Click Here

The objective of this tutorial is to explain how to configure the ESP32 to act as a discoverable Bluetooth device and then find it using a Python program. The tests of this ESP32 tutorial were performed using a DFRobot’s ESP-WROOM-32 device integrated in a ESP32 FireBeetle board. Introduction The objective of this tutorial is to explain how to configure the ESP32 to act as a discoverable Bluetooth device and then find it using a Python program. The ESP32 will be running the Arduino core. On Python, we will be using Pybluez, a module that will allow us to use the Bluetooth functionalities of a computer that supports this protocol. Note that the use of Pybluez was already covered in this previous tutorial. As mentioned there, you can get the .exe file from the module’s page and install it. Since the Bluetooth functionalities have just recently arrived to the Arduino core at the time of writing, you may need to get the latest version from the Github page to be able to follow this tutorial. On the ESP32 side, we will use some lower level IDF functions to start the Bluetooth stacks and to make the device discoverable. This was already covered in detail on this previous tutorial. You can check here the Bluetooth classic API from IDF. The tests of this ESP32 tutorial were performed using a DFRobot’s ESP-WROOM-32 device integrated in a ESP32 FireBeetle board. The Python code We start our Python script by importing the new library we have just installed. Note that even tough the library is called Pybluez, the actual code module we are going to use is called bluetooth. import bluetooth To start the discovery, we simply need to call the discover_devices function. In order for the discovery to return the names of the devices, we pass the value True in the lookup_names parameter of the function. Note that this is an optional parameter that defaults to False and without setting it to True we would only get the address of the devices. Take in consideration that the execution of this function takes a while, in order to scan the nearby devices. By default, the discovery duration is 8 seconds, and we will not change it. devices = bluetooth.discover_devices(lookup_names=True) This function call will return a list that we can iterate in a for in loop, in order to print each device’s information. Each element of the list will contain both the Bluetooth address and the device name. Before this loop, we will also print the size of the list, which will correspond to the number of Bluetooth devices found during the discovery procedure. print("Devices found: %s" % len(devices)) for item in devices: print(item) The final code can be seen below. import bluetooth devices = bluetooth.discover_devices(lookup_names=True) print("Devices found: %s" % len(devices)) for item in devices: print(item) The Arduino code We will start our code by including the libraries needed to start the Bluetooth stacks (esp_bt_main.h) and to make the device discoverable (esp_gap_bt_api.h). Since one of the parameters that will be printed by the Python script is the address of the device, we will also print it in the ESP32 side, for comparison. For a detailed tutorial on how to print the Bluetooth address of the ESP32, please check this previous post. For that functionality, we will need the esp_bt_device.h library. #include "esp_bt_main.h" #include "esp_gap_bt_api.h" #include "esp_bt_device.h" Now that we have all the needed libraries included, we need a function to initialize both the controller and host stacks of Bluetooth. This was also covered in greater detail in this previous tutorial. This function will receive as input a string with the name of the device, which will be seen by other Bluetooth enabled devices when discovering it. Thus, this name should be later obtained in the Python program. bool initBluetooth(const char *deviceName) { // Initialization code } In the implementation of the function, we will first call the btStart function, to initialize the Bluetooth controller stack. Them we will call the esp_bluedroid_init and esp_bluedroid_enable functions to both init and enable Bluedroid (the host stack). We will also perform an error checking on the call of each of the previously mentioned functions, so we are sure everything has initialized correctly. if (!btStart()) { Serial.println("Failed to initialize controller"); return false; } if (esp_bluedroid_init()!= ESP_OK) { Serial.println("Failed to initialize bluedroid"); return false; } if (esp_bluedroid_enable()!= ESP_OK) { Serial.println("Failed to enable bluedroid"); return false; } After the initialization, we will set the device name. To do it, we need to call the esp_bt_dev_set_device_name function. This function will receive as input a string with the name of the device. We will use the name which is the argument of our initBluetooth function. esp_bt_dev_set_device_name(deviceName); Then we need to make the device discoverable with a call to the esp_bt_gap_set_scan_mode function, passing as input the ESP_BT_SCAN_MODE_CONNECTABLE_DISCOVERABLE enumerated value. esp_bt_gap_set_scan_mode(ESP_BT_SCAN_MODE_CONNECTABLE_DISCOVERABLE); With this, we finish our initBluetooth function. Now we still need to declare and implement a function that will be used to retrieve the Bluetooth address of the ESP32. We will basically reuse the same function of the already mentioned previous post to get the address. In its implementation, we first call the esp_bt_dev_get_address function to get the six bytes that compose the unique address. Then we will print them in the standard format, which corresponds to printing each byte in hexadecimal, separated by colons. void printDeviceAddress() { const uint8_t* point = esp_bt_dev_get_address(); for (int i = 0; i < 6; i++) { char str[3]; sprintf(str, "%02X", (int)point[i]); Serial.print(str); if (i < 5){ Serial.print(":"); } } } Moving on to the Arduino setup function, we will start by opening a wired serial connection, to print the results of our program. Note that the Serial object is being used by both our previously declared functions, so we need to make sure it is initialized before using it. Serial.begin(115200); Next we call the initBluetooth function, passing as input the name to assign to the ESP32. I’m using “ESP32 BT”, but you can use other name. initBluetooth("ESP32 BT"); To finalize the setup function, we call the printDeviceAddress function, which will output the Bluetooth address of the ESP32. We will later be able to compare against the one obtained in the Python script. printDeviceAddress(); The final Arduino source code for the ESP32 can be seen below. #include "esp_bt_main.h" #include "esp_bt_device.h" #include "esp_gap_bt_api.h" bool initBluetooth(const char *deviceName) { if (!btStart()) { Serial.println("Failed to initialize controller"); return false; } if (esp_bluedroid_init()!= ESP_OK) { Serial.println("Failed to initialize bluedroid"); return false; } if (esp_bluedroid_enable()!= ESP_OK) { Serial.println("Failed to enable bluedroid"); return false; } esp_bt_dev_set_device_name(deviceName); esp_bt_gap_set_scan_mode(ESP_BT_SCAN_MODE_CONNECTABLE_DISCOVERABLE); } void printDeviceAddress() { const uint8_t* point = esp_bt_dev_get_address(); for (int i = 0; i < 6; i++) { char str[3]; sprintf(str, "%02X", (int)point[i]); Serial.print(str); if (i < 5){ Serial.print(":"); } } } void setup() { Serial.begin(115200); initBluetooth("ESP32 BT"); printDeviceAddress(); } void loop() {} Testing the code The first step for testing the code is to compile it and upload it to the ESP32 using the Arduino IDE. If you run into compilation problems, then you may be using an older version of the Arduino core without support for Bluetooth, which you can easily update by following this guide. When the procedure finishes, simply open the Arduino IDE serial monitor. You should get an output similar to figure 1, which shows the device address getting printed. Figure 1 – Bluetooth address of the ESP32. After this, simply run the Python script we have developed on the environment of your choice. I’m running it on IDLE, the Python IDE that comes with the language installation. You should get an output similar to figure 1, which shows the ESP32 getting detected during the scan. Note that the address matches the one we obtained on the Arduino IDE serial monitor and the device name is the same we specified in the Arduino code. Figure 2 – Finding the device with Pybluez. Related posts The ESP32 comes with Wi-Fi, Bluetooth Low Energy and Bluetooth Classic. In this tutorial, you’ll learn how to use ESP32 Bluetooth Classic with Arduino IDE to exchange data between an ESP32 and an Android smartphone. We’ll control an ESP32 output, and send sensor readings to an Android smartphone using Bluetooth Classic. Note: this project is only compatible with Android smartphones. You can watch the video tutorial or keep reading this page for the written instructions. 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. In this tutorial, we’ll start by using an example that comes with the Arduino IDE. Then, we’ll build a simple project to exchange data between the ESP32 and your Android smartphone. To follow this tutorial, you need the following parts: You can use the preceding links or go directly to MakerAdvisor.com/tools to find all the parts for your projects at the best price! 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. We’ll program the ESP32 using Arduino IDE, so make sure you have the ESP32 add-on installed before proceeding: 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); } View raw code 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. 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!”. 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. 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. 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. Recommended reading: ESP32 Pinout Reference: Which GPIO pins should you use? 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 here to download the One Wire library. You should have a .zip folder in your DownloadsUnzip the .zip folder and you should get OneWire-master folderRename your folder from OneWire-master to OneWireMove the OneWire folder to your Arduino IDE installation libraries folderFinally, re-open your Arduino IDE Dallas Temperature library Click here to download the Dallas Temperature library. You should have a .zip folder in your DownloadsUnzip the .zip folder and you should get Arduino-Temperature-Control-Library-master folderRename your folder from Arduino-Temperature-Control-Library-master to DallasTemperatureMove the DallasTemperaturefolder to your Arduino IDE installation libraries folderFinally, re-open your Arduino IDE After assembling the circuit and installing the necessary libraries, copy the following sketch to your Arduino IDE. /********* Rui Santos Complete project details at *********/ // Load libraries #include "BluetoothSerial.h" #include #include // 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() { pinMode(ledPin, OUTPUT); Serial.begin(115200); // 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 != ''){ 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); } View raw code Let’s take a quick look at the code and see how it 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 #include 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 auxiliar 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 . 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 , we’ll concatenate that char character to our message. if (incomingChar!= ''){ 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); } 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. 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. In summary, the ESP32 supports BLE and Bluetooth Classic. Using Bluetooth Classic is as simple as using serial communication and its functions. If you want to learn how to use BLE with the ESP32, you can read our guide: Getting Started with ESP32 Bluetooth Low Energy (BLE) on Arduino IDE We hope you’ve found this tutorial useful. For more projects with the ESP32 you can check our project’s compilation: 20+ ESP32 Projects and Tutorials. This tutorial is a preview of the “Learn ESP32 with Arduino IDE” course. If you like this project, make sure you take a look at the ESP32 course page where we cover this and a lot more topics with the ESP32. Tags: Bluetooth, ESP32, Wireless communication In this tutorial, we’ll learn how to activate, manage and test Bluetooth on an ESP32 using the Arduino programming language. Bluetooth is a wireless technology widely used for communication between electronic devices. It enables you to quickly transform your system into a connected object. ESP32 module (on-board Bluetooth+Wifi) A computer with Python installed or smartphone USB cable for ESP32-computer connection To program your ESP32 with the Arduino IDE, you can follow this previous tutorial. This information is not necessarily necessary, but it’s always a good idea to know how to retrieve the MAC address from the ESP32. #include "esp_bt_main.h" #include "esp_bt_device.h" #include "BluetoothSerial.h" BluetoothSerial SerialBT; void printDeviceAddress() { const uint8_t* point = esp_bt_dev_get_address(); for (int i = 0; i < 6; i++) { char str[3]; sprintf(str, "%02X", (int)point[i]); Serial.print(str); if (i < 5){ Serial.print(":"); } } } void setup() { Serial.begin(115200); SerialBT.begin("ESP32BT"); Serial.print("MaxAddr : "); printDeviceAddress(); } void loop() {} Output 14:42:43.448 -> MaxAddr : 3C:61:05:31:5F:12 Bluetooth communication is activated in the same way as serial communication. The method is similar for the HC-06 module #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("ESP32BT"); //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); } N.B.: There seems to be a way of adding a PIN code, but I can’t get it to work. SerialBT.setPin(pin);SerialBT.begin("ESP32BT ", true); Once you’ve configured the module as you wish, you can pair the ESP32 with the system of your choice, just like any other Bluetooth device. Select the name from the list of detected devices (name ESP32BT) We’re going to test Bluetooth communication using the Serial Bluetooth Terminal application. The message is exchanged between the phone and the ESP32 via Bluetooth In the previous code, we made a byte-by-byte copy of the message to send it back to the monitor. Here, we’ll save the complete command in a String msg. This will enable you to analyze the command and define the corresponding action (e.g.: switch on #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 String msg; BluetoothSerial SerialBT; const char *pin = "1234"; void setup() { Serial.begin(115200); SerialBT.begin("ESP32BT"); //Bluetooth device name Serial.println("ESP32BT device started, now you can pair it!"); } void loop(){ readSerialPort(); // Send answer to master if(msg!=""){ Serial.print("Master sent : "); Serial.println(msg); SerialBT.print("received : "+msg); msg=""; } } void readSerialPort(){ while (SerialBT.available()) { delay(10); if (SerialBT.available() >0) { char c = SerialBT.read(); //gets one byte from serial buffer msg += c; //makes the string readString } } SerialBT.flush(); } You can manage Bluetooth communication from your PC. To do this, install the PyBluez package python -m pip install pybluez import bluetooth target_name = "ESP32BT" target_address = None nearby_devices = bluetooth.discover_devices(lookup_names=True,lookup_class=True) print(nearby_devices) for btaddr, btname, btclass in nearby_devices: if target_name == btname: target_address = btaddr break if target_address is not None: print("found target {} bluetooth device with address {} ".format(target_name,target_address)) else: print("could not find target bluetooth device nearby") Output [('88:C6:26:7E:F2:7A', 'UE\xa0BOOM\xa02', 2360344), ('4C:EA:AE:D6:92:08', 'OPPO A94 5G', 5898764), ('41:42:DD:1F:45:69', 'MX_light', 2360344), ('3C:61:05:31:5F:12', 'ESP32BT', 7936)] found target ESP32BT bluetooth device with address 3C:61:05:31:5F:12 Here’s a Python script to automatically connect to the ESP32 Bluetooth device from a PC import bluetooth import socket target_name = "ESP32BT" target_address = None nearby_devices = bluetooth.discover_devices(lookup_names=True,lookup_class=True) print(nearby_devices) for btaddr, btname, btclass in nearby_devices: if target_name == btname: target_address = btaddr break if target_address is not None: print("found target {} bluetooth device with address {} ".format(target_name,target_address)) """ # With PyBluez NOT WORKING serverMACAddress = target_address port = 1 s = bluetooth.BluetoothSocket(bluetooth.RFCOMM) s.connect((serverMACAddress, port)) while 1: text = raw_input() # Note change to the old (Python 2) raw_input if text == "quit": break s.send(text) data = s.recv(1024) if data: print(data) sock.close()""" serverMACAddress = target_address port = 1 s = socket.socket(socket.AF_BLUETOOTH, socket.SOCK_STREAM, socket.BTPROTO_RFCOMM) s.connect((serverMACAddress,port)) print("connected to {}".format(target_name)) while 1: text = input() if text == "quit": break s.send(bytes(text, 'UTF-8')) data = s.recv(1024) if data: print(data) s.close() else: print("could not find target bluetooth device nearby") N.B.: Only the socket library works for Bluetooth communication. There seems to be a maintenance issue with the PyBluez library. Create a native React application to communicate with ESP32 ESP-IDF Documentation Socket PyBluez Client Server