

# Part 17

-

# Bluetooth Serial

## ESP32 Bluetooth Classic with Arduino IDE

### Bluetooth Classic with ESP32

At the moment, using Bluetooth Classic is much more simpler than Bluetooth Low Energy. If you've already programmed an Arduino with a Bluetooth module like the HC-06, this is very similar. It uses the standard serial protocol and functions.

### Bluetooth Terminal Application

To proceed with this tutorial, you need a Bluetooth Terminal application installed in your smartphone. We recommend using the Android app "Serial Bluetooth Terminal" available in the Play Store.



## Serial Bluetooth Terminal

Kai Morich

Tools

UNINSTALL

OPEN

In-app purchases

## Serial to Serial Bluetooth

Open your Arduino IDE, and go to File > Examples > BluetoothSerial > SerialtoSerialBT. The following code should load.

```
//This example code is in the Public Domain (or CC0 licensed, at your
option.)
//By Evandro Copercini - 2018
//
//This example creates a bridge between Serial and Classical Bluetooth (SPP)
//and also demonstrate that SerialBT have the same functionalities of a
normal Serial

#include "BluetoothSerial.h"

#if !defined(CONFIG_BT_ENABLED) || !defined(CONFIG_BLUEDROID_ENABLED)
#error Bluetooth is not enabled! Please run `make menuconfig` to and enable
it
#endif

BluetoothSerial SerialBT;

void setup() {
  Serial.begin(115200);
  SerialBT.begin("ESP32test"); //Bluetooth device name
  Serial.println("The device started, now you can pair it with bluetooth!");
}

void loop() {
  if (Serial.available()) {
    SerialBT.write(Serial.read());
  }
  if (SerialBT.available()) {
    Serial.write(SerialBT.read());
  }
  delay(20);
}
```

### How the Code Works

This code establishes a two-way serial Bluetooth communication between two devices.

The code starts by including the BluetoothSerial library.

```
#include "BluetoothSerial.h"
```

The next three lines check if Bluetooth is properly enabled.

```
#if !defined(CONFIG_BT_ENABLED) || !defined(CONFIG_BLUEDROID_ENABLED)
#error Bluetooth is not enabled! Please run `make menuconfig` to and enable
it
#endif
```

Then, create an instance of BluetoothSerial called SerialBT:

```
BluetoothSerial SerialBT;
```

In the `setup()` initialize a serial communication at a baud rate of 115200.

```
Serial.begin(115200);
```

Initialize the Bluetooth serial device and pass as an argument the Bluetooth Device name. By default it's called `ESP32test` but you can rename it and give it a unique name.

```
SerialBT.begin("ESP32test"); //Bluetooth device name
```

In the `loop()`, send and receive data via Bluetooth Serial.

In the first if statement, we check if there are bytes being received in the serial port. If there are, send that information via Bluetooth to the connected device.

```
if (Serial.available()) {  
  SerialBT.write(Serial.read());  
}
```

`SerialBT.write()` sends data using bluetooth serial.

`Serial.read()` returns the data received in the serial port.

The next if statement, checks if there are bytes available to read in the Bluetooth Serial port. If there are, we'll write those bytes in the Serial Monitor.

```
if (SerialBT.available()) {  
  Serial.write(SerialBT.read());  
}
```

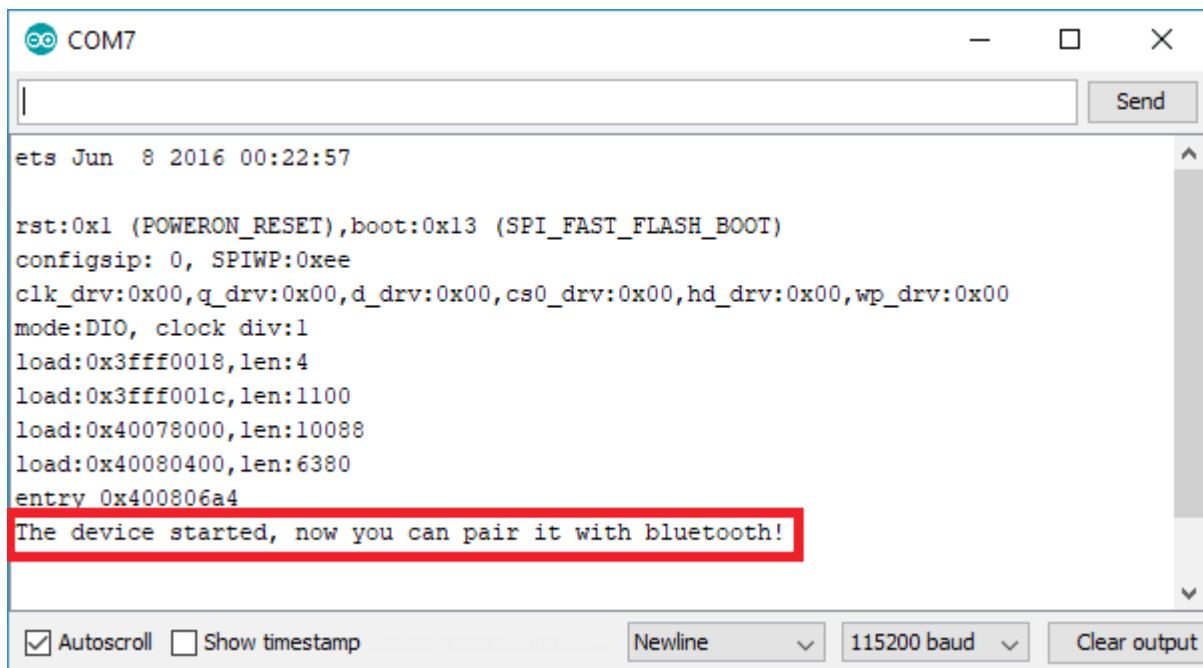
It will be easier to understand exactly how this sketch works in the demonstration.

### Testing the Code

Upload the previous code to the ESP32. Make sure you have the right board and COM port selected.

After uploading the code, open the Serial Monitor at a baud rate of 115200. Press the ESP32 Enable button.

After a few seconds, you should get a message saying: *"The device started, now you can pair it with bluetooth!"*.



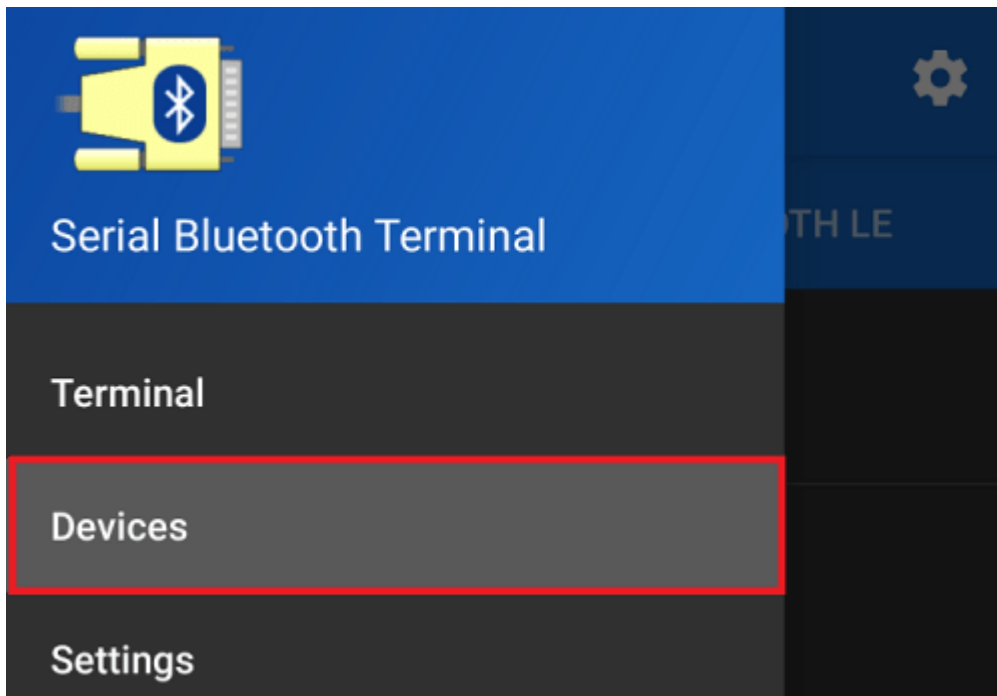
The screenshot shows the Serial Monitor window for COM7. The output text is as follows:

```
ets Jun  8 2016 00:22:57  
  
rst:0x1 (POWERON_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)  
config: 0, SPIWP:0xee  
clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x00  
mode:DIO, clock div:1  
load:0x3fff0018,len:4  
load:0x3fff001c,len:1100  
load:0x40078000,len:10088  
load:0x40080400,len:6380  
entry 0x400806a4  
The device started, now you can pair it with bluetooth!
```

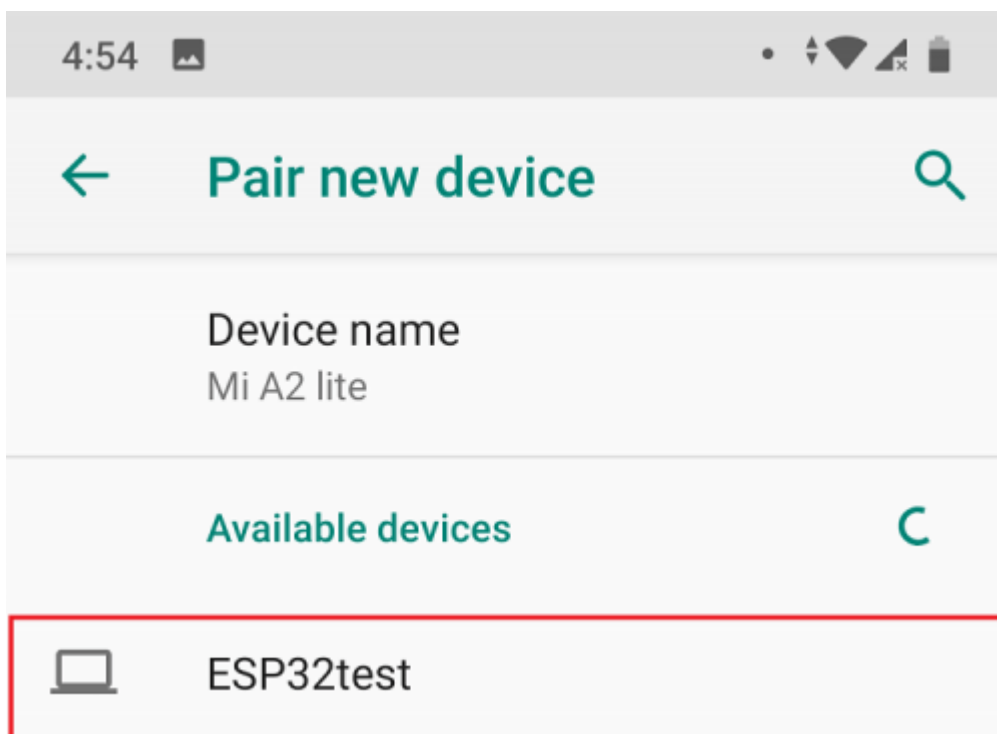
At the bottom of the window, the following options are visible:  Autoscroll,  Show timestamp, Newline (dropdown), 115200 baud (dropdown), and Clear output.

Go to your smartphone and open the "Serial Bluetooth Terminal" app. Make sure you've enable your smartphone's Bluetooth.

To connect to the ESP32 for the first time, you need to pair a new device. Go to `Devices`.

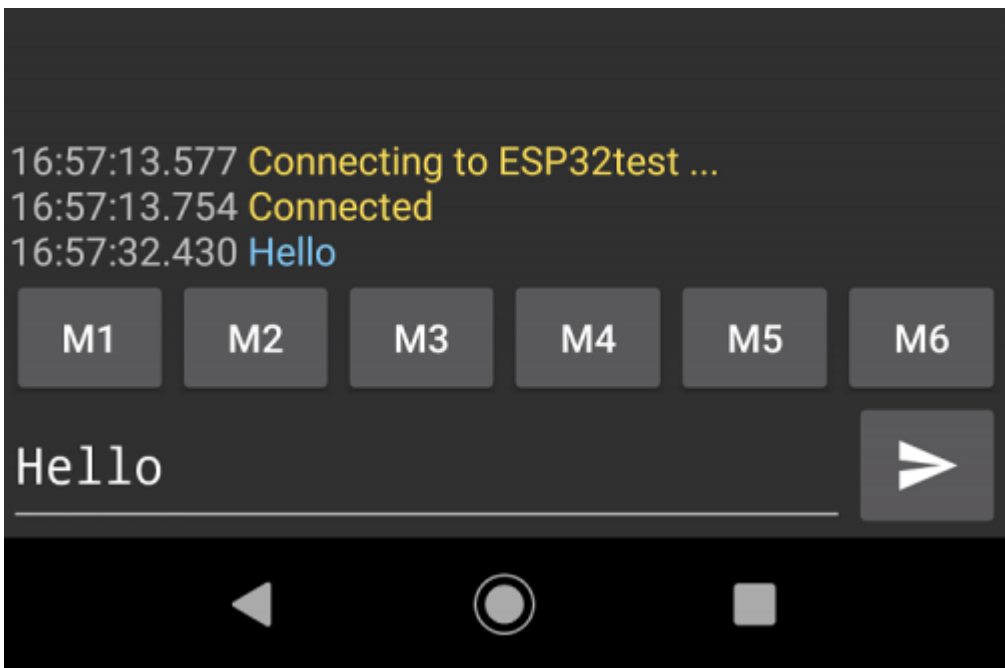


Click the settings icon, and select `Pair new device`. You should get a list with the available Bluetooth devices, including the `ESP32test`. Pair with the `ESP32test`.

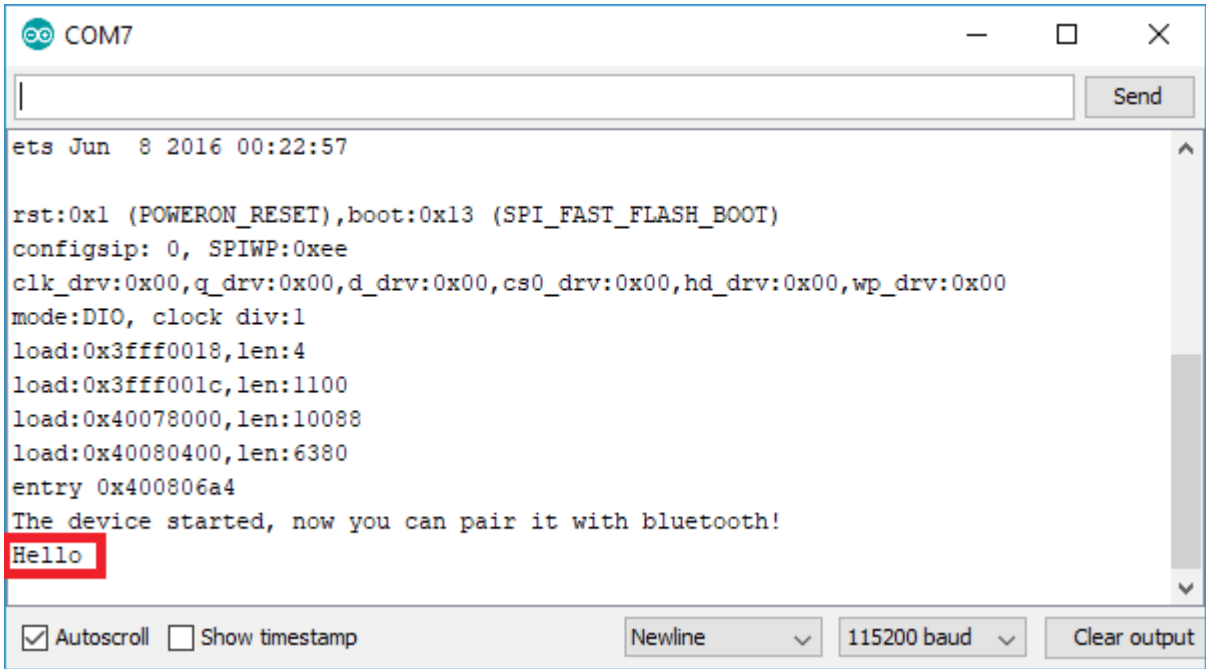


Then, go back to the Serial Bluetooth Terminal. Click the icon at the top to connect to the ESP32. You should get a **"Connected"** message.

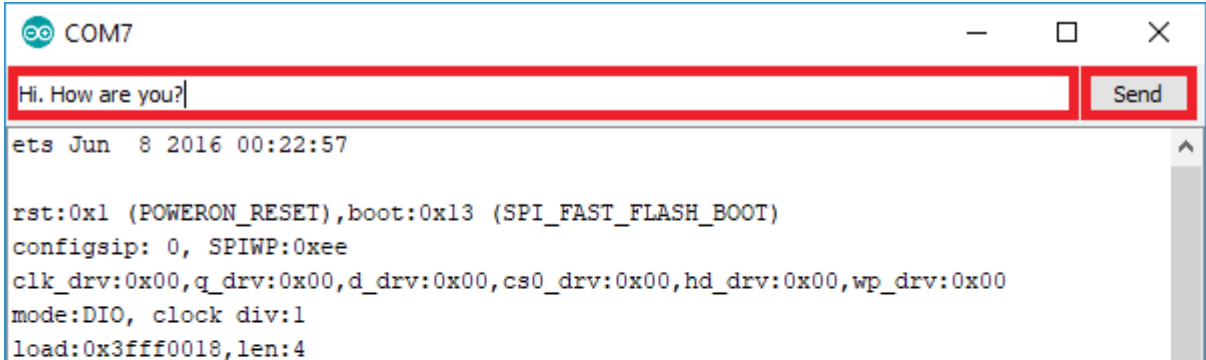
After that, type something in the Serial Bluetooth Terminal app. For example, **"Hello"**.



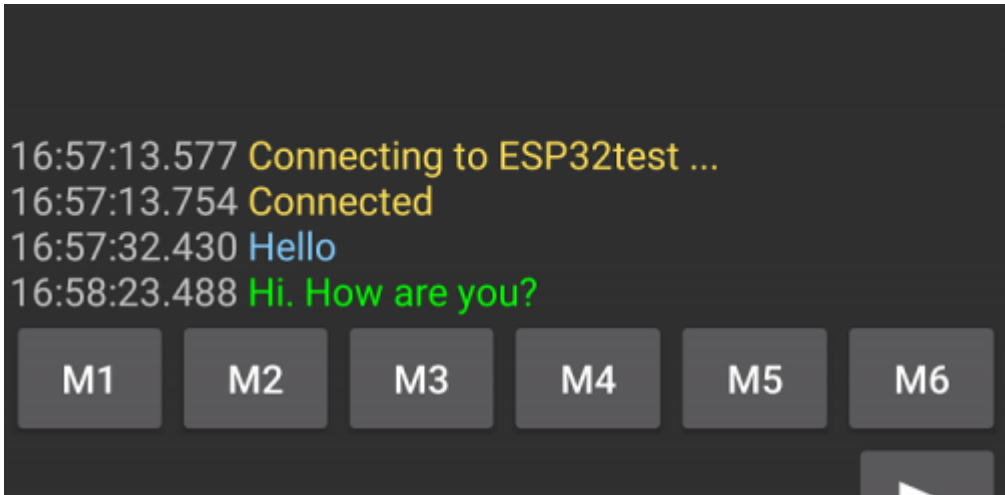
You should instantly receive that message in the Arduino IDE Serial Monitor.



You can also exchange data between your Serial Monitor and your smartphone. Type something in the Serial Monitor top bar and press the "Send" button.



You should instantly receive that message in the Serial Bluetooth Terminal App.



## Exchange Data using Bluetooth Serial

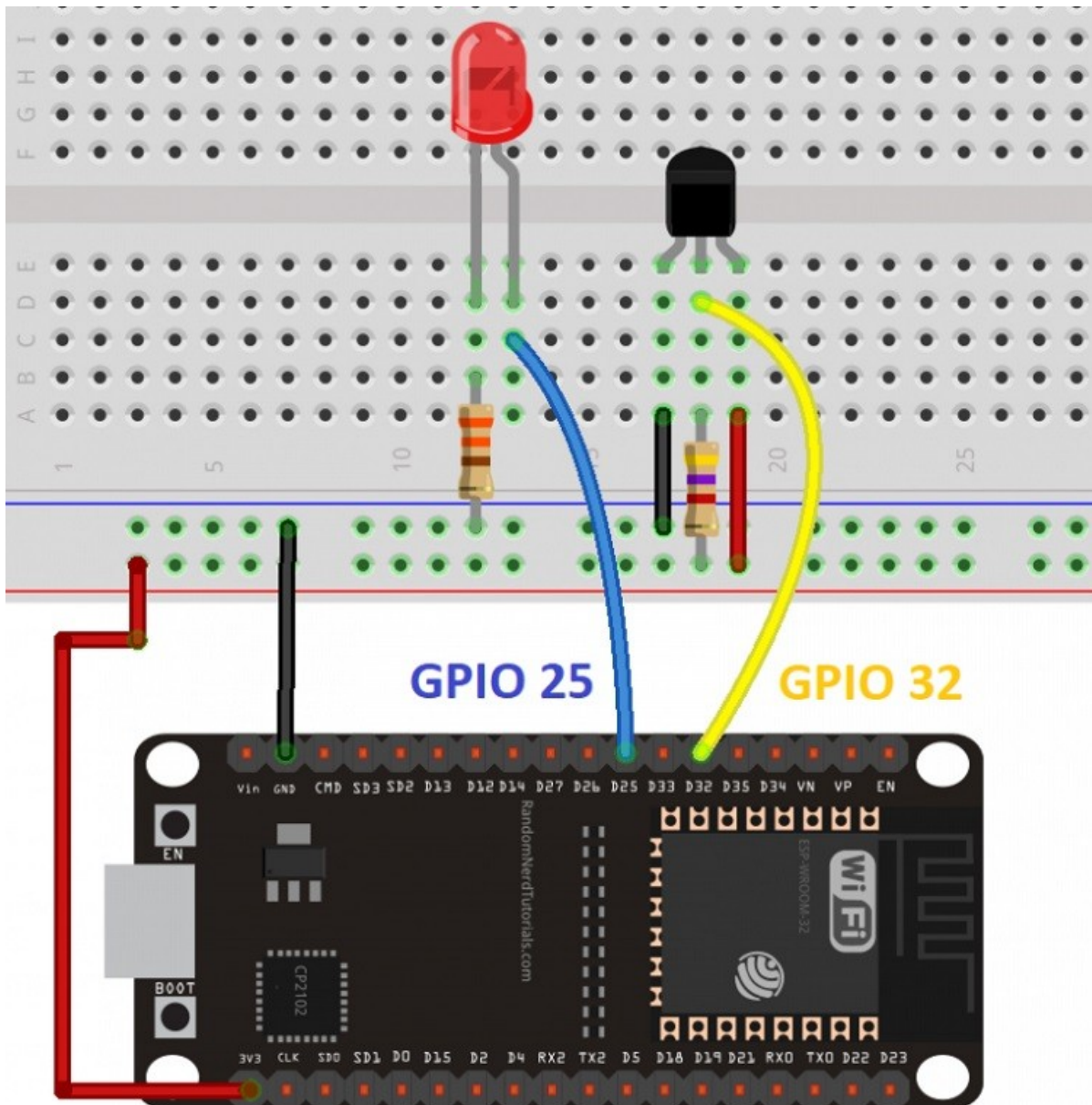
Now that you know how to exchange data using Bluetooth Serial, you can modify the previous sketch to make something useful. For example, control the ESP32 outputs when you receive a certain message, or send data to your smartphone like sensor readings.

The project we'll build sends temperature readings every 10 seconds to your smartphone. We'll be using the DS18B20 temperature sensor.

Through the Android app, we'll send messages to control an ESP32 output. When the ESP32 receives the **led\_on** message, we'll turn the GPIO on, when it receives the **led\_off** message, we'll turn the GPIO off.

## Schematic

Before proceeding with this project, assemble the circuit by following the next schematic diagram. Connect an LED to GPIO25, and connect the DS18B20 data pin to GPIO32.



To work with the DS18B20 temperature sensor, you need to install the One Wire library by Paul Stoffregen and the Dallas Temperature library. Follow the next instructions to install these libraries, if you haven't already.



### ***One Wire library***

- Click <https://github.com/milesburton/Arduino-Temperature-Control-Library/archive/master.zip> to download the One Wire library. You should have a .zip folder in your Downloads
- Unzip the .zip folder and you should get **OneWire-master** folder
- Rename your folder from **OneWire-master** to **OneWire**
- Move the **OneWire** folder to your Arduino IDE installation **libraries** folder
- Finally, re-open your Arduino IDE

### ***Dallas Temperature library***

- Click here <https://github.com/milesburton/Arduino-Temperature-Control-Library/archive/master.zip> to download the Dallas Temperature library. You should have a .zip folder in your Downloads
- Unzip the .zip folder and you should get **Arduino-Temperature-Control-Library-master** folder
- Rename your folder from **Arduino-Temperature-Control-Library-master** to **DallasTemperature**
- Move the **DallasTemperature** folder to your Arduino IDE installation **libraries** folder
- Finally, re-open your Arduino IDE

After assembling the circuit and installing the necessary libraries, copy the following sketch to your Arduino IDE.

```

// Load libraries
#include "BluetoothSerial.h"
#include <OneWire.h>
#include <DallasTemperature.h>

// Check if Bluetooth configs are enabled
#if !defined(CONFIG_BT_ENABLED) || !defined(CONFIG_BLUEDROID_ENABLED)
#error Bluetooth is not enabled! Please run `make menuconfig` to and enable
it
#endif

// Bluetooth Serial object
BluetoothSerial SerialBT;

// GPIO where LED is connected to
const int ledPin = 25;

// GPIO where the DS18B20 is connected to
const int oneWireBus = 32;
// Setup a oneWire instance to communicate with any OneWire devices
OneWire oneWire(oneWireBus);
// Pass our oneWire reference to Dallas Temperature sensor
DallasTemperature sensors(&oneWire);

// Handle received and sent messages
String message = "";
char incomingChar;
String temperatureString = "";

// Timer: auxiliar variables
unsigned long previousMillis = 0;    // Stores last time temperature was
published
const long interval = 10000;        // interval at which to publish sensor
readings

void setup()
{
  Serial.begin(115200);

  pinMode(ledPin, OUTPUT);

  // Bluetooth device name
  SerialBT.begin("ESP32");
  Serial.println("The device started, now you can pair it with bluetooth!");
}

void loop()
{
  unsigned long currentMillis = millis();

  // Send temperature readings
  if (currentMillis - previousMillis >= interval)
  {
    previousMillis = currentMillis;
    sensors.requestTemperatures();
    temperatureString = String(sensors.getTempCByIndex(0)) + "C " +
String(sensors.getTempFByIndex(0)) + "F";
    SerialBT.println(temperatureString);
  }

  // Read received messages (LED control command)
  if (SerialBT.available())
  {
    char incomingChar = SerialBT.read();

```

```
if (incomingChar != '\n')
{
    message += String(incomingChar);
}
else
{
    message = "";
}
Serial.write(incomingChar);
}

// Check received message and control output accordingly
if (message == "led_on")
{
    digitalWrite(ledPin, HIGH);
}
else if (message == "led_off")
{
    digitalWrite(ledPin, LOW);
}

delay(20);
}
```

## **How the Code Works**

Start by including the necessary libraries. The `BluetoothSerial` library for Bluetooth, and the `OneWire` and `DallasTemperature` for the DS18B20 temperature sensor.

```
#include "BluetoothSerial.h"
#include <OneWire.h>
#include <DallasTemperature.h>
```

Create a `BluetoothSerial` instance called `SerialBT`.

```
BluetoothSerial SerialBT;
```

Create a variable called `ledPin` to hold the GPIO you want to control. In this case, GPIO25 has an LED connected.

```
const int ledPin = 25;
```

Define the DS18B20 sensor pin and create objects to make it work. The temperature sensor is connected to GPIO32.

```
// GPIO where the DS18B20 is connected to
const int oneWireBus = 32;
// Setup a oneWire instance to communicate with any OneWire devices
OneWire oneWire(oneWireBus);
// Pass our oneWire reference to Dallas Temperature sensor
DallasTemperature sensors(&oneWire);
```

Create an empty string called `message` to store the received messages.

```
String message = "";
```

Create a char variable called `incomingChar` to save the characters coming via Bluetooth Serial.

```
char incomingChar;
```

The `temperatureString` variable holds the temperature readings to be sent via Bluetooth.

```
String temperatureString = "";
```

Create auxiliary timer variables to send readings every 10 seconds.

```
unsigned long previousMillis = 0; // Stores last time temperature was
published
const long interval = 10000; // interval at which to publish sensor
readings
```

In the `setup()`, set the `ledPin` as an output.

```
pinMode(ledPin, OUTPUT);
Initialize the ESP32 as a bluetooth device with the "ESP32" name.
SerialBT.begin("ESP32"); //Bluetooth device name
```

In the `loop()`, send the temperature readings, read the received messages and execute actions accordingly. The following snippet of code, checks if 10 seconds have passed since the last reading. If it's time to send a new reading, we get the latest temperature and save it in Celsius and Fahrenheit in the `temperatureString` variable.

```
unsigned long currentMillis = millis();
if (currentMillis - previousMillis >= interval) {
    previousMillis = currentMillis;
    sensors.requestTemperatures();
```

```
temperatureString = " " + String(sensors.getTempCByIndex(0)) + "C " +  
String(sensors.getTempFByIndex(0)) + "F";
```

Then, to send the temperatureString via bluetooth, use SerialBT.println().

```
SerialBT.println(temperatureString);
```

The next if statement reads incoming messages. When you receive messages via serial, you receive a character at a time. You know that the message ended, when you receive \n. So, we check if there's data available in the Bluetooth serial port.

```
if (SerialBT.available())
```

If there is, we'll save the characters in the incomingChar variable.

```
char incomingChar = SerialBT.read();
```

If the incomingChar is different than \n, we'll concatenate that character to our message.

```
if (incomingChar!= '\n')  
{  
  message += String(incomingChar);  
}
```

When we're finished reading the characters, we clear the message variable. Otherwise all received messages would be appended to each other.

```
Message = "";
```

After that, we have two if statements to check the content of the message. If the message is **led\_on**, the LED turns on.

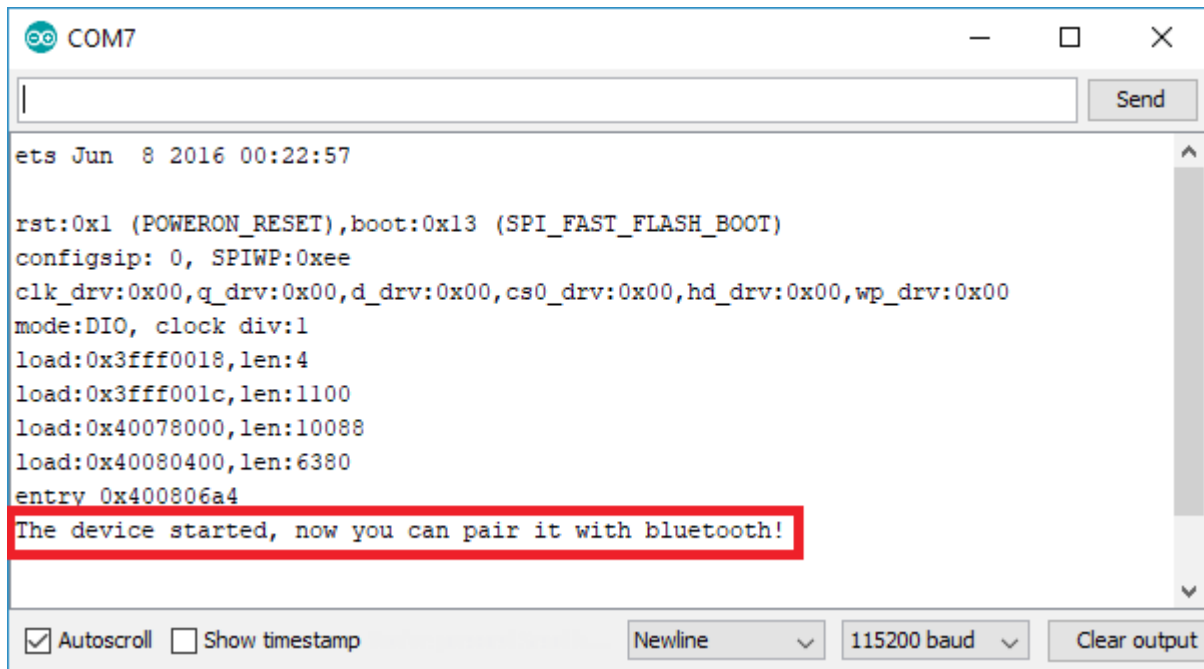
```
if (message == "led_on"){  
  digitalWrite(ledPin, HIGH);  
}
```

If the message is **led\_off**, the LED turns off.

```
else if (message == "led_off"){  
  digitalWrite(ledPin, LOW);  
}
```

### Testing the Project

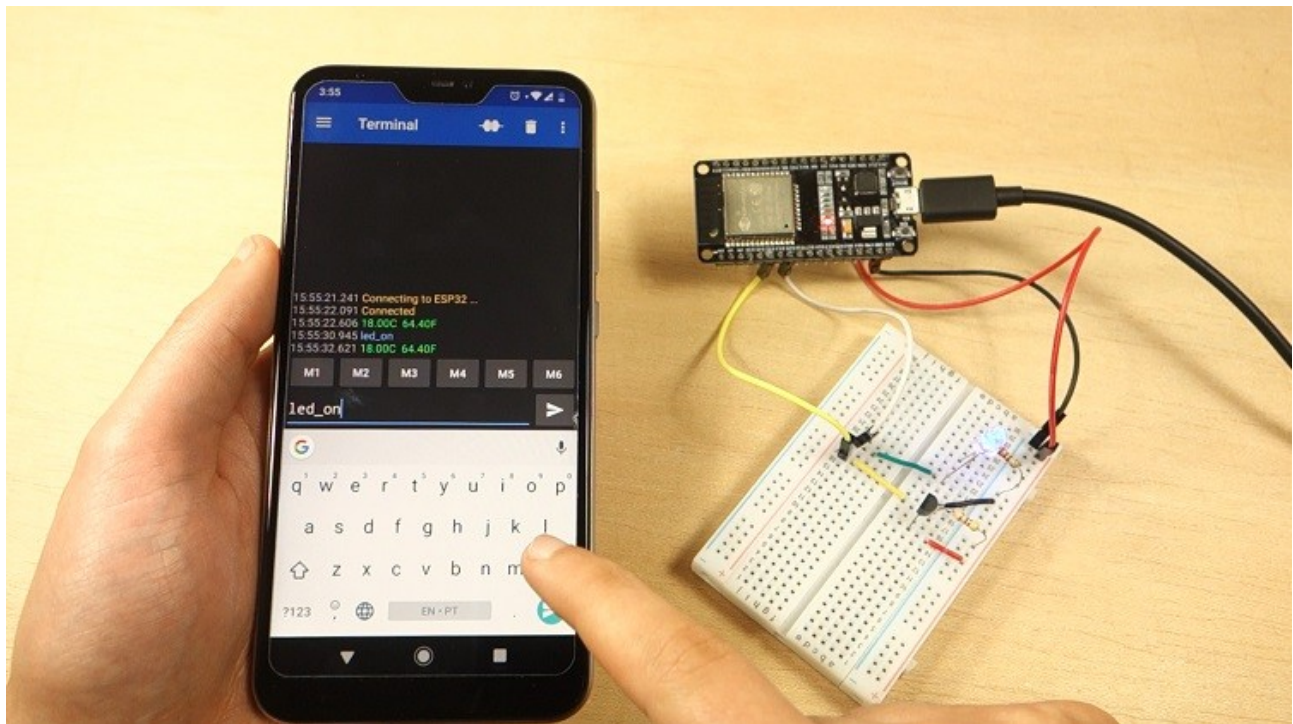
Upload the previous sketch to your ESP32 board. Then, open the Serial Monitor, and press the ESP32 Enable button. When you receive the following message, you can go to your smartphone and connect with the ESP32.



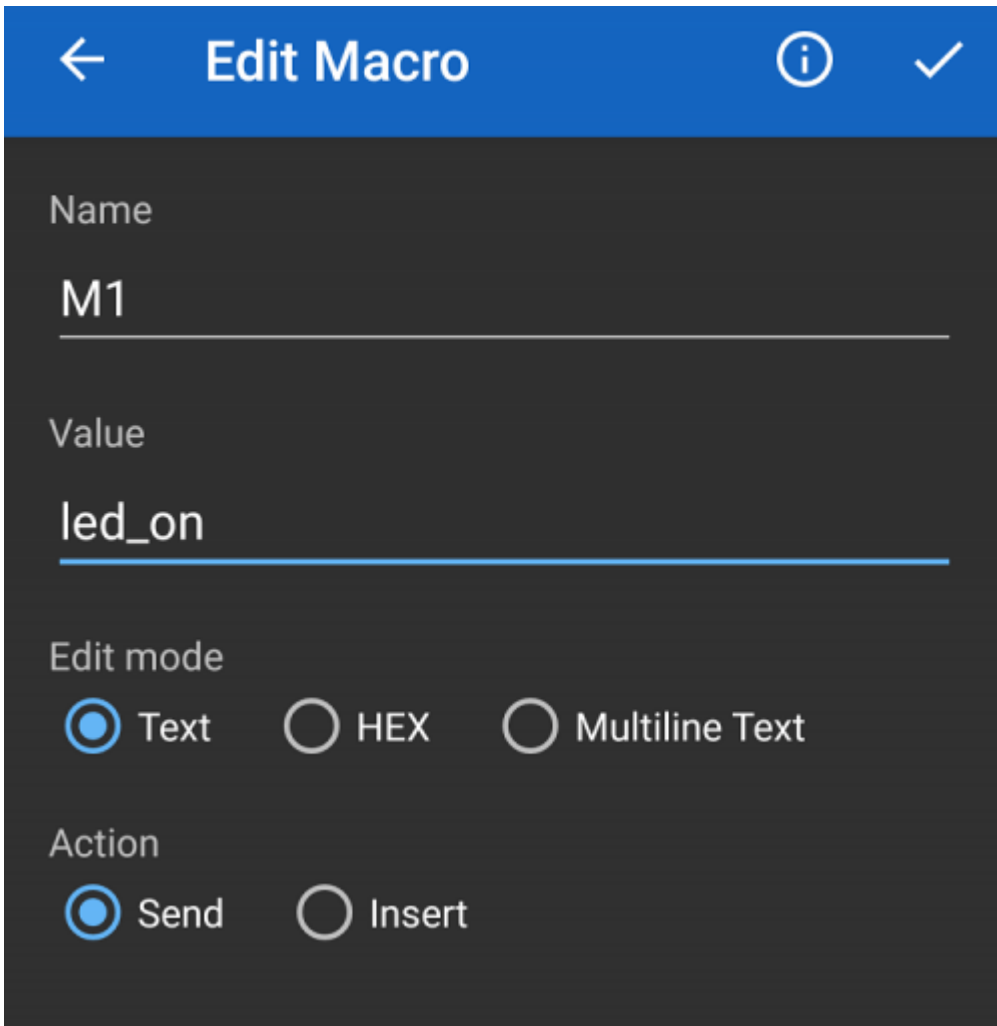
```
ets Jun 8 2016 00:22:57

rst:0x1 (POWERON_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
configsip: 0, SPIWP:0xee
clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x00
mode:DIO, clock div:1
load:0x3fff0018,len:4
load:0x3fff001c,len:1100
load:0x40078000,len:10088
load:0x40080400,len:6380
entry 0x400806a4
The device started, now you can pair it with bluetooth!
```

Then, you can write the "led\_on" and "led\_off" messages to control the LED.



The application has several buttons in which you can save default messages. For example, you can associate **M1** with the "led\_on" message, and **M2** with the "led\_off" message.



Now, you are able to control the ESP32 GPIOs. At the same time, you should be receiving the temperature readings every 10 seconds.

