote v15 and Plug & Sense! v15, released on October 2016.

Anyway, if you are using previous versions of our products, please use the corresponding guides, available on our Development website.

You can get more information about the generation change on the document "New generation of Libelium product lines".

1.1. Waspmote Libraries

1.1.1. Waspmote SD files

WaspSD.h ; WaspSD.cpp

Other utilities inside the sd_utilities subdirectory: ufstream.h, ostream.cpp, SdBaseFile.h, SdFile.cpp, SdSpi.h , ios.h, ostream.h, SdFatConfig.h, SdFile.h, SdStream.cpp , iostream.h, Sd2Card.cpp, SdFat.cpp, SdInfo.h, SdStream.h , istream.cpp, Sd2Card.h, SdFat.h, SdSpiArduino.cpp, SdVolume.cpp , istream.h, SdBaseFile.cpp, SdFatStructs.h, SdSpiAVR.cpp, SdVolume.h

1.1.2. Constructor

To start using Waspmote SD library, an object from class 'WaspSD' must be created. This object, called 'SD', is created inside Waspmote SD library and it is public to all libraries. It is used through the guide to show how Waspmote SD library works.

When creating this constructor, no variables are initialized by default.

1.1.3. Flag

A flag to indicate if there have been any problem during execution of a function has been created. This flag shows the state of the SD card during initialization and operation. Possible values are:

- 0 : nothing failed, all processes have been executed properly.
- 1 : no SD card in the slot.
- 2 : initialization failed.
- 4 : volume partition failed.
- 8 : root failed.
- 16 : truncated data. Data length is bigger than buffer length, so data will be truncated to fit the buffer.
- 32 : error when opening a file.
- 64 : error when creating a file.
- 128 : error when creating a directory.
- 256 : error when writing to a file.

1.1.4. Error messages

When executing some functions, a string is returned explaining the state of SD card. Possible messages are:

- “no SD” : SD has not been found in the card slot
- “Invalid filename”: An invalid name for a file or a directory. It must respect the “8.3 filename” format (also called “short filename” or “SFN”)

1.1.5. Buffer

Due to memory restrictions, a buffer has been created to limit the length of the data managed by Waspmote API libraries when working with SD cards. Buffer size has been set to 256Bytes due to it is a sufficient value to manage strings and it occupies little memory.

This limit must be considered when developing applications, using the flag previously explained to know when data has been truncated. Obviously this does not mean you can handle 256B only, but you have to make writings and readings of this size.

1.2. Formatting SD card

When formatting the SD card before starting using Waspmote, there are some considerations to have in mind. The most important matter to know before formatting an SD card is setting the right size of the allocation tables to address the card properly. Depending on the SD card used FAT16 (2 GB) or FAT32 (8 GB or more) will be needed.

1.3. Short filename format

An 8.3 filename (also called a short filename or SFN) is a filename convention which is followed by the SD card library. 8.3 filenames have at most eight characters, optionally followed by a “.” character and a filename extension of at most three characters.

Only upper-case letters A–Z are valid. When using lower-case letters a-z, these are converted to upper-case letter.

Besides, illegal characters for directories and filenames include the following:

| < > ^ + = ? / [ ] ; , * \ “ \ \

Example of valid and invalid filenames:

FILE.TXT → valid
FILENAME → valid
FILENAME.TXT → valid
FOLDER → valid
SUBFOLDER → invalid (more than 8 characters)
FILENAME1.TXT → invalid (more than 8 characters before '.')
FILENAME.AAAA → invalid (more than 3 characters after '.')
file.txt → valid (but it will be interpreted as FILE.TXT)

2. Initialization

Before start using the SD card, it needs to be initialized. This process checks if an SD card is present on the slot, initializes SPI bus, opens partition and opens root directory.

2.1. Initializing a card

The following function checks if an SD card is present in the slot, sets the microcontroller pin which powers the SD card up, initializes the SPI bus, opens the FAT volume partition and opens the root directory. It returns nothing but it updates the flag with an error code indicating the possible error messages.

Example of use:

```
{
   SD.ON(); // Set SD card on
}
```

Available Information

`SD.flag` → stores the error code indicating the state of SD initialization process.

The SD card cannot be removed or inserted without powering off Waspmote. If a SD card is removed, the initialization function should be called to initialize the card again. Before that, the SD Card should be closed using functions explained in section “Closing a card”.

2.2. Closing a card

Closes the root directory and the SPI bus. It also switches off the microcontroller pin that powers the SD card. It returns nothing and it does not change the flag value.

Example of use:

```
{
   SD.OFF(); // Powers SD card down
}
```

2.3. SD present

It reads the associated pin to know if there is an SD in card slot.

It returns ‘1’ if SD card is present and ‘0’ if not.

If SD is not present, it closes card to avoid problems with pointers.

Example of use:

```
{
    uint8_t present;

    // Reads associated pin to know if there is a SD in card slot
    present=SD.isSD();
}
```

Available Information

present→ stores ‘1’ if SD card is detected and ‘0’ if not.

3. Disk operations

When SD has been initialized properly, pointers can be used to access partition, file system and root directory. There are some functions that return information about the SD card.

3.1. Disk information

Stores all the data containing the disk info into the buffer. It returns a filled buffer if success on getting disk info and an empty string if not.
Example of use

```
{
  // Gets disk info, returning it and storing this info in 'SD.buffer'
  SD.print_disk_info();

  USB.println(SD.buffer);
}
```

Available Information

`SD.buffer` → stores the data received as a human-readable encoded string.
`diskInfo` → pointer to `SD.buffer`

An example of the output by this system would be:

```
manuf: 0x1b
oem:   SM
prod:  21e7
rev:   1.0
serial: 0xedb6c604
date:  11/10
```

3.2. Disk Size

Gets the total size of the SD card.

It returns 'diskSize' variable and updates its value. Size is stored and returned in Bytes.

Example of use:

```
{
  // Get total size of SD card
  diskSize = SD.getDiskSize();

  USB.println(SD.diskSize);
}
```

Related Variables:

SD.diskSize → stores the total size of the SD card.

An example of the value would be: SD.diskSize=992608256

4. Directory operations

To organize an SD card, it is possible to create and manage directories. There are some functions related with directories.

4.1. Creating a directory

The `mkdir()` function creates a directory given as a valid directory path (according to short filename format) in the current working directory. The root directory is the default directory each time SD card is initialized.

It returns '1' on creation and '0' on error, activating the flag too.

If a directory name already exists, it will occur an error and the flag will be activated.

Example of use:

```
{

   boolean dirCreation;
   char name[] = "FOLDER1";
   char path[] = "FOLDER3/FOLDER4/FOLDER5";

   // creates a directory in the current directory called "FOLDER1"
   dirCreation = SD.mkdir(name);

   // creates a directory in the current directory called "FOLDER2"
   dirCreation = SD.mkdir("FOLDER2");

   // creates a three-directory path in the current directory
   dirCreation = SD.mkdir(path);
}
```

Creating and deleting directories example:
http://www.libelium.com/development/waspmote/examples/sd-05-create-delete-directories

Note 1: *All directory names must be defined according to 8.3 short filename format (see section "Short filename format")*

Note 2: *Be careful when calling this function to create a directory. If it is interrupted, the directory results damaged and it is necessary to delete it as a regular file using* `SD.del()`

4.2. Deleting a directory

Empty directories

The `rmdir()` function deletes the empty directory specified as input. The directory file will be removed only if it is empty and is not the root directory.

It returns '1' if the directory has been erased properly and '0' if error.

Note: *It allows erasing a complete path of directories always they are empty.*

Example of use:

```
{

   const char name[] = ”FOLDER”;
   char path[] = ”FOLDER3/FOLDER4/FOLDER5”;
   uint8_t delState;

   // deletes the directory in the current directory called “FOLDER”
   delState = SD.rmdir(name);

   //deletes the directory in the current directory called “FOLDER2”
   delState = SD.rmdir (“FOLDER2”);

   //deletes a three-empty-directory path in the current directory
   dirCreation = SD.rmdir(path);
}
```

Creating and deleting directories example:
http://www.libelium.com/development/waspmote/examples/sd-05-create-delete-directories

Non-empty directories

The `rmRfDir()` function deletes the non-empty directory specified as input and all contained files. It returns ‘1’ if the directory has been erased properly and ‘0’ if error.

Example of use:

```
{
   const char* name = ”FOLDER”;
   char* path = ”FOLDER3/FOLDER4/FOLDER5”;
   boolean delState;

   // deletes the directory in the current directory called “FOLDER”
   delState = SD.rmRfDir(name);

   // deletes the directory in the current directory called “FOLDER2”
   delState = SD.rmRfDir (“FOLDER2”);

   // deletes a three-directory path in the current directory
   dirCreation = SD.rmRfDir (“FOLDER3”);
}
```

4.3. Directory listing

The `ls()` function prints through the USB port the contents of the current working directory. It is possible to introduce three different flags which may be an inclusive OR of:

`LS_DATE` - Print file modification date

`LS_SIZE` - Print file size.

`LS_R` - Recursive list of subdirectories.

It returns nothing. The information is printed through the USB port.

Example of use:

```
{
   // lists the name of all the files of the directory indicating the size of the files
   SD.ls();

   // lists the name of all files of the directory indicating the size of the files
   SD.ls(LS_SIZE);

   // lists the name of all files of the directory indicating the date of the files
   SD.ls(LS_DATE);

   // lists the name of all files of the directory and all subdirectories
   SD.ls(LS_R);

   //lists the name of the files recursively indicating size and date
   SD.ls(LS_R|LS_DATE|LS_SIZE);
}
```

An example of the output for `SD.ls(LS_R|LS_DATE|LS_SIZE);` would be:

```
FILE8          1980-01-01 00:00:00 204
FILE2          1980-01-01 00:00:00 2754
FOLDER/        2000-01-01 01:00:00
SUBFOLD/       2000-01-01 01:00:00
FILE.TXT       2012-06-11 11:58:10 811
```

Listing directories example:
http://www.libelium.com/development/waspmote/examples/sd-06-list-files

4.4. Finding a directory

The `isDir()` function finds a sub-directory in the current directory. If it exists and it is a directory '1' will be returned, '0' will be returned if it exists but it is not a directory and '-1' will be returned if it does not exist.

Example of use:

```
{
   uint8_t isdir;
   const char name[] = ”FOLDER”;

   // tests existence of “FOLDER” in the current directory
   isdir = SD.isDir(name);

   // tests existence of “FOLDER” in the current directory
   isdir = SD.isDir(“FOLDER”);
}
```

4.5. Number of files

The `numFiles()` function gets the amount of files and subdirectories in the current directory. It returns the number of files or directories found, or zero if there are no files or directories. It does not count '.' and '..' directories, so if there are no directories or files in the current directory, zero will be returned.

If an error occurs, a negative number is returned.

Example of use:

```
{
   int8_t numfiles;

   // returns the number of files in the current directory
   numfiles = SD.numFiles();
}
```

4.6. Changing directory

The `cd()` functionchanges the current working directory pointer to the directory given as a parameter. It returns '0' if error, and '1' if not.

Note: *In root directory it has no sense changing directory to '..', so function will return error when doing that.*

Example of use

```
{

   uint8_t cdState;
   const char command[] = "FOLDER";

   // Change to directory specified in 'command'
   cdState = SD.cd(command);

   // Go one directory up
   cdState = SD.cd("..");
}
```

Change current working directory example:
http://www.libelium.com/development/waspmote/examples/sd-08-change-directories

Go directly to root directory

The `goRoot()` function permits to go to root directory directly. It returns '1' when ok, '0' when error.

Example of use:

```
{
   uint8_t cdState;

   // define the directory path to change
   char path[] = "/FOLD1/FOLD2/FOLD3/FOLD4/FOLD5";

   // Change to 'fold5' directory specified in 'path'
   cdState = SD.cd(path);

   // Go to root directory
   cdState = SD.goRoot();
}
```

5. File operations

To store data, files can be created and managed. There are some functions related to files operations.

5.1. Creating files

The `create()` function creates a file.

It returns '1' on file creation and '0' if error and it will mark the flag too.

Example of use:

```
{
   const char name[] = "FILE.TXT";
   boolean fileCreation;

   // It creates a file named "FILE.TXT"
   fileCreation = SD.create(name);

   // It creates a file named "FILE.TXT"
   fileCreation = SD.create("FILE.TXT");
}
```

Creating and deleting files example:

http://www.libelium.com/development/waspmote/examples/sd-01-create-delete-file

Note 1:*All file names must be defined according to 8.3 short filename format (see section "Short filename format")*

Note 2: *The maximum number of files which can be created in the root directory are 341, while a 2nd level directory can store 1.000+ files. Thus, in the case the user needs to create hundreds of files in an SD card, it is highly advised to create them inside a 2nd level directory.*

5.2. Deleting files

The `del()` function deletes a file in the current directory. It returns '1' on file delete and '0' if error.

Example of use:

```
{
   const char name[] = "FILE.TXT";
   boolean fileDelete;

   // It deletes a file named "FILE.TXT"
   fileDelete = SD.del(name);

   // It deletes previously created file named "FILE.TXT"
   fileDelete = SD.del("FILE.TXT");
}
```

Creating and deleting files example:

http://www.libelium.com/development/waspmote/examples/sd-01-create-delete-file

5.3. Opening files

The `openFile()` function opens the filepath if available. It is possible to select a bitwise-inclusive OR of flags from the following list:

`O_READ` - Open for reading.
`O_WRITE` - Open for writing.
`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` - Synchronous writes.

Returns '1' on success, '0' otherwise

Example of use:

```
{
   const char filepath[] = ”FILE.TXT”;

   // declare an SdFile object
   SdFile file;

   // open “FILE” for reading
   SD.openFile( filepath, &file, O_READ);
}
```

Note: *All file names must be defined according to 8.3 short filename format (see section "Short filename format")*

5.4. Closing files

The `closeFile()` function closes the pointer which pointed to the previously defined file.

Note: *If a file is opened with previous function and it is not closed before using another file function, it will not work properly. Only one file pointer can be managed at the same time.*

Example of use:

```
{
   // previously declared file
   SdFile file;

   // close file, referencing the previously opened file
   SD.closeFile(&file);
}
```

5.5. Finding files

The `isFile()` function finds a file path in the current directory.

If it exists and it is a file '1' will be returned, '0' will be returned if it exists but it is not a file and '-1' will be returned if it does not exist.

Example of use:

```
{
   const char* name = "FILE.TXT";
   int8_t fileFound;

   // looks for "FILE.TXT" in the current directory
   fileFound = SD.isFile(name);

   // looks for "FILE.TXT" in the current directory
   fileFound = SD.isFile("FILE.TXT");
}
```

5.6. Reading data

The `cat()` function stores into the buffer the amount of bytes indicated as input. This function needs three input parameters:

- **filename:** string which defines the filename
- **offset:** number of bytes skipped from the beginning of the file to start reading
- **scope:** number of bytes to read from file from the offset indicated

The `catln()` function stores into the buffer the amount of lines indicated as input. This function needs three input parameters:

- **filename:** string which defines the filename
- **offset:** number of lines skipped from the beginning of the file to start reading
- **scope:** number of lines to read from file from the offset indicated

The information is returned as a string where each one of the characters are printed one after the next, EOL ('\n') will be encoded as EOL, and will be accounted as one byte.

Note: *There is a limitation in size, due to buffer size. If the data read was bigger than that, the function will include the characters ">>" at the end and activate the* `TRUNCATED_DATA` *value in the flag. It is recommended to check this value to ensure data integrity.*

If 'offset' or 'scope', or both of them, are greater than file size, there will be no error but only possible data will be copied into buffer.

Example of use

```
{
   const char* name=”FILE.TXT”;

   //It stores in ‘SD.buffer’ 17 characters after jumping 3
   SD.cat(name,3,17);

   //It stores in ‘SD.buffer’ 100 characters from the beginning
   SD.cat(name,0,100);

   //It stores in ‘SD.buffer’ 3 lines after jumping over the 2 first
   SD.catln(name,2,3);
}
```

Reading files example:

http://www.libelium.com/development/waspmote/examples/sd-04-read-file

5.7. Writing data

There are several ways to write data to a file:

- Indicating the position to start writing. Function `writeSD()`. It is possible to indicate the amount of bytes to write.
- At the end of the file. Function `append()`. It is possible to indicate the amount of bytes to write.
- At the end of the file including an EOL character. Function `appendln()`

It returns ‘1’ on success and ‘0’ if error.

Note: *Due to buffer size, 256Bytes is the limit for writing data into a file at once. If more data needs to be written, it will have to be divided in blocks of 256Bytes.*

Example of use:

```
{
   const char* file = ”FILE.TXT”;
   uint8_t writeState;

   // It writes “hello” on file at position 0
   writeState = SD.writeSD(file,”hello”,0);

   // It writes “hello” on file at position 5
   writeState = SD.writeSD(file,”hello”,5);

   // It writes “hel” on file at position 10
   writeState = SD.writeSD(file,”hello”,10,3);

   // It writes “hello” at the end of file
   writeState = SD.append(file,”hello”);

   // It writes “hel” on file at end of file
   writeState = SD.append(file,”hello”,3);

   // It writes “hello” at end of file with EOL
   writeState = SD.appendln(file,”hello”);
}
```

Writing and appending data examples:

http://www.libelium.com/development/waspmote/examples/sd-02-write-file

http://www.libelium.com/development/waspmote/examples/sd-03-append-file

http://www.libelium.com/development/waspmote/examples/sd-07-datalogger

5.8. Number of lines

The `numln()` function counts the number of lines in a file.

The number of lines are counted as the number of '\n' that are found in a file.

It returns the amount of lines and negative value if an error occurred. If there are no lines in file selected, zero will be returned.

Example of use:

```
{
   const char name[] = "FILE.TXT";
   int32_t numberLines;

   // counts number of lines in file named "FILE.TXT"
   numberLines = SD.numln(name);

   // counts number of lines in file named "FILE.TXT"
   numberLines = SD.numln("FILE.TXT");
}
```

5.9. Getting file size

The `getFileSize()` function gets the size of the selected file.

It returns its size in Bytes or '-1' if an error occurs.

Example of use:

```
{
   const char name[] = "FILE.TXT";
   int32_t sizeFile;

   // gets size of file named "FILE.TXT"
   sizeFile = SD.getFileSize(name);

   // gets size of file named "FILE.TXT"
   sizeFile = SD.getFileSize("FILE.TXT");
}
```

Available Information

A possible value would be: `sizeFile=16`, indicating the size file is 16 Bytes.

5.10. Finding patterns

The `indexOf()` function looks into the file for the first occurrence of the pattern after a certain offset. The algorithm will jump over offset bytes before starting to search for the pattern.

It will return the amount of bytes to the pattern from the offset.

The special characters like '\n' (EOL) are accounted as one byte and files are indexed from 0.

Example of use:

```
{
   const char name[] = "FILE.TXT";
   int32_t pattern;

   // It returns position at which "11" appears on file jumping over first 17 positions
   pattern = SD.indexOf(name,"11",17);

   // It returns position at which "11" is on file
   pattern = SD.indexOf("FILE.TXT","11",0);
}
```

Example file "`FILE.TXT`" contains: `'hola caracola\nhej hej\n hola la[EOF]'`

The following table shows the results from searching different patterns:

Function	Answer
`SD.indexOf("FILE.TXT", "hola", 0);`	`0`
`SD.indexOf("FILE.TXT", "hola", 1);`	`23`
`SD.indexOf("FILE.TXT", "hej", 3)`	`11`

Finding patterns example:
http://www.libelium.com/development/waspmote/examples/sd-09-indexof-pattern

6. Code examples and extended information

In the Waspmote Development section you can find complete examples:
http://www.libelium.com/development/waspmote/examples

7. API changelog

Keep track of the software changes on this link:
www.libelium.com/development/waspmote/documentation/changelog/#SDcard

8. Certifications

Libelium offers 2 types of IoT sensor platforms, Waspmote OEM and Plug & Sense!:

- **Waspmote OEM** is intended to be used for research purposes or as part of a major product so it needs final certification on the client side. More info at: www.libelium.com/products/waspmote
- **Plug & Sense!** is the line ready to be used out-of-the-box. It includes market certifications. See below the specific list of regulations passed. More info at: www.libelium.com/products/plug-sense

Besides, Meshlium, our multiprotocol router for the IoT, is also certified with the certifications below. Get more info at:

www.libelium.com/products/meshlium

List of certifications for Plug & Sense! and Meshlium:

- CE (Europe)
- FCC (US)
- IC (Canada)
- ANATEL (Brazil)
- RCM (Australia)
- PTCRB (cellular certification for the US)
- AT&T (cellular certification for the US)

Figure 1: Certifications of the Plug & Sense! product line

You can find all the certification documents at:

www.libelium.com/certifications

9. Documentation changelog

From v7.0 to v7.1:

- Added references to FAT32 formatted SD cards