**Bluetooth Low Energy (BLE)**



*By: Johnson Domacasse (#4471709)*

*13 March 2024*

Contents

[1. Introduction: 3](#_Toc161329663)

[2. BLE Scanner: 3](#_Toc161329664)

[3. BLE Server: 4](#_Toc161329665)

[4. Conclusion: 5](#_Toc161329666)

[5. Environmental implementation: 5](#_Toc161329667)

[6. Bibliography: 5](#_Toc161329668)

# Introduction:

Bluetooth is a wireless technology standard for the exchanging data over short distances. Just like Wi-Fi, Bluetooth also works on 2.4 GHz. It is used in many different application that require wireless communication and control. Like transmitting audio to headphones or a car. Devices use Bluetooth in a point-to-point communication to **continuously** transmit data. This is not optimal for power usage. Bluetooth low energy (BLE) is a power-conserving alternative to this. Its primary application is short distance transmission of small amounts of data. Unlike Bluetooth that is always on, BLE is always in sleep mode constantly, unless a connection is initiated. This conserves power. This technology is extremely useful in machine to machine communication.

The aim of this challenge is to give us a better understanding of how this technology works.

Some important terminology will be discussed, such as UUID, characteristics services etc**[1].**

A diagram of a battery level

Description automatically generatedAn example of these can be seen in the figure on the right. 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].**

# BLE 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 example below.

A screenshot of a computer program

Description automatically generated

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].**

**Note: I tried uploading the .ino file to the board. Although it did compile and upload, the serial print didn’t seem to work, even after adding the Arduino header file.**

# BLE Server:

The second/last step of this challenge is to get acquainted with setting up your own BLE server. This is done by using the Arduino library: “ArduinoBLE”. Once this was integrated in the project, the server example can be used. Again we copy and paste the Arduino file into our main file. From there we set up a few things. First the local name of our BLE device.  It was kept simple by just using my full name.

From there, we set up a value that we are able to see if we were to connect on a different device. Again kept simple with the message: “BLE Research”. 

The next step is to confirm that this all works. After the code has been uploaded we confirm this by checking if the BLE device is ready to be scanned by other devices. A screenshot of a computer

Description automatically generatedafter this is confirmed we download the program on our phones, set it up correctly so that we can scan devices around us and from there see if we can detect our devices.

A screenshot of a phone

Description automatically generated

A screenshot of a phone

Description automatically generated

From the two images on the left we can see that after scanning for devices, we can see the device that we configured. From there once we connect to it and click on the service that it provided, we can see the value that was also set up as well. Additionally, there are two small details that can also be seen from the figure on the right. We can see both the UUIDs of the service and the characteristics. This was also predefined in the code**[1].**

# Conclusion:

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.

# Environmental implementation:

Finally, If I were to implement the environmental sensor in the optional section I would take the following approach. Both the humidity and temperature would have a separate characteristic and not a service. From there instead of uploading a hardcoded string, I would take the values from the sensor in the while loop. From there, use the “c\_str” function to convert it to a string so that it can be published.

# 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>