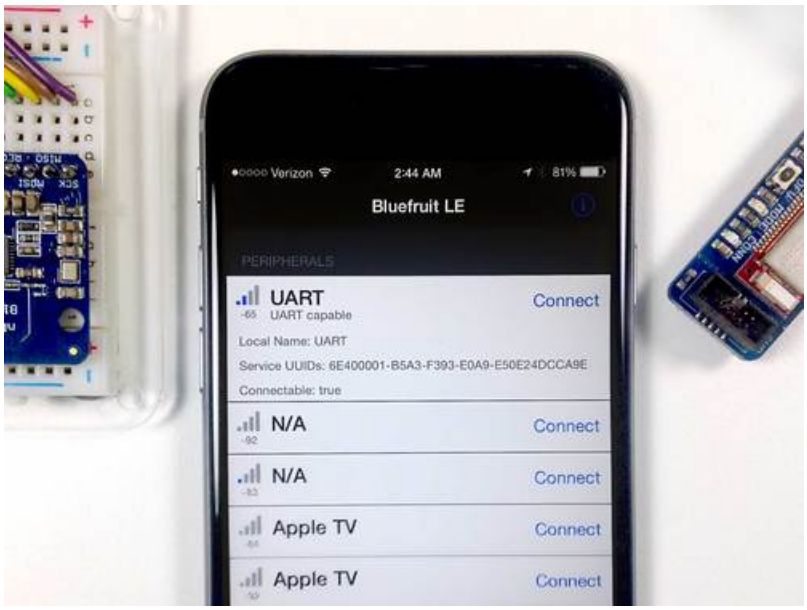
adafruit learning system

Bluefruit LE Connect for iOS

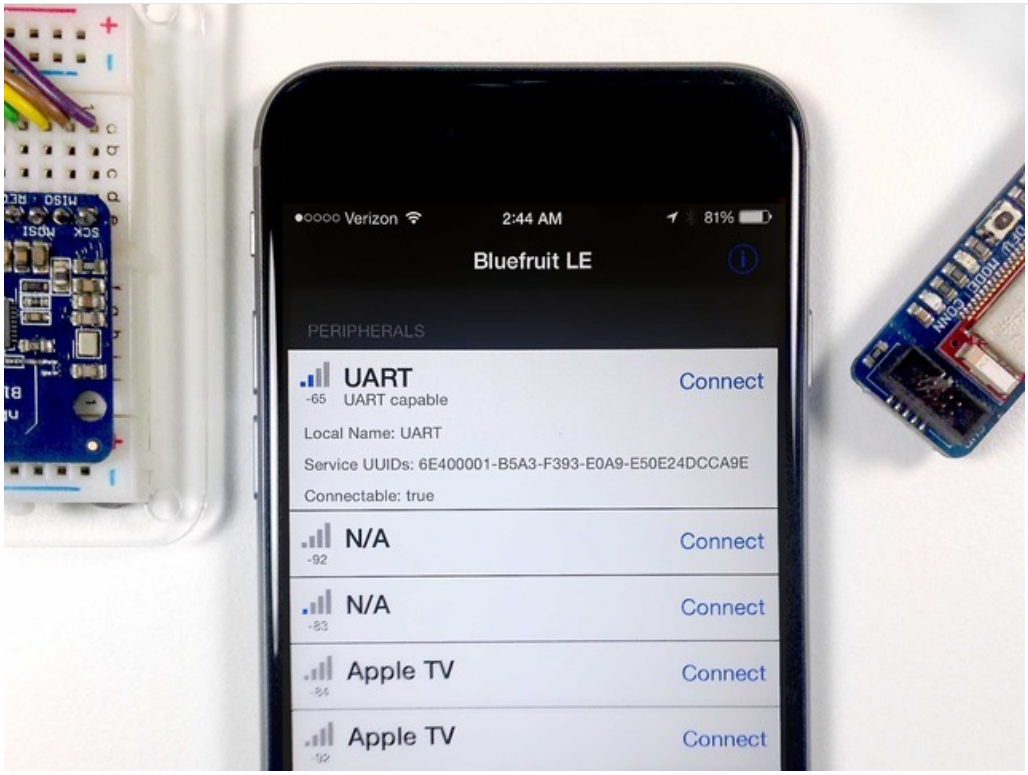
Created by Collin Cunningham



Last updated on 2018-03-26 09:54:50 PM UTC

Guide Contents

iOS Setup



The Bluefruit LE Connect app provides iOS devices with a variety of tools to communicate with Bluefruit LE devices. These tools cover basic communication and info reporting as well as more project specific uses such as Arduino Pin Control and a Color Picker.

The app is available as a free download from Apple's App Store and is compatible with the following iOS devices:

- iPhone 4s or newer
- iPad 3rd generation or newer
- iPod touch 5th generation or newer

The app is compatible with these BLE devices from Adafruit:

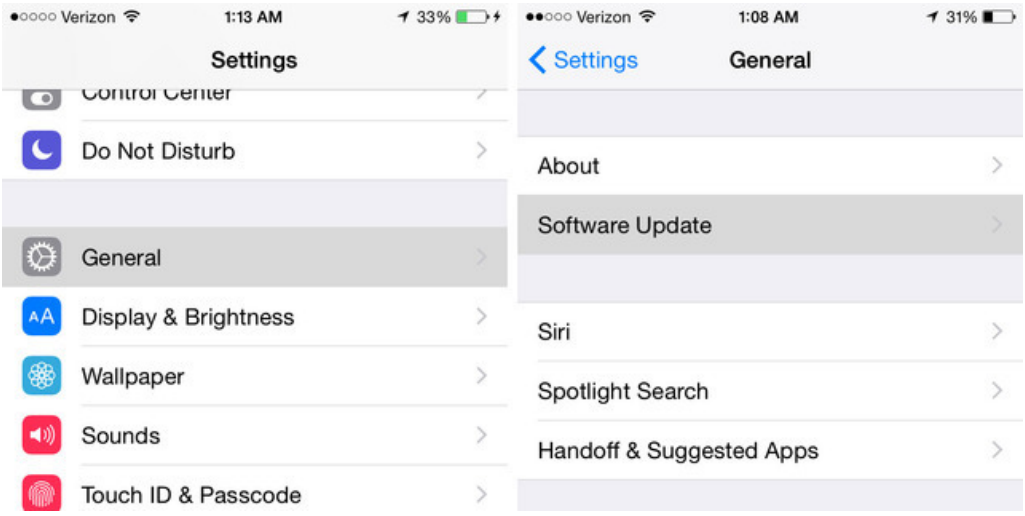
- Bluefruit LE nRF8001 Breakout
- Bluefruit LE Friend
- Flora Wearable Bluefruit LE Module
- Adafruit Bluefruit LE SPI Friend
- Adafruit Bluefruit LE Micro
- Adafruit Feather 32u4 Bluefruit LE
- Adafruit Feather M0 Bluefruit LE

First off - install the app from the App Store if you haven't already.

Update iOS

Update your device to the latest version of iOS by going to:

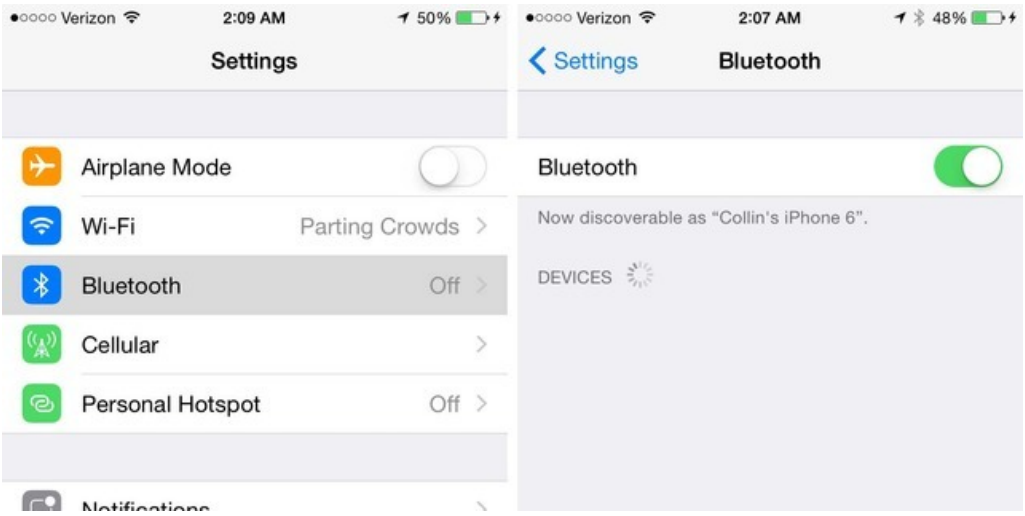
Settings->General->Software Update



Enable Bluetooth

If Bluetooth is disabled on your device, enable it by going to:

Settings->Bluetooth



Enable Location Services

If you plan to use the app to send location/GPS data to Bluefruit LE, enable Location Services via:

Settings->Privacy->Location Services


Verizon
2:09 AM
50%
Settings
Wallpaper
Sounds
Touch ID & Passcode
Privacy
iCloud
me@collinmel.com
iTunes & App Store


Verizon
2:11 AM
51%
Settings
Privacy
Location Services
Off
Contacts
Calendars
Reminders
Photos
Bluetooth Sharing

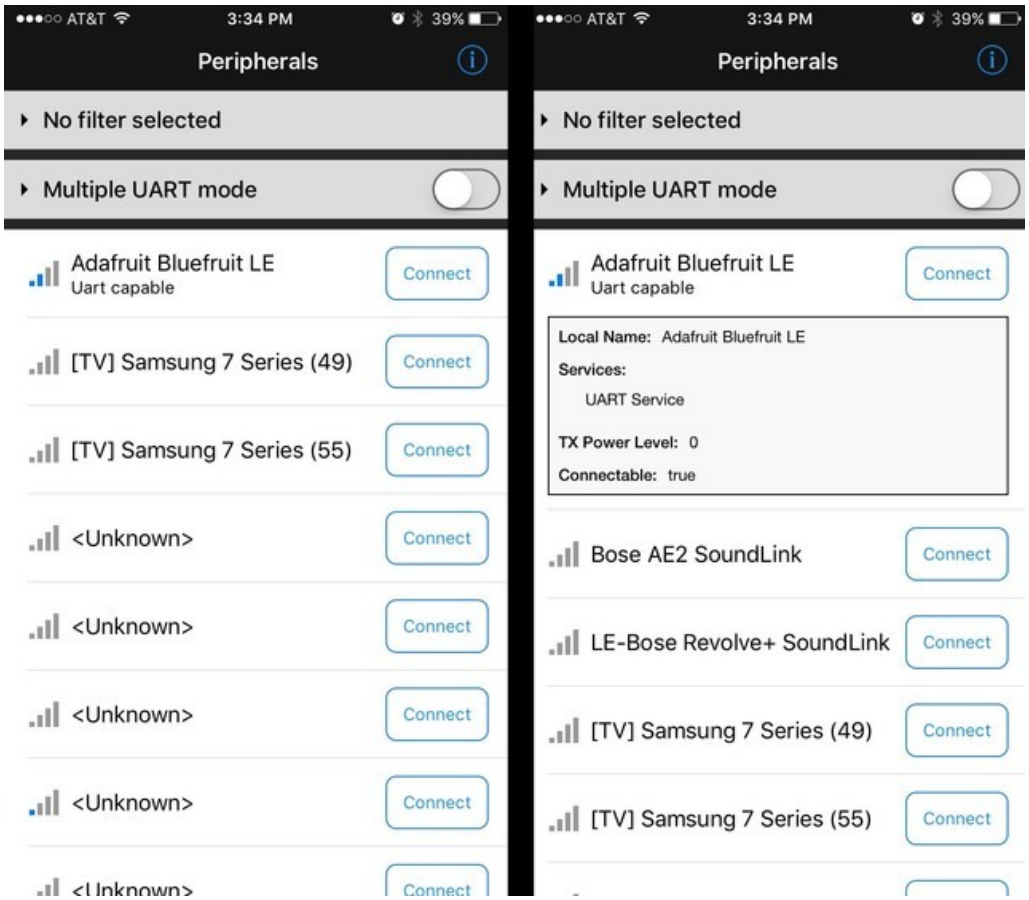
Scan for Devices

On launch, the app will automatically begin to scan for nearby Bluetooth LE devices. Devices are presented in a table view in the order in which they were discovered.

Don't forget to turn on Bluetooth on your device! Airplane Mode turns off BLE

iPhone

The following images depict the app when used on the iPhone

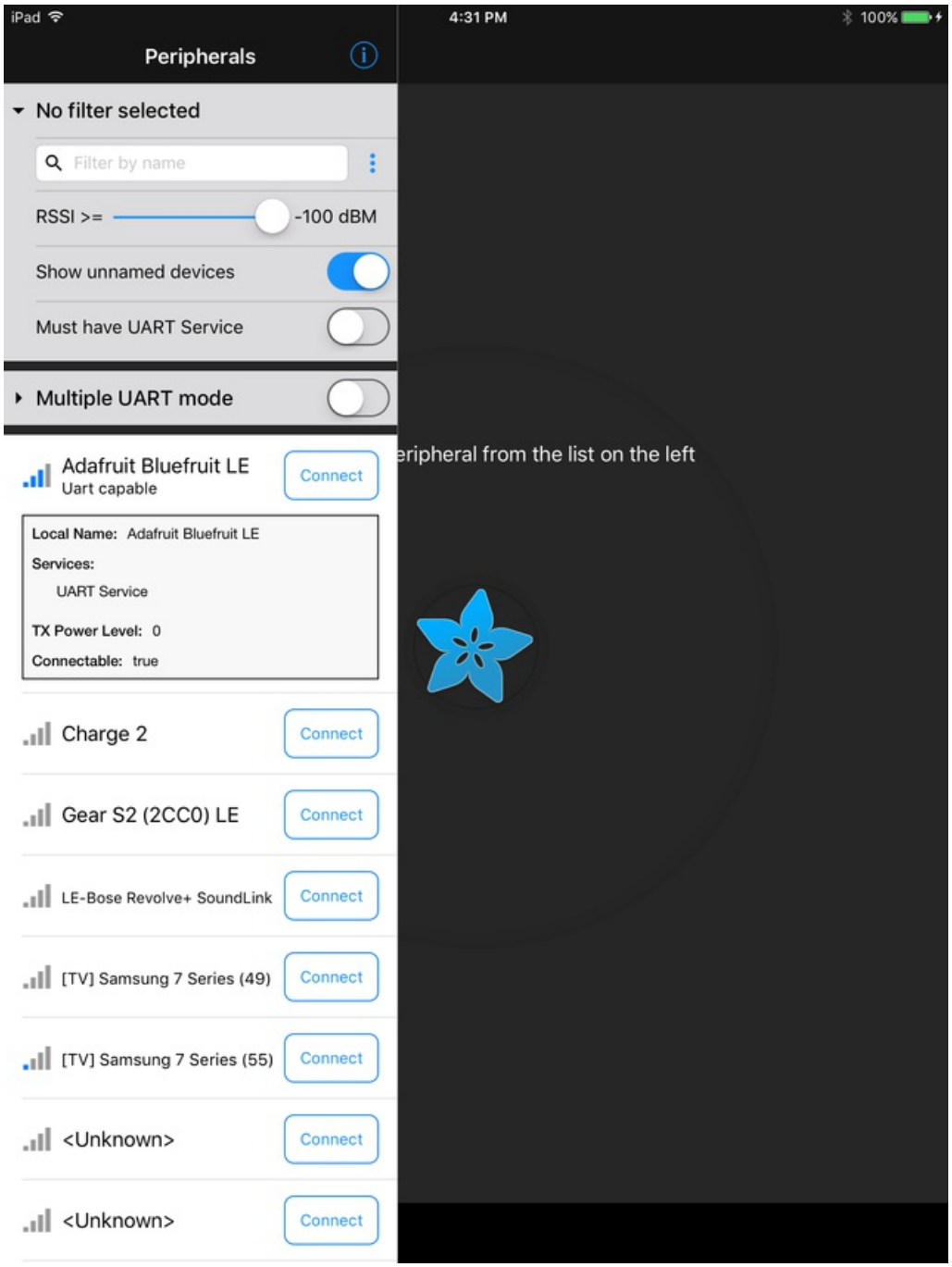


The device list will display all BLE devices discovered by the app (not just Bluefruit hardware) - so you may see a quite a few "N/A" entries for devices that don't share their name, as seen above.

- If scanning does not automatically begin, it can be started by tapping "Scan for peripherals" in the bottom bar.
- To refresh the list and start a new scan, simply swipe down on the current list.
- Each device's signal strength is displayed in the left side of its row.

iPad

The following images depict the app when used on the iPad



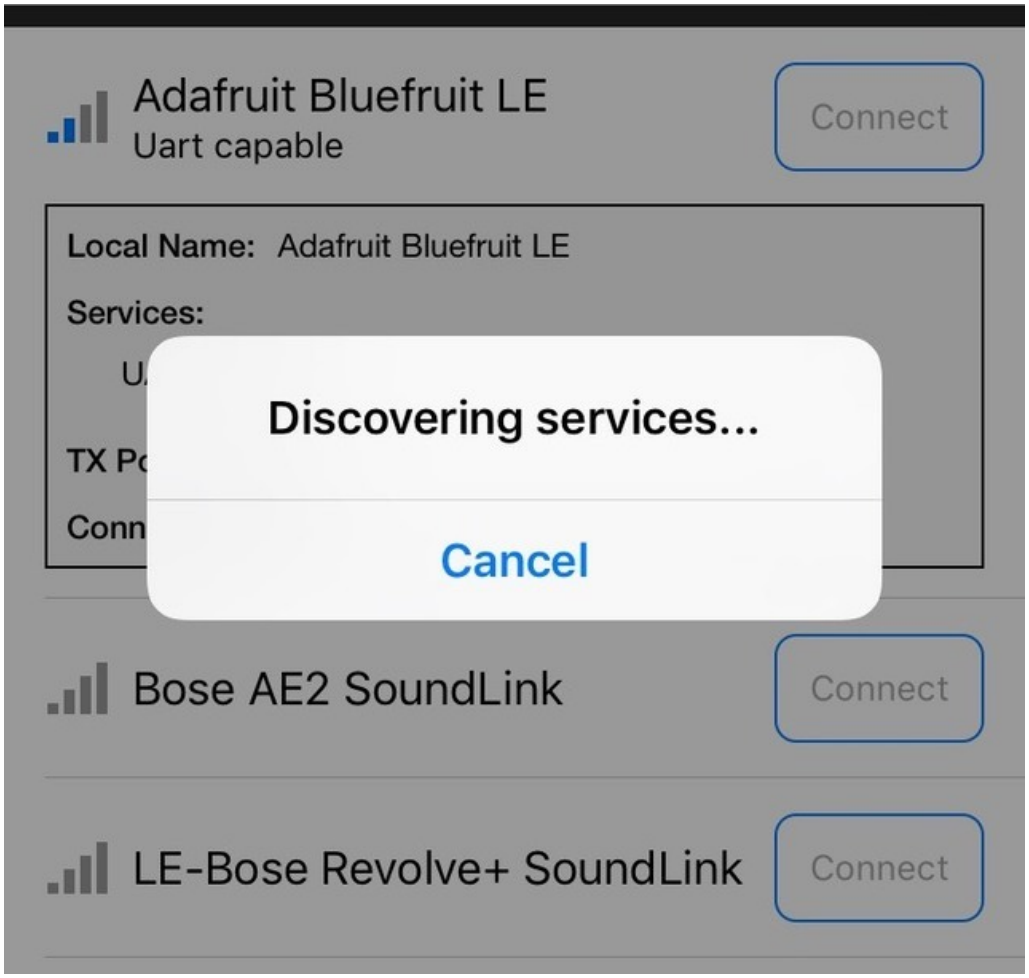
Tap the middle of a device's table row to reveal its relevant advertisement data.

- Any device listed with a "Connect" button at the right can be accessed in Info mode.
- Any device listed as "UART Capable" can be used with all modes - Info, UART, Pin I/O, & Controller.

To use the Bluefruit Connect app with your device it must be "UART capable"! All Adafruit BLE devices implement the UART interface, but other devices may not

Connect

Tap the Connect button on the UART capable list entry you wish to use and choose a connection mode from the menu that appears.



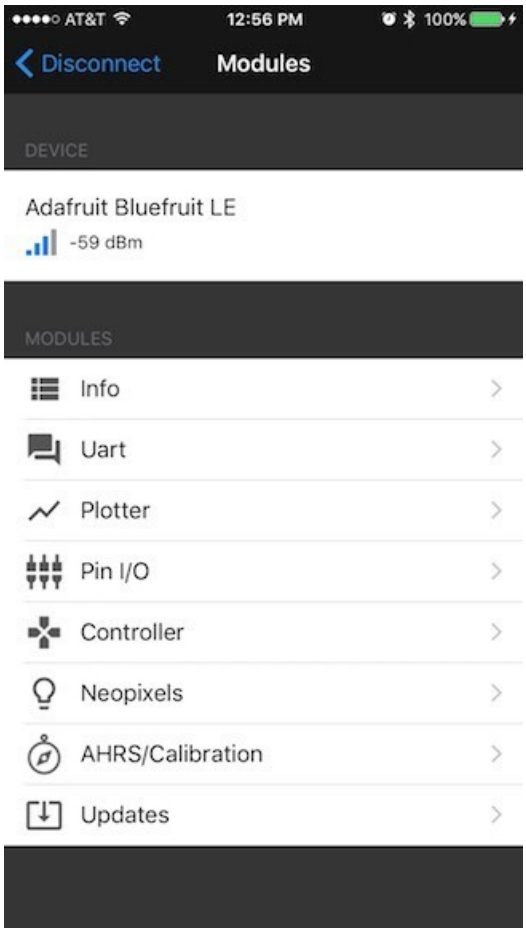
Features

Even though the Bluefruit Connect app only uses **UART** as the *transport* for sending and receiving data to/from your BLE device, it has **multiple interaction modes**

Each of these modes can do different things and let you interact in a unique way.

Don't forget! All of these modes use the **UART** Service, but present the data in a different way. On the hardware side, your firmware will have to know what it is expecting and sometimes may need to parse the data coming back from the app.

For example, if using the **Color Picker**, the app will send the color data in a mini packet. If using the button controller, you'll get button presses/releases in packets instead.



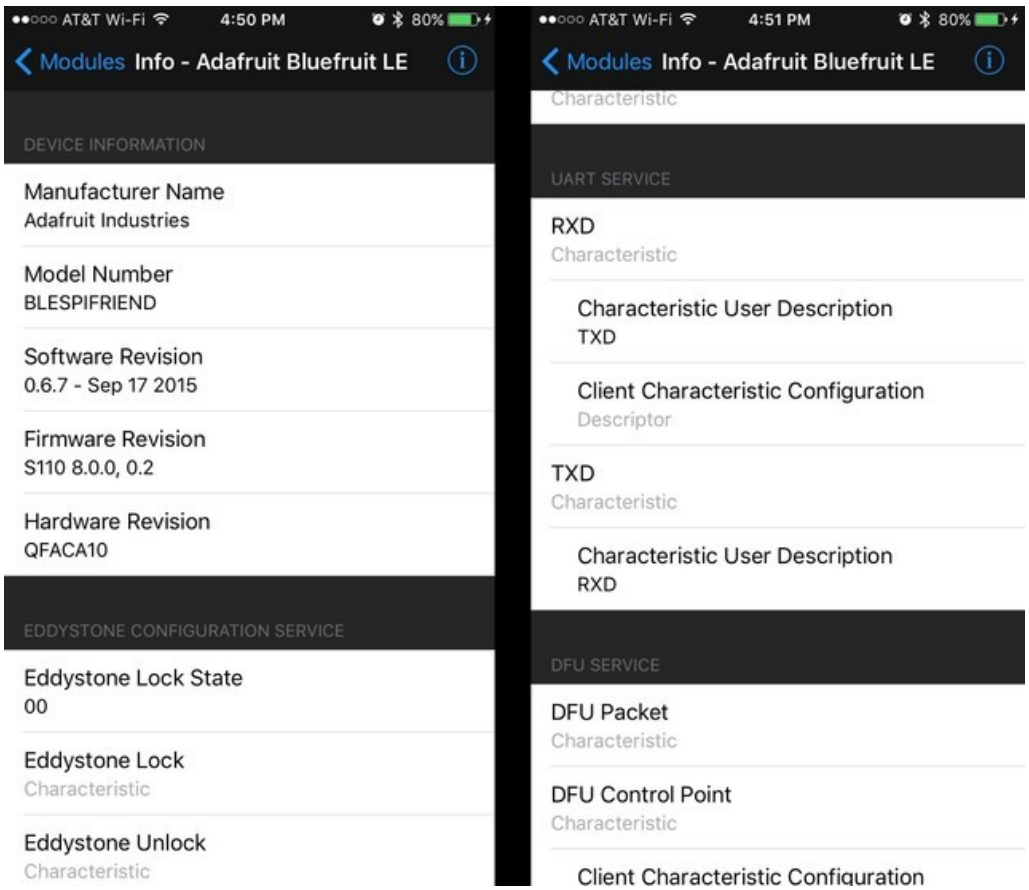
Available Modes:

- Info
- UART Terminal Mode
- Plotter
- Controller
- Neopixels
- Updates

Info

Connecting to a peripheral in Info mode will display its Generic Attribute Profile (**GATT**) in the form of a table.

- This mode is available for all connectable BLE devices and can be helpful for learning, troubleshooting, and general curious snooping.
- Tapping on a service row will reveal that service's included characteristics.



To learn more about Bluetooth Services & Characteristics, be sure to check out the Introduction to Bluetooth Low Energy guide.

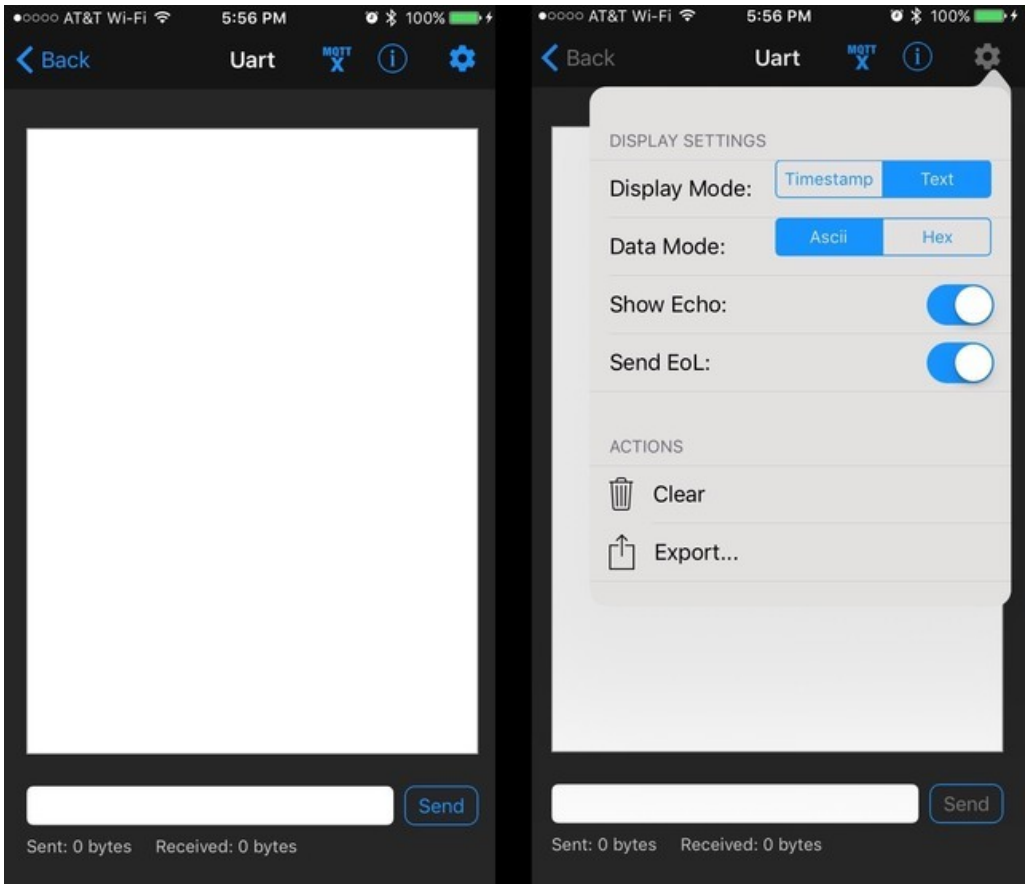
UART Terminal

The UART Terminal mode provides a classic 'serial terminal' interface for sending and receiving strings from a Bluefruit LE device.

It's perfect for sending and receiving data without any interpretation.

Note: This mode can be used in conjunction with Bluefruit LE Friend's Command mode to configure or get additional info about the device.

Main Window



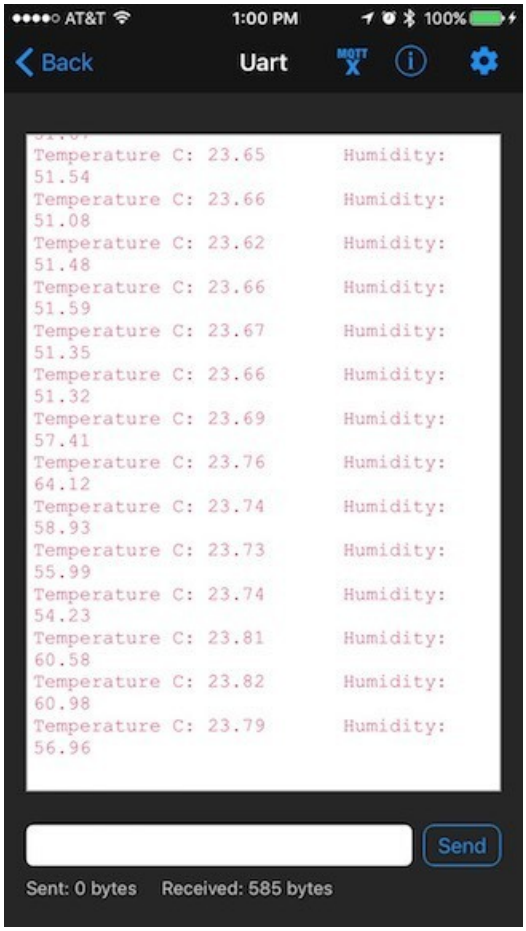
The main log window is in the middle, and will display data both received and, if Echo is on, sent.

Echo

Toggle the **Echo** switch in the settings the upper right to also display outgoing data sent from the app.

Data received from the remote BLE device appears in red.

Data sent from the app appears in blue.

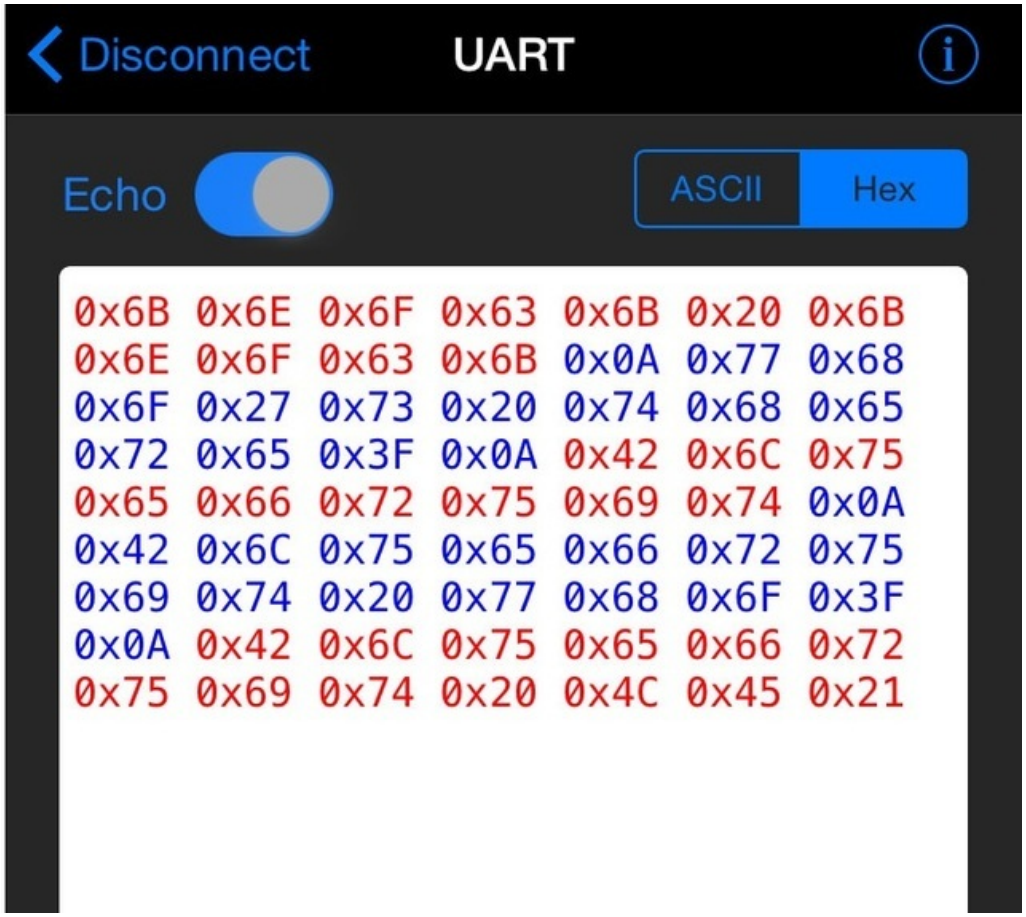


ASCII / HEX

The display format of the log window can be controlled using the **ASCII**/**Hex** switch in settings in the upper right.

ASCII will do its best to translate the data to 8-bit human-viewable text characters.

Hex will give you 0xnn formatted bytes, still color coded.

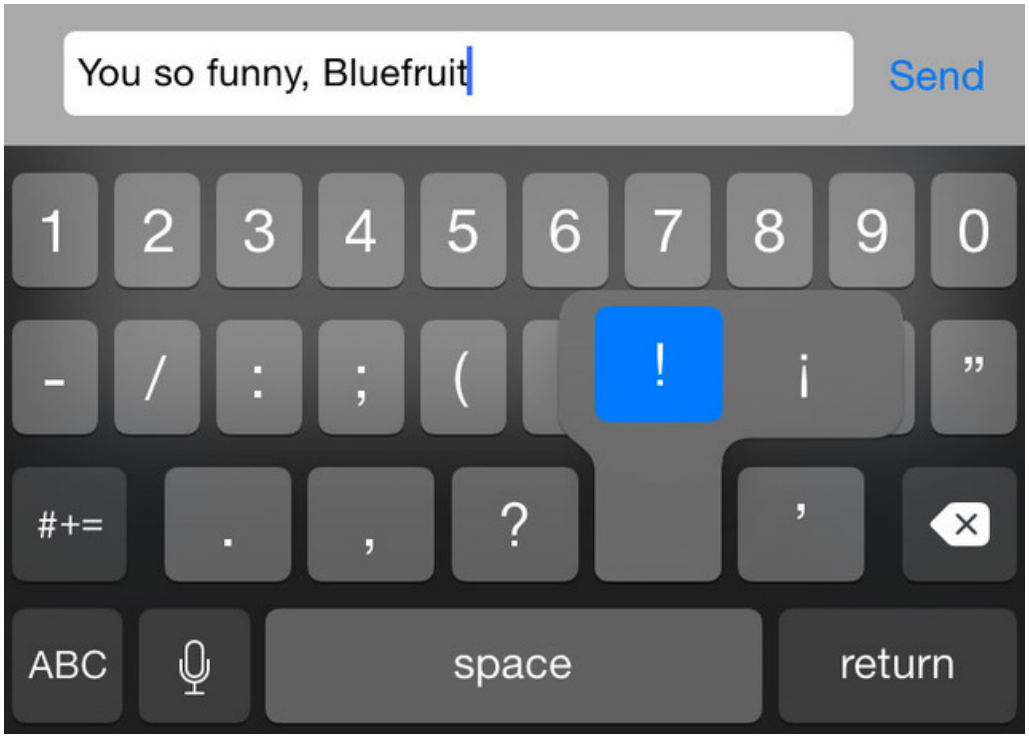


Copy & Clear

Tapping **Copy** will copy all text from the log window to the iOS clipboard.

Tapping **Clear** will delete all text from the window.

Sending Strings



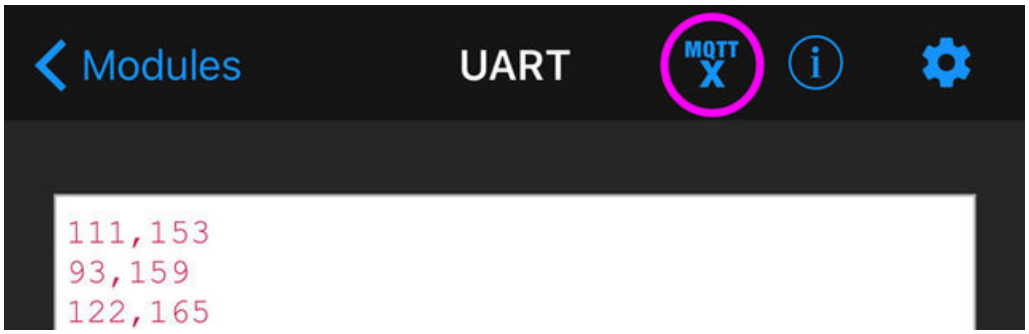
- Tap the text field at the bottom of the screen to bring up the keyboard and begin composing a new string to send.
- Newline characters can be added using the return key.
- Press the **Send** button to send the string over UART to your Bluefruit LE device.
- To hide the keyboard, simply tap the log window while the keyboard is shown.

MQTT

MQTT stands for Message Queue Telemetry Transport. It's a protocol designed for low-bandwidth, high latency networks. You can learn more about it here.

The Bluefruit LE Connect App allows you to send and receive data using the MQTT protocol. For example - this can be useful for making sensor readings from a Bluefruit device viewable on the web.

To access the MQTT settings, simply click the MQTT button in the top right corner of the UART Terminal.

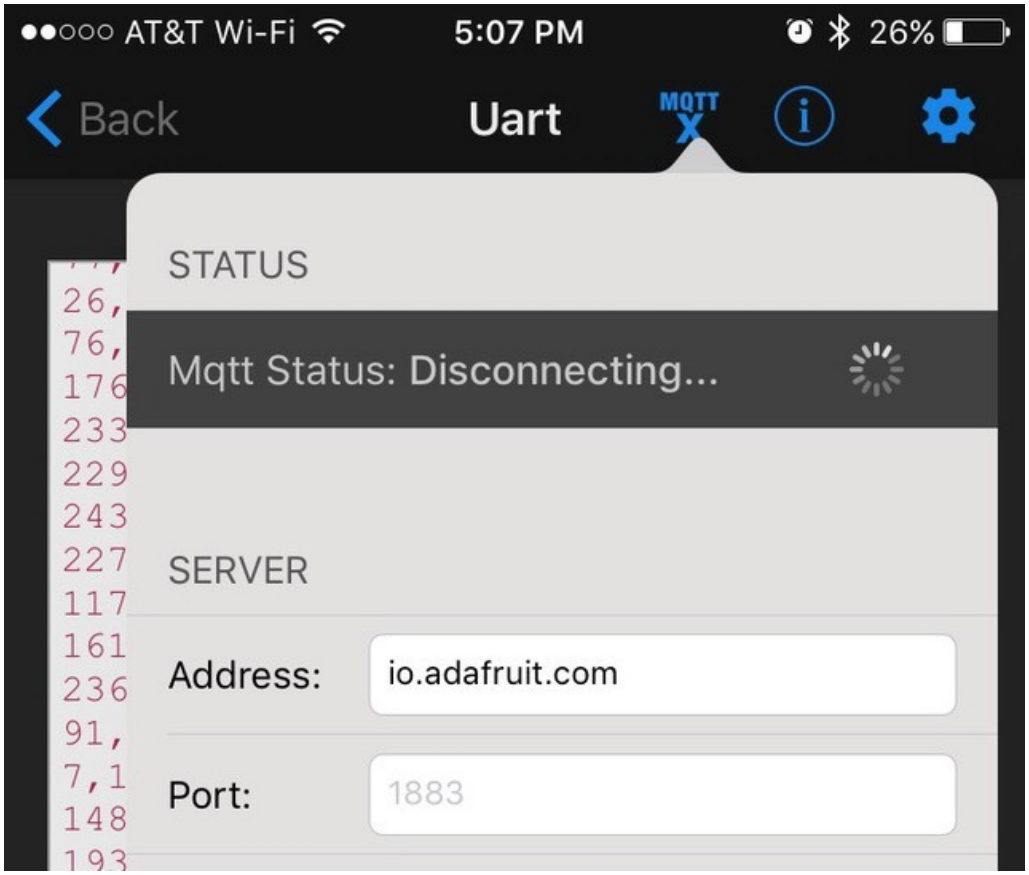


Configuration

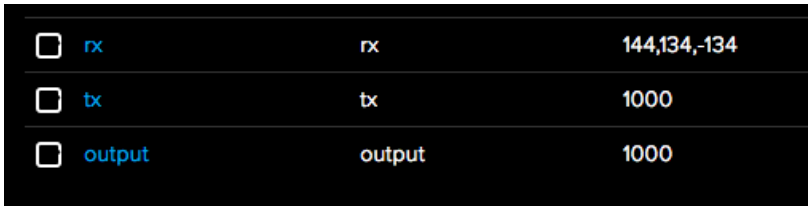
Firstly, you will need an adafruit.io account in order to log and collect your data. You can make one here. To learn more about adafruit.io, check out the guide for it here.

Once you have your adafruit.io account set up, connect to your device using the Bluefruit LE Connect app and open the **UART** mode. In the UART module, tap the "**MQTT X**" button at the top of the screen.

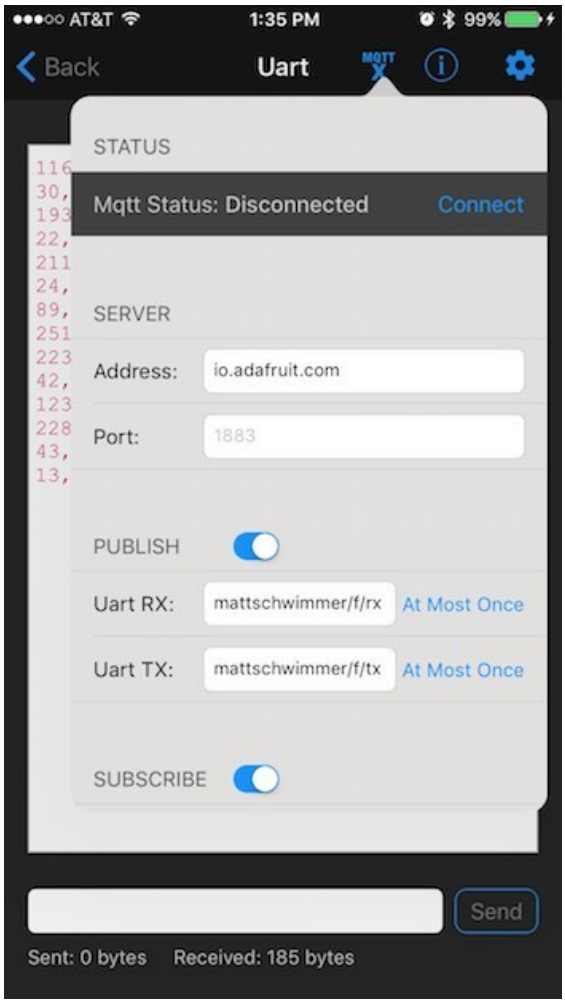
Under **SERVER**, enter "io.adafruit.com" as the server **address** and "1883" as the **port**.



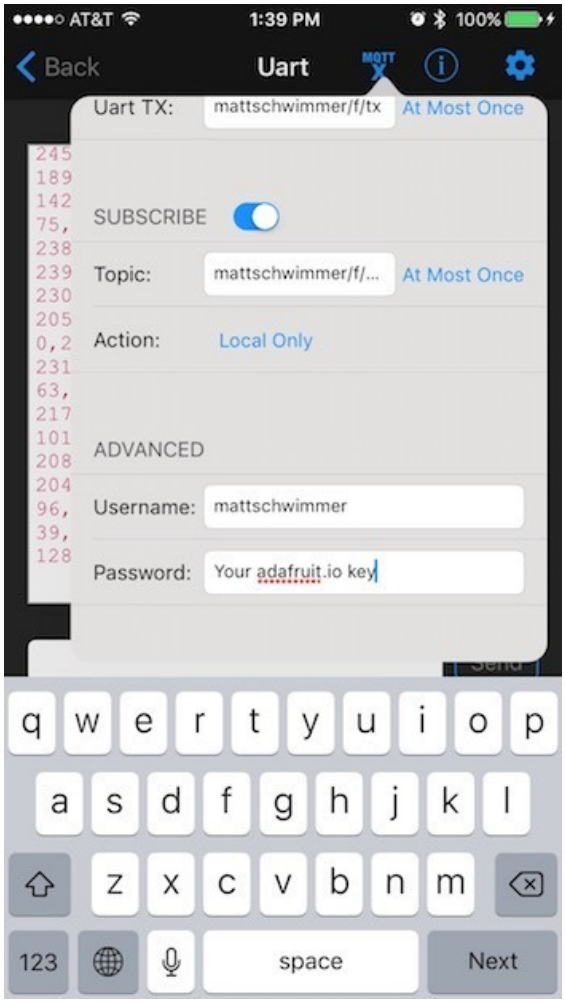
Before filling in the necessary information, you will need to make 3 feeds on your adafruit.io account.



You can name them anything you like, but in this example will name the feed used to display data received by the board as "**rx**"), and another used to display data entered through the app as **"tx".** We'll also have a third feed named **"output"** which will display new data on the Bluefruit LE Connect app, called "**output**".

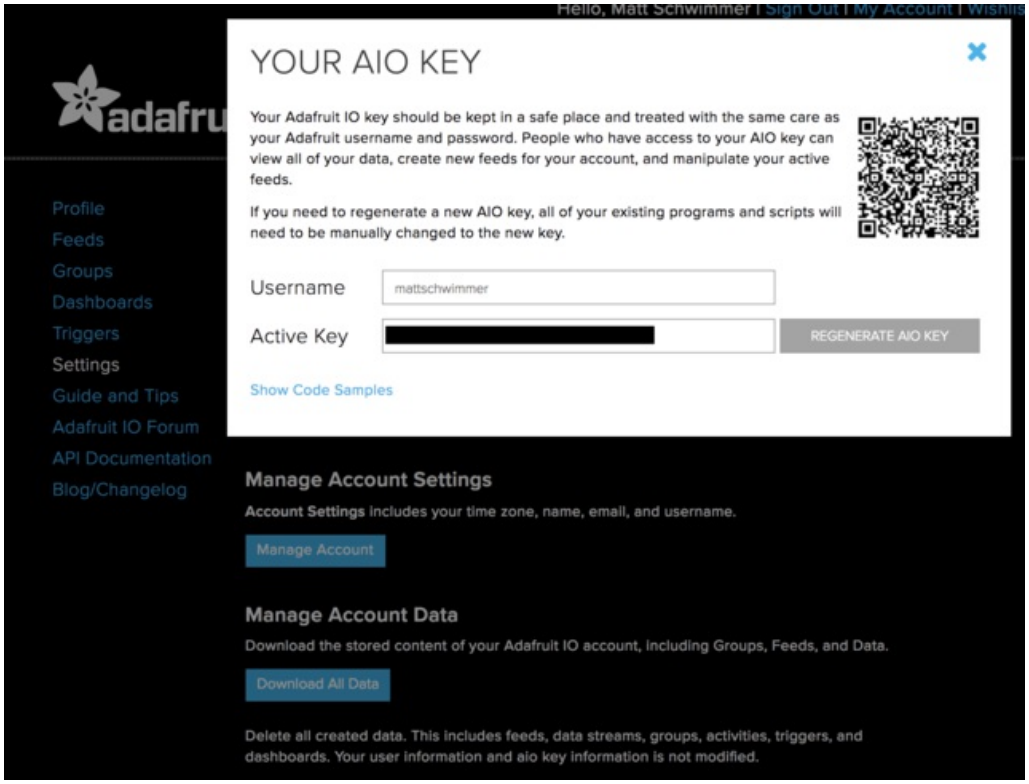


Under PUBLISH, enter in <username>/f/<respective feed name> for both UART RX and TX.



Under SUBSCRIBE, use your third output feed name as input.

Finally, under ADVANCED, use your adafruit.io username. Under Password, use your adafruit.io account's KEY. This can be found under the settings of your adafruit.io account.

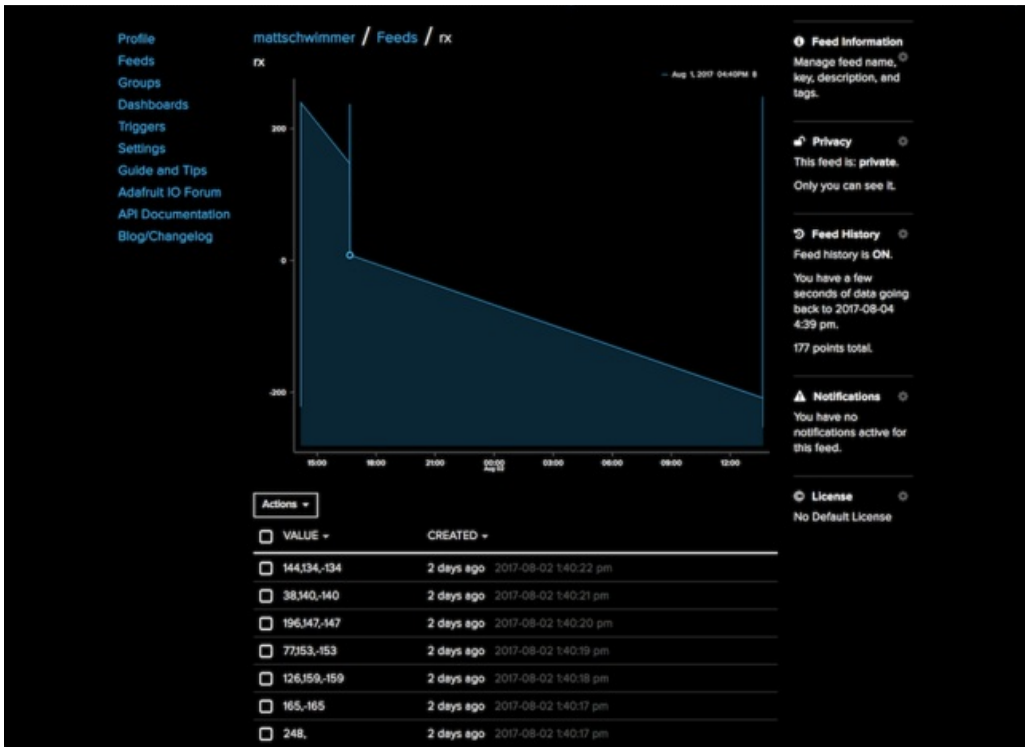


DO NOT USE YOUR ADAFRUIT.IO ACCOUNT PASSWORD. Simply use the generated key for your account.

Example

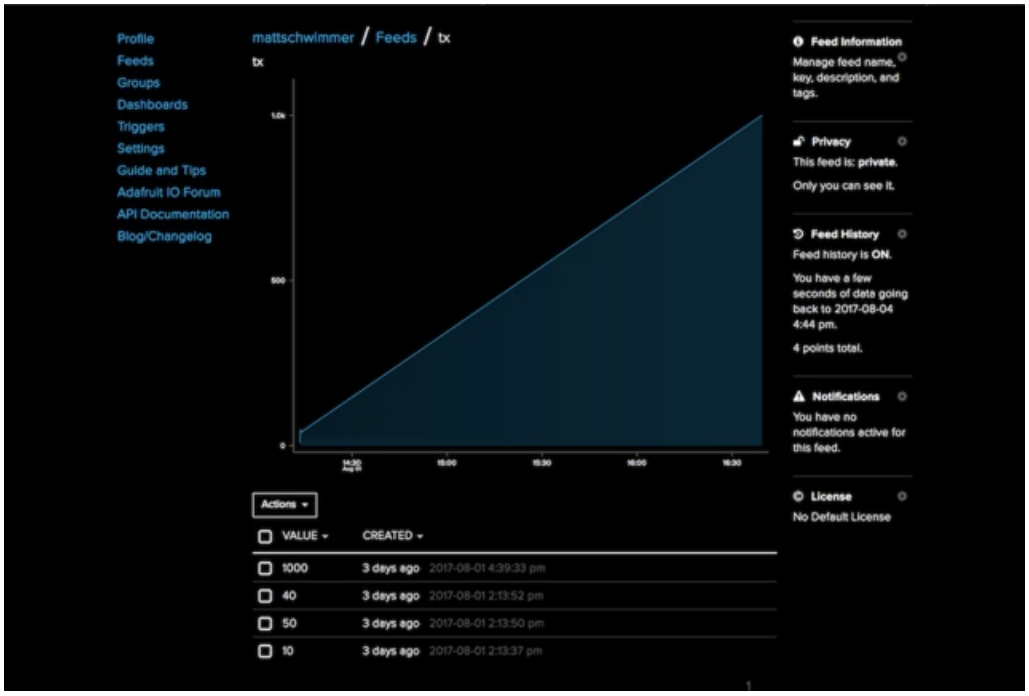
Here are some examples of feeds created using the Bluefruit LE Connect app in the same manor as described above.

Received Data



This is an example of an RX feed that presents data **sent** from an Adafruit Feather Bluefruit and is **received** by iPhone using the Bluefruit LE Connect app via MQTT. The data is sent using an ASCII Numeric format similar to how plotter data is sent. You can see the plotter feature here.

Written Data



This is an example of a TX feed that presents data **sent** from the Bluefruit LE Connect app to adafruit.io via MQTT.

Output Data

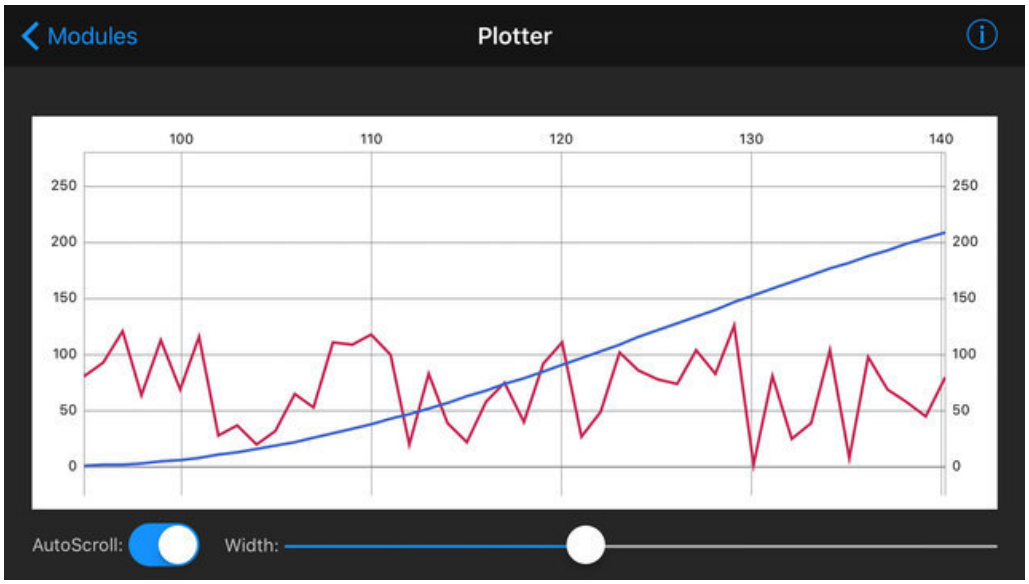
This is an example of an output feed that presents data created on adafruit.io **sent** to the Bluefruit LE Connect app via MQTT

Plotter

The Plotter mode allows for users to visualize data received from Bluetooth LE compatible devices.

This data is sent **from the Bluefruit device** to the phone/tablet.

Feel free to test the plotting feature using this demo sketch!

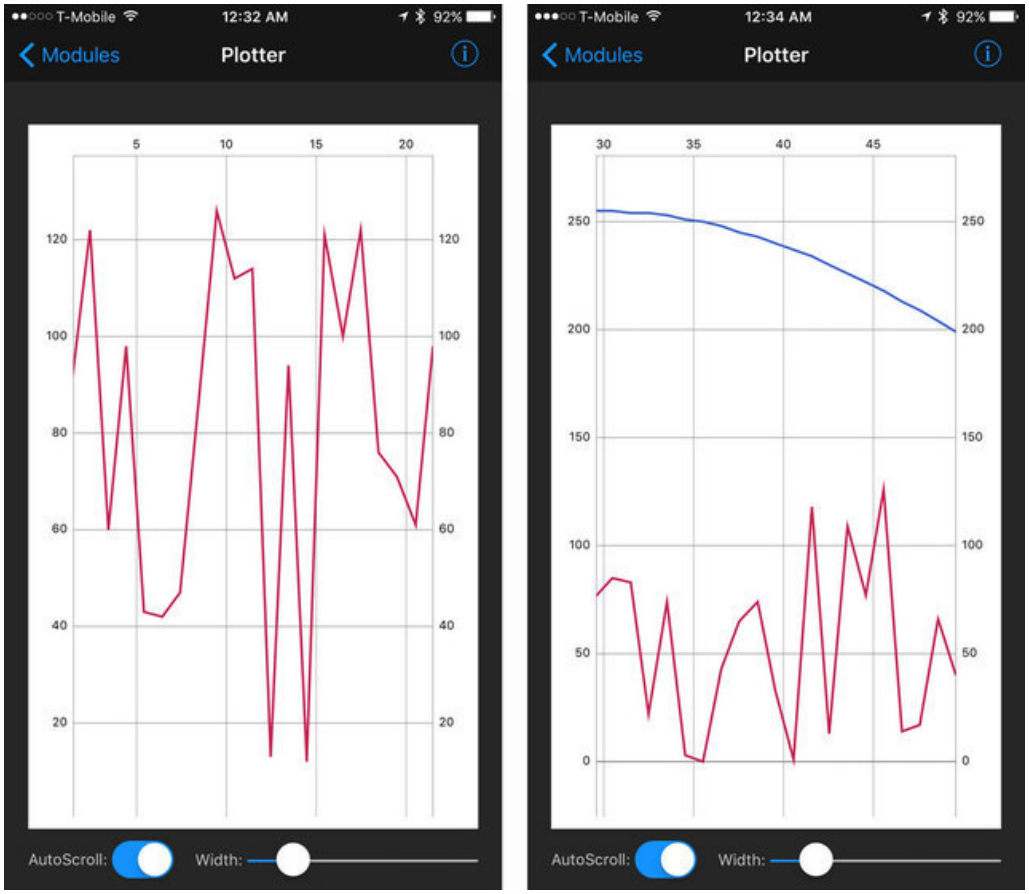


Main Plotter View

The main plotter window does not display X and Y axes unless the Bluefruit Connect app receives the necessary data to plot.

This is what the Bluefruit LE Connect app displays by default if there is no data to plot.

Once the required data is received, the plotting begins!

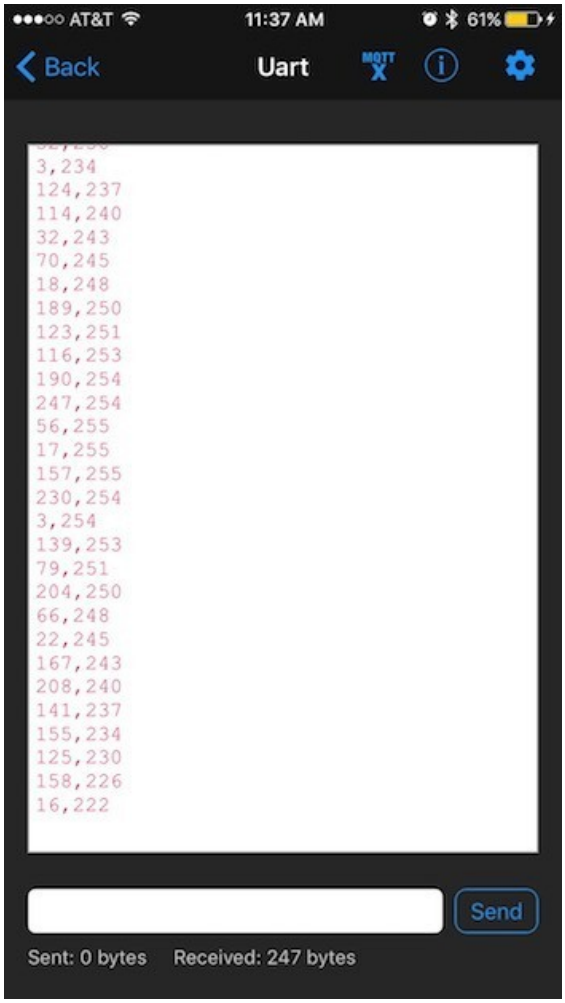


If your project utilizes multiple data streams, Bluefruit LE Connect will plot both for you! Above, notice that the graph on the right shows a second set of data.

Formatting

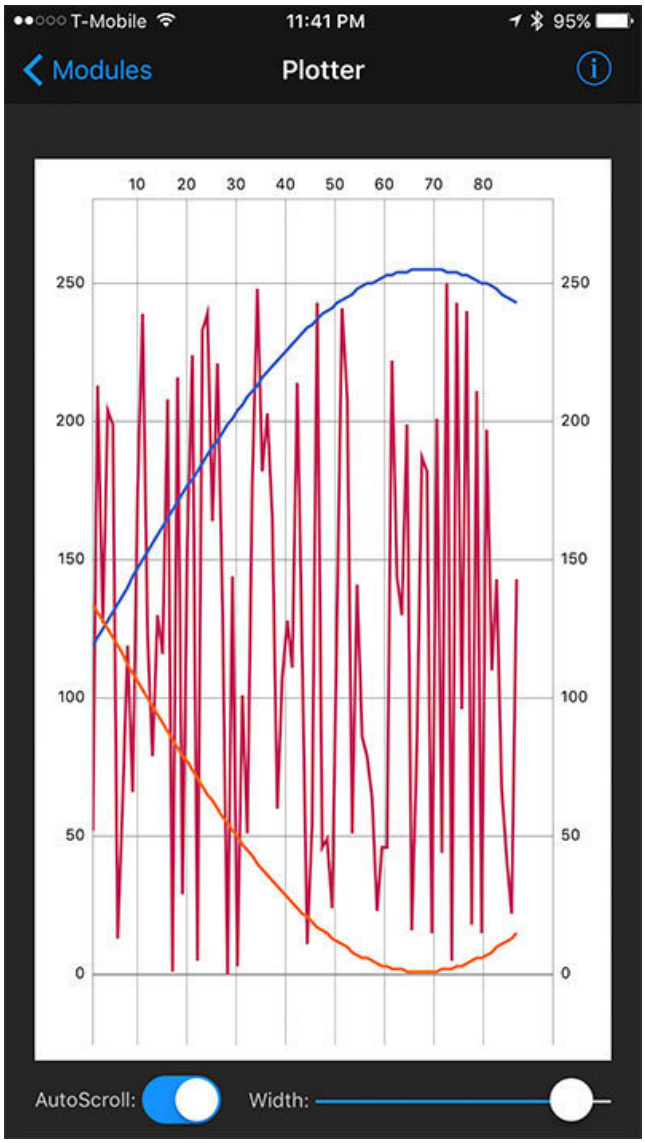
The app interprets incoming numeric values in ASCII format. Separate data values should be followed by a comma or tab character. Separate each set of values sent using a newline character or simply use `ble.println()` (which automatically appends a newline char). This allows Bluefruit LE Connect to know when to plot the next set of values.

For example, a properly formatted stream of plotter values will look like this when viewed in the app's UART terminal:



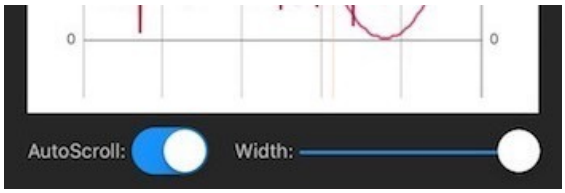
If you need to plot more data streams, simply add an additional comma followed by another numeric value.

A plot utilizing 3 data streams would look something like the picture shown below:



The plotter also supports landscape view!

AutoScroll and Plot Width



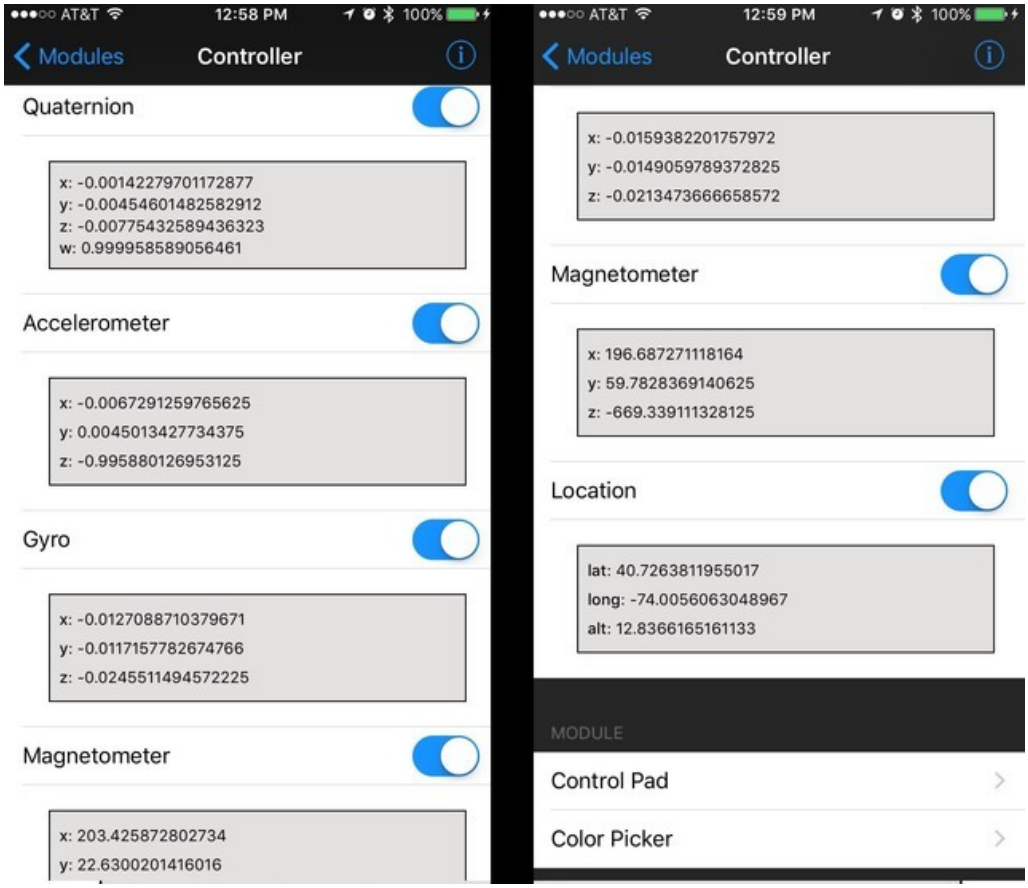
When switched on, AutoScroll will adjust the graph size and follow the most recent data collected by the Bluefruit LE Connect app

When turned off, swipe left and right to scroll through your data. Use the Width slider to adjust the width of the graph.

Controller

The Controller mode provides a variety of ways to control your Bluefruit LE enabled project including Sensor Data, Control Pad, & Color Picker.

This data is sent **from the phone/tablet** to the Bluefruit device.



Format for Sent Data

For an example of how to parse the data on Arduino, check out the BLE_Controller_Test code on github.

- Each Controller data packet sent is **prefixed** with single byte char "**!**" (0x21) followed by a single byte char initial for identification.
- Sensor data values are encoded as **floats of 4 byte length**.
- Each packet ends with a single byte checksum for validation.

Checksum

The single-byte checksum that appends each Controller data packet is calculated by adding all previous bytes of the packet and then inverting the sum.

An example of how to use the checksum to validate a Controller packet can be found in the BLE_Controller_Test Arduino sketch:

```
boolean checkCRC(uint8_t *buffer) {

 uint8_t len = sizeof(buffer);
```

```
  uint8_t crc = buffer[len-2];
  uint8_t sum = 0;

  for (int i = 0; i < (len-1); i++) {
    sum += buffer[i];
  }

  Serial.print("CRC ");

  if ((crc & ~sum) == 0) {
    Serial.println("PASS");
    return true;
  }

  else {
    Serial.println("FAIL");
    return false;
  }

}
```

Sensors

The top section of the Controller table lists the available types of sensor data which can be streamed from your iOS device. Tap the button at the right of each sensor row to begin streaming its relevant data.

- All sensor data updates, except for Location, are sent out over BLE ten times per second.
- Location updates are sent whenever GPS data changes, or every 30 seconds if no change occurs.

• **Quaternion** - sends iOS Device Motion data to describe device attitude. This data is derived from Accelerometer, Gyro, and Magnetometer readings.

Prefix: `!Q`

Format:

`['!'] ['Q'] [float x] [float y] [float z] [float w] [CRC]`

• **Accelerometer** - sends raw accelerometer data.

Prefix: `!A`

Format:

`['!'] ['A'] [float x] [float y] [float z] [CRC]`

• **Gyro** - sends raw gyroscope data.

Prefix: `!G`

Format:

['!'] ['G'] [float x] [float y] [float z] [CRC]

• **Magnetometer** - sends raw, uncalibrated magnetometer data.

Prefix: !M

Format:

['!'] ['M'] [float x] [float y] [float z] [CRC]

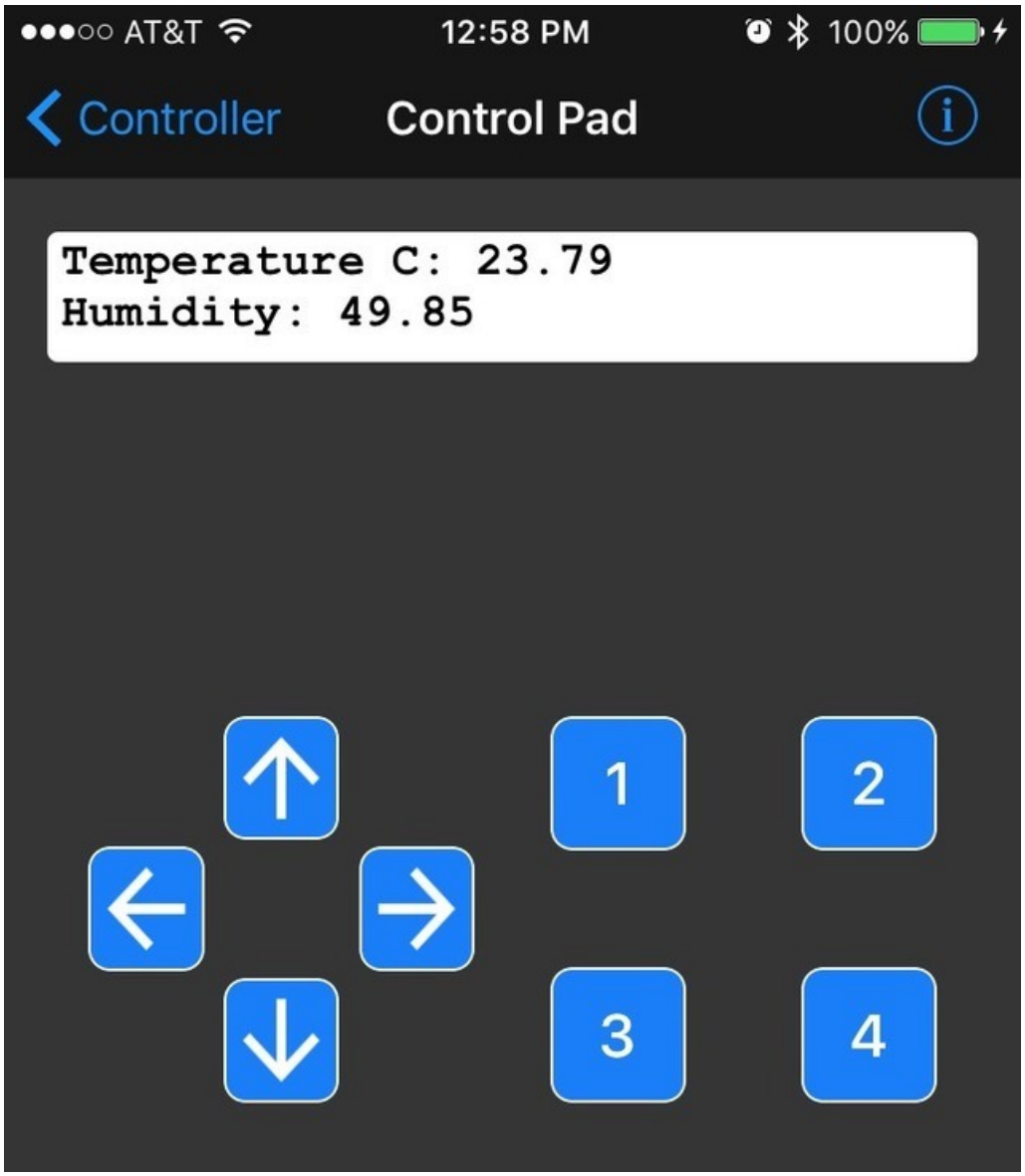
• **Location** - sends GPS data, requires user permission before initial use.

Prefix: !L

Format:

['!'] ['L'] [float lat.] [float long.] [float alt.] [CRC]

Control Pad



The Control Pad function provides a familiar momentary button interface for common control scenarios. Data is sent on the press and release of each button. Each packet consists of 4 bytes, each representing a char value. The first two chars identify the packet as a button message, the third specifies a button, and the fourth signifies either a press or release.

Prefix: `!B`

Examples:

Button 4 pressed: `['!'] ['B'] ['4'] ['1'] [CRC]`

Button 4 released: `['!'] ['B'] ['4'] ['0'] [CRC]`

Button Up pressed: `['!'] ['B'] ['5'] ['1'] [CRC]`

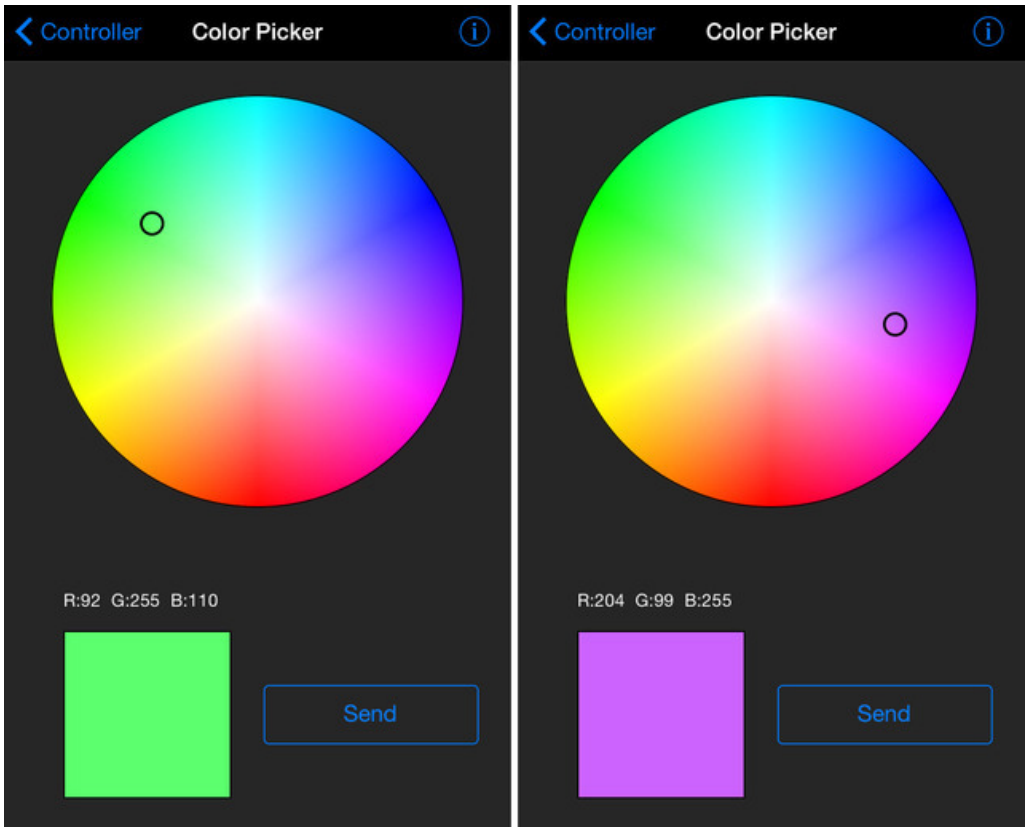
Button Down pressed: `['!'] ['B'] ['6'] ['1'] [CRC]`

Button Left pressed: `['!'] ['B'] ['7'] ['1'] [CRC]`

Button Right pressed: `['!'] ['B'] ['8'] ['1'] [CRC]`

Note: Any activated sensor data streams will continue while using the Control Pad.

Color Picker



The Color Picker sends a color's RGB values to Bluefruit LE. This can be used to control the state of RGB LEDs such as Neopixels.

• Touch the color wheel to choose desired color

• Press Send to send the chosen color's red, green, and blue values to Bluefruit via UART in the following format:

Prefix: `!C`

Format:

`['!'] ['C'] [byte red] [byte green] [byte blue] [CRC]`

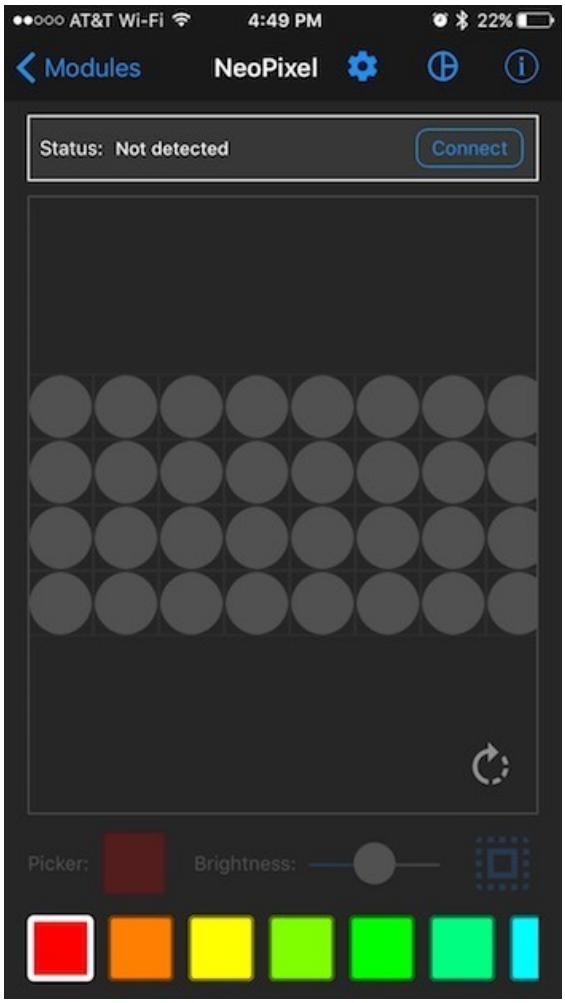
Note: Any activated sensor data streams will continue while using the Color Picker.

Neopixels

The Neopixel mode provides an interface for controlling the color and brightness of neopixels on your Bluefruit LE device.

The default view shows a grid of neopixels.

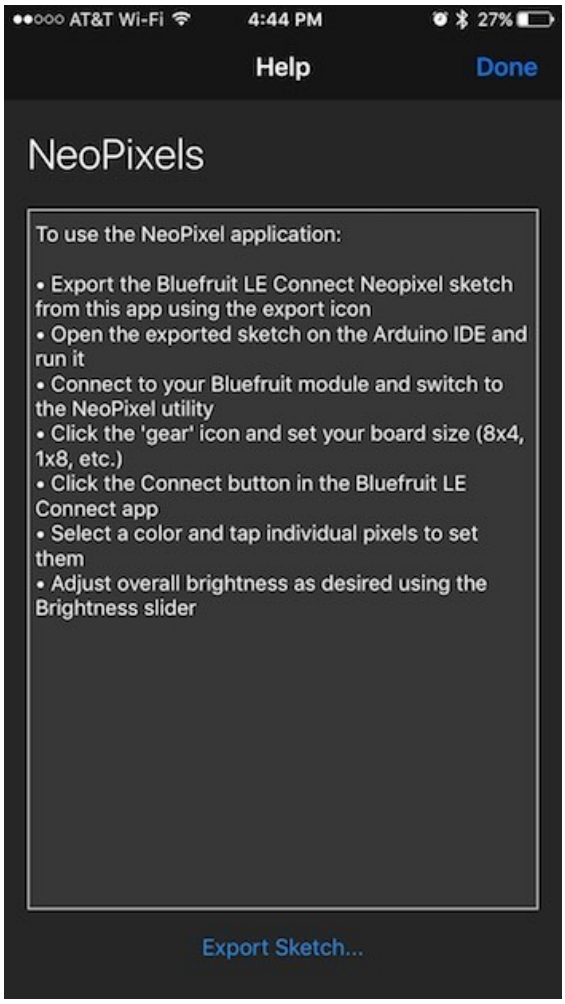
Make sure to check the Status indicator to make sure that the Bluefruit LE Connect app has connected to the neopixels. If not, simply press connect.



Arduino Sketch

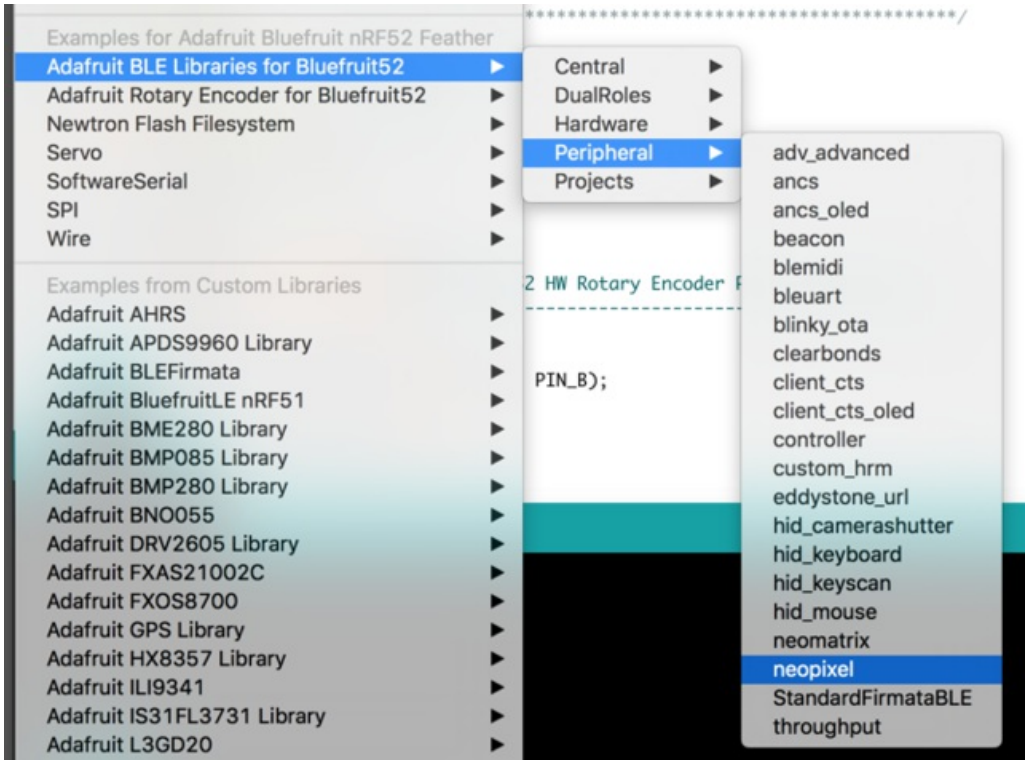
nRF51 Based Bluefruit Modules (Red Bluefruit Modules)

Firstly, make sure to click the info icon in the top right corner of the screen in order to export the Bluefruit LE Connect Neopixel sketch



nRF52 Based Bluefruit Modules (Blue Bluefruit Modules)

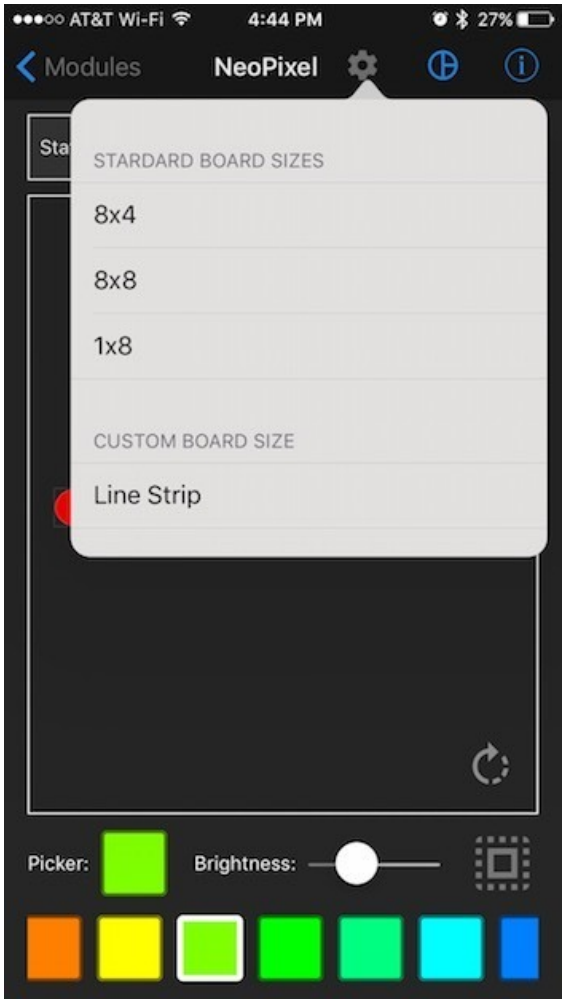
For nRF52 based boards, which have a different API than the red nRF51 models, the NeoPixel sketch is included as part of the nRF52 BSP and can be found at the following location:



You can also browse the nRF52 source code on Github here.

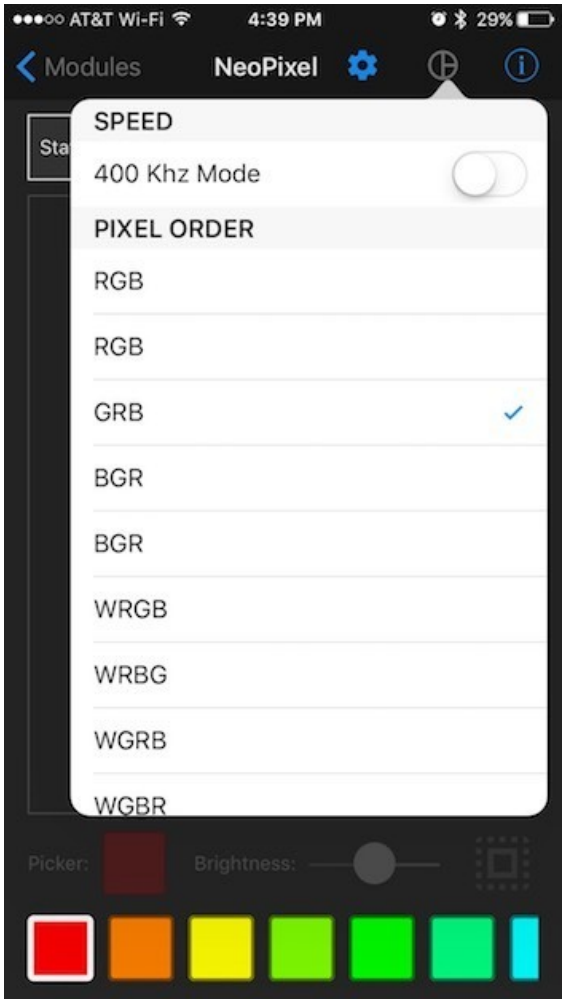
Board Size

Select the gear icon in the upper right side of the screen to view the options for neopixel board sizes. For a NeoPixel Ring, simply choose the Line Strip option and input the number of neopixels in the ring.



Pixel Order

Select the Pixel Order icon to choose between different pixel order combinations such as RGB, GRB, BGR, WEGB and more.



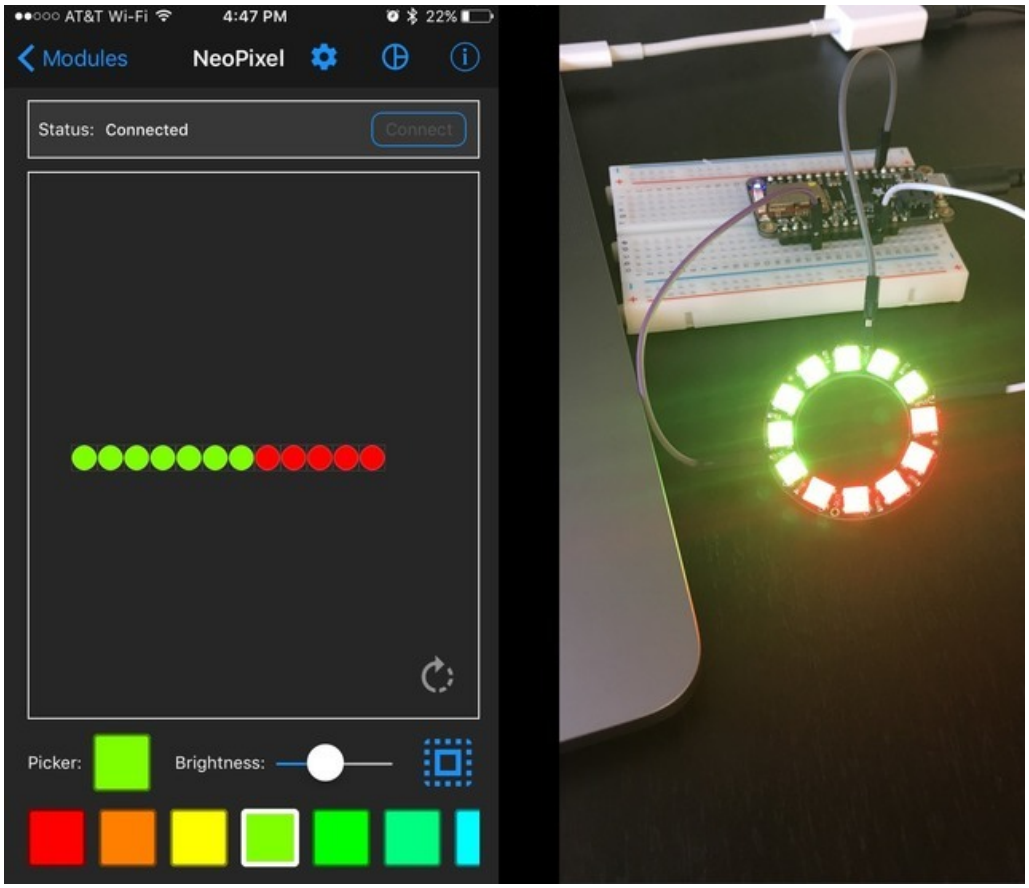
Choose your colors!

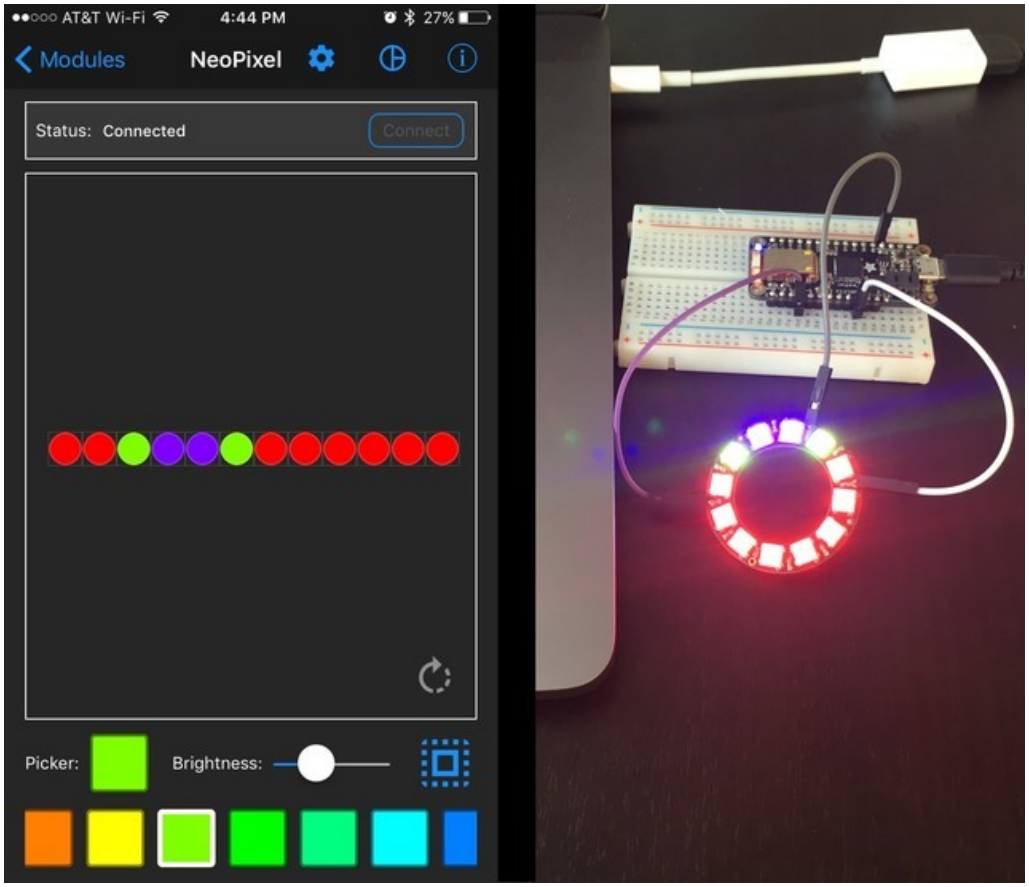
Once you have successfully connected the Bluefruit LE Connect app to your neopixel device, it is time to choose the desired color and brightness for your neopixels.

Select a color from the row on the bottom of the screen or tap the selected color to bring up more color options.

Once the desired color is chosen, tap any of the pixel cells on the screen and watch the corresponding neopixel light up.

Select a color and then select the icon to the right of the brightness slider to set all pixels to the same color.

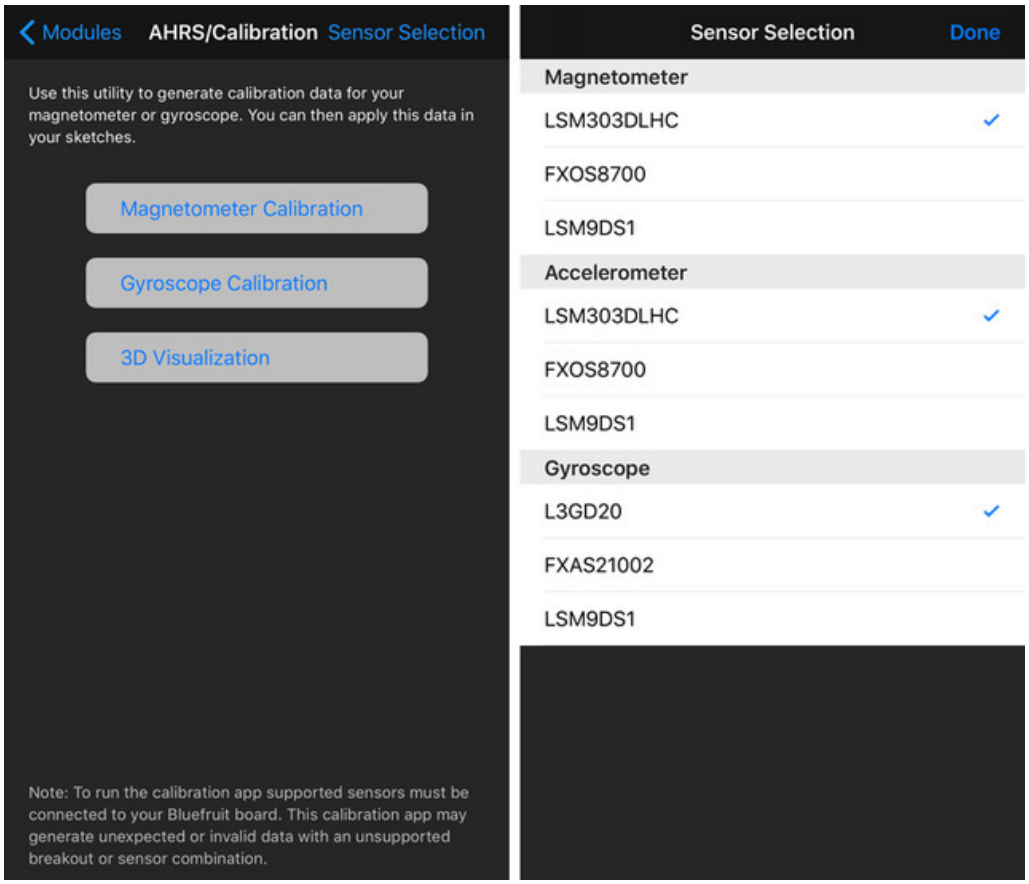

AT&T Wi-Fi
4:47 PM
22%
Modules
NeoPixel
Status: Connected
Connect
Picker:
Brightness:


AT&T Wi-Fi
4:44 PM
27%
Modules
NeoPixel
Status: Connected
Connect
Picker:
Brightness:

AHRS/Calibrarion

This feature is still in development

The AHRS/Calibrarion feature allows you to generate and test calibration code for Magnetometer and Gyroscope sensors connected to your NRF51 based Bluefruit LE device.



Process

Download and run this sketch on your NRF51 powered Bluefruit board:

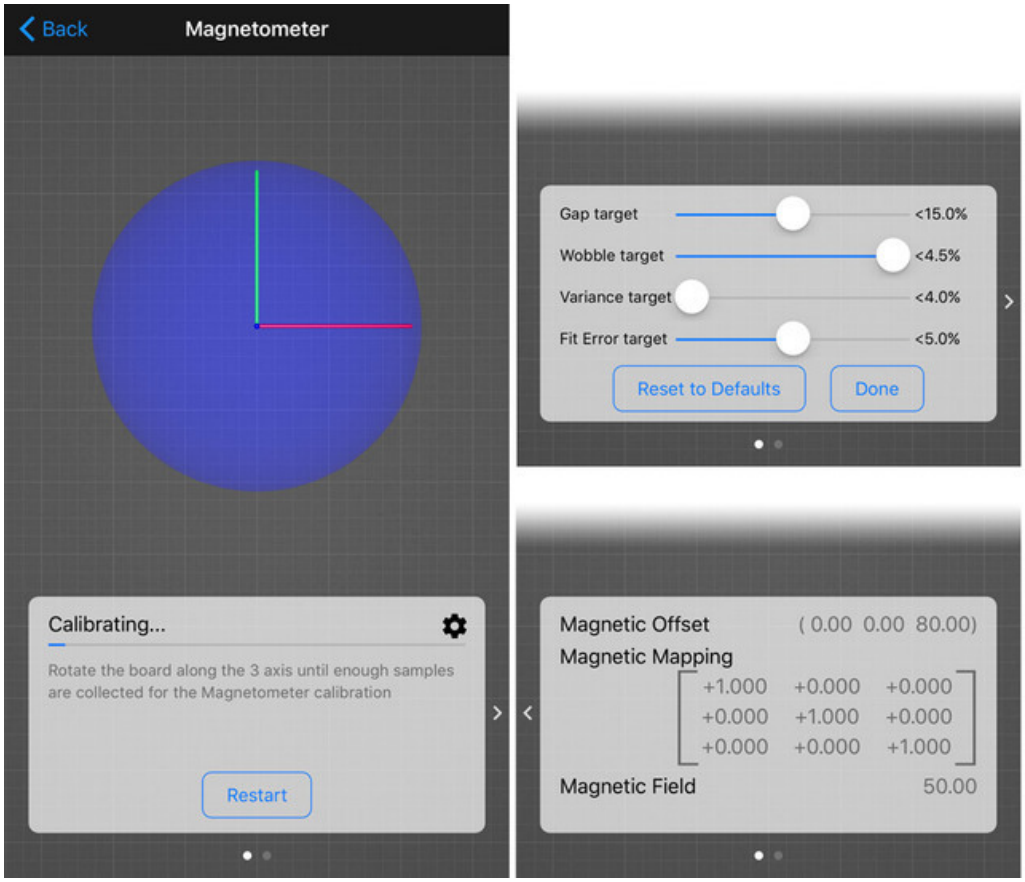
AHRS Calibration sketch for NRF51

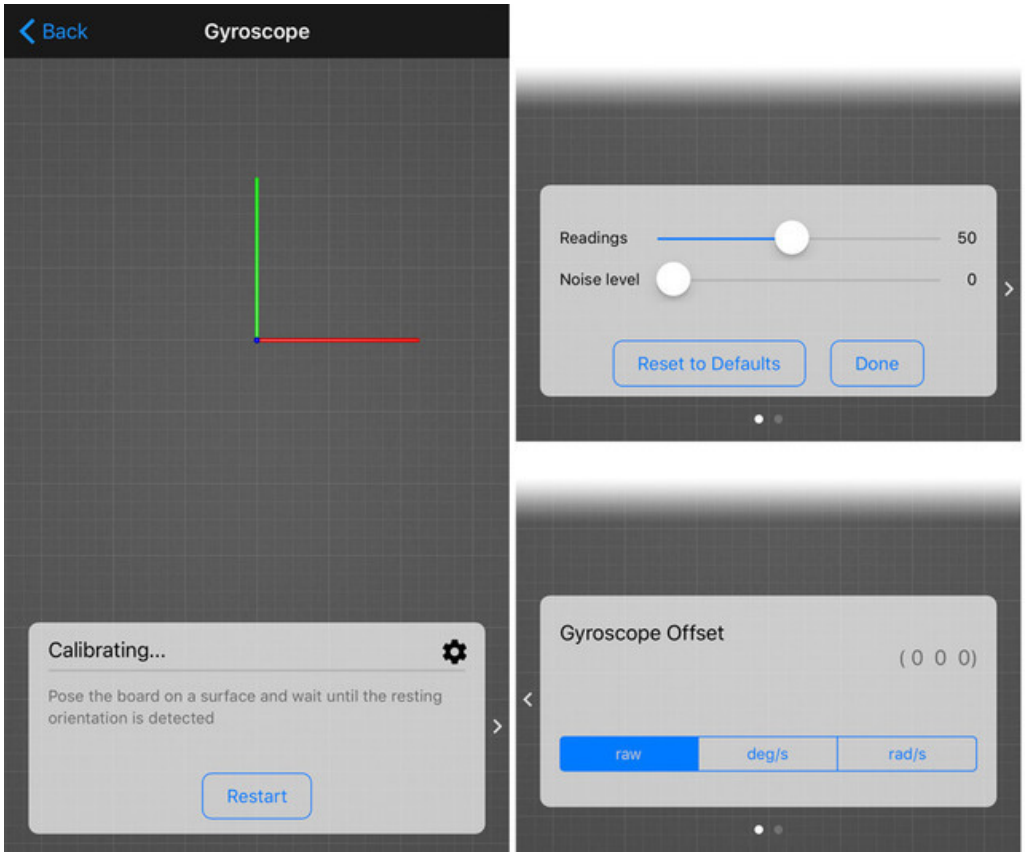
https://adafru.it/A7T

Connect to your Bluefruit board using the calibration app and choose either **Magnetometer** or **Gyroscope** mode depending on what sensor you'll be calibrating.

Keep rotating slowly in a sphere until you get decent coefficients (this can be a bit tricky and takes some trial and error to get right).

Once you have data, note it down as shown in the USB example above.


Back
Magnetometer
Gap target
<15.0%
Wobble target
<4.5%
Variance target
<4.0%
Fit Error target
<5.0%
Reset to Defaults
Done
Calibrating...
Rotate the board along the 3 axis until enough samples are collected for the Magnetometer calibration
Restart
Magnetic Offset
(0.00 0.00 80.00)
Magnetic Mapping
+1.000 +0.000 +0.000
+0.000 +1.000 +0.000
+0.000 +0.000 +1.000
Magnetic Field
50.00


Back
Gyroscope
Readings
50
Noise level
0
Reset to Defaults
Done
Calibrating...
Pose the board on a surface and wait until the resting orientation is detected
Restart
Gyroscope Offset
(0 0 0)
raw
deg/s
rad/s

Next download this sketch, and plug the mag co-efficient values into it:

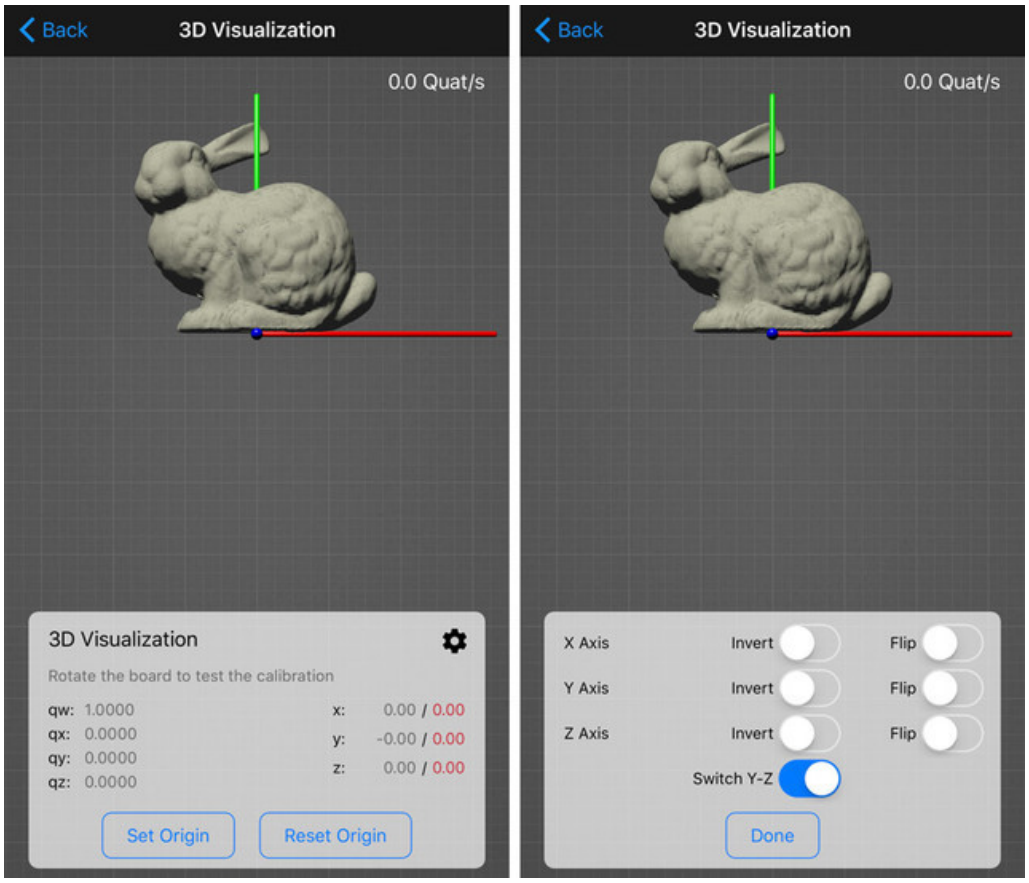
AHRS Fusion BLE NRF51

https://adafru.it/A7U

Once the values are plugged in, upload the sketch to your Bluefruit board.

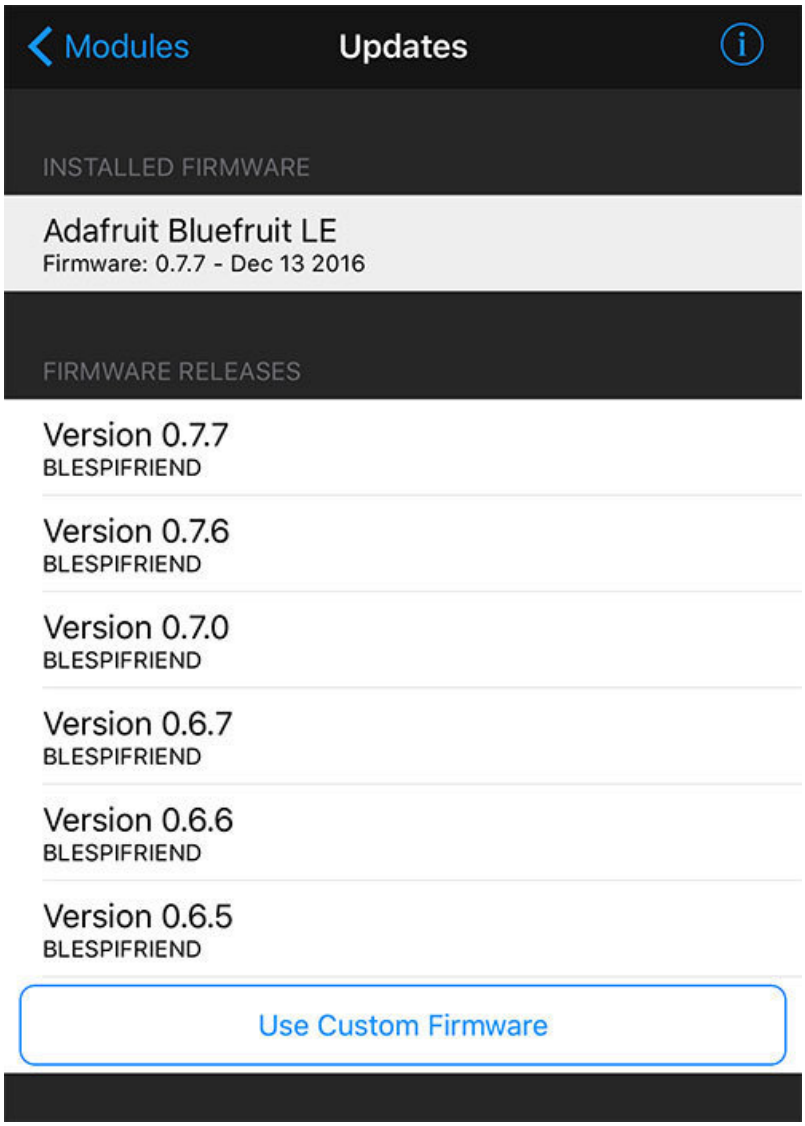
Testing

Connect to your Bluefruit board and run the '3D Visualisation' mode on the app. You will see a 3D model of a bunny. Readings sent from your Bluefruit board should allow you to rotate the model by rotating your board.



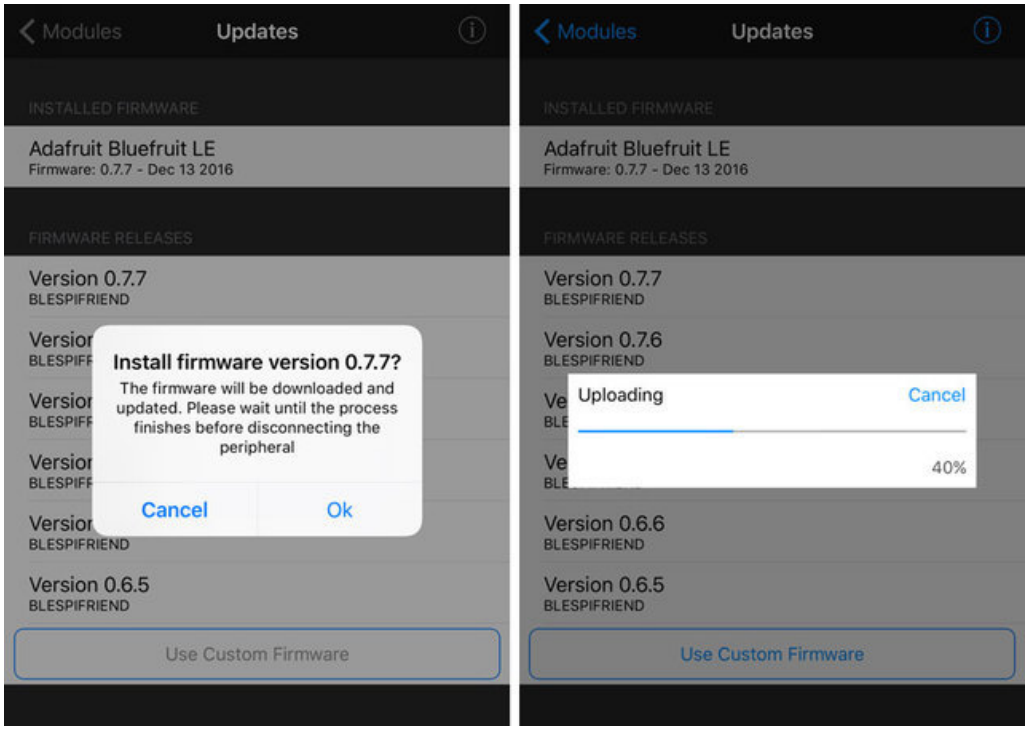
Updates

The Updates features allows you to update the device firmware of your Bluefruit LE hardware over the air. This makes it easy to keep your device firmware up to date.



Updating Firmware

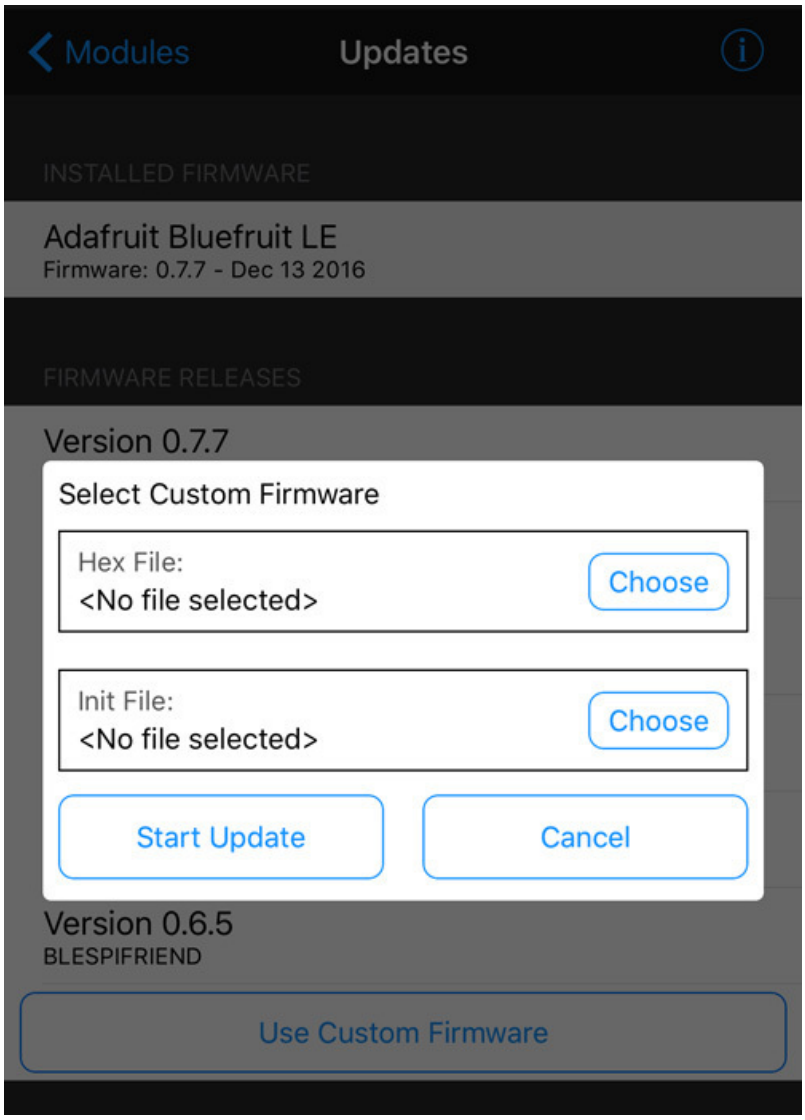
You'll first be presented with a list of compatible firmware versions. Choose the one you want to use (you'll usually want the newest one at the top of the list).



You'll then be asked to confirm the update process. After tapping **Ok**, the process will begin. Once the update process is completed, the app will automatically disconnect from the device and the device will reboot.

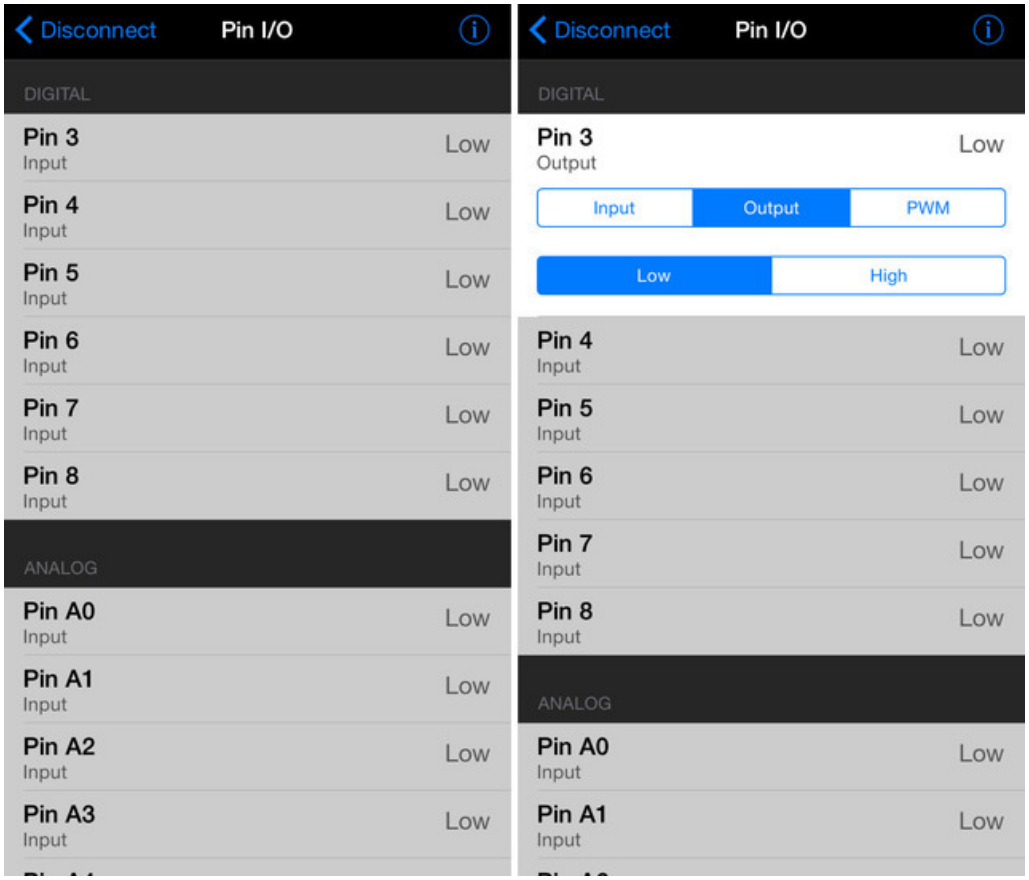
Custom Firmware

You can also update your device with an your own unlisted firmware by tapping the **Use Custom Firmware** button. You'll be asked to locate the relevant file (**iOS** users can access files via **iCloud**) and then update process will begin as above.


Modules
Updates
INSTALLED FIRMWARE
Adafruit Bluefruit LE
Firmware: 0.7.7 - Dec 13 2016
FIRMWARE RELEASES
Version 0.7.7
Select Custom Firmware
Hex File:
<No file selected>
Choose
Init File:
<No file selected>
Choose
Start Update
Cancel
Version 0.6.5
BLESPIFRIEND
Use Custom Firmware

Pin I/O

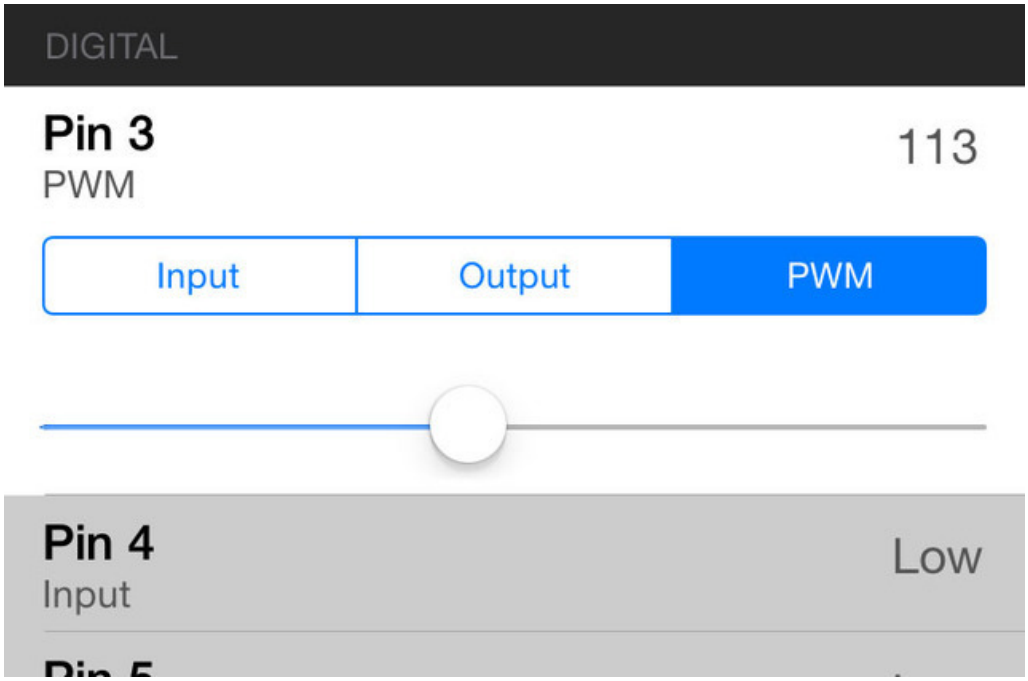
This mode gives you a way to quickly control basic digital inputs and ouputs. You are given a menu of all the pins that are available and you can set the direction and logic level. For some pins, you may also have PWM output & analog inputs! You can do quite a bit just with this method of control



• Each row in the table represents a pin on your Arduino. Pin name and current state are displayed on the left side of the cell, while pin value is displayed on the right.

• Tap a row to change the relevant pin's current mode and value.

• Tap the row a second time to hide its controls



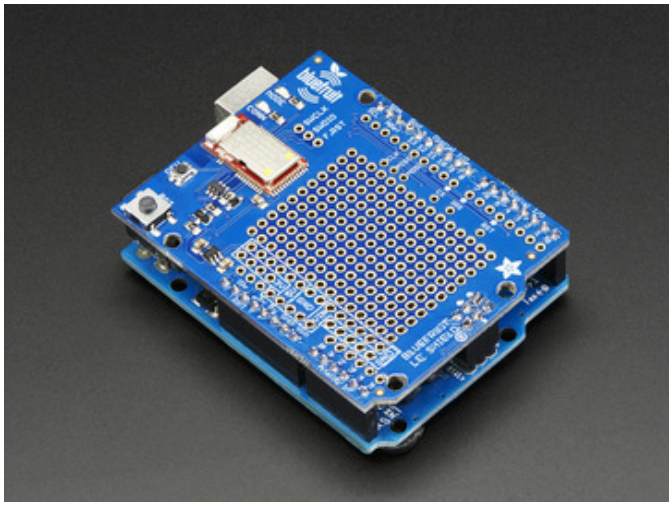
Read through the Wiring, Configuration and Usage pages for more details

Wiring Options

The Pin I/O mode lets you control an Arduino or compatible which is connected to a Bluefruit LE module.

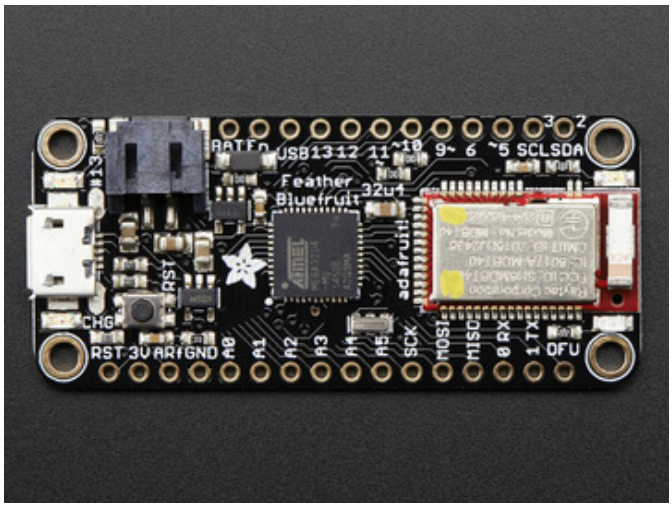
You can use the **nrf8001**-based or **nrf51822**-based breakouts. However, the setup will vary slightly.

Let's start with *wiring options*



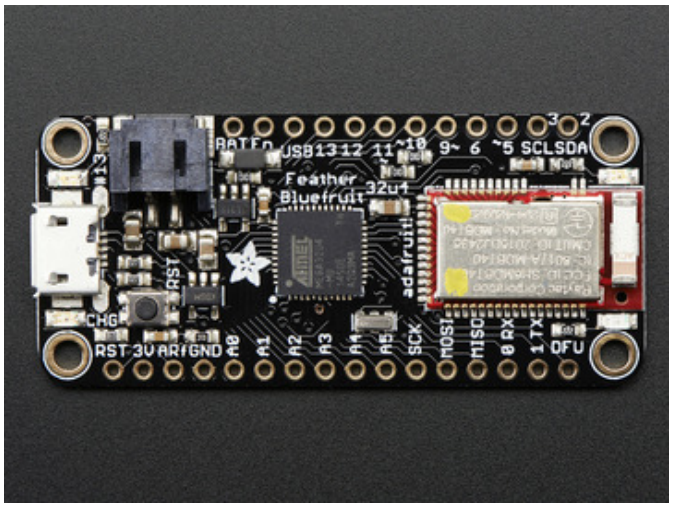
Arduino with Bluefruit LE Shield

If you are using the Bluefruit LE Shield then you have an **SPI-connected NRF51822** module. You can use this with **Atmega328** (Arduino UNO or compatible)**, ATmega32u4** (Arduino Leonardo, compatible) or **ATSAMD21** (Arduino Zero) Your pinouts are **Hardware SPI, CS** = **8, IRQ** = **7, RST** = **4**



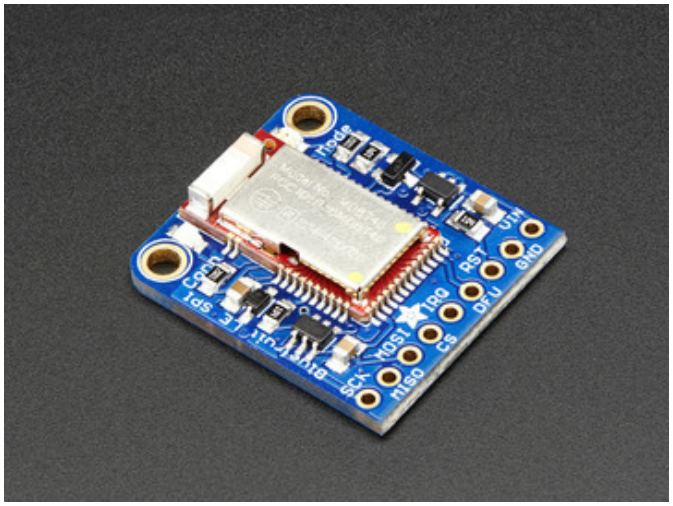
Bluefruit Micro or Feather 32u4 Bluefruit

If you have a Bluefruit Micro or Feather 32u4 Bluefruit LE then you have an **ATmega32u4** chip with **Hardware SPI**, CS = **8**, IRQ = **7**, RST = **4**



Feather M0 Bluefruit LE

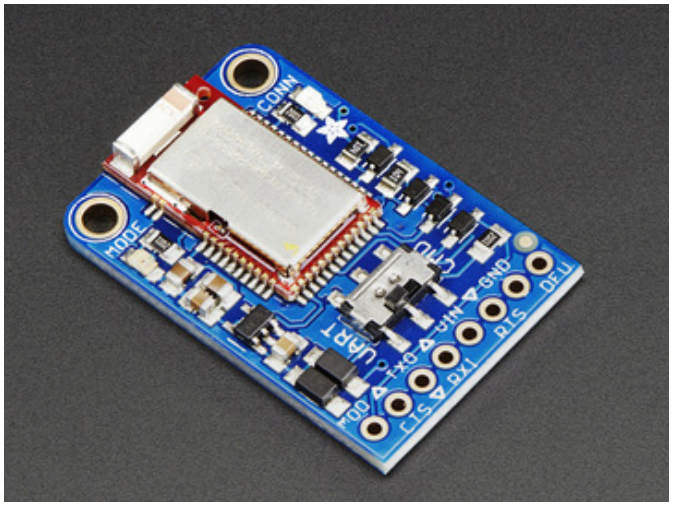
If you have a Feather M0 Bluefruit LE then you have an **ATSAMD21** chip with **Hardware SPI**, CS = **8**, IRQ = **7**, RST = **4**



Bluefruit LE SPI Friend

If you have a stand-alone module, you have a bit of flexibility with wiring however we strongly recommend **Hardware SPI**, CS = **8**, IRQ = **7**, RST = **4**

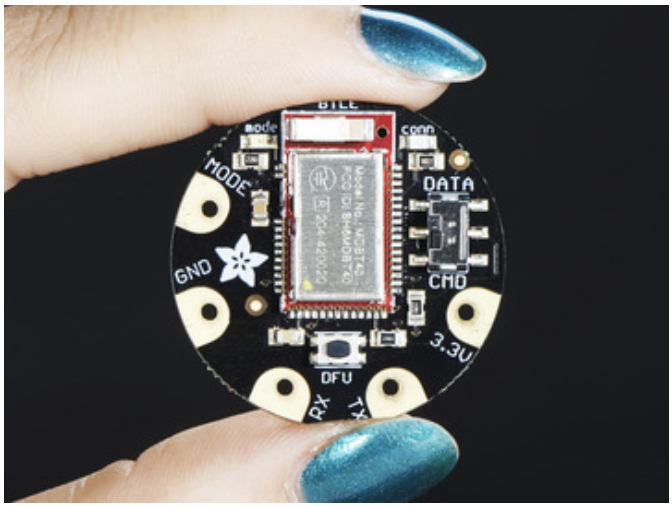
You can use this with **Atmega328** (Arduino UNO or compatible)**, ATmega32u4** (Flora, Arduino Leonardo, compatible) or **ATSAMD21** (Arduino Zero) Your default pinouts should be **Hardware SPI, CS** = **8, IRQ** = **7, RST** = **4**



Bluefruit LE UART Friend

If you have a stand-alone UART module you have some flexibility with wiring. However we suggest hardware UART if possible. You will definitely need to use the flow control CTS pin if you are not using hardware UART. You will need to set up the MODE pin as well since Firmata uses both command and data modes

You can use this with **Atmega328** (Arduino UNO or compatible)**, ATmega32u4** (Arduino Leonardo, compatible) or **ATSAMD21** (Arduino Zero



Flora BLE

This simplified UART friend is BLE for FLORA, its really only intended for use with Hardware Serial, there's no flow control.

You can use this **ATmega32u4** (Flora) although in theory you could wire it up to a different processor

Library and Config

Before loading Firmata BLE...

Make sure you have the basic tutorials working, that you can send/receive data via the UART with your wiring. If you can't talk to the module and send/receive data via the basic UART demo, Firmata won't work!

Install Libraries

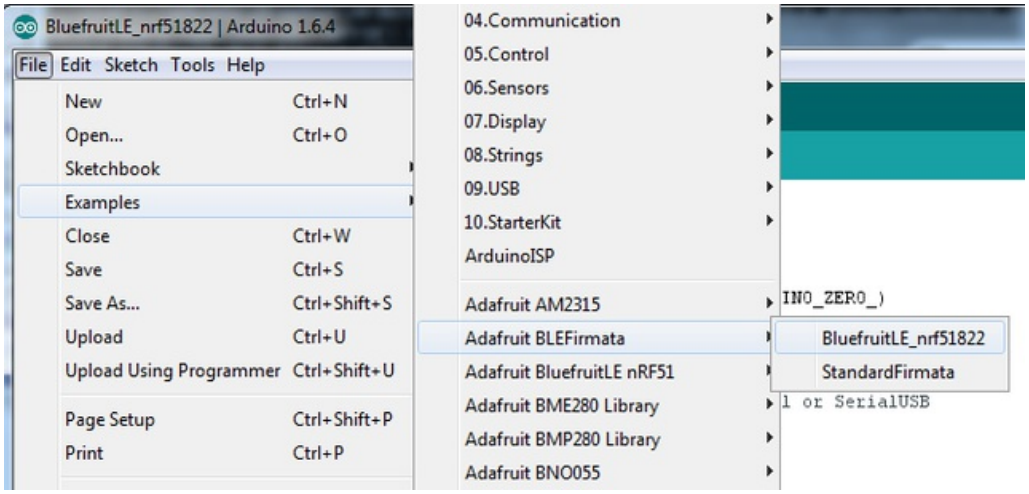
You'll want to start by installing the BLE Firmata library from github. You can grab the latest zip by clicking below. You'll also of course need the library you use to talk to the hardware module

Download Adafruit_BLE_PinIO

https://adafru.it/fTP

Open Sketch and Configure

Restart the IDE and load the **Adafruit_BLE_PinIO->BluefruitLE_nrf51822** sketch

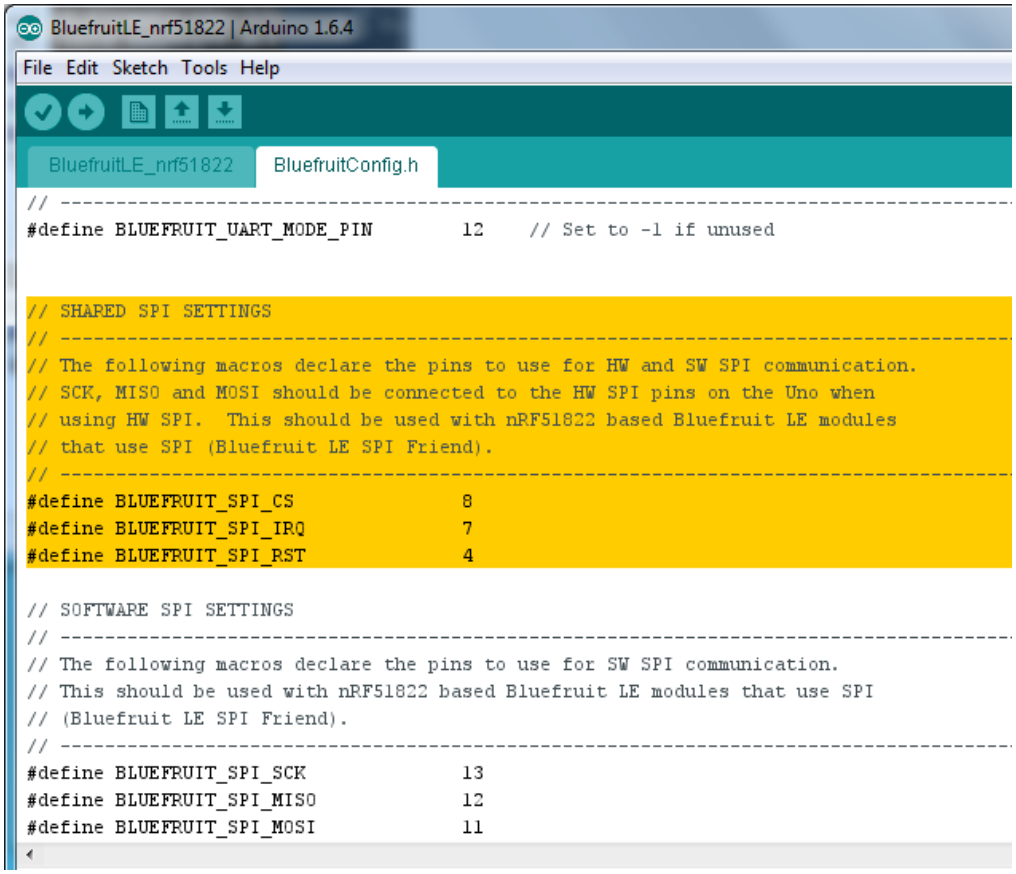


At the top of the sketch is the Firmata configuration section, there's also **Bluefruit LE pin configuration** in the BluefruitConfig.h tab

Bluefruit LE Config

Start by opening the BluefruitConfig.h tab. By default the sketch is set up for **hardware SPI** and **CS** = 8, **IRQ** = 7 and **RST** = 4. You'll need to change your pins if you are using UART or different SPI pins.

Check out the generic "Configuration!" details for information on what every pin does. If you're using software serial, be sure to set up flow control and a MODE pin.



Then in the main sketch, if you're not using hardware SPI, uncomment the connection style you want

```
// Create the bluefruit object, either software serial...uncomment these lines
/*
SoftwareSerial bluefruitSS = SoftwareSerial(BLUEFRUIT_SWUART_TXD_PIN, BLUEFRUIT_SWUART_RXD_PIN);

Adafruit_BluefruitLE_UART ble(bluefruitSS, BLUEFRUIT_UART_MODE_PIN,
                      BLUEFRUIT_UART_CTS_PIN, BLUEFRUIT_UART_RTS_PIN);
*/

/* ...or hardware serial, which does not need the RTS/CTS pins. Uncomment this line */
// Adafruit_BluefruitLE_UART ble(BLUEFRUIT_HWSERIAL_NAME, BLUEFRUIT_UART_MODE_PIN);

/* ...hardware SPI, using SCK/MOSI/MISO hardware SPI pins and then user selected CS/IRQ/RST */
Adafruit_BluefruitLE_SPI ble(BLUEFRUIT_SPI_CS, BLUEFRUIT_SPI_IRQ, BLUEFRUIT_SPI_RST);

/* ...software SPI, using SCK/MOSI/MISO user-defined SPI pins and then user selected CS/IRQ/RST */
//Adafruit_BluefruitLE_SPI ble(BLUEFRUIT_SPI_SCK, BLUEFRUIT_SPI_MISO,
//                             BLUEFRUIT_SPI_MOSI, BLUEFRUIT_SPI_CS,
//                             BLUEFRUIT_SPI_IRQ, BLUEFRUIT_SPI_RST);
```

Firmata Debug Config

Next up in the main sketch there's a few settings you'll have to configure. First up is debug and serial details

```
// Change this to whatever is the Serial console you want, either Serial or SerialUSB
#define FIRMATADEBUG    Serial
// Pause for Serial console before beginning?
#define WAITFORSERIAL   true
// Print all BLE interactions?
#define VERBOSE_MODE    false
```

The first setting **FIRMATADEBUG** is how output is printed. 99% of the time you'll be happy with Serial, but for Arduino Zeros you may need to use SerialUSB for the native port.

WAITFORSERIAL determines whether the sketch waits for the Serial port to be opened before it runs. Set it to true while debugging/testing and open up the serial console to kick off the sketch. Set to false once it's working great

VERBOSE_MODE allows you to see all of the data passing between the BLE module and App.

Available Pins Config

Below the debug config is where you can set up what pins are available for the Pin IO app to twiddle

```
/************** For Bluefruit Micro or Feather 32u4 Bluefruit ************/
//uint8_t boards_digitaliopins[] = {0,1,2,3,5,6,9,10,11,12,13,A0,A1,A2,A3,A4,A5};

/************** For UNO + nRF58122 SPI & shield ************/
//uint8_t boards_digitaliopins[] = {2, 3, 5, 6, 9, 10, A0, A1, A2, A3, A4, A5};

/************** For Bluefruit M0 Bluefruit ************/
//uint8_t boards_digitaliopins[] = {0,1,5,6,9,10,11,12,13,20,21,A0,A1,A2,A3,A4,A5};
```

You'll need to uncomment *one* of these lines. Also, they are set up by default for the SPI Bluefruit module, on pins 4, 7, 8. For that reason, the hardware SPI pins and 4,7,8 don't appear. If you are using different pins you can re-add those to the list. If there are any other pins you don't want to show up in the app, remove those pins as well.

For the 32u4 and M0 examples, there's a lot of pins, & perhaps not all of them are necessary! You can have as few as you like.

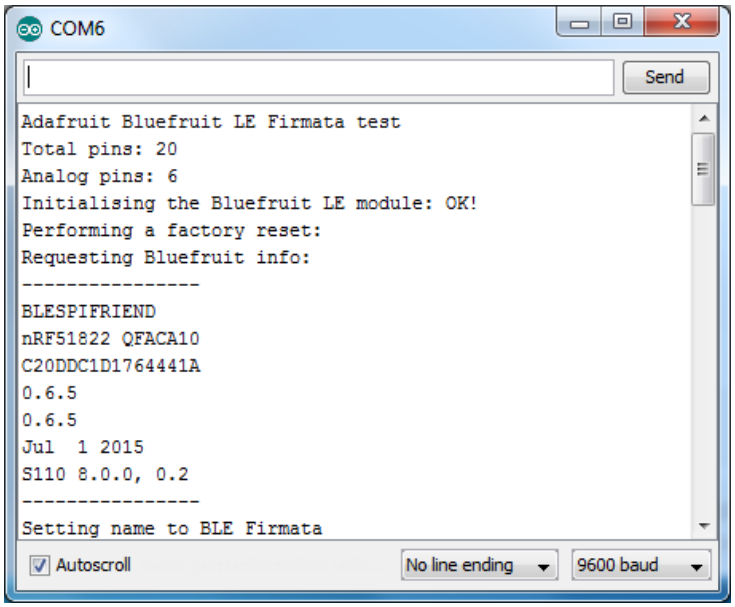
Below the setup lines you can see the way we tell Firmata which pins do what

```
#if defined(__AVR_ATmega328P__)
  // Standard setup for UNO, no need to tweak
  uint8_t boards_analogiopins[] = {A0, A1, A2, A3, A4, A5};  // A0 == digital 14, etc
  uint8_t boards_pwmpins[] = {3, 5, 6, 9, 10, 11};
  uint8_t boards_servopins[] = {9, 10};
  uint8_t boards_i2cpins[] = {SDA, SCL};
```

Don't mess with these! Change only the digital IO pins array!

Upload and test

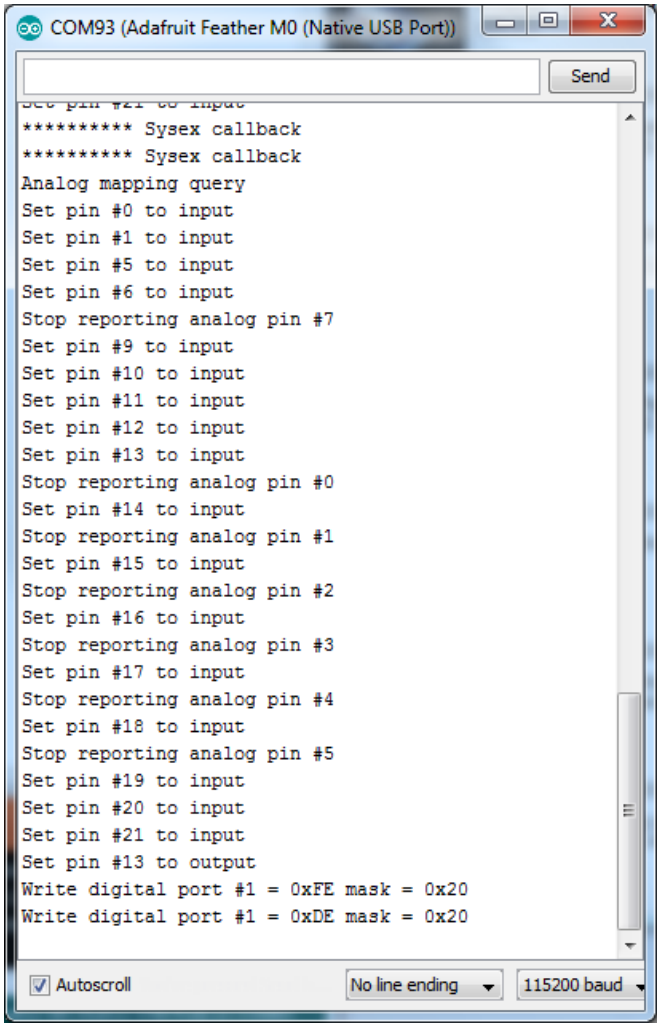
Once you're done compile and upload. Open up the serial console. You should see that the sketch was able to initialize the Bluefruit LE module, reset it and print out some details about the BLE firmware. It will now wait for the app to connect



If you dont see anything, check:

- If you're using an ATSAMD21/M0 chip, do you have FIRMATADEBUG set to SerialUSB?
- Do you have WAITFORSERIAL true?

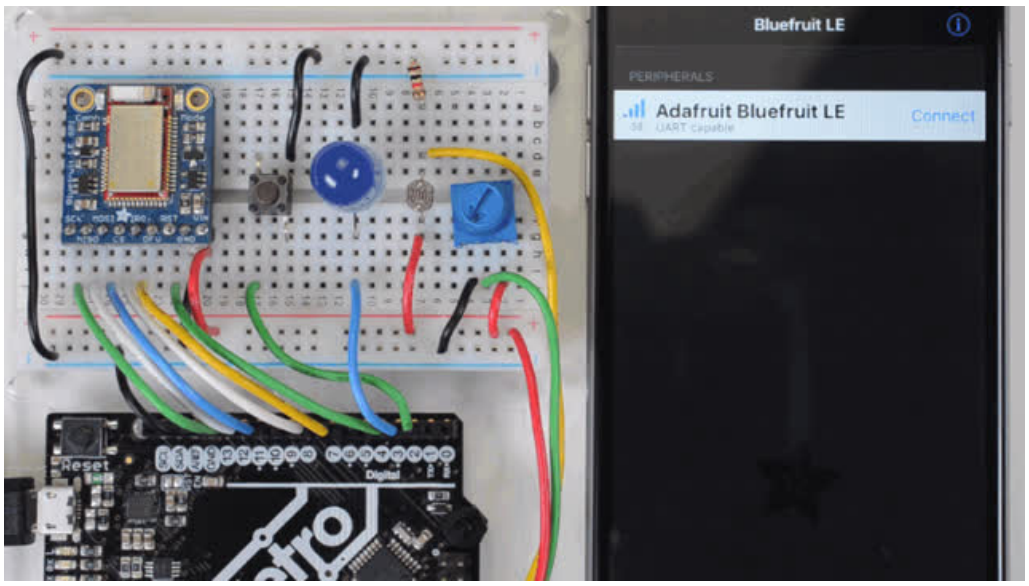
Once you connect in Pin IO mode you can see a stream of commands that are received and acted upon

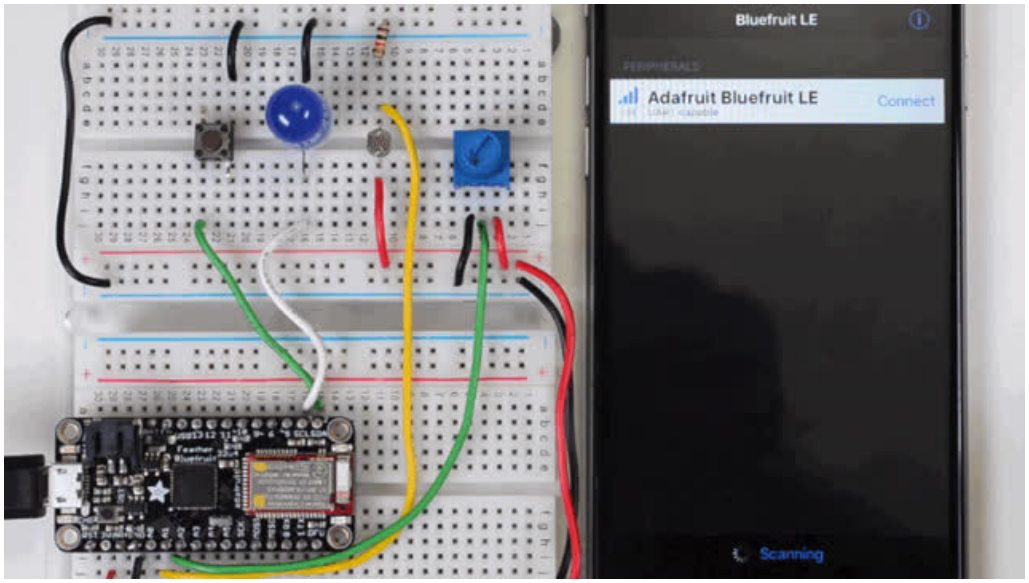

COM93 (Adafruit Feather M0 (Native USB Port))
Send
********** Sysex callback
********** Sysex callback
Analog mapping query
Set pin #0 to input
Set pin #1 to input
Set pin #5 to input
Set pin #6 to input
Stop reporting analog pin #7
Set pin #9 to input
Set pin #10 to input
Set pin #11 to input
Set pin #12 to input
Set pin #13 to input
Stop reporting analog pin #0
Set pin #14 to input
Stop reporting analog pin #1
Set pin #15 to input
Stop reporting analog pin #2
Set pin #16 to input
Stop reporting analog pin #3
Set pin #17 to input
Stop reporting analog pin #4
Set pin #18 to input
Stop reporting analog pin #5
Set pin #19 to input
Set pin #20 to input
Set pin #21 to input
Set pin #13 to output
Write digital port #1 = 0xFE mask = 0x20
Write digital port #1 = 0xDE mask = 0x20
Autoscroll
No line ending
115200 baud

Usage

The Pin IO capability basically lets you control the pins of the Arduino one by one. Its meant for basic prototyping and control where you may not want to write a full app from scratch!

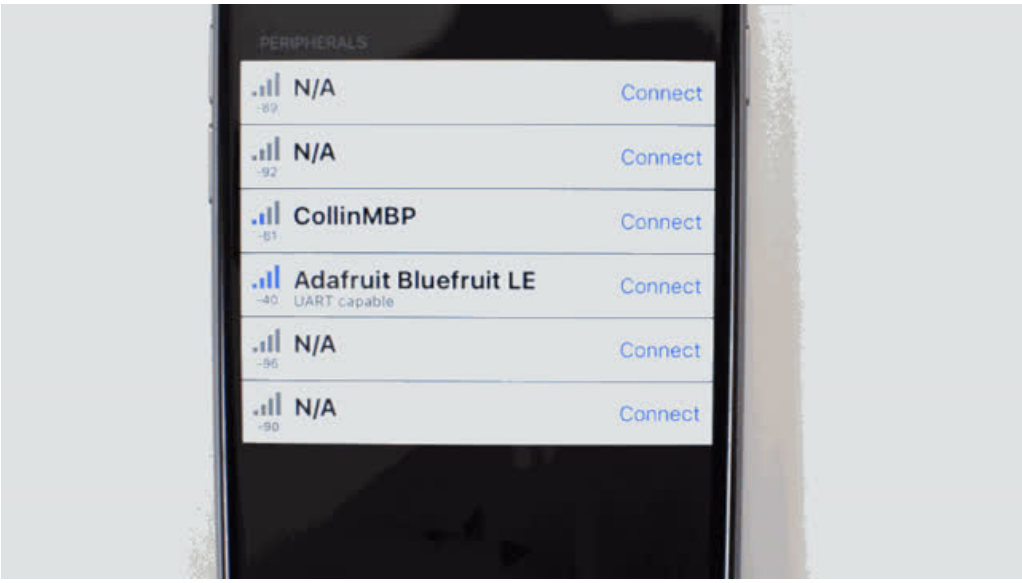
The app and sketch are also 'smart' in that when you connect, the app will query the Arduino what pins are available and what they can do!



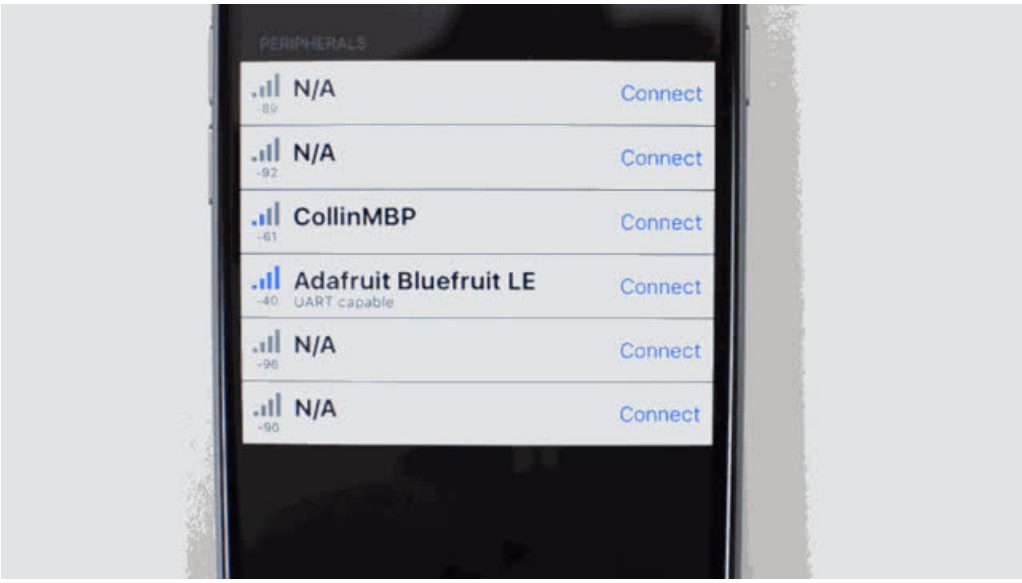


Initial Query

Make sure you have the most recent version of the app, and the correct configuration. During connection you'll see the app **Querying Pin Capabilities...** It will then get the correct details and fill out the pin map

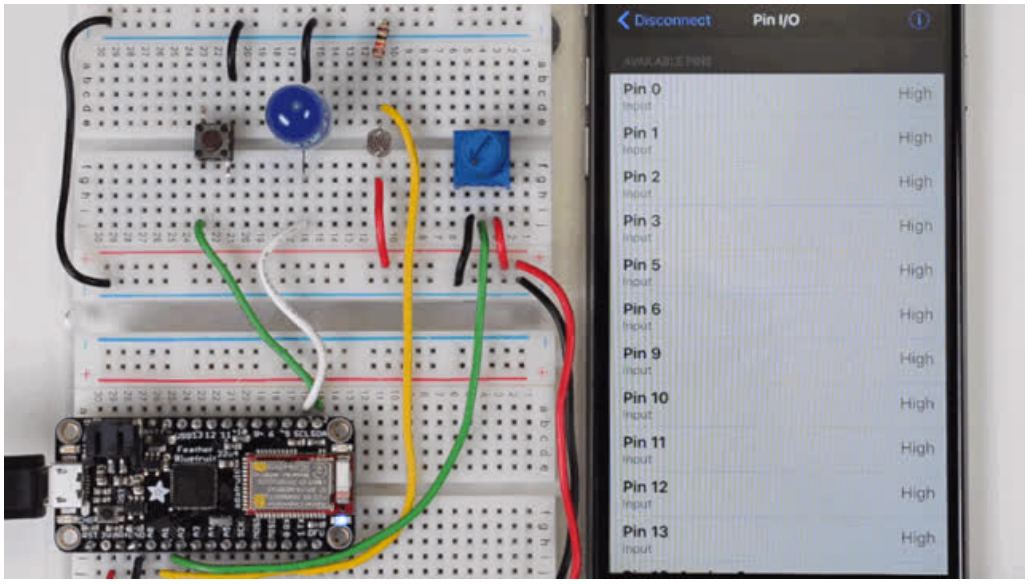


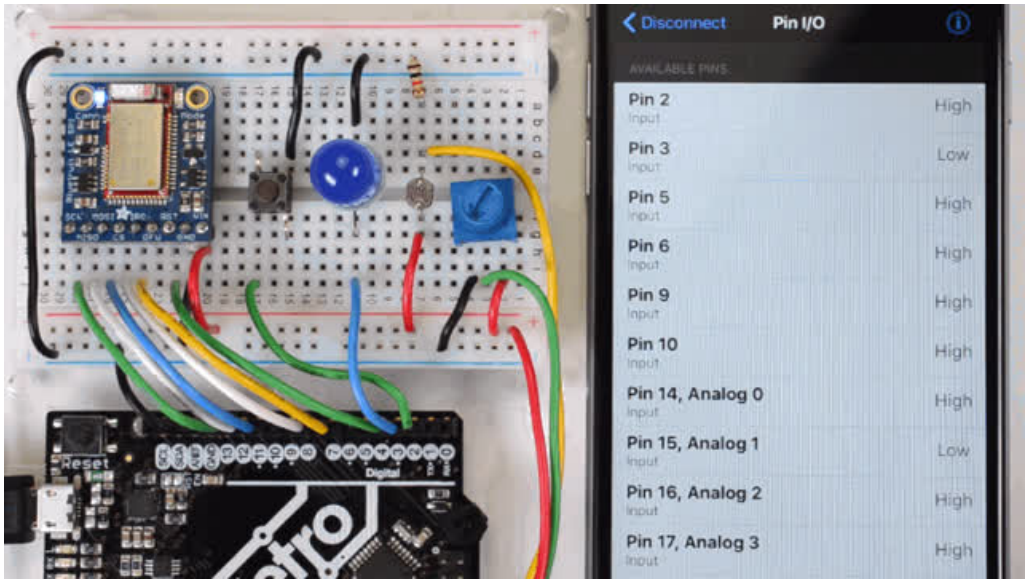
If the query fails, the app will default back to our old 'UNO' setup, which may not work well for you. Check debug output for the Bluefruit firmata sketch, make sure you're running the latest version of the sketch



Digital Input

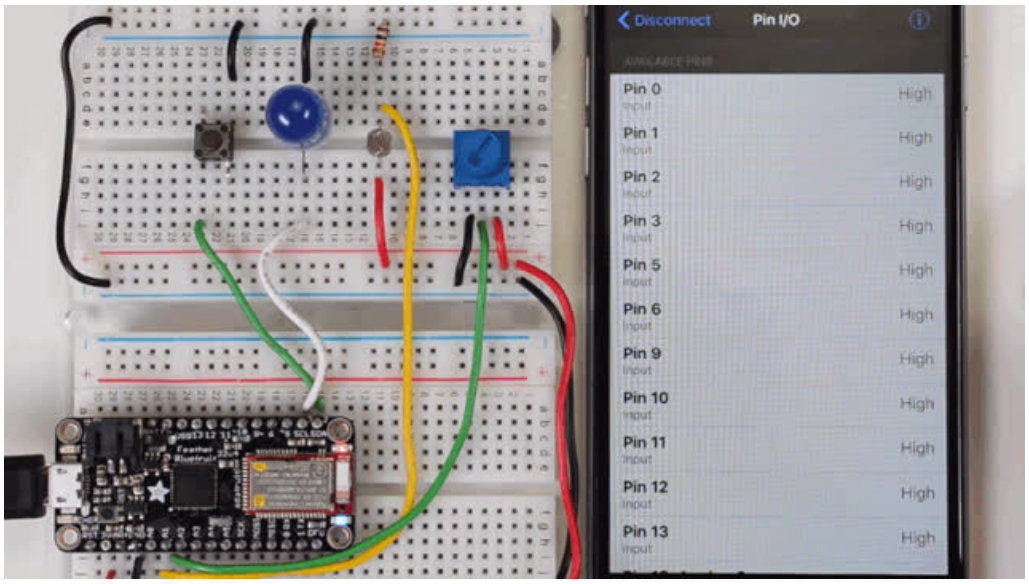
All of the pins default to digital inputs with built-in pullup resistors. This means that by default the pins read **HIGH**. When a button is wired to the pin & pressed or the pin is shorted to ground, you will receive a **LOW** signal

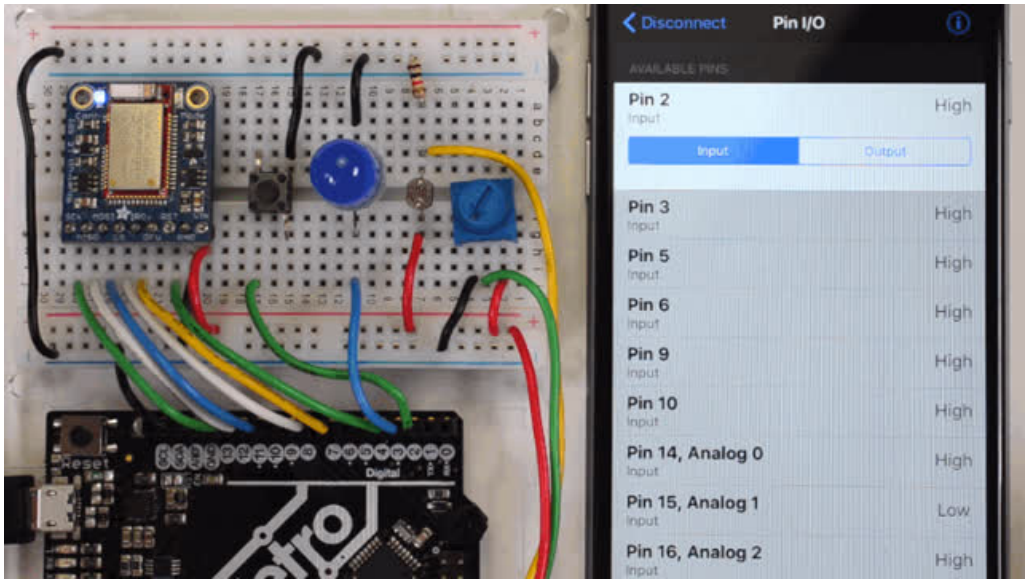




Digital Output

You can also set the pins to digital outputs. This will let you set the pins HIGH (3V or 5V, depending on the microcontroller voltage) or LOW (0V a.k.a ground). Great for turning on & off LEDs

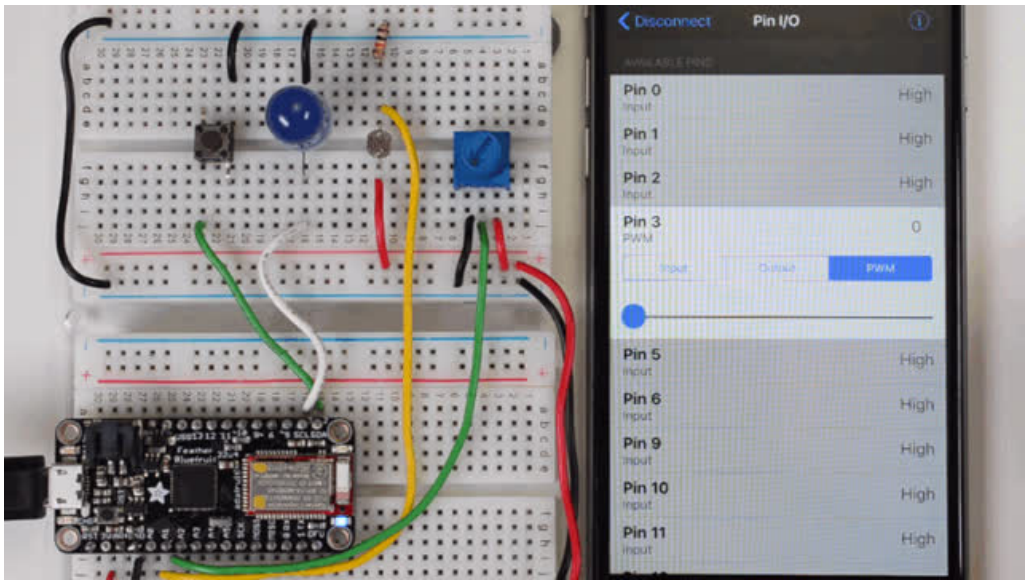


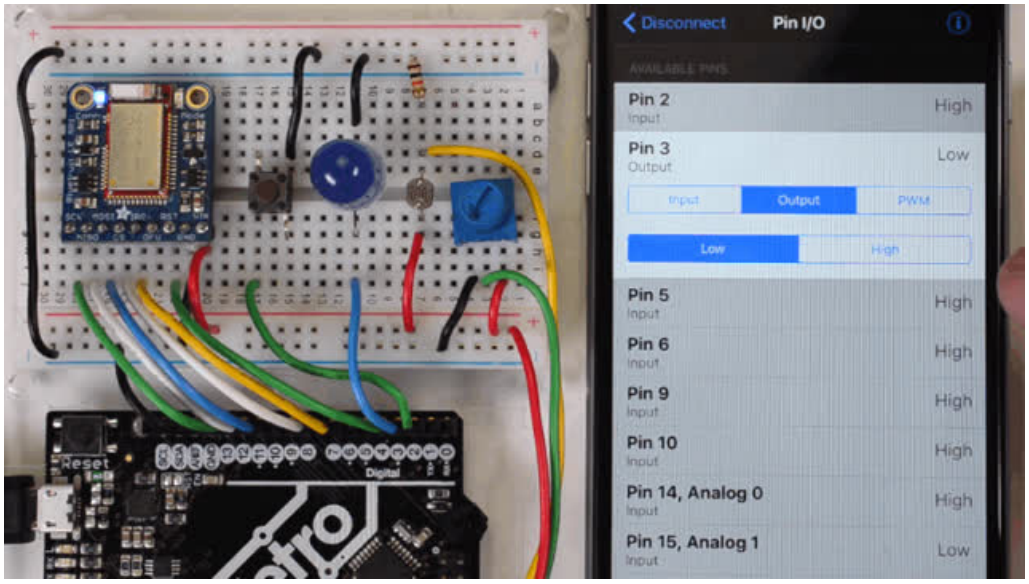


You can also wire it up to a PowerSwitch Tail which gives you a safe way to control appliances!

PWM Output

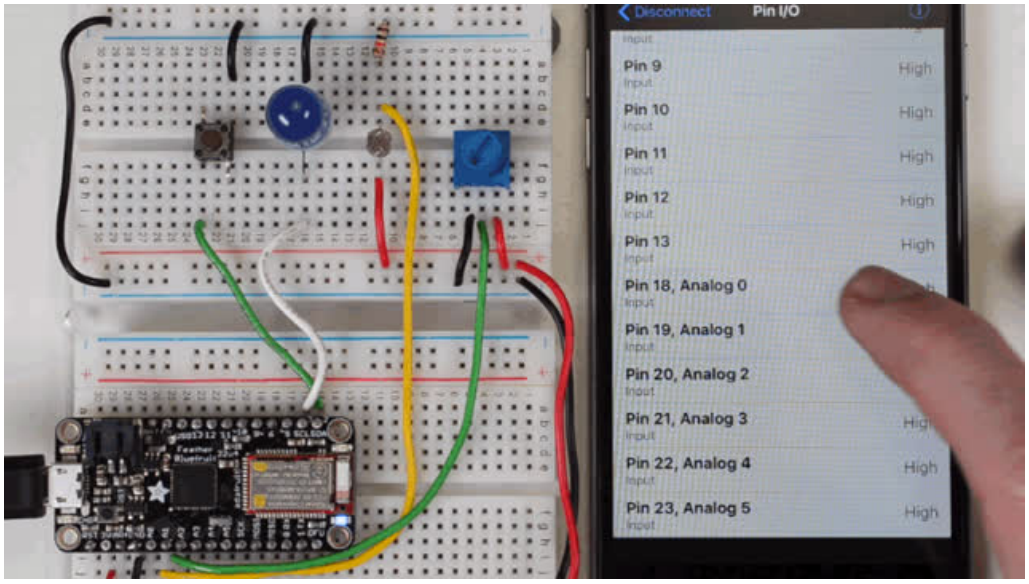
On some pins, there's PWM output available, you can dim or brighten an LED. You could also control a DC motor through a driver, if it can be PWM controlled

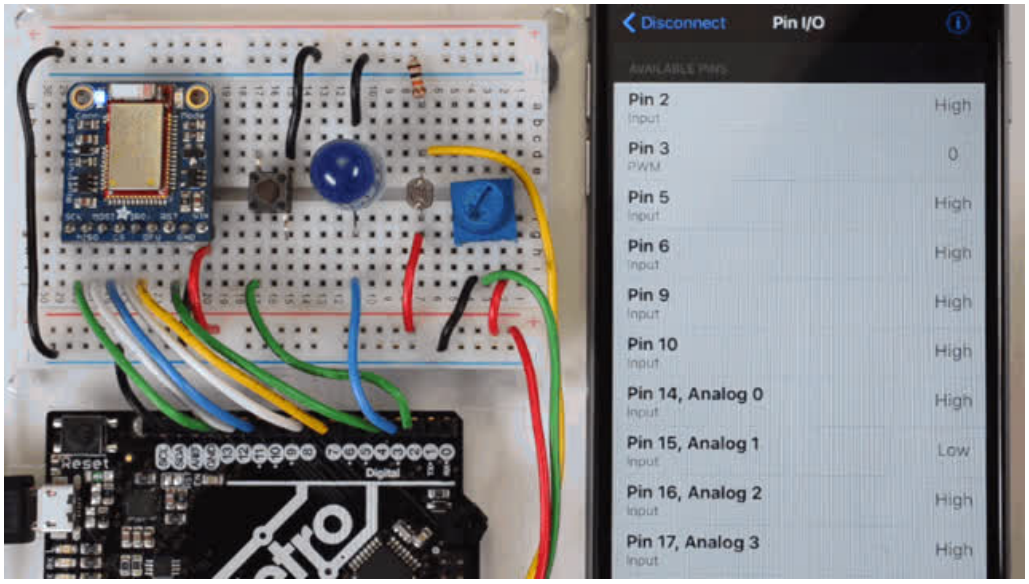




Analog Input

There are also some pins that have Analog pin capability. You can read various sensors or potentiometers that are wired to these pins. Its best to make sure you know how to get the analog sensor working first, using plain Arduino code, before wiring it up for use with Pin IO




Disconnect
Pin I/O
Pin 2
Input
High
Pin 3
PWM
0
Pin 5
Input
High
Pin 6
Input
High
Pin 9
Input
High
Pin 10
Input
High
Pin 14, Analog 0
Input
High
Pin 15, Analog 1
Input
Low
Pin 16, Analog 2
Input
High
Pin 17, Analog 3
Input
High
Reset
Digital