**Name: ANISSH BABU**

**Email:** [**anisshbabuln@gmail.com**](mailto:anisshbabuln@gmail.com)

**Linkedin:** [**www.linkedin.com/in/anisshbabu**](http://www.linkedin.com/in/anisshbabu)

**Low Energy Bluetooth Service Broadcasting**

**Requirements:** Any IOT Development board like ESP32 or Arduino board with Bluetooth on it.

**Bluetooth Low Energy**, or BLE for short, is a version of Bluetooth that uses less power. Low bandwidth, short-distance data transfer is the main use case for BLE technology. BLE is always in sleep mode, as opposed to Bluetooth, which is always on, unless a connection is made.



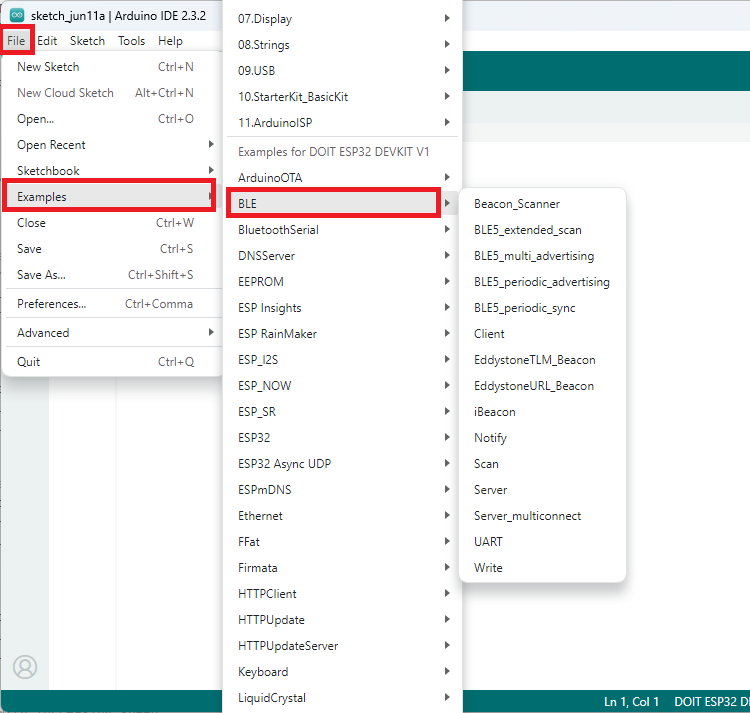
**BLE and ESP 32:** The ESP32 can operate as both a BLE client and server. The [ESP32 BLE library](https://github.com/nkolban/ESP32_BLE_Arduino) for the Arduino IDE contains several BLE examples for the ESP32. As soon as you install the ESP32 on the Arduino IDE, this library is automatically installed.

Here is a reference site on how to install ESP 32 on Arduino IDE(version 2.0 and above): [Installation link](https://randomnerdtutorials.com/installing-esp32-arduino-ide-2-0/)

In your Arduino IDE, you can go to **File** > **Examples** > **BLE** and explore the examples that come with the BLE library.

**Note:** to see the ESP32 examples, you must have the ESP32 board selected on **Tools** > **Board**.

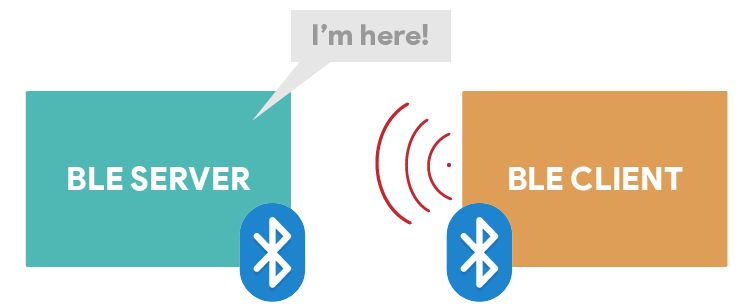
**Reference screenshot for BLE examples:**



**BLE Client and Server:**

There are two kinds of devices that work with Bluetooth Low Energy: the client and the server, which is also known as a peripheral. The ESP32 is capable of serving as a server or a client.

The server has data that the client can access and broadcasts its presence for other devices to find. After searching among the nearby devices, the client connects to the server it's looking for and begins listening for incoming data. We'll utilise the ESP32 in a communication method known as point-to-point communication.

****

**GATT:**

The abbreviation GATT represents Generic Attributes and outlines a structured hierarchy of data that is made available to connected BLE devices. This indicates that GATT specifies the manner in which two BLE devices exchange standardised messages. Understanding this hierarchy holds significance as it facilitates comprehension of BLE usage with the ESP32.

**GATT hierarchy consists of:**

- **Profile:** a standard set of services tailored for a specific purpose;

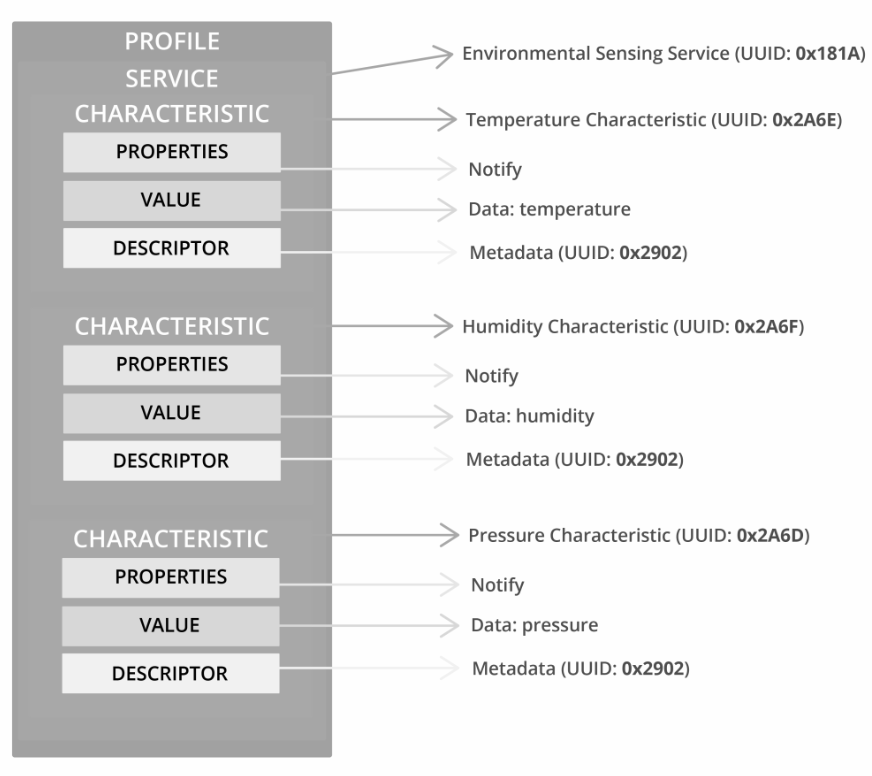
- **Service:** a compilation of correlated information such as sensor readings, battery level, heart rate, etc.;

- **Characteristic:** the specific location within the hierarchy where the actual data (value) is stored;

- **Descriptor:** additional information about the data;

- **Properties:** details about how the characteristic value can be interacted with, such as read, write, notify, broadcast, indicate, etc.

**According to this project:**

****

**\*NOTE: THESE UUID ARE ACCORDING TO DEFAULT UUID FOR ENVIRONMENTAL SENSING.**

[If you go to this page](https://www.bluetooth.com/specifications/assigned-numbers/) and open the [Assigned Numbers Document (PDF)](https://btprodspecificationrefs.blob.core.windows.net/assigned-numbers/Assigned%20Number%20Types/Assigned_Numbers.pdf), you’ll find all the default assigned UUID numbers. If you search for the *Environmental* Sensing Service, you’ll find all the permitted characteristics that you can use with that service. You can see that it supports temperature, humidity, and pressure.

**Services to broadcast:**

1. Temperature Measurement

2. Humidity

**Bluetooth Application:** nRF - Connect

**Service broadcast:** 00000002-0000-0000-FDFD-FDFDFDFDFDFD

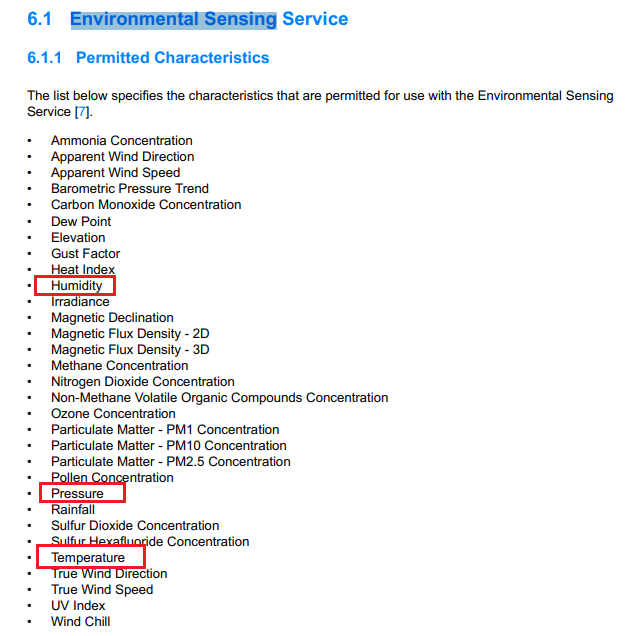
(According to our project)

**Characteristics**:

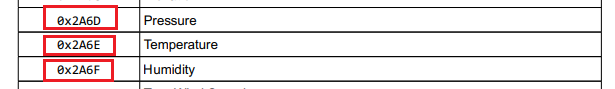
**Temperature:**0x2A6E

**Humidity:**0x2A6F

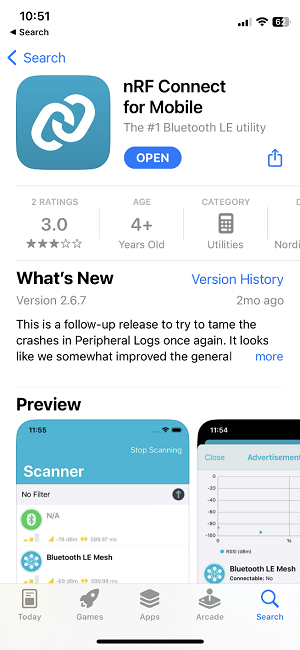
**Pressure:**0x2A6D

****

****

****

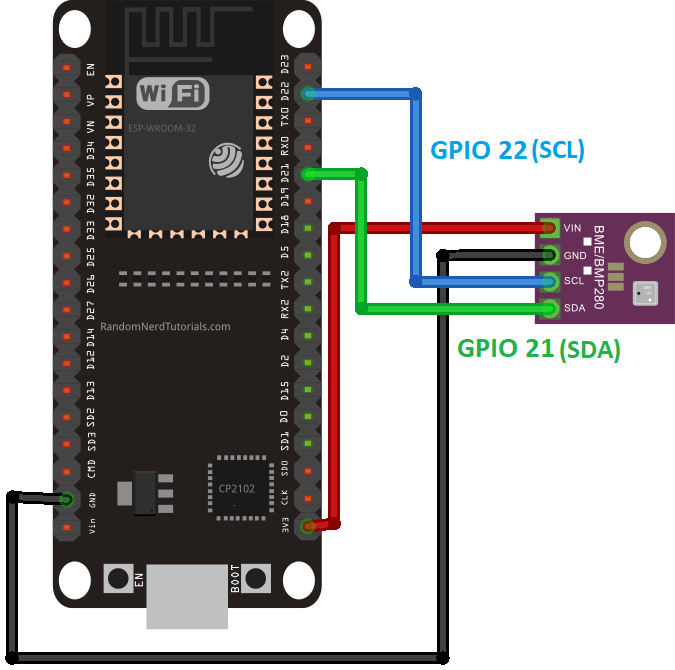
nRF Connect for Mobile from Nordic. It works on [Android (Google Play Store)](https://play.google.com/store/apps/details?id=no.nordicsemi.android.mcp) and [iOS (App Store)](https://itunes.apple.com/us/app/nrf-connect/id1054362403?mt=8). Go to Google Play Store or App Store, search for “nRF Connect for Mobile” and install the app.



**Components required:**

1. ESP32 Board
2. BME280 sensor module
3. Breadboard
4. Jumper wires

**Schematic diagram:**



**Steps for BLE:**

The steps to establish an ESP32 BLE peripheral with an Environmental Sensing BLE service that has characteristics related to pressure, temperature, and humidity are as follows:

* Assign a name to your BLE device (server); we'll call it ESP32\_BME2820, but you can use any other name.
* Establish a service for environmental sensing
* (UUID: 00000002-0000-0000-FDFD-FDFDFDFDFDFD).
* Give that service the following extra features:
* Temperature: 0x2A6E
* Pressure: 0x2A6D
* Humidity: 0x246F
* Give the attributes additional descriptors.
* Launch the BLE server.
* Upon establishing a connection with a client, the system will update the characteristics with new values and promptly notify the client of any changes.

**ESP 32 code:**

1. **Importing libraries:**

You start by importing the required libraries: the libraries to use BLE and the libraries to interface with the BME280 sensor.

#include <BLEDevice.h>

#include <BLEServer.h>

#include <BLEUtils.h>

#include <BLE2902.h>

#include <Adafruit\_Sensor.h>

#include <Adafruit\_BME280.h>

2. **BLE server name:**

In the following line, you can define the name of your BLE device. We’ll call it **ESP32\_BME2820**, but you can call it any other name.

//BLE server name

#define bleServerName "ESP32\_BME280"

**3. Bluetooth UUIDs:**

**#defineSERVICE\_UUID“00000002-0000-0000-FDFD-FDFDFDFDFDFD”**

BLECharacteristic

// Temperature Characteristic and Descriptor (default UUID)

temperatureCharacteristic(BLEUUID((uint16\_t)0x2A6E), BLECharacteristic::PROPERTY\_NOTIFY);

BLEDescriptor

temperatureDescriptor(BLEUUID((uint16\_t)0x2902))

// Humidity Characteristic and Descriptor (default UUID)

BLECharacteristic humidityCharacteristic(BLEUUID((uint16\_t)0x2A6F), BLECharacteristic::PROPERTY\_NOTIFY);

BLEDescriptor humidityDescriptor(BLEUUID((uint16\_t)0x2902));

// Pressure Characteristic and Descriptor (default UUID)

BLECharacteristic pressureCharacteristic(BLEUUID((uint16\_t)0x2A6D), BLECharacteristic::PROPERTY\_NOTIFY);

BLEDescriptor pressureDescriptor(BLEUUID((uint16\_t)0x2902));

**4. Initialize the BME280 Sensor:**

// Create a sensor object

Adafruit\_BME280 bme;

// Init BME280

void initBME(){

if (!bme.begin(0x76)) {

Serial.println("Could not find a valid BME280 sensor, check wiring!");

while (1);

}

}

**5. BLE Callback functions:**

//Setup callbacks onConnect and onDisconnect

class MyServerCallbacks: public BLEServerCallbacks {

void onConnect(BLEServer\* pServer) {

deviceConnected = true;

Serial.println("Device Connected");

};

void onDisconnect(BLEServer\* pServer) {

deviceConnected = false;

Serial.println("Device Disconnected");

}

};

bool deviceConnected = false;

void onConnect(BLEServer\* pServer) {

deviceConnected = true;

Serial.println("Device Connected");

};

void onDisconnect(BLEServer\* pServer) {

deviceConnected = false;

Serial.println("Device Disconnected");

}

Serial.begin(115200);

// Start BME sensor

initBME();

/ Create the BLE Device

BLEDevice::init(bleServerName);

// Create the BLE Server

BLEServer \*pServer = BLEDevice::createServer();

pServer->setCallbacks(new MyServerCallbacks());

// Create the BLE Service

BLEService \*bmeService = pServer->createService(SERVICE\_UUID);

// Create BLE Characteristics and corresponding Descriptors

bmeService->addCharacteristic(&temperatureCharacteristic);

temperatureCharacteristic.addDescriptor(&temperatureDescriptor);

bmeService->addCharacteristic(&humidityCharacteristic);

humidityCharacteristic.addDescriptor(&humidityDescriptor);

bmeService->addCharacteristic(&pressureCharacteristic);

pressureCharacteristic.addDescriptor(&pressureDescriptor);

// Start the service

bmeService->start();

// Start advertising

pServer->getAdvertising()->start();

if (deviceConnected) {

// Read temperature as Celsius

float t = bme.readTemperature();

// Read humidity

float h = bme.readHumidity();

// Read pressure

float p = bme.readPressure()/100.0F;

uint16\_t temperature = (uint16\_t)t;

//Set temperature Characteristic value and notify connected client

temperatureCharacteristic.setValue(temperature);

temperatureCharacteristic.notify();

Serial.print("Temperature Celsius: ");

Serial.print(t);

Serial.println(" ºC");

//Notify humidity reading

uint16\_t humidity = (uint16\_t)h;

//Set humidity Characteristic value and notify connected client

humidityCharacteristic.setValue(humidity);

humidityCharacteristic.notify();

Serial.print("Humidity: ");

Serial.print(h);

Serial.println(" %");

//Notify pressure reading

uint16\_t pressure = (uint16\_t)p;

//Set humidity Characteristic value and notify connected client

pressureCharacteristic.setValue(pressure);

pressureCharacteristic.notify();

Serial.print("Pressure: ");

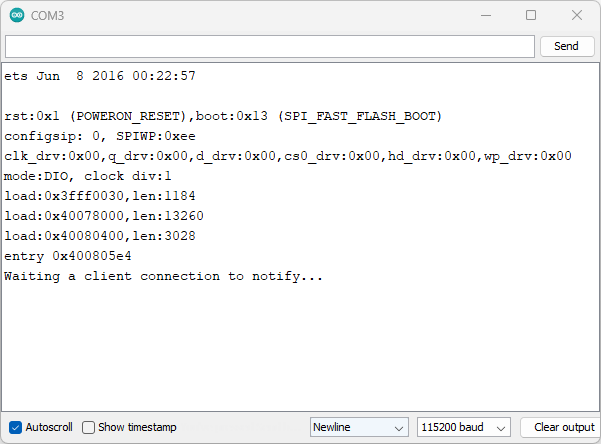
Serial.print(p);

Serial.println(" hPa");

delay(10000);

**Working:**

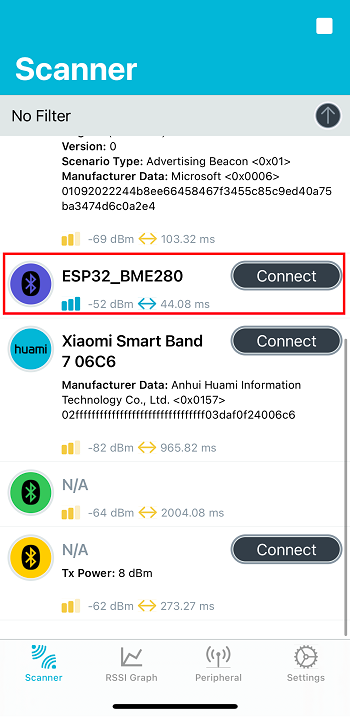
**NOTE: Upload the code to your board. After uploading, open the Serial Monitor, and restart the ESP32 by pressing the RST/EN button. You should get a similar message in the Serial Monitor.**

****

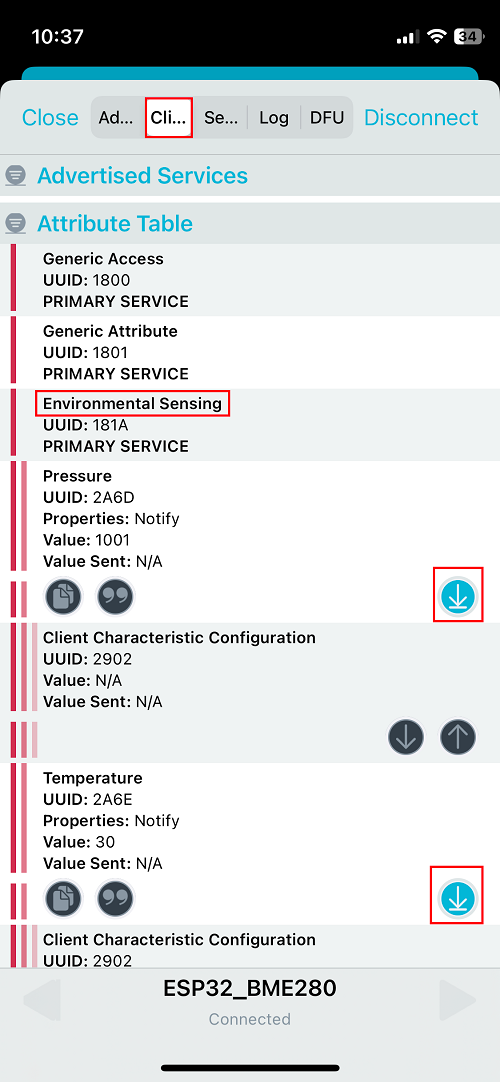
This means everything is working as expected and the ESP32 is waiting for a BLE client to connect.

Then, go to your smartphone, open the nRF Connect app from Nordic, and start scanning for new devices.

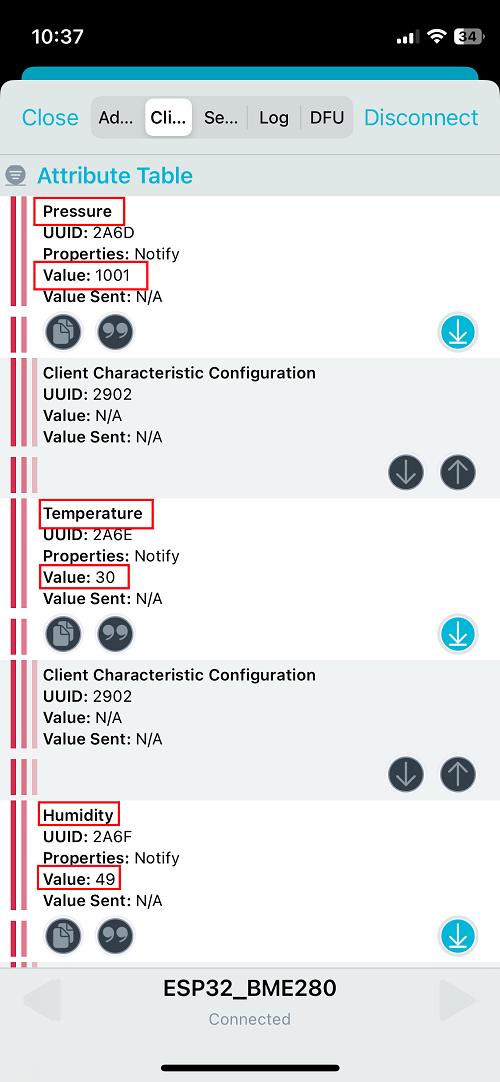
You should find a device called ESP32\_BME280—this is the BLE server name you defined earlier.



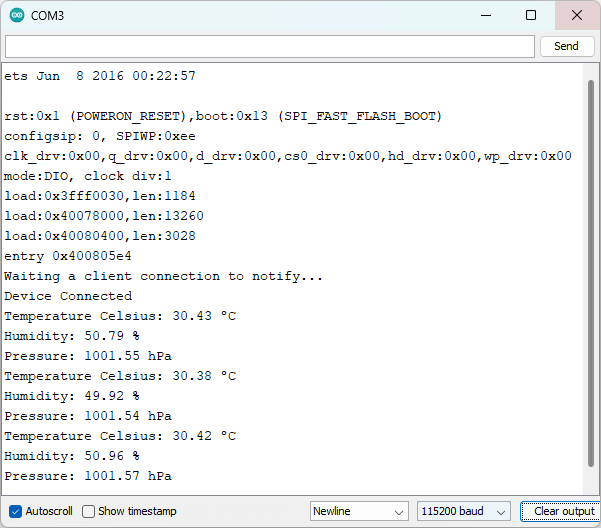
Connect to it. You’ll see that it displays the Environmental Sensing service with the temperature, humidity, and pressure characteristics. Click on the arrows to activate the notifications.



Then, click on the second icon at the left to change the format. You can change to unsigned int for all characteristics. You’ll start seeing the temperature, humidity, and pressure values being reported every 10 seconds.



**Serial monitor readings:**



Now we have successfully created an ESP32 BLE Peripheral that advertises the Environmental Sensing Service. Now, you can develop an app, or program another ESP32 to interface with the ESP32 BLE device.