The SPI Flash File System (SPIFFS)

Pieter P

SPI Flash File System

Up until now, we've always included the HTML for our web pages as string literals in our sketch. This makes our code very hard to read, and you'll run out of memory rather quickly.
If you remember the introduction, I mentioned the Serial Peripheral Interface Flash File System, or SPIFFS for short. It's a light-weight file system for microcontrollers with an SPI flash chip. The on-board flash chip of the ESP8266 has plenty of space for your webpages, especially if you have the 1MB, 2MB or 4MB version.

SPIFFS let's you access the flash memory as if it was a normal file system like the one on your computer (but much simpler of course): you can read and write files, create folders ...

The easiest way to learn how to use SPIFFS is to look at some examples. But a file server with no files to serve is pretty pointless, so I'll explain how to upload files to the SPIFFS first.

Uploading files to SPIFFS

To select the right files to upload, you have to place them in a folder called *data*, inside the sketch folder of your project: Open your sketch in the Arduino IDE, and hit CTRL+K. Wait for a file explorer window to open, and create a new folder named *data*. Copy your files over to this folder. (Only use small files like text files or icons. There's not enough space for large photos or videos.)
Next, select all files in the folder (CTRL+A) and check the size of all files combined (don't forget subfolders). Go to the Arduino IDE again, and under Tools > Flash Size, select an option with the right flash size for your board, and a SPIFFS size that is larger than the size of your data folder.
Then upload the sketch. When that's finished, make sure that the Serial Monitor is closed, then open the *Tools* menu, and click *ESP8266 sketch data upload*. If your ESP has auto-reset and auto-program, it should work automatically, if you don't have auto-program, you have to manually enter program mode before uploading the data to SPIFFS. The procedure is exactly the same as entering program mode before uploading a sketch.

If you get an error saying `SPIFFS_write error(-10001): File system is full`, this means that your files are too large to fit into the SPIFFS memory. Select a larger SPIFFS size under Tools > Flash Size, or delete some files.
Even if your computer says that the files are smaller than the selected SPIFFS size, you can still get this error: this has to do with block sizes, and metadata like file and folder names that take up space as well.

If you change the SPIFFS size, you have to reupload your sketch, because when you change the SPIFFS size, the memory location will be different. The program has to know the updated SPIFFS address offset to be able to read the files.

SPIFFS File Server

The following example is a very basic file server: it just takes the URI of the HTTP request, checks if the URI points to a file in the SPIFFS, and if it finds the file, it sends it as a response.

```
#include <ESP8266WiFi.h>
#include <WiFiClient.h>
#include <ESP8266WiFiMulti.h>
#include <ESP8266mDNS.h>
#include <ESP8266WebServer.h>
#include <FS.h>   // Include the SPIFFS library

ESP8266WiFiMulti wifiMulti;     // Create an instance of the ESP8266WiFiMulti class, called 'wifiMulti'

ESP8266WebServer server(80);    // Create a webserver object that listens for HTTP request on port 80

String getContentType(String filename); // convert the file extension to the MIME type
bool handleFileRead(String path);       // send the right file to the client (if it exists)

void setup() {
  Serial.begin(115200);         // Start the Serial communication to send messages to the computer
  delay(10);
  Serial.println('\n');

  wifiMulti.addAP("ssid_from_AP_1", "your_password_for_AP_1");   // add Wi-Fi networks you want to connect to
  wifiMulti.addAP("ssid_from_AP_2", "your_password_for_AP_2");
  wifiMulti.addAP("ssid_from_AP_3", "your_password_for_AP_3");

  Serial.println("Connecting ...");
  int i = 0;
  while (wifiMulti.run() != WL_CONNECTED) { // Wait for the Wi-Fi to connect
    delay(250);
    Serial.print('.');
  }
  Serial.println('\n');
  Serial.print("Connected to ");
  Serial.println(WiFi.SSID());              // Tell us what network we're connected to
  Serial.print("IP address:\t");
  Serial.println(WiFi.localIP());           // Send the IP address of the ESP8266 to the computer

  if (MDNS.begin("esp8266")) {              // Start the mDNS responder for esp8266.local
```

```
    Serial.println("mDNS responder started");
  } else {
    Serial.println("Error setting up MDNS responder!");
  }

  SPIFFS.begin();                           // Start the SPI Flash Files System

  server.onNotFound([]() {                              // If the client requests any URI
    if (!handleFileRead(server.uri()))                  // send it if it exists
      server.send(404, "text/plain", "404: Not Found"); // otherwise, respond with a 404 (Not Found) error
  });

  server.begin();                           // Actually start the server
  Serial.println("HTTP server started");
}

void loop(void) {
  server.handleClient();
}

String getContentType(String filename) { // convert the file extension to the MIME type
  if (filename.endsWith(".html")) return "text/html";
  else if (filename.endsWith(".css")) return "text/css";
  else if (filename.endsWith(".js")) return "application/javascript";
  else if (filename.endsWith(".ico")) return "image/x-icon";
  return "text/plain";
}

bool handleFileRead(String path) { // send the right file to the client (if it exists)
  Serial.println("handleFileRead: " + path);
  if (path.endsWith("/")) path += "index.html";         // If a folder is requested, send the index file
  String contentType = getContentType(path);            // Get the MIME type
  if (SPIFFS.exists(path)) {                            // If the file exists
    File file = SPIFFS.open(path, "r");                 // Open it
    size_t sent = server.streamFile(file, contentType); // And send it to the client
    file.close();                                       // Then close the file again
```

As you can see, we don't use `server.on` in this example. Instead, we use `server.onNotFound` : this will match any URI, since we didn't declare any specific URI handlers like in the previous server examples.
When a URI is requested, we call the function `handleFileRead` . This function checks if the URI of the HTTP request is the path to an existing file in the SPIFFS. If that's the case, it sends the file back to the client. If the path doesn't exist, it returns false, and a 404 (Not Found) HTTP status will be sent.

The MIME type for the different files is based on the file extension.
You could add other file types as well. For instance:

```
String getContentType(String filename){
  if(filename.endsWith(".htm")) return "text/html";
  else if(filename.endsWith(".html")) return "text/html";
  else if(filename.endsWith(".css")) return "text/css";
  else if(filename.endsWith(".js")) return "application/javascript";
  else if(filename.endsWith(".png")) return "image/png";
  else if(filename.endsWith(".gif")) return "image/gif";
  else if(filename.endsWith(".jpg")) return "image/jpeg";
  else if(filename.endsWith(".ico")) return "image/x-icon";
  else if(filename.endsWith(".xml")) return "text/xml";
  else if(filename.endsWith(".pdf")) return "application/x-pdf";
  else if(filename.endsWith(".zip")) return "application/x-zip";
  else if(filename.endsWith(".gz")) return "application/x-gzip";
  return "text/plain";
}
```

This example is adapted from the FSBrowser example by Hristo Gochkov.

Compressing files

The ESP8266's flash memory isn't huge, and most text files, like html, css etc. can be compressed by quite a large factor. Modern web browsers accept compressed files as a response, so we'll take advantage of this by uploading compressed versions of our html and icon files to the SPIFFS, in order to save space and bandwidth.

To do this, we need to add the GNU zip file type to our list of MIME types:

```
String getContentType(String filename){
  if(filename.endsWith(".html")) return "text/html";
  else if(filename.endsWith(".css")) return "text/css";
  else if(filename.endsWith(".js")) return "application/javascript";
  else if(filename.endsWith(".ico")) return "image/x-icon";
  else if(filename.endsWith(".gz")) return "application/x-gzip";
  return "text/plain";
}
```

And we need to change our `handleFileRead` function as well:

```
bool handleFileRead(String path){  // send the right file to the client (if it exists)
  Serial.println("handleFileRead: " + path);
  if(path.endsWith("/")) path += "index.html";          // If a folder is requested, send the index file
  String contentType = getContentType(path);            // Get the MIME type
  String pathWithGz = path + ".gz";
  if(SPIFFS.exists(pathWithGz) || SPIFFS.exists(path)){  // If the file exists, either as a compressed archive, or normal
    if(SPIFFS.exists(pathWithGz))                         // If there's a compressed version available
```

```
      path += ".gz";                                          // Use the compressed version
    File file = SPIFFS.open(path, "r");                    // Open the file
    size_t sent = server.streamFile(file, contentType);    // Send it to the client
    file.close();                                          // Close the file again
    Serial.println(String("\tSent file: ") + path);
    return true;
  }
  Serial.println(String("\tFile Not Found: ") + path);
  return false;                                          // If the file doesn't exist, return false
}
```

Now, try compressing some of the files to the GNU zip format (.gz), and uploading them to SPIFFS. Or you can just download the new data folder (unzip it first).
Every time a client requests a certain file, the ESP will check if a compressed version is available. If so, it will use that instead of the uncompressed file. The output in the Serial Monitor should look something like this:

```
handleFileRead: /
        Sent file: /index.html.gz
handleFileRead: /main.css
        Sent file: /main.css
handleFileRead: /JavaScript.js
        Sent file: /JavaScript.js
handleFileRead: /folder/JavaScript.js
        Sent file: /folder/JavaScript.js
handleFileRead: /favicon.ico
        Sent file: /favicon.ico.gz
```

It automatically detected that it had to send the compressed versions of index.html and favicon.ico.
