Bluetooth Low Energy (BLE) for Arduino

What is Bluetooth?

(Taken directly from https://www.bluetooth.com/)

Bluetooth is a short-range wireless technology standard used for exchanging data between fixed and mobile devices over short distances using UHF radio waves in the ISM bands, from 2.402 GHz to 2.480 GHz, and building personal area networks (PANs). It was originally conceived as a wireless alternative to RS-232 data cables. Bluetooth is managed by the Bluetooth Special Interest Group (SIG), which has more than 35,000 member companies in the areas of telecommunication, computing, networking, and consumer electronics. The IEEE standardized Bluetooth as IEEE 802.15.1, but no longer maintains the standard. The Bluetooth SIG oversees development of the specification, manages the qualification program, and protects the trademarks. Bluetooth comes in two different radio versions: Bluetooth Low Energy and Bluetooth Classic and has 3 key characteristics: Range, Reliability, and Security.

Bluetooth Low Energy

The Bluetooth Low Energy (LE) radio is designed for very low power operation. To enable reliable operation in the 2.4 GHz frequency band, it leverages a robust frequency-hopping spread spectrum approach that transmits data over 40 channels. The Bluetooth LE radio provides developers a tremendous amount of flexibility, including multiple PHY (physical layer) options that support data rates from 125 Kb/s to 2 Mb/s, multiple power levels, from 1mW to 100 mW, as well as multiple security options up to government grade. Bluetooth LE also supports multiple network topologies, including point-to-point, broadcast and mesh networking.

Bluetooth Classic

The Bluetooth Classic radio, also referred to as Bluetooth Basic Rate/Enhanced Data Rate (BR/EDR), is designed for low power operation and also leverages a robust Adaptive Frequency Hopping approach, transmitting data over 79 channels. The Bluetooth BR/EDR radio includes multiple PHY options that support data rates from 1 Mb/s to 3 Mb/s, and supports multiple power levels, from 1mW to 100 mW, multiple security options, and a point-to-point network topology.

Range

Bluetooth's range can vary depending on the application and circumstance. The effective, reliable range between Bluetooth devices is anywhere from more than a kilometer down to less than a meter. Several key factors influence the effective range of a reliable Bluetooth connection, including the following: Radio Spectrum, Physical Layer, Receiver Sensitivity, Transmit Power, Antenna Gain and Path Loss. If you would like to learn more about these factors visit Understanding Bluetooth Range | Bluetooth® Technology Website.

Reliability

Unlike wired data communications technologies, wireless technologies must share the transmission medium. For example, Bluetooth technology operates in the same 2.4 GHz ISM (Industrial, Scientific, and Medical) radio frequency band as Wi-Fi and technologies that utilize the IEEE 802.15.4 standard. As a result, it's possible for a packet being transmitted between two Bluetooth devices to be corrupted or lost if it collides with a packet being transmitted at the exact same time and frequency channel as other in-range Bluetooth, Wi-Fi, or 802.15.4 devices. Bluetooth technology has adopted several techniques to lower the probability of collisions and offset inevitable packet loss. These techniques include: small, fast packets, adaptive frequency hopping, acknowledgements, and automatic retransmission. If you would like to learn more about Bluetooth Reliability visit Reliability | Bluetooth® Technology Website.

Security

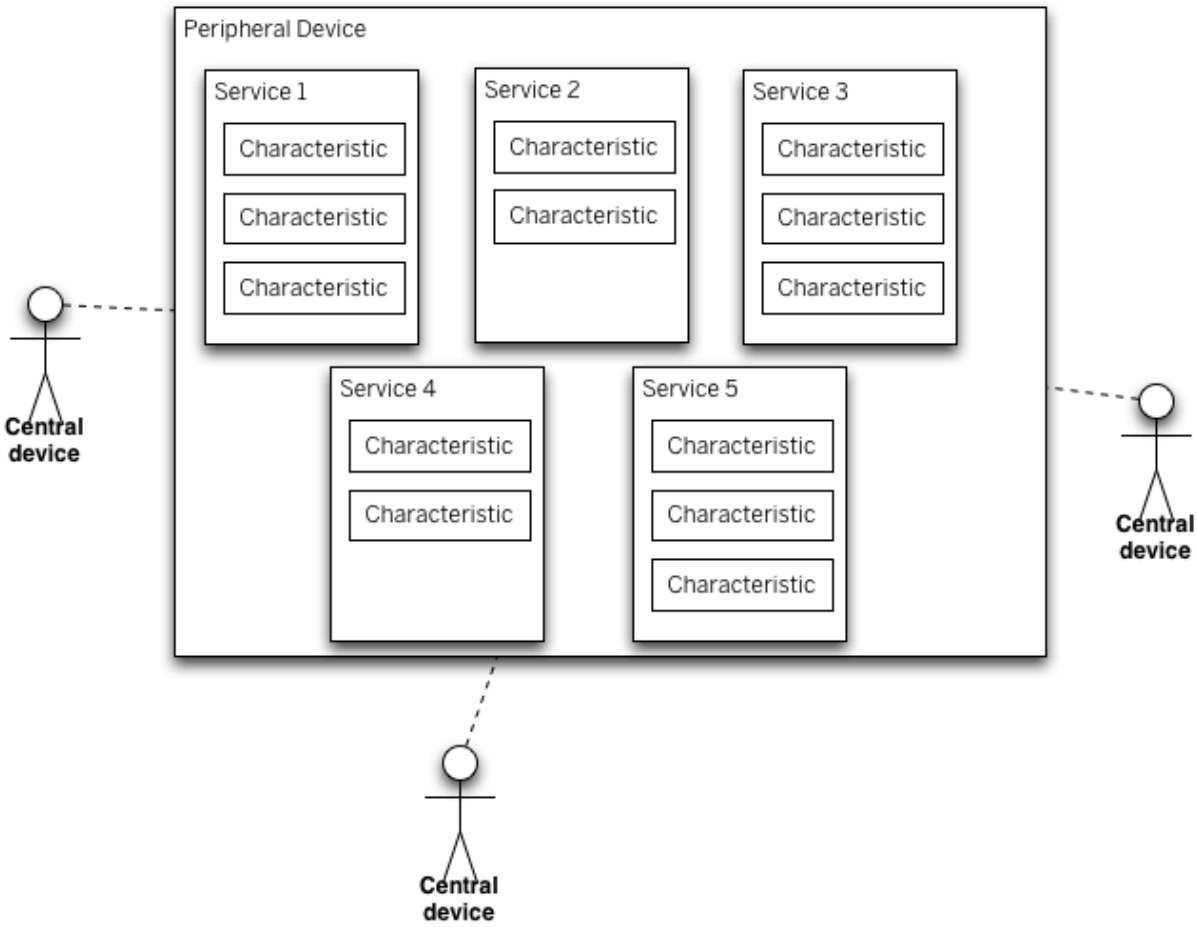
Bluetooth® specifications include a collection of features that provide developers the tools they need to secure communications between Bluetooth devices and implement the appropriate level of security for their products. All Bluetooth specifications are subject to security reviews during the development process. In addition, Bluetooth technology is an open, global standard, and the Bluetooth SIG encourages active review of the specifications by the security research community. If you would like to learn more about Bluetooth Security visit Bluetooth Security | Bluetooth® Technology Website.

Specifications

Bluetooth® specifications define the technology building blocks that developers use to create the interoperable devices that make up the thriving Bluetooth ecosystem. Bluetooth specifications are overseen by the Bluetooth Special Interest Group (SIG) and are regularly updated and enhanced by Bluetooth SIG Working Groups to meet evolving technology and market needs. If you would like to learn more about Bluetooth Specifications visit Specifications | Bluetooth® Technology Website.

(*Stop taking directly from https://www.bluetooth.com/*)

General Schematic for BLE



(taken from https://www.arduino.cc/en/Reference/ArduinoBLE)

Introduction to BLE

(*taking directly from* https://www.arduino.cc/en/Reference/ArduinoBLE)

Unlike standard bluetooth communication basically based on an universal asynchronous serial receiver-transmitter (UART) a Bluetooth LE radio acts like a community bulletin board. The computers that connect to it are like community members that read the bulletin board. Each radio acts as either the bulletin board or the reader. If your radio is a bulletin board (called a peripheral device in Bluetooth LE parlance) it posts data for all radios in the community to read. If your radio is a reader (called a central device in Bluetooth LE terms) it reads from any of the bulletin boards (peripheral devices) that have information about which it cares. You can also think of peripheral devices as the servers in a client-server transaction, because they contain

the information that reader radios ask for. Similarly, central devices are the clients of the Bluetooth LE world because they read information available from the peripherals.
Think of a Bluetooth LE peripheral device as a bulletin board and central devices as viewers of the board. Central devices view the services, get the data, then move on. Each transaction is quick (a few milliseconds), so multiple central devices can get data from one peripheral.

The information presented by a peripheral is structured as services, each of which is subdivided into characteristics. You can think of services as the notices on a bulletin board, and characteristics as the individual paragraphs of those notices. If you're a peripheral device, you just update each service characteristic when it needs updating and don't worry about whether the central devices read them or not. If you're a central device, you connect to the peripheral then read the boxes you want. If a given characteristic is readable and writable, then the peripheral and central can both change it.

UUIDs, GAP, and GATT

Services are identified by unique numbers known as UUIDs (universal unique identifiers). Standard services have a 16-bit UUID and custom services have a 128-bit UUID. The ability to define services and characteristics depends on the radio you're using and its firmware. To learn more about UUIDs visit Assigned Numbers | Bluetooth® Technology Website.

BLE devices let other devices know that they exist by advertising using the General Advertising Profile (GAP). Advertising packets can contain a device name, some other information, and also a list of the services it provides. Advertising packets have a limited size. You will only be able to fit a single 128-bit service UUID in the packet. To learn more about GAP visit Intro to Bluetooth Generic Access Profile (GAP) | Bluetooth® Technology Website

The Bluetooth LE protocol operates on multiple layers. General Attribute Profile (GATT) is the layer that defines services and characteristics and enables read/write/notify/indicate operations on them. When reading more about GATT, you may encounter GATT concepts of a "server" and "client". These don't always correspond to central and peripherals. In most cases, though, the peripheral is the GATT server (since it provides the services and characteristics), while the central is the GATT client. To learn more about GATT visit Intro to Bluetooth Generic Attribute Profile (GATT) | Bluetooth® Technology Website

Service design

A characteristic value can be up to 512 bytes long. This is a key constraint in designing services. Given this limit, you should consider how best to store data about your sensors and actuators most effectively for your application. The simplest design pattern is to store one sensor or actuator value per characteristic, in ASCII encoded values.

Characteristic	Value
Accelerometer X	200
Accelerometer Y	134
Accelerometer Z	150

This is also the most expensive in memory terms, and would take the longest to read. But it's the simplest for development and debugging.

You could also combine readings into a single characteristic, when a given sensor or actuator has multiple values associated with it.

Characteristic	Value
Motor Speed, Direction	150,1
Accelerometer X, Y, Z	200,133,150

This is more efficient, but you need to be careful not to exceed the 512-byte limit. The accelerometer characteristic above, for example, takes 11 bytes as a ASCII-encoded string.

Read/Write/Notify/Indicate

There are 4 things a central device can do with a characteristic:

Read	Ask the peripheral to send back the current value of the characteristic. Often used for characteristics that don't change very often, for example characteristics used for configuration, version numbers, etc.
Write	Modify the value of the characteristic. Often used for things that are like commands, for example telling the peripheral to turn a motor on or off.
Notify and Indicate	Ask the peripheral to continuously send updated values of the characteristic, without the central having to constantly ask for it.

(Stop taking directly from https://www.arduino.cc/en/Reference/ArduinoBLE)

Examples with Arduino MKR Wifi 1010

Before completing any of the examples below make sure to do/note the following:

- Complete Getting started with the MKR WiFi 1010
- Install the WifiNINA library
    - **Sketch > Include Library > Manage Libraries...**
    - Search WifiNINA
    - Select Install
    - Reload the IDE
- Complete WiFiNINA Firmware Updater | Arduino
    - **You have to google “WiFiNINA Firmware Updater”**

- Install the ArduinoBLE library
    - **Sketch > Include Library > Manage Libraries...**
    - Search ArduinoBLE
    - Select Install
    - Reload the IDE
- All examples will be using LightBlue as the smartphone app to send/receive data
    - Download LightBlue [iOS] [android]
- All examples will be using MATLAB as the programming language to send/receive data on a BLE-enabled computer.
- Generating a random UUID

Troubleshooting for all examples

- Sometimes you could get a message that says, failed to connect, in this case try to reconnect right away. If this does not work close the Serial Monitor, reset the arduino, and reset the application before trying again.
- Make sure your smartphone/computer has bluetooth turned on
- Make sure you have opened the Serial Monitor
- Make sure you have updated to the latest firmware for the board
- Make sure you gave the Arduino a unique name so that it is not confused with other BLE devices in the area
- For MATLAB, if you cannot connect to the device try typing **blelist** in the command Window to see if your device is listed
    - If you are running macOS sometimes the Device Name will be advertised, in this case you should change the **localName** variable to be equal to: **"MKR1010_" + pennKey;**
    - **NOTE: M1 Macs have an issue with the BLE library in matlab and my not be able to be used as a Central device (noticed of 3/2/2022)**

Writing Data to Arduino to Control an LED

Overview

In this example you will be using your smartphone (iOS or Android) and then a BLE enabled computer [These are acting as the **central device**] to control the status of the built-in LED on the Arduino MKR WiFi 1010 [this is acting as the **peripheral devices**].

Equipment needed:

- Arduino MKR WiFi 1010
- Micro USB cable
- Smartphone with a generic BLE central app (we demonstrate with LightBlue) or BLE enabled computer with coding software (we will demonstrate with MATLAB)

Arduino Setup (Peripheral)

In order to access the code navigate to **File > Examples > ArduinoBLE > Peripheral > LED**. Once there note the line: **BLE.setLocalName("LED");** If you are in a space where other people are using the same script, you will need to edit this line to something like: **BLE.setLocalName("LED_pennKey");** where pennKey is your penn key [or any other short unique identifier] (this will be the name you are looking for in the BLE app). You need to have a unique name so that your smartphone/computer can distinguish your Arduino from others in the area. You will **not** need to adjust the *ledService* **nor** the *switchCharacteristic* UUIDs as these are used to define the data structure being passed over BLE.

***NOTE:* If you are using an iOS device, your BLE application might show Arduino** instead of LED as the peripheral name. If this is the case you should add: **BLE.setDeviceName("MKR1010_pennkey");** where pennKey is your penn key [or any other short unique identifier] above **BLE.setLocalName("LED_pennKey");** to the arduino script. Now you will look for **MKR1010_pennkey** in the BLE application.

Upload the code to the MKR WiFi 1010 and then open the Serial Monitor. This should print out "*BLE LED Peripheral*" if all is working properly.

NOTE: You will need to have the Serial Monitor open to establish a BLE connection with this script because of the line: *while (!Serial);* (this is not needed in all use cases so you will need to determine if it is necessary in your final script)

Smartphone Setup (Central)

Now we will use the LightBlue Application to turn on and off the built-in LED on the Arduino. Follow along with the steps below on how to navigate through the application. LightBlue is a simple and easy "sanity-check" for your bluetooth application to make sure it can actually pass along information. If you are using a different application the steps should still be similar to what is outlined below.

1. Connect your Arduino to a computer and open the Serial Monitor (wait for it to display BLE LED Peripheral)
2. Connect to **LED (or LED_pennkey)** from the list of devices (you may need to scroll down or refresh the list) *[if you are on iOS this could be **MKR1010_pennkey**]*
    a. *We have noticed that some iOS devices are keeping the name as Arduino*
3. Select the UUID that matches the switchCharacteristic UUID set in the Arduino Code
4. Select the input box to write values
    a. Writing 1 will turn on the LED
    b. Writing 0 will turn off the LED

You can follow along with LightBlue on Android and with iOS.

Computer Setup (Central)

If you were able to successfully control the built-in LED with a smartphone then the next step will be establishing control through a BLE enabled computer. We will now walk through how to control the Arduino with MATLAB (you are not limited to using MATLAB in your projects but the code provided will give you a framework for other programming languages).

Follow the link to download the MATLAB file (BLE_matlab_led_control.m) needed for LED control and then follow the steps below to control the LED:

1. Connect your Arduino to a computer and open the Serial Monitor (wait for it to display "BLE LED Peripheral")
2. Edit the **localName** variable in the MATLAB code so that it matches the one defined in the Arduino script
3. Run the **Establish BLE connection with Arduino** section of the MATLAB code. Wait for the Command Window to display "**Connected…**"
4. To turn on the LED run the **Write 1 to the switchCharacteristic to turn on the LED** section of the MATLAB code
5. To turn off the LED run the **Write 0 to the switchCharacteristic to turn off the LED** section of the MATLAB code.

Analog Read of Single Sensor Data from Arduino

Overview

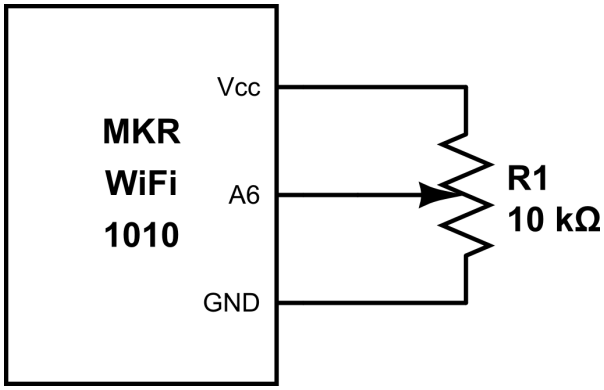
In this example you will be using your smartphone (iOS or Android) and then a BLE enabled computer [These are acting as the **central device**] to read data from a potentiometer (pot) connected to an Analog Pin on the Arduino MKR WiFi 1010 [this is acting as the **peripheral devices**]. The pot is a stand-in for whatever sensor you need for your project.

Equipment needed:

- Arduino MKR WiFi 1010
- Micro USB cable
- Smartphone with a generic BLE central app (we demonstrate with LightBlue) or BLE enabled computer with coding software (we will demonstrate with MATLAB)
- Breadboard
- Jumper wire
- Potentiometer

Circuit

Note: Vcc on the MKR WiFi 1010 is 3.3V



Arduino Setup (Peripheral)

Follow the link to download the necessary Arduino file (BLE_arduino_analog_read.ino). Once downloaded you will need to edit the **pennKey** variable to your penn key [or any other short unique identifier]. The local name (which is usually what gets advertised) will be **AnalogRead_pennKey**. The device name (which can be advertised on iOS devices) will be **MKR1010_pennKey**. You need to have unique names so that your smartphone/computer can distinguish your Arduino from others in the area.

Upload the code to the MKR WiFi 1010 and then open the Serial Monitor. This should print out "*Bluetooth device active, waiting for connections...*" if all is working properly.

NOTE: You will need to have the Serial Monitor open to establish a BLE connection with this script because of the line: *while (!Serial);* (this is not needed in all use cases so you will need to determine if it is necessary in your final script)

Smartphone Setup (Central)

Now we will use the LightBlue Application to read the data from the pot on the Arduino. Follow along with the steps below on how to navigate through the application. If you are using a different application the steps should still be similar to what is outlined below.

1. Connect your Arduino to a computer and open the Serial Monitor (wait for it to display *"Bluetooth device active, waiting for connections..."*)
    a. The built-in LED will turn on to indicate the arduino is searching for connections.
    b. If the built-in LED starts blinking on and off every 0.1 seconds, then it failed to initialize the BLE library and you will need to try again.
2. Connect to **AnalogRead_pennKey** from the list of devices (you may need to scroll down or refresh the list) *[if you are on iOS this could be **MKR1010_pennkey**]*
    a. The built-in LED will turn off to indicate a successful connection.
3. Select the UUID that matches the newChar UUID set in the Arduino Code
4. Change the **Data Format** to **Unsigned Little-Endian** and then select the **Read Again** button to see the pot value

   a. This should match (or be 1 or 2 off from) the value being printed to the Serial Monitor
5. Now try turning the pot to a new position and reading the value again
   a. You should see the value change in the LightBlue Application

You can follow along with LightBlue on Android and with iOS.

Computer Setup (Central)

If you were able to successfully read the value of the pot with a smartphone then the next step will be establishing readings with a BLE enabled computer. We will now walk through how to control the Arduino with MATLAB (you are not limited to using MATLAB in your projects but the code provided will give you a framework for other programming languages).

Follow the link to download the MATLAB file (BLE_matlab_arduino_analog_read.m) needed to read the value of the pot and then follow the steps below to acquire the data:

1. Connect your Arduino to a computer and open the Serial Monitor (wait for it to display **"Bluetooth device active, waiting for connections..."**)
2. Edit the **pennKey** variable in the MATLAB code so that it matches the one defined in the Arduino script
3. Run the **Establish BLE connection with Arduino** section of the MATLAB code. Wait for the Command Window to display "**Connected…**"
4. Run the **Read value from characteristic** section of the MATLAB code.
   a. raw_data -- the data from the Arduino in a 4 byte array
   b. timestamp -- the date and time the data was collected
   c. data1 -- the data as a int32 data type converted with the OS's endian style
   d. data2 -- the data as a int32 data type converted with the swapped OS's endian style

Note: When you designing your own applications note if you need to use data1 or data2

Streaming Single Sensor Data from Arduino

Overview

In this example you will be using your smartphone (iOS or Android) and then a BLE enabled computer [These are acting as the **central device**] to read streamed data (read at 20 Hz) from a potentiometer (pot) connected to an Analog Pin on the Arduino MKR WiFi 1010 [this is acting as the **peripheral devices**]. The pot is a stand-in for whatever sensor you need for your project.

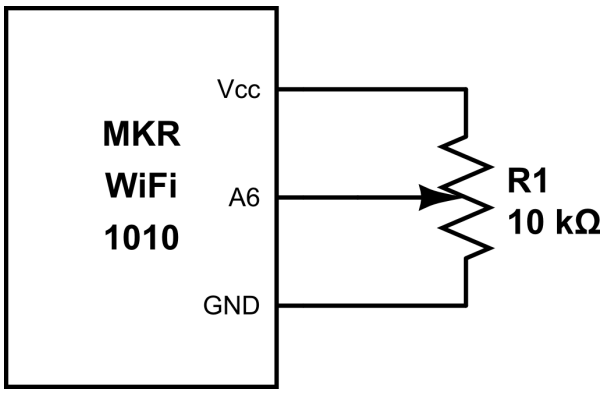
Note: From ArduinoBLE documentation it takes "a few milliseconds" to send a packet of data over bluetooth. I was able to get up to a sample rate of ~175 before significant signal damage occurred (in order to get the 175Hz you would need to calculate the sample period in microseconds). I believe this was due to the few milliseconds needed to send the packet of data. Because of this I moved down to a much more reliable 20 Hz sample rate.

Equipment needed:

- Arduino MKR WiFi 1010
- Micro USB cable
- Smartphone with a generic BLE central app (we demonstrate with LightBlue [iOS] [android]) or BLE enabled computer with coding software (we will demonstrate with MATLAB)
- Breadboard
- Jumper wire
- Potentiometer

Circuit

Note: Vcc on the MKR WiFi 1010 is 3.3V



Arduino Setup (Peripheral)

Follow the link to download the necessary Arduino file (BLE_arduino_stream.ino). Once downloaded you will need to edit the **pennKey** variable to your penn key [or any other short unique identifier]. The local name (which is usually what gets advertised) will be **stream_pennKey**. The device name (which can be advertised on iOS devices) will be **MKR1010_pennKey**. You need to have unique names so that your smartphone/computer can distinguish your Arduino from others in the area.

Upload the code to the MKR WiFi 1010

Smartphone Setup (Central)

Now we will use the LightBlue Application to read the streamed data from the pot on the Arduino. Follow along with the steps below on how to navigate through the application. If you are using a different application the steps should still be similar to what is outlined below.

1. Connect your Arduino to a computer
    a. The built-in LED will turn on to indicate the arduino is searching for connections.
    b. If the built-in LED starts blinking on and off every 0.1 seconds, then it failed to initialize the BLE library and you will need to try again.

2. Connect to **stream_pennKey** from the list of devices (you may need to scroll down or refresh the list) *[if you are on iOS this could be **MKR1010_pennkey**]*
    a. The built-in LED will turn off to indicate a successful connection.
3. Select the UUID that matches the newChar UUID set in the Arduino Code
4. Change the **Data Format** to **Unsigned Little-Endian** and then select the **Subscribe** button to see the pot value
    a. This should be a value between 0 and 1023 if it is larger switch to **Unsigned Big-Endian**
5. Now try turning the pot to a new position and observe how the value changes in the LightBlue Application

You can follow along with LightBlue on Android and with iOS.

Computer Setup (Central)

If you were able to successfully stream the value of the pot with a smartphone then the next step will be establishing the stream with a BLE enabled computer. We will now walk through how to control the Arduino with MATLAB (you are not limited to using MATLAB in your projects but the code provided will give you a framework for other programming languages).

Follow the links to download the MATLAB files (acquireBLEData.m, BLE_matlab_arduino_stream.m) needed to stream the value of the pot and then follow the steps below to acquire the data:

1. Connect your Arduino to a computer
2. Edit the **pennKey** variable in the MATLAB code so that it matches the one defined in the Arduino script
3. Run the **Establish BLE connection with Arduino** section of the MATLAB code. Wait for the Command Window to display “Connected…”
4. Run the **Live Plotting of Data** section of the MATLAB code.
5. While the plot is running try turning the pot to new positions and see how it affects the graph
6. You can change the **sample_time** variable if you would like more time before having to rerun the script

Note: If you want to recollect data you will need to rerun the entire script.

Streaming Multi-Sensor Data from Arduino

Overview

In this example you will be using your smartphone (iOS or Android) and then a BLE enabled computer [These are acting as the **central device**] to read streamed data (read at 20 Hz) from two potentiometers (pots) connected to Analog Pins on the Arduino MKR WiFi 1010 [this is acting as the **peripheral devices**]. The pots are a stand-in for whatever sensors you need for your project. In this tutorial each pot is treated as its own characteristic, which keeps data the

most readable. However, this is not the optimal configuration of the sensors if you are concerned about having a fast sample rate. Check out this Arduino Form post for information on how to package multiple sensors into one characteristic to increase your sample rate (https://forum.arduino.cc/t/ble-very-weak-signal/631751/33).

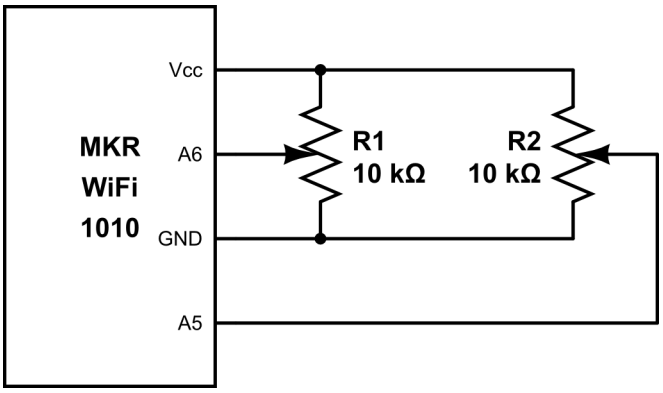
Note: From ArduinoBLE documentation it takes "a few milliseconds" to send a packet of data over bluetooth. I was able to get up to a sample rate of ~150 before significant signal damage occurred (in order to get the 150Hz you would need to calculate the sample period in microseconds).. I believe this was due to the few milliseconds needed to send the packet of data. Because of this I moved down to a much more reliable 20 Hz sample rate.

Equipment needed:

- Arduino MKR WiFi 1010
- Micro USB cable
- Smartphone with a generic BLE central app (we demonstrate with LightBlue [iOS] [android]) or BLE enabled computer with coding software (we will demonstrate with MATLAB)
- Breadboard
- Jumper wire
- 2 x Potentiometer

Circuit

Note: Vcc on the MKR WiFi 1010 is 3.3V



Arduino Setup (Peripheral)

Follow the link to download the necessary Arduino file (BLE_arduino_stream_multiple.ino). Once downloaded you will need to edit the **pennKey** variable to your penn key [or any other short unique identifier]. The local name (which is usually what gets advertised) will be **stream2_pennKey**. The device name (which can be advertised on iOS devices) will be **MKR1010_pennKey**. You need to have unique names so that your smartphone/computer can distinguish your Arduino from others in the area.

Upload the code to the MKR WiFi 1010

Smartphone Setup (Central)

Now we will use the LightBlue Application to read the streamed data from the pots on the Arduino. Follow along with the steps below on how to navigate through the application. If you are using a different application the steps should still be similar to what is outlined below.

1. Connect your Arduino to a computer
    a. The built-in LED will turn on to indicate the arduino is searching for connections.
    b. If the built-in LED starts blinking on and off every 0.1 seconds, then it failed to initialize the BLE library and you will need to try again.
2. Connect to **stream2_pennKey** from the list of devices (you may need to scroll down or refresh the list) *[if you are on iOS this could be **MKR1010_pennkey**]*
    a. The built-in LED will turn off to indicate a successful connection.
3. Select the UUID that matches the first characteristic UUID set in the Arduino Code
4. Change the **Data Format** to **Unsigned Little-Endian** and then select the **Subscribe** button to see the pot's value
    a. This should be between 0 and 1023 if it is larger switch to **Unsigned Big-Endian**
5. Now try turning the pots to new positions and observe how the values change in the LightBlue Application
6. Repeat steps 3-5 with the second characteristic UUID set in the Arduino Code

You can follow along with LightBlue on Android and with iOS.

Computer Setup (Central)

If you were able to successfully stream both values of the pots with a smartphone then the next step will be establishing the stream with a BLE enabled computer. We will now walk through how to control the Arduino with MATLAB (you are not limited to using MATLAB in your projects but the code provided will give you a framework for other programming languages).

Follow the links to download the MATLAB files (acquireBLEData.m, acquireBLEData1.m, BLE_matlab_arduino_stream_multiple.m) needed to stream both values of the pots and then follow the steps below to acquire the data:

1. Connect your Arduino to a computer
2. Edit the **pennKey** variable in the MATLAB code so that it matches the one defined in the Arduino script
3. Run the **Establish BLE connection with Arduino** section of the MATLAB code. Wait for the Command Window to display "Connected…"
4. Run the **Live Plotting of Data** section of the MATLAB code.
5. While the plot is running try turning both pots to new positions and see how it affects the graph
6. You can change the **sample_time** variable if you would like more time before having to rerun the script

Note: If you want to recollect data you will need to rerun the entire script.

Streaming Single Sensor Data from Arduino Fast Sample Rate

Overview

In this example you will be using your smartphone (iOS or Android) and then a BLE enabled computer [These are acting as the **central device**] to read streamed data (read at 1000 Hz) from a potentiometer (pot) connected to an Analog Pin on the Arduino MKR WiFi 1010 [this is acting as the **peripheral devices**]. The pot is a stand-in for whatever sensor you need for your project.

In order to achieve faster sampling rates then the previous example, you will need a buffer (something to store new values) and a sending rate (how frequently you send your buffer).

There are three different limiting factors at play when trying to get faster sampling rates. The first is how fast the Arduino can record a sample. This will depend on the arduinos clock speed and the computational cost and efficiency of the Arduino script being run. The second is how fast the BLE library can write reliable data (your buffer) to the BLE Characteristic. The third is how large of a buffer/packet the BLE library can write

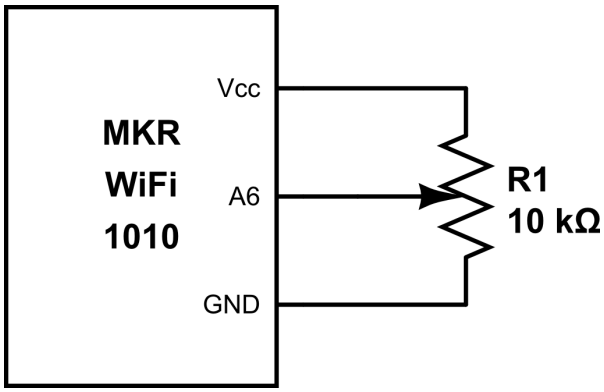
Note: I was able to get a max sample rate of ~1750Hz. In order to achieve this, I had to change the clock speed of the arduino, reduce my resolution to 8-bit and send multiple readings at once with a buffer. If you desire a faster sample rate you will have to modify code. One potential approach would be to incorporate interrupts.

Equipment needed:

- Arduino MKR WiFi 1010
- Micro USB cable
- Smartphone with a generic BLE central app (we demonstrate with LightBlue [iOS] [android]) or BLE enabled computer with coding software (we will demonstrate with MATLAB)
- Breadboard
- Jumper wire
- Potentiometer

Circuit

Note: Vcc on the MKR WiFi 1010 is 3.3V



Arduino Setup (Peripheral)

Follow the link to download the necessary Arduino file (BLE_arduino_stream_fast.ino). Once downloaded you will need to edit the **pennKey** variable to your penn key [or any other short unique identifier]. The local name (which is usually what gets advertised) will be **stream3_pennKey**. The device name (which can be advertised on iOS devices) will be **MKR1010_pennKey**. You need to have unique names so that your smartphone/computer can distinguish your Arduino from others in the area.

Upload the code to the MKR WiFi 1010

Smartphone Setup (Central)

Now we will use the LightBlue Application to read the streamed data from the pot on the Arduino. Follow along with the steps below on how to navigate through the application. If you are using a different application the steps should still be similar to what is outlined below.

1. Connect your Arduino to a computer
    a. The built-in LED will turn on to indicate the arduino is searching for connections.
    b. If the built-in LED starts blinking on and off every 0.1 seconds, then it failed to initialize the BLE library and you will need to try again.
2. Connect to **stream3_pennKey** from the list of devices (you may need to scroll down or refresh the list) *[if you are on iOS this could be **MKR1010_pennkey**]*
    a. The built-in LED will turn off to indicate a successful connection.
3. Select the UUID that matches the newChar UUID set in the Arduino Code
4. Change the **Data Format** to **Unsigned Little-Endian** and then select the **Subscribe** button to see the pot value
    a. This should be a value between 0 and 1023 if it is larger switch to **Unsigned Big-Endian**
5. Now try turning the pot to a new position and observe how the value changes in the LightBlue Application

You can follow along with LightBlue on Android and with iOS.

Computer Setup (Central)

If you were able to successfully stream the value of the pot with a smartphone then the next step will be establishing the stream with a BLE enabled computer. We will now walk through how to control the Arduino with MATLAB (you are not limited to using MATLAB in your projects but the code provided will give you a framework for other programming languages).

Follow the links to download the MATLAB files (acquireBLEDataFast.m, BLE_matlab_arduino_stream_fast.m) needed to stream the value of the pot and then follow the steps below to acquire the data:

1. Connect your Arduino to a computer
2. Edit the **pennKey** variable in the MATLAB code so that it matches the one defined in the Arduino script
3. Run the **Establish BLE connection with Arduino** section of the MATLAB code. Wait for the Command Window to display "Connected…"
4. Run the **Live Plotting of Data** section of the MATLAB code.
5. While the plot is running try turning the pot to new positions and see how it affects the graph
6. You can change the **sample_time** variable if you would like more time before having to rerun the script

Note: If you want to recollect data you will need to rerun the entire script.

Receiving Streamed Data Single Sensor

Overview

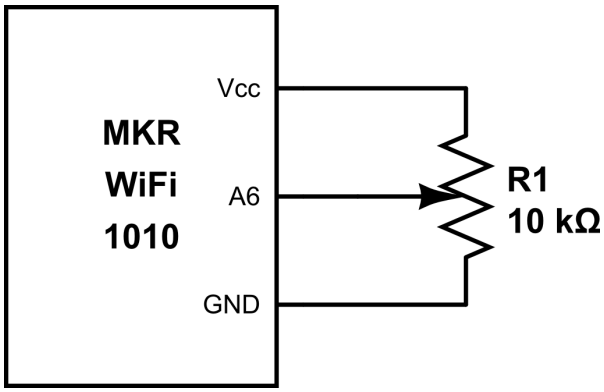
In this example you will be using two Arduino MKR WiFi 1010s [One acting as the **central device** and the other acting as the **peripheral devices**] to read streamed data (read at 1000 Hz) from a potentiometer (pot) connected to an Analog Pin on one of the Arduino MKR WiFi 1010s. The pot is a stand-in for whatever sensor you need for your project.

Equipment needed:

- 2X Arduino MKR WiFi 1010
- 2X Micro USB cable
- Breadboard
- Jumper wire
- Potentiometer

Circuit

Note: Vcc on the MKR WiFi 1010 is 3.3V



Even though you have two Arduinos, you only need one circuit as the Central Device will just be plotting the received data.

Arduino Setup (Peripheral)

We will use the same file from the Streaming Single Sensor Data from Arduino Fast Sample Rate Section. If you do not already have the file, follow the link to download it (BLE_arduino_stream_fast.ino). Once downloaded you will need to edit the **pennKey** variable to your penn key [or any other short unique identifier]. The local name (which is usually what gets advertised) will be **stream3_pennKey**. The device name (which can be advertised on iOS devices) will be **MKR1010_pennKey**. You need to have unique names so that your smartphone/computer can distinguish your Arduino from others in the area. Upload the code to the MKR WiFi 1010

Arduino Setup (Central)

1. Follow the link to download the necessary Arduino file (BLE_Arduino_Central.ino).
2. Since you are using a new arduino, you will need to update the firmware before uploading the BLE_Arduino_Central sketch.
3. Once downloaded you will need to edit the **pennKey** variable to your penn key [or any other short unique identifier]. You need to have unique names so that your central Arduino can distinguish your peripheral Arduino from others in the area.
4. Upload the code to the MKR WiFi 1010
5. Open the serial monitor to see the changes in the printed value as you turn the potentiometer

Sources

Bluetooth Radio Versions | Bluetooth® Technology Website

Understanding Bluetooth Range | Bluetooth® Technology Website
Reliability | Bluetooth® Technology Website
Bluetooth Security | Bluetooth® Technology Website
Specifications | Bluetooth® Technology Website
Arduino - ArduinoBLE
Getting started with the MKR WiFi 1010
WiFiNINA Firmware Updater | Arduino
MKR WiFi 1010 Bluetooth Low Energy | Arduino
How to convert 2 byte data to integer? - MATLAB Answers - MATLAB Central
BLE very weak signal - Page 3
Arduino fill array with values from analogRead