

[All Collections](#)[Connect your devices](#)[Connect the ESP8266 as a telemetry unit with Ubidots](#)

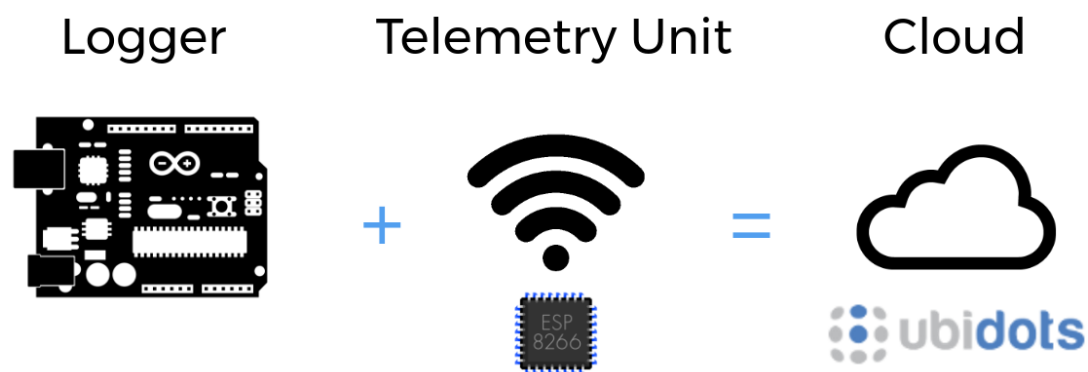
# Connect the ESP8266 as a telemetry unit with Ubidots

Learn how to manage between a logger and the ESP8266



Written by Maria Carlina Hernandez

Updated over a week ago



ESP8266 offers a complete and self-contained Wi-Fi networking solution, allowing it to either host the application or to offload all Wi-Fi networking functions from another application processor.

When the ESP8266 hosts an application and it is the only application processor in the device, and therefore able to boot up directly from an external flash. The ESP8266 has integrated cache to improve the performance and minimize the memory requirements.

Alternately, serving as a Wi-Fi adapter, wireless internet access can be added to any microcontroller-based design with simple connectivity through UART interface or the CPU AHB bridge interface.

This library will let you use the **ESP8266** module as a telemetry unit with any microcontroller connected to it via serial as logger. Please note that if the logger does not have more than one

serial port you will **not** able to visualize the response, but it can send data without any issues.

Before to start, you must choose which logger to use, in this case we decided use an **Arduino MEGA**.

**IMPORTANT NOTE:** It's very important know that we're going to manage two codes, one for the logger(Arduino MEGA), and a second for the telemetry unit(ESP8266).

## Requirements

- Any microcontroller with more than one serial port. e.i: [Arduino MEGA](#)
- [An ESP8266 module](#)
- [UartSBee](#) or any Uart to USB device.
- [Arduino IDE 1.6.0 or higher](#)
- [Ubidots Arduino ESP8266 library](#).
- [Ubidots Account](#).

## Command Request

We have designed a simple payload format you can use to send/get data from your devices to/from Ubidots using this protocol.

To know how to use the protocol and learn how to manage the data please reference to the README of the library on following [link](#).

## Setup

As we mentioned before, we have to manage two codes. That's mean we have two different setups, one for the **ESP8266 as telemetry unit** and a second one for the **Arduino MEGA as logger**.

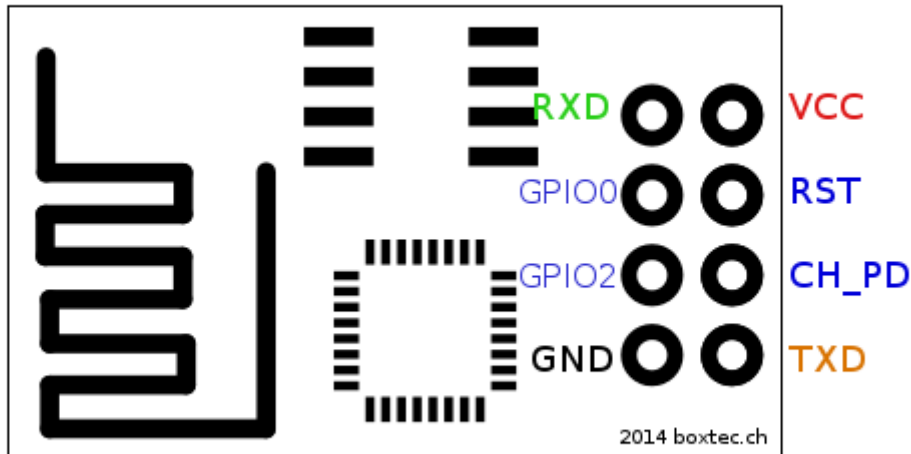
## Telemetry unit - ESP8266

As we know **Telemetry** is an automated communications process by which measurements and other data is collected at remote or inaccessible points and transmitted to receiving equipment for monitoring.

In this case, we are using the ESP8266 as a telemetry unit. We collect the data from the logger (Arduino MEGA) and the ESP8266 will be in charge of sending the collected data to Ubidots.

## Hardware setup - Wiring

**IT IS IMPORTANT IDENTIFY CAREFULLY THE PINOUT OF THE ESP8266, BECAUSE THE ESP8266 CAN BREAK IF YOU MAKE A WRONG CONNECTION**



To program the **ESP8266** you have to use a programmer as **UartSBee**, also you can use the **Arduino MEGA** to program the ESP8266. To upload the code into the ESP8266 you have to follow the connections below.

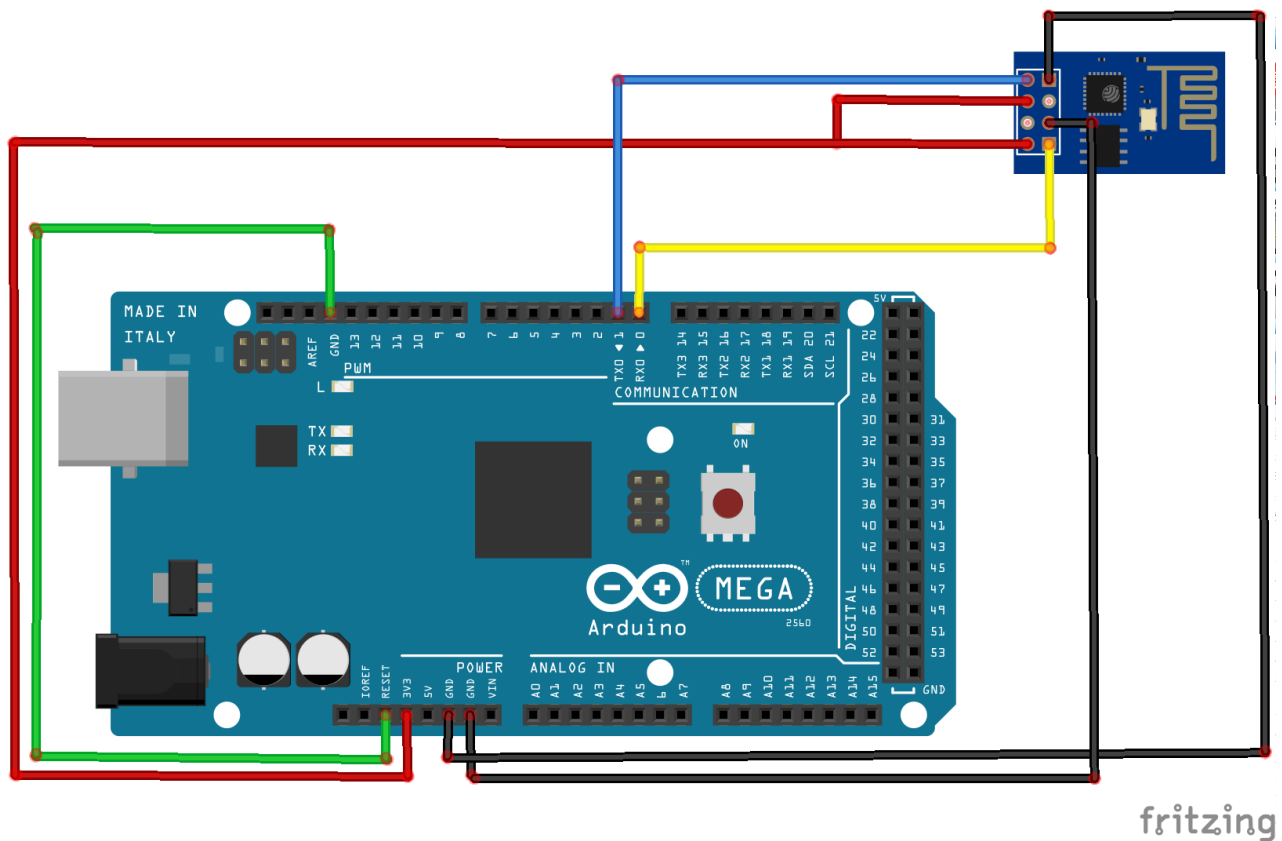
- ESP8266 - Arduino MEGA / ESP8266 - UartSBee connections:

ESP8266	Arduino MEGA	UartSBee
RXD	RX0 - (0)	TXD
GPIO0	GND	GND
GPIO2	—	—
GND	GND	GND
VCC	3.3V	3.3V
RST	—	—
CH_PD	3.3V	3.3V
TXD	TX0 -(1)	RXD

**NOTE:** Please be careful with the VCC of the ESP8266, it works only with a 3.3V supply.

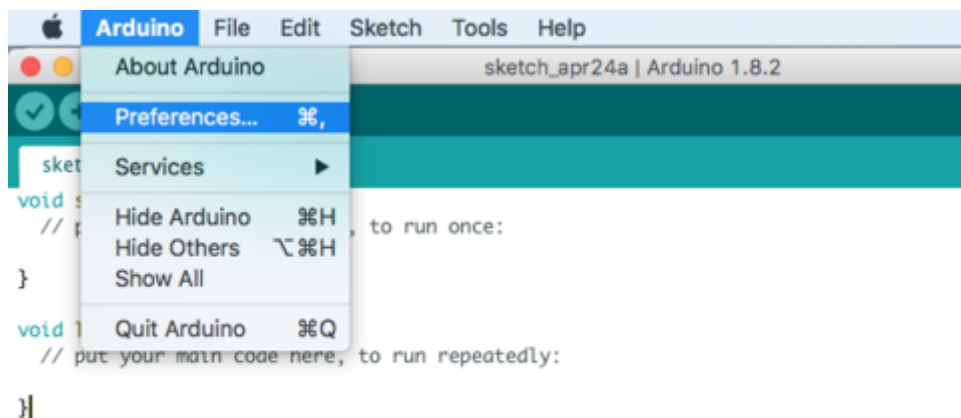
If you're using the **Arduino MEGA**, makes sure that there are no other programs running and use the serial communication channel. Also, you have to set Arduino **RST** to **GND**(green wire); if you do not, you will get a console error and will not able to compile the ESP8266.

#### ESP8266 - Arduino MEGA diagram connections:

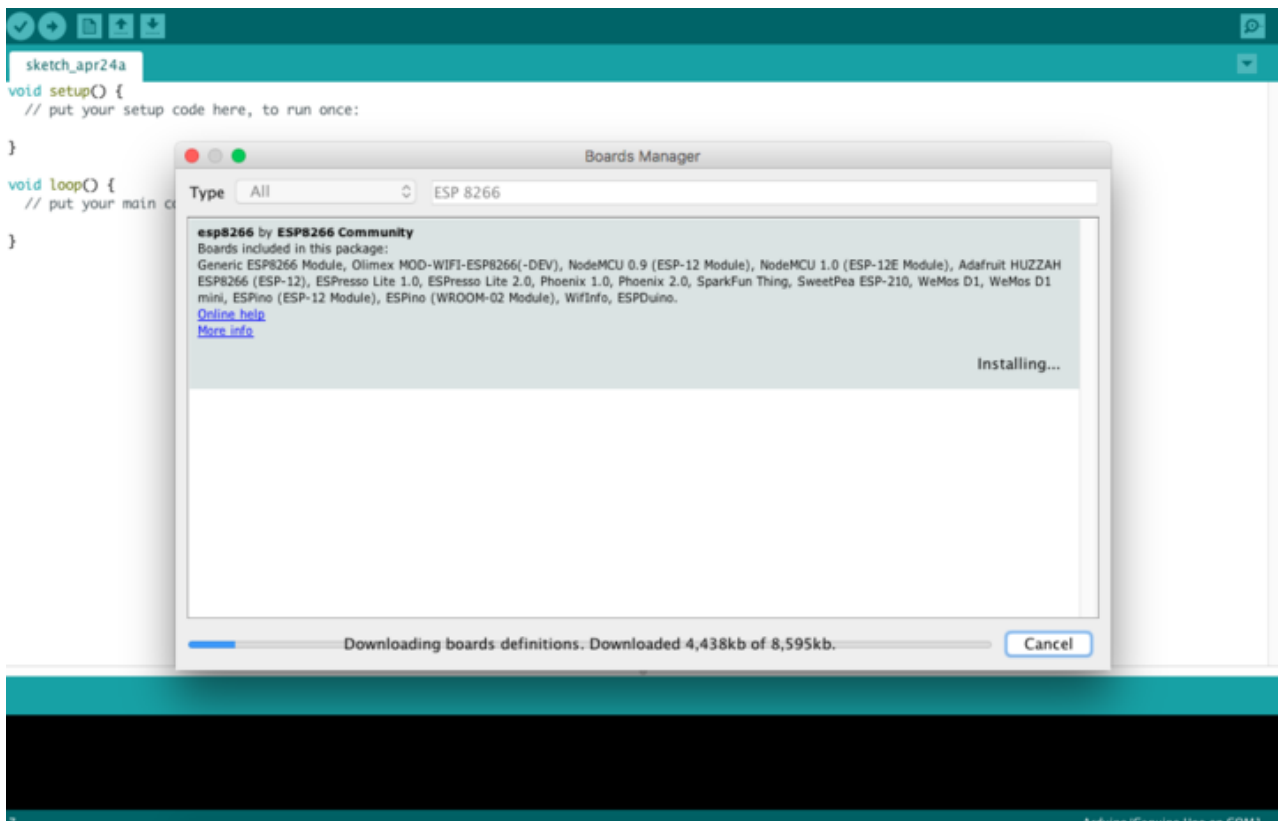


## Firmware Setup - Arduino IDE

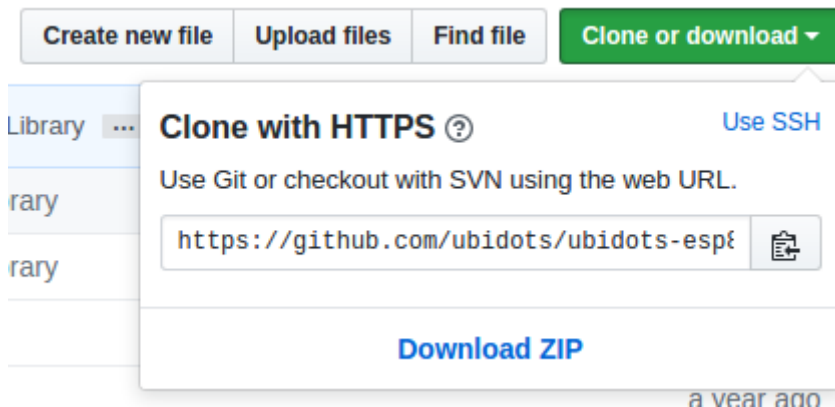
1. Go to the [Arduino IDE](https://www.arduino.cc/en/main/software).
2. Open the Arduino IDE, select Files -> Preferences and enter [http://arduino.esp8266.com/stable/package\\_esp8266com\\_index.json](http://arduino.esp8266.com/stable/package_esp8266com_index.json) into Additional Board Manager URLs field. You can add multiple URLs, separating them with commas.



3. Open **Boards Manager** from **Tools -> Board -> Boards Manager** and install **esp8266** platform. To simply find the correct device, search ESP8266 within the search bar.



4. Select the **Generic ESP8266 Module** from **Tools > Board** menu.
5. Assigns **115200** as baudrate. Click on **Tools > Upload Speed: "115200"**.
6. Download the **UbidotsESP8266Serial** library [here](#).



7. Now, click on **Sketch -> Include Library -> Add .ZIP Library**.
8. Select the **.ZIP** file of **UbidotsESP8266Serial** and then **"Accept"** or **"Choose"**.
9. Close the Arduino IDE and open it again.
10. Paste the code provide below.

## Telemetry Unit - ESP8266 code

Once you have pasted the code below into the **Arduino IDE**, you will have to assign the parameters required to connect the **ESP8266** to Wi-Fi and Ubidots. Please assign your Wi-Fi SSID, Wi-Fi password, and Ubidots TOKEN where are indicated:

```
/*
 * Include Libraries
 */
#include "UbidotsESP8266.h"

/*
 * Define Constants
 */
namespace {
    const char * WIFISSID = "Put_your_WIFI_SSID_here"; // Assign your WiFi SSID
    const char * PASSWORD = "Put_your_WIFI_password_here"; // Assign your WiFi password
    const char * TOKEN = "Put_your_Ubidots_Token_here"; // Assign your Ubidots TOKEN
}

Ubidots client(TOKEN);

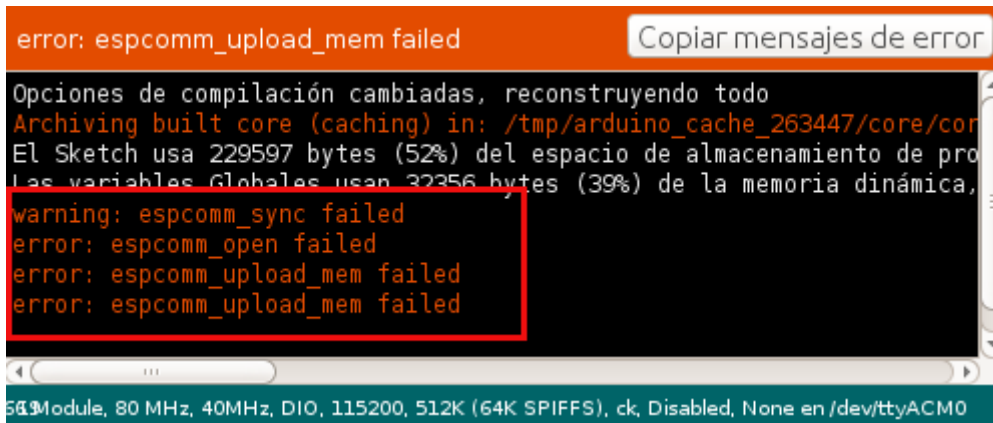
/*
 * Main Functions
 */
void setup() {
    Serial.begin(115200);
    client.wifiConnection(WIFISSID, PASSWORD);
}

void loop() {
    client.readData(); // Reads the command from the logger
    delay(1000);
}
```

**NOTE:** The sampling frequency on the ESP8266 is of 1000 milliseconds

## FAQs and Troubleshooting

One of the most common troubleshooting with the ESP8266 is this one:



The screenshot shows the Arduino IDE terminal window. At the top, there is an orange banner with the text "error: espcomm\_upload\_mem failed" and a button labeled "Copiar mensajes de error". Below this, the terminal displays the following text: "Opciones de compilación cambiadas, reconstruyendo todo", "Archiving built core (caching) in: /tmp/arduino\_cache\_263447/core/core-arduino-esp8266", "El Sketch usa 229597 bytes (52%) del espacio de almacenamiento de programación", "Las variables Globales usan 32356 bytes (39%) de la memoria dinámica", "warning: espcomm\_sync failed", "error: espcomm\_open failed", "error: espcomm\_upload\_mem failed", and "error: espcomm\_upload\_mem failed". A red rectangle highlights the last four lines of text. At the bottom of the terminal, there is a green banner with the text "563 Module, 80 MHz, 40MHz, DIO, 115200, 512K (64K SPIFFS), ck, Disabled, None en /dev/ttyACM0".

If you got the issue above, please verify if the connections are in the right way, check if the Arduino **RST** is setted to **GND** , also verify if the board **Generic ESP8266 Module** is selected on the Arduino IDE.

**NOTE:** If you use the Arduino MEGA to program the ESP8266 don't forget establish the new wires connections to be able to communicate the devices through the hardware serial port.

## Logger - Arduino MEGA

The logger will be in charge of taking sensor (pin) readings and send values to the **Telemetry Unit** via serial.

You can use any microcontroller as logger, but please note that if the logger (**microcontroller**) does not have more than one hardware serial ports you **will not able to visualize the response**, but it is also possible to send data without any issue.

We decided to used the **Arduino MEGA** as logger, because it has more than one hardware serial ports available.

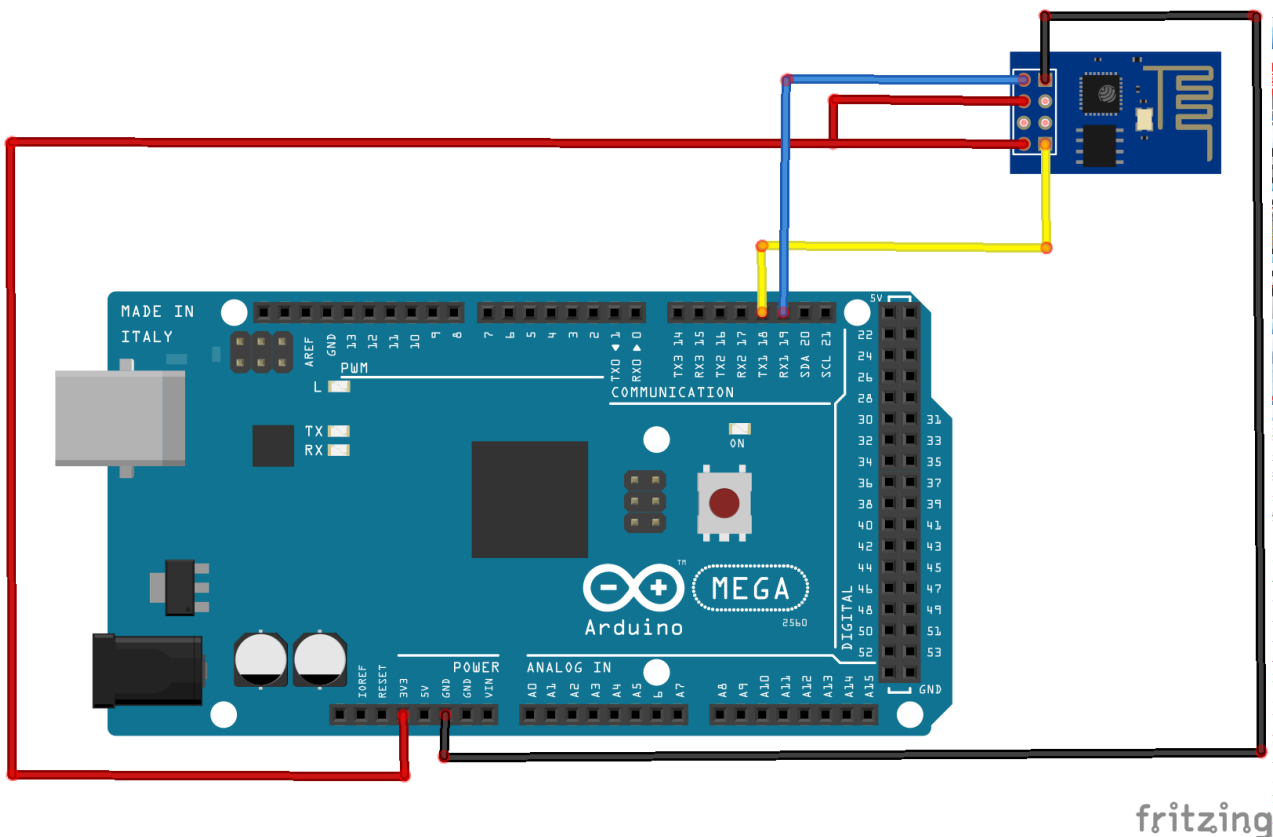
## Hardware setup - Wiring

Once the code is uploaded into the telemetry unit (**ESP8266**), you have to make the right connections to be able to communicate the telemetry unit with the logger via serial. Please follow the table below to make the connections, also verify the connections with the diagram provided:

- Arduino MEGA - ESP8266 Connections:

Arduino MEGA	ESP8266
RX1 - (19)	TX
TX1 - (18)	RX
3.3V	3V3
3.3V	CH_PD
GND	GND

Diagram Connections



## Firmware Setup - Arduino IDE

1. Once you completed the step before go to the Arduino IDE.
2. Assign the board to compile, in this case Arduino Mega. Click on **Tools > Board:** **"Arduino/Genuino Mega or Mega 2560"**.
3. Select the Port assigned for the board. Click on **Tools > Port > Select the port assigned.**
4. Paste the code provided below with the modification required (reference to the next step).
5. Upload the code into the board. Then, verify it works through the Serial Monitor.



## Send command via serial - Manual

The following code lets you send commands through the Serial Monitor. This will help you to verify if the logger and the Telemetry unit are working together.

**NOTE:** To know how to build the command required, reference to this [link](#).

Once the code is uploaded, open the **Serial Monitor** and send the command, then, when all is programmed correctly, you have to receive the **"OK"** response from the server. If you receive an **"ERROR"** response verify the structure of the command.

**NOTE:** Take in count that the response is printed with a carriage return.

```
/******  
 * Define Constants  
******/  
char command[128]; // command  
char answer[100]; // answer from server  
  
/******  
 * Main Functions  
******/  
void setup() {  
  Serial.begin(115200);  
  Serial1.begin(115200);  
}  
  
void loop() {  
  
  int i = 0;  
  int j = 0;  
  
  if (Serial.available()){  
    while (Serial.available() > 0) {  
      command[i++] = (char)Serial.read();  
    }  
    /* Sends the command to the telemetry unit */  
    Serial1.print(command);  
    i = 0;  
  }  
  
  if (Serial1.available() > 0) {  
    /* Reading the telemetry unit */  
    while (Serial1.available() > 0) {  
      answer[j++] = (char)Serial1.read();  
    }  
    /* Response from the server */  
    Serial.print(answer);  
    j = 0;  
  }  
}
```

```
}
}
```

## Send command via serial - Automatically

Once you verify that everything works as it should, you can use the following code to manage your sensors readings and send the values to Ubidots.

As you can see at the first part of the code you just have to assign your Ubidots TOKEN where is indicated. And also, please assign your device and variable parameters appropriately if you adjusted any portion of code provided in this documentation.

```

/*****
 * Define Constants
 *****/
namespace {
    bool flow_control = true; // control the flow of the requests
    const char * USER_AGENT = "UbidotsESP8266"; // Assign the user agent
    const char * VERSION = "1.0"; // Assign the version
    const char * METHOD = "POST"; // Set the method
    const char * TOKEN = "....."; // Assign your Ubidots TOKEN
    const char * DEVICE_LABEL = "ESP8266"; // Assign the device label
    const char * VARIABLE_LABEL = "temp"; // Assign the variable label
}

char telemetry_unit[100]; // response of the telemetry unit

/* Space to store values to send */
char str_sensor1[10];
char str_sensor2[10];

/*****
 * Main Functions
 *****/
void setup() {
    Serial.begin(115200);
    Serial1.begin(115200);
}

void loop() {
    char* command = (char *) malloc(sizeof(char) * 128);
    /* Wait for the server response to read the values and built the command */
    /* While the flag is true it will take the sensors readings, build the command,
       and post the command to Ubidots */
    if (flow_control) {
        /* Analog reading */
        float sensor1 = analogRead(A0);
        float sensor2 = analogRead(A1);

```

```

/* 4 is minimum width, 2 is precision; float value is copied onto str_sensor*/
dtostrf(sensor1, 4, 2, str_sensor1);
dtostrf(sensor2, 4, 2, str_sensor2);

/* Building the logger command */
sprintf(command, "init#");
sprintf(command, "%s%s/%s|s|s|", command, USER_AGENT, VERSION, METHOD, TOKEN);
sprintf(command, "%s=s>", command, DEVICE_LABEL);
sprintf(command, "%s:s:s", command, VARIABLE_LABEL, str_sensor1);
sprintf(command, "%s|end#final", command);

/* Prints the command sent */
//Serial.println(command);// uncomment this line to print the command

/* Sends the command to the telemetry unit */
Serial1.print(command);
/* free memory*/
free(command);
/* Change the status of the flag to false. Once the data is sent, the status
  of the flag will change to true again */
flow_control = false;
}

/* Reading the telemetry unit */
int i = 0;
while (Serial1.available() > 0) {
  telemetry_unit[i++] = (char)Serial1.read();
  /* Change the status of the flag; allows the next command to be built */
  flow_control = true;
}

if (flow_control) {
  /* Print the server response -> OK */
  Serial.write(telemetry_unit);
  /* free memory */
  memset(telemetry_unit, 0, i);
}

delay(1000);
}

```

If you desire sending more values, reference the **README** of the [library](#) to learn how to build the request containing more than one variable. Then, you will have to modify the command (as seen above) `/* Building the logger command */` to include a data traffic rate that is greater than one variable.

## Ubidots' data management

Once the you complete the steps above and works without any issue, go to your [Ubidots account](#) to visualize the data received.

## Result

Now it is time to create a dashboard to manage the variables of your device. To learn more about Ubidots widgets and events, check out these [video tutorials](#).

Happy hacking :)

Did this answer your question?

[Home](#)[Create an account](#)[Blog](#)[Video tutorials](#)

We run on Intercom