Wi-Fi for EPOSMote III with the ESP8266 Module

This page is a step-by-step guide for uploading the firmware of the ESP module. The factory firmware could be used, but it can not handle secure connections properly. The solution is to create a custom firmware capable of handling secure connections and provide a UFSC self-signed certificate to the server.

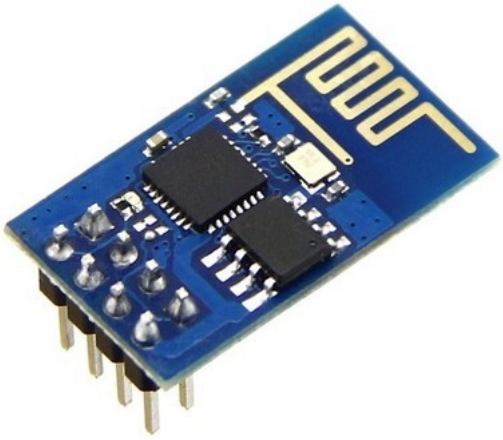
Our goal on this page is to establish a secure connection between EPOSMote3 (eMote) and the IoT Server. To achieve this, a micro-controller will be used as an interface between the eMote and the network.

This microcontroller is an ESP2866, in which this firmware looks forward to bridge EPOSMote III apps with LISHA IoT Platform. The firmware supports HTTPS, certificate-based authentication, and EDUROAM connection.

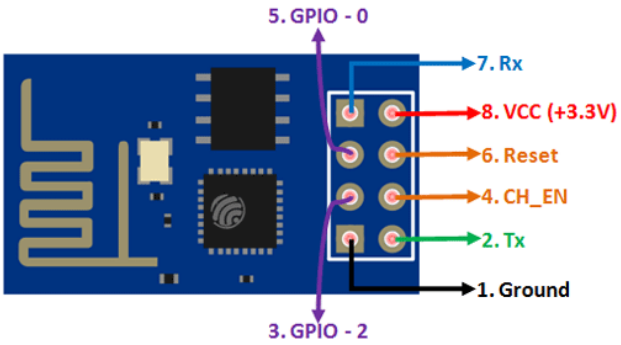
Hardware Requirements

ESP2866

An ESP2866 module is required, there are many module versions for this chip and the one used in this article is the ESP-01. Use the Hardware module versions as a reference. Version 12 of the ESP module has SPI communication pins, but due to its pin size, this version couldn't be tested.

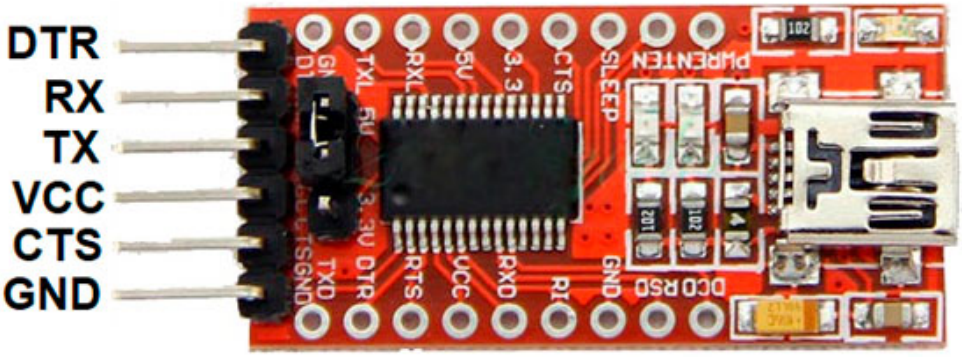


- Pinout



FTDI

To write data into the ESP, an FTDI board is required.



And the connections between the ESP and the FTDI are shown in the next sections.

Pinout for flashing Flash the ESP

It is important to note that when you are uploading the certificate files or the firmware, the GPIO0 pin on the module must be connected to GROUND. And when the code is ready for production, GPIO0 must be disconnected from the GROUND

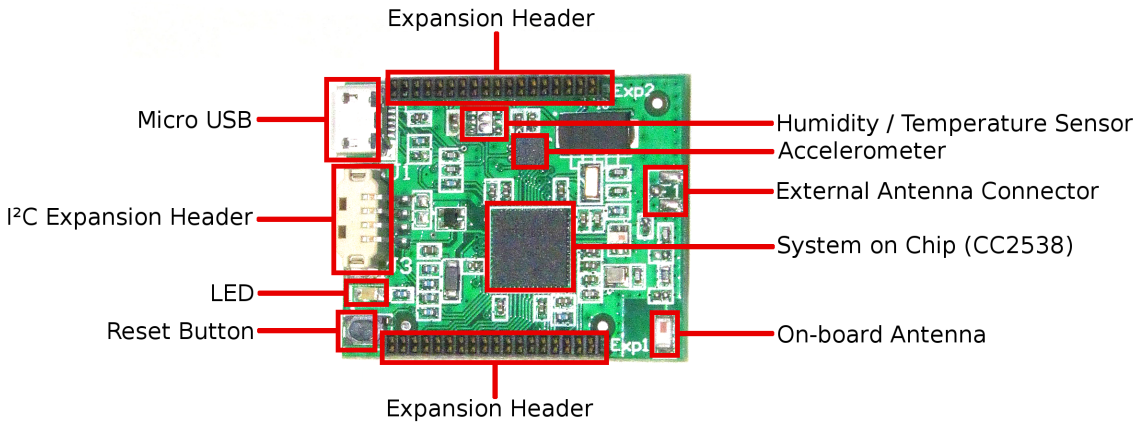
- FTDI connecting the computer and the ESP. ESP PIN configuration (FLASH boot mode):

FTDI	**ESP8266**
TX	RX
RX	TX
GROUND	GROUND and GPIO 0
VCC	VCC and RESET and CH_EN

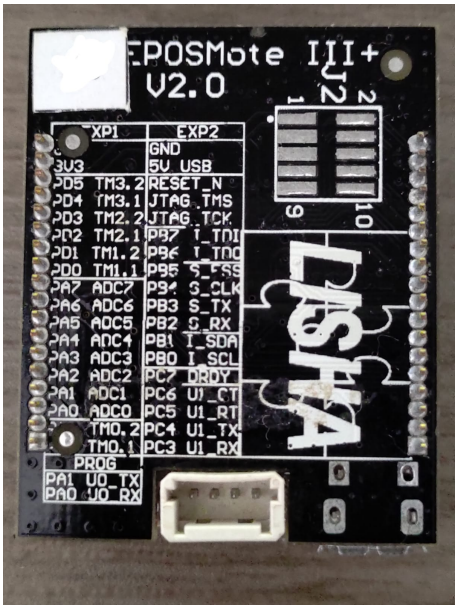
Pinout in RUN Mode (for testing the module with the computer)

To boot ESP8266 in RUN Mode, use the same configuration and disconnect GROUND from GPIO 0

eMote III



- EPOSMote III pinout:



Connecting ESP and eMote

The ESP and the eMote3 communicate through UART, using the Rx-Tx pins available on each board. Make sure you have the following connections ready:

- Pin Connection

1. ESP's **RX** pin connected to eMote's **C4** pin (UART1 TX)
2. ESP's **TX** pin connected to eMote's **C3** pin (UART1 RX)
3. ESP's **CH** and **Vin** connected to external 3v3 source (ESP module requires over 100mA to run and eMote can't provide that)
4. ESP's **GND** connected to eMote's **GND** pin and the external source's **GND**

EPOS Documentation

Users are encouraged to take a look at the complete documentation for EPOS and EPOS Mote3:

- Complete documentation for EPOS 2.2
- EPOSMote III Quick Start
- EPOSMote III User Guide
- Embedded Systems Labs with EPOS and EPOSMote
- EPOSMoteIII usage documentation
- EPOSMoteIII hardware documentation

For further assistance, please check EPOS Project at LISHA's GitLab.

Software Requirements

The firmware code was meant to be flashed into an ESP2866 through Arduino IDE.
For the firmware code to be supported by the Arduino IDE, the ESP2866 library is needed.

Arduino IDE Recommended Configuration

- Arduino IDE version used: **1.8.11**
- Tools Menu:
  - Board: "Generic ESP8266 Module"
  - Upload Speed: "115200"
  - CPU Frequency: "160 MHz"
  - Crystal Frequency: "26 MHz"
  - Flash Size: "512K (no SPIFFS)"
  - Flash Mode: "DOUT (compatible)"

- Flash Frequency: "40MHz"
- Reset Method: "ck"
- Debug Port: "Disabled"
- Debug Level: "None"
- lwIP Variant: "v2 Lower Memory"
- VTables: "Flash"
- Exceptions: "Disabled"
- Builtin Led: "2"
- Erase Flash: "Only Sketch"
- SSL Support: "All SSL ciphers(most compatible)"
- Programmer: "AVRISP mkII"

ESP's Firmware Source Code

The source code for the ESP8266 Wifi Firmware for the EPOSMote III can be found in:

- LISHA's GitLab.

This source code for the firmware is defined in a ".ino" file. The main components of the firmware are the following:

- "Configuration Section": here you must define the Wifi credentials that the ESP will connect to and also the server to which the application desires to send the messages. Moreover, it is in this section that the certificates must be copied from the content of the .pem and .key files;
- "setup()": establish the wifi connection;
- "connect_client()": Create/restart server connection "CONNECTION_TRIES_LIMIT" times;
- "post()": Sends the contents of the parsing buffer to the server;
- "parse_post()": Parse the SmartData string coming from the serial;
- "loop()": Arduino's main function. Keeps reading the serial until the beginning of a SmartData String is identified coming from the serial;

Step-by-Step Source Code Configuration:

1. Look for the "Configuration Section" inside the source code (esp_wifi_firmware.ino)
2. Modify the variables in this section as needed:
2.1. "DEBUG(x) x": Enable Debug messages, remove "x" to disable Debug messages;
2.2. IF WPA2 wifi conection is needed, set WPA2 to 1 and
2.2.1. Finish the WPA2 configuration setting the login info (ssid, password, user, pass);
2.3. IF WPA2 wifi will not be used, set WPA2 to 0 and
2.3.1 Finish the Wifi configuration setting the login info (ssid and password of wifi NOT WPA2);
2.4. Set the HTTP configuration (host address, host port, api_attach and api_put):
2.4.1. **API Attach is deprecated, configure this address to API Create**;
2.4.2. Configure the acceptable retries to host connection (0 if unlimited);
2.4.3. Configure the time out for a HTTP response from host (0 if ignored);
2.4.4. IF a certificate will be used to validate the connection with the server, set SSL_VERIFICATION to 1, else 0, and
2.4.4.1. Fill the **client_cert** and **private_key** information to fully configure SSL certificate verification;
2.4.5. IF certificates are NOT being used and a Default credential will be used for every post set DEFAULT Credentials to 1 and configure client_credentials_json accordingly;

Sending Data and Parsing

Interactions by DEFAULT are done as follows:

- API_PUT:
  - "++post(\"your smartdata here\")\0x255\0x255";

- API_CREATE (__API ATTACH is deprecated, change from API_ATTACH to API_CREATE):
  - "--post("your series here")\0x255\0x255";

Note that the — and ++ and \0x255\0x255 control characters are used during the parsing algorithm to control the state machine for serial readings. Nevertheless, they can be modified as needed by changing the variables **PUT_CONTROL_CHAR**, **ATTACH_CONTROL_CHAR**, and **END_CONTROL_CHAR**. Also the methods **loop()** and **parse_post()** are the ones that configure the serial parsing state machine and can be modified as aswell to attain the application needs.

Old Versions of the firmware (**Deprecated**)

Note: read this article on MicroPython for the ESP8266 before continuing!

Installation and Setup

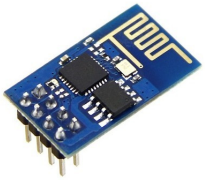
Software Requirements

The following list shows what you need to install/setup/prepare:

1. Install the Arduino IDE. The version used here is **v1.8.3**
2. Install the ESP8266 library on your system. The tutorial can be found at https://github.com/esp8266/Arduino . The version used for this tutorial is **v2.2.0**
3. You need to emit a P1 certificate to ensure your identity, the server uses this certificate to verify if you are part of the university. This can be done at https://p1.icpedu.rnp.br/default/public/default . The certificate will be saved in your browser, and it should be exported to a folder on your computer.
4. To upload the certificate to the ESP's flash memory, you need a plugin into your Arduino IDE. Installation and Usage can be found at https://github.com/esp8266/arduino-esp8266fs-plugin. Link for the ESP File System plugin: ESP8266FS-0.3.0.zip

Hardware Requirements

A ESP module is required, there are many module versions for this chip and the one used on this article is the ESP-01. Use this link as reference https://github.com/esp8266/esp8266-wiki/wiki/Hardware_versions for module versions. The version 12 of the ESP module has SPI communication pins, but due to its pin size, this version couldn't be tested.



It is important to note that when you are uploading the certificate files or the firmware, the GPIO0 pin on the module must be connected to GROUND. And when the code is ready for production, GPIO0 must be disconnected from the GROUND

To write data into the ESP, a FTDI board is required. And the connections between the ESP and the FTDI are shown below.

Preparing the ESP

Make sure the GPIO0 of the module is connected to ground before you start the procedures.

Adding the certificate

As explained before, the certificate must be exported into your computer. Assuming this step is done and you have a **certificate.p12** file ready, you need to break this file into a key and a certificate using openssl. Go to the folder where the certificate is, and type the following:

1. openssl pkcs12 -in certificate.p12 -nocerts -out key.pem -nodes
2. openssl rsa -outform der -in key.pem -out key.der
3. openssl pkcs12 -in certificate.p12 -nokeys -out cert.pem -nodes
4. openssl x509 -outform der -in cert.pem -out cert.der

This creates 2 new files that are going to be added into the ESP (**cert.der** and **key.der**). Copy those files inside your sketch/data directory and upload it to the ESP, as explained on the item 3 in Software Requirements section.

If you are successful, the certificates are written inside the ESP's flash memory.

Uploading the Firmware

Get the new firmware and add it to your sketch. Uploading the code is the same as it is for any Arduino board.

Current firmware version is 0.4: ESP-LishaFirmware_secure-v0.4
Older Version: ESP-LishaFirmware_secure-v0.3

The default Wifi network on the firmware is:
Name: LishaJoinville
Pass: 12345678

Before connecting the ESP module with EPOSMote module

Using the Arduino Serial IDE to test if your setup was done correctly. Put the commands and check if the response is expected.

The following image shows an example of use of the **AT+SENDTSTP** command to **iot.lisha.ufsc.br/api/put.php**. You can see that the server response is what we were expecting.

Available Commands and Responses

It is important to note that when the ESP module is powered on, it connects to the default network (redeUFSCSemFio), and all the commands need to contain "\r" as the last character. The following table shows the available commands and responses.

For the latest firmware version (v0.4), the ESP's use is simplified. Instead of sending raw commands through UART, just import <esp8266.h> and use the class ESP8266.

Command	Description	Response	Example
AT+SYSTEMCHECK	Gets the status of the ESP Module	CONNECTION=(OK/FAIL)&FILESYSTEM=(OK/FAIL)	eMote: AT+SYSTEMCHECK\r -> ESP: CONNECTION=FAIL&FILESYSTEM=OK\r

AT+SYSTEMREADY	Checks if the module is ready	OK/FAIL	eMote: AT+SYSTEMREADY\r -> ESP: OK\r
AT+RESPONSETIME	Gets the response time of the last request	OK=(integer in ms)	eMote: AT+RESPONSETIME\r -> ESP: OK=25\r
AT+GETHOST	Gets the host of the connection. Default is **iot.ufsc.br**	OK=(host name)	eMote: AT+GETHOST\r -> ESP: OK=iot.ufsc.br\r
AT+GETROUTE	Gets the route reached by the connection. Default is **/api/put.php**	OK=(route)	eMote: AT+GETROUTE\r -> ESP: OK=/api/put.php\r
AT+GETPORT	Gets the port used on the connection. Default is **443**.	OK=(port)	eMote: AT+GETPORT\r -> ESP: OK=443\r
AT+SETHOST	Sets the host of the connection	OK	eMote: AT+SETHOST=www.google.com\r -> ESP: OK\r
AT+SETROUTE	Sets the route of the connection	OK	eMote: AT+SETROUTE=/api/get_data.php\r -> ESP: OK\r
AT+SETPORT	Sets the port used for the connection	OK	eMote: AT+SETPORT=80\r -> ESP: OK\r
AT+SETSSID	Sets the default wifi network name. Default is **RedeWifi**, pass: **12345678**.	OK	eMote: AT+SETSSID=eduroam\r -> ESP: OK\r
AT+SETPASSWORD	Sets the default wifi password	OK	eMote: AT+SETPASSWORD=12345678\r -> ESP: OK\r
AT+CONNECTWIFI	Connects to the chosen network	OK	eMote: AT+CONNECTWIFI\r -> ESP: OK\r
AT+GETTIMESTAMP	Gets the actual timestamp from network	(timestamp)	eMote: AT+GETTIMESTAMP\r -> ESP: (unix timestamp)\r
AT+GETHEAPSIZE	Gets the esp heap size in bytes	(heap size in bytes)	eMote: AT+GETHEAPSIZE\r -> ESP: (heap size in bytes)\r

AT+SENDTSTP	Sends data over the selected route to the selected host. It is used to authenticate the user.	if content size is less than 82 bytes, response is: **ERR=INVALIDCONTENT**. If content is valid, response will be the same as the server response	eMote: AT+AUTH=(82 bytes string)\r -> ESP: OK=DENIED\r
AT+SEND!32	Sends data over the selected route to the selected host. The number is used to define the message size in bytes. The response will be the server http response	(HTTP RESPONSE)	eMote: AT+SEND!4=1234\r -> ESP: OK=HTTP/1.1 400 Bad Request\r

Example

The following example shows how to declare the UART inside EPOS and send commands to the ESP module.

This example works with the version 0.4 of the ESP firmware.

EPOS UART Example

```
#include <utility/ostream.h> #include <utility/string.h> #include <alarm.h> #include <uart.h> #include <machine/cortex/esp8266.h> #include <riffs.h> using namespace EPOS; OStream cout; class Data_Record { public: Data_Record() { for(int i = 0; i < 84; i++) a[i] = 0; } char a[84]; }; int main() { Delay(2000000); UART uart(1, 115200, 8, 0, 1); GPIO rst('B', 3, GPIO::OUT); char host[] = "iot.lisha.ufsc.br"; char route[] = "/api/put_hydro.php"; cout << "ESP Com TEST" << endl; ESP8266 esp(&uart, &rst); cout << "ESP8266 created" << endl; esp.command_mode(); esp.config_endpoint(443, host, sizeof(host) - 1, route, sizeof(route) - 1); cout << "End point configured" << endl; esp.connect("LishaJoinville", 14, "12345678", 8); Data_Record aux; char res[255]; cout << endl << "Posting..." << endl; int res_size = esp.post(&aux, sizeof(aux), res); res[res_size - 1] = '\0'; cout << "Res size: " << res_size << endl; cout << "Response: " << res << endl; while(1){ } return 0; }
```

Troubleshooting guide

If problems are found, here are some things you can double check to make sure everything is setup correctly.

1. Make sure the ESP is powered with 3.3V and the source supply can provide at least 200mA.
2. The firmware default wifi network is LishaJoinville (pass: 12345678). Make sure this network exists, or change this setting on the firmware.
3. Make sure the traits file of your application is setup correctly according to the example above.
4. Are the components of your EPOS branch working as expected? A reliable UART is needed to transfer

data between the EPOSMote and the ESP

5. Is the ESP's GPIO-0 disconnected? If you power the ESP with GPIO-0 connected to ground, the module will boot on the flashing mode and the firmware won't run. Make sure GPIO-0 is not connected to anything
6. Double check the connections between the ESP and the eMote. So many wires can get us confused sometimes. **ESP's RX on eMote's TX** and **ESP's TX on eMote's RX.**
7. Certificates load should be done inside the setup method, connection might fail if the certificates are loaded during execution time.
8. Due to a wrong implementation of an internal TLS library, there was a memory leak causing the ESP to crash and restart. If you face some reset issues, make sure you are using the same version described in this tutorial.