Table of Contents

Flash layout

Even though file system is stored on the same flash chip as the program, programming new sketch will not modify file system contents. This allows to use file system to store sketch data, configuration files, or content for Web server.

The following diagram illustrates flash layout used in Arduino environment:

```
|--------------|-------|---------------|--|--|--|--|--|
^              ^       ^               ^     ^
Sketch    OTA update   File system   EEPROM  WiFi config (SDK)
```

File system size depends on the flash chip size. Depending on the board which is selected in IDE, you have the following options for flash size:

Board	Flash chip size, bytes	File system size, bytes
Generic module	512k	64k, 128k
Generic module	1M	64k, 128k, 256k, 512k
Generic module	2M	1M
Generic module	4M	3M
Adafruit HUZZAH	4M	1M, 3M

ESPresso Lite 1.0	4M	1M, 3M
ESPresso Lite 2.0	4M	1M, 3M
NodeMCU 0.9	4M	1M, 3M
NodeMCU 1.0	4M	1M, 3M
Olimex MOD-WIFI-ESP8266(-DEV)	2M	1M
SparkFun Thing	512k	64k
SweetPea ESP-210	4M	1M, 3M
WeMos D1 & D1 mini	4M	1M, 3M
ESPDuino	4M	1M, 3M

Note: to use any of file system functions in the sketch, add the following include to the sketch:

```
#include "FS.h"
```

Uploading files to file system

ESP8266FS is a tool which integrates into the Arduino IDE. It adds a menu item to *Tools* menu for uploading the contents of sketch data directory into ESP8266 flash file system.

- Download the tool: https://github.com/esp8266/arduino-esp8266fs-plugin/releases/download/0.2.0/ESP8266FS-0.2.0.zip.
- In your Arduino sketchbook directory, create `tools` directory if it doesn't exist yet
- Unpack the tool into `tools` directory (the path will look like`<home_dir>/Arduino/tools/ESP8266FS/tool/esp8266fs.jar`)
- Restart Arduino IDE
- Open a sketch (or create a new one and save it)
- Go to sketch directory (choose Sketch > Show Sketch Folder)
- Create a directory named `data` and any files you want in the file system there
- Make sure you have selected a board, port, and closed Serial Monitor
- Select Tools > ESP8266 Sketch Data Upload. This should start uploading the files into ESP8266 flash file system. When done, IDE status bar will display `SPIFFS Image Uploaded` message.

File system object (SPIFFS)

begin

```
SPIFFS.begin()
```

This method mounts SPIFFS file system. It must be called before any other FS APIs are used. Returns *true* if file system was mounted successfully, false otherwise.

end

```
SPIFFS.end()
```

This method unmounts SPIFFS file system. Use this method before updating SPIFFS using OTA.

format

```
SPIFFS.format()
```

Formats the file system. May be called either before or after calling `begin`. Returns *true* if formatting was successful.

open

```
SPIFFS.open(path, mode)
```

Opens a file. `path` should be an absolute path starting with a slash (e.g. `/dir/filename.txt`). `mode` is a string specifying access mode. It can be one of "r", "w", "a", "r+", "w+", "a+". Meaning of these modes is the same as for `fopen` C function.

```
   r      Open text file for reading.  The stream is positioned at the
          beginning of the file.

   r+     Open for reading and writing.  The stream is positioned at the
          beginning of the file.

   w      Truncate file to zero length or create text file for writing.
          The stream is positioned at the beginning of the file.

   w+     Open for reading and writing.  The file is created if it does
          not exist, otherwise it is truncated.  The stream is
          positioned at the beginning of the file.

   a      Open for appending (writing at end of file).  The file is
          created if it does not exist.  The stream is positioned at the
          end of the file.

   a+     Open for reading and appending (writing at end of file).  The
          file is created if it does not exist.  The initial file
          position for reading is at the beginning of the file, but
          output is always appended to the end of the file.
```

Returns *File* object. To check whether the file was opened successfully, use the boolean operator.

```
File f = SPIFFS.open("/f.txt", "w");
if (!f) {
    Serial.println("file open failed");
}
```

exists

```
SPIFFS.exists(path)
```

Returns *true* if a file with given path exists, *false* otherwise.

openDir

```
SPIFFS.openDir(path)
```

Opens a directory given its absolute path. Returns a *Dir* object.

remove

```
SPIFFS.remove(path)
```

Deletes the file given its absolute path. Returns *true* if file was deleted successfully.

rename

```
SPIFFS.rename(pathFrom, pathTo)
```

Renames file from `pathFrom` to `pathTo`. Paths must be absolute. Returns *true* if file was renamed successfully.

info

```
FSInfo fs_info;
SPIFFS.info(fs_info);
```

Fills FSInfo structure with information about the file system. Returns `true` is successful, `false` otherwise.

Filesystem information structure

```
struct FSInfo {
    size_t totalBytes;
    size_t usedBytes;
    size_t blockSize;
    size_t pageSize;
    size_t maxOpenFiles;
    size_t maxPathLength;
};
```

This is the structure which may be filled using FS::info method.

- `totalBytes` — total size of useful data on the file system
- `usedBytes` — number of bytes used by files

- `blockSize` — SPIFFS block size
- `pageSize` — SPIFFS logical page size
- `maxOpenFiles` — max number of files which may be open simultaneously
- `maxPathLength` — max file name length (including one byte for zero termination)

Directory object (Dir)

The purpose of *Dir* object is to iterate over files inside a directory. It provides three methods: `next()`, `fileName()`, and`openFile(mode)`.

The following example shows how it should be used:

```
Dir dir = SPIFFS.openDir("/data");
while (dir.next()) {
    Serial.print(dir.fileName());
    File f = dir.openFile("r");
    Serial.println(f.size());
}
```

`dir.next()` returns true while there are files in the directory to iterate over. It must be called before calling `fileName` and`openFile` functions.

`openFile` method takes *mode* argument which has the same meaning as for `SPIFFS.open` function.

File object

`SPIFFS.open` and `dir.openFile` functions return a *File* object. This object supports all the functions of *Stream*, so you can use `readBytes`, `findUntil`, `parseInt`, `println`, and all other *Stream* methods.

There are also some functions which are specific to *File* object.

seek

```
file.seek(offset, mode)
```

This function behaves like `fseek` C function. Depending on the value of `mode`, it moves current position in a file as follows:

- if `mode` is `SeekSet`, position is set to `offset` bytes from the beginning.
- if `mode` is `SeekCur`, current position is moved by `offset` bytes.
- if `mode` is `SeekEnd`, position is set to `offset` bytes from the end of the file.

Returns *true* if position was set successfully.

position

```
file.position()
```

Returns the current position inside the file, in bytes.

size

```
file.size()
```

Returns file size, in bytes.

name

```
String name = file.name();
```

Returns file name, as `const char*`. Convert it to *String* for storage.

close

```
file.close()
```

Close the file. No other operations should be performed on *File* object after `close` function was called.