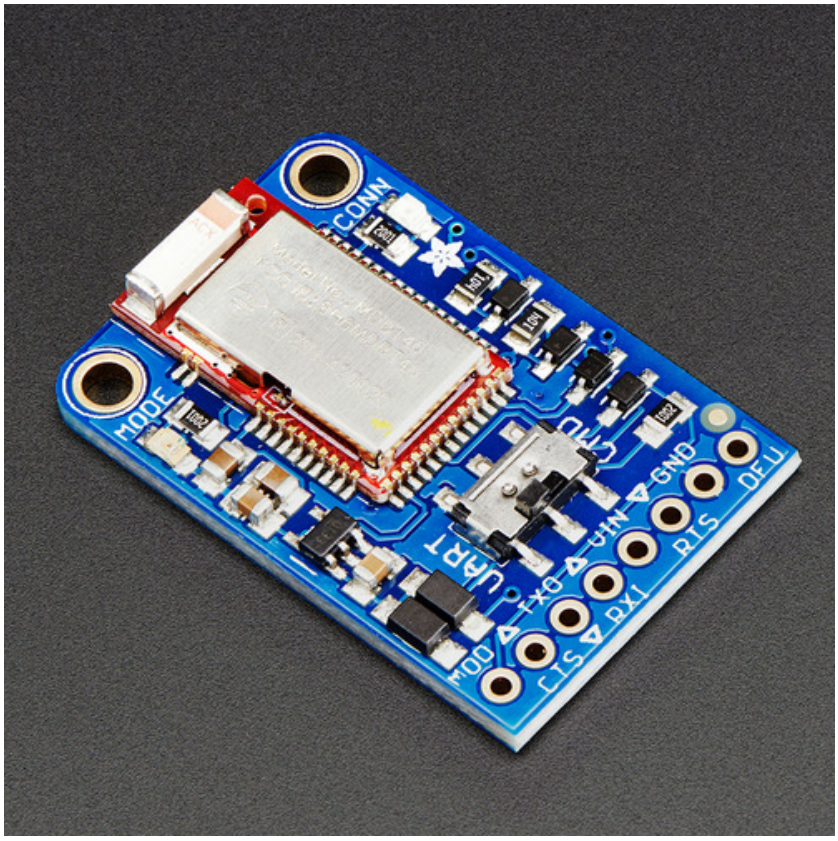
adafruit learning system

Introducing the Adafruit Bluefruit LE UART Friend

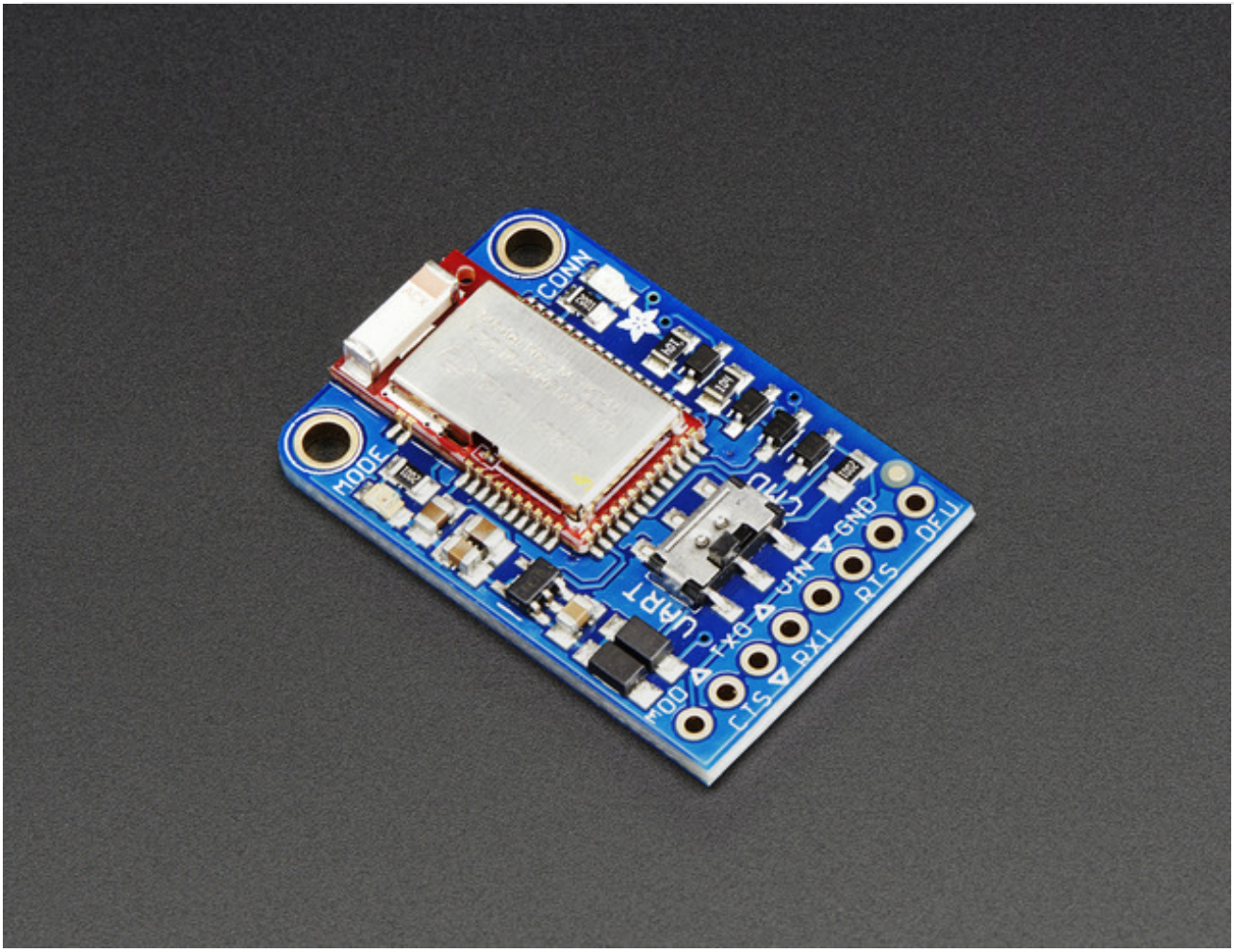
Created by Kevin Townsend



Last updated on 2015-07-07 03:20:14 PM EDT

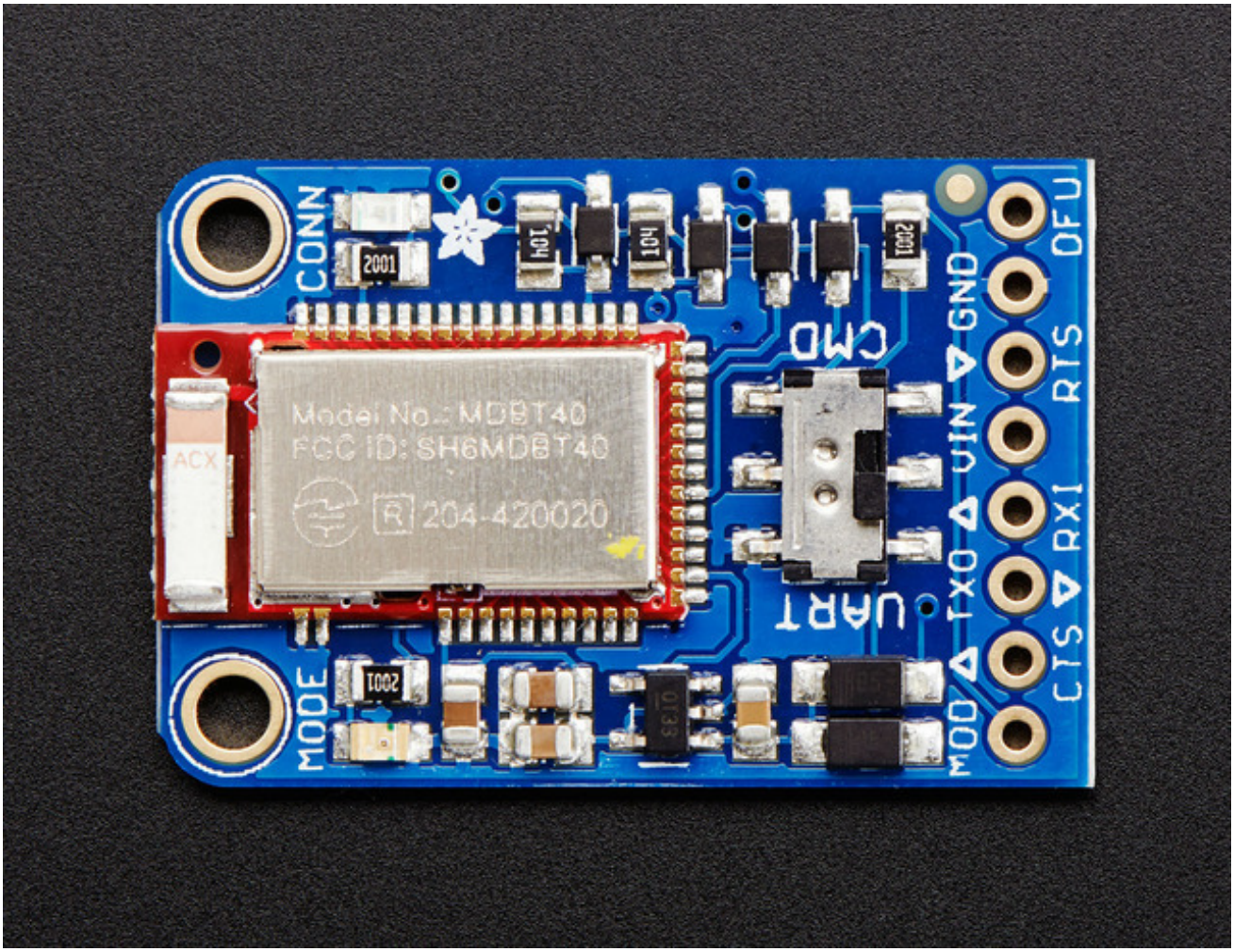
Guide Contents

Introduction



Would you like to add powerful and easy-to-use Bluetooth Low Energy to your robot, art or other electronics project? Heck yeah! With BLE now included in modern smart phones and tablets, its fun to add wireless connectivity. So what you really need is the new Adafruit Bluefruit LE UART Friend!

The Bluefruit LE UART Friend makes it easy to add Bluetooth Low Energy connectivity to anything with a hardware or software serial port. We even have nice hardware flow control so you won't have to think about losing data. Connect to your Arduino or other microcontroller or even just a standard FTDI cable for debugging and testing.



This multi-function module can do quite a lot! For most people, they'll be very happy to use the standard Nordic UART RX/TX connection profile. In this profile, the Bluefruit acts as a data pipe, that can 'transparently' transmit back and forth from your iOS or Android device. You can use our iOS App (http://adafru.it/f4Z) or Android App (http://adafru.it/f4G), or write your own to communicate with the UART service (http://adafru.it/f50).

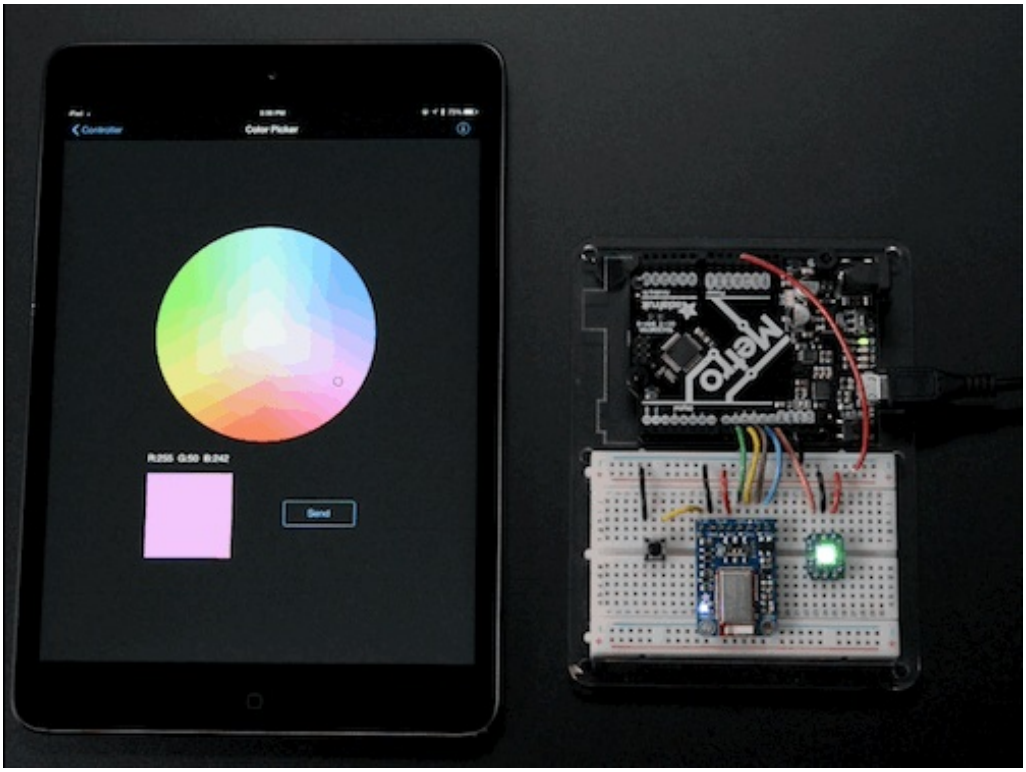
So it's a Fancy Pants Wireless UART Adapter?

The board is capable of much more than just sending strings over the air! Thanks to an easy to learn AT command set (http://adafru.it/f51), you have full control over how the device behaves, including the ability to define and manipulate your own GATT Services and Characteristics (http://adafru.it/f52), or change the way that the device advertises itself for other Bluetooth Low Energy devices to see. You can also use the AT commands to query the die temperature, check the battery voltage, and more, check the connection RSSI or MAC address, and tons more. Really, way too long to list here!

Download our free Android/iOS app and you're ready to rock!

Using our Bluefruit iOS App (http://adafru.it/f4Z) or Android App (http://adafru.it/f4G), you can quickly

get your project prototyped by using your iOS or Android phone/tablet as a controller. We have a color picker (http://adafru.it/f53), quaternion/accelerometer/gyro/magnetometer or location (GPS) (http://adafru.it/f53), and an 8-button control game pad (http://adafru.it/f53).



You can do a lot more too!

- The Bluefruit can also act like an HID Keyboard (http://adafru.it/f6w) (for devices that support BLE HID)
- Can become a BLE Heart Rate Monitor (http://adafru.it/f6x) (a standard profile for BLE) - you just need to add the pulse-detection circuitry
- Turn it into a UriBeacon (http://adafru.it/f6y), the Google standard for Bluetooth LE beacons. Just power it and the 'Friend will bleep out a URL to any nearby devices with the UriBeacon app installed.
- Built in over-the-air bootloading capability so we can keep you updated with the hottest new firmware. (http://adafru.it/f6z) Use any Android or iOS device to get updates and install them!

Why Use Adafruit's Module?

There are plenty of BLE modules out there, with varying quality on the HW design as well as the firmware.

One of the biggest advantages of the Adafruit Bluefruit LE family is that **we wrote all of the firmware running on the devices ourselves from scratch.**

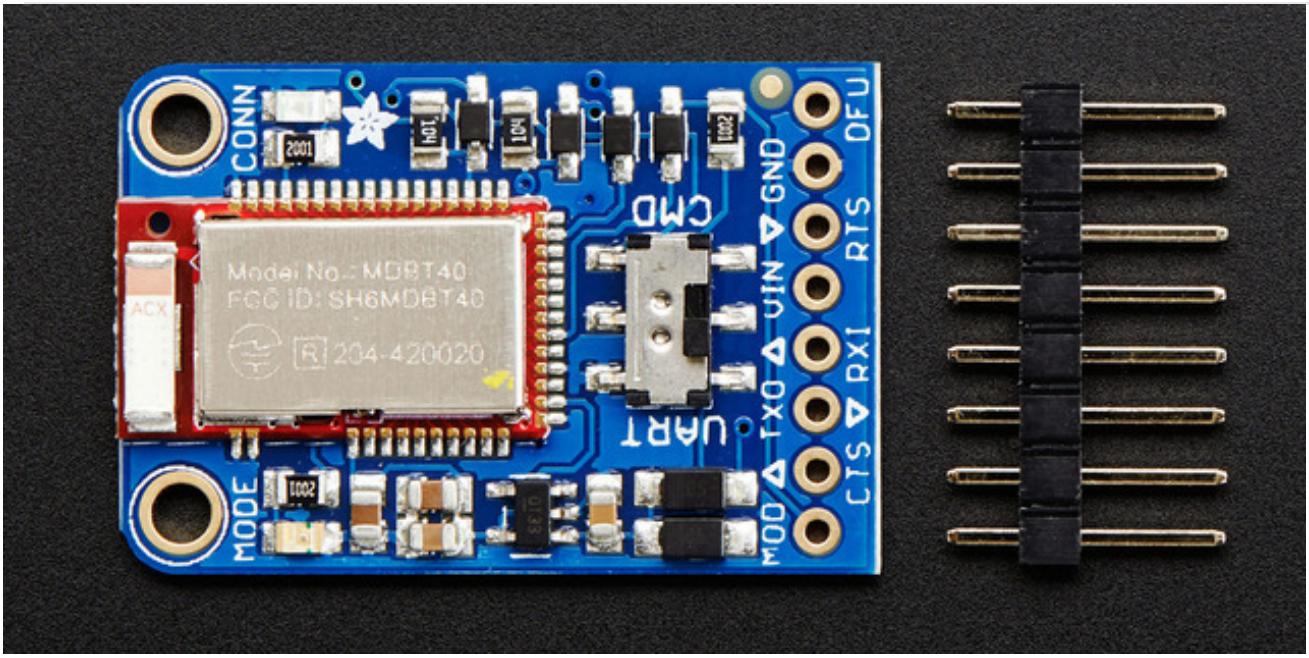
We control every line of code that runs on our modules ... and so we aren't at the mercy of any third party vendors who may or may not be interested in keeping their code up to date or catering to our customer's needs.

Because we control everything about the product, we add features that are important to *our* customers, can solve any issues that do come up without begging any 3rd parties, and we can even change Bluetooth SoCs entirely if the need ever arises!

Technical Specifications

- ARM Cortex M0 core running at 16MHz
- 256KB flash memory
- 32KB SRAM
- Peak current draw <20mA (radio actively transmitting/receiving)
- Transport: UART @ 9600 baud with HW flow control (CTS+RTS required)
- 5V-safe inputs (Arduino Uno friendly, etc.)
- On-board 3.3V voltage regulation
- Bootloader with support for safe OTA firmware updates
- Easy AT command set to get up and running quickly

Pinouts



Power Pins

- **VIN**: This is the power supply for the module, supply with 3.3-16V power supply input. This will be regulated down to 3.3V to run the chip
- **GND**: The common/GND pin for power and logic

UART Pins

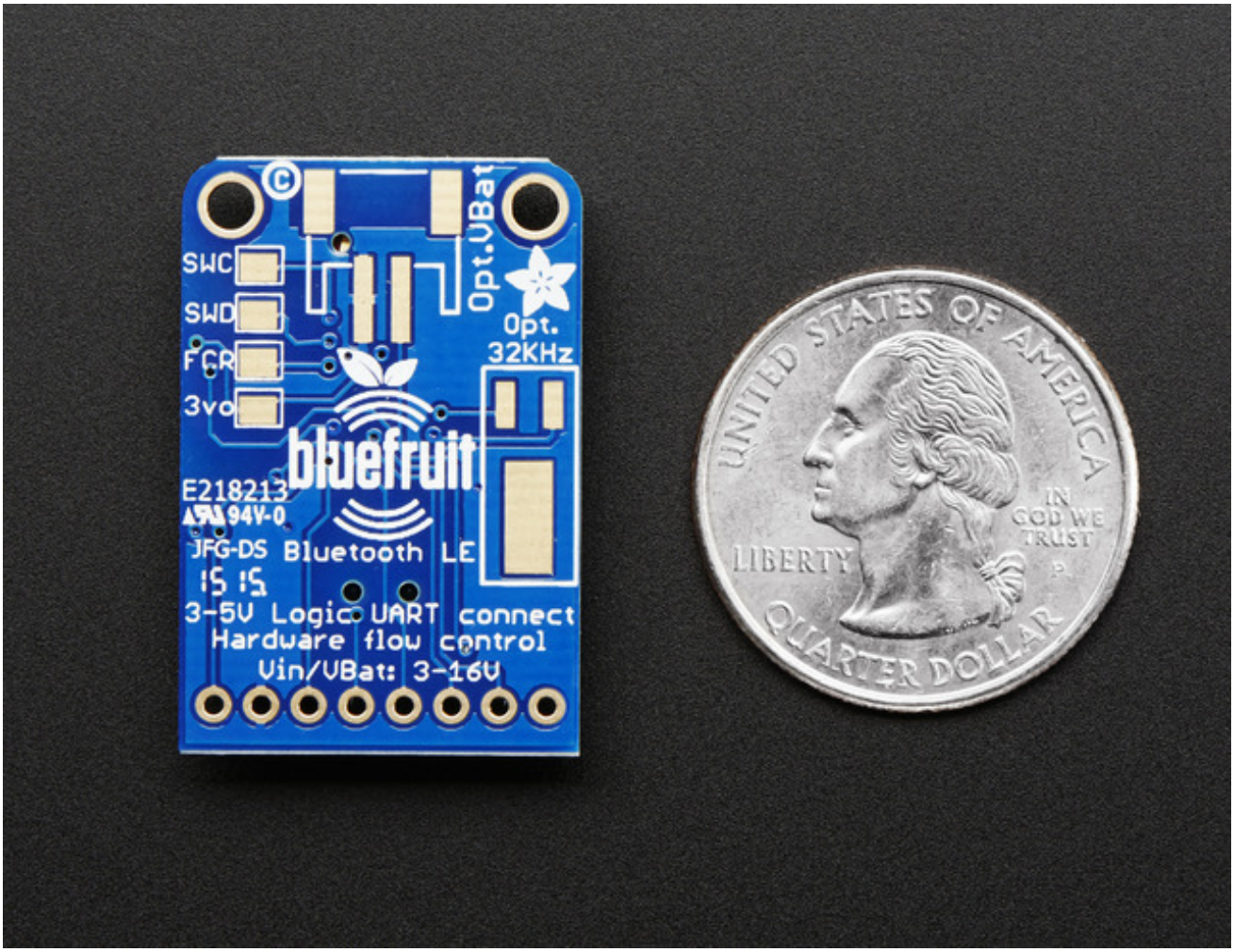
- **TXO** - This is the UART Transmit pin *out* of the breakout (Bluefruit LE --> MCU), it's at 3.3V logic level.
- **RXI** - This is the UART Receive pin *into* the breakout (MCU --> Bluefruit LE). This has a logic level shifter on it, you can use 3-5V logic.
- **CTS** - Clear to Send hardware flow control pin *into the* the breakou (MCU --> Bluefruit LE). Use this pin to tell the Bluefruit that it can send data back to the microcontroller over the TXO pin. **This pin is pulled high by default and must be set to ground in order to enable data transfer out! If you do not need hardware flow control, tie this pin to ground** it is a level shifted pin, you can use 3-5V logic
- **RTS** - Read to Send flow control pin *out of* the module (Bluefruit LE --> MCU). This pin will be low when its fine to send data to the Bluefruit. In general, at 9600 baud we haven't seen a need for this pin, but you can watch it for full flow control! This pin is 3.3V out

Other Pins

- **MOD**: Mode Selection. The Bluefruit has two modes, Command and Data. You can keep this pin disconnected, and use the slide switch to select the mode. Or, you can control the mode by setting this pin voltage, it will override the switch setting! High = Command Mode, Low =

UART/DATA mode. This pin is level shifted, you can use 3-5V logic

- **DFU**: Setting this pin low when you power the device up will force the Bluefruit LE module to enter a special firmware update mode to update the firmware over the air. Once the device is powered up, this pin can *also* be used to perform a factory reset. Wire the pin to GND for >5s until the two LEDs start to blink, then release the pin (set it to 5V or logic high) and a factory reset will be performed.



Reverse Side Breakouts

On the back we also have a few breakouts!

Opt VBat: If you fancy, you can solder on a JST 2-PH connector (http://adafru.it/1769), this will let you easily plug in a Lithium Ion or other battery pack. This connector pad is diode protected so you can use both Vin and VBat and the regulator will automatically switch to the higher voltage

Opt. 32 KHz: If you're doing some funky low power work, we wanted to give you the option of solderin in a 32khz oscillator. (http://adafru.it/f4U)Our firmware doesn't support it *yet* but its there!

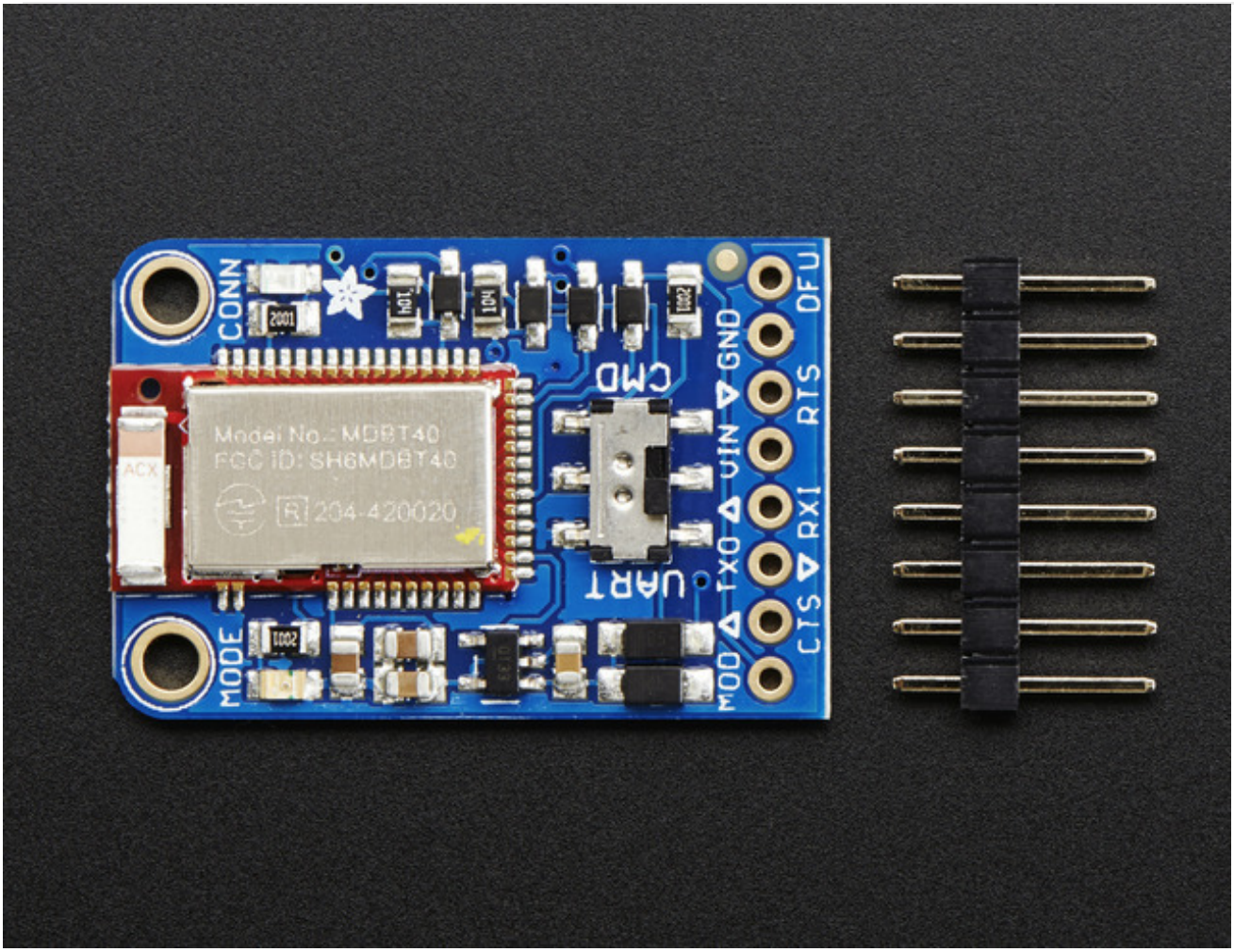
SWC: This is the SWD clock pin, 3v logic - for advanced hackers!

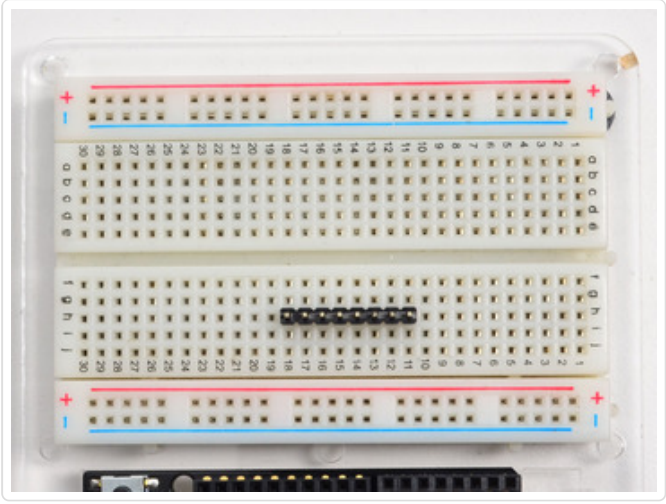
SWD: This is the SWD data pin, 3v logic - for advanced hackers!

3Vo: This is the output from the 3V regulator, for testing and also if you really need regulated 3V, up to 250mA available

FCR: This is the factory reset pin. When all else fails and you did something to really weird out your module, tie this pad to ground while powering up the module and it will factory reset. You should try the DFU reset method first tho (see that tutorial page)

Assembly

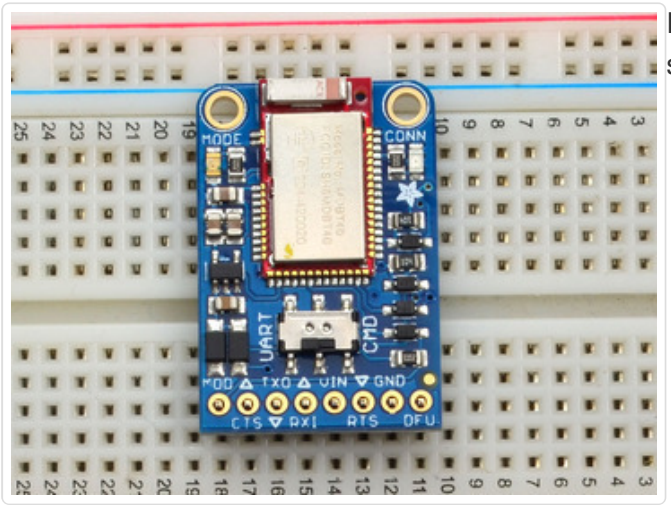




Prepare the header strip:

Cut the strip to length if necessary. It will be easier to solder if you insert it into a breadboard - **long pins down**

Add the breakout board:



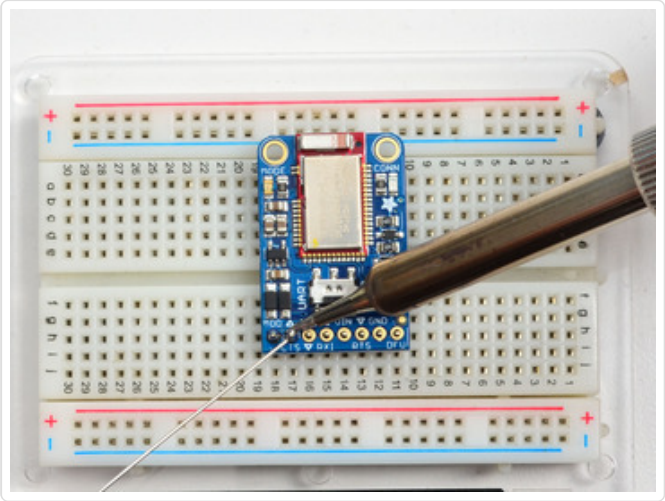
Place the breakout board over the pins so that the short pins poke through the breakout pads

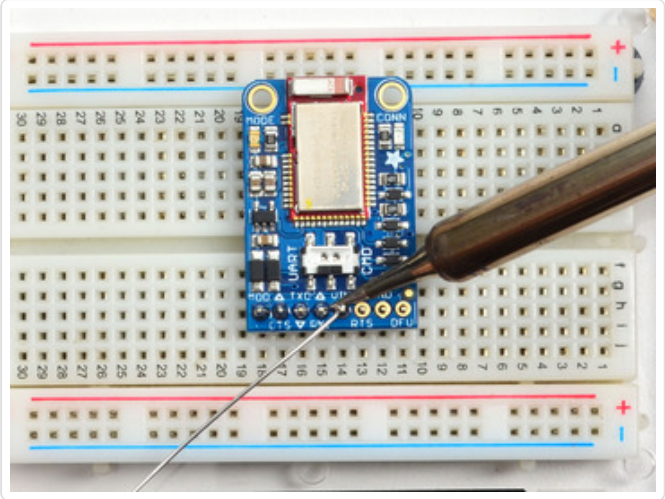
And Solder!

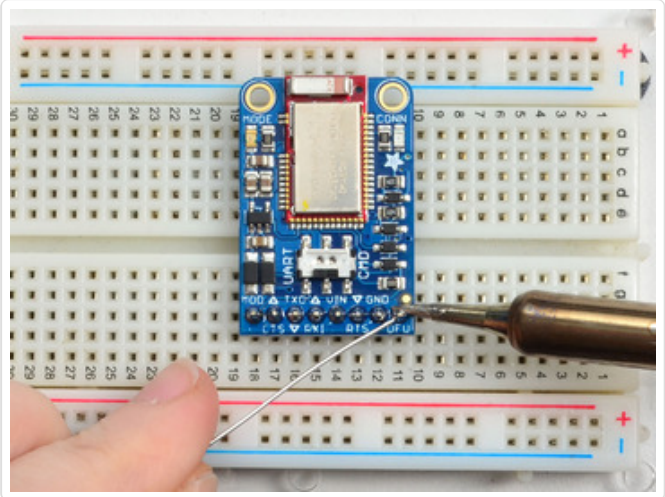
Be sure to solder all pins for reliable electrical contact.

Solder the longer power/data strip first

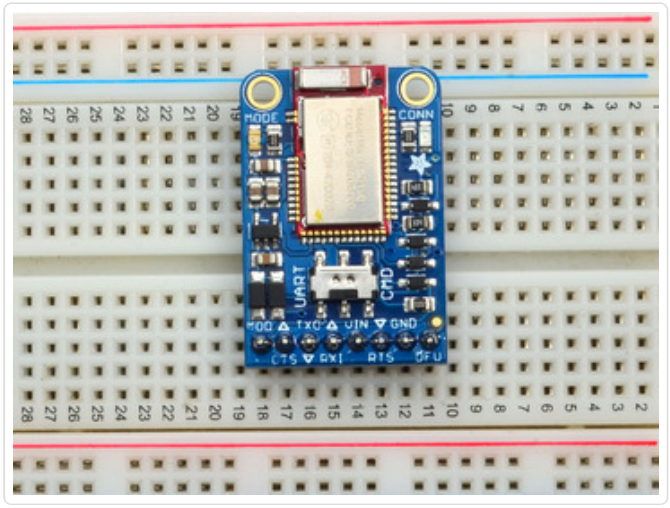
(For tips on soldering, be sure to check out our Guide to Excellent Soldering (http://adafru.it/aTk).







You're done! Check your solder joints visually and continue onto the next steps


MODE
CONN
UART
CMD
MOD
TXO
VIN
GND
CTS
RXI
RTS
DFU

Wiring

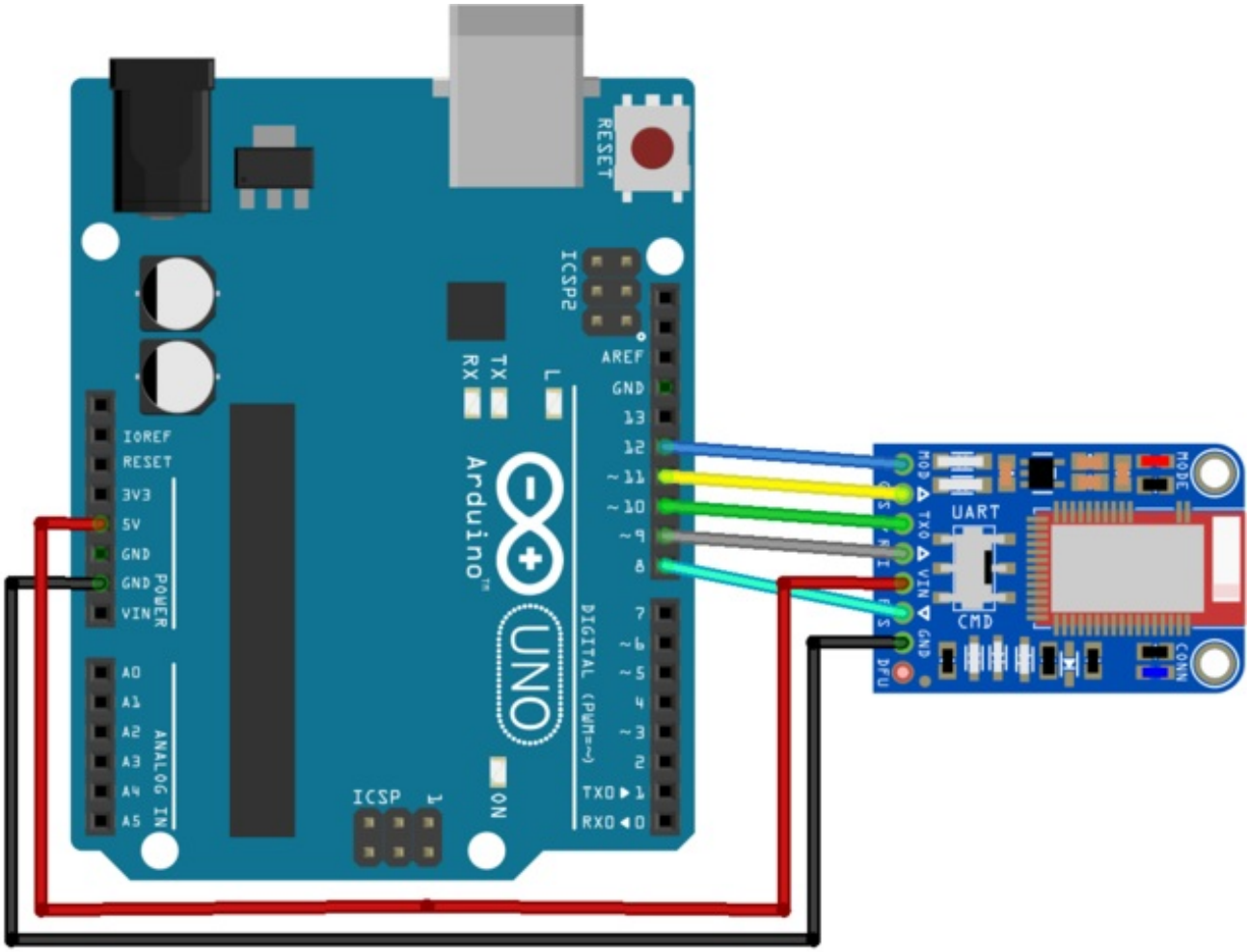
You can use the Bluefruit LE UART friend with any microcontroller with 3 or 5V logic, but we'll be demoing it with an Arduino UNO. Depending on whether your microcontroller has a hardware or software UART, adjust pins as necessary!

Wiring for Arduino Uno

To connect the Bluefruit LE UART Friend to your Arduino Uno using the default pinout in our sample sketches, connect the pins up as follows:

- **MOD** to **Pin 12**
- **CTS** to **Pin 11**
- **TXO** to **Pin 10**
- **RXI** to **Pin 9**
- **VIN** to **5V**
- **RTS** to **Pin 8**
- **GND** to **GND**

The wiring diagram below shows how this might look on your system:



All of these pins are 'flexible' and you can change them around as necesary after you get your setup running nicely, but we recommend starting out with our default wiring.

Wiring for an Arduino Micro or Leonardo

To connect the Bluefruit LE UART Friend to an Arduino Micro or Leonardo, use the following pinout and make sure you've selected HW UART as the constructor in your sample sketches (they default to SW Serial):

- **1/TX** to **RXI**
- **0/RX** to **TXO**
- **+5V** on the Micro/Leonardo to **VIN**
- **GND** to **GND**
- Connect **CTS** on the Bluefruit LE Module to **GND**

The HW Serial constructor can be seen below, and should be uncommented in the sample sketches:

```
/* ...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);
```

Wiring for an FTDI Cable

Since the UART Friend is, well, a serial port, you can use an FTDI Friend or cable to quickly connect using any serial console. You won't get MODE or DFU connections, so don't forget to flick the mode switch as necessary if you want to be in a particular mode.

Simply insert an FTDI cable directly into the Bluefruit LE UART Friend header by using the six pins in the middle, being careful to align the power pins up correctly (the red wire to VIN and the black wire to GND):

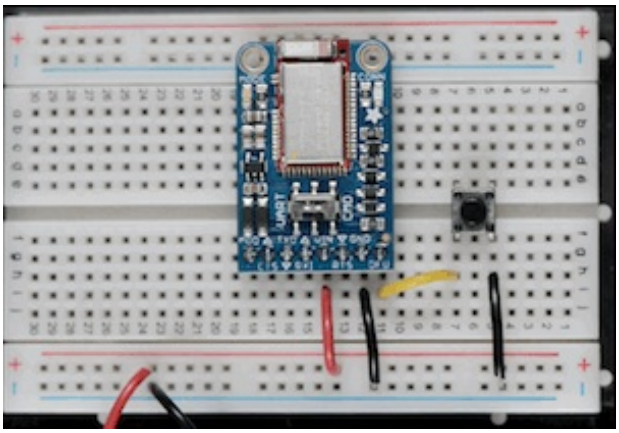
Factory Reset

There are several methods that you can use to perform a factory reset on your Bluefruit LE module if something gets misconfigured, or to delete persistent changes like UriBeacon or advertising payload changes, etc.

These methods are the same for both UART and SPI versions of Bluefruit LE

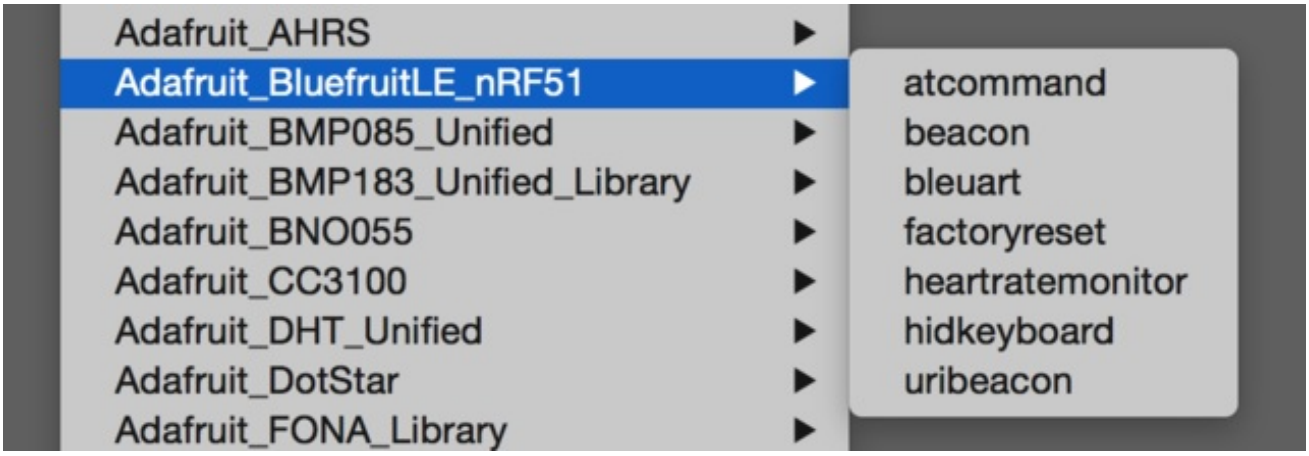
Factory Reset via DFU Pin

If you hold the DFU pin low (set the pin to GND) for **>5 seconds**, the two LEDs on the module will start to blinking and the device will perform a factory reset as soon as you release the DFU pin (removing it from GND).

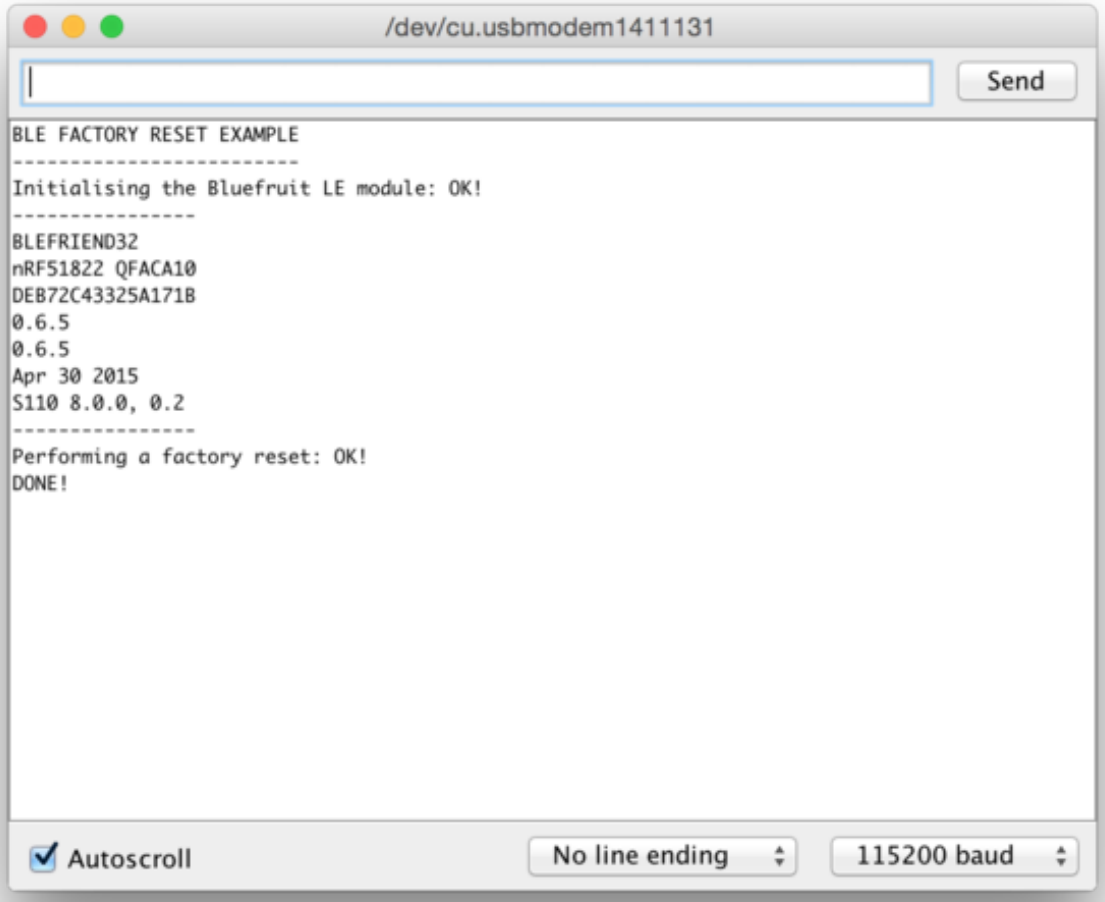


FactoryReset Sample Sketch

There is a FactoryReset sample sketch in the Adafruit Bluefruit LE library, which can be access in the **File > Examples > Adafruit_BluefruitLE_nRF51** folder:



Upload this sketch and open the Serial Monitor and it should perform a factory reset for you:



AT+FACTORYRESET

You can also perform a factory reset by sending the **AT+FACTORYRESET** command to your Bluefruit LE module in your favorite terminal emulator or using the ATCommand (http://adafru.it/f4E) example sketch.

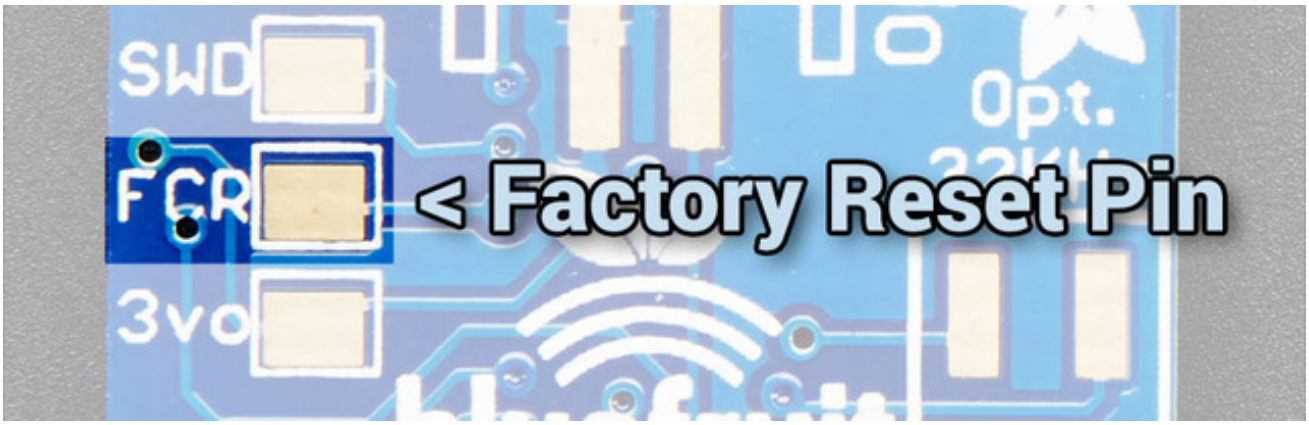
```
AT+FACTORYRESET
OK
```

This command will also cause the device to reset.

Factory Reset via FCTR Test Pad

On the bottom of the Bluefruit LE UART Friend board there is a test pad that exposes the Factory Reset pin on the modules (marked **FCR**). Setting this pad low when the device is powered up will

cause a factory reset at startup.



DFU Updates

We're constantly working on the Bluefruit LE firmware to add new features, and keep up to date with what customers need and want.

To make sure you stay up to date with those changes, we've included an easy to use over the air updater on all of our nRF51 based Bluefruit LE modules.

Adafruit Bluefruit LE Connect

Updating your Bluefruit LE device to the latest firmware is as easy as installing Adafruit's Bluefruit LE Connect application (http://adafru.it/f4G) from the Google Play Store. (An updated iOS version with DFU support will be available shortly!)

Any time a firmware update is available, the application will propose to download the latest binaries and take care of all the details of transferring them to your Bluefruit device, ans shown in the video below:

Current Measurements

The following tables give you an idea of the average current draw in three different operating modes with the Bluefruit LE UART Friend:

- Fast Advertising Mode (the first 30 seconds after power up)
- Slow Advertising Mode (>30s since power up)
- Connected Mode (using the UART profile in this case)

Test Conditions

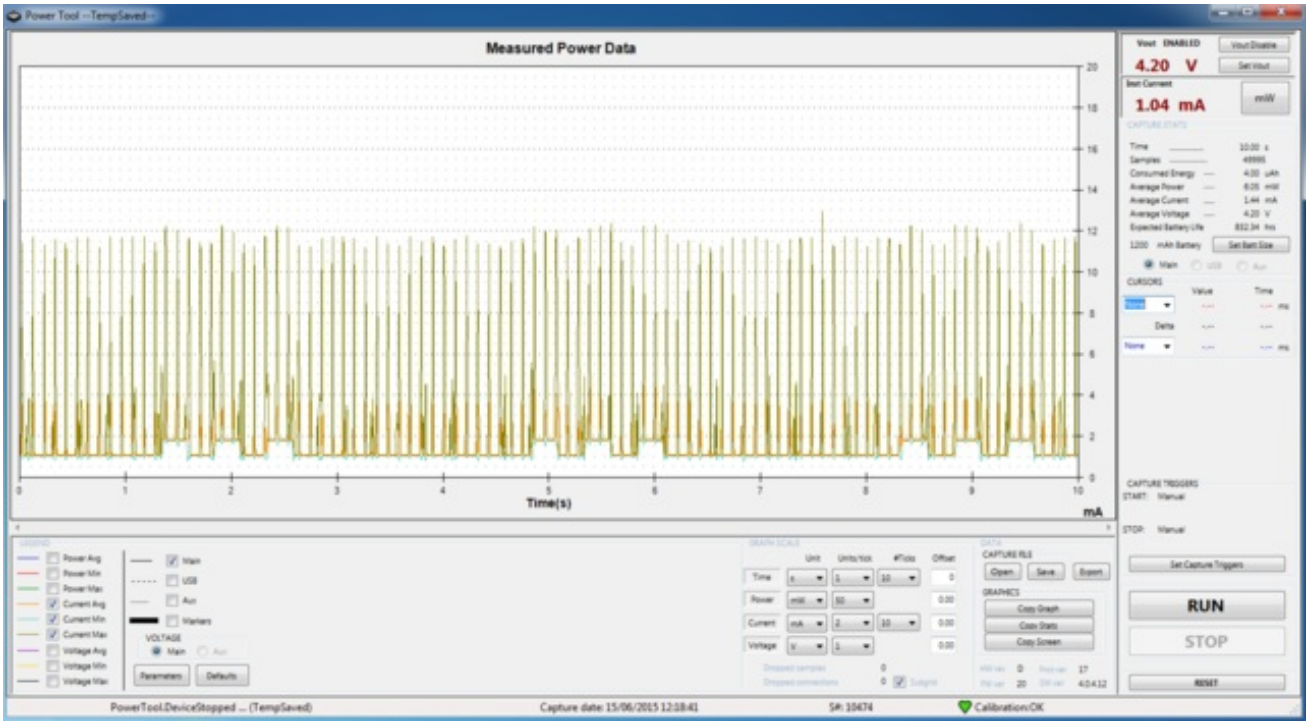
The board was powered from a fully charged 1200mAh 4.2V LIPO (http://adafru.it/dyW) cell running at 4.2V. Power efficiency will generally increase as the LIPO battery voltage drops, so 4.2V should be considered a worst case scenario.

- **Power Supply**: 1200mAh 4.2V LIPO Cell (at 4.2V)
- **Firmware**: 0.6.2

Fast Advertising Mode

The first 30 seconds that the Bluefruit LE UART Friend is powered up it will enter 'Fast Advertising Mode', which sends an advertising packet once every 100ms.

- **Average Current**: 1.44mA
- **Peak Current**: 13.5mA
- **Expected Battery Life**: 832 hours (~34.6 days)

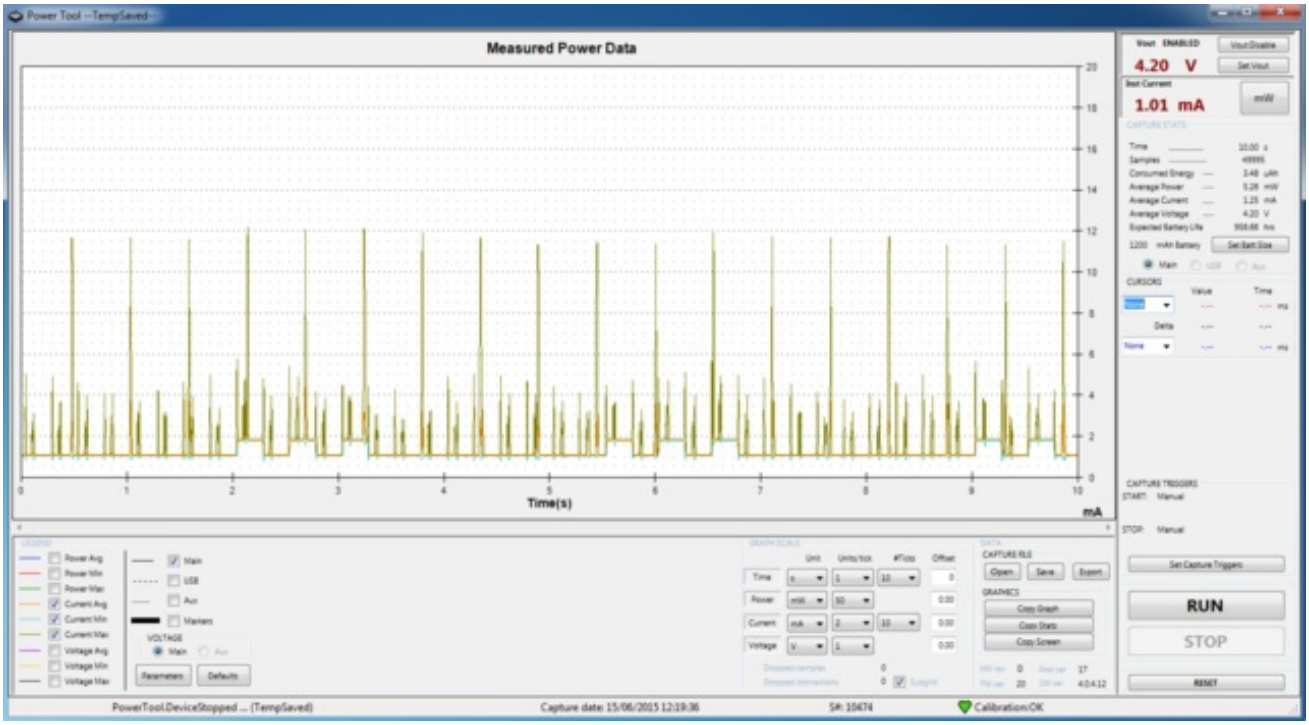


The MODE LEDs can be seen blinking in regular intervals as the three rectangular bumps at the bottom of the chart.

Slow Advertising Mode

After 30 seconds the Bluefruit LE UART Friend will enter 'Slow Advertising Mode', which sends an advertising packet once every 546.25ms.

- **Average Current**: 1.25mA
- **Peak Current**: 13.5mA
- **Expected Battery Life**: 956 hours (~40 days)



Connected Mode (UART)

The following measurements illustrate the average current draw in connected mode (with the connected LED enabled). UART mode was used as a test case.

- **Average Current**: 1.86mA
- **Peak Current**: 15.2mA
- **Expected Battery Life**: 645 hours (~26.8 days)


Measured Power Data
Time(s)
mA
4.20 V
2.18 mA
RUN
STOP
RESET

Software

In order to try out our demos, you'll need to download the Adafruit BLE library for the nRF51 based modules such as this one (a.k.a. Adafruit_BluefruitLE_nRF51)

You can check out the code here at github, (http://adafru.it/f4V) but its likely easier to just download by clicking:

Download
Adafruit_BluefruitLE_nRF51

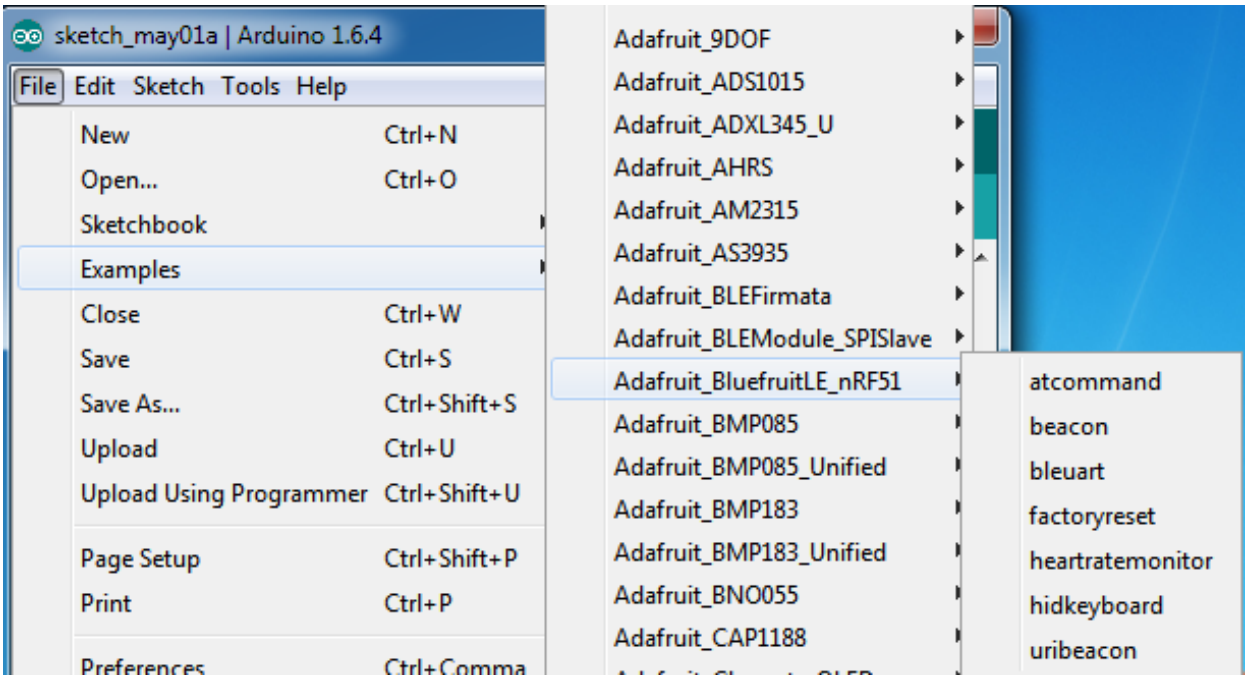
http://adafru.it/f4W

Rename the uncompressed folder **Adafruit_BluefruitLE_nRF51** and check that the **Adafruit_BluefruitLE_nRF51** folder contains **Adafruit_BLE.cpp** and **Adafruit_BLE.h** (as well as a bunch of other files)

Place the **Adafruit_BluefruitLE_nRF51** library folder your ***arduinosketchfolder*/libraries/** folder. You may need to create the **libraries** subfolder if its your first library. Restart the IDE.

We also have a great tutorial on Arduino library installation at:
http://learn.adafruit.com/adafruit-all-about-arduino-libraries-install-use (http://adafru.it/aYM)

After restarting, check that you see the library folder with examples:



Select the Serial Bus

The Adafruit_BluefruitLE_nRF51 library supports four different serial bus options, depending on the HW you are using.

UART Based Boards (Bluefruit LE UART Friend)

For software serial (Arduino Uno, Adafruit Metro) you should uncomment the software serial contructor below, and make sure the other three options are commented out:

```
SoftwareSerial bluefruitSS = SoftwareSerial(BLUEFRUIT_SWUART_TXD_PIN, BLUEFRUIT_SWUART_RXD_P

Adafruit_BluefruitLE_UART ble(bluefruitSS, BLUEFRUIT_UART_MODE_PIN,
                      BLUEFRUIT_UART_CTS_PIN, BLUEFRUIT_UART_RTS_PIN);
```

For boards that require hardware serial (Adafruit Flora, etc.), uncomment the hardware serial constructor, and make sure the other three options are commented out:

```
Adafruit_BluefruitLE_UART ble(BLUEFRUIT_HWSERIAL_NAME, BLUEFRUIT_UART_MODE_PIN);
```

SPI Based Boards (Bluefruit LE SPI Friend)

For SPI based boards, you should uncomment the hardware SPI constructor below, making sure the other constructors are commented out:

```
/* ...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);
```

If for some reason you can't use HW SPI, you can switch to software mode to bit-bang the SPI transfers via the following constructor:

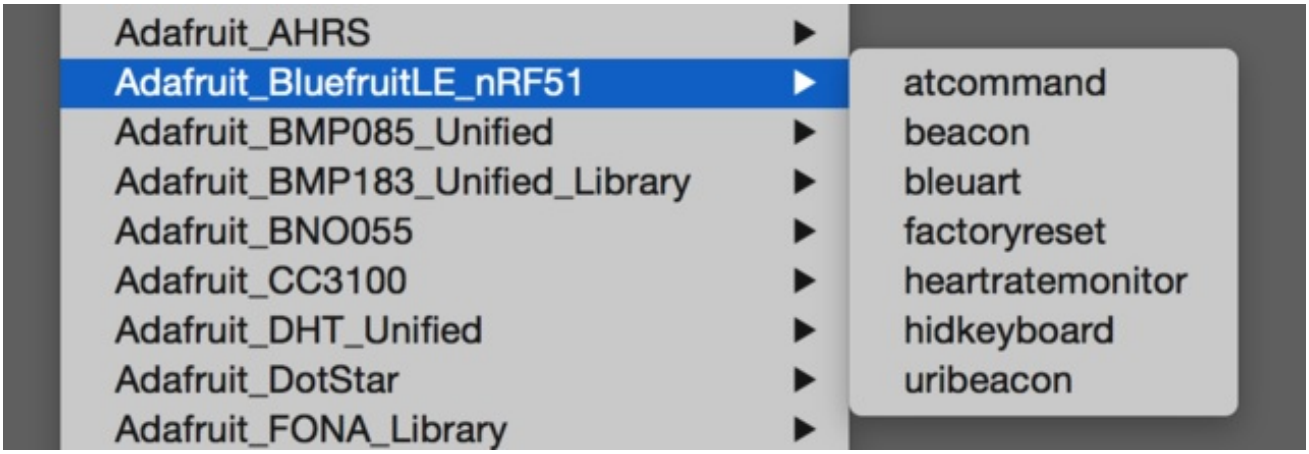
```
/* ...software SPI, using SCK/MOSI/MISO user-defined SPI pins and then user selected CS/IRQ/RST */
Adafruit_BluefruitLE_SPI ble(BLUEFRUIT_SPI_SCK, BLUEFRUIT_SPI_MISO,
                             BLUEFRUIT_SPI_MOSI, BLUEFRUIT_SPI_CS,
                             BLUEFRUIT_SPI_IRQ, BLUEFRUIT_SPI_RST);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

ATCommand

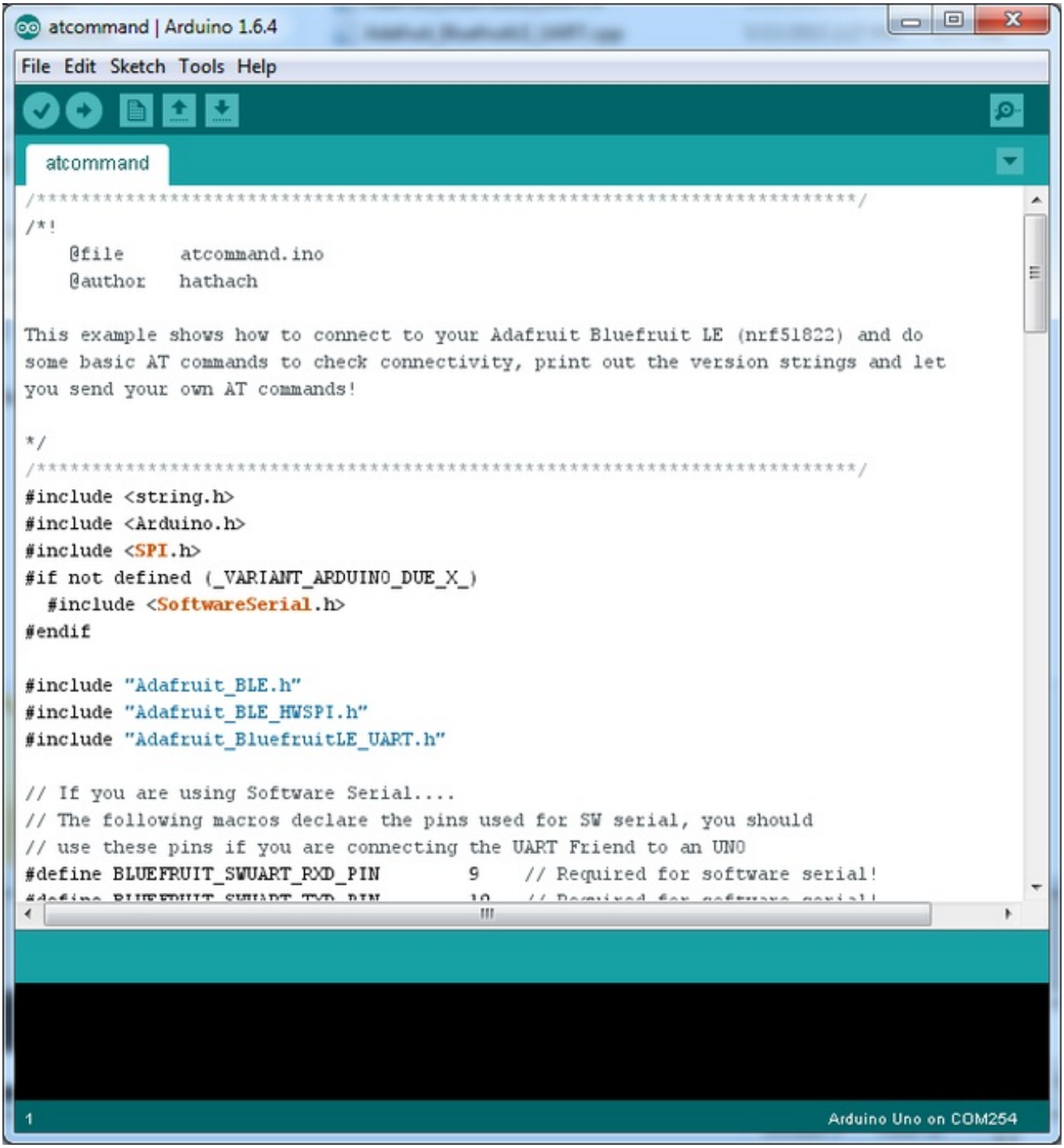
The **ATCommand** example allows you to execute AT commands from your sketch, and see the results in the Serial Monitor. This can be useful for debugging, or just testing different commands out to see how they work in the real world. It's a good one to start with!

Opening the Sketch

To open the ATCommand sketch, click on the **File > Examples > Adafruit_BluefruitLE_nRF51** folder in the Arduino IDE and select **atcommand**:



This will open up a new instance of the example in the IDE, as shown below:



Wiring

This example can be used with different wiring setups such as **Hardware Serial** (Flora/Micro/Leonardo/Due) or **Software Serial** (UNO/Metro & other Atmega328's), or **SPI** hardware or software (any microcontroller with 5 pins)

First determine which you have. If you are using ...

Serial Wiring

Software Serial

For UNO/Metro/Pro Trinket & other Atmega328's! **Three pins are required:** RX, TX, and CTS. They can connect to any pins on the Arduino but they must be wired up.

```
#define BLUEFRUIT_SWUART_RXD_PIN      9   // Required for software serial!
#define BLUEFRUIT_SWUART_TXD_PIN      10  // Required for software serial!
#define BLUEFRUIT_UART_CTS_PIN        11  // Required for software serial!
```

Hardware Serial

For Flora/Micro/Leonardo/Due! You must connect up the RX and TX pins of the Bluefruit to your Arduino or microcontroller's hardware serial port, then uncomment the following line, to set the HW Serial port name

```
//#define BLUEFRUIT_HWSERIAL_NAME          Serial1
```

and:

```
//Adafruit_BluefruitLE_UART ble(BLUEFRUIT_HWSERIAL_NAME, BLUEFRUIT_UART_MODE_PIN);
```

Then comment or remove:

```
SoftwareSerial bluefruitSS = SoftwareSerial(BLUEFRUIT_SWUART_TXD_PIN, BLUEFRUIT_SWUART_RXD_P

Adafruit_BluefruitLE_UART ble(bluefruitSS, BLUEFRUIT_UART_MODE_PIN,
                      BLUEFRUIT_UART_CTS_PIN, BLUEFRUIT_UART_RTS_PIN);
```

Don't forget to also connect the CTS pin on the Bluefruit to ground if you are not using it!

Other Serial Pins

This tutorial will also be easier to use if you wire up the MODE pin, you can use any pin but our tutorial has pin 12 by default. You can change this to any pin. If you do not set the MODE pin then **make sure you have the mode switch in CMD mode!**

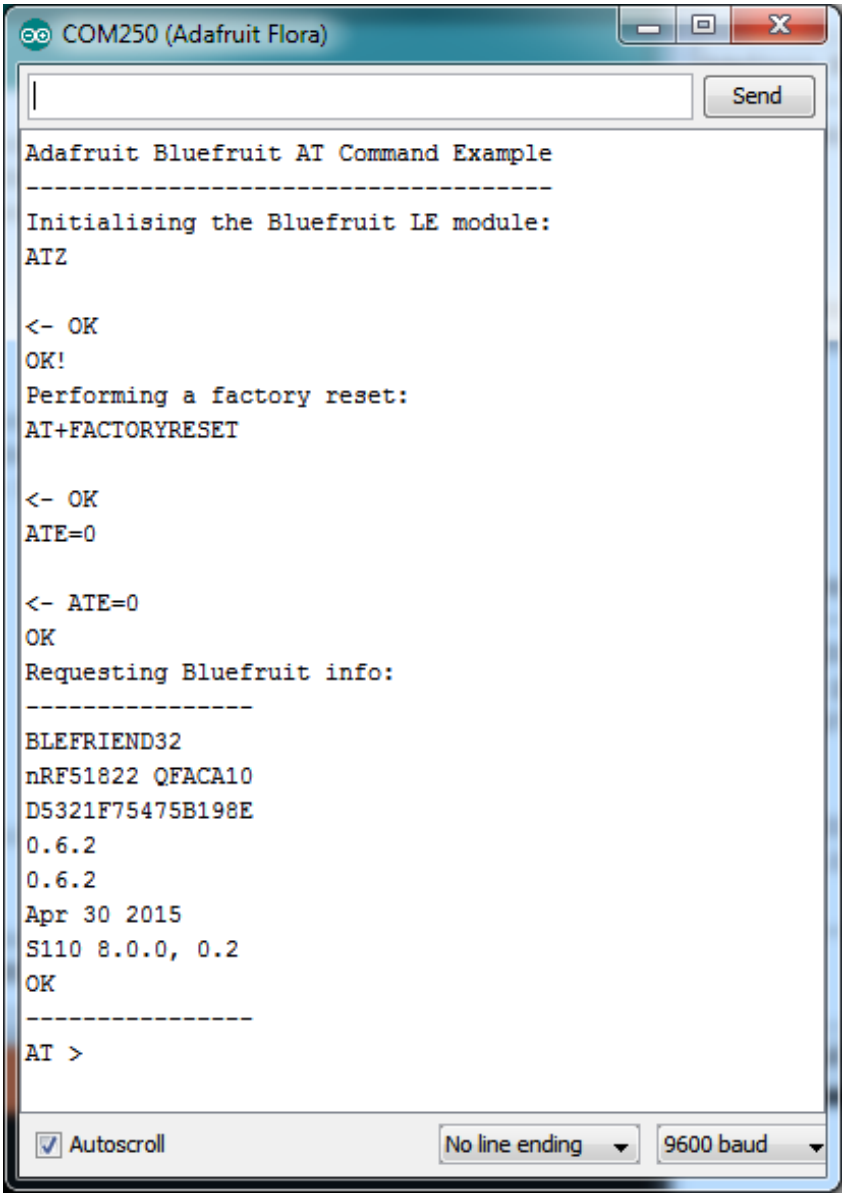
```
#define BLUEFRUIT_UART_MODE_PIN      12  // Optional but recommended, set to -1 if unused
```

SPI Wiring

If using hardware SPI, connect SCK/MOSI/MISO to the hardware SPI port of your microcontroller. If using software SPI, you can use any three pins. Then edit the config.h file to set up which pins you want to use. (http://adafru.it/fyg) SPI does not have flow control pins, or a MODE pin. The RST pin is optional but recommended if you can spare a pin.

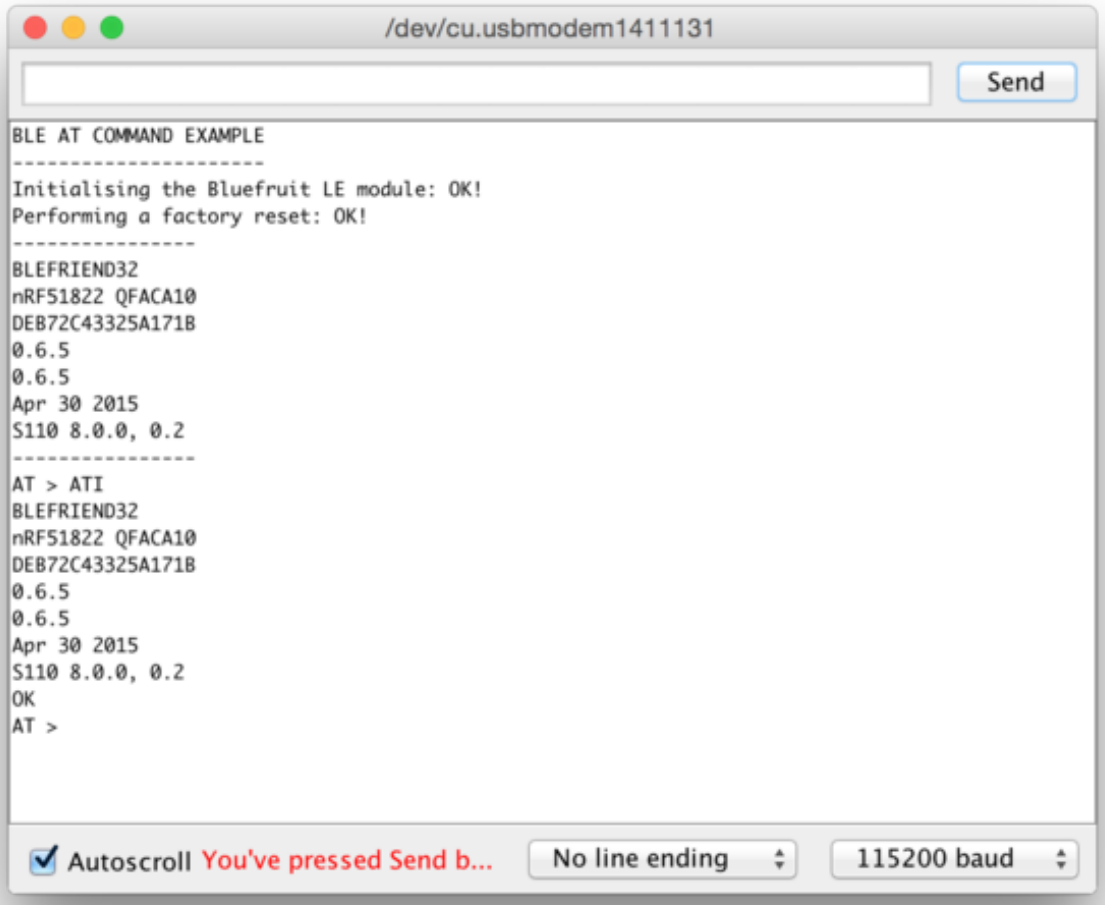
Running the Sketch

Once you upload the sketch to your board (via the arrow-shaped upload icon), and the upload process has finished, open up the Serial Monitor via **Tools > Serial Monitor**, and make sure that the baud rate in the lower right-hand corner is set to **115200**:

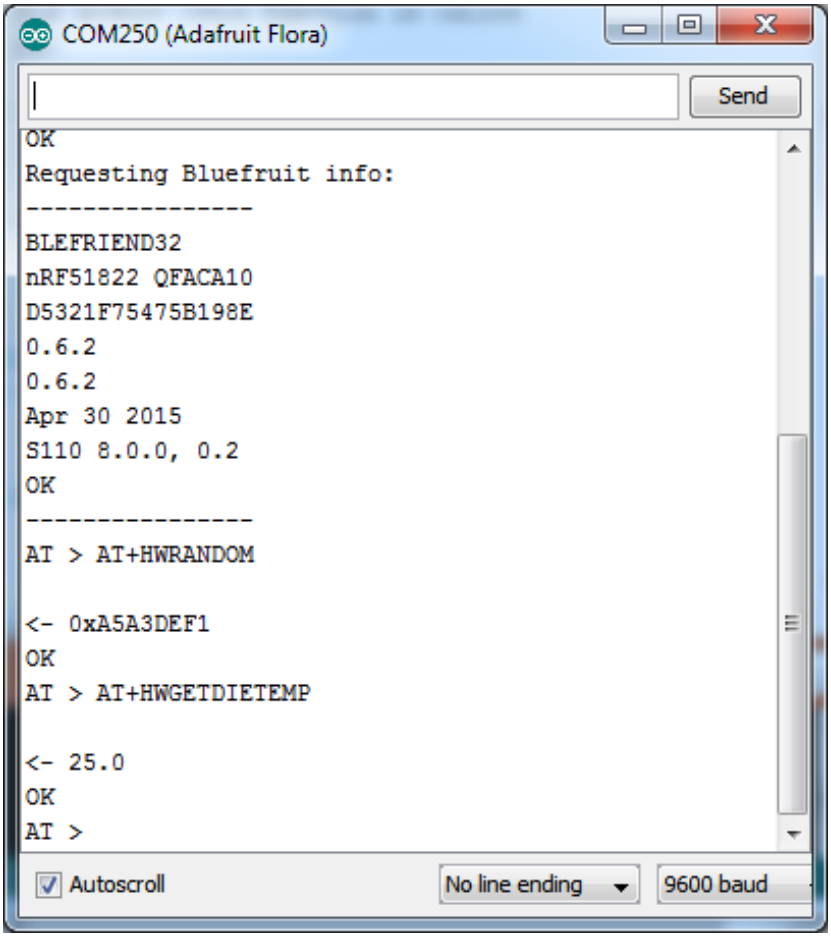


To send an AT command to the Bluefruit LE module, enter the command in the textbox at the top of the Serial Monitor and click the **Send** button:

The response to the AT command will be displayed in the main part of the Serial Monitor. The response from '**ATI**' is shown below:



You can do pretty much anything at this prompt, with the AT command set. Try **AT+HELP** to get a list of all commands, and try out ones like **AT+HWGETDIETEMP** (get temperature at the nRF51822 die) and **AT+HWRANDOM** (generate a random number)


COM250 (Adafruit Flora)
Send
OK
Requesting Bluefruit info:

BLEFRIEND32
nRF51822 QFACA10
D5321F75475B198E
0.6.2
0.6.2
Apr 30 2015
S110 8.0.0, 0.2
OK

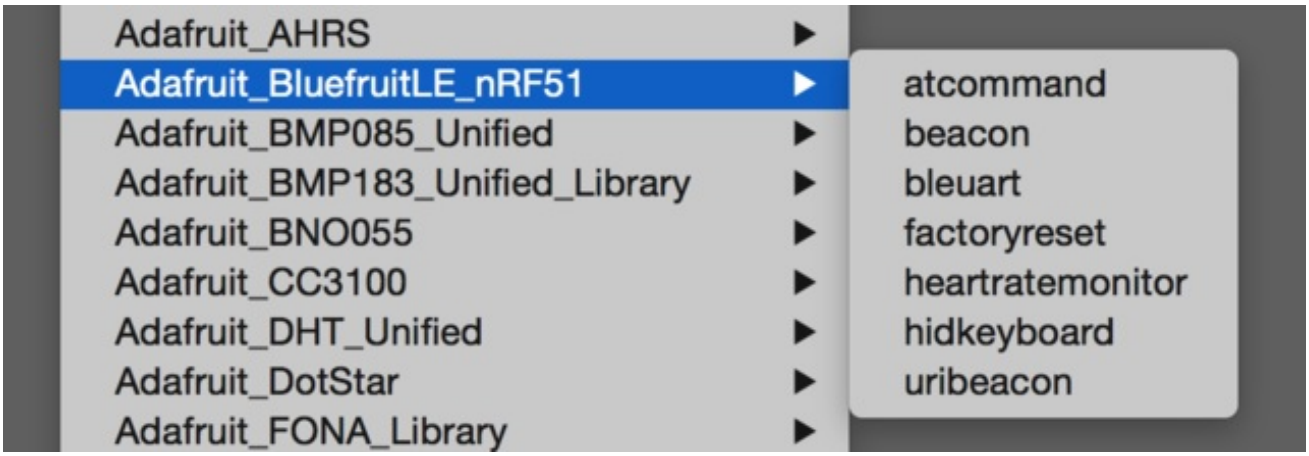
AT > AT+HWRANDOM
<- 0xA5A3DEF1
OK
AT > AT+HWGETDIETEMP
<- 25.0
OK
AT >
Autoscroll
No line ending
9600 baud

BLEUart

The **BLEUart** example sketch allows you to send and receive text data between the Arduino and a connected Bluetooth Low Energy Central device on the other end (such as you mobile phone using the **Adafruit Bluefruit LE Connect** application for Android (http://adafru.it/f4G) or iOS (http://adafru.it/f4H) in UART mode).

Opening the Sketch

To open the ATCommand sketch, click on the **File > Examples > Adafruit_BluefruitLE_nRF51** folder in the Arduino IDE and select **bleuart_cmdmode**:



This will open up a new instance of the example in the IDE, as shown below:

Wiring

This example can be used with different wiring setups such as **Hardware Serial** (Flora/Micro/Leonardo/Due) or **Software Serial** (UNO/Metro & other Atmega328's), or **SPI** hardware or software (any microcontroller with 5 pins)

First determine which you have. If you are using ...

Serial Wiring

Software Serial

For UNO/Metro/Pro Trinket & other Atmega328's! **Three pins are required:** RX, TX, and CTS. They can connect to any pins on the Arduino but they must be wired up.

```
#define BLUEFRUIT_SWUART_RXD_PIN     9   // Required for software serial!
#define BLUEFRUIT_SWUART_TXD_PIN     10  // Required for software serial!
#define BLUEFRUIT_UART_CTS_PIN       11  // Required for software serial!
```

Hardware Serial

For Flora/Micro/Leonardo/Due! You must connect up the RX and TX pins of the Bluefruit to your Arduino or microcontroller's hardware serial port, then uncomment the following line, to set the HW Serial port name

```
//#define BLUEFRUIT_HWSERIAL_NAME          Serial1
```

and:

```
//Adafruit_BluefruitLE_UART ble(BLUEFRUIT_HWSERIAL_NAME, BLUEFRUIT_UART_MODE_PIN);
```

Then comment or remove:

```
SoftwareSerial bluefruitSS = SoftwareSerial(BLUEFRUIT_SWUART_TXD_PIN, BLUEFRUIT_SWUART_RXD_P

Adafruit_BluefruitLE_UART ble(bluefruitSS, BLUEFRUIT_UART_MODE_PIN,
                      BLUEFRUIT_UART_CTS_PIN, BLUEFRUIT_UART_RTS_PIN);
```

Don't forget to also connect the CTS pin on the Bluefruit to ground if you are not using it! (The Flora has this already done)

Other Serial Pins

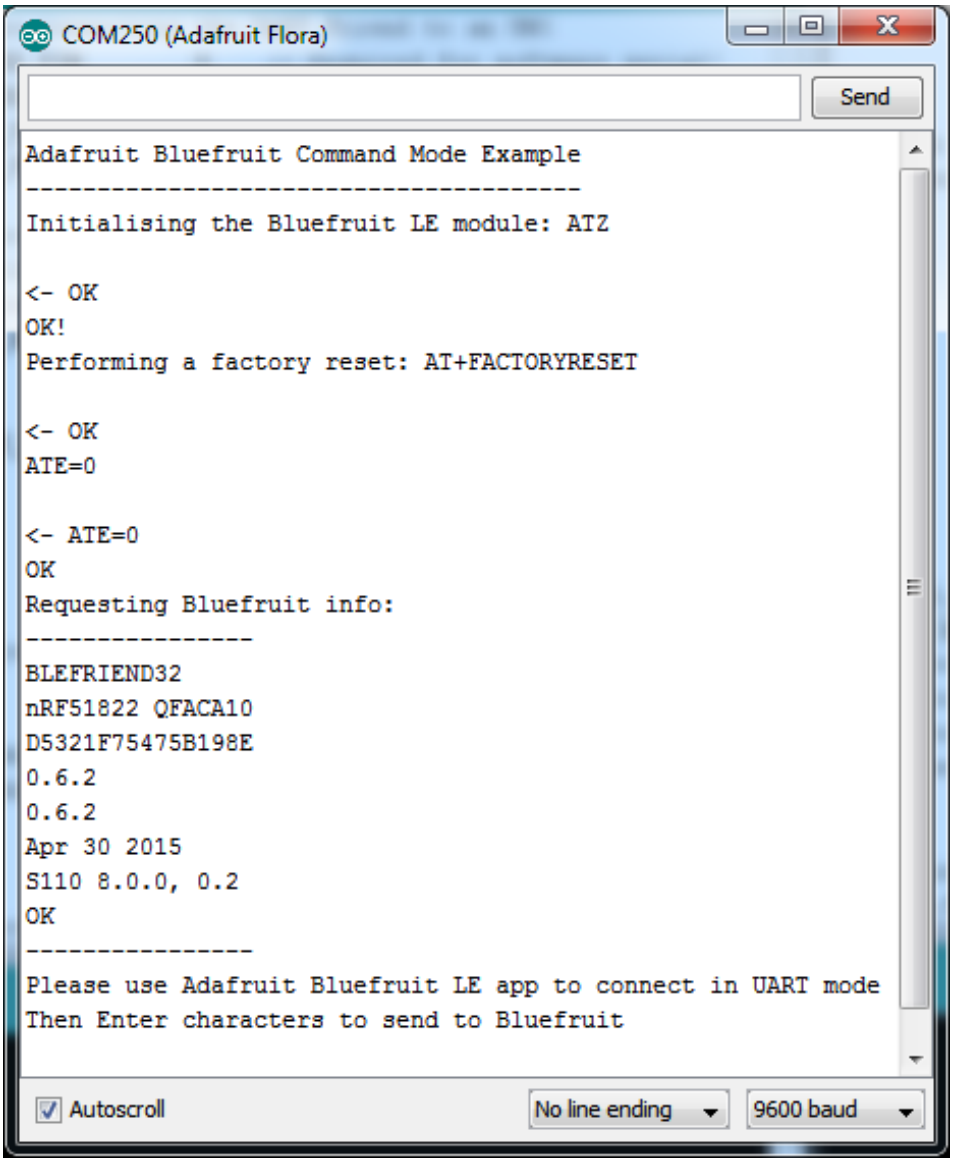
This sketch does not use any other serial pins on the Bluefruit

SPI Wiring

If using hardware SPI, connect SCK/MOSI/MISO to the hardware SPI port of your microcontroller. If using software SPI, you can use any three pins. Then edit the config.h file to set up which pins you want to use. (http://adafru.it/fyg) SPI does not have flow control pins, or a MODE pin. The RST pin is optional but recommended if you can spare a pin.

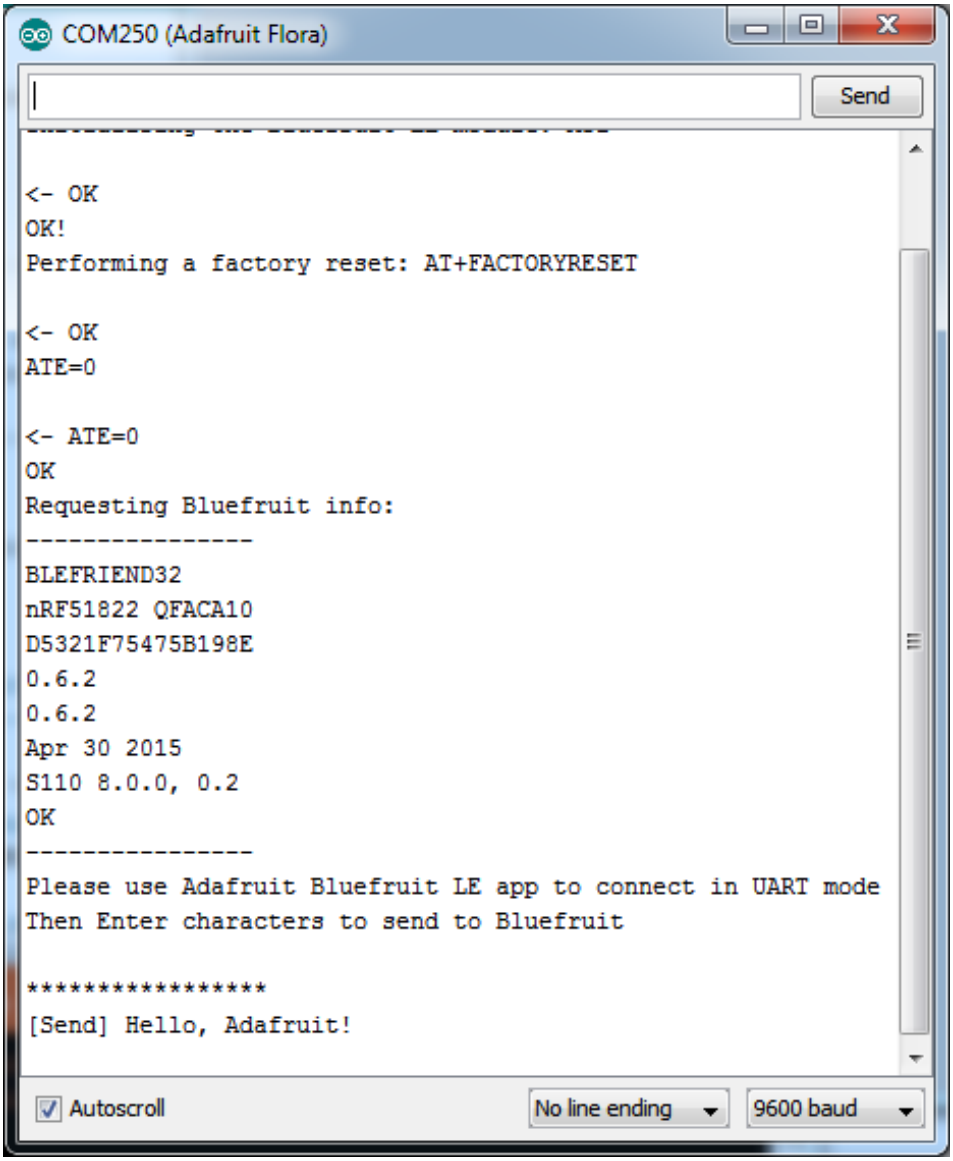
Running the Sketch

Once you upload the sketch to your board (via the arrow-shaped upload icon), and the upload process has finished, open up the Serial Monitor via **Tools > Serial Monitor**, and make sure that the baud rate in the lower right-hand corner is set to **115200**:

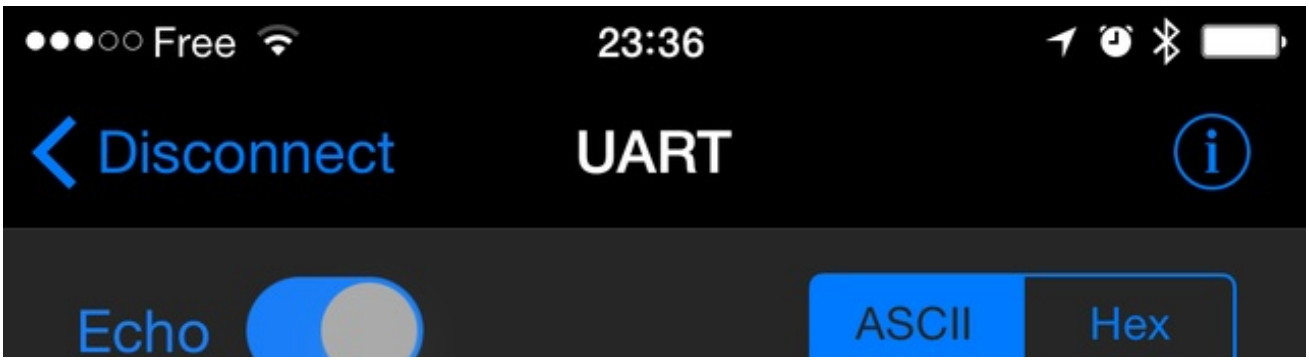


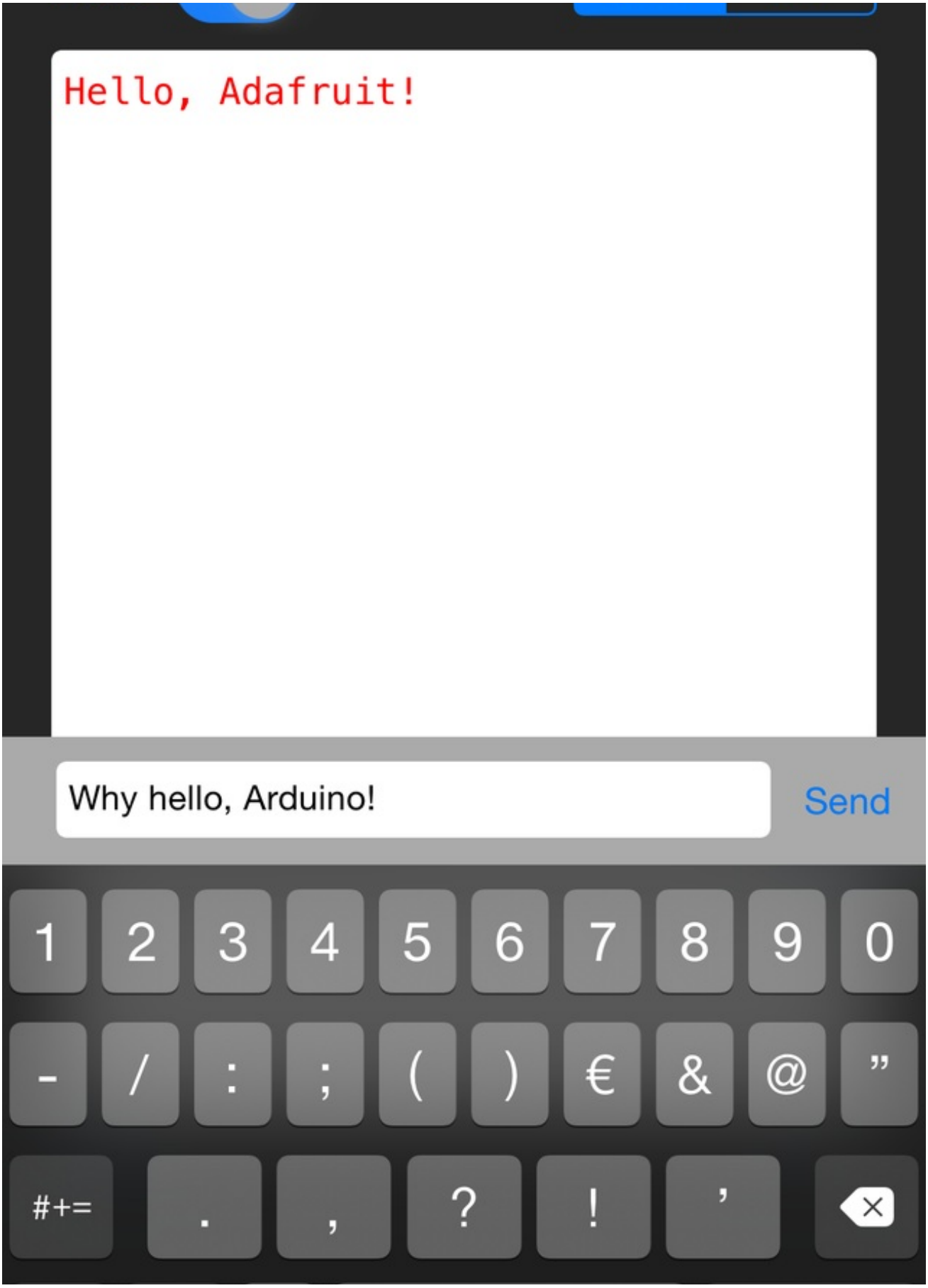
Once you see the request, use the App to connect to the Bluefruit LE module in **UART** mode so you get the text box on your phone

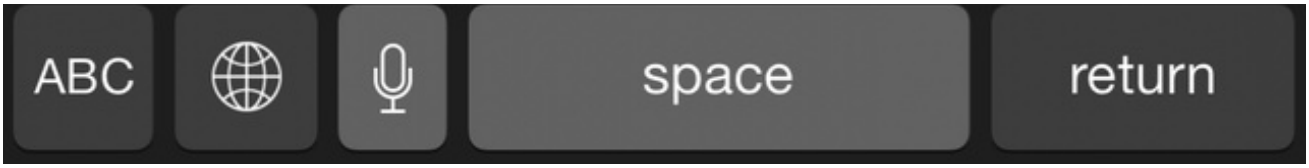
Any text that you type in the box at the top of the Serial Monitor will be sent to the connected phone, and any data sent from the phone will be displayed in the serial monitor:



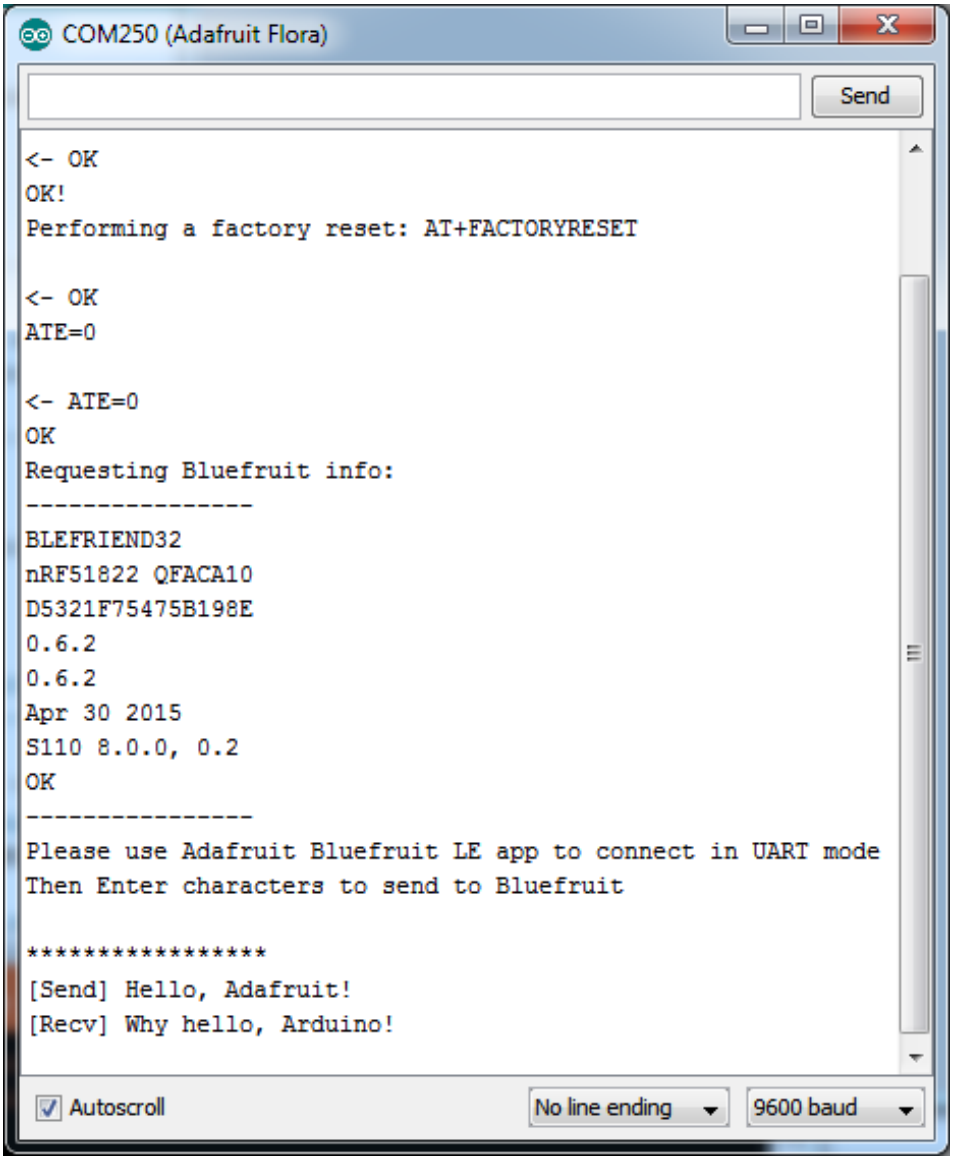
You can see the incoming string here in the Adafruit Bluefruit LE Connect app below (iOS in this case):




Hello, Adafruit!
Why hello, Arduino!
Send
1 2 3 4 5 6 7 8 9 0
- / : ; ( ) € & @ ”
#+= . , ? ! ’



The response text ('Why hello, Arduino!') can be seen below:

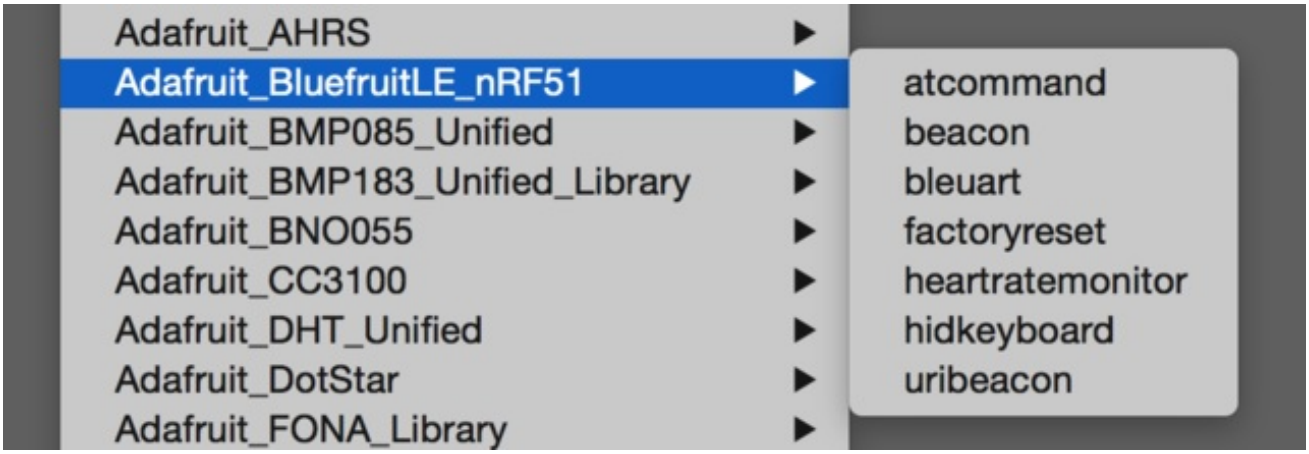


HIDKeyboard

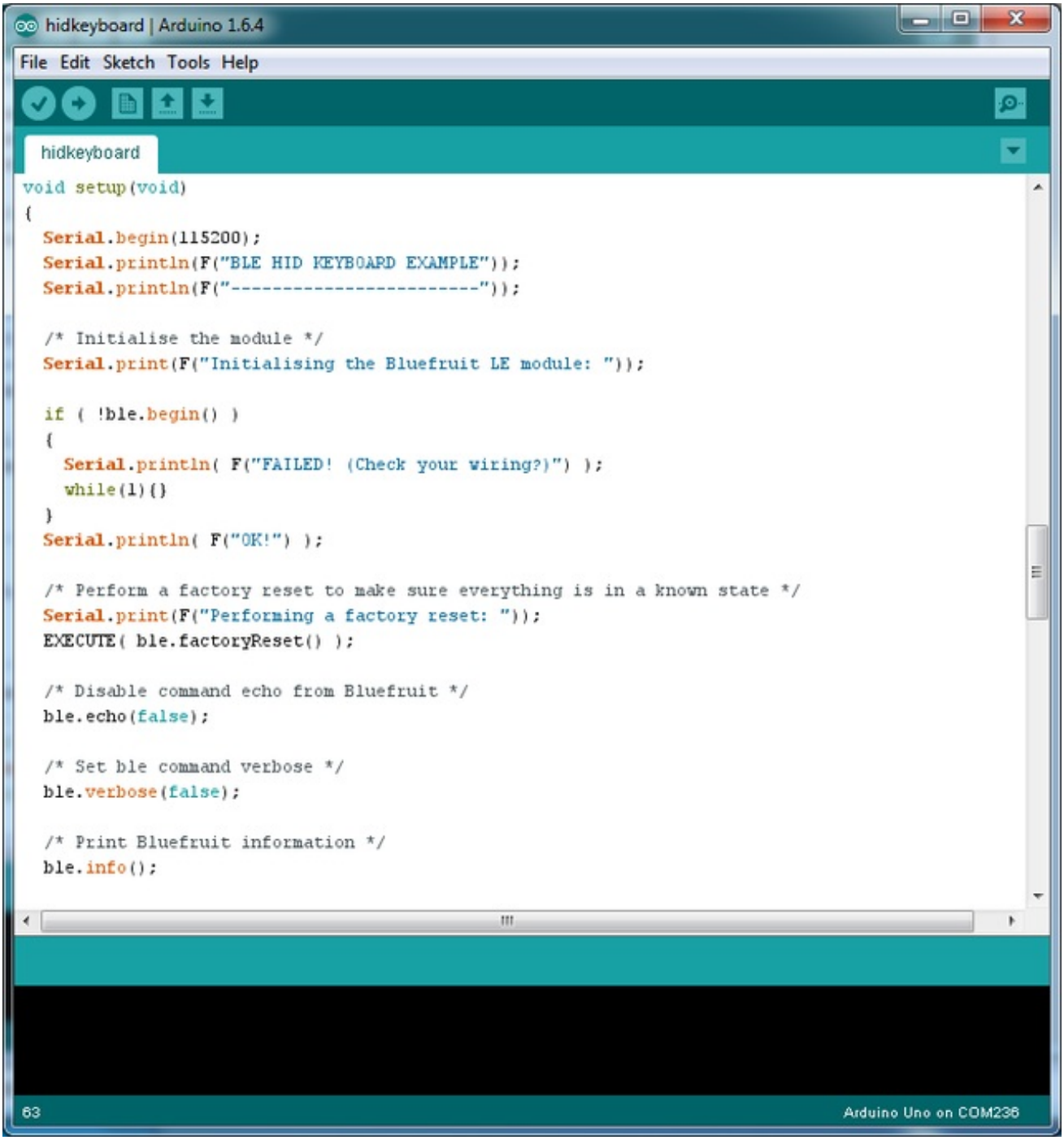
The **HIDKeyboard** example shows you how you can use the built-in HID keyboard AT commands to send keyboard data to any BLE-enabled Android or iOS phone, or other device that supports BLE HID peripherals.

Opening the Sketch

To open the ATCommand sketch, click on the **File > Examples > Adafruit_BluefruitLE_nRF51** folder in the Arduino IDE and select **hidkeyboard**:



This will open up a new instance of the example in the IDE, as shown below:



Wiring

This example can be used with different wiring setups such as **Hardware Serial** (Flora/Micro/Leonardo/Due) or **Software Serial** (UNO/Metro & other Atmega328's), or **SPI** hardware or software (any microcontroller with 5 pins)

First determine which you have. If you are using ...

Serial Wiring

Software Serial

For UNO/Metro/Pro Trinket & other Atmega328's! **Three pins are required:** RX, TX, and CTS. They can connect to any pins on the Arduino but they must be wired up.

```
#define BLUEFRUIT_SWUART_RXD_PIN      9   // Required for software serial!
#define BLUEFRUIT_SWUART_TXD_PIN      10  // Required for software serial!
#define BLUEFRUIT_UART_CTS_PIN        11  // Required for software serial!
```

Hardware Serial

For Flora/Micro/Leonardo/Due! You must connect up the RX and TX pins of the Bluefruit to your Arduino or microcontroller's hardware serial port, then uncomment the following line, to set the HW Serial port name

```
//#define BLUEFRUIT_HWSERIAL_NAME          Serial1
```

and:

```
//Adafruit_BluefruitLE_UART ble(BLUEFRUIT_HWSERIAL_NAME, BLUEFRUIT_UART_MODE_PIN);
```

Then comment or remove:

```
SoftwareSerial bluefruitSS = SoftwareSerial(BLUEFRUIT_SWUART_TXD_PIN, BLUEFRUIT_SWUART_RXD_P

Adafruit_BluefruitLE_UART ble(bluefruitSS, BLUEFRUIT_UART_MODE_PIN,
                      BLUEFRUIT_UART_CTS_PIN, BLUEFRUIT_UART_RTS_PIN);
```

Don't forget to also connect the CTS pin on the Bluefruit to ground if you are not using it! (The Flora has this already done)

Other Serial Pins

This sketch does not use any other pins on the Bluefruit

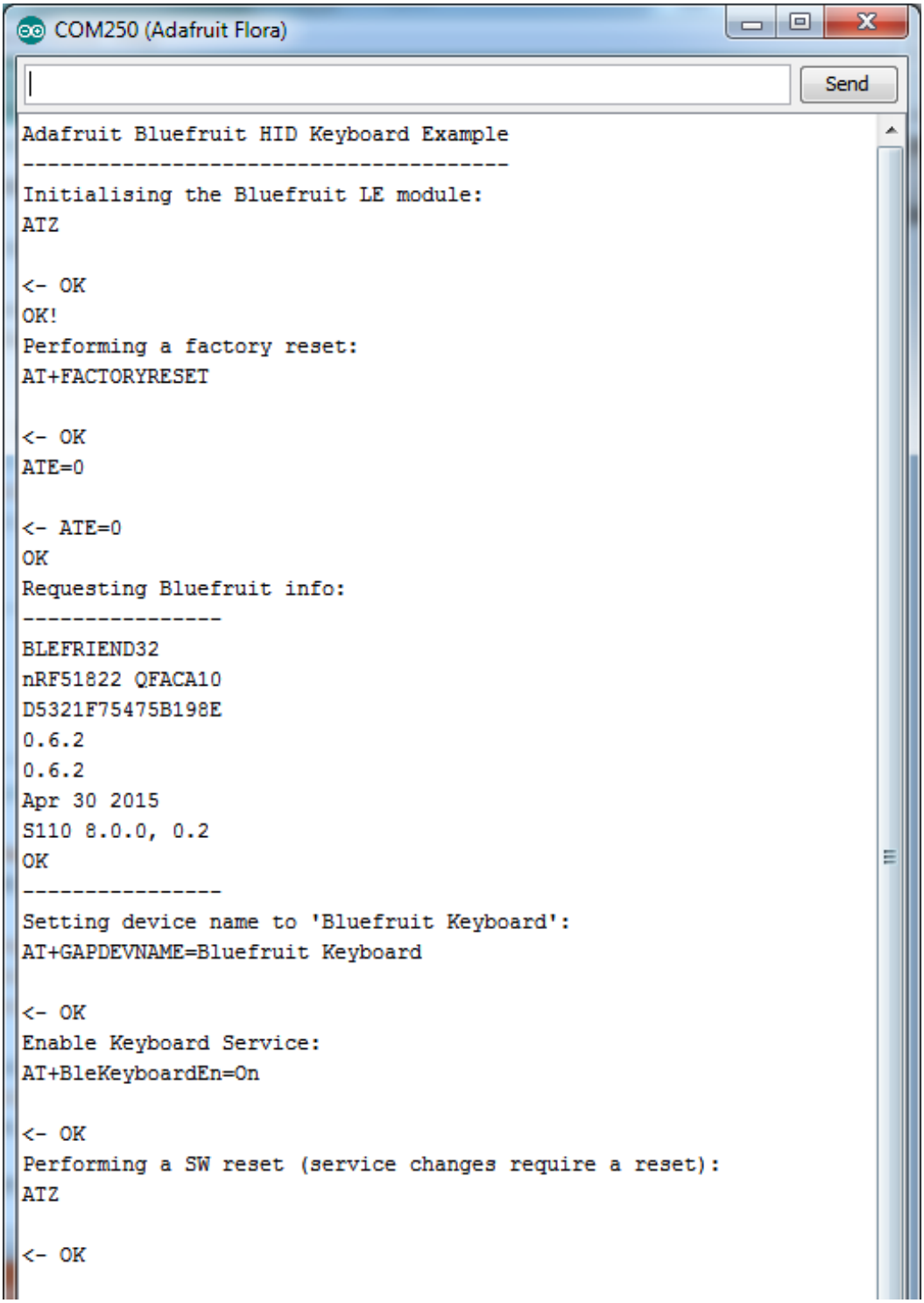
SPI Wiring

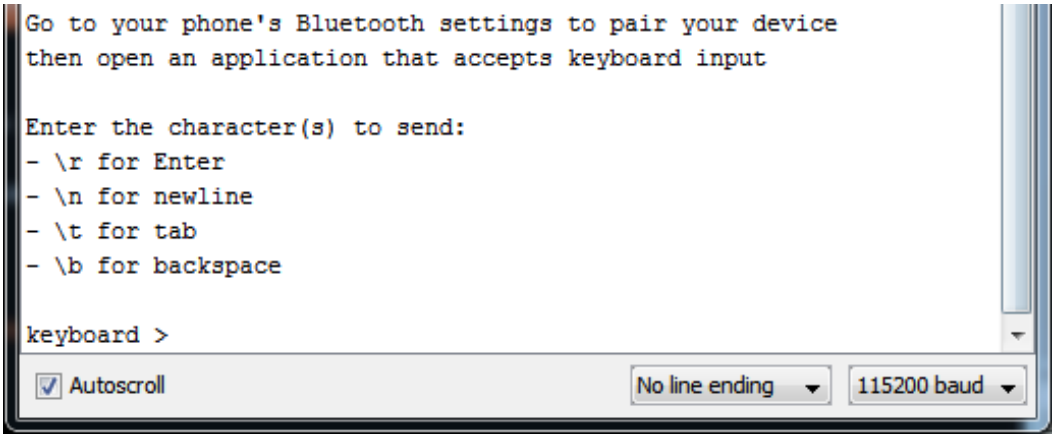
If using hardware SPI, connect SCK/MOSI/MISO to the hardware SPI port of your microcontroller. If using software SPI, you can use any three pins. Then edit the config.h file to set up which pins you want to use. (http://adafru.it/fyg) SPI does not have flow control pins, or a MODE pin. The RST pin is

optional but recommended if you can spare a pin.

Running the Sketch

Once you upload the sketch to your board (via the arrow-shaped upload icon), and the upload process has finished, open up the Serial Monitor via **Tools > Serial Monitor**, and make sure that the baud rate in the lower right-hand corner is set to **115200**:





To send keyboard data, type anything into the textbox at the top of the Serial Monitor and click the **Send** button.

Bonding the HID Keyboard

Before you can use the HID keyboard, you will need to 'bond' it to your phone or PC. The bonding process establishes a permanent connection between the two devices, meaning that as soon as your phone or PC sees the Bluefruit LE module again it will automatically connect.

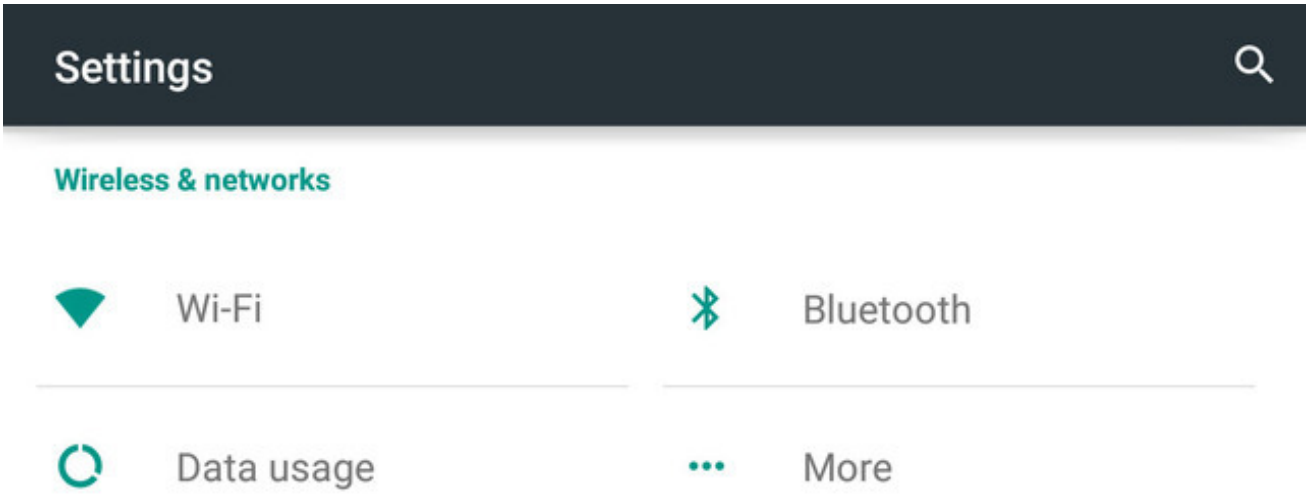
The exact procedures for bonding the keyboard will varying from one platform to another.

> When you no longer need a bond, or wish to bond the Bluefruit LE module to another device, be sure to delete the bonding information on the phone or PC, otherwise you may not be able to connect on a new device!

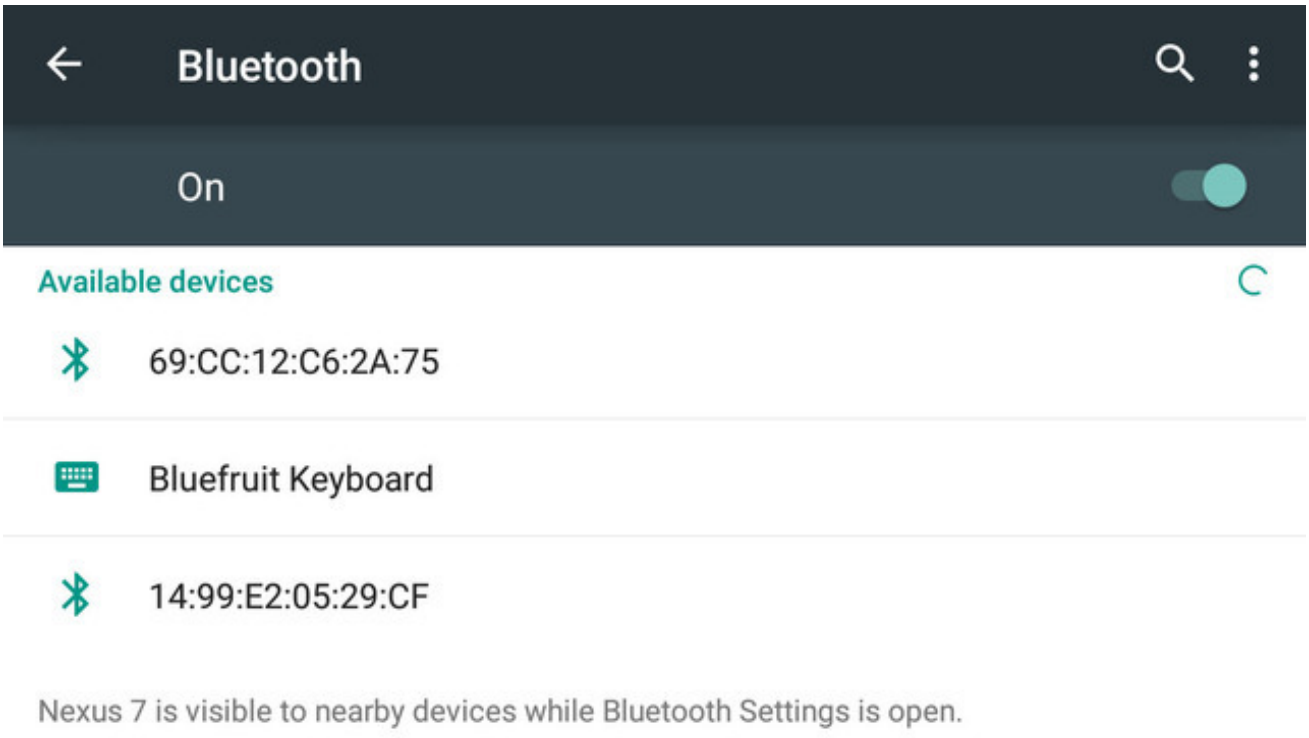
Android

To bond the keyboard on a Bluetooth Low Energy enabled Android device, go to the **Settings** application and click the **Bluetooth** icon.

> These screenshots are based on Android 5.0 running on a Nexus 7 2013. The exact appearance may vary depending on your device and OS version.

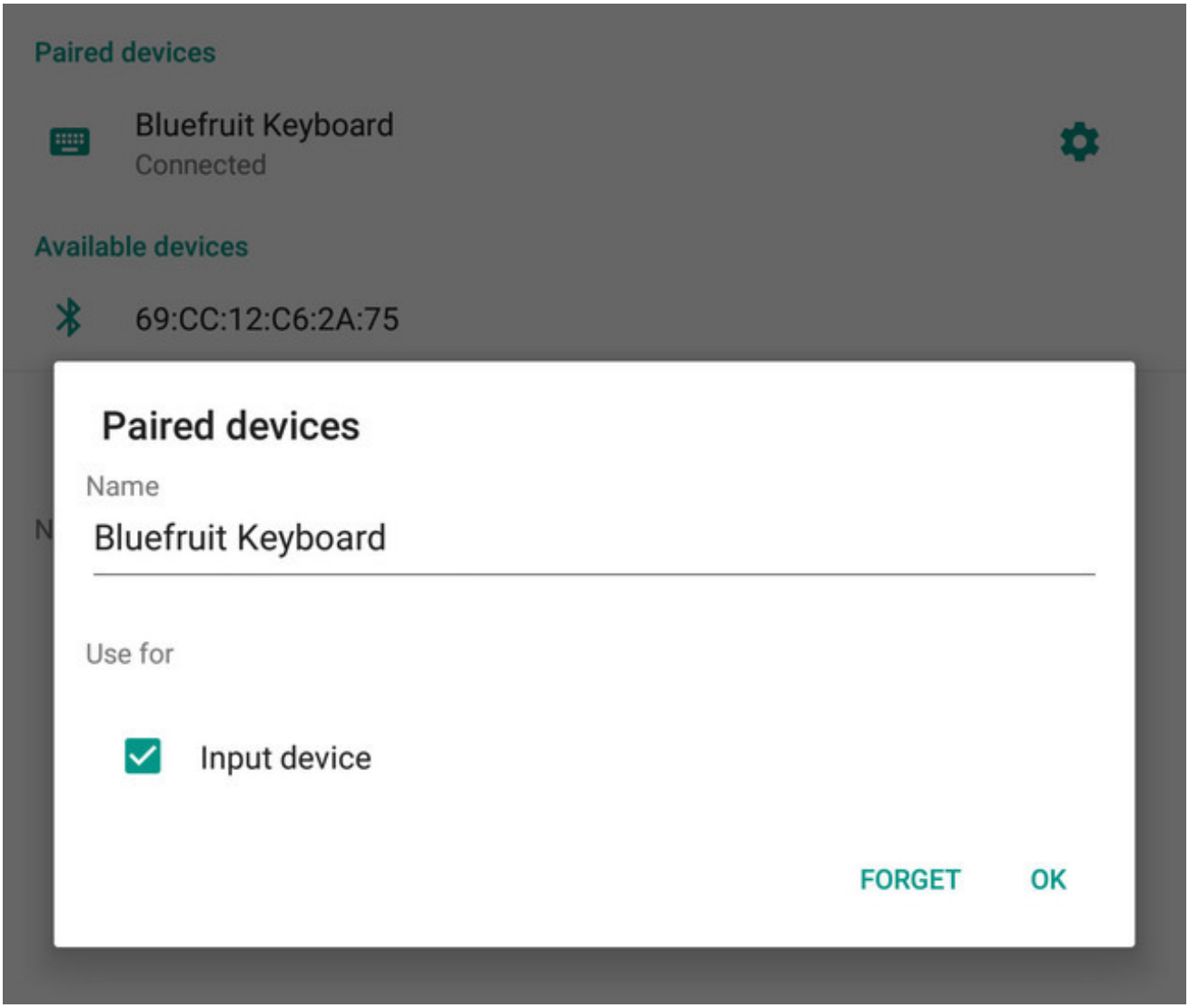


Inside the Bluetooth setting panel you should see the Bluefruit LE module advertising itself as **Bluefruit Keyboard** under the 'Available devices' list:



Tapping the device will start the bonding process, which should end with the Bluefruit Keyboard device being moved to a new 'Paired devices' list with 'Connected' written underneath the device name:

To delete the bonding information, click the gear icon to the right of the device name and the click the **Forget** button:

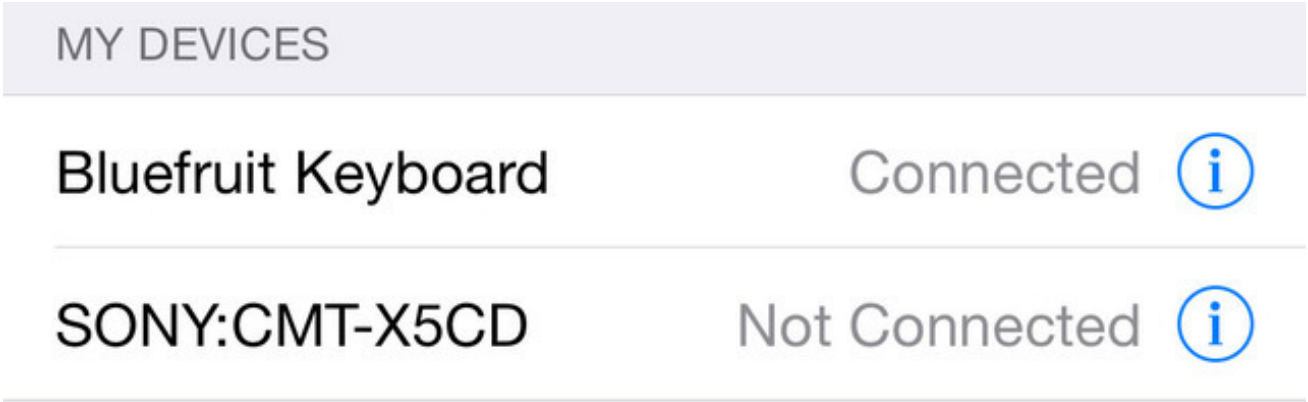


iOS

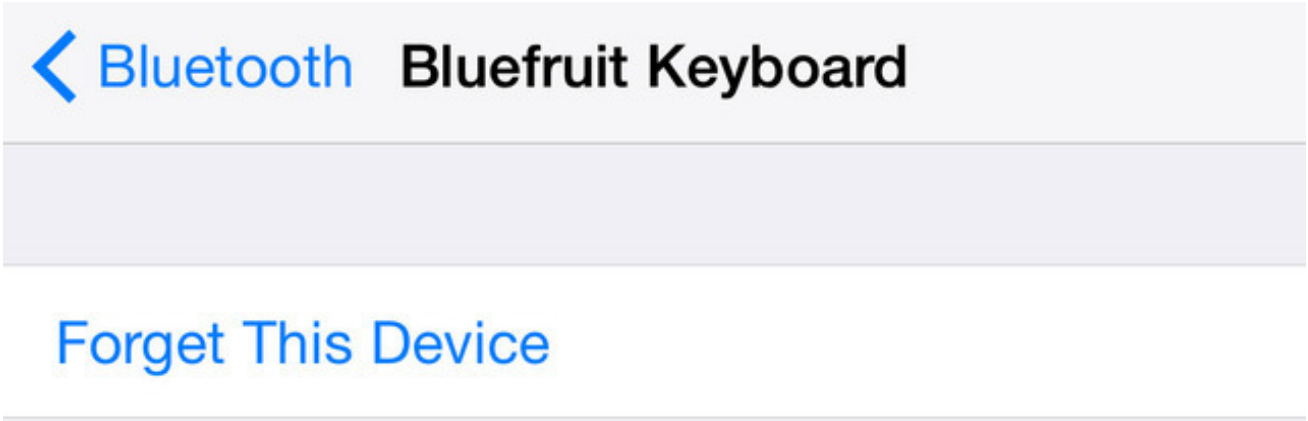
To bond the keyboard on an iOS device, go to the **Settings** application on your phone, and click the **Bluetooth** menu item.

The keyboard should appear under the **OTHER DEVICES** list:

Once the bonding process is complete the device will be moved to the **MY DEVICES** category, and you can start to use the Bluefruit LE module as a keyboard:

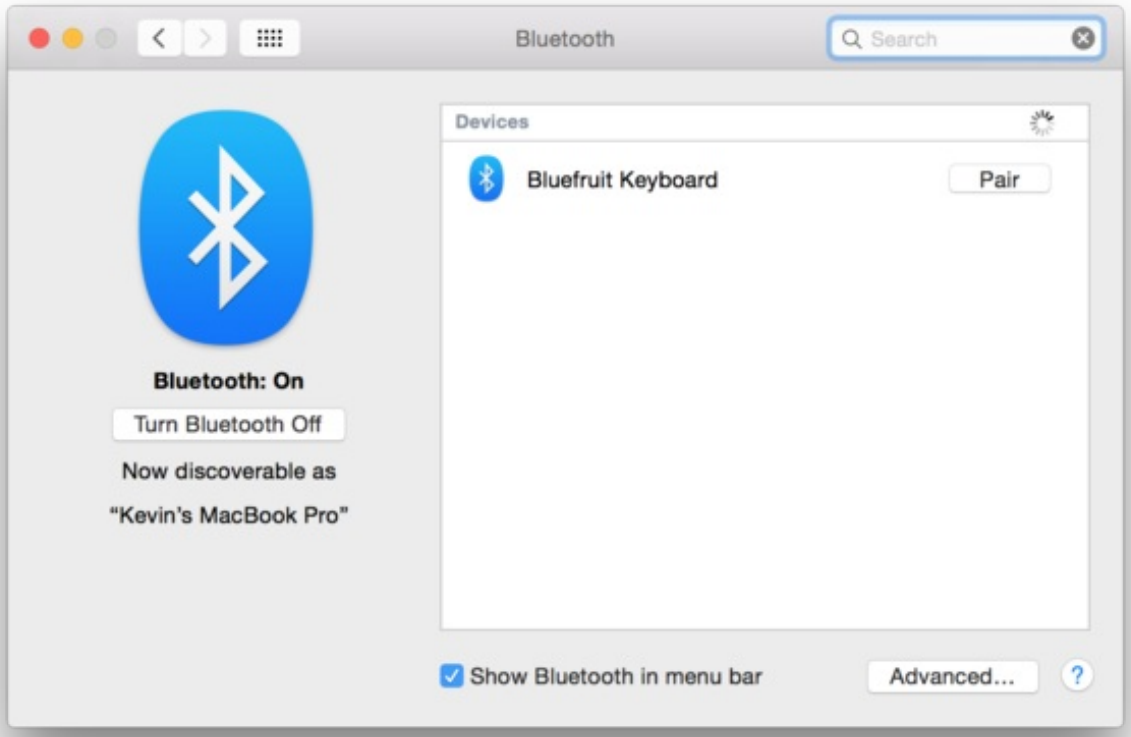


To unbond the device, click the 'info' icon and then select the **Forget this Device** option in the menu:

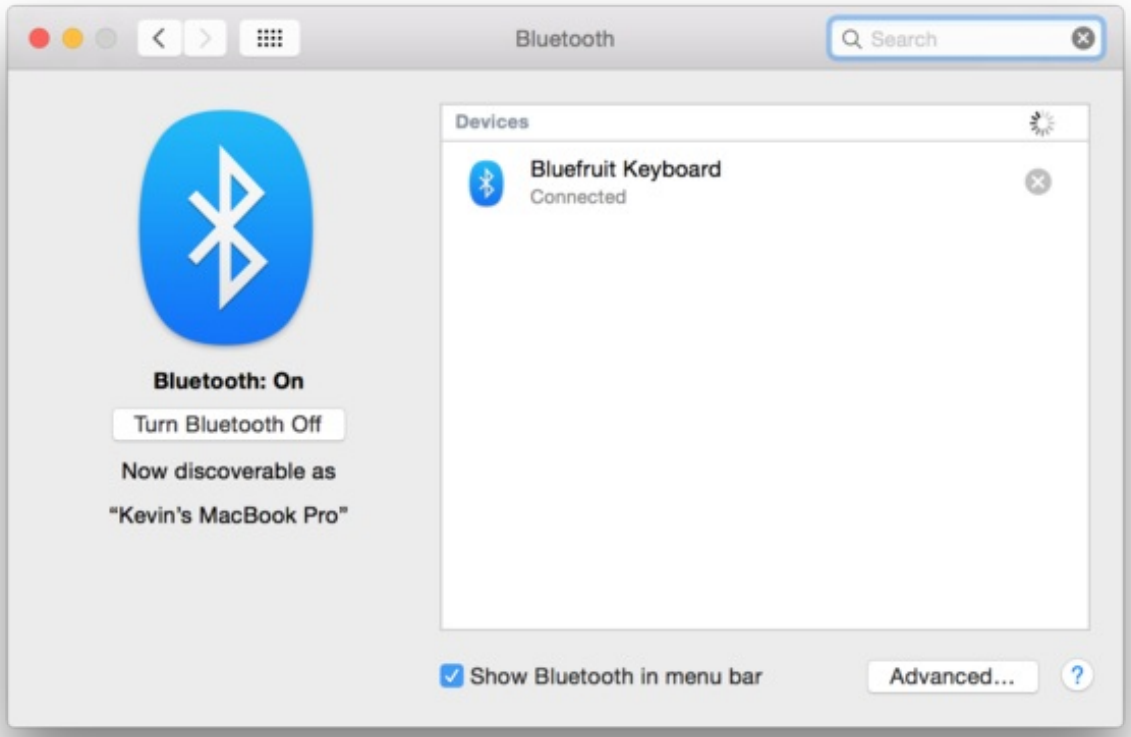


OS X

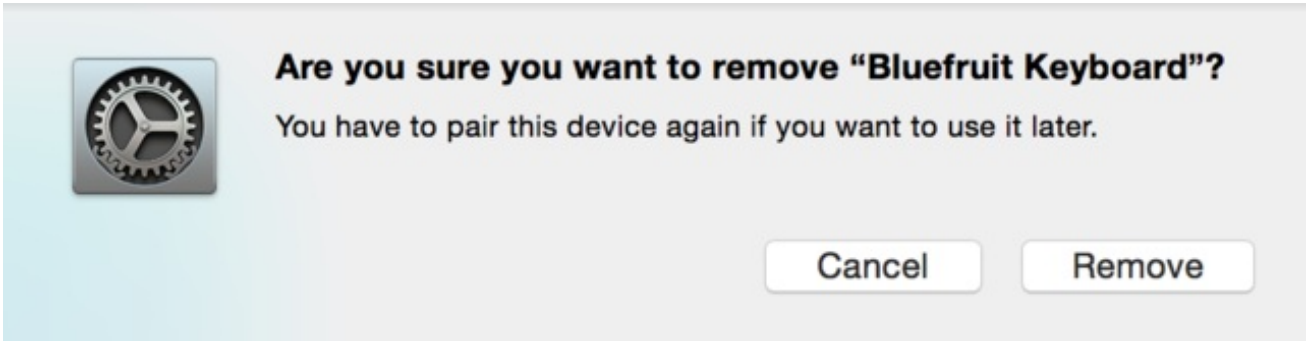
To bond the keyboard on an OS X device, go to the **Bluetooth Preferences** window and click the **Pair** button beside the **Bluefruit Keyboard** device generated by this example sketch:



To unbond the device once it has been paired, click the small 'x' icon beside **Bluefruit Keyboard**:



... and then click the **Remove** button when the confirmation dialogue box pops up:



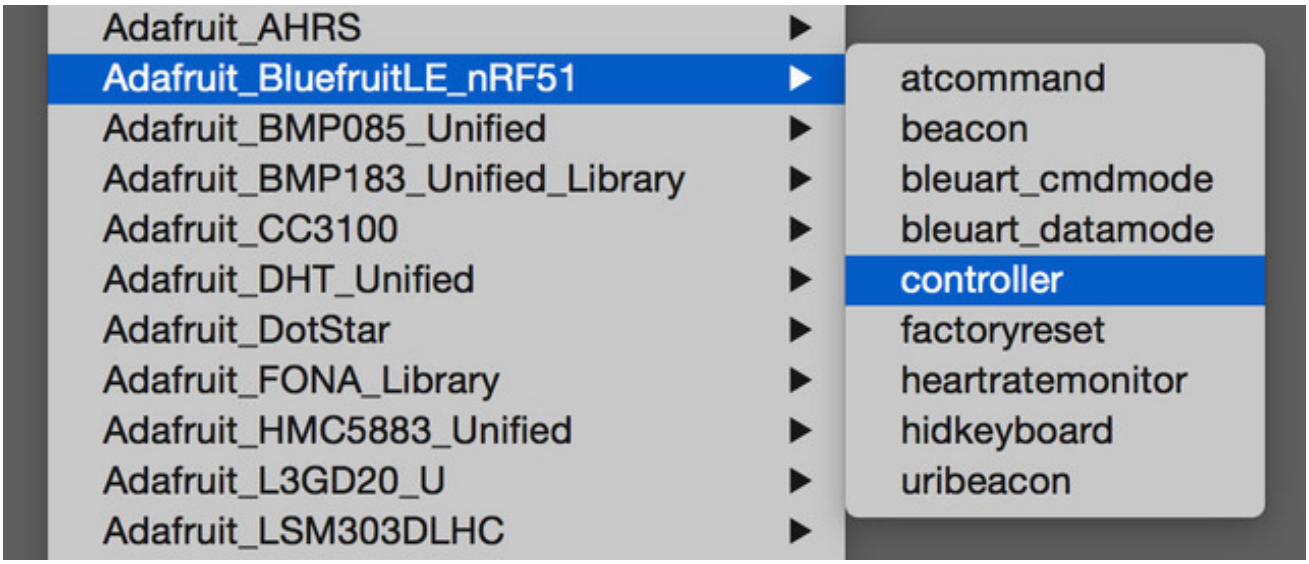
Controller

The **Controller** sketch allows you to turn your BLE-enabled iOS or Android device in a hand-held controller or an external data source, taking advantage of the wealth of sensors on your phone or tablet.

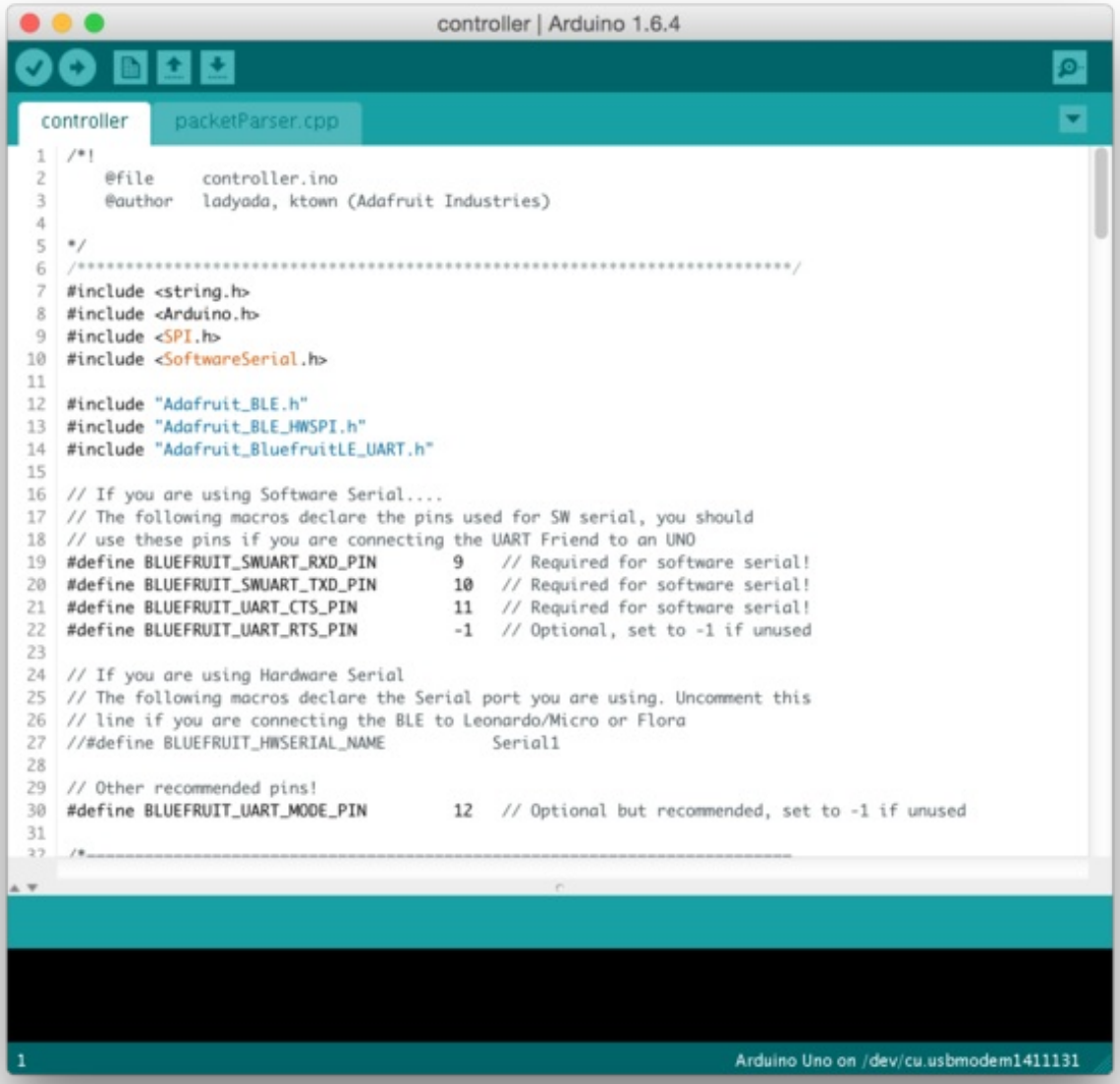
You can take accelerometer or quaternion data from your phone, and push it out to your Arduino via BLE, or get the latest GPS co-ordinates for your device without having to purchase (or power!) any external HW.

Opening the Sketch

To open the Controller sketch, click on the **File > Examples > Adafruit_BluefruitLE_nRF51** folder in the Arduino IDE and select **controller**:



This will open up a new instance of the example in the IDE, as shown below:



Wiring

This example can be used with different wiring setups such as **Hardware Serial** (Flora/Micro/Leonardo/Due) or **Software Serial** (UNO/Metro & other Atmega328's), or **SPI** hardware or software (any microcontroller with 5 pins)

First determine which you have. If you are using ...

Serial Wiring

Software Serial

For UNO/Metro/Pro Trinket & other Atmega328's! **Three pins are required:** RX, TX, and CTS. They can connect to any pins on the Arduino but they must be wired up.

```
#define BLUEFRUIT_SWUART_RXD_PIN      9   // Required for software serial!
#define BLUEFRUIT_SWUART_TXD_PIN      10  // Required for software serial!
#define BLUEFRUIT_UART_CTS_PIN        11  // Required for software serial!
```

Hardware Serial

For Flora/Micro/Leonardo/Due! You must connect up the RX and TX pins of the Bluefruit to your Arduino or microcontroller's hardware serial port, then uncomment the following line, to set the HW Serial port name

```
//#define BLUEFRUIT_HWSERIAL_NAME         Serial1
```

and:

```
//Adafruit_BluefruitLE_UART ble(BLUEFRUIT_HWSERIAL_NAME, BLUEFRUIT_UART_MODE_PIN);
```

Then comment or remove:

```
SoftwareSerial bluefruitSS = SoftwareSerial(BLUEFRUIT_SWUART_TXD_PIN, BLUEFRUIT_SWUART_RXD_P

Adafruit_BluefruitLE_UART ble(bluefruitSS, BLUEFRUIT_UART_MODE_PIN,
                      BLUEFRUIT_UART_CTS_PIN, BLUEFRUIT_UART_RTS_PIN);
```

Don't forget to also connect the CTS pin on the Bluefruit to ground if you are not using it! (The Flora has this already done)

Other Serial Pins

This sketch uses the **MODE** pin if its wired up, we recommend having it connected to digital #12 as indicated in the sketch. If you are using a Flora or otherwise don't want to wire up the Mode pin, set it to -1 in the code so that the sketch will use the +++ method to switch between Command and Data mode!

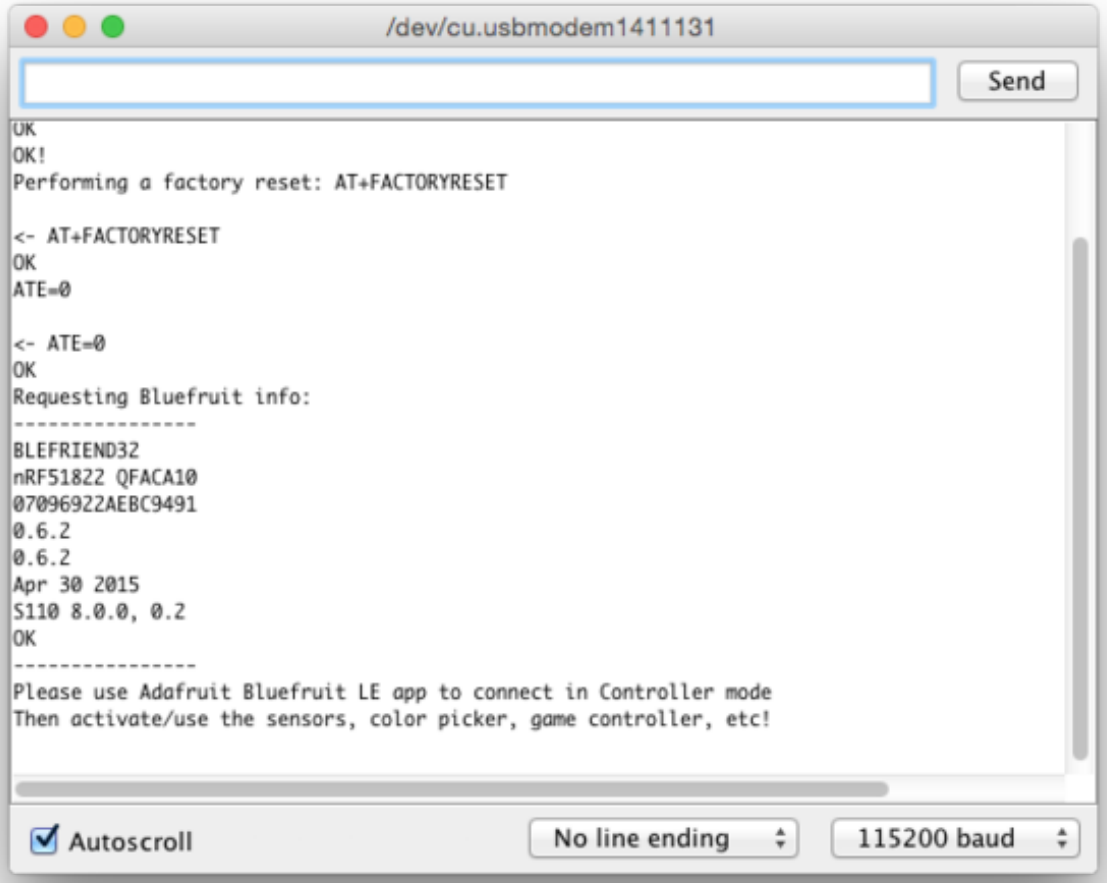
SPI Wiring

If using hardware SPI, connect SCK/MOSI/MISO to the hardware SPI port of your microcontroller. If

using software SPI, you can use any three pins. Then edit the config.h file to set up which pins you want to use. (http://adafru.it/fyg) SPI does not have flow control pins, or a MODE pin. The RST pin is optional but recommended if you can spare a pin.

Running the Sketch

Once you upload the sketch to your board (via the arrow-shaped upload icon), and the upload process has finished, open up the Serial Monitor via **Tools > Serial Monitor**, and make sure that the baud rate in the lower right-hand corner is set to **115200**:

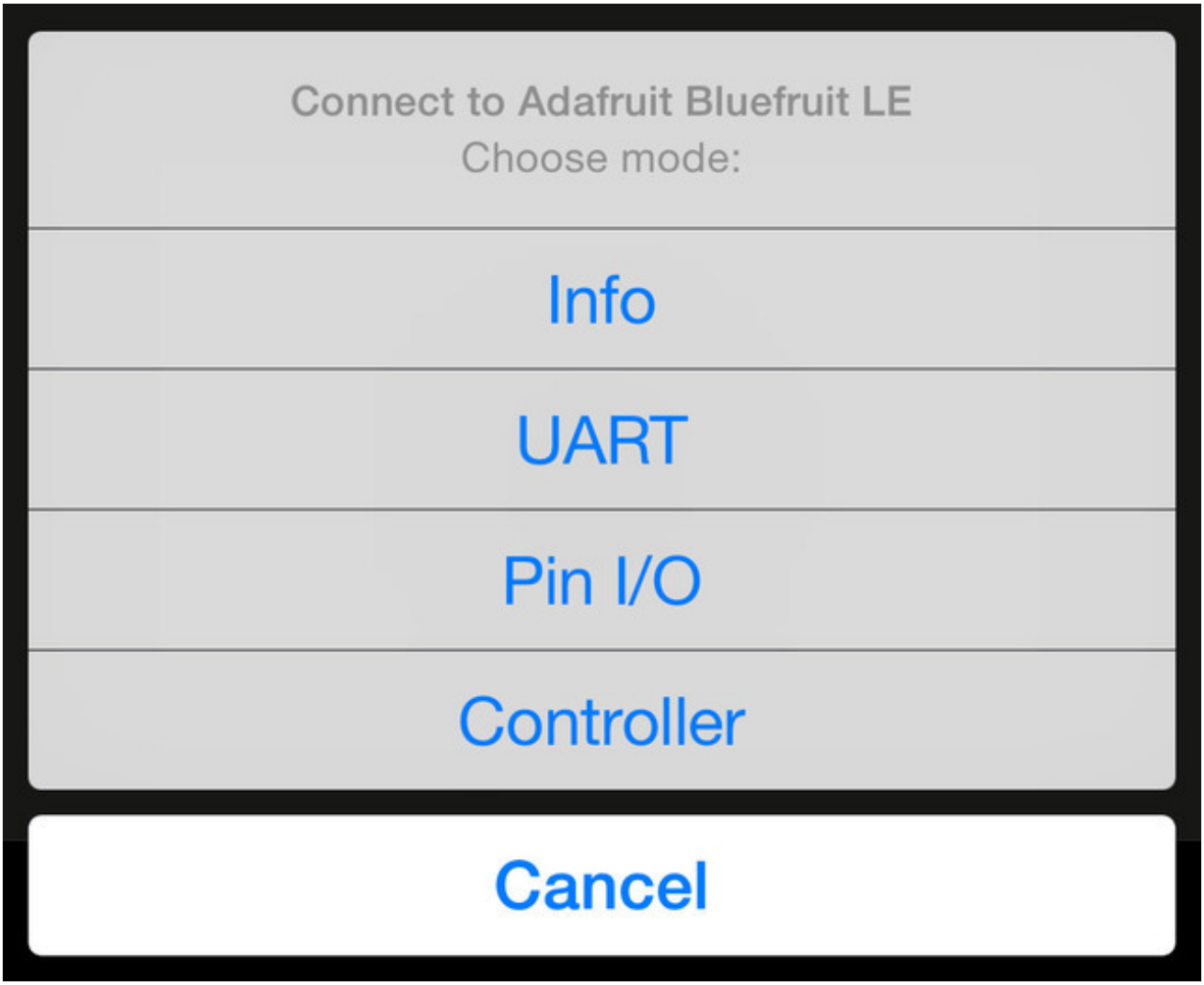


Using Bluefruit LE Connect in Controller Mode

Once the sketch is running you can open Adafruit's Bluefruit LE Connect application (available for Android (http://adafru.it/f4G) or iOS (http://adafru.it/f4H)) and use the **Controller** application to interact with the sketch. (If you're new to Bluefruit LE Connect, have a look at our dedicated Bluefruit LE Connect learning guide (http://adafru.it/faQ).)

On the welcome screen, select the **Adafruit Bluefruit LE** device from the list of BLE devices in range:

Then from the activity list select **Controller**:

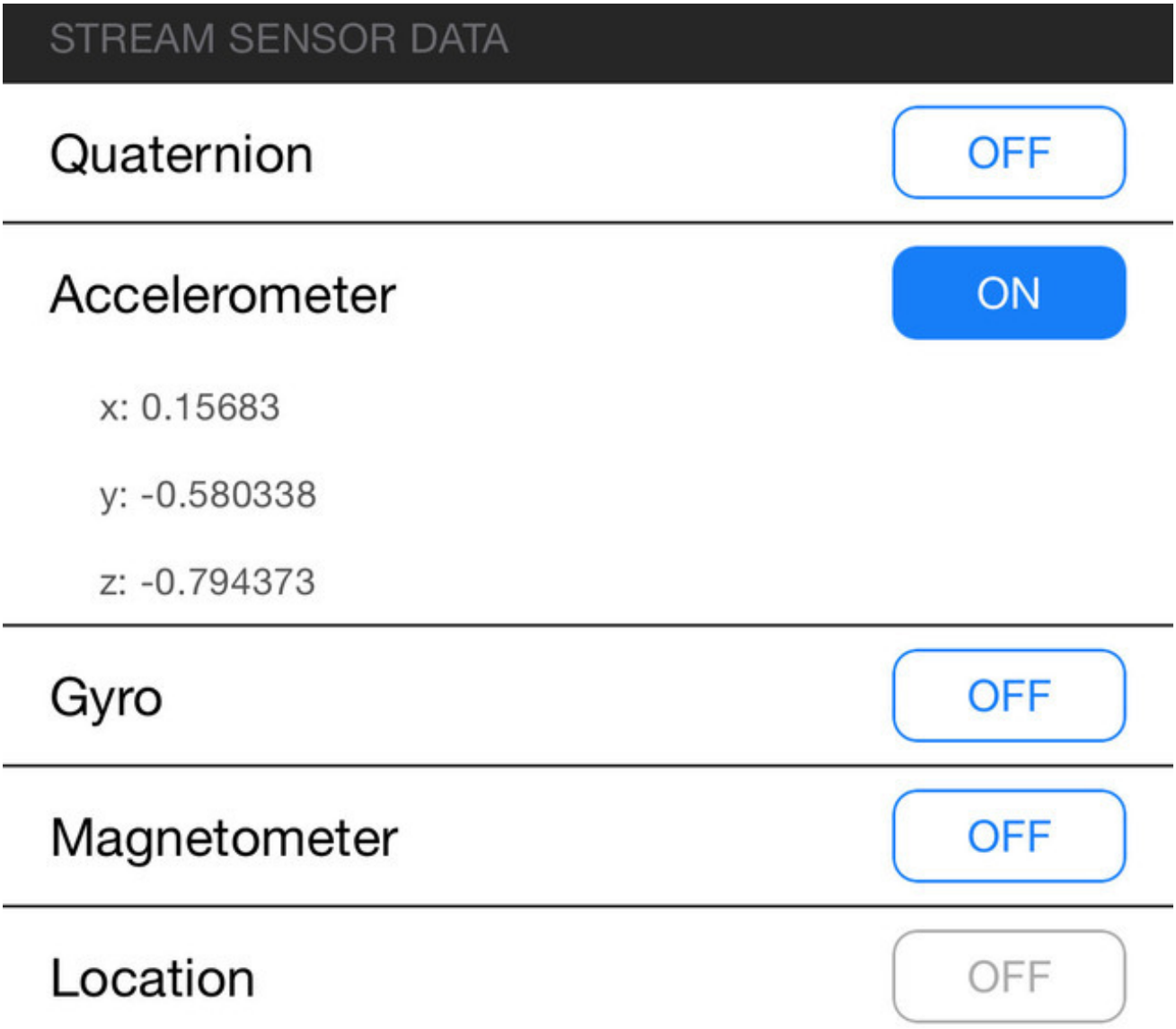


This will bring up a list of data points you can send from your phone or tablet to your Bluefruit LE module, by enabling or disabling the appropriate sensor(s).

Streaming Sensor Data

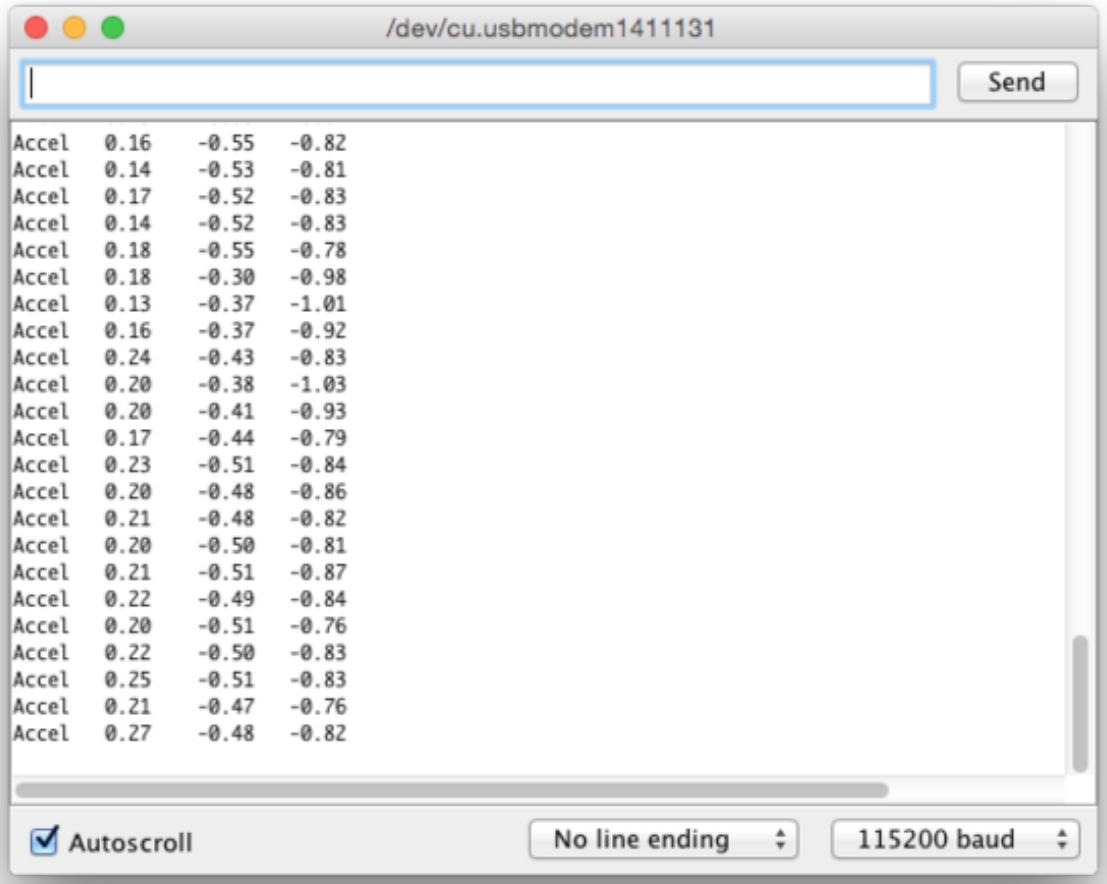
You can take Quaternion (absolute orientation), Accelerometer, Gyroscope, Magnetometer or GPS Location data from your phone and send it directly to your Arduino from the Controller activity.

By enabling the **Accelerometer** field, for example, you should see accelerometer data update in the app:



The data is parsed in the example sketch and output to the Serial Monitor as follows:

```
Accel 0.20 -0.51 -0.76
Accel 0.22 -0.50 -0.83
Accel 0.25 -0.51 -0.83
Accel 0.21 -0.47 -0.76
Accel 0.27 -0.48 -0.82
```

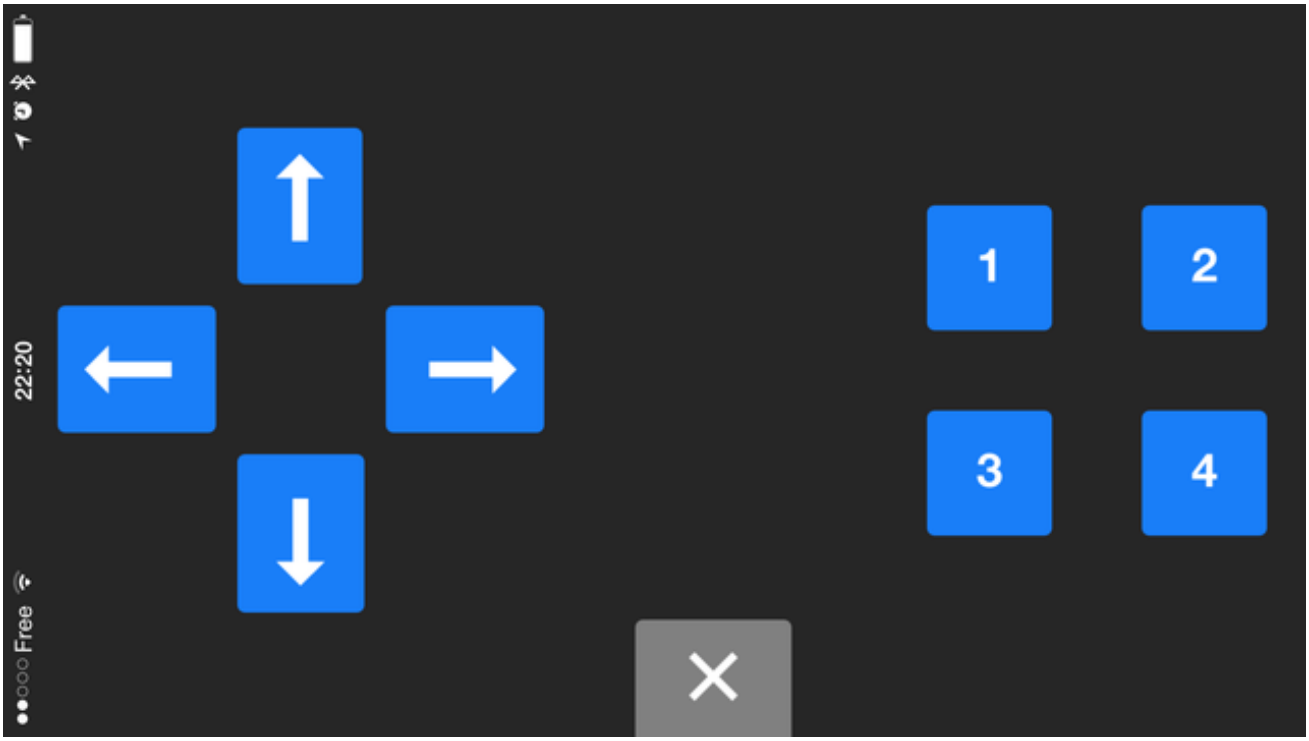


Note that even though we only print 2 decimal points, the values are received from the App as a full 4-byte floating point.

Control Pad Module

You can also use the **Control Pad Module** to capture button presses and releases by selecting the appropriate menu item:

This will bring up the Control Pad panel, shown below:

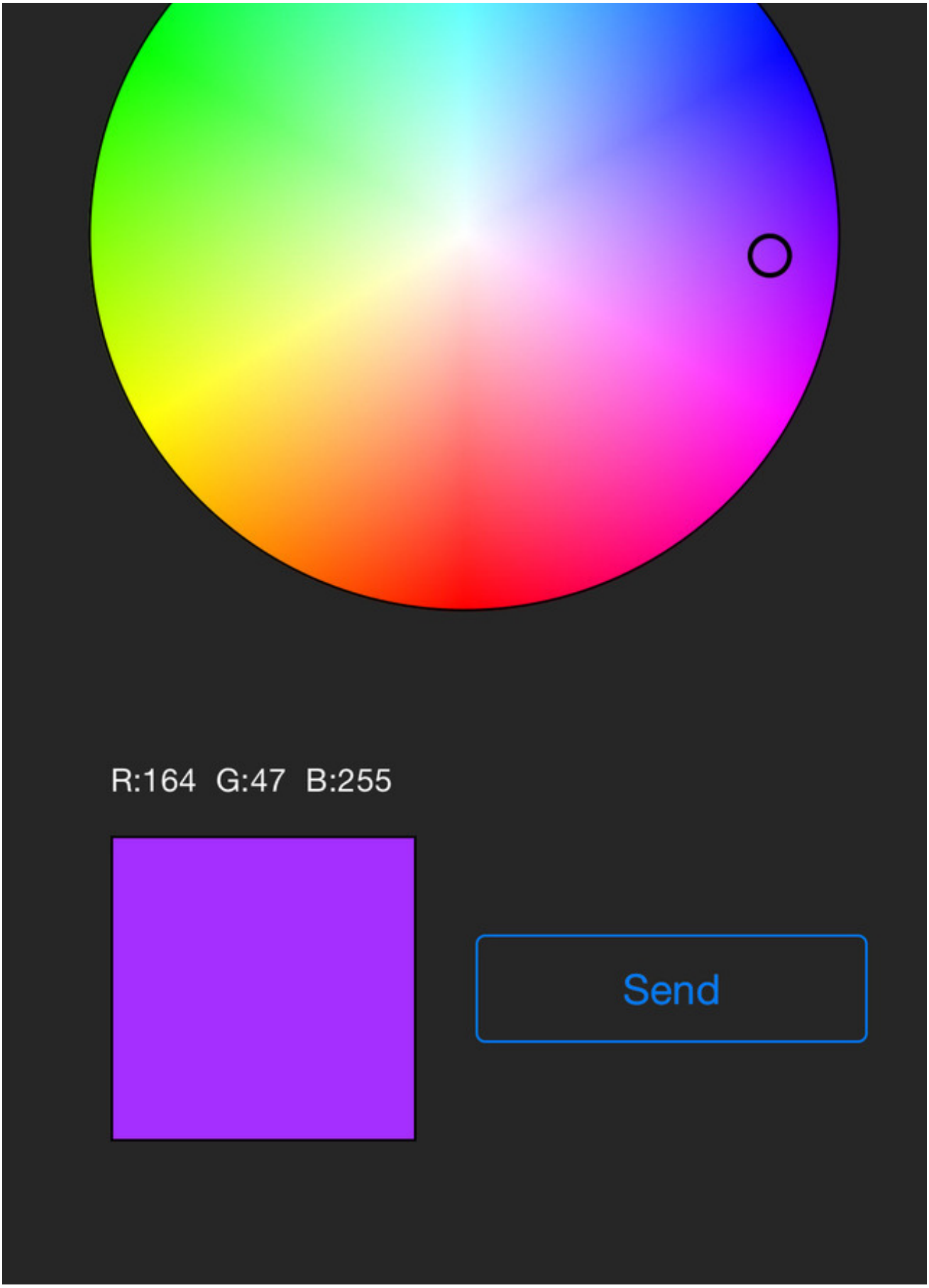


Button presses and releases will all be logged to the Serial Monitor with the ID of the button used:

```
Button 8 pressed
Button 8 released
Button 3 pressed
Button 3 released
```

Color Picker Module

You can also send RGB color data via the **Color Picker** module, which presents the following color selection dialogue:


R:164 G:47 B:255
Send

This will give you Hexadecimal color data in the following format:

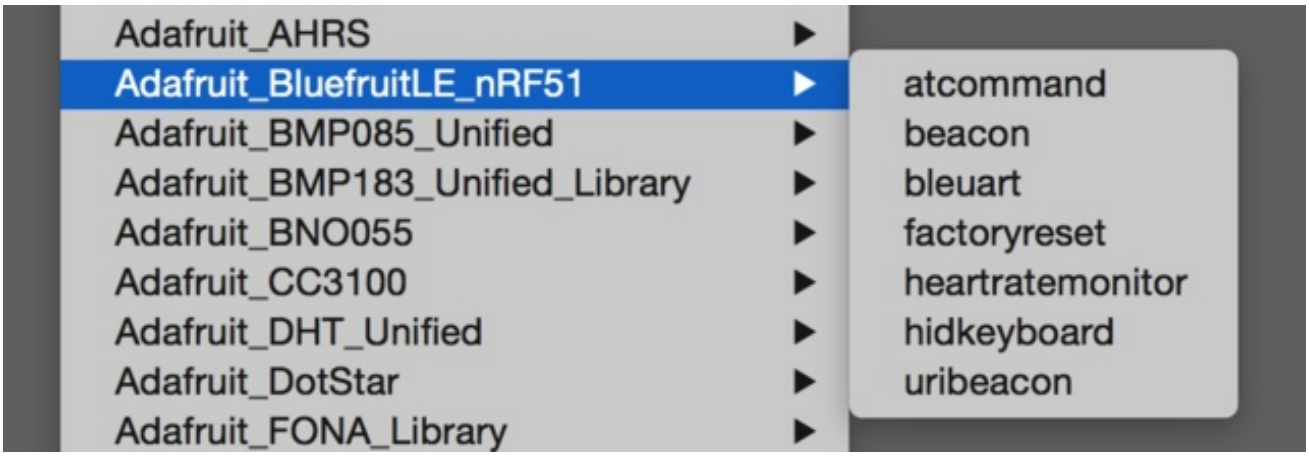
```
RGB #A42FFF
```

HeartRateMonitor

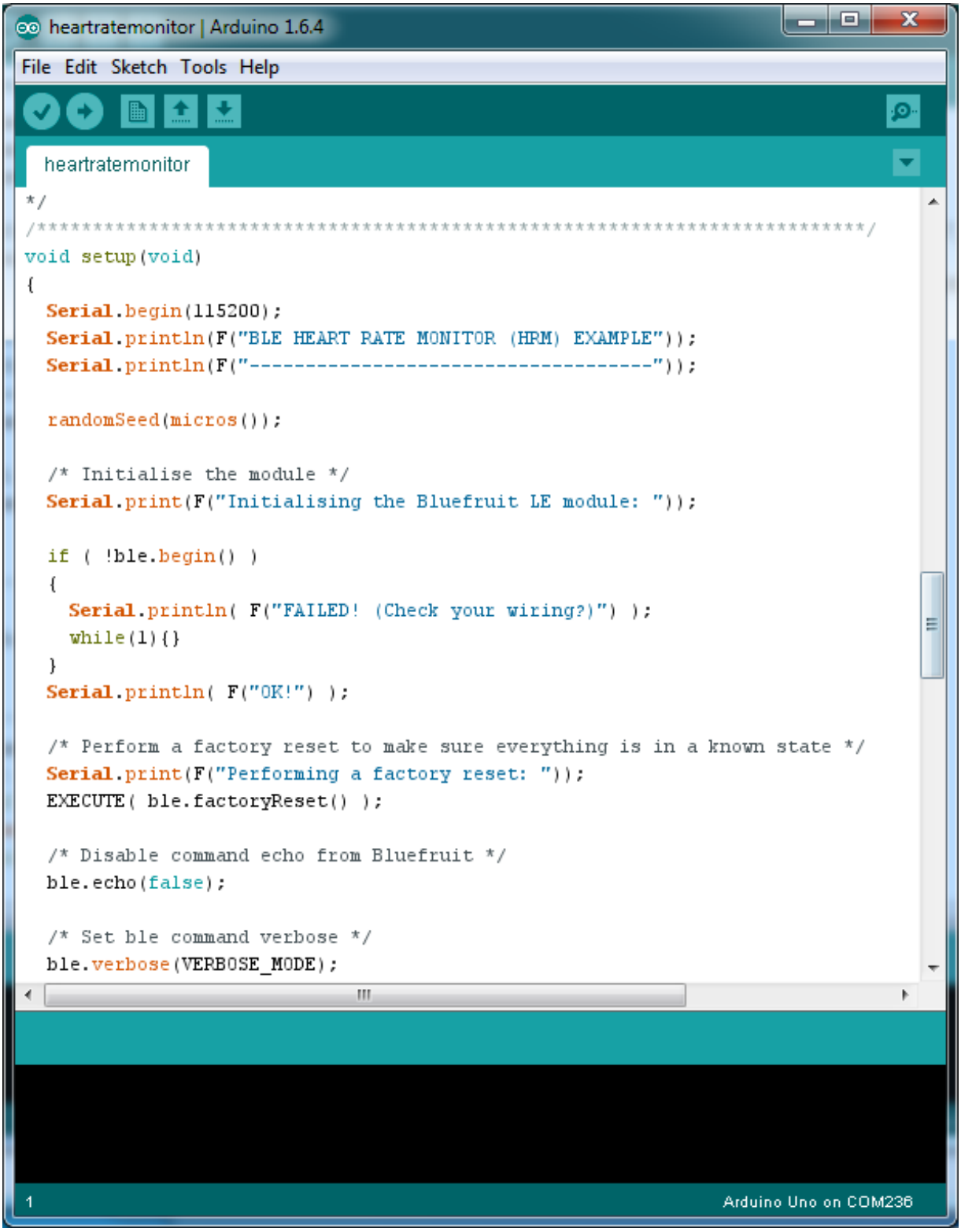
The **HeartRateMonitor** example allows you to define a new GATT Service and associated GATT Characteristics, and update the characteristic values using standard AT commands.

Opening the Sketch

To open the ATCommand sketch, click on the **File > Examples > Adafruit_BluefruitLE_nRF51** folder in the Arduino IDE and select **heartratemonitor**:



This will open up a new instance of the example in the IDE, as shown below:



Wiring

This example can be used with different wiring setups such as **Hardware Serial** (Flora/Micro/Leonardo/Due) or **Software Serial** (UNO/Metro & other Atmega328's), or **SPI** hardware or software (any microcontroller with 5 pins)

First determine which you have. If you are using ...

Serial Wiring

Software Serial

For UNO/Metro/Pro Trinket & other Atmega328's! **Four pins are required:** RX, TX, RTS and CTS. They can connect to any pins on the Arduino but they must be wired up!

```
#define BLUEFRUIT_SWUART_RXD_PIN      9   // Required for software serial!
#define BLUEFRUIT_SWUART_TXD_PIN      10  // Required for software serial!
#define BLUEFRUIT_UART_CTS_PIN        11  // Required for software serial!
#define BLUEFRUIT_UART_RTS_PIN        8   // Strongly recommended with this demo!
```

Hardware Serial

For Flora/Micro/Leonardo/Due! You must connect up the RX and TX pins of the Bluefruit to your Arduino or microcontroller's hardware serial port, then uncomment the following line, to set the HW Serial port name

```
//#define BLUEFRUIT_HWSERIAL_NAME          Serial1
```

and..

```
/* ...or hardware serial, requires only the RTS pin to keep the Bluefruit happy. Uncomment this line */
//Adafruit_BluefruitLE_UART ble(BLUEFRUIT_HWSERIAL_NAME, BLUEFRUIT_UART_MODE_PIN, BLUEFRU
```

Then comment or remove:

```
SoftwareSerial bluefruitSS = SoftwareSerial(BLUEFRUIT_SWUART_TXD_PIN, BLUEFRUIT_SWUART_RXD_P

Adafruit_BluefruitLE_UART ble(bluefruitSS, BLUEFRUIT_UART_MODE_PIN,
                      BLUEFRUIT_UART_CTS_PIN, BLUEFRUIT_UART_RTS_PIN);
```

Other Serial Pins

This demo uses some long data transfer strings, so we recommend defining and connecting both CTS and RTS, even if you are using hardware serial.

If you are using a Flora or just dont want to connect CTS or RTS, set the pin #define's to -1 and **Don't forget to also connect the CTS pin on the Bluefruit to ground!** (The Flora has this already done)

If you are using RTS and CTS, you can remove this line which will slow down the data transmission

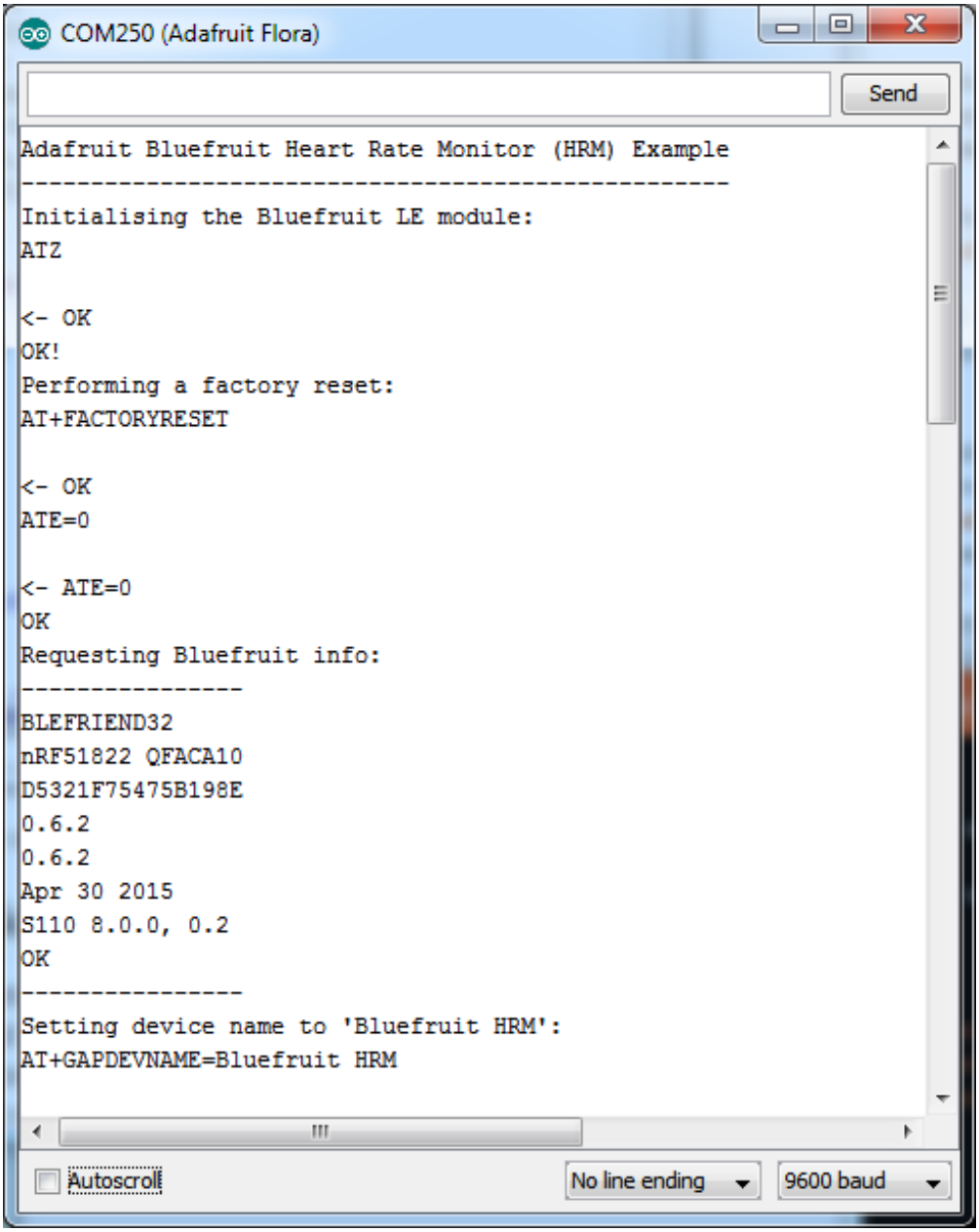
```
// this line is particularly required for Flora, but is a good idea
  // anyways for the super long lines ahead!
  ble.setInterCharWriteDelay(5); // 5 ms
```

SPI Wiring

If using hardware SPI, connect SCK/MOSI/MISO to the hardware SPI port of your microcontroller. If using software SPI, you can use any three pins. Then edit the config.h file to set up which pins you want to use. (http://adafru.it/fyg) SPI does not have flow control pins, or a MODE pin. The RST pin is optional but recommended if you can spare a pin.

Running the Sketch

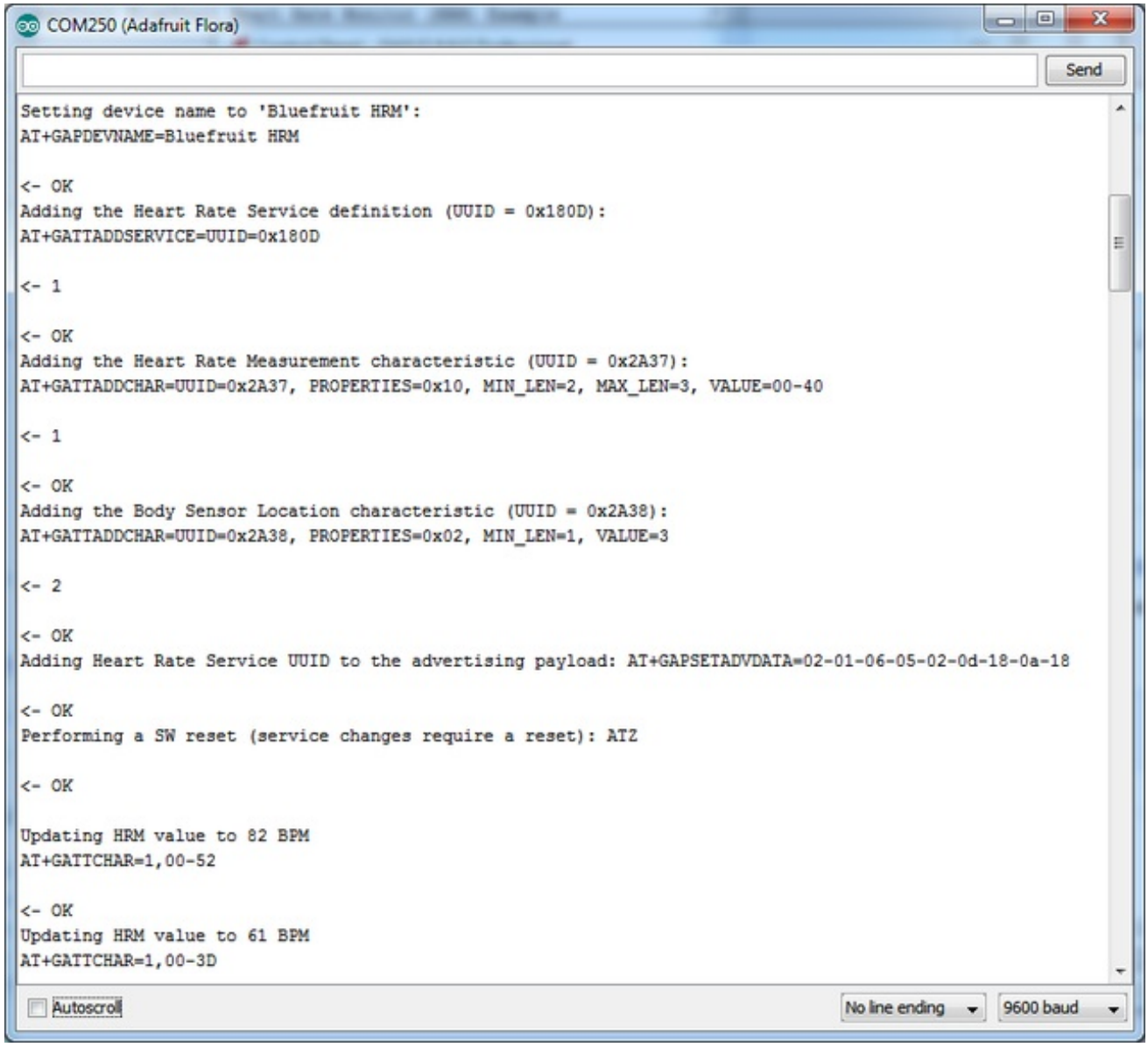
Once you upload the sketch to your board (via the arrow-shaped upload icon), and the upload process has finished, open up the Serial Monitor via **Tools > Serial Monitor**, and make sure that the baud rate in the lower right-hand corner is set to **115200**:


COM250 (Adafruit Flora)
Send
Adafruit Bluefruit Heart Rate Monitor (HRM) Example

Initialising the Bluefruit LE module:
ATZ
<- OK
OK!
Performing a factory reset:
AT+FACTORYRESET
<- OK
ATE=0
<- ATE=0
OK
Requesting Bluefruit info:

BLEFRIEND32
nRF51822 QFACA10
D5321F75475B198E
0.6.2
0.6.2
Apr 30 2015
S110 8.0.0, 0.2
OK

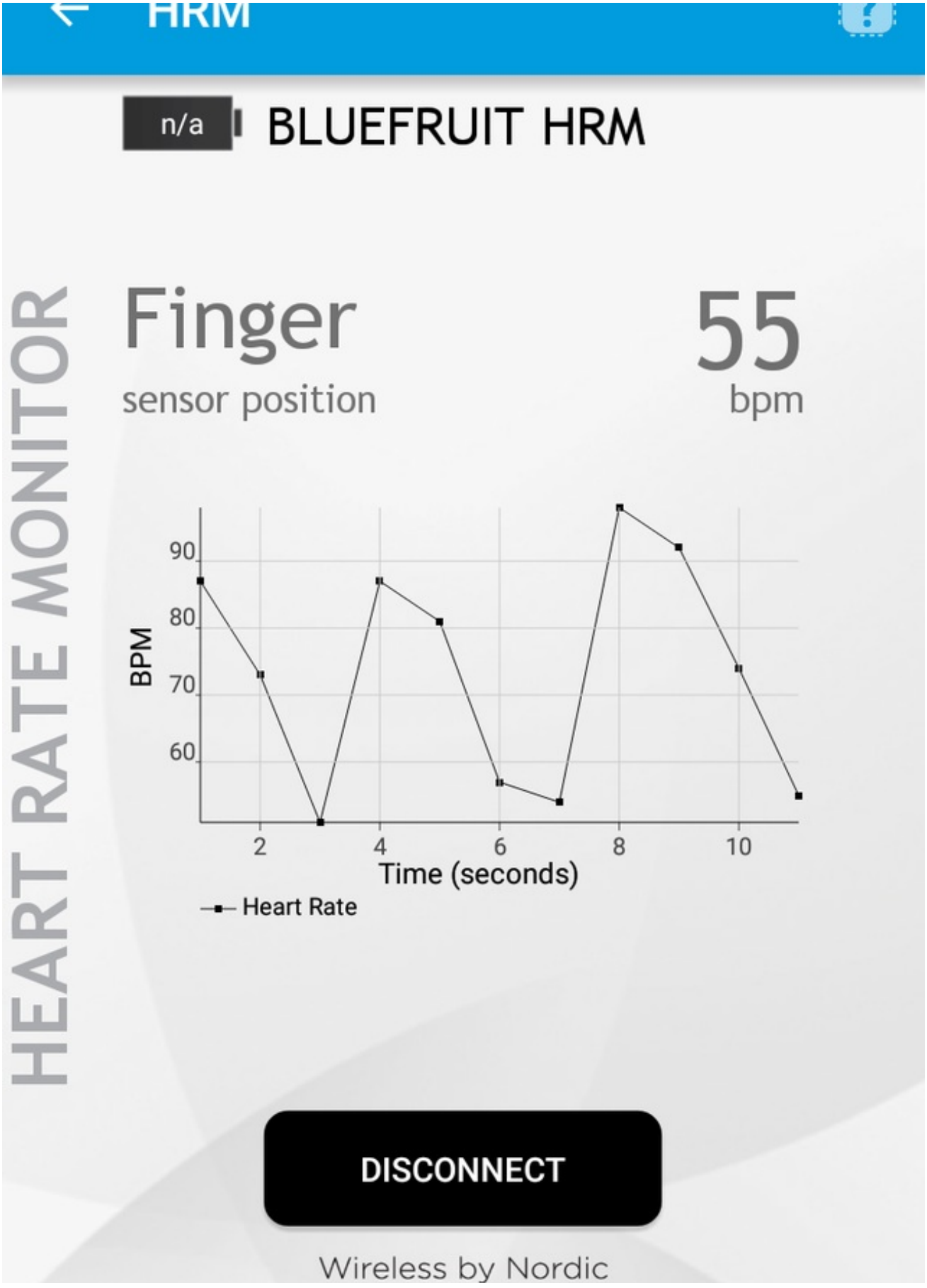
Setting device name to 'Bluefruit HRM':
AT+GAPDEVNAME=Bluefruit HRM
Autoscroll
No line ending
9600 baud

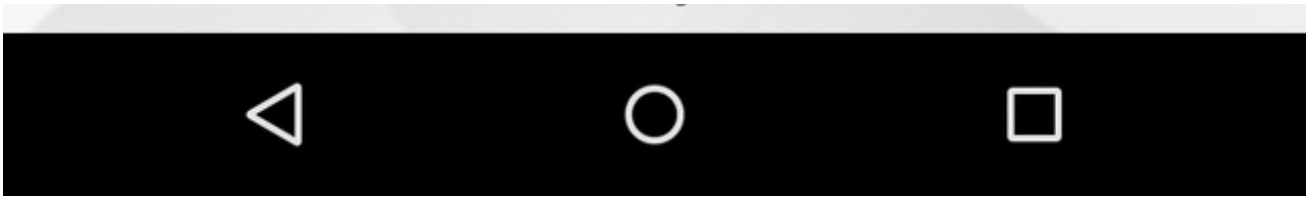


If you open up an application on your mobile device or laptop that support the standard Heart Rate Monitor Service (http://adafru.it/f4I), you should be able to see the heart rate being updated in sync with the changes seen in the Serial Monitor:

nRF Toolbox HRM Example

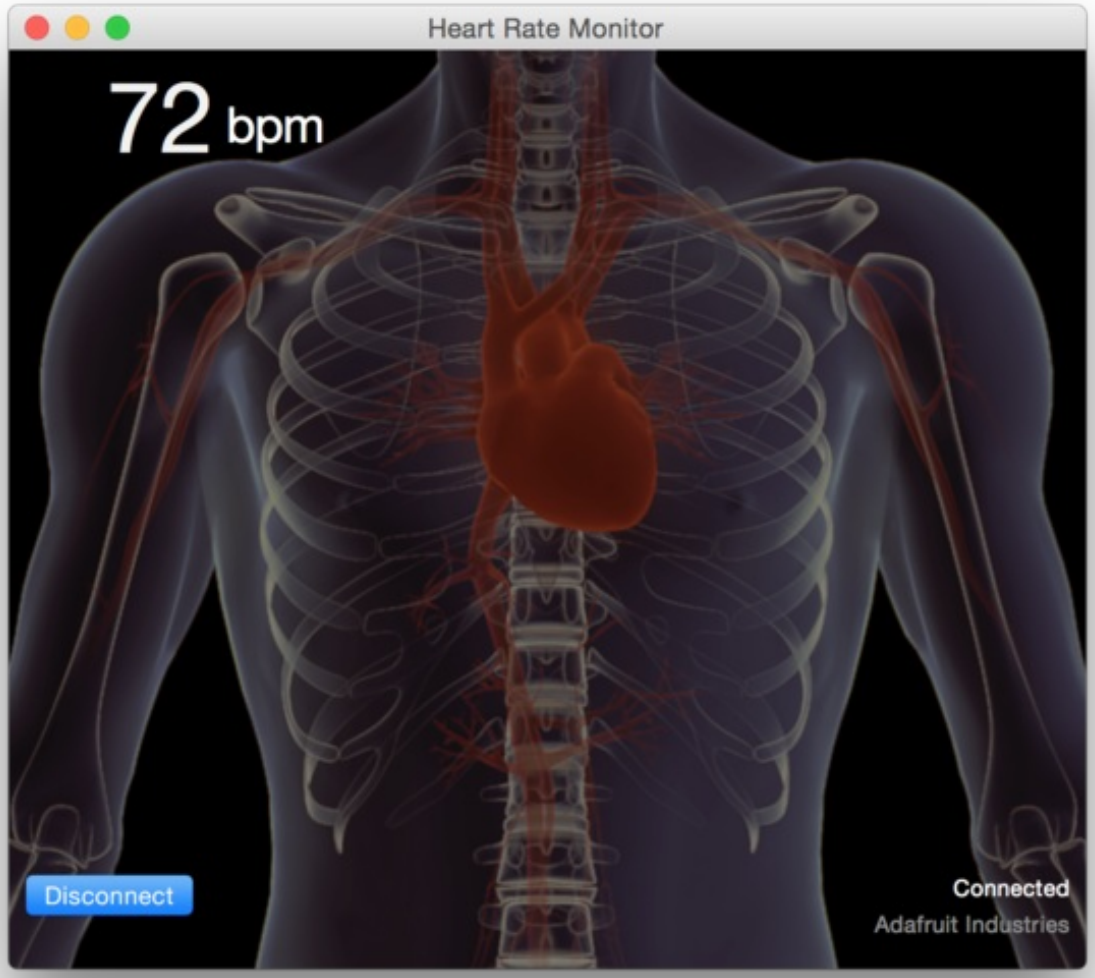
The image below is a screenshot from the free nRF Toolbox (http://adafru.it/e9M) application from Nordic on Android (also available on iOS (http://adafru.it/f4J)), showing the incoming Heart Rate Monitor data:


HRM
n/a
BLUEFRUIT HRM
HEART RATE MONITOR
Finger
sensor position
55
bpm
BPM
90
80
70
60
2
4
6
8
10
Time (seconds)
Heart Rate
DISCONNECT
Wireless by Nordic



CoreBluetooth HRM Example

The image below is from a freely available CoreBluetooth sample application (http://adafru.it/f4K) from Apple showing how to work with Bluetooth Low Energy services and characteristics:

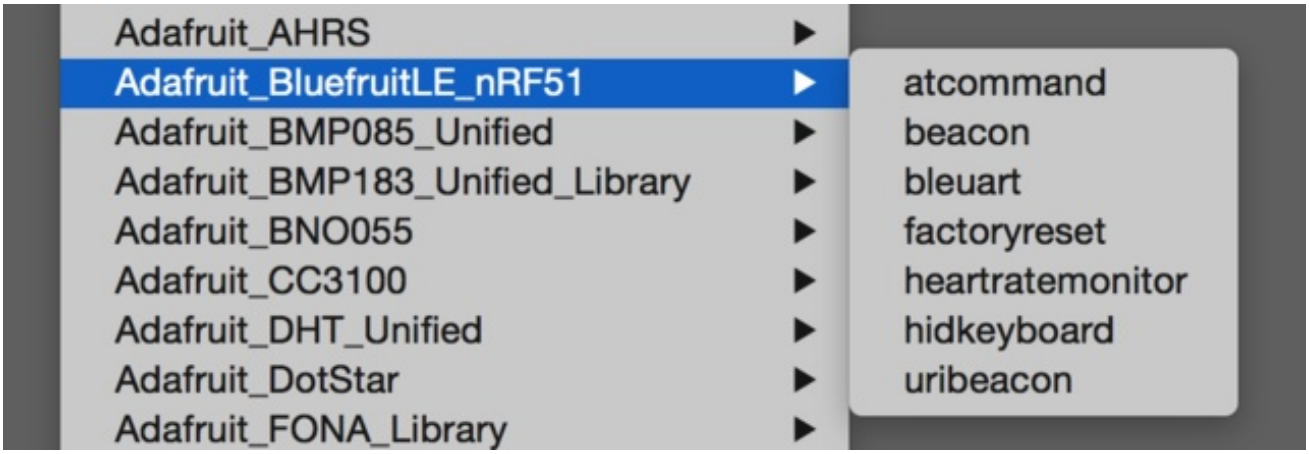


UriBeacon

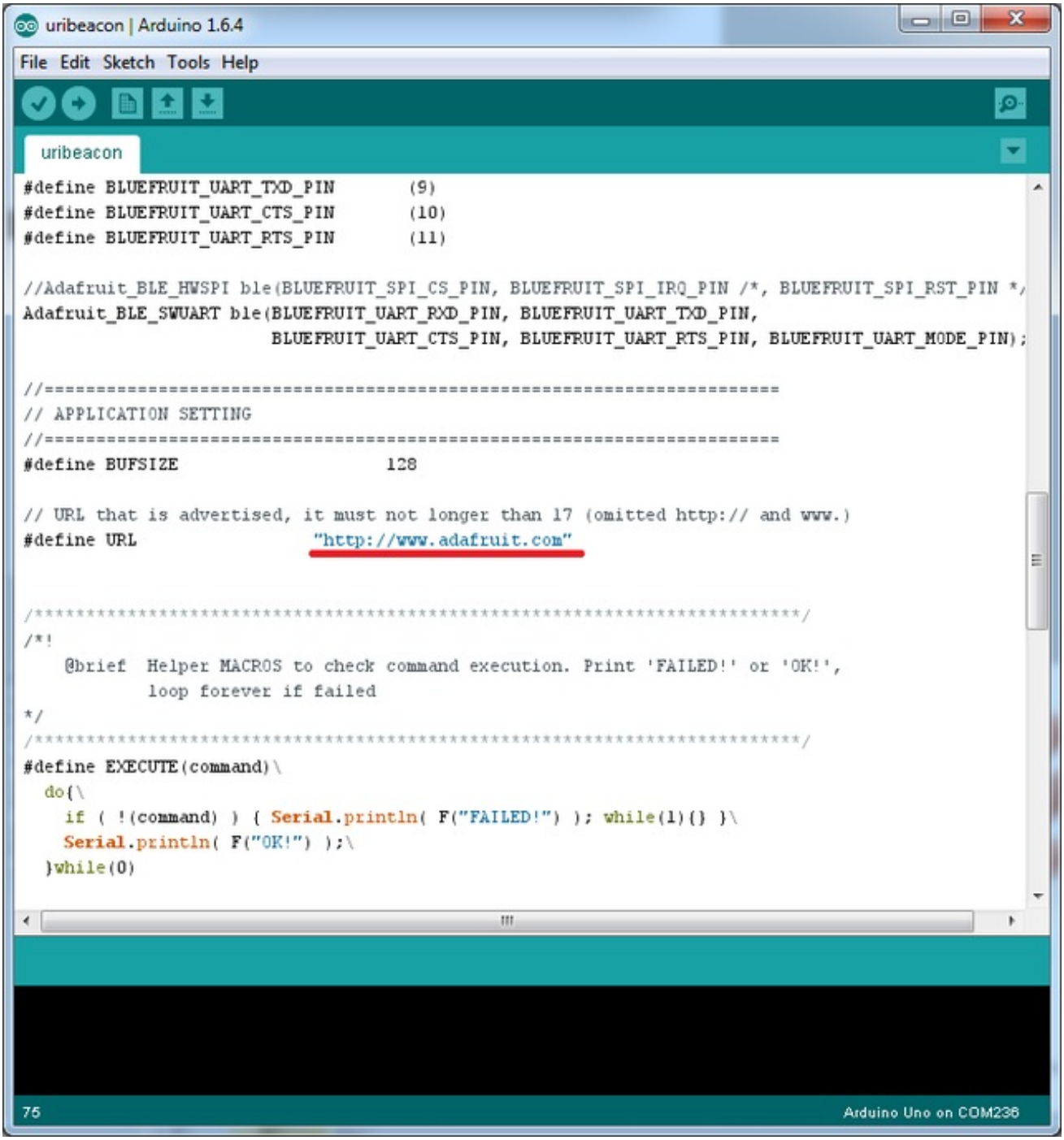
The **UriBeacon** example shows you how to use the built-in UriBeacon AT commands to configure the Bluefruit LE module as a UriBeacon advertiser, following Google's Physical Web UriBeacon (http://adafru.it/edk) specification.

Opening the Sketch

To open the ATCommand sketch, click on the **File > Examples > Adafruit_BluefruitLE_nRF51** folder in the Arduino IDE and select **uribeacon**:



This will open up a new instance of the example in the IDE, as shown below. You can edit the URL that the beacon will point to, from the default **http://www.adafruit.com** or just upload as is to test



Wiring

This example can be used with different wiring setups such as **Hardware Serial** (Flora/Micro/Leonardo/Due) or **Software Serial** (UNO/Metro & other Atmega328's), or **SPI** hardware or software (any microcontroller with 5 pins)

First determine which you have. If you are using ...

Serial Wiring

Software Serial

For UNO/Metro/Pro Trinket & other Atmega328's! **Three pins are required:** RX, TX, and CTS. They can connect to any pins on the Arduino but they must be wired up.

```
#define BLUEFRUIT_SWUART_RXD_PIN      9   // Required for software serial!
#define BLUEFRUIT_SWUART_TXD_PIN      10  // Required for software serial!
#define BLUEFRUIT_UART_CTS_PIN        11  // Required for software serial!
```

Hardware Serial

For Flora/Micro/Leonardo/Due! You must connect up the RX and TX pins of the Bluefruit to your Arduino or microcontroller's hardware serial port, then uncomment the following line, to set the HW Serial port name

```
//#define BLUEFRUIT_HWSERIAL_NAME          Serial1
```

and:

```
//Adafruit_BluefruitLE_UART ble(BLUEFRUIT_HWSERIAL_NAME, BLUEFRUIT_UART_MODE_PIN);
```

Then comment or remove:

```
SoftwareSerial bluefruitSS = SoftwareSerial(BLUEFRUIT_SWUART_TXD_PIN, BLUEFRUIT_SWUART_RXD_P

Adafruit_BluefruitLE_UART ble(bluefruitSS, BLUEFRUIT_UART_MODE_PIN,
                      BLUEFRUIT_UART_CTS_PIN, BLUEFRUIT_UART_RTS_PIN);
```

Don't forget to also connect the CTS pin on the Bluefruit to ground if you are not using it! (The Flora has this already done)

Other Serial Pins

This sketch does not use any other pins on the Bluefruit

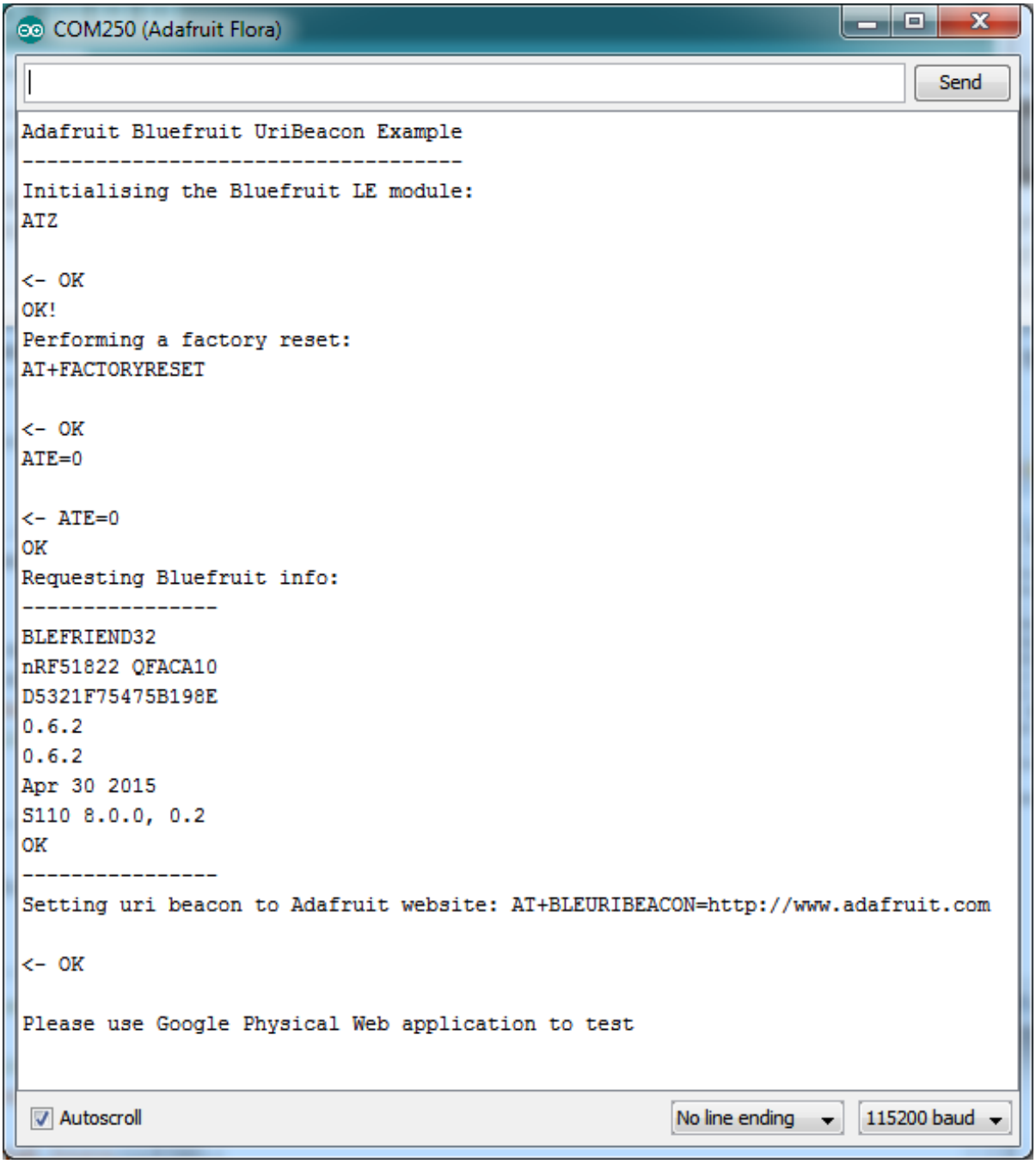
SPI Wiring

If using hardware SPI, connect SCK/MOSI/MISO to the hardware SPI port of your microcontroller. If using software SPI, you can use any three pins. Then edit the config.h file to set up which pins you want to use. (http://adafru.it/fyg) SPI does not have flow control pins, or a MODE pin. The RST pin is

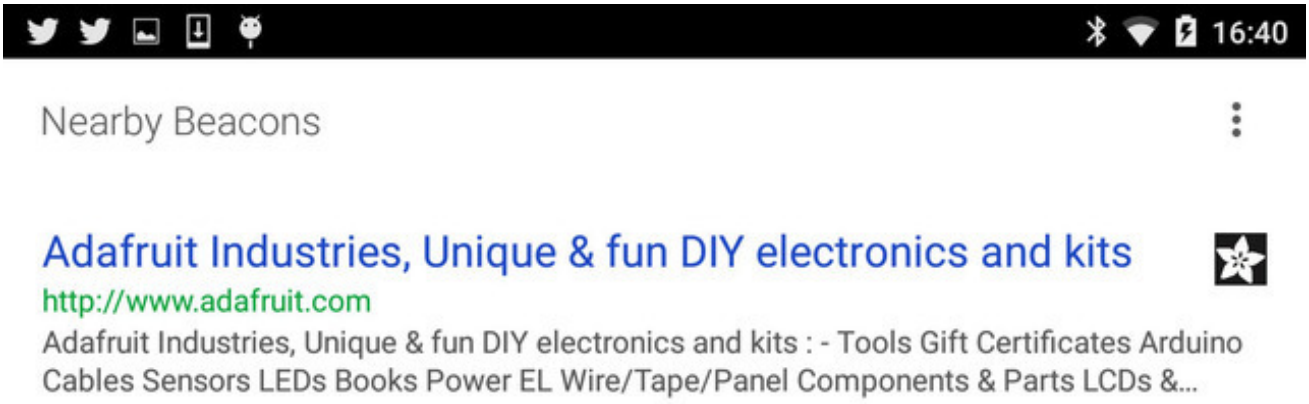
optional but recommended if you can spare a pin.

Running the Sketch

Once you upload the sketch to your board (via the arrow-shaped upload icon), and the upload process has finished, open up the Serial Monitor via **Tools > Serial Monitor**, and make sure that the baud rate in the lower right-hand corner is set to **115200**:



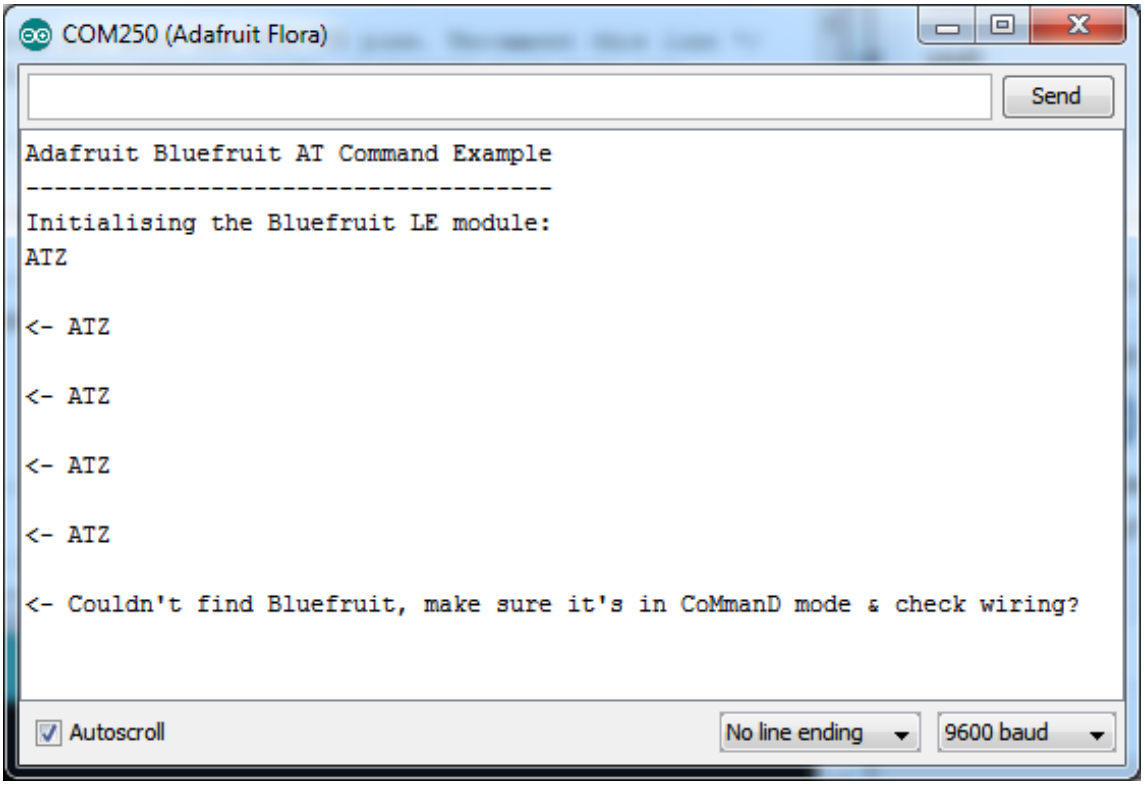
At this point you can open the Physical Web Application for Android (http://adafru.it/edi) or for iOS (http://adafru.it/edj), and you should see a link advertising Adafruit's website:


16:40
Nearby Beacons
Adafruit Industries, Unique & fun DIY electronics and kits
http://www.adafruit.com
Adafruit Industries, Unique & fun DIY electronics and kits : - Tools Gift Certificates Arduino Cables Sensors LEDs Books Power EL Wire/Tape/Panel Components & Parts LCDs &...

HALP!

I can't seem to "Find" the Bluefruit LE!

Getting something like this?



For UART/Serial Bluefruits:

- Check you have the **MODE** switch in CMD and the MODE pin not wired to anything if it isnt used!
- If you are trying to control the **MODE** from your micro, make sure you set the MODE pin in the sketch
- Make sure you have **RXI** and **TXO** wired right! They are often swapped by accident
- Make sure **CTS** is tied to GND if you are using hardware serial and not using CTS
- Check the MODE red LED, is it blinking? If its blinking continuously, you might be in DFU mode, power cycle the module!
- If you are using Hardware Serial/Software Serial make sure you know which one and have that set up

If using SPI Bluefruit:

- Make sure you have all 5 (or 6) wires connected properly.
- If using hardware SPI, you need to make sure you're connected to the hardware SPI port, which differs depending on the main chipset.

When using with Flora/Due/Leonardo/Micro the examples dont run?

We add a special line to **setup()** to make it so the Arduino will halt until it sees you've connected over the Serial console. This makes debugging great but makes it so you cannot run the program disconnected from a computer.

Solution? Once you are done debugging, remove these two lines from setup()

```
while (!Serial);
delay(500);
```

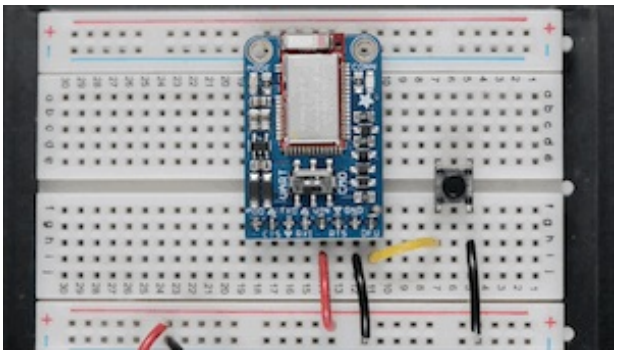
Data Mode

By placing the BLEFriend module in 'UART Data' mode (set the mode selection switch to **UART** or setting the MODE pin to gound) you can use the module as a 'tranparent UART connection' to the Bluefruit app. This makes data transfer super simple. Data is sent to the app when any 9600 baud data is received on the **RXI** pin and any data from the app is automatically transmitted via the **TXO** pin

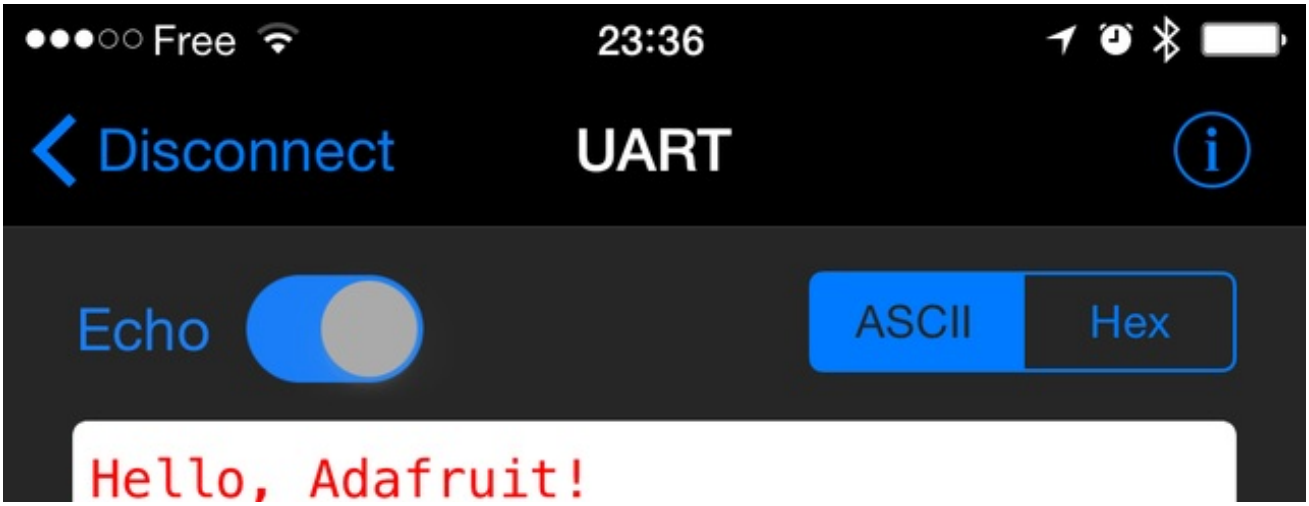
In order to keep from overloading your microcontroller, you can use the flow control pins. Keep the CTS pin high until you're ready for more data, then ground it to let the Bluefruit module know you're ready for more data!

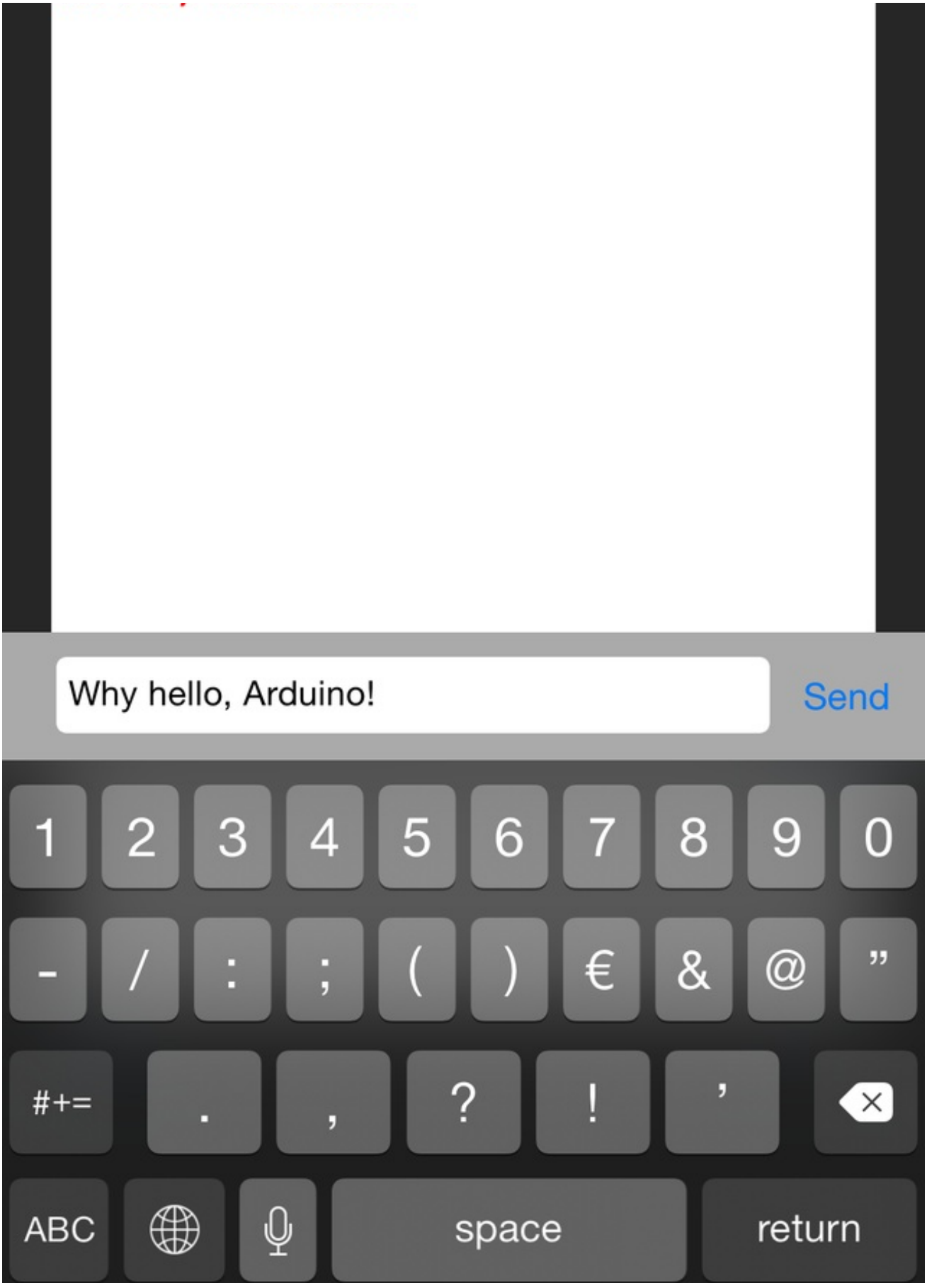
> This mode uses hardware flow control! You must set the CTS pin to ground in order to enable the TXO pin, so if you're wondering why its not sending data, check that CTS is being used right!

You can determine if you are in Data Mode by looking at the mode LED. It should blink two times followed by a three second pause, as shown below:



You can then connect the the app in **UART** mode and send/receive data transparently




Why hello, Arduino!
Send
1 2 3 4 5 6 7 8 9 0
- / : ; ( ) € & @ ”
#+= . , ? ! ’
ABC
space
return

Switching Command/Data Mode via +++

On either side of the connection (via the Arduino Uno or in your mobile app), you can dynamically switch between command and data mode by sending the "**+++\n**" string, as detailed in the +++ command summary (http://adafru.it/f4Y).

If you start in data mode, you can send text for example, with "+++\nATI\n+++\n", which will cause the Bluefruit LE module to switch to command mode, execute the ATI command, and then switch back to data mode.

> The +++ command can be sent from either side, making it possible to execute commands from the mobile application as well as on the Bluefruit LE side.

```
# Start in Data Mode
> Hello, World! Data mode!

# Send command to switch modes
> +++

# Bluefruit LE module switches to CMD mode
# Send ATI command and wait for the response
> ATI
< BLEFRIEND
< nRF51822 QFAAG00
< B122AAC33F3D2296
< 0.6.2
< 0.6.2
< May 01 2015
< OK

# Switch back to DATA mode
> +++
< OK

# We're back in data mode now
Welcome back!
```

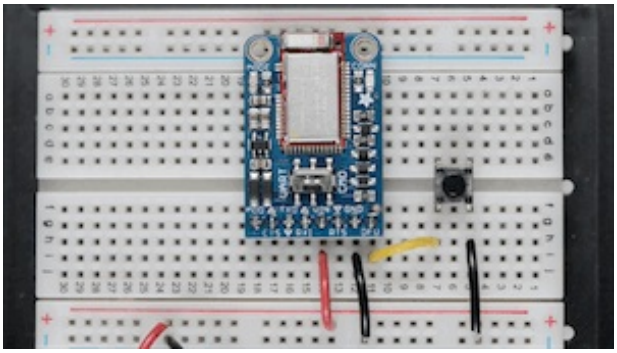
Command Mode

By placing the BLEFriend module in 'Command' mode (set the mode selection switch to **CMD** or setting the MODE pin to a high voltage) you can enter a variety of Hayes AT style commands to configure the device or retrieve basic information about the module of BLE connection.

In order to keep from overloading your microcontroller, you can use the flow control pins. Keep the CTS pin high until you're ready for more data, then ground it to let the Bluefruit module know you're ready for more data!

> This mode uses hardware flow control! You must set the CTS pin to ground in order to enable the TXO pin, so if you're wondering why its not responding to commands, check that CTS is being used right!

You can determine if you are in Command Mode by looking at the mode LED. It should blink three times followed by a three second pause, as shown below:



Hayes/AT Commands

When operating in command mode, the Bluefruit LE Pro modules use a Hayes AT-style command set (http://adafru.it/ebJ)to configure the device.

The advantage of an AT style command set is that it's easy to use in machine to machine communication, while still being somewhat user friendly for humans.

Test Command Mode '=?'

'Test' mode is used to check whether or not the specified command exists on the system or not.

Certain firmware versions or configurations may or may not include a specific command, and you can determine if the command is present by taking the command name and appending **'=?'** to it, as shown below

```
AT+BLESTARTADV=?
```

If the command is present, the device will reply with **'OK'**. If the command is not present, the device will reply with **'ERROR'**.

```
AT+BLESTARTADV=?
OK\r\n
AT+MISSINGCMD=?
ERROR\r\n
```

Write Command Mode '=xxx'

'Write' mode is used to assign specific value(s) to the command, such as changing the radio's transmit power level using the command we used above.

To write a value to the command, simple append an **'='** sign to the command followed by any paramater(s) you wish to write (other than a lone **'?'** character which will be interpretted as tet mode):

```
AT+BLEPOWERLEVEL=-8
```

If the write was successful, you will generally get an **'OK'** response on a new line, as shown below:

```
AT+BLEPOWERLEVEL=-8
OK\r\n
```

If there was a problem with the command (such as an invalid parameter) you will get an **'ERROR'** response on a new line, as shown below:

```
AT+BLEPOWERLEVEL=3
ERROR\r\n
```

Note: This particular error was generated because '3' is not a valid value for the AT+BLEPOWERLEVEL command. Entering '-4', '0' or '4' *would* succeed since these are all valid values for this command.

Execute Mode

'Execute' mode will cause the specific command to 'run', if possible, and will be used when the

command name is entered with no additional parameters.

```
AT+FACTORYRESET
```

You might use execute mode to perform a factory reset, for example, by executing the AT+FACTORYRESET command as follows:

```
AT+FACTORYRESET
OK\r\n
```

NOTE: Many commands that are means to be read will perform the same action whether they are sent to the command parser in 'execute' or 'read' mode. For example, the following commands will produce identical results:

```
AT+BLEGETPOWERLEVEL
-4\r\n
OK\r\n
AT+BLEGETPOWERLEVEL?
-4\r\n
OK\r\n
```

If the command doesn't support execute mode, the response will normally be **'ERROR'** on a new line.

Read Command Mode '?'

'Read' mode is used to read the current value of a command.

Not every command supports read mode, but you generally use this to retrieve information like the current transmit power level for the radio by appending a **'?'** to the command, as shown below:

```
AT+BLEPOWERLEVEL?
```

If the command doesn't support read mode or if there was a problem with the request, you will normally get an **'ERROR'** response.

If the command read was successful, you will normally get the read results followed by **'OK'** on a new line, as shown below:

```
AT+BLEPOWERLEVEL?
-4\r\n
OK\r\n
```

Note: For simple commands, 'Read' mode and 'Execute' mode behave identically.

Dynamically Switching Modes via +++

When operating in Command Mode you can dynamically switch to Data Mode (and back again) in software via the +++ command (http://adafru.it/f4Y). See the full command description for details.

Standard AT

The following standard Hayes/AT commands are available on Bluefruit LE modules:

AT

Acts as a ping to check if we are in command mode. If we are in command mode, we should receive the 'OK' response.

Codebase Revision: 0.3.0

Parameters: None

Output: None

```
AT
OK
```

ATI

Displays basic information about the Bluefruit module.

Codebase Revision: 0.3.0

Parameters: None

Output: Displays the following values:

- Board Name
- Microcontroller/Radio SoC Name
- Unique Serial Number
- Core Bluefruit Codebase Revision
- Project Firmware Revision
- Firmware Build Date
- Softdevice, Softdevice Version, Bootloader Version (0.5.0+)

```
ATI
BLEFRIEND
nRF51822 QFAAG00
FB462DF92A2C8656
0.5.0
0.5.0
Feb 24 2015
S110 7.1.0, 0.0
OK
```

Updates:

- Version **0.4.7+** of the firmware adds the chip revision after the chip name if it can be detected (ex. 'nRF51822 QFAAG00').
- Version **0.5.0+** of the firmware adds a new 7th record containing the softdevice, softdevice version and bootloader version (ex. 'S110 7.1.0, 0.0').

ATZ

Performs a system reset.

Codebase Revision: 0.3.0

Parameters: None

Output: None

```
ATZ
OK
```

ATE

Enables or disables echo of input characters with the AT parser

Codebase Revision: 0.3.0

Parameters: '1' = enable echo, '0' = disable echo

Output: None

```
# Disable echo support
ATE=0
OK
#Enable echo support
ATE=1
OK
```

+++

Dynamically switches between DATA and COMMAND mode without changing the physical CMD/UART select switch.

When you are in COMMAND mode, entering '+++\n' or '+++\r\n' will cause the module to switch to DATA mode, and anything typed into the console will go direct to the BLUE UART service.

To switch from DATA mode back to COMMAND mode, simply enter '+++\n' or '+++\r\n' again (be sure to include the new line character!), and a new 'OK' response will be displayed letting you know that you are back in COMMAND mode (see the two 'OK' entries in the sample code below).

Codebase Revision: 0.4.7

Parameters: None

Output: None

> Note that +++ can also be used on the mobile device to send and receive AT command on iOS or Android, though this should always be used with care.

```
ATI
BLEFRIEND
nRF51822 QFAAG00
B122AAC33F3D2296
0.4.6
0.4.6
Dec 22 2014
OK
+++
OK
OK
```

General Purpose

The following general purpose commands are available on all Bluefruit LE modules:

AT+FACTORYRESET

Clears any user config data from non-volatile memory and performs a factory reset before resetting the Bluefruit module.

Codebase Revision: 0.3.0

Parameters: None

Output: None

```
AT+FACTORYRESET
OK
```

> As of version 0.5.0+ of the firmware, you can perform a factory reset by holding the DFU button down for 10s until the blue CONNECTED LED lights up, and then releasing the button.

AT+DFU

Forces the module into DFU mode, allowing over the air firmware updates using a dedicated DFU app on iOS or Android.

Codebase Revision: 0.3.0

Parameters: None

Output: None

> The AT parser will no longer responsd after the AT+DFU command is entered, since normal program execution effectively halts and a full system reset is performed to start the bootloader code

```
AT+DFU
OK
```

AT+HELP

Displays a comma-separated list of all AT parser commands available on the system.

Codebase Version: 0.3.0

Parameters: None

Output: A comma-separated list of all AT parser commands available on the system.

> The sample code below may not match future firmware releases and is provided for illustration purposes only

```
AT+HELP
AT+FACTORYRESET,AT+DFU,ATZ,ATI,ATE,AT+DBGMEMRD,AT+DBGNVMRD,AT+HWLEDPOLARITY,AT-
OK
```

Hardware

The following commands allow you to interact with the low level HW on the Bluefruit LE module, such as reading or toggling the GPIO pins, performing an ADC conversion ,etc.:

AT+HWADC

Performs an ADC conversion on the specified ADC pin

Codebase Revision: 0.3.0

Parameters: The ADC channel (0..7)

Output: The results of the ADC conversion

```
AT+HWADC=0
178
OK
```

AT+HWGETDIETEMP

Gets the temperature in degree celcius of the BLE module's die. This can be used for debug purposes (higher die temperature generally means higher current consumption), but does not corresponds to ambient temperature and can nto be used as a replacement for a normal temperature sensor.

Codebase Revision: 0.3.0

Parameters: None

Output: The die temperature in degrees celcius

```
AT+HWGETDIETEMP
32.25
OK
```

AT+HWGPIO

Gets or sets the value of the specified GPIO pin (depending on the pin's mode).

Codebase Revision: 0.3.0

Parameters: The parameters for this command change depending on the pin mode.

OUTPUT MODE: The following comma-separated parameters can be used when updating a pin that is set as an output:

- Pin numbers
- Pin state, where:
    - 0 = clear the pin (logic low/GND)
    - 1 = set the pin (logic high/VCC)

INPUT MODE: To read the current state of an input pin or a pin that has been configured as an output, enter the pin number as a single parameter.

Output: The pin state if you are reading an input or checking the state of an input pin (meaning only 1 parameter is supplied, the pin number), where:

- 0 means the pin is logic low/GND
- 1 means the pin is logic high/VCC

Trying to set the value of a pin that has not been configured as an output will produce an 'ERROR' response.

Some pins are reserved for specific functions on Bluefruit modules and can not be used as GPIO. If you try to make use of a reserved pin number an 'ERROR' response will be generated.

```
# Set pin 14 HIGH
AT+HWGPIO=14,1
OK

# Set pin 14 LOW
AT+HWGPIO=14,0
OK

# Read the current state of pin 14
AT+HWGPIO=14
0
OK

# Try to update a pin that is not set as an output
AT+HWGPIOMODE=14,0
OK
AT+HWGPIO=14,1
ERROR
```

AT+HWGPIOMODE

This will set the mode for the specified GPIO pin (input, output, etc.).

Codebase Revision: 0.3.0

Parameters: This command one or two values (comma-separated in the case of two parameters being used):

- The pin number
- The new GPIO mode, where:
    - 0 = Input
    - 1 = Output
    - 2 = Input with pullup enabled
    - 3 = Input with pulldown enabled

Output: If a single parameters is passed (the GPIO pin number) the current pin mode will be returned.

> Some pins are reserved for specific functions on Bluefruit modules and can not be used as GPIO. If you try to make use of a reserved pin number an 'ERROR' response will be generated.

```
# Configure pin 14 as an output
AT+HWGPIOMODE=14,0
OK

# Get the current mode for pin 14
AT+HWPGIOMODE=14
0
OK
```

AT+HWI2CSCAN

Scans the I2C bus to try to detect any connected I2C devices, and returns the address of devices that were found during the scan process.

Codebase Revision: 0.3.0

Parameters: None

Output: A comma-separated list of any I2C address that were found while scanning the valid address range on the I2C bus, or nothing is no devices were found.

```
# I2C scan with two devices detected
AT+HWI2CSCAN
0x23,0x35
OK

# I2C scan with no devices detected
AT+HWI2CSCAN
OK
```

AT+HWVBAT

Returns the main power supply voltage level in millivolts

Codebase Revision: 0.3.0

Parameters: None

Output: The VBAT level in millivolts

```
AT+HWVBAT
3248
OK
```

AT+HWRANDOM

Generates a random 32-bit number using the HW random number generator on the nRF51822 (based on white noise).

Codebase Revision: 0.4.7

Parameters: None

Output: A random 32-bit hexadecimal value (ex. '0x12345678')

```
AT+HWRANDOM
0x769ED823
OK
```

Beacon

Bluefruit LE modules can be configured to act as 'Beacons' using the following commands:

AT+BLEBEACON

Codebase Revision: 0.3.0

Parameters: The following comma-separated parameters are required to enable beacon mode:

- Bluetooth Manufacturer ID (uint16_t)
- 128-bit UUID
- Major Value (uint16_t)
- Minor Value (uint16_t)
- RSSI @ 1m (int8_t)

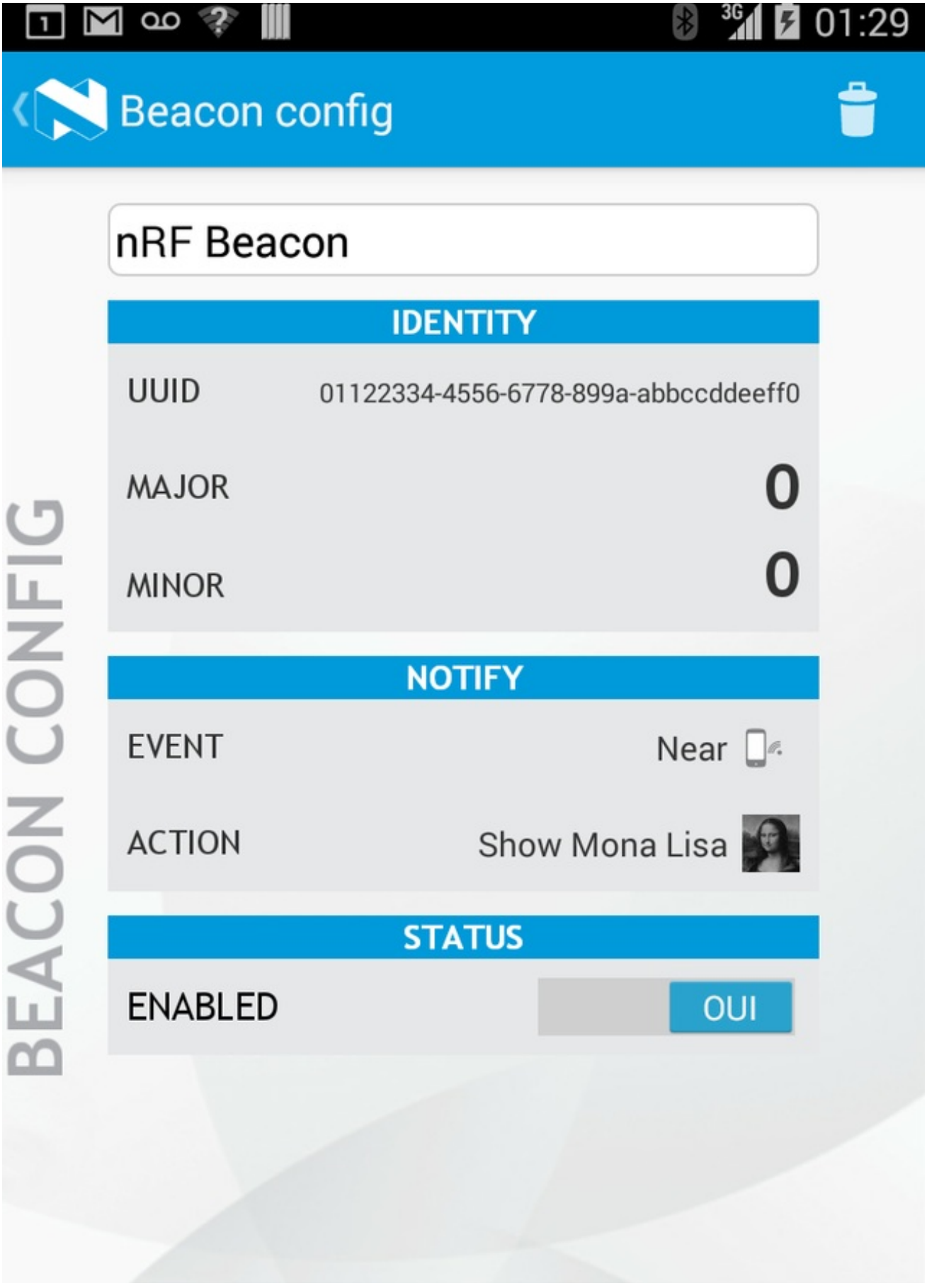
Output: None

```
# Enable Apple iBeacon emulation
# Manufacturer ID = 0x004C
AT+BLEBEACON=0x004C,01-12-23-34-45-56-67-78-89-9A-AB-BC-CD-DE-EF-F0,0x0000,0x0000,-59
OK
# Reset to change the advertising data
ATZ
OK

# Enable Nordic Beacon emulation
# Manufacturer ID = 0x0059
AT+BLEBEACON=0x0059,01-12-23-34-45-56-67-78-89-9A-AB-BC-CD-DE-EF-F0,0x0000,0x0000,-59
OK
# Reset to change the advertising data
ATZ
OK
```

> AT+BLEBEACON will cause the beacon data to be stored in non-volatile config memory on the Bluefruit LE module, and these values will be persisted across system resets and power cycles. To remove or clear the beacon data you need to enter the 'AT+FACTORYRESET' command in command mode.

Entering Nordic Beacon emulation using the sample code above, you can see the simulated beacon in Nordic's 'Beacon Config' tool below:


01:29
Beacon config
nRF Beacon
IDENTITY
UUID
01122334-4556-6778-899a-abbccddeeff0
MAJOR
0
MINOR
0
NOTIFY
EVENT
Near
ACTION
Show Mona Lisa
STATUS
ENABLED
OUI
BEACON CONFIG

AT+BLEURIBEACON

Converts the specified URI into a UriBeacon (http://adafru.it/edk) advertising packet, and configures the module to advertise as a UriBeacon (part of Google's Physical Web (http://adafru.it/ehZ) project).

To view the UriBeacon URIs you can use one of the following mobile applications:

- Android 4.3+: Physical Web (http://adafru.it/edi) on the Google Play Store
- iOS: Physical Web (http://adafru.it/edj) in Apple's App Store

Codebase Revision: 0.4.7

Parameters: The URI to encode (ex. http://www.adafruit.com/blog (http://adafru.it/ei0))

Output: None of a valid URI was entered (length is acceptable, etc.).

```
AT+BLEURIBEACON=http://www.adafruit.com/blog
OK

# Reset to change the advertising data
ATZ
OK
```

If the supplied URI is too long you will get the following output:

```
AT+BLEURIBEACON=http://www.adafruit.com/this/uri/is/too/long
URL is too long
ERROR
```

If the URI that you are trying to encode is too long, try using a shortening service like bit.ly, and encode the shortened URI.

BLE Generic

The following general purpose BLE commands are available on Bluefruit LE modules:

AT+BLEPOWERLEVEL

Gets or sets the current transmit power level for the module's radio (higher transmit power equals better range, lower transmit power equals better battery life).

Codebase Revision: 0.3.0

Parameters: The TX power level (in dBm), which can be one of the following values (from lowest to higher transmit power):

- -40
- -20
- -16
- -12
- -8
- -4
- 0
- 4

Output: The current transmit power level (in dBm)

> The updated power level will take affect as soon as the command is entered. If the device isn't connected to another device, advertising will stop momentarily and then restart once the new power level has taken affect.

```
# Get the current TX power level (in dBm)
AT+BLEPOWERLEVEL
0
OK

# Set the TX power level to 4dBm (maximum value)
AT+BLEPOWERLEVEL=4
OK

# Set the TX power level to -12dBm (better battery life)
AT+BLEPOWERLEVEL=-12
OK

# Set the TX power level to an invalid value
AT+BLEPOWERLEVEL=-3
ERROR
```

AT+BLEGETADDRTYPE

Gets the address type (for the 48-bit BLE device address).

Normally this will be '1' (random), which means that the module uses a 48-bit address that was randomly generated during the manufacturing process and written to the die by the manufacturer.

Random does not mean that the device address is randomly generated every time, only that a one-time random number is used.

Codebase Revision: 0.3.0

Parameters: None

Output: The address type, which can be one of the following values:

- 0 = public
- 1 = random

```
AT+BLEGETADDRTYPE
1
OK
```

AT+BLEGETADDR

Gets the 48-bit BLE device address.

Codebase Revision: 0.3.0

Parameters: None

Output: The 48-bit BLE device address in the following format: 'AA:BB:CC:DD:EE:FF'

```
AT+BLEGETADDR
E4:C6:C7:31:95:11
OK
```

AT+BLEGETPEERADDR

Gets the 48-bit address of the peer (central) device we are connected to.

Codebase Revision: 0.6.5

Parameters: None

Output: The 48-bit address of the connected central device in hex format. The command will return **ERROR** if we are not connected to a central device.

> Please note that the address returned by the central device is almost always a random value that will change over time, and this value should generally not be trusted. This command is provided for certain edge cases, but is not useful in most day to day scenarios.

```
AT+BLEGETPEERADDR
48:B2:26:E6:C1:1D
OK
```

AT+BLEGETRSSI

Gets the RSSI value (Received Signal Strength Indicator), which can be used to estimate the reliability of data transmission between two devices (the lower the number the better).

Codebase Revision: 0.3.0

Parameters: None

Output: The RSSI level (in dBm) if we are connected to a device, otherwise '0'

```
# Connected to an external device
AT+BLEGETRSSI
-46
OK

# Not connected to an external device
AT+BLEGETRSSI
0
OK
```

BLE Services

AT+BLEUARTTX

This command will transmit the specified text message out via the UART Service (http://adafru.it/ekD) while you are running in Command Mode.

Codebase Revision: 0.3.0

Parameters: The message payload to transmit (20 characters max)

Output: None

> You must be connected to another device for this command to execute ('ERROR' will be displayed if no connection has been established).

```
# Send a string when connected to another device
AT+BLEUARTTX=THIS IS A TEST
OK

# Send a string when not connected
AT+BLEUARTTX=THIS IS A TEST
ERROR
```

As of firmware release **0.6.2** and higher, AT+BLEUARTTX can accept a limited set of escape code sequences:

- \r = carriage return
- \n = new line
- \t = tab
- \b = backspace
- \\ = backward slash

AT+BLEUARTRX

This command will dump the UART service (http://adafru.it/ekD)'s RX buffer to the display if any data has been received from from the UART service while running in Command Mode. The data will be removed from the buffer once it is displayed using this command.

Any characters left in the buffer when switching back to Data Mode will cause the buffered characters to be displayed as soon as the mode switch is complete (within the limits of available buffer space, typically 256 characters).

Codebase Revision: 0.3.0

Parameters: None

Output: The RX buffer's content if any data is available, otherwise nothing.

```
# Command results when data is available
AT+BLEUARTRX
Sent from Android
OK

# Command results when no data is available
AT+BLEUARTRX
OK
```

The following commands allow you to interact with mandatory GATT services present on Bluefruit LE Pro modules like the BLEFRiend when running in Command Mode.

AT+BLEKEYBOARDEN

This command will enable GATT over HID (GoH) keyboard support, which allows you to emulate a keyboard on supported iOS and Android devices. By default HID keyboard support is disabled, so you need to set BLEKEYBOARDEN to 1 and then perform a system reset before the keyboard will be enumerated and appear in the Bluetooth preferences on your phone, where if can be bonded as a BLE keyboard.

Codebase Revision: 0.5.0

Parameters: 1 or 0 (1 = enable, 0 = disable)

Output: None

> You must perform a system reset (ATZ) before the changes take effect!

> Before you can use your HID over GATT keyboard, you will need to bond your mobile device with the Bluefruit LE module in the Bluetooth preferences panel.

```
# Enable BLE keyboard support then reset
AT+BLEKEYBOARDEN=1
OK
ATZ
OK

# Diable BLE keyboard support then reset
AT+BLEKEYBOARDEN=0
OK
ATZ
OK
```

AT+BLEKEYBOARD

Sends text data over the BLE keyboard interface (if it has previously been enabled via AT+BLEKEYBOARDEN).

Any valid alpha-numeric character can be sent, and the following escape sequences are also supported:

- \r - Carriage Return
- \n - Line Feed
- \b - Backspace
- \t - Tab
- \\ - Backslash

Codebase Revision: 0.5.0

Parameters: The text string (optionally including escape characters) to transmit

Output: None

```
# Send a URI with a new line ending to execute in Chrome, etc.
AT+BLEKEYBOARD=http://www.adafruit.com\r\n
OK
```

AT+BLEKEYBOARDCODE

Sends a raw hex sequence of characters to the BLE keyboard interface including key modifiers and up to six alpha-numeric characters.

This command excepts the following ascii-encoded HEX payload, matching the way HID over GATT sends keyboard data:

- **Byte 0**: Modifier
- **Byte 1**: Reserved (should always be 00)
- **Bytes 2..7**: Hexadecimal values for ASCII-encoded characters (if no character is used you can enter '00' or leave trailing characters empty)

After a keycode sequence is sent with the AT+BLEKEYBOARDCODE command, **you must send a second AT+BLEKEYBOARDCODE command with at least two 00 characters to indicate the keys were released!**

Modifier Values

The modifier byte can have one or more of the following bits set:

- **Bit 0 (0x01)**: Left Control
- **Bit 1 (0x02)**: Left Shift
- **Bit 2 (0x04)**: Left Alt
- **Bit 3 (0x08)**: Left Window
- **Bit 4 (0x10)**: Right Control
- **Bit 5 (0x20)**: Right Shift
- **Bit 6 (0x40)**: Right Alt
- **Bit 7 (0x80)**: Right Window

Codebase Revision: 0.5.0

Parameters: A set of ASCII-encoded hexadecimal characters values separated by a hyphen ('-')

Output: None

```
# send ABC with shift key --> "ABC"
AT+BLEKEYBOARDCODE=01-00-04-05-06-00-00
OK
# Indicate that the keys were released (mandatory!)
AT+BLEKEYBOARDCODE=00-00
OK
```

BLE GAP

GAP (http://adafru.it/eci), which stands for the *Generic Access Profile*, governs advertising and connections with Bluetooth Low Energy devices.

The following commands can be used to configure the GAP settings on the BLE module.

You can use these commands to modify the advertising data (for ex. the device name that appears during the advertising process), to retrieve information about the connection that has been established between two devices, or the disconnect if you no longer wish to maintain a connection.

AT+GAPGETCONN

Diplays the current connection status (if we are connected to another BLE device or not).

Codebase Revision: 0.3.0

Parameters: None

Output: 1 if we are connected, otherwise 0

```
# Connected
AT+GAPGETCONN
1
OK

# Not connected
AT+GAPGETCONN
0
OK
```

AT+GAPDISCONNECT

Disconnects to the external device if we are currently connected.

Codebase Revision: 0.3.0

Parameters: None

Output: None

```
AT+GAPDISCONNECT
OK
```

AT+GAPDEVNAME

Gets or sets the device name, which is included in the advertising payload for the Bluefruit LE module

Codebase Revision: 0.3.0

Parameters:

- None to read the current device name
- The new device name if you want to change the value

Output: The device name if the command is executed in read mode

> Updating the device name will persist the new value to non-volatile memory, and the updated name will be used when the device is reset. To reset the device to factory settings and clean the config data from memory run the AT+FACTORYRESET command.

```
# Read the current device name
AT+GAPDEVNAME
UART
OK

# Update the device name to 'BLEFriend'
AT+GAPDEVNAME=BLEFriend
OK
# Reset to take effect
ATZ
OK
```

AT+GAPDELBONDS

Deletes and bonding information stored on the Bluefruit LE module.

Codebase Revision: 0.3.0

Parameters: None

Output: None

```
AT+GAPDELBONDS
OK
```

AT+GAPINTERVALS

Gets or sets the various advertising and connection intervals for the Bluefruit LE module.

Be extremely careful with this command since it can be easy to cause problems changing the intervals, and depending on the values selected some mobile devices may no longer recognize the module or refuse to connect to it.

Codebase Revision: 0.3.0

Parameters: If updating the GAP intervals, the following comma-separated values can be entered:

- Minimum connection interval (in milliseconds)
- Maximum connection interval (in milliseconds)
- Advertising interval (in milliseconds)
- Advertising timeout (in milliseconds)

> If you only wish to update one interval value, leave the other comma-separated values empty (ex. ',,110,' will only update the third value, advertising interval).

Output: If reading the current GAP interval settings, the following comma-separated information will be displayed:

- Minimum connection interval (in milliseconds)
- Maximum connection interval (in milliseconds)
- Advertising interval (in milliseconds)
- Advertising timeout (in milliseconds)

> Updating the GAP intervals will persist the new values to non-volatile memory, and the updated values will be used when the device is reset. To reset the device to factory settings and clean the config data from memory run the AT+FACTORYRESET command.

```
# Read the current GAP intervals
AT+GAPINTERVALS
20,100,100,30

# Update all values
AT+GAPINTERVALS=20,200,200,30
OK

# Update only the advertising interval
AT+GAPINTERVALS=,,150,
OK
```

AT+GAPSTARTADV

Causes the Bluefruit LE module to start transmitting advertising packets if this isn't already the case (assuming we aren't already connected to an external device).

Codebase Revision: 0.3.0

Parameters: None

Output: None

```
# Command results when advertising data is not being sent
AT+GAPSTARTADV
OK

# Command results when we are already advertising
AT+GAPSTARTADV
ERROR

# Command results when we are connected to another device
AT+GAPSTARTADV
ERROR
```

AT+GAPSTOPADV

Stops advertising packets from being transmitted by the Bluefruit LE module.

Codebase Revision: 0.3.0

Parameters: None

Output: None

```
AT+GAPSTOPADV
OK
```

AT+GAPSETADVDATA

Sets the raw advertising data payload to the specified byte array (overriding the normal advertising data), following the guidelines in the Bluetooth 4.0 or 4.1 Core Specification (http://adafru.it/ddd).

In particular, **Core Specification Supplement (CSS) v4** contains the details on common advertising data fields like 'Flags' (Part A, Section 1.3) and the various Service UUID lists (Part A, Section 1.1). A list of all possible GAP Data Types is available on the Bluetooth SIG's Generic Access Profile (http://adafru.it/cYs) page.

The Advertising Data payload consists of Generic Access Profile (http://adafru.it/cYs) data that is inserted into the advertising packet in the following format: [U8:LEN] [U8:Data Type Value] [n:Value]

WARNING: This command requires a degree of knowledge about the low level details of the Bluetooth 4.0 or 4.1 Core Specification, and should only be used by expert users. Misuse of this command can easily cause your device to be undetectable by central devices in radio range.

WARNING: This command will override the normal advertising payload and may prevent some services from acting as expected.

To restore the advertising data to the normal default values use the AT+FACTORYRESET command.

For example, to insert the 'Flags' Data Type (Data Type Value 0x01), and set the value to 0x06/0b00000110 (BR/EDR Not Supported and LE General Discoverable Mode) we would use the following byte array:

```
02-01-06
```

- 0x02 indicates the number of bytes in the entry

- 0x01 is the 'Data Type Value (http://adafru.it/cYs)' and indicates that this is a '**Flag**'
- 0x06 (0b00000110) is the Flag value, and asserts the following fields (see Core Specification 4.0, Volume 3, Part C, 18.1):
    - **LE General Discoverable Mode** (i.e. anyone can discover this device)
    - **BR/EDR Not Supported** (i.e. this is a Bluetooth Low Energy only device)

If we also want to include two 16-bit service UUIDs in the advertising data (so that listening devices know that we support these services) we could append the following data to the byte array:

```
05-02-0D-18-0A-18
```

- 0x05 indicates that the number of bytes in the entry (5)
- 0x02 is the 'Data Type Value (http://adafru.it/cYs)' and indicates that this is an '**Incomplete List of 16-bit Service Class UUIDs**'
- 0x0D 0x18 is the first 16-bit UUID (which translates to **0x180D**, corresponding to the Heart Rate Service (http://adafru.it/ddB)).
- 0x0A 0x18 is another 16-bit UUID (which translates to **0x180A**, corresponding to the Device Information Service (http://adafru.it/ecj)).

> Including the service UUIDs is important since some mobile applications will only work with devices that advertise a specific service UUID in the advertising packet. This is true for most apps from Nordic Semiconductors, for example.

Codebase Revision: 0.3.0

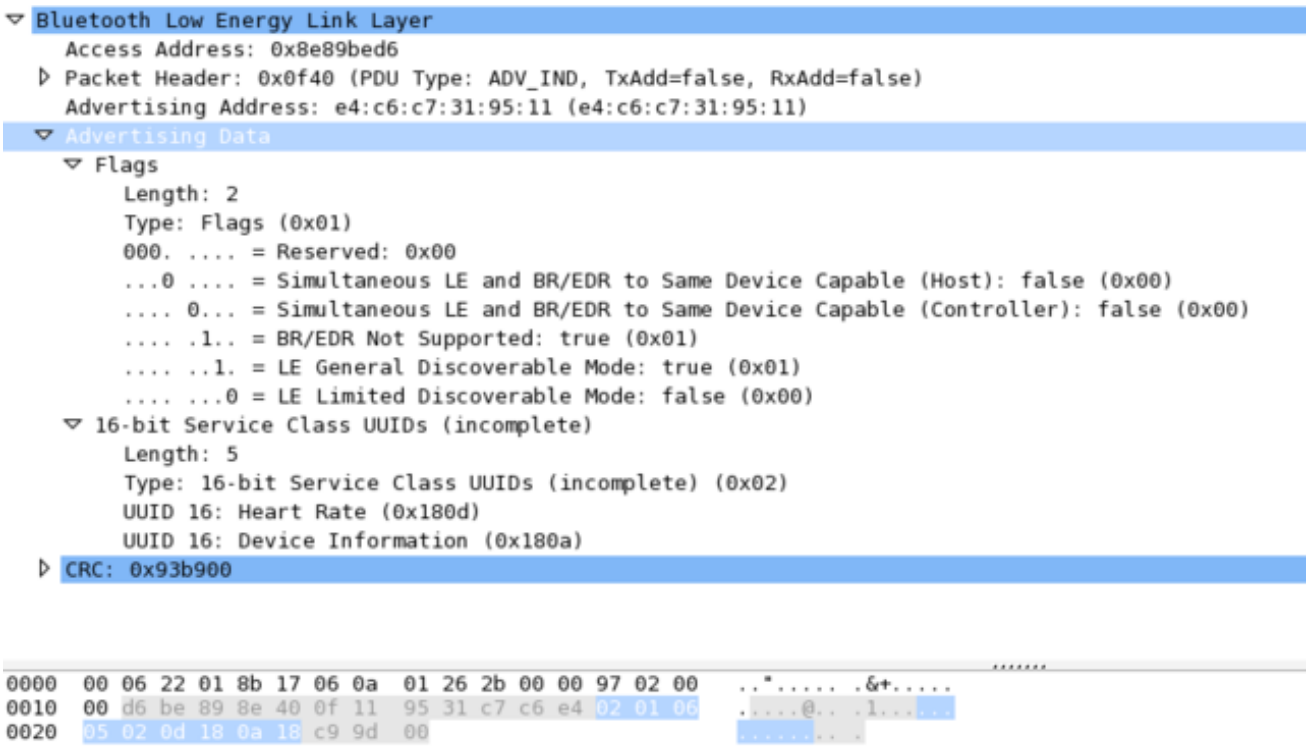
Parameters: The raw byte array that should be inserted into the advertising data section of the advertising packet, being careful to stay within the space limits defined by the Bluetooth Core Specification.

Response: None

```
# Advertise as Discoverable and BLE only with 16-bit UUIDs 0x180D and 0x180A
AT+GAPSETADVDATA=02-01-06-05-02-0d-18-0a-18
OK
```

The results of this command can be seen in the screenshot below, taken from a sniffer analyzing the advertising packets in Wireshark. The advertising data payload is higlighted in blue in the raw

byte array at the bottom of the image, and the packet analysis is in the upper section:



BLE GATT

GATT (http://adafru.it/ebg), which standards for the *Generic ATTribute Profile*, governs data organization and data exchanges between connected devices. One device (the peripheral) acts as a GATT Server, which stores data in *Attribute* records, and the second device in the connection (the central) acts as a GATT Client, requesting data from the server whenever necessary.

The following commands can be used to create custom GATT services and characteristics on the BLEFriend, which are used to store and exchange data.

Please note that any characteristics that you define here will automatically be saved to non-volatile FLASH config memory on the device and re-initialised the next time the device starts. **You need to perform a system reset before most of the commands below will take effect!**

If you want to clear any previous config value, enter the '**AT+FACTORYRESET**' command before working on a new peripheral configuration.

AT+GATTCLEAR

Clears any custom GATT services and characteristics that have been defined on the device.

Codebase Revision: 0.3.0

Parameters: None

Response: None

```
AT+GATTCLEAR
OK
```

AT+GATTADDSERVICE

Adds a new custom service definition to the device.

Codebase Revision: 0.3.0

Parameters: This command accepts a set of comma-separated key-value pairs that are used to define the service properties. The following key-value pairs can be used:

- **UUID**: The 16-bit UUID to use for this service. 16-bit values should be in hexadecimal format (0x1234).
- **UUID128**: The 128-bit UUID to use for this service. 128-bit values should be in the following format: 00-11-22-33-44-55-66-77-88-99-AA-BB-CC-DD-EE-FF

Response: The index value of the service in the custom GATT service lookup table. (It's important to keep track of these index values to work with the service later.)

Note: Key values are not case-sensitive

Only one UUID type can be entered for the service (either UUID or UUID128)

```
# Clear any previous custom services/characteristics
AT+GATTCLEAR
OK

# Add a battery service (UUID = 0x180F) to the peripheral
AT+GATTADDSERVICE=UUID=0x180F
1
OK

# Add a battery measurement characteristic (UUID = 0x2A19), notify enabled
AT+GATTADDCHAR=UUID=0x2A19,PROPERTIES=0x10,MIN_LEN=1,VALUE=100
1
OK
```

```
# Clear any previous custom services/characteristics
AT+GATTCLEAR
OK

# Add a custom service to the peripheral
AT+GATTADDSERVICE=UUID128=00-11-00-11-44-55-66-77-88-99-AA-BB-CC-DD-EE-FF
1
OK

# Add a custom characteristic to the above service (making sure that there
# is no conflict between the 16-bit UUID and bytes 3+4 of the 128-bit service UUID)
AT+GATTADDCHAR=UUID=0x0002,PROPERTIES=0x02,MIN_LEN=1,VALUE=100
1
OK
```

AT+GATTADDCHAR

Adds a custom characteristic to the last service that was added to the peripheral (via AT+GATTADDSERVICE).

AT+GATTADDCHAR must be run AFTER AT+GATTADDSERVICE, and will add the new characteristic to the last service definition that was added.

Codebase Revision: 0.3.0

Parameters: This command accepts a set of comma-separated key-value pairs that are used to define the characteristic properties. The following key-value pais can be used:

- **UUID**: The 16-bit UUID to use for the characteristic (which will be insert in the 3rd and 4th bytes of the parent services 128-bit UUID). This value should be entered in hexadecimal format (ex. 'UUID=0x1234'). This value must be unique, and should not conflict with bytes 3+4 of the parent service's 128-bit UUID.
- **PROPERTIES**: The 8-bit characteristic properties field, as defined by the Bluetooth SIG. The following values can be used:
    - 0x02 - Read
    - 0x04 - Write Without Response
    - 0x08 - Write
    - 0x10 - Notify
    - 0x20 - Indicate
- **MIN_LEN**: The minimum size of the values for this characteristic (in bytes, min = 1, max = 20, default = 1)
- **MAX_LEN**: The maximum size of the values for the characteristic (in bytes, min = 1, max = 20, default = 1)
- **VALUE**: The initial value to assign to this characteristic (within the limits of 'MIN_LEN' and 'MAX_LEN'). Value can be an integer ("-100", "27"), a hexadecimal value ("0xABCD"), a byte array ("aa-bb-cc-dd") or a string ("GATT!").

Response: The index value of the characteristic in the custom GATT characteristic lookup table. (It's important to keep track of these characteristic index values to work with the characteristic later.)

Note: Key values are not case-sensitive

Make sure that the 16-bit UUID is unique and does not conflict with bytes 3+4 of the 128-bit service UUID

```
# Clear any previous custom services/characteristics
AT+GATTCLEAR
OK

# Add a battery service (UUID = 0x180F) to the peripheral
AT+GATTADDSERVICE=UUID=0x180F
1
OK

# Add a battery measurement characteristic (UUID = 0x2A19), notify enabled
AT+GATTADDCHAR=UUID=0x2A19,PROPERTIES=0x10,MIN_LEN=1,VALUE=100
1
OK
```

```
# Clear any previous custom services/characteristics
AT+GATTCLEAR
OK

# Add a custom service to the peripheral
AT+GATTADDSERVICE=UUID128=00-11-00-11-44-55-66-77-88-99-AA-BB-CC-DD-EE-FF
1
OK

# Add a custom characteristic to the above service (making sure that there
# is no conflict between the 16-bit UUID and bytes 3+4 of the 128-bit service UUID)
AT+GATTADDCHAR=UUID=0x0002,PROPERTIES=0x02,MIN_LEN=1,VALUE=100
1
OK
```

AT+GATTCHAR

Gets or sets the value of the specified custom GATT characteristic (based on the index ID returned when the characteristic was added to the system via AT+GATTADDCHAR).

Codebase Revision: 0.3.0

Parameters: This function takes one or two comma-separated functions (one parameter = read, two parameters = write).

- The first parameter is the characteristic index value, as returned from the AT+GATTADDCHAR function. This parameter is always required, and if no second parameter is entered the current value of this characteristic will be returned.
- The second (optional) parameter is the new value to assign to this characteristic (within the

MIN_SIZE and MAX_SIZE limits defined when creating it).

Response: If the command is used in read mode (only the characteristic index is provided as a value), the response will display the current value of the characteristics. If the command is used in write mode (two comma-separated values are provided), the characteristics will be updated to use the provided value.

```
# Clear any previous custom services/characteristics
AT+GATTCLEAR
OK

# Add a battery service (UUID = 0x180F) to the peripheral
AT+GATTADDSERVICE=UUID=0x180F
1
OK

# Add a battery measurement characteristic (UUID = 0x2A19), notify enabled
AT+GATTADDCHAR=UUID=0x2A19,PROPERTIES=0x10,MIN_LEN=1,VALUE=100
1
OK

# Read the battery measurement characteristic (index ID = 1)
AT+GATTCHAR=1
0x64
OK

# Update the battery measurement characteristic to 32 (hex 0x20)
AT+GATTCHAR=1,32
OK

# Verify the previous write attempt
AT+GATTCHAR=1
0x20
OK
```

AT+GATTLIST

Lists all custom GATT services and characteristics that have been defined on the device.

Codebase Revision: 0.3.0

Parameters: None

Response: A list of all custom services and characteristics defined on the device.

```
# Clear any previous custom services/characteristics
AT+GATTCLEAR
OK

# Add a battery service (UUID = 0x180F) to the peripheral
AT+GATTADDSERVICE=UUID=0x180F
1
OK

# Add a battery measurement characteristic (UUID = 0x2A19), notify enabled
AT+GATTADDCHAR=UUID=0x2A19,PROPERTIES=0x10,MIN_LEN=1,VALUE=100
1
OK

# Add a custom service to the peripheral
AT+GATTADDSERVICE=UUID128=00-11-00-11-44-55-66-77-88-99-AA-BB-CC-DD-EE-FF
2
OK

# Add a custom characteristic to the above service (making sure that there
# is no conflict between the 16-bit UUID and bytes 3+4 of the 128-bit service UUID)
AT+GATTADDCHAR=UUID=0x0002,PROPERTIES=0x02,MIN_LEN=1,VALUE=100
2
OK

# Get a list of all custom GATT services and characteristics on the device
AT+GATTLIST
ID=01,UUID=0x180F
  ID=01,UUID=0x2A19,PROPERTIES=0x10,MIN_LEN=1,MAX_LEN=1,VALUE=0x64
ID=02,UUID=0x11, UUID128=00-11-00-11-44-55-66-77-88-99-AA-BB-CC-DD-EE-FF
  ID=02,UUID=0x02,PROPERTIES=0x02,MIN_LEN=1,MAX_LEN=1,VALUE=0x64
OK
```

Debug

The following debug commands are available on Bluefruit LE modules:

> Use these commands with care since they can easily lead to a HardFault error on the ARM core, which will cause the device to stop responding.

AT+DBGMEMRD

Displays the raw memory contents at the specified address.

Codebase Revision: 0.3.0

Parameters: The following comma-separated parameters can be used with this command:

- The starting address to read memory from (in hexadecimal form, with or without the leading '0x')
- The word size (can be 1, 2, 4 or 8)
- The number of words to read

Output: The raw memory contents in hexadecimal format using the specified length and word size (see examples below for details)

```
# Read 12 1-byte values starting at 0x10000009
AT+DBGMEMRD=0x10000009,1,12
FF FF FF FF FF FF FF 00 04 00 00 00
OK

# Try to read 2 4-byte values starting at 0x10000000
AT+DBGMEMRD=0x10000000,4,2
55AA55AA 55AA55AA
OK

# Try to read 2 4-byte values starting at 0x10000009
# This will fail because the Cortex M0 can't perform misaligned
# reads, and any non 8-bit values must start on an even address
AT+DBGMEMRD=0x10000009,4,2
MISALIGNED ACCESS
ERROR
```

AT+DBGNVMREAD

Displays the raw contents of the config data section of non-volatile memory

Codebase Revision: 0.3.0

Properties: None

Output: The raw config data from non-volatile memory

```
AT+DBGNVMRD
FE CA 38 05 00 03 00 00 01 12 01 00 55 41 52 54 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 0
OK
```

AT+DBGSTACKSIZE

Returns the current stack size, to help detect stack overflow or detect stack memory usage when optimising memory usage on the system.

Codebase Revision: 0.4.7

Parameters: None

Output: The current size of stack memory in bytes

```
AT+DBGSTACKSIZE
1032
OK
```

AT+DBGSTACKDUMP

Dumps the current stack contents. Unused sections of stack memory are filled with '0xCAFEFOOD' to help determine where stack usage stops.

This command is purely for debug and development purposes.

Codebase Revision: 0.4.7

Parameters: None

Output: The memory contents of the entire stack region

```
AT+DBGSTACKDUMP
```

```
AT+DBGSTACKDUMP
0x20003800: CAFEF00D CAFEF00D CAFEF00D CAFEF00D
0x20003810: CAFEF00D CAFEF00D CAFEF00D CAFEF00D
0x20003820: CAFEF00D CAFEF00D CAFEF00D CAFEF00D
0x20003830: CAFEF00D CAFEF00D CAFEF00D CAFEF00D
0x20003840: CAFEF00D CAFEF00D CAFEF00D CAFEF00D
0x20003850: CAFEF00D CAFEF00D CAFEF00D CAFEF00D
0x20003860: CAFEF00D CAFEF00D CAFEF00D CAFEF00D
0x20003870: CAFEF00D CAFEF00D CAFEF00D CAFEF00D
0x20003880: CAFEF00D CAFEF00D CAFEF00D CAFEF00D
0x20003890: CAFEF00D CAFEF00D CAFEF00D CAFEF00D
0x200038A0: CAFEF00D CAFEF00D CAFEF00D CAFEF00D
0x200038B0: CAFEF00D CAFEF00D CAFEF00D CAFEF00D
0x200038C0: CAFEF00D CAFEF00D CAFEF00D CAFEF00D
0x200038D0: CAFEF00D CAFEF00D CAFEF00D CAFEF00D
0x200038E0: CAFEF00D CAFEF00D CAFEF00D CAFEF00D
0x200038F0: CAFEF00D CAFEF00D CAFEF00D CAFEF00D
0x20003900: CAFEF00D CAFEF00D CAFEF00D CAFEF00D
0x20003910: CAFEF00D CAFEF00D CAFEF00D CAFEF00D
0x20003920: CAFEF00D CAFEF00D CAFEF00D CAFEF00D
0x20003930: CAFEF00D CAFEF00D CAFEF00D CAFEF00D
0x20003940: CAFEF00D CAFEF00D CAFEF00D CAFEF00D
0x20003950: CAFEF00D CAFEF00D CAFEF00D CAFEF00D
0x20003960: CAFEF00D CAFEF00D CAFEF00D CAFEF00D
0x20003970: CAFEF00D CAFEF00D CAFEF00D CAFEF00D
0x20003980: CAFEF00D CAFEF00D CAFEF00D CAFEF00D
0x20003990: CAFEF00D CAFEF00D CAFEF00D CAFEF00D
0x200039A0: CAFEF00D CAFEF00D CAFEF00D CAFEF00D
0x200039B0: CAFEF00D CAFEF00D CAFEF00D CAFEF00D
0x200039C0: CAFEF00D CAFEF00D CAFEF00D CAFEF00D
0x200039D0: CAFEF00D CAFEF00D CAFEF00D CAFEF00D
0x200039E0: CAFEF00D CAFEF00D CAFEF00D CAFEF00D
0x200039F0: CAFEF00D CAFEF00D CAFEF00D CAFEF00D
0x20003A00: CAFEF00D CAFEF00D CAFEF00D CAFEF00D
0x20003A10: CAFEF00D CAFEF00D CAFEF00D CAFEF00D
0x20003A20: CAFEF00D CAFEF00D CAFEF00D CAFEF00D
0x20003A30: CAFEF00D CAFEF00D CAFEF00D CAFEF00D
0x20003A40: CAFEF00D CAFEF00D CAFEF00D CAFEF00D
0x20003A50: CAFEF00D CAFEF00D CAFEF00D CAFEF00D
0x20003A60: CAFEF00D CAFEF00D CAFEF00D CAFEF00D
0x20003A70: CAFEF00D CAFEF00D CAFEF00D CAFEF00D
0x20003A80: CAFEF00D CAFEF00D CAFEF00D CAFEF00D
0x20003A90: CAFEF00D CAFEF00D CAFEF00D CAFEF00D
0x20003AA0: CAFEF00D CAFEF00D CAFEF00D CAFEF00D
0x20003AB0: CAFEF00D CAFEF00D CAFEF00D CAFEF00D
0x20003AC0: CAFEF00D CAFEF00D CAFEF00D CAFEF00D
0x20003AD0: CAFEF00D CAFEF00D CAFEF00D CAFEF00D
```

```
0x20003AE0: CAFEF00D CAFEF00D CAFEF00D CAFEF00D
0x20003AF0: CAFEF00D CAFEF00D CAFEF00D CAFEF00D
0x20003B00: CAFEF00D CAFEF00D CAFEF00D CAFEF00D
0x20003B10: CAFEF00D CAFEF00D CAFEF00D CAFEF00D
0x20003B20: CAFEF00D CAFEF00D CAFEF00D CAFEF00D
0x20003B30: CAFEF00D CAFEF00D CAFEF00D CAFEF00D
0x20003B40: CAFEF00D CAFEF00D CAFEF00D CAFEF00D
0x20003B50: CAFEF00D CAFEF00D CAFEF00D CAFEF00D
0x20003B60: CAFEF00D CAFEF00D CAFEF00D CAFEF00D
0x20003B70: CAFEF00D CAFEF00D CAFEF00D CAFEF00D
0x20003B80: CAFEF00D CAFEF00D CAFEF00D CAFEF00D
0x20003B90: CAFEF00D CAFEF00D CAFEF00D CAFEF00D
0x20003BA0: CAFEF00D CAFEF00D CAFEF00D CAFEF00D
0x20003BB0: CAFEF00D CAFEF00D CAFEF00D CAFEF00D
0x20003BC0: CAFEF00D CAFEF00D CAFEF00D CAFEF00D
0x20003BD0: CAFEF00D CAFEF00D CAFEF00D CAFEF00D
0x20003BE0: CAFEF00D CAFEF00D CAFEF00D CAFEF00D
0x20003BF0: CAFEF00D CAFEF00D 00000000 CAFEF00D
0x20003C00: 00000004 20001D04 CAFEF00D FFFFEF68
0x20003C10: CAFEF00D 00001098 CAFEF00D CAFEF00D
0x20003C20: CAFEF00D CAFEF00D 00001006 200018D8
0x20003C30: 00000001 200018D8 20001C50 00000004
0x20003C40: 20001BB0 000134A5 0000100D 20001D28
0x20003C50: 00000006 00000006 20001C38 20001D44
0x20003C60: 20001C6C 20001D44 00000006 00000005
0x20003C70: 20001D38 00000005 20001D38 00000000
0x20003C80: 00000001 00012083 200018C8 000013D3
0x20003C90: 00550000 00000001 80E80000 4FC40000
0x20003CA0: 000080E8 00000009 60900000 000080E8
0x20003CB0: 60140000 20002764 0009608F 000080E8
0x20003CC0: 80000000 000080E8 00000000 00129F5F
0x20003CD0: 00000000 0001E4D9 80E80000 200018C8
0x20003CE0: 200018D4 00000000 80E80000 000000FF
0x20003CF0: 0000011C 0001BCE1 0000203A 0001BC1D
0x20003D00: 00000000 0001BC1D 80E80000 0001BCE1
0x20003D10: 0000011C 0001BDA9 80E80000 0001BDA9
0x20003D20: 0000011C FFFFFFF9 008B8000 0001BC1D
0x20003D30: 00000048 00000010 0000A000 00000009
0x20003D40: 0001E326 00000001 80E80000 51538000
0x20003D50: 000080E8 0001E9CF 00000000 00000009
0x20003D60: 61C78000 000080E8 00000048 00000504
0x20003D70: 0000A1FC 0002125C 00000000 000080E8
0x20003D80: 00000000 0012A236 00000000 0001E4D9
0x20003D90: 000080E8 00000009 00004998 000080E8
0x20003DA0: 622C8000 0012A29B 00000042 0001E479
0x20003DB0: 40011000 000185EF 00006E10 00000000
0x20003DC0: 00000000 00000004 0000000C 00000000
```

```
0x20003DD0: 62780000 00018579 2000311B 0001ACDF
0x20003DE0: 00000000 20003054 20002050 00000001
0x20003DF0: 20003DF8 0002085D 00000001 200030D4
0x20003E00: 00000200 0001F663 00000001 200030D4
0x20003E10: 00000001 2000311B 0001F631 00020A6D
0x20003E20: 00000001 00000000 0000000C 200030D4
0x20003E30: 2000311B 00000042 200030D4 00020AD7
0x20003E40: 20002050 200030D4 20002050 00020833
0x20003E50: 20002050 20003F1B 20002050 0001FF89
0x20003E60: 20002050 0001FFA3 00000005 20003ED8
0x20003E70: 20002050 0001FF8B 00000010 00020491
0x20003E80: 00000001 0012A54E 00000020 00022409
0x20003E90: 00000000 20002050 200030D4 0001FF8B
0x20003EA0: 00021263 00000005 0000000C 20003F74
0x20003EB0: 20003ED8 20002050 200030D4 00020187
0x20003EC0: 20003ED4 20003054 00000000 20003F75
0x20003ED0: 00000008 20003F64 00000084 FFFFFFFF
0x20003EE0: FFFFFFFF 00000008 00000001 00000008
0x20003EF0: 20302058 2000311B 0001F631 00020A6D
0x20003F00: 20002050 00000000 0000000C 200030D4
0x20003F10: 32002050 32303032 00323330 000258D7
0x20003F20: 20002050 200030D4 20002050 00020833
0x20003F30: 00000000 20002050 00000020 000001CE
0x20003F40: 20003F40 200030D4 00000004 0001ED83
0x20003F50: 200030D4 20003F60 000001D6 000001D7
0x20003F60: 000001D8 00016559 0000000C 00000000
0x20003F70: 6C383025 00000058 200030D4 FFFFFFFF
0x20003F80: 1FFF4000 00000028 00000028 000217F8
0x20003F90: 200020C7 000166C5 000166AD 00017ED9
0x20003FA0: FFFFFFFF 200020B8 2000306C 200030D4
0x20003FB0: 200020B4 000180AD 1FFF4000 200020B0
0x20003FC0: 200020B0 200020B0 1FFF4000 0001A63D
0x20003FD0: CAFEF00D CAFEF00D 200020B4 00000002
0x20003FE0: FFFFFFFF FFFFFFFF 1FFF4000 00000000
0x20003FF0: 00000000 00000000 00000000 00016113
OK
```

History

This page tracks additions or changes to the AT command set based on the firmware version number (which you can obtain via the 'ATI' command):

Version 0.6.5

The following AT commands were added in the 0.6.5 release:

- AT+BLEGETPEERADDR (http://adafru.it/fuU)

The following commands were changed in the 0.6.5 release:

- Increased the UART buffer size (on the nRF51) from 128 to 256 bytes
- +++ now responds with the current operating mode
- Fixed a bug with AT+GATTCHAR values sometimes not being saved to NVM
- Fixed a bug with AT+GATTCHAR max_len value not being taken into account after a reset (min_len was always used when repopulating the value)

Version 0.6.2

This is the first release targetting **32KB SRAM parts (QFAC)**. 16KB SRAM parts can't be used with this firmware due to memory management issues, and should use the earlier 0.5.0 firmware.

The following AT commands were changed in the 0.6.2 release:

- AT+BLEUARTTX (http://adafru.it/eBK)
  Basic escape codes were added for new lines, tabs and backspace
- AT+BLEKEYBOARD (http://adafru.it/eBK)
  Also works with OS X now, and *may* function with other operating systems that support BLE HID keyboards

Version 0.5.0

The following AT commands were added in the 0.5.0 release:

- AT+BLEKEYBOARDEN (http://adafru.it/eBK)
- AT+BLEKEYBOARD (http://adafru.it/eBK)
- AT+BLEKEYBOARDCODE (http://adafru.it/eBK)

The following AT commands were changed in the 0.5.0 release:

- ATI (http://adafru.it/ei1)
  The SoftDevice, SoftDevice version and bootloader version were added as a new (7th)

record. For Ex: "S110 7.1.0, 0.0" indicates version 7.1.0 of the S110 softdevice is used with the 0.0 bootloader (future boards will use a newer 0.1 bootloader).

Other notes concerning 0.5.0:

Starting with version 0.5.0, you can execute the **AT+FACTORYRESET** command at any point (and without a terminal emulator) by holding the DFU button down for 10 seconds until the blue CONNECTED LED starts flashing, then releasing it.

Version 0.4.7

The following AT commands were added in the 0.4.7 release:

- +++ (http://adafru.it/ei1)
- AT+HWRANDOM (http://adafru.it/ei2)
- AT+BLEURIBEACON (http://adafru.it/ei3)
- AT+DBGSTACKSIZE (http://adafru.it/ei4)
- AT+DBGSTACKDUMP (http://adafru.it/ei4)

The following commands were changed in the 0.4.7 release:

- ATI (http://adafru.it/ei1)The chip revision was added after the chip name. Whereas ATI would previously report 'nRF51822', it will now add the specific HW revision if it can be detected (ex 'nRF51822 QFAAG00')

Version 0.3.0

- First public release

GATT Service Details

Data in Bluetooth Low Energy is organized around units called 'GATT Services (http://adafru.it/ebg)' and 'GATT Characteristics'. To expose data to another device, you must instantiate at least one service on your device.

Adafruit's Bluefruit LE Pro modules support some 'standard' services, described below (more may be added in the future).

UART Service

The UART Service is the standard means of sending and receiving data between connected devices, and simulates a familiar two-line UART interface (one line to transmit data, another to receive it).

The service is described in detail on the dedicated UART Service (http://adafru.it/ekD) page.

UART Service

Base UUID: 6E40**0001**-B5A3-F393- E0A9- E50E24DCCA9E

This service simulates a basic UART connection over two lines, TXD and RXD.

It is based on a proprietary UART service specification by Nordic Semiconductors. Data sent to and from this service can be viewed using the nRFUART apps from Nordic Semiconductors for Android and iOS.

> This service is available on every Bluefruit LE Pro module and is automatically started during the power-up sequence.

Characteristics

Nordic's UART Service includes the following characteristics:

Name	Mandatory	UUID	Type	R	W	N	I
TX	Yes	0x0002	U8[20]	X		X	
RX	Yes	0x0003	U8[20]		X		

R = Read; W = Write; N = Notify; I = Indicate

TX (0x0002)

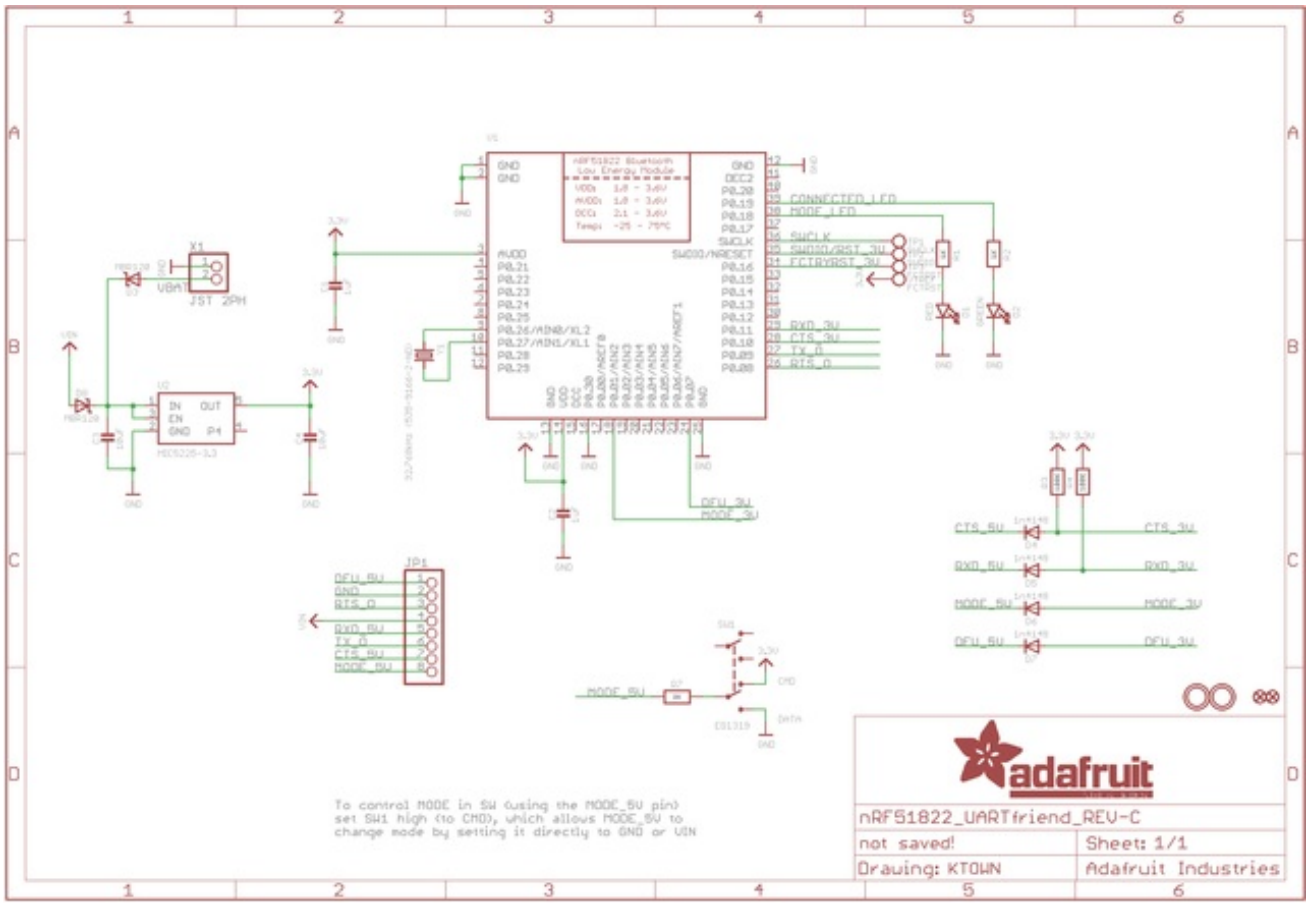
This characteristic is used to send data out to the connected Central device. Notify can be enabled by the connected device so that an alert is raised every time the TX channel is updated.

RX (0x0003)

This characteristic is used to send data back to the sensor node, and can be written to by the connected Central device (the mobile phone, tablet, etc.).

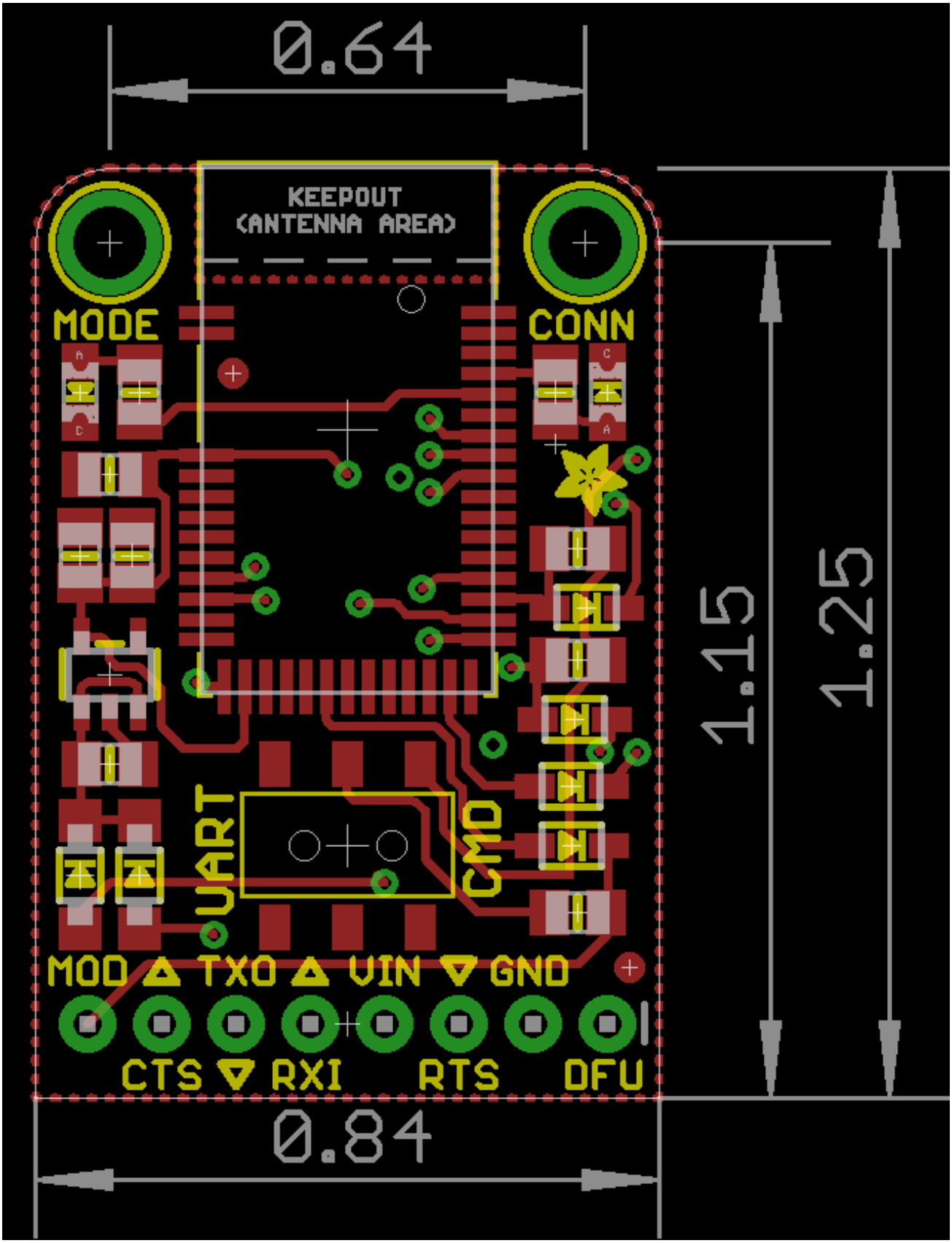
Downloads

Schematic



Board Layout

All measurements are in inches.


0.64
KEEPOUT
(ANTENNA AREA)
MODE
CONN
1.15
1.25
UART
CMD
MOD
TXO
VIN
GND
CTS
RXI
RTS
DFU
0.84