

# BLE Research

---

*By: Johnson Domacasse (#4471709)*

*23 March 2024*

## Contents

1. Introduction: .....	3
2. Methods: .....	4
2.1 Scanner: .....	4
2.2 Server: .....	4
2.3 Environmental Service: .....	4
3. Results: .....	5
3.1 Scanner results: .....	5
3.2 Server results: .....	5
3.3 Environmental Service results: .....	7
3. Discussion: .....	7
4. Bibliography: .....	8

## 1. Introduction:

This research is focused on experimenting with short distance protocol. This specific research revolves around Bluetooth low energy (BLE). One of the reasons we use BLE over standard Bluetooth has to do with power consumption because of the protocol being on continuously. This problem was solved in BLE by having the protocol constantly in sleep mode, unless a connection is initiated. With this research we will dive a bit deeper into the protocol.

Some important terms such as UUID, service, characteristics etc. will be discussed as this research goes on. These all form the Generic Attributes (GATT) which is exposed to connected BLE devices. See figure 1 for a standard GATT hierarchy. Figure 2 shows a bit more regarding the details of each of these terms. Here we can see the **profile** of the device labelled as the "Bluetooth device". The **service** that this device provides is the battery service. The **characteristic** of this service is the battery level. Each of these (except for the **profile**) have a predefined UUID. It is used for uniquely identifying information. For example, the service that is provided by a specific device[1].

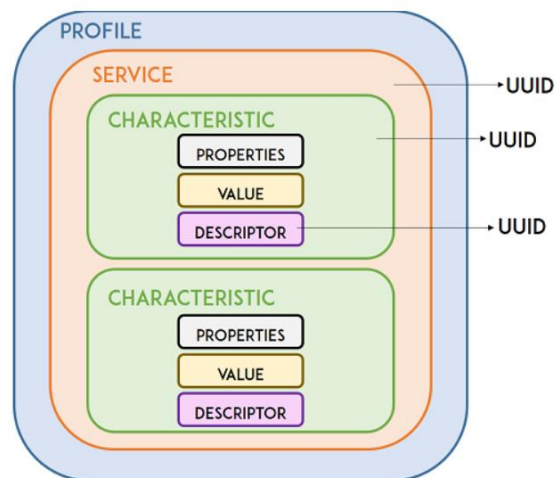


Figure 1: BLE standard outline.

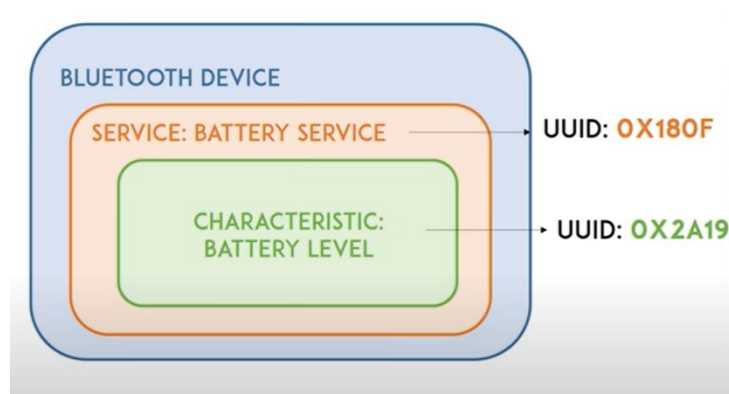


Figure 2: BLE example.

## 2. Methods:

This research was performed by using tools that are available to us. These include the following:

- 1x ESP32-C3-DevKitM-1 board
- 1x DHT22 sensor (with the DHTESP library for proper control)
- Assigned numbers pdf document
- 1x Android smart phone with the NRF connect app
- 1x IOS smart phone with the NRF connect app

### 2.1 Scanner:

We begin by setting up the environment we will be working with. Then the Arduino BLE library is added to the project so that we can work with the technology. Then we use the already made example and add it into our main file.

Finally we take a look at the serial monitor and there we can already see some devices in the area that are running on BLE. See figure 3 for the results for these devices.

Here we can see the addresses of the different devices, some of them have local names assigned to them. Others have service UUIDs. What they all have in common is that each one has an RSSI value. Which is simply a measure that represents the relative quality level of the Bluetooth signal received on the device. The signal is better when this value is closer to zero. So a pretty good value is -50 or below. A reasonable value is between -70 and -80 while -100 indicates no signal at all [2].

Another fun detail we can notice is that using the assigned numbers document, we can determine that the service the last two devices provide is from Google LLC (0XFE9F)[3].

### 2.2 Server:

The second step is to get acquainted with setting up your own personal BLE server. This is done by using the Arduino library: "ArduinoBLE". Once this was integrated in the project, the server example can be used. The local name of the device can be changed. Here use my full name as an example: "Johnson Domacasse".

From there, we set up a value that we are able to see if we were to connect on a different device. Again this was kept simple with the message: "BLE Research". We confirm if all of this working by using our smart phone. We check the Serial monitor before hand to see if everything works correctly. See figure 4. The results can be checked to see if we can read the correct value from our smart phone. See figure 5 and figure 6.

### 2.3 Environmental Service:

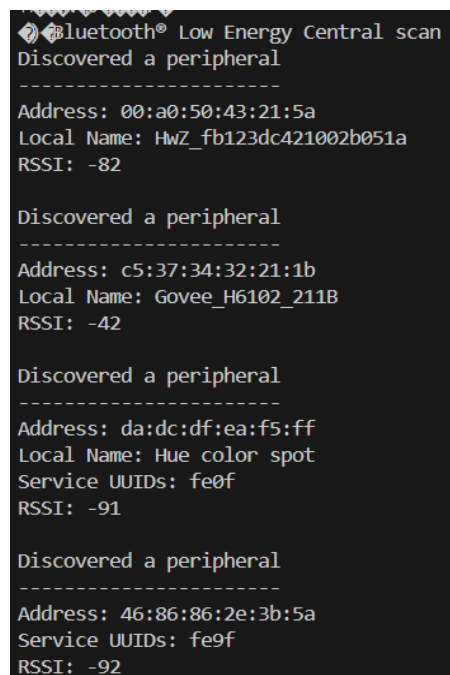
This section gives us a chance to go a bit deeper into this BLE research. One part of this is to use the services that are already provided by the assigned number document. In the previous examples, we had the option to choose from predefined UUIDs for both the service and the characteristics. An environmental service already exists with its own given UUID according to the assigned numbers document. Under section 3.4.2 Services by UUID. The UUID of this service is 0x181A. Additionally if you traverse to section 6.1 you can see a list of the permitted characteristics that are allowed on this service. Here we can find both temperature and humidity. We can then traverse to section 3.8.1 to find the temperature and humidity and find their UUIDs. They are 0x2A6E and 0x2A6F.

Like the server implementation, we create our server and service. We use the environmental service here. We then make a separate characteristic for both humidity and temperature. Each of the will have the notify property rather than the standard read/write property. This is so that the devices only get notified when the value changes. From there in our loop, we read the temperature/Humidity. Cast this into an unsigned 16 integer. And then from there set it as the value that needs to be sent to our devices[3][4].

### 3. Results:

The following was found when performing this research. The results of each section will be categorized here in the form of figures and/or extra descriptions.

#### 3.1 Scanner results:



```
Bluetooth® Low Energy Central scan
Discovered a peripheral
-----
Address: 00:a0:50:43:21:5a
Local Name: HwZ_fb123dc421002b051a
RSSI: -82

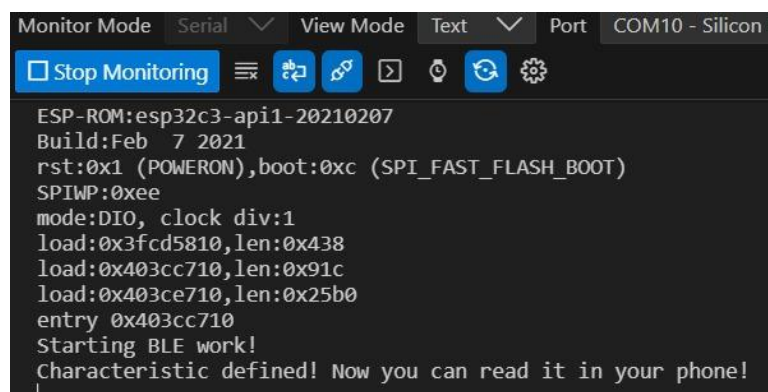
Discovered a peripheral
-----
Address: c5:37:34:32:21:1b
Local Name: Govee_H6102_211B
RSSI: -42

Discovered a peripheral
-----
Address: da:dc:df:ea:f5:ff
Local Name: Hue color spot
Service UUIDs: fe0f
RSSI: -91

Discovered a peripheral
-----
Address: 46:86:86:2e:3b:5a
Service UUIDs: fe9f
RSSI: -92
```

**Figure 3:** List of Bluetooth low energy devices.

#### 3.2 Server results:



```
Monitor Mode Serial View Mode Text Port COM10 - Silicon L
[ ] Stop Monitoring
ESP-ROM:esp32c3-api1-20210207
Build:Feb 7 2021
rst:0x1 (POWERON),boot:0xc (SPI_FAST_FLASH_BOOT)
SPIWP:0xee
mode:DIO, clock div:1
load:0x3fcd5810,len:0x438
load:0x403cc710,len:0x91c
load:0x403ce710,len:0x25b0
entry 0x403cc710
Starting BLE work!
Characteristic defined! Now you can read it in your phone!
```

**Figure 4:** Server is up.

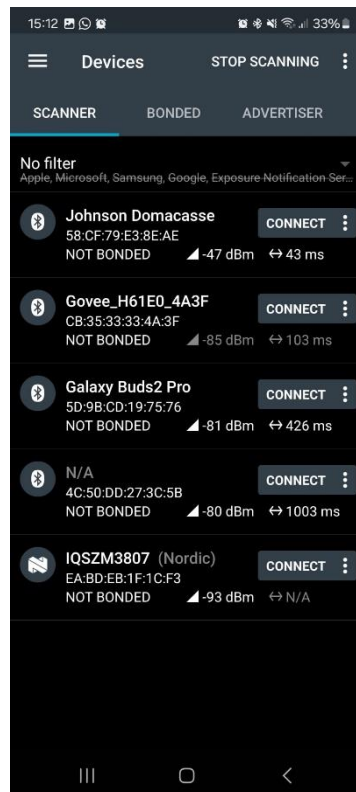


Figure 5: List of devices to be found (android devices)

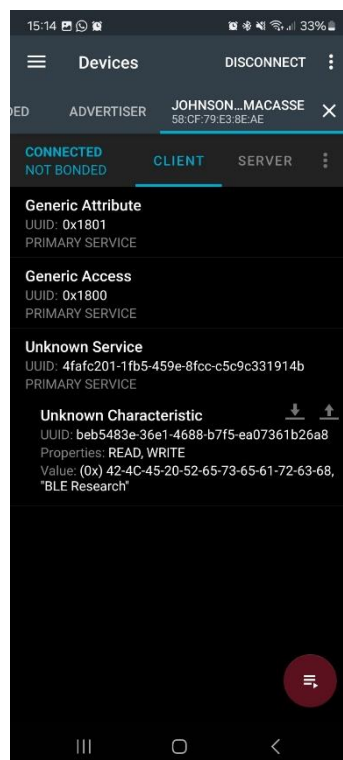
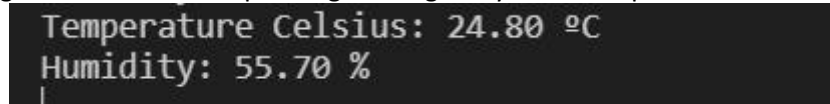


Figure 6: BLE server and characteristic value. (android devices)

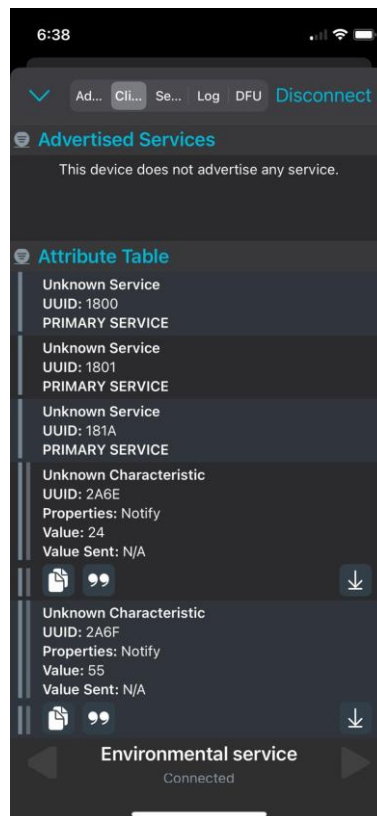
### 3.3 Environmental Service results:

After connecting our device to our BLE server, we can start seeing the readings within the serial monitor. These same values can be found as values on your smart phone. See figure 7 for the readings and figure 8 for the corresponding readings on your smart phone.



```
Temperature Celsius: 24.80 °C
Humidity: 55.70 %
```

**Figure 7:** Serial monitor readings.



**Figure 8:** Corresponding readings on app (IOS devices)

### 3. Discussion:

This challenge offered me some good insight on what BLE is and how it can be used to send data to another device while also conserving power. This research provided me with not only some skill as to how the technology works, but also additional knowledge about the protocol in itself.

Naturally while working, I ran into some problems. The first being the ESP32 model that we use. Sadly I didn't get a screenshot of this before I fixed the issue. The issue was that the code was uploading correctly on the board, but when I take a look at the serial output, it is nothing from which I am expecting. They are values that look similar to registers however nothing that looks like a UUID. Another issue was that in the beginning when setting up my environment, I was unable to upload the code to the microcontroller board because there were certain functions that were not being recognized. I tried solving these problems by instead using the Arduino IDE but this gave me the same results. After closing a re-opening the projects, they worked fine. I asked a classmate and she said that she uses the WROOM model because that proved to work. I made this implementation work with the Devkit-C3-M1 model.

Finally, the provided environment service reference provided is somewhat outdated as well. With this implementation, the code was not compiling at all. It had to with the way I was defining my characteristics and some memory issues. To fix this issue, these characteristics were defined similarly to the server implementation. Additionally, this implementation did not seem to work with my android device (I was not connecting to the serial monitor despite it saying connected on my device). I used an IOS device and this was able to connect, start the serial monitor and begin transmitting immediately.

#### 4. Bibliography:

- [1] - Santos, R. (2019, June 4). *Esp32 Bluetooth Low Energy (BLE) on Arduino Ide*. Random Nerd Tutorials. <https://randomnerdtutorials.com/esp32-bluetooth-low-energy-ble-arduino-ide/>
- [2] - Li, M. (2023, March 16). *Understanding the measures of bluetooth RSSI*. MOKOBlue. <https://www.mokoblue.com/measures-of-bluetooth-rssi/>
- [3] - Assigned numbers | bluetooth SIG. (n.d.-a). [https://www.bluetooth.com/wp-content/uploads/Files/Specification/HTML/Assigned\\_Numbers/out/en/Assigned\\_Numbers.pdf?v=1705536000119](https://www.bluetooth.com/wp-content/uploads/Files/Specification/HTML/Assigned_Numbers/out/en/Assigned_Numbers.pdf?v=1705536000119)
- [4] - Santos, S. (2023, October 29). *ESP32 Ble Peripheral (server): Environmental Sensing Service*. Random Nerd Tutorials. <https://randomnerdtutorials.com/esp32-ble-server-environmental-sensing-service/>