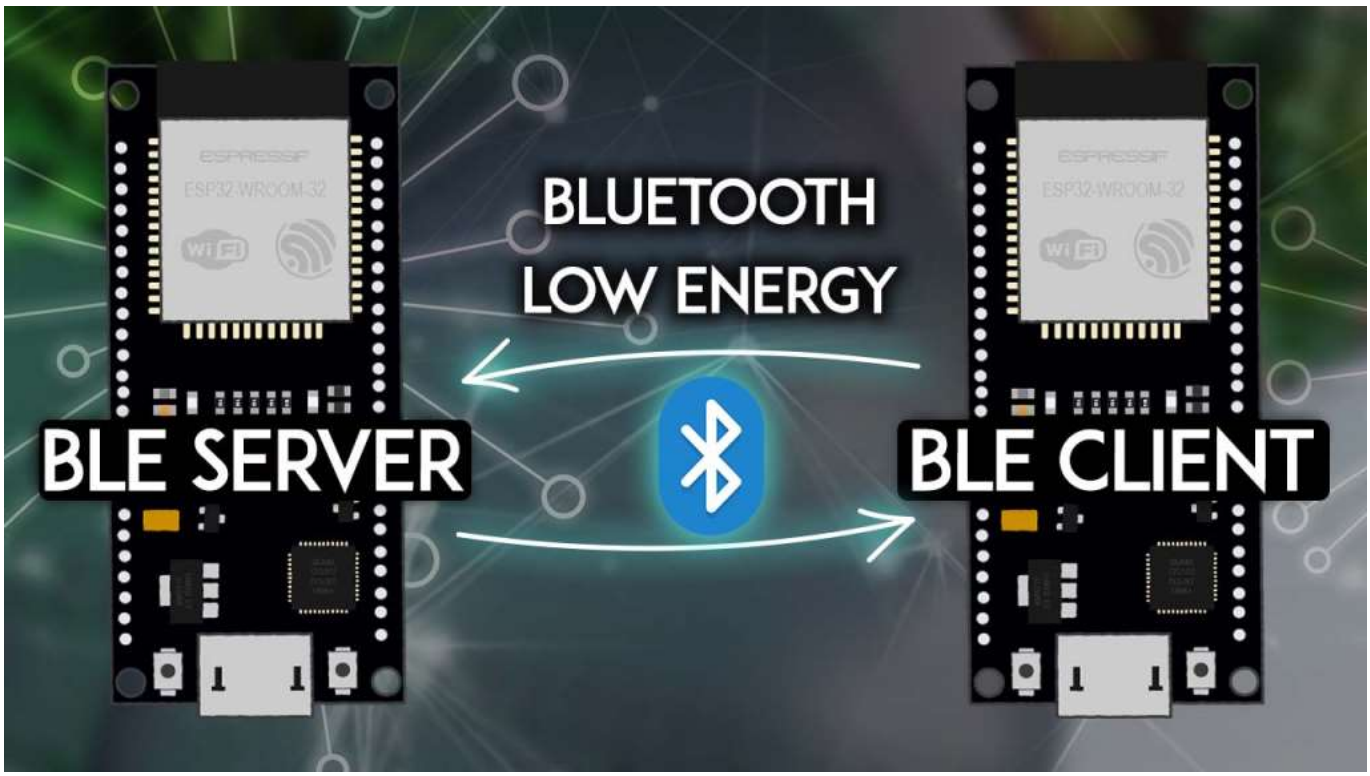
ESP32 BLE Server and Client (Bluetooth Low Energy)

Learn how to make a BLE (Bluetooth Low Energy) connection between two ESP32 boards. One ESP32 is going to be the server, and the other ESP32 will be the client. The BLE server advertises characteristics that contain sensor readings that the client can read. The ESP32 BLE client reads the values of those characteristics (temperature and humidity) and displays them on an OLED display.



Recommended Reading: Getting Started with ESP32 Bluetooth Low Energy (BLE)

What is Bluetooth Low Energy?

Before going straight to the project, it is important to take a quick look at some essential BLE concepts so that you're able to better understand the project later on. If

you're already familiar with BLE, you can skip to the Project Overview section.

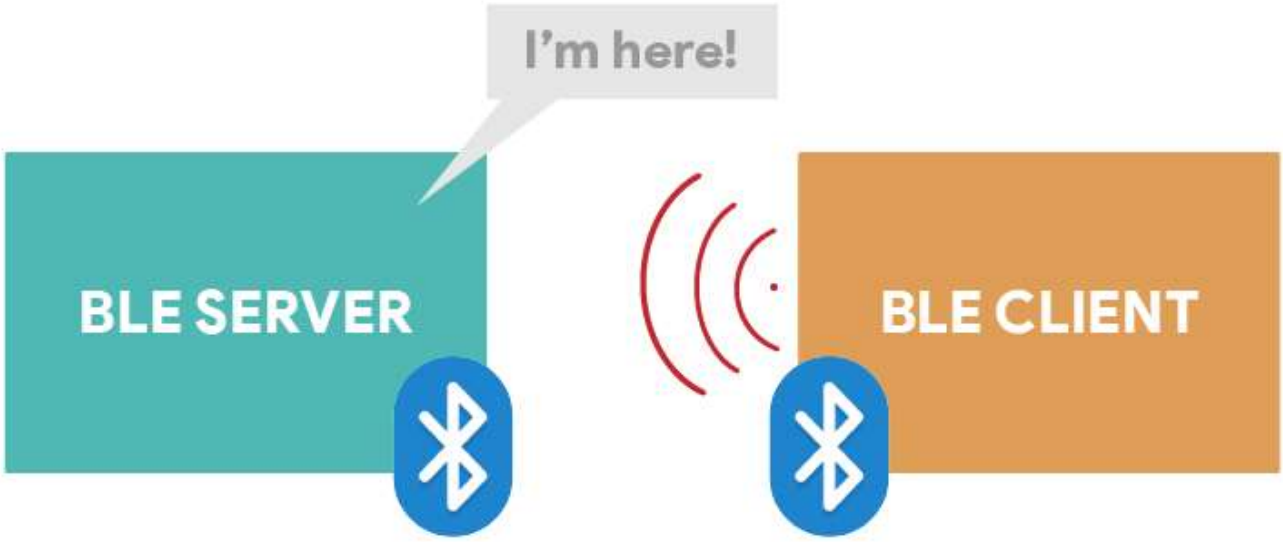
Bluetooth Low Energy, BLE for short, is a power-conserving variant of Bluetooth. BLE's primary application is short-distance transmission of small amounts of data (low bandwidth). Unlike Bluetooth that is always on, BLE remains in sleep mode constantly except for when a connection is initiated.

This makes it consume very low power. BLE consumes approximately 100x less power than Bluetooth (depending on the use case). You can check the main differences between Bluetooth and Bluetooth Low Energy here.

BLE Server and Client

With Bluetooth Low Energy, there are two types of devices: the server and the client. The ESP32 can act either as a client or as a server.

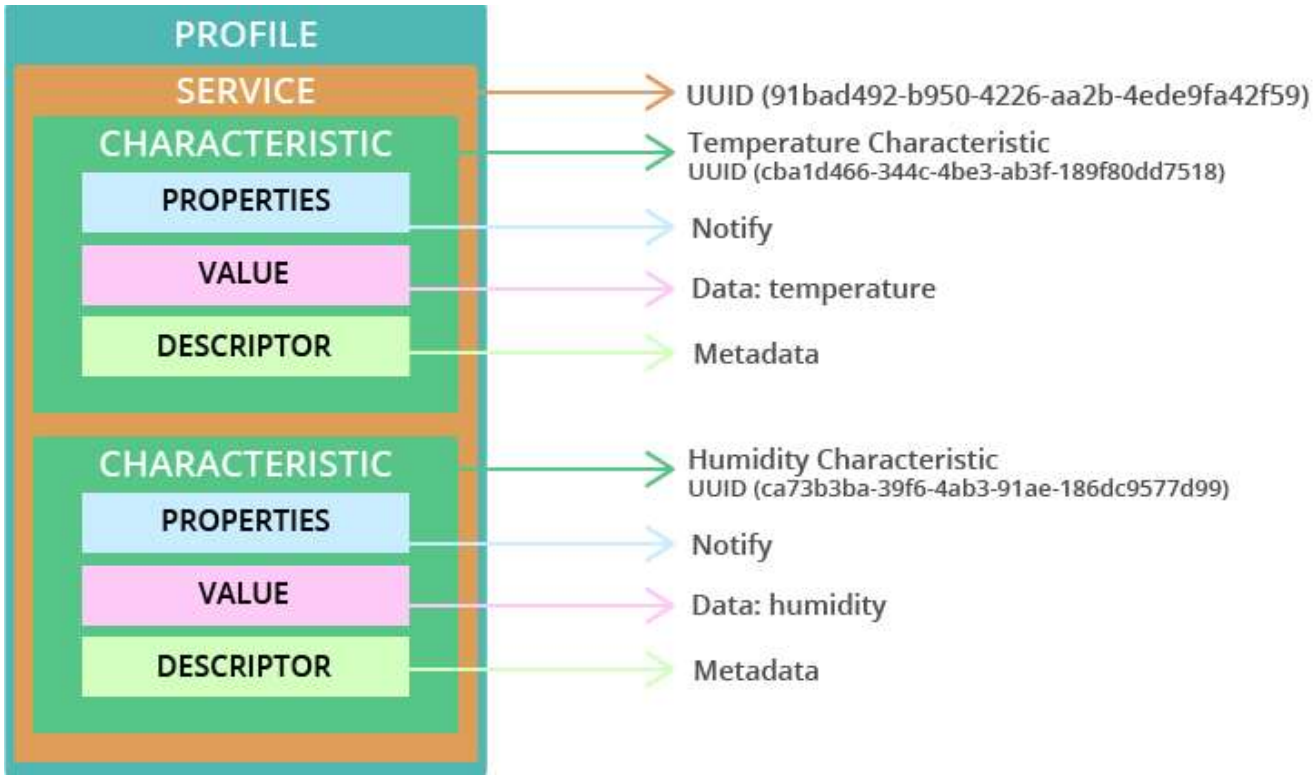
The server advertises its existence, so it can be found by other devices and contains data that the client can read. The client scans the nearby devices, and when it finds the server it is looking for, it establishes a connection and listens for incoming data. This is called point-to-point communication.



There are other possible communication modes like broadcast mode and mesh network (not covered in this tutorial).

GATT

GATT stands for Generic Attributes and it defines a hierarchical data structure that is exposed to connected BLE devices. This means that GATT defines the way that two BLE devices send and receive standard messages. Understanding this hierarchy is important because it will make it easier to understand how to use BLE with the ESP32.



- **Profile**: standard collection of services for a specific use case;
- **Service**: collection of related information, like sensor readings, battery level, heart rate, etc. ;
- **Characteristic**: it is where the actual data is saved on the hierarchy (**value**);
- **Descriptor**: metadata about the data;
- **Properties**: describe how the characteristic value can be interacted with. For example: read, write, notify, broadcast, indicate, etc.

In our example, we'll create a service with two *characteristics*. One for the temperature and another for the humidity. The actual temperature and humidity readings are saved on the *value* under their *characteristics*. Each characteristic has the *notify* property, so that it notifies the client whenever the values change.

UUID

Each service, characteristic, and descriptor have a UUID (Universally Unique Identifier). A UUID is a unique 128-bit (16 bytes) number. For example:

```
55072829-bc9e-4c53-938a-74a6d4c78776
```

There are shortened UUIDs for all types, services, and profiles specified in the SIG (Bluetooth Special Interest Group).

But if your application needs its own UUID, you can generate it using this UUID generator website.

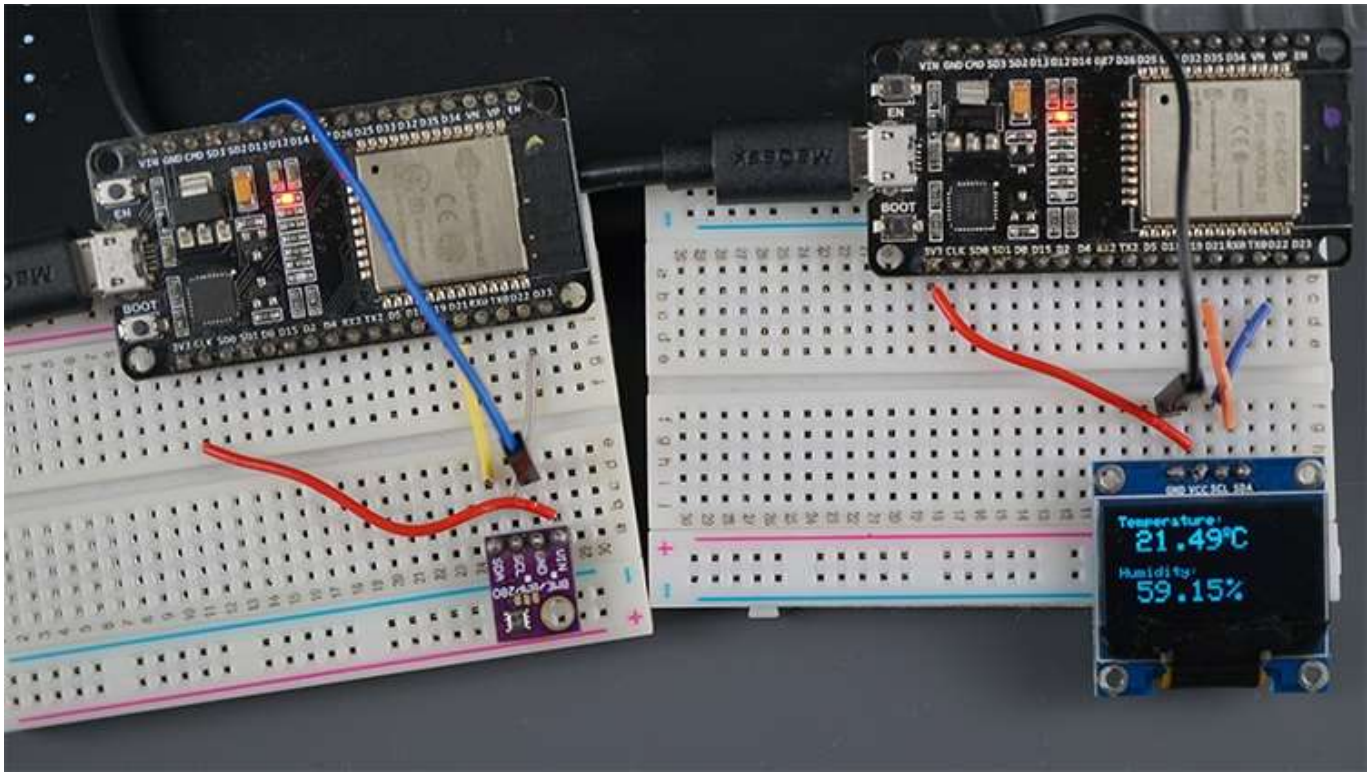
In summary, the UUID is used for uniquely identifying information. For example, it can identify a particular service provided by a Bluetooth device.

For a more detailed introduction about BLE, read our getting started guide:

- Getting Started with ESP32 Bluetooth Low Energy (BLE) on Arduino IDE

Project Overview

In this tutorial, you're going to learn how to make a BLE connection between two ESP32 boards. One ESP32 is going to be the BLE server, and the other ESP32 will be the BLE client.



The ESP32 BLE server is connected to a BME280 sensor and it updates its temperature and humidity characteristic values every 30 seconds.

The ESP32 client connects to the BLE server and it is notified of its temperature and humidity characteristic values. This ESP32 is connected to an OLED display and it prints the latest readings.

This project is divided into two parts:

- **Part 1** – ESP32 BLE server
- **Part 2** – ESP32 BLE client

Parts Required

Here's a list of the parts required to follow this project:

ESP32 BLE Server:

- ESP32 DOIT DEVKIT V1 Board (read Best ESP32 development boards)
- BME280 Sensor
- Jumper wires

- Breadboard
- Smartphone with Bluetooth (optional)

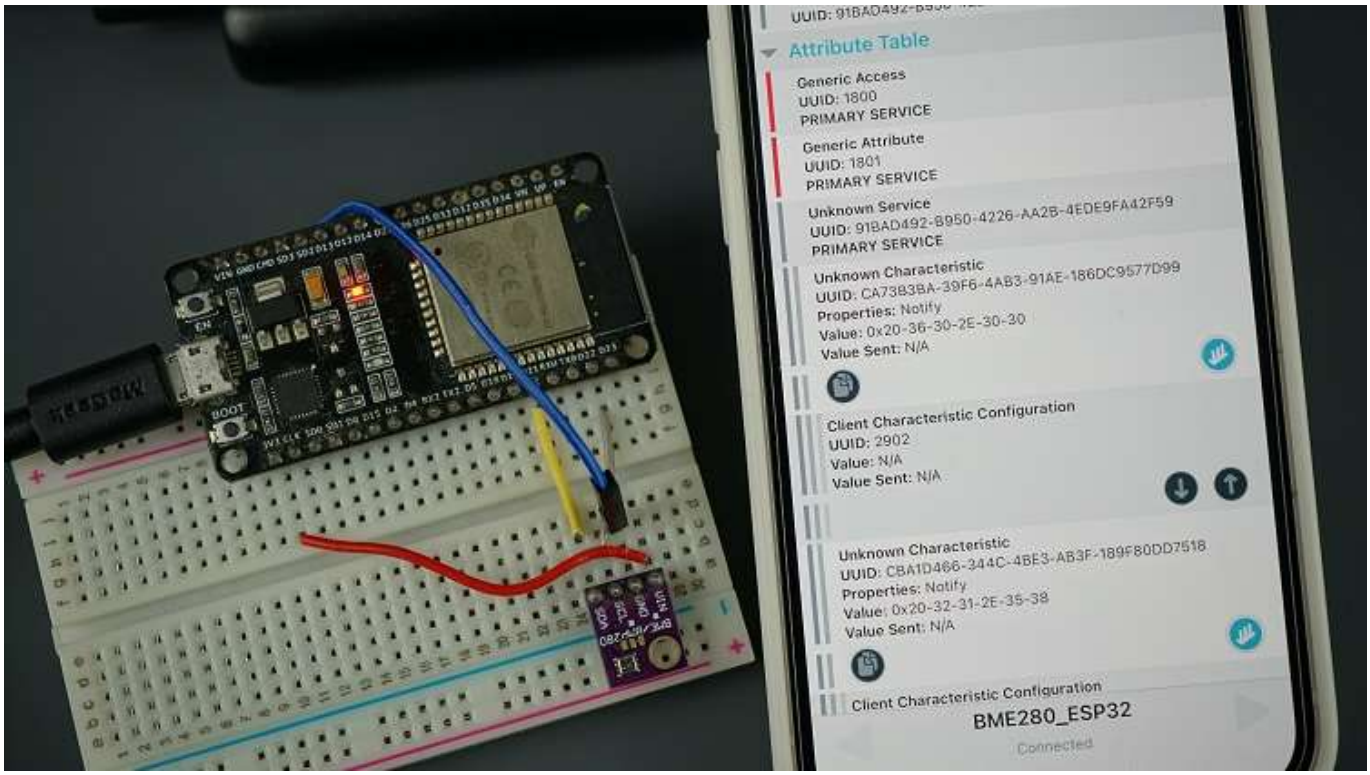
ESP32 BLE Client:

- ESP32 DOIT DEVKIT V1 Board (read Best ESP32 development boards)
- OLED display
- Jumper wires
- Breadboard

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!

1) ESP32 BLE Server

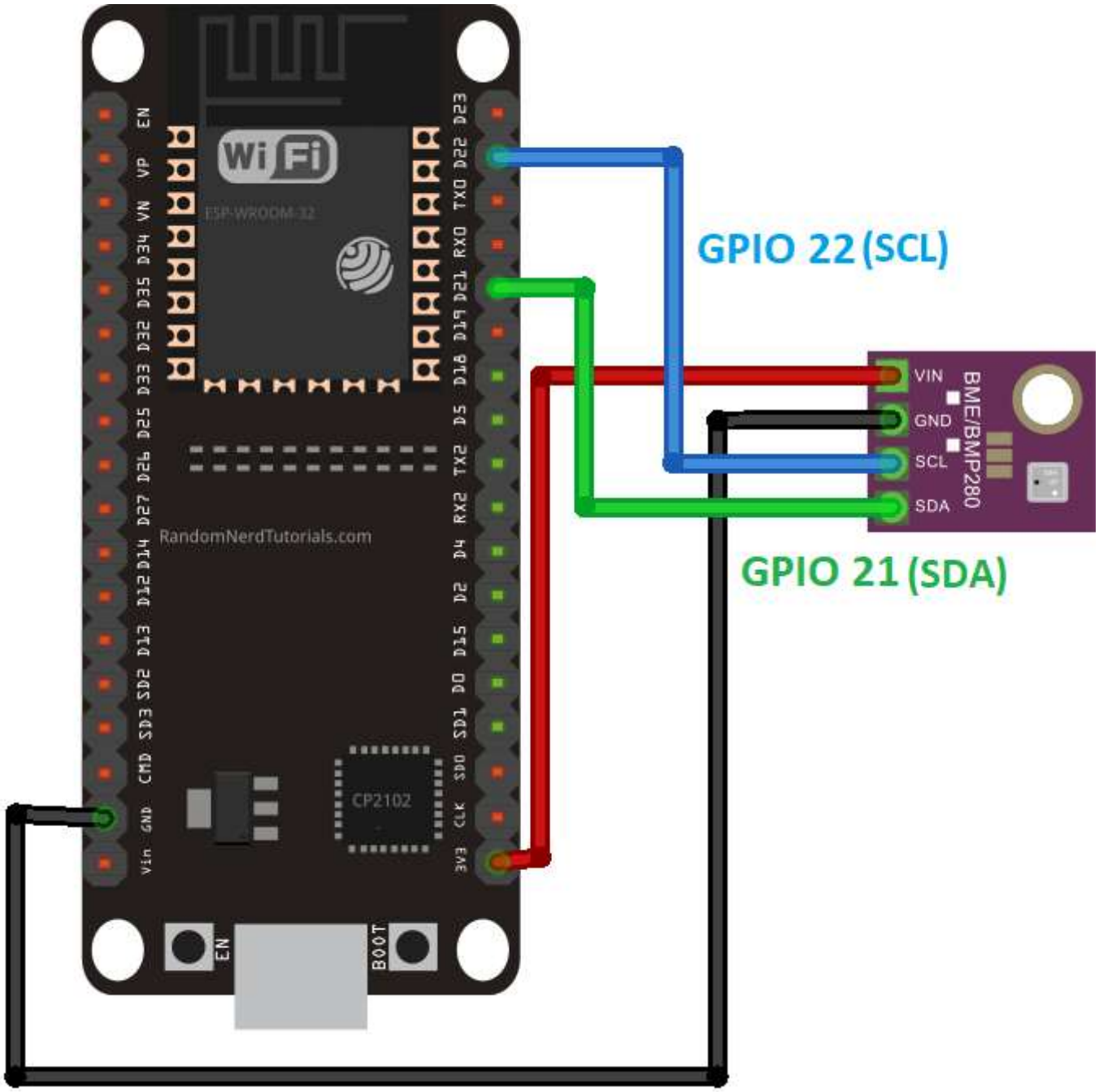
In this part, we'll set up the BLE Server that advertises a service that contains two characteristics: one for temperature and another for humidity. Those characteristics have the *Notify* property to notify new values to the client.



Schematic Diagram

The ESP32 BLE server will advertise characteristics with temperature and humidity from a BME280 sensor. You can use any other sensor as long as you add the required lines in the code.

We're going to use I2C communication with the BME280 sensor module. For that, wire the sensor to the default ESP32 SCL (GPIO 22) and SDA (GPIO 21) pins, as shown in the following schematic diagram.



Recommended reading: ESP32 Pinout Reference: Which GPIO pins should you use?

Installing BME280 Libraries

As mentioned previously, we'll advertise sensor readings from a BME280 sensor. So, you need to install the libraries to interface with the BME280 sensor.

- Adafruit_BME280 library
- Adafruit_Sensor library

You can install the libraries using the Arduino Library Manager. Go to **Sketch** > **Include Library** > **Manage Libraries** and search for the library name.

Installing Libraries (VS Code + PlatformIO)

If you're using VS Code with the PlatformIO extension, copy the following to the `platformio.ini` file to include the libraries.

```
lib_deps = adafruit/Adafruit Unified Sensor @ ^1.1.4
           adafruit/Adafruit BME280 Library @ ^2.1.2
```

ESP32 BLE Server – Code

With the circuit ready and the required libraries installed, copy the following code to the Arduino IDE, or to the `main.cpp` file if you're using VS Code.

```
/*********
  Rui Santos
  Complete instructions at https://RandomNerdTutorials.com/esp32-b
  Permission is hereby granted, free of charge, to any person obta
  The above copyright notice and this permission notice shall be i
*********/

#include <BLEDevice.h>
```

```
#include <BLEServer.h>
#include <BLEUtils.h>
#include <BLE2902.h>
#include <Wire.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_BME280.h>

//Default Temperature is in Celsius
//Comment the next line for Temperature in Fahrenheit
#define temperatureCelsius

//BLE server name
#define bleServerName "BME280_ESP32"

Adafruit_BME280 bme; // I2C

float temp;
float tempF;
```

View raw code

You can upload the code, and it will work straight away advertising its service with the temperature and humidity characteristics. Continue reading to learn how the code works, or skip to the Client section.

There are several examples showing how to use BLE with the ESP32 in the *Examples* section. In your Arduino IDE, go to **File** > **Examples** > **ESP32 BLE Arduino**. This server sketch is based on the *Notify* example.

Importing Libraries

The code starts by importing the required libraries.

```
#include <BLEDevice.h>
#include <BLEServer.h>
```

```
#include <BLEUtils.h>
#include <BLE2902.h>
#include <Wire.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_BME280.h>
```

Choosing Temperature Unit

By default, the ESP sends the temperature in Celsius degrees. You can comment the following line or delete it to send the temperature in Fahrenheit degrees.

```
//Comment the next line for Temperature in Fahrenheit
#define temperatureCelsius
```

BLE Server Name

The following line defines a name for our BLE server. Leave the default BLE server name. Otherwise, the server name in the client code also needs to be changed (because they have to match).

```
//BLE server name
#define bleServerName "BME280_ESP32"
```

BME280 Sensor

Create an `Adafruit_BME280` object called `bme` on the default ESP32 I2C pins.

```
Adafruit_BME280 bme; // I2C
```

The `temp`, `tempF` and `hum` variables will hold the temperature in Celsius degrees, the temperature in Fahrenheit degrees, and the humidity read from the BME280 sensor.

```
float temp;
float tempF;
float hum;
```

Other Variables

The following timer variables define how frequently we want to write to the temperature and humidity characteristic. We set the `timerDelay` variable to 30000 milliseconds (30 seconds), but you can change it.

```
// Timer variables
unsigned long lastTime = 0;
unsigned long timerDelay = 30000;
```

The `deviceConnected` boolean variable allows us to keep track if a client is connected to the server.

```
bool deviceConnected = false;
```

BLE UUIDs

In the next lines, we define UUIDs for the service, for the temperature characteristic in celsius, for the temperature characteristic in Fahrenheit, and for the humidity.

```
// https://www.uuidgenerator.net/
#define SERVICE_UUID "91bad492-b950-4226-aa2b-4ede9fa42f59"

// Temperature Characteristic and Descriptor
#ifdef temperatureCelsius
  BLECharacteristic bmeTemperatureCelsiusCharacteristics("cba1d466-
  BLEDescriptor bmeTemperatureCelsiusDescriptor(BLEUUID((uint16_t)0
#else
```

```
  BLECharacteristic bmeTemperatureFahrenheitCharacteristics("f78ebb
  BLEDescriptor bmeTemperatureFahrenheitDescriptor(BLEUUID((uint16_
#endif

// Humidity Characteristic and Descriptor
BLECharacteristic bmeHumidityCharacteristics("ca73b3ba-39f6-4ab3-91
BLEDescriptor bmeHumidityDescriptor(BLEUUID((uint16_t)0x2903));
```

I recommend leaving all the default UUIDs. Otherwise, you also need to change the code on the client side—so the client can find the service and retrieve the characteristic values.

setup()

In the `setup()`, initialize the Serial Monitor and the BME280 sensor.

```
// Start serial communication
Serial.begin(115200);

// Init BME Sensor
initBME();
```

Create a new BLE device with the BLE server name you've defined earlier:

```
// Create the BLE Device
BLEDevice::init(bleServerName);
```

Set the BLE device as a server and assign a callback function.

```
// Create the BLE Server
BLEServer *pServer = BLEDevice::createServer();
```

```
pServer->setCallbacks(new MyServerCallbacks());
```

The callback function `MyServerCallbacks()` changes the boolean variable `deviceConnected` to `true` or `false` according to the current state of the BLE device. This means that if a client is connected to the server, the state is `true`. If the client disconnects, the boolean variable changes to `false`. Here's the part of the code that defines the `MyServerCallbacks()` function.

```
//Setup callbacks onConnect and onDisconnect
class MyServerCallbacks: public BLEServerCallbacks {
  void onConnect(BLEServer* pServer) {
    deviceConnected = true;
  };
  void onDisconnect(BLEServer* pServer) {
    deviceConnected = false;
  }
};
```

Start a BLE service with the service UUID defined earlier.

```
BLEService *bmeService = pServer->createService(SERVICE_UUID);
```

Then, create the temperature BLE characteristic. If you're using Celsius degrees it sets the following characteristic and descriptor:

```
#ifdef temperatureCelsius
  bmeService->addCharacteristic(&bmeTemperatureCelsiusCharacteristi
  bmeTemperatureCelsiusDescriptor.setValue("BME temperature Celsius
  bmeTemperatureCelsiusCharacteristics.addDescriptor(new BLE2902())
```

Otherwise, it sets the Fahrenheit characteristic:

```
#else
  bmeService->addCharacteristic(&dhtTemperatureFahrenheitCharacteri
  bmeTemperatureFahrenheitDescriptor.setValue("BME temperature Fahr
  bmeTemperatureFahrenheitCharacteristics.addDescriptor(new BLE2902
#endif
```

After that, it sets the humidity characteristic:

```
// Humidity
bmeService->addCharacteristic(&bmeHumidityCharacteristics);
bmeHumidityDescriptor.setValue("BME humidity");
bmeHumidityCharacteristics.addDescriptor(new BLE2902());
```

Finally, you start the service, and the server starts the advertising so other devices can find it.

```
// Start the service
bmeService->start();

// Start advertising
BLEAdvertising *pAdvertising = BLEDevice::getAdvertising();
pAdvertising->addServiceUUID(SERVICE_UUID);
pServer->getAdvertising()->start();
Serial.println("Waiting a client connection to notify...");
```

loop()

The `loop()` function is fairly straightforward. You constantly check if the device is connected to a client or not. If it's connected, and the `timerDelay` has passed, it reads the current temperature and humidity.

```
if (deviceConnected) {
  if ((millis() - lastTime) > timerDelay) {
    // Read temperature as Celsius (the default)
    temp = bme.readTemperature();
    // Fahrenheit
    tempF = temp*1.8 +32;
    // Read humidity
    hum = bme.readHumidity();
```

If you're using temperature in Celsius it runs the following code section. First, it converts the temperature to a char variable (`temperatureCTemp` variable). We must convert the temperature to a char variable type to use it in the `setValue()` function.

```
static char temperatureCTemp[6];
dtostrf(temp, 6, 2, temperatureCTemp);
```

Then, it sets the `bmeTemperatureCelsiusCharacteristic` value to the new temperature value (`temperatureCTemp`) using the `setValue()` function. After settings the new value, we can notify the connected client using the `notify()` function.

```
//Set temperature Characteristic value and notify connected client
bmeTemperatureCelsiusCharacteristics.setValue(temperatureCTemp);
bmeTemperatureCelsiusCharacteristics.notify();
```

We follow a similar procedure for the Temperature in Fahrenheit.

```
#else
    static char temperatureFTemp[6];
    dtostrf(f, 6, 2, temperatureFTemp);
    //Set temperature Characteristic value and notify connected cli
    bmeTemperatureFahrenheitCharacteristics.setValue(tempF);
```

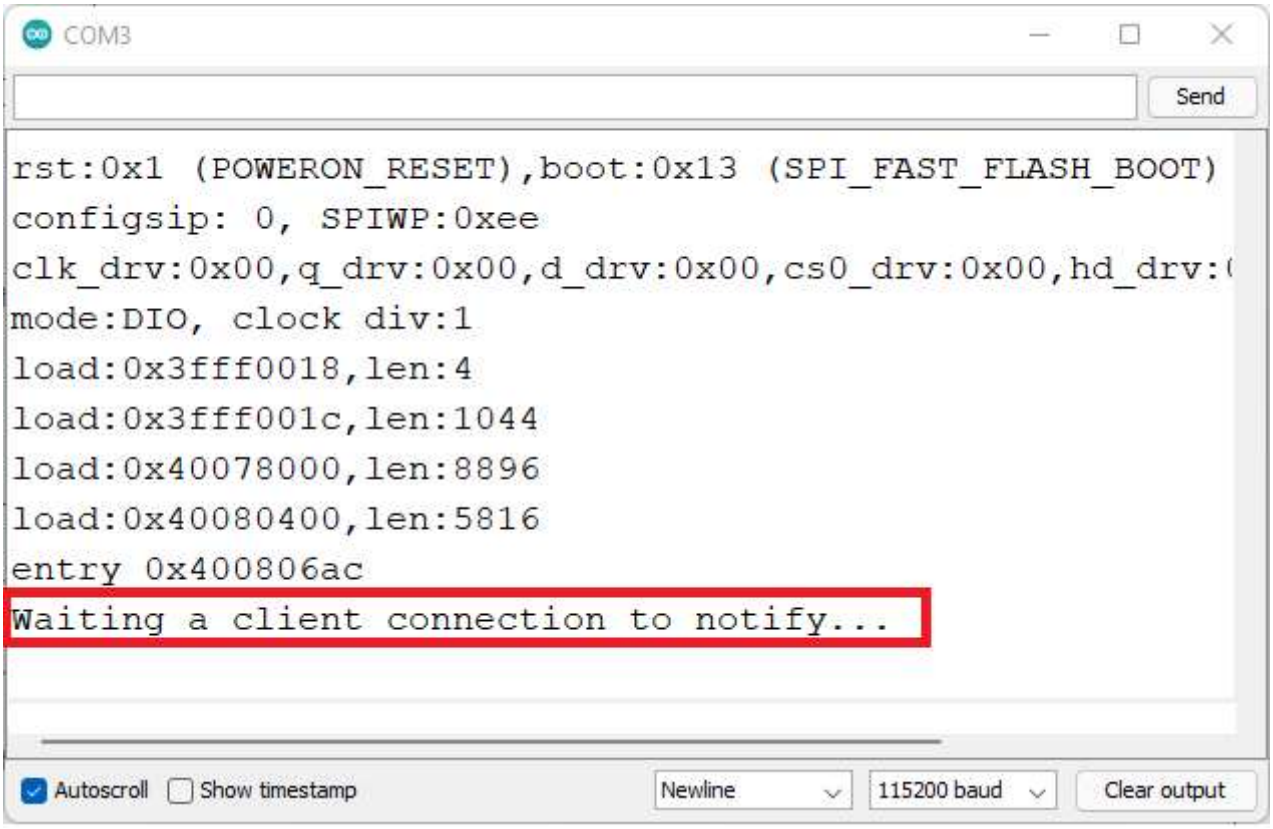
```
    bmeTemperatureFahrenheitCharacteristics.notify();
    Serial.print("Temperature Fahrenheit: ");
    Serial.print(tempF);
    Serial.print(" *F");
#endif
```

Sending the humidity also uses the same process.

```
//Notify humidity reading from DHT
static char humidityTemp[6];
dtostrf(hum, 6, 2, humidityTemp);
//Set humidity Characteristic value and notify connected client
bmeHumidityCharacteristics.setValue(humidityTemp);
bmeHumidityCharacteristics.notify();
Serial.print(" - Humidity: ");
Serial.print(hum);
Serial.println(" %");
```

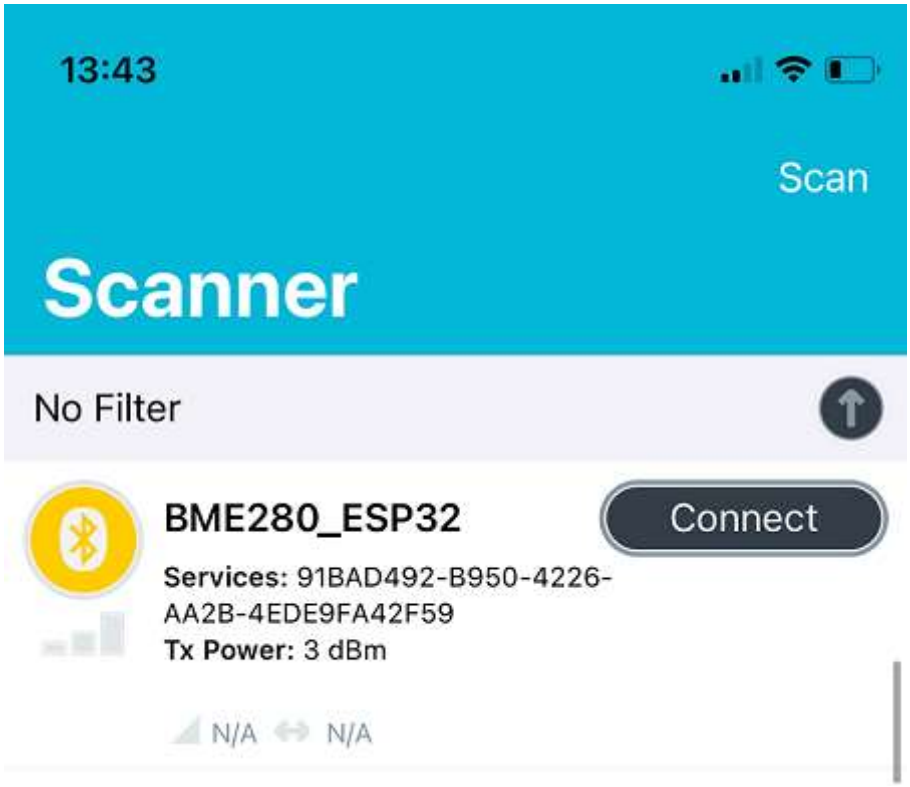
Testing the ESP32 BLE Server

Upload the code to your board and then, open the Serial Monitor. It will display a message as shown below.

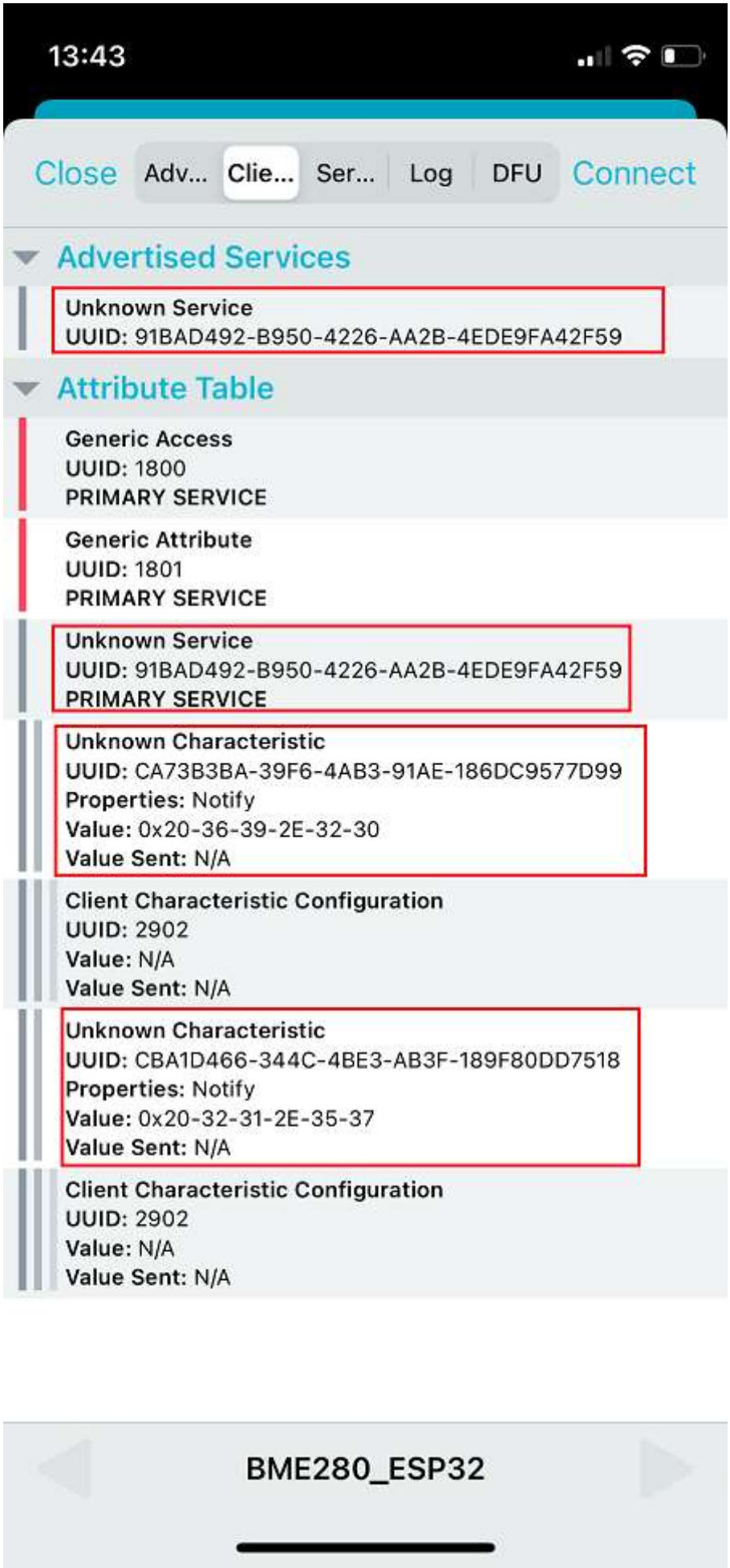


Then, you can test if the BLE server is working as expected by using a BLE scan application on your smartphone like **nRF Connect**. This application is available for Android and iOS.

After installing the application, enable Bluetooth on your smartphone. Open the nRF Connect app and click on the Scan button. It will find all Bluetooth nearby devices, including your **BME280_ESP32** device (it is the BLE server name you defined on the code).



Connect to your BME280_ESP32 device and then, select the client tab (the interface might be slightly different). You can check that it advertises the service with the UUID we defined in the code, as well as the temperature and humidity characteristics. Notice that those characteristics have the *Notify* property.


13:43
Close
Adv...
Clie...
Ser...
Log
DFU
Connect
Advertised Services
Unknown Service
UUID: 91BAD492-B950-4226-AA2B-4EDE9FA42F59
Attribute Table
Generic Access
UUID: 1800
PRIMARY SERVICE
Generic Attribute
UUID: 1801
PRIMARY SERVICE
Unknown Service
UUID: 91BAD492-B950-4226-AA2B-4EDE9FA42F59
PRIMARY SERVICE
Unknown Characteristic
UUID: CA73B3BA-39F6-4AB3-91AE-186DC9577D99
Properties: Notify
Value: 0x20-36-39-2E-32-30
Value Sent: N/A
Client Characteristic Configuration
UUID: 2902
Value: N/A
Value Sent: N/A
Unknown Characteristic
UUID: CBA1D466-344C-4BE3-AB3F-189F80DD7518
Properties: Notify
Value: 0x20-32-31-2E-35-37
Value Sent: N/A
Client Characteristic Configuration
UUID: 2902
Value: N/A
Value Sent: N/A
BME280_ESP32

Your ESP32 BLE Server is ready!

Go to the next section to create an ESP32 client that connects to the server to get access to the temperature and humidity characteristics and get the readings to display them on an OLED display.

---

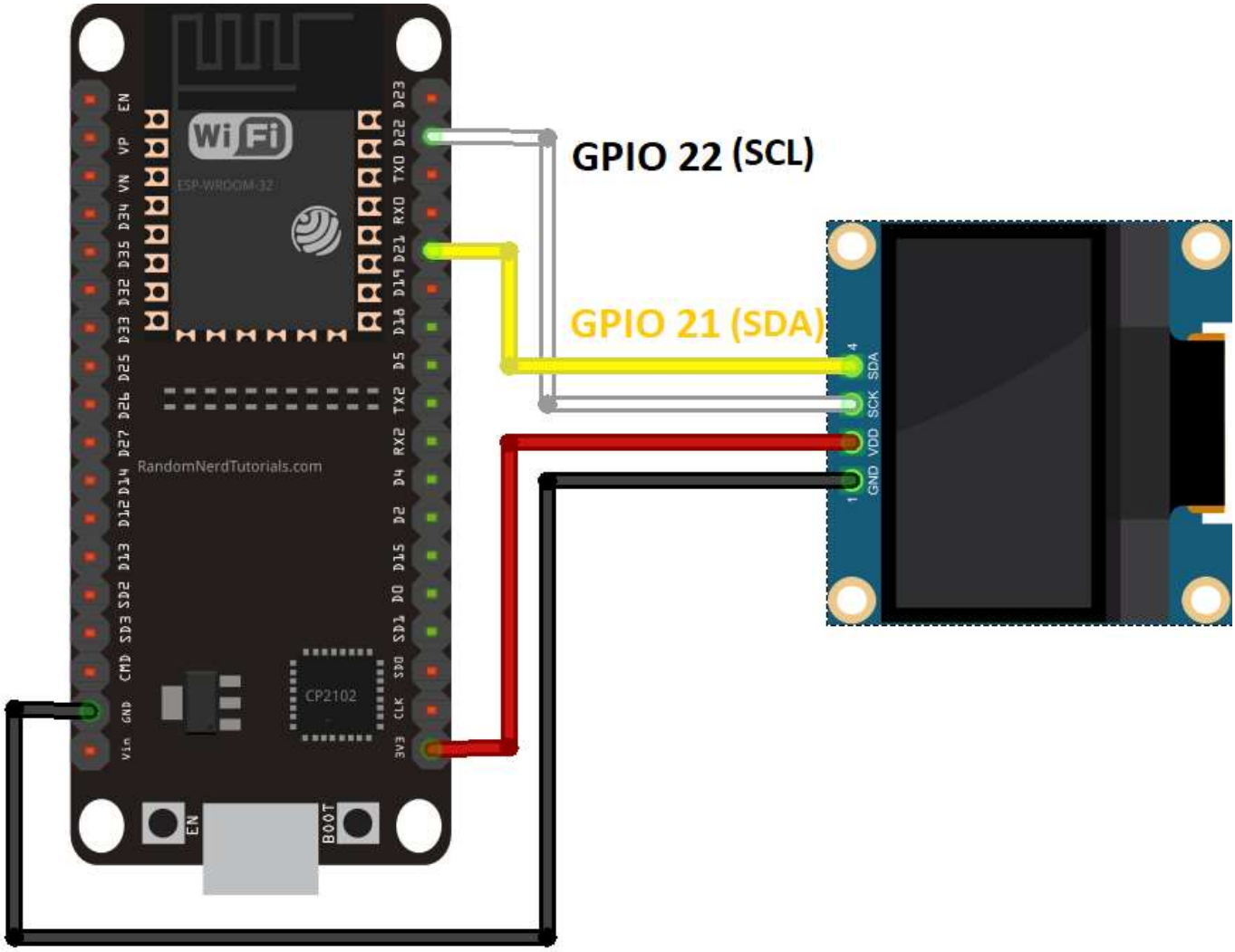
2) ESP32 BLE Client

In this section, we'll create the ESP32 BLE client that will establish a connection with the ESP32 BLE server, and display the readings on an OLED display.

Schematic

The ESP32 BLE client is connected to an OLED display. The display shows the readings received via Bluetooth.

Wire your OLED display to the ESP32 by following the next schematic diagram. The SCL pin connects to GPIO 22 and the SDA pin to GPIO 21 .



Installing the SSD1306, GFX and BusIO Libraries

You need to install the following libraries to interface with the OLED display:

- Adafruit_SSD1306 library
- Adafruit GFX library
- Adafruit BusIO library

To install the libraries, go **Sketch**> **Include Library** > **Manage Libraries**, and search for the libraries' names.

Installing Libraries (VS Code + PlatformIO)

If you're using VS Code with the PlatformIO extension, copy the following to the `platformio.ini` file to include the libraries.

```
lib_deps =
    adafruit/Adafruit GFX Library@^1.10.12
    adafruit/Adafruit SSD1306@^2.4.6
```

ESP32 BLE Client – Code

Copy the BLE client Sketch to your Arduino IDE or to the `main.cpp` file if you're using VS Code with PlatformIO.

```
/*********
  Rui Santos
  Complete instructions at https://RandomNerdTutorials.com/esp32-l
  Permission is hereby granted, free of charge, to any person obta
  The above copyright notice and this permission notice shall be i
*********/

#include "BLEDevice.h"
#include <Wire.h>
#include <Adafruit_SSD1306.h>
#include <Adafruit_GFX.h>

//Default Temperature is in Celsius
//Comment the next line for Temperature in Fahrenheit
#define temperatureCelsius

//BLE Server name (the other ESP32 name running the server sketch
#define bleServerName "BME280_ESP32"

/* UUID's of the service, characteristic that we want to read*/
// BLE Service
static BLEUUID bmeServiceUUID("91bad492-b950-4226-aa2b-4ede9fa42f

// BLE Characteristics
#ifdef temperatureCelsius
```

View raw code

Continue reading to learn how the code works or skip to the Demonstration section.

Importing libraries

You start by importing the required libraries:

```
#include "BLEDevice.h"
#include <Wire.h>
#include <Adafruit_SSD1306.h>
#include <Adafruit_GFX.h>
```

Choosing temperature unit

By default the client will receive the temperature in Celsius degrees, if you comment the following line or delete it, it will start receiving the temperature in Fahrenheit degrees.

```
//Default Temperature is in Celsius
//Comment the next line for Temperature in Fahrenheit
#define temperatureCelsius
```

BLE Server Name and UUIDs

Then, define the BLE server name that we want to connect to and the service and characteristic UUIDs that we want to read. Leave the default BLE server name and UUIDs to match the ones defined in the server sketch.

```
//BLE Server name (the other ESP32 name running the server sketch)
#define bleServerName "BME280_ESP32"
```

```
/* UUID's of the service, characteristic that we want to read*/
// BLE Service
static BLEUUID bmeServiceUUID("91bad492-b950-4226-aa2b-4ede9fa42f59

// BLE Characteristics
#ifdef temperatureCelsius
  //Temperature Celsius Characteristic
  static BLEUUID temperatureCharacteristicUUID("cba1d466-344c-4be3-
#else
  //Temperature Fahrenheit Characteristic
  static BLEUUID temperatureCharacteristicUUID("f78ebbff-c8b7-4107-
#endif

// Humidity Characteristic
static BLEUUID humidityCharacteristicUUID("ca73b3ba-39f6-4ab3-91ae-
```

Declaring variables

Then, you need to declare some variables that will be used later with Bluetooth to check whether we're connected to the server or not.

```
//Flags stating if should begin connecting and if the connection is
static boolean doConnect = false;
static boolean connected = false;
```

Create a variable of type `BLEAddress` that refers to the address of the server we want to connect. This address will be found during scanning.

```
//Address of the peripheral device. Address will be found during sc
static BLEAddress *pServerAddress;
```

Set the characteristics we want to read (temperature and humidity).

```
//Characteristicd that we want to read
static BLERemoteCharacteristic* temperatureCharacteristic;
static BLERemoteCharacteristic* humidityCharacteristic;
```

OLED Display

You also need to declare some variables to work with the OLED. Define the OLED width and height:

```
#define SCREEN_WIDTH 128 // OLED display width, in pixels
#define SCREEN_HEIGHT 64 // OLED display height, in pixels
```

Instantiate the OLED display with the width and height defined earlier.

```
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire);
```

Temperature and Humidity Variables

Define char variables to hold the temperature and humidity values received by the server.

```
//Variables to store temperature and humidity
char* temperatureChar;
char* humidityChar;
```

The following variables are used to check whether new temperature and humidity readings are available and if it is time to update the OLED display.

```
//Flags to check whether new temperature and humidity readings are
boolean newTemperature = false;
boolean newHumidity = false;
```

printReadings()

We created a function called `printReadings()` that displays the temperature and humidity readings on the OLED display.

```
void printReadings(){

  display.clearDisplay();
  // display temperature
  display.setTextSize(1);
  display.setCursor(0,0);
  display.print("Temperature: ");
  display.setTextSize(2);
  display.setCursor(0,10);
  display.print(temperatureChar);
  display.print(" ");
  display.setTextSize(1);
  display.cp437(true);
  display.write(167);
  display.setTextSize(2);
  Serial.print("Temperature:");
  Serial.print(temperatureChar);
  #ifdef temperatureCelsius
    //Temperature Celsius
    display.print("C");
    Serial.print("C");
  #else
    //Temperature Fahrenheit
    display.print("F");
```

```
    Serial.print("F");
  #endif

  //display humidity
  display.setTextSize(1);
  display.setCursor(0, 35);
  display.print("Humidity: ");
  display.setTextSize(2);
  display.setCursor(0, 45);
  display.print(humidityChar);
  display.print("%");
  display.display();
  Serial.print(" Humidity:");
  Serial.print(humidityChar);
  Serial.println("%");
}
```

Recommended reading: ESP32 OLED Display with Arduino IDE

setup()

In the `setup()`, start the OLED display.

```
//OLED display setup
// SSD1306_SWITCHCAPVCC = generate display voltage from 3.3V intern
if(!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) { // Address 0x3C fo
  Serial.println(F("SSD1306 allocation failed"));
  for(;;); // Don't proceed, loop forever
}
```

Then, print a message in the first line saying “BME SENSOR”.

```
  display.clearDisplay();
  display.setTextSize(2);
  display.setTextColor(WHITE,0);
  display.setCursor(0,25);
  display.print("BLE Client");
  display.display();
```

Start the serial communication at a baud rate of 115200.

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

And initialize the BLE device.

```
  //Init BLE device
  BLEDevice::init("");
```

Scan nearby devices

The following methods scan for nearby devices.

```
  // Retrieve a Scanner and set the callback we want to use to be inf
  // have detected a new device.  Specify that we want active scannin
  // scan to run for 30 seconds.
  BLEScan* pBLEScan = BLEDevice::getScan();
  pBLEScan->setAdvertisedDeviceCallbacks(new MyAdvertisedDeviceCallba
  pBLEScan->setActiveScan(true);
  pBLEScan->start(30);
```

MyAdvertisedDeviceCallbacks() function

Note that the `MyAdvertisedDeviceCallbacks()` function, upon finding a BLE device, checks if the device found has the right BLE server name. If it has, it stops the scan and changes the `doConnect` boolean variable to `true`. This way we know that we found the server we're looking for, and we can start establishing a connection.

```
//Callback function that gets called, when another device's adverti
class MyAdvertisedDeviceCallbacks: public BLEAdvertisedDeviceCallba
  void onResult(BLEAdvertisedDevice advertisedDevice) {
    if (advertisedDevice.getName() == bleServerName) { //Check if t
      advertisedDevice.getScan()->stop(); //Scan can be stopped, we
      pServerAddress = new BLEAddress(advertisedDevice.getAddress()
      doConnect = true; //Set indicator, stating that we are ready
      Serial.println("Device found. Connecting!");
    }
  }
};
```

Connect to the server

If the `doConnect` variable is `true`, it tries to connect to the BLE server. The `connectToServer()` function handles the connection between the client and the server.

```
//Connect to the BLE Server that has the name, Service, and Charact
bool connectToServer(BLEAddress pAddress) {
   BLEClient* pClient = BLEDevice::createClient();

  // Connect to the remove BLE Server.
  pClient->connect(pAddress);
  Serial.println(" - Connected to server");

  // Obtain a reference to the service we are after in the remote B
  BLERemoteService* pRemoteService = pClient->getService(bmeService
```

```
    if (pRemoteService == nullptr) {
      Serial.print("Failed to find our service UUID: ");
      Serial.println(bmeServiceUUID.toString().c_str());
      return (false);
    }

    // Obtain a reference to the characteristics in the service of th
    temperatureCharacteristic = pRemoteService->getCharacteristic(tem
    humidityCharacteristic = pRemoteService->getCharacteristic(humidi

    if (temperatureCharacteristic == nullptr || humidityCharacteristi
      Serial.print("Failed to find our characteristic UUID");
      return false;
    }
    Serial.println(" - Found our characteristics");

    //Assign callback functions for the Characteristics
    temperatureCharacteristic->registerForNotify(temperatureNotifyCal
    humidityCharacteristic->registerForNotify(humidityNotifyCallback)
    return true;
  }
```

It also assigns a callback function responsible to handle what happens when a new value is received.

```
  //Assign callback functions for the Characteristics
  temperatureCharacteristic->registerForNotify(temperatureNotifyCallb
  humidityCharacteristic->registerForNotify(humidityNotifyCallback);
```

After the BLE client is connected to the server, you need to active the *notify* property for each characteristic. For that, use the `writeValue()` method on the descriptor.

```
  if (connectToServer(*pServerAddress)) {
    Serial.println("We are now connected to the BLE Server.");
    //Activate the Notify property of each Characteristic
    temperatureCharacteristic->getDescriptor(BLEUUID((uint16_t)0x2902
    humidityCharacteristic->getDescriptor(BLEUUID((uint16_t)0x2902))-
```

Notify new values

When the client receives a new notify value, it will call these two functions: `temperatureNotifyCallback()` and `humidityNotifyCallback()` that are responsible for retrieving the new value, update the OLED with the new readings and print them on the Serial Monitor.

```
  //When the BLE Server sends a new temperature reading with the noti
  static void temperatureNotifyCallback(BLERemoteCharacteristic* pBLE
                                        uint8_t* pData, size_t leng
    //store temperature value
    temperatureChar = (char*)pData;
    newTemperature = true;
  }
```

```
  //When the BLE Server sends a new humidity reading with the notify
  static void humidityNotifyCallback(BLERemoteCharacteristic* pBLERem
                                     uint8_t* pData, size_t length,
    //store humidity value
    humidityChar = (char*)pData;
    newHumidity = true;
    Serial.print(newHumidity);
  }
```

These two previous functions are executed every time the BLE server notifies the client with a new value, which happens every 30 seconds. These functions save the values received on the `temperatureChar` and `humidityChar` variables. These also change the `newTemperature` and `newHumidity` variables to `true`, so that we know we've received new readings.

Display new temperature and humidity readings

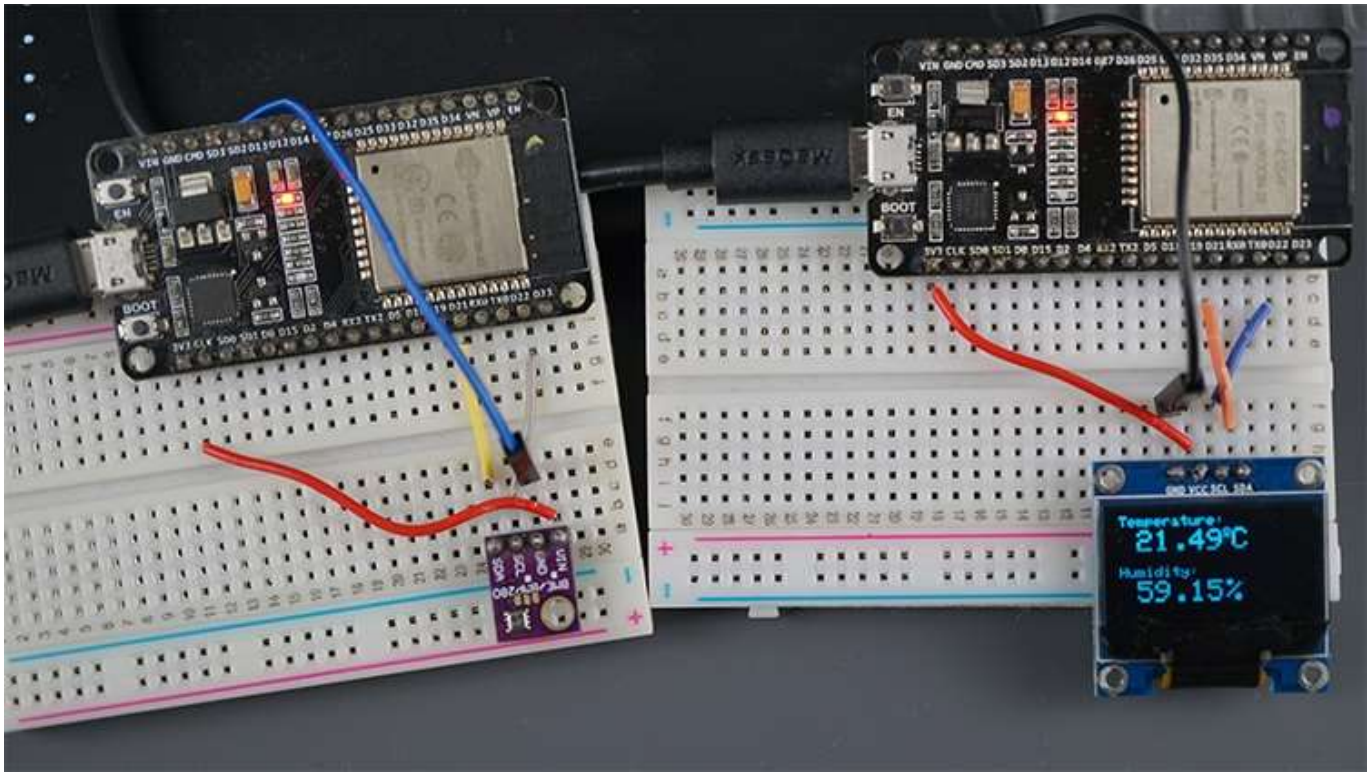
In the `loop()`, there is an if statement that checks if new readings are available. If there are new readings, we se the `newTemperature` and `newHumidity` variables to `false`, so that we are able to receive new readings later on. Then, we call the `printReadings()` function to display the readings on the OLED.

```
//if new temperature readings are available, print in the OLED
if (newTemperature && newHumidity){
  newTemperature = false;
  newHumidity = false;
  printReadings();
}
```

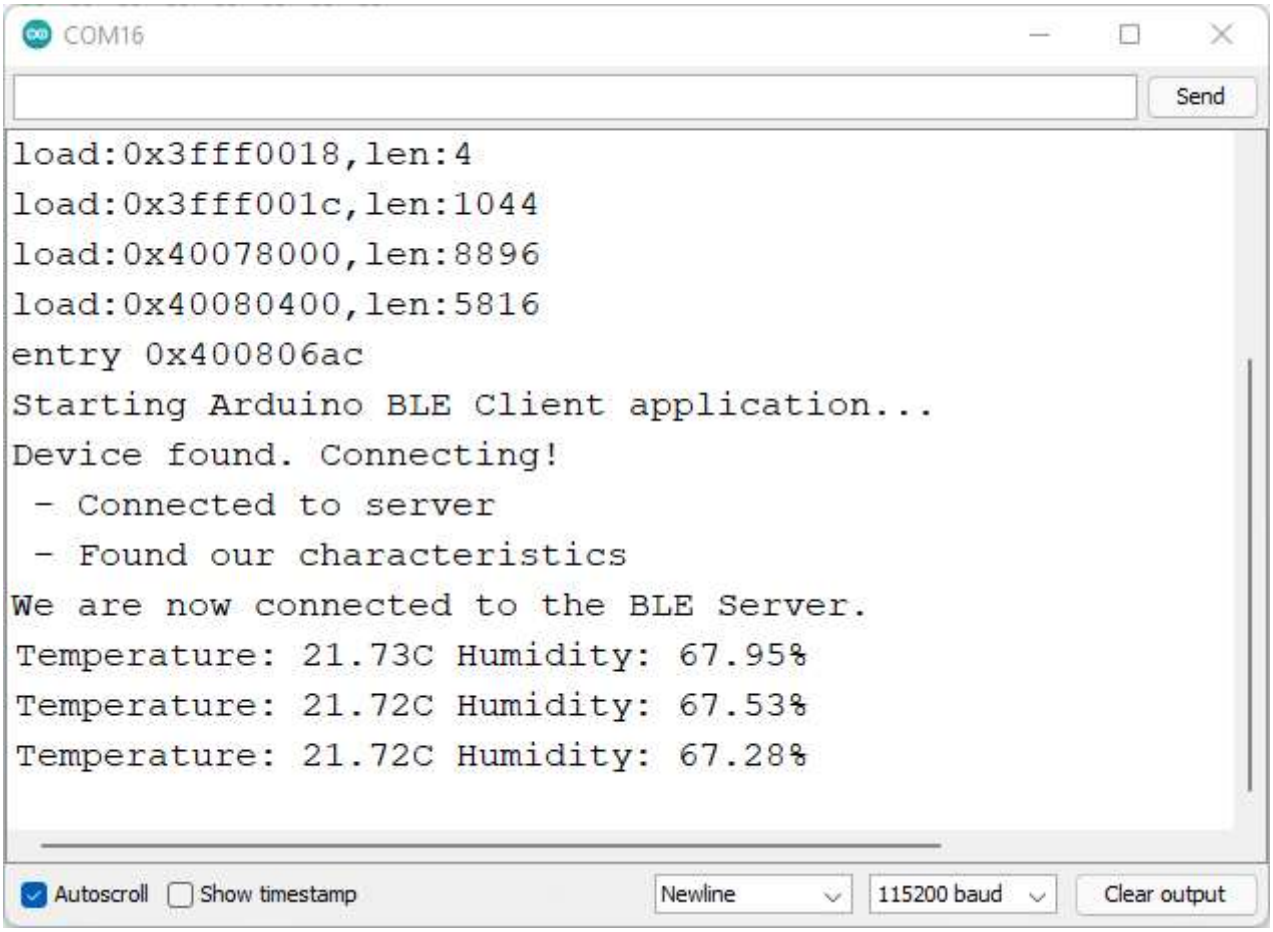
Testing the Project

That's it for the code. You can upload it to your ESP32 board.

Once the code is uploaded. Power the ESP32 BLE server, then power the ESP32 with the client sketch. The client starts scanning nearby devices, and when it finds the other ESP32, it establishes a Bluetooth connection. Every 30 seconds, it updates the display with the latest readings.



Important: don't forget to disconnect your smartphone from the BLE server. Otherwise, the ESP32 BLE Client won't be able to connect to the server.



Wrapping Up

In this tutorial, you learned how to create a BLE Server and a BLE Client with the ESP32. You learned how to set new temperature and humidity values on the BLE server characteristics. Then, other BLE devices (clients) can connect to that server and read those characteristic values to get the latest temperature and humidity values. Those characteristics have the *notify* property, so that the client is notified whenever there's a new value.

Using BLE is another communication protocol you can use with the ESP32 boards besides Wi-Fi. We hope you found this tutorial useful. We have tutorials for other communication protocols that you may find useful.

- ESP32 Bluetooth Classic with Arduino IDE – Getting Started
- ESP32 Useful Wi-Fi Library Functions (Arduino IDE)
- ESP-MESH with ESP32 and ESP8266: Getting Started (painlessMesh library)
- Getting Started with ESP-NOW (ESP32 with Arduino IDE)

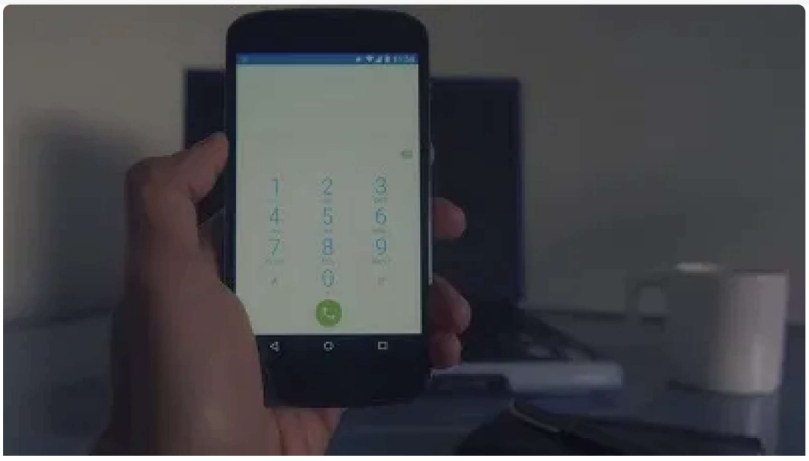
Learn more about the ESP32 with our resources:

- Learn ESP32 with Arduino IDE (eBook + Video Course)
- More ESP32 projects and tutorials…

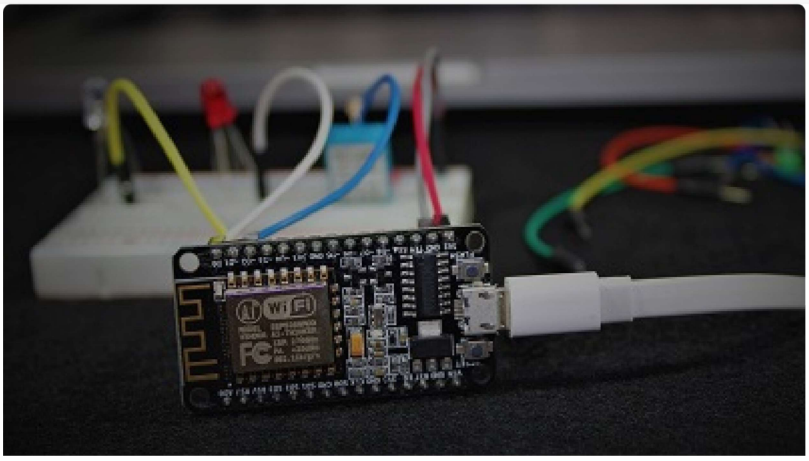
[eBook] Build Web Servers with ESP32 and ESP8266 (2nd Edition)

Build Web Server projects with the ESP32 and ESP8266 boards to control outputs and monitor sensors remotely. Learn HTML, CSS, JavaScript and client-server communication protocols **DOWNLOAD »**

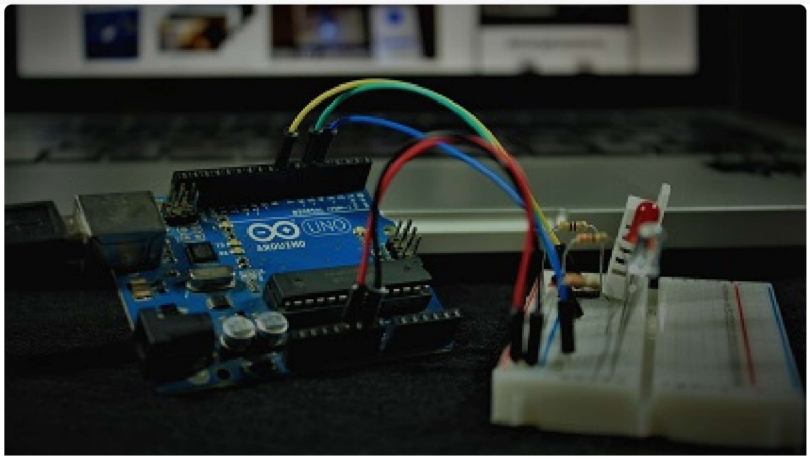
Recommended Resources



Build a Home Automation System from Scratch » With Raspberry Pi, ESP8266, Arduino, and Node-RED.



Home Automation using ESP8266 eBook and video course » Build IoT and home automation projects.



What to Read Next…

Telegram: ESP8266 NodeMCU Motion Detection with Notifications (Arduino IDE)

ESP32/ESP8266: Control Outputs with Web Server and a Physical Button Simultaneously

47 thoughts on “ESP32 BLE Server and Client (Bluetooth Low Energy)”

Carlos Bruni

November 11, 2021 at 3:44 pm

Excelente Tutorial! Parabéns.

Reply

Sara Santos

November 12, 2021 at 11:52 am

Obrigada 🙂

Reply

Allen R Mulvey

November 11, 2021 at 4:43 pm

Should the line in the server:
dtostrf(f, 6, 2, temperatureFTemp);
actually be:
dtostrf(tempF, 6, 2, temperatureFTemp);

Reply

Sara Santos

November 12, 2021 at 11:53 am

Hi.
Yes. You are right.
I'll fix the code.
Thanks for pointing that out.
Regards,
Sara

Reply

Allen R Mulvey

November 14, 2021 at 1:06 am

I'm afraid I noticed a couple of other typos in the server sketch. You apparently switched from dht to bme sensors but missed one reference in line 93:
bmeService->addCharacteristic(&dhtTemperatureFahrenheitCharacteristics);

You also created new descriptors when, I'm sure, you intended to point to the ones created earlier. Lines 95 and 101:
bmeTemperatureFahrenheitCharacteristics.addDescriptor(new BLE2902());
bmeHumidityCharacteristics.addDescriptor(new BLE2902());
should be:
bmeTemperatureCharacteristics.addDescriptor(&bmeTemperatureDescriptor);
bmeHumidityCharacteristics.addDescriptor(&bmeHumidityDescriptor);

Thanks for all the work you do. You are always the first source I go to when I have a problem with ESP microcontrollers.

Reply

Sara Santos

November 14, 2021 at 11:42 am

Hi.
You're right. Thanks for pointing that out.
I'll fix it soon.
I guess I need a vacation. I should have seen that.
Regards,
Sara

Reply

Allen R Mulvey

November 14, 2021 at 1:36 pm

Don't feel too bad. When I looked for more documentation, every example I found created a new Descriptor. None explained how to modify that Descriptor after it was created. I applaud you for putting together such a good sketch when documentation is apparently so hard to find. That is why I always go to RNT first.

Sara Santos

November 15, 2021 at 10:47 am

Thanks.
It is fixed now.
Regards,
Sara

zygfryd
November 11, 2021 at 5:20 pm

superb
and if you add sleeping to the server it will be even more battery friendly 😉

Reply

Sara Santos
November 12, 2021 at 11:53 am

Thank you 🙂

Reply

JB
November 11, 2021 at 6:51 pm

It’s a pleasure to read yet another of your and Sara’s well explained tutorials.

Reply

Sara Santos

November 12, 2021 at 11:43 am

Thank you.

Reply

Bert

November 11, 2021 at 8:02 pm

Hey.
Can you explain this rule to me in more detail?
BLECharacteristic bmeTemperatureCelsiusCharacteristics(“cba1d466-344c-4be3-ab3f-189f80dd7518”, BLECharacteristic::PROPERTY_NOTIFY);

I almost understand how everything works but where does that long number come from or what does it do.
Greetings old man Bert

Reply

Sara Santos

November 12, 2021 at 11:43 am

Hi.
Basically, you need to set UUIDs for your characteristics to identify them.
You can create UUIDs (use the UUID generator website: https://www.uuidgenerator.net/) or use predefined UUIDs (https://www.bluetooth.com/specifications/assigned-numbers/).

The UUIDs are used to identify the characteristics. There are predefined UUIDs for the most common characteristics used by BLE devices, for example the Battery Level has a default UUID. This is useful because other devices that connect, know exactly what to search for to get the information they want.

We also define that we want the property of that characteristic to be notify. The property defines how the client can interact with those characteristics, it can be read, write, notify, and others.

I hope this is clear.

Regards,
Sara

Reply

Bert
November 12, 2021 at 12:31 pm

Hey.
Very clear and understandable.
Sometimes there is a translation problem because I can’t write english.
Certain concepts are then not clear.
You explain everything you do very well and it is very instructive.
I thank you for that.

I'm going to work on it.
Greetings Old man Bert

Reply

Sara Santos
November 12, 2021 at 1:39 pm

That's great!
Thank you.
Regards,
Sara

Reply

Bert
November 12, 2021 at 4:55 pm

You once had a wish list.
If you still have it I'd love to
Esp32 Web server Hosting files from MicroSd Card.
Turns into an Esp32 captive portal Hosting Files from MicroSd Card.
But I have no idea if that's possible.
I do have a version here that does that without SDcart, but that is very inconvenient if you want to change the html.

Sara Santos
November 12, 2021 at 6:25 pm

Hi.
I also don't know. I haven't investigated that.
But, I'll add it to my list.
Regards,
Sara

Helmut
November 11, 2021 at 10:12 pm

Please tell us the power consume from server and client if you can.
And thankyou for this blog

Reply

Sara Santos
November 12, 2021 at 11:54 am

Hi.
Unfortunately, I don't have data about power consumption for this scenario.
Regards,
Sara

Reply

Allen Mulvey
November 17, 2021 at 5:02 am

I found a way to make the client much more stable if the connection is broken and reconnected.

Make pClient global by moving the following statement to the top area outside all loops and functions:

```
BLEClient* pClient = BLEDevice::createClient();
```

Then add this to the top of the main loop:

```
bool status = pClient->isConnected(); // check BLE connection
if (!status) {
doConnect = true;
temperatureChar[0] = 0; humidityChar[0] = 0; // purge stale data
Serial.println("Reconnecting BLE…");
printReadings(); // redraw display so it does not display stale data
}
```

It will then connect and reconnect as necessary.

Reply

Sara Santos

November 17, 2021 at 3:21 pm

Thanks. I'll try it out and update the code.
Regards,
Sara

Reply

Chris

November 21, 2021 at 2:23 am

Thanks RNT crew,
… for doing the BLE thing for esp32… and everything else.!

I am trying to incorporate [AsyncElegantOTA]
and am confused as to the order of the [#includes].
— I am only guessing that there is a correct “pecking order” ! ?

Is there a location where this issue is documented?

Currently I have:
#include <BLEDevice.h>
#include <BLEServer.h>
#include <BLEUtils.h>
#include <BLE2902.h>
#include <Wire.h>

#include <Arduino.h>
#include “WiFi.h”
#include “AsyncTCP.h”
#include “ESPAsyncWebServer.h”
#include “AsyncElegantOTA.h”

#include <Adafruit_Sensor.h>
#include <Adafruit_BME280.h>

But haven’t even tested that yet.
…before I proceed only to find headaches that I can’t identify the cause of.

I would appreciate someone with more experience would mind checking that and advise whether OK. If not, please re-order.

Any other pitfalls of trying to do OTA with BLE.
My server is going outside to a hard-to-reach spot.

TIA

Reply

Sara Santos
November 21, 2021 at 6:38 pm

Hi.
Everything looks fine.
Just put the following at the beginning:
#include <Arduino.h>
Regards,
Sara

Reply

Roland
December 30, 2021 at 5:44 pm

Maybe this will be useful for someone.
Use NimBLEDevice.h instead BLEDevice.h for ESP32BLE !!!.
The BLEDevice.h eating too much memory, and if you will use wifi & BLE – the free memory will be dramatically low.
Found NimBLEDevice.h library,
Its use up to 44% less memory, compared to BLEDevice.h !!!

Reply

Jack

January 30, 2022 at 8:58 am

thank you for your information dude, have a nice day

Reply

Caqueux

January 8, 2022 at 3:48 am

Hello Sara!

I'm using your client example to get info from a pressure sensor. I don't why I can really connect to my sensor that act like a server (tested via the LightBlue application in Android)
My console output look like this:

Starting Arduino BLE Client application…
Device found. Connecting!
Setup done
[E][BLEClient.cpp:238] gattClientEventHandler(): Failed to connect, status=Unknown ESP_ERR error
– Connected to server

Any idea why?

Thanks if you have any idea 🙂

Reply

Mathéo

March 8, 2022 at 6:46 pm

If you want i can read your code and look that

Reply

Dexter

January 29, 2022 at 12:57 pm

Can i use multiple servers with single client. Is that possible

Reply

indika de silva

February 12, 2022 at 11:09 am

Dear Sara,

Only the temperature and humidity value updates shown on the serial monitor. No any other details related to communication between both ESP32 modules shown on the serial monitor(Both server and client are same). What can be the reason?

Thanks,
Indika

Reply

Mathéo
March 8, 2022 at 6:48 pm

If you want i can read your code and look that , did you pay attention to the specific address of the sensor in its datasheet?

Reply

Marendaz Serge
February 13, 2022 at 1:24 pm

ESP BLE client
Est il possible que le client se déconnecte du serveur après avoir reçu ses informations, pour laisser la place à un autre client
Merci pour votre réponse

Cordialement
Serge

Reply

Chris Maloney

March 10, 2022 at 12:50 am

Hi Sara, Rui,

I am wondering if I could persuade you wonderful folk to have a look into [NimBLE-Arduino] by h2zero.

The advantages of NimBLE-Arduino over the Standard model BLE are *huge* in terms of RAM usage.
— This is even acknowledged on on the http://www.arduino.cc site.

For my tired old brain, it's too much to get my head around and my project is now stretching the limits such that I am now using esp32-WROVER-IE.

some links:
– https://www.arduino.cc/reference/en/libraries/nimble-arduino/
– https://github.com/h2zero/NimBLE-Arduino
– https://h2zero.github.io/esp-nimble-cpp/md__migration_guide.html#autotoc_md46

If anyone else reading this is already into NimBLE and have a solid grasp of how to migrate, please drop a not in the RNT Labs. I could really use some help.

In BLE, I find it difficult to determine which are the library "keywords" and which are User defined.
— The third of the above links, I've only just found and that is helping a lot.

I feel sure you will be impressed by this library…
AND widely applauded for a tutorial and/or book on this subject. I would be first in line to buy. 🙂

TIA,
Chris

Reply

Sara Santos
March 10, 2022 at 12:15 pm

Hi.
Thank you so much for sharing that.
Other readers have also said great things about that library.
I haven't tried it yet. But, that's on my endless to-do list.
Thanks for your comment.
Regards,
Sara

Reply

Mathéo
March 14, 2022 at 11:05 pm

Hello i want use value temperature in my client to make a condition but i can't because it's a pointer do you have a solution or idea for make a condition with data receive ?

Reply

Sara Santos

March 15, 2022 at 12:19 am

Hi.
Check this that might help: https://stackoverflow.com/questions/13145777/c-char-to-int-conversion
Regards,
Sara

Reply

Alex

March 27, 2022 at 9:30 pm

Is it possible to connect more than one client to the same server?

Reply

Sara Santos

March 28, 2022 at 10:10 pm

Hi.
Not at the same time.
Regards,
Sara

Reply

Johnny

December 6, 2022 at 2:16 pm

Hello Sara
I want to use multipel Client to one Server ESP32, what should i do in your code? may be few suggestions to start with ?
thanks Johnny

Reply

suryateja

April 11, 2022 at 7:05 am

Guys can you tell is there a method where i can broadcast some text and sensor data using esp32 BLE

Reply

Ricky

June 21, 2022 at 11:07 am

Which part do i need to change and add if i want to use 2 type of sensors which is Soil Moisture sensor and DS18B20 Temperature sensor to display the soil moisture values, temperature and status if the soil is too wet or too dry on the client’s OLED

Reply

José Santos

June 23, 2022 at 5:02 pm

Dear Sara,
Excelent Tutorial!
I need to identify the device I am connected to.
Is there a way of reading the MAC address or the Manufactorer ID of the device that is connected to my ESP32 BLE?
Thank you and regards

Reply

Sara Santos

June 24, 2022 at 11:05 am

Hi.
I think this discussion might help: https://rntlab.com/question/ble-profile/
Regards,
Sara

Reply

José Santos

June 27, 2022 at 9:36 am

Thank you Sara.
I will take a look.
Obrigado e cumprimentos
José Miguel Santos

Reply

Greg
September 1, 2022 at 7:08 am

Great tutorial! Thanks.

I have experienced an issue that after a successful connect, disconnect I could not connect to the device again. I had to restart advertising after disconnect like so:

```
//Setup callbacks onConnect and onDisconnect
class MyServerCallbacks: public BLEServerCallbacks {
void onConnect(BLEServer* pServer) {
deviceConnected = true;
};
void onDisconnect(BLEServer* pServer) {
deviceConnected = false;
pServer->startAdvertising(); // restart advertising after disconnecting
}
};
```

Now it works like charm.
Link to where I have found this solution: https://github.com/espressif/arduino-esp32/issues/6016

Reply

Ken Blanch

December 5, 2022 at 6:46 pm

Hi I hope you can help me i get a
error: ‘temperatureNotifyCallback’ was not declared in this scope
message when compiling ESP32 BLE Client – Code

Reply

Sara Santos

December 9, 2022 at 5:29 pm

Hi.
Move the following function in the code so that it comes before the connectToServer() function.
//When the BLE Server sends a new temperature reading with the notify property
static void temperatureNotifyCallback(BLERemoteCharacteristic* pBLERemoteCharacteristic,
uint8_t* pData, size_t length, bool isNotify) {
//store temperature value
temperatureChar = (char*)pData;
newTemperature = true;
}

Reply

Leave a Comment

Name *

Email *

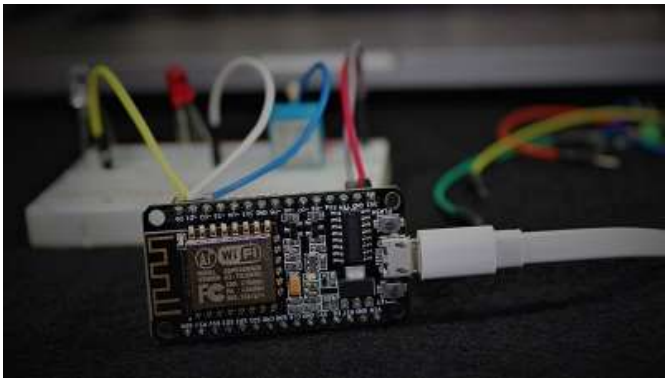
Website

☐ Notify me of follow-up comments by email.

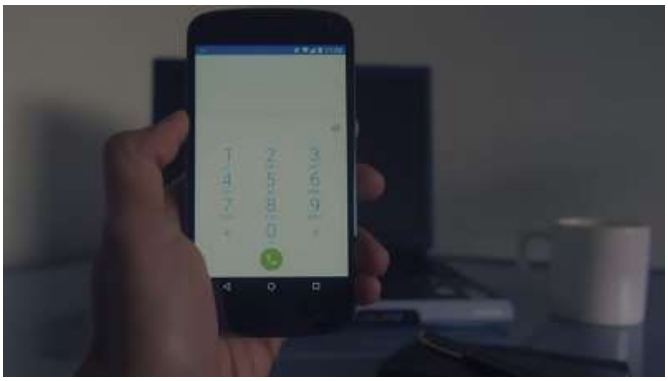
☐ Notify me of new posts by email.

Post Comment

Visit Maker Advisor – Tools and Gear for makers, hobbyists and DIYers »



Home Automation using ESP8266 eBook and video course » Build IoT and home automation projects.



Build Web Servers with ESP32 and ESP8266 » boards to control outputs and monitor sensors remotely.