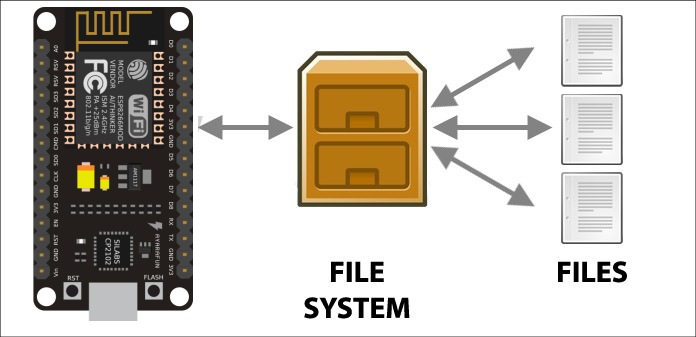
This tutorial explains in depth **ESP8266** Flash File System Called as (**SPIFFS**). There are two ways to store data on ESP8266 one is using [internal EEPROM](https://circuits4you.com/2016/12/14/internal-eeprom-esp8266/) which is of 512 Bytes but you can write data 1 millions of times (no file system). and Second is use of SPI Flash (64kBytes to 3Mbyte), when you see ESP-01 a small 8-Pin Chip is present near to the ESP8266 which is FLASH memory connected to ESP through SPI. In this flash memory ESP stores the program. Along with program you can store your files on it. Limitation of this memory is it has only 10000 (ten thousand) write cycles.

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 |
| Generic module | 1M | 64k, 128k, 256k, 512k |
| Generic module | 2M | 1M |
| Generic module | 4M | 3M |
| Adafruit HUZZAH | 4M | 1M, 3M |
| NodeMCU 0.9 | 4M | 1M, 3M |
| NodeMCU 1.0 | 4M | 1M, 3M |

**ESP8266 SPIFFS Simple File Creation and Reading, Writing Example**

Before we go into depth we see basic File creation and writing and reading.

/\*

\* ESP8266 SPIFFS Basic Reading and Writing File

\*

\*/

#include <ESP8266WiFi.h>

#include <FS.h> //Include File System Headers

const char\* filename = "/samplefile.txt";

void setup() {

delay(1000);

Serial.begin(115200);

Serial.println();

//Initialize File System

if(SPIFFS.begin())

{

Serial.println("SPIFFS Initialize....ok");

}

else

{

Serial.println("SPIFFS Initialization...failed");

}

//Format File System

if(SPIFFS.format())

{

Serial.println("File System Formated");

}

else

{

Serial.println("File System Formatting Error");

}

//Create New File And Write Data to It

//w=Write Open file for writing

File f = SPIFFS.open(filename, "w");

if (!f) {

Serial.println("file open failed");

}

else

{

//Write data to file

Serial.println("Writing Data to File");

f.print("This is sample data which is written in file");

f.close(); //Close file

}

}

void loop() {

int i;

//Read File data

File f = SPIFFS.open(filename, "r");

if (!f) {

Serial.println("file open failed");

}

else

{

Serial.println("Reading Data from File:");

//Data from file

for(i=0;i<f.size();i++) //Read upto complete file size

{

Serial.print((char)f.read());

}

f.close(); //Close file

Serial.println("File Closed");

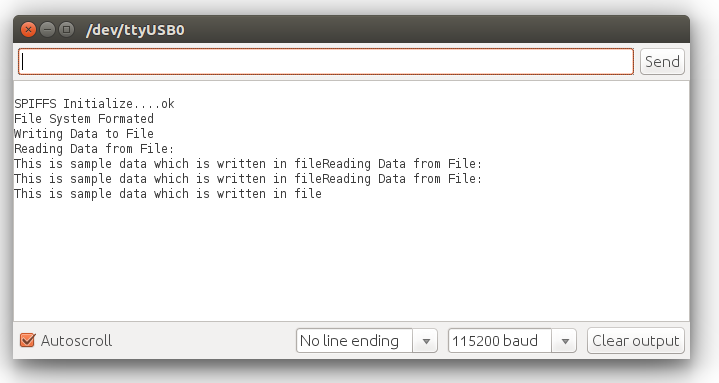
}

delay(5000);

}

**Results**

Open serial monitor and see file data writing and reading.

ESP8266 File Writing and Reading Example

**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.

**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.

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

File f = SPIFFS.open("/samplefile.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.

**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.

This way you can perform File System operations in ESP8266 and NodeMCU