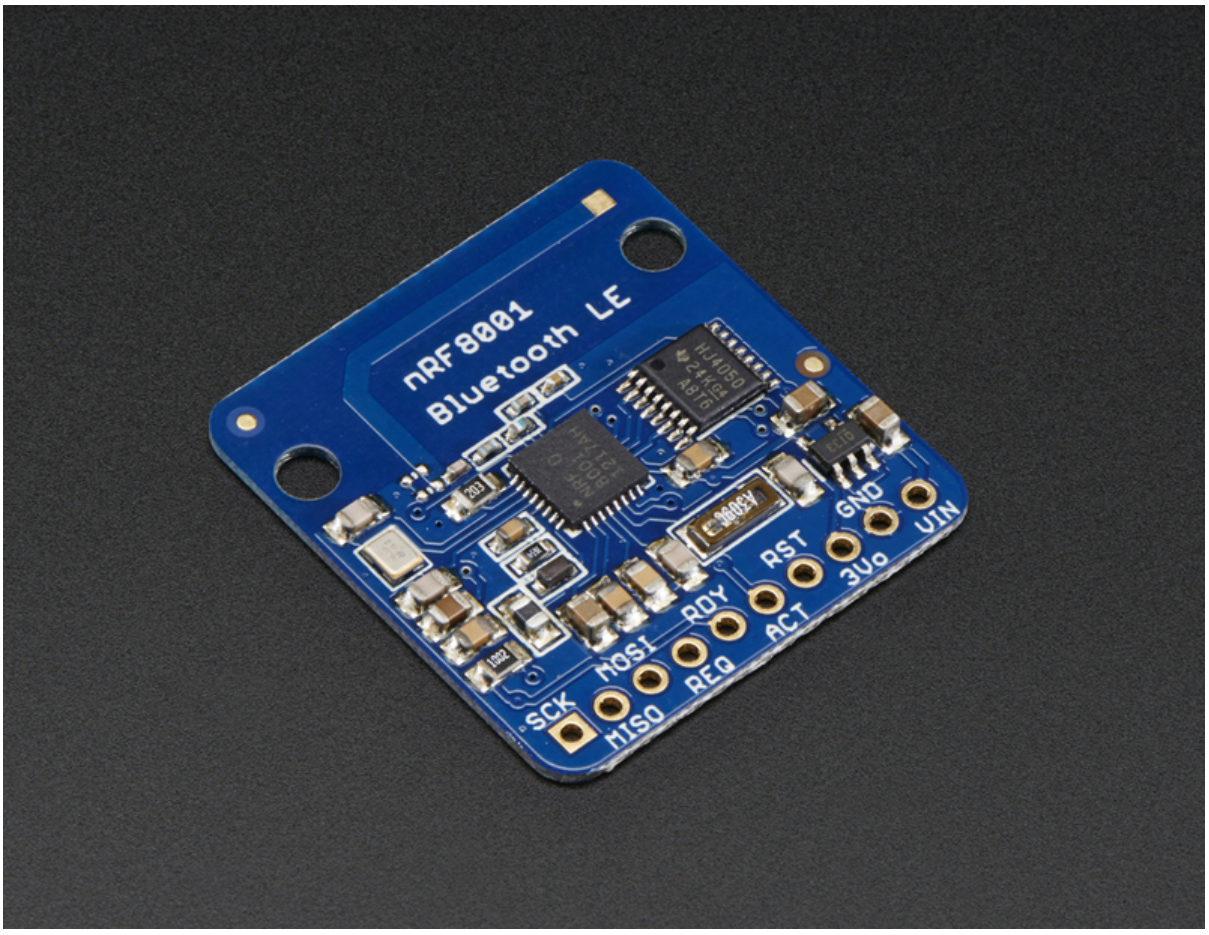
Getting Started with the nRF8001 Bluefruit LE Breakout

Created by Kevin Townsend



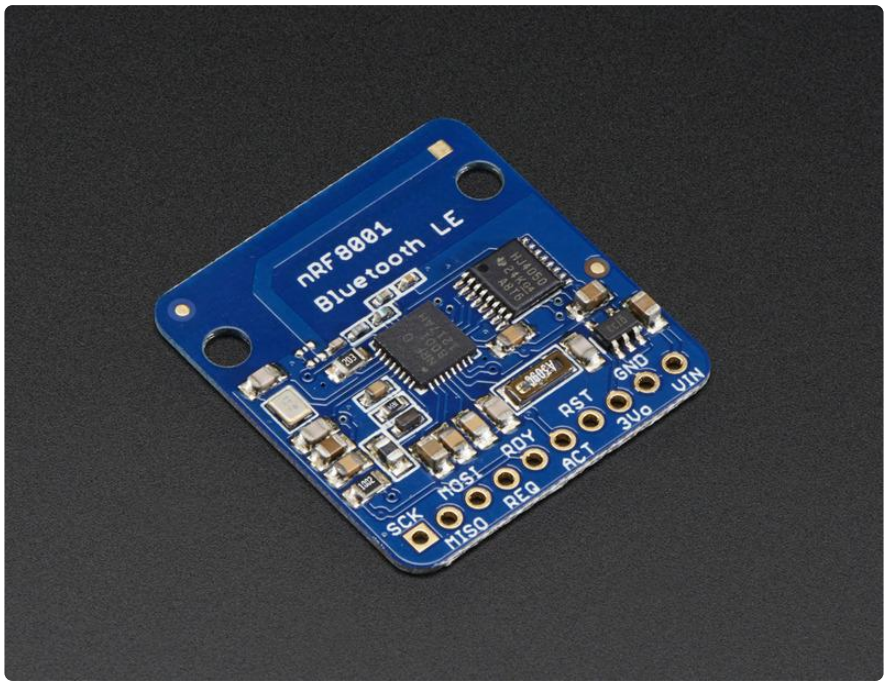
https://learn.adafruit.com/getting-started-with-the-nrf8001-bluefruit-le-breakout

Last updated on 2024-06-03 01:28:42 PM EDT

Table of Contents

- Fabrication Print

Introduction



Our nRF8001 Breakout allows you to establish an easy to use wireless link between your Arduino and any compatible iOS or Android (4.3+) device. It works by simulating a UART device beneath the surface, sending ASCII data back and forth between the devices, letting you decide what data to send and what to do with it on either end of the connection.

Unlike classic Bluetooth, BLE has no big contracts to sign and no major hoops that you have to jump through to create iOS peripherals that you can legally design and distribute in the App Store, which makes it a great choice compared to classic Bluetooth which had (and still has) a lot of restrictions around it on the iOS platform.

And now that Android also officially supports Bluetooth Low Energy (as of Android 4.3), it's also -- finally! -- a universal communication channel covering the main mobile operating systems people are using today.

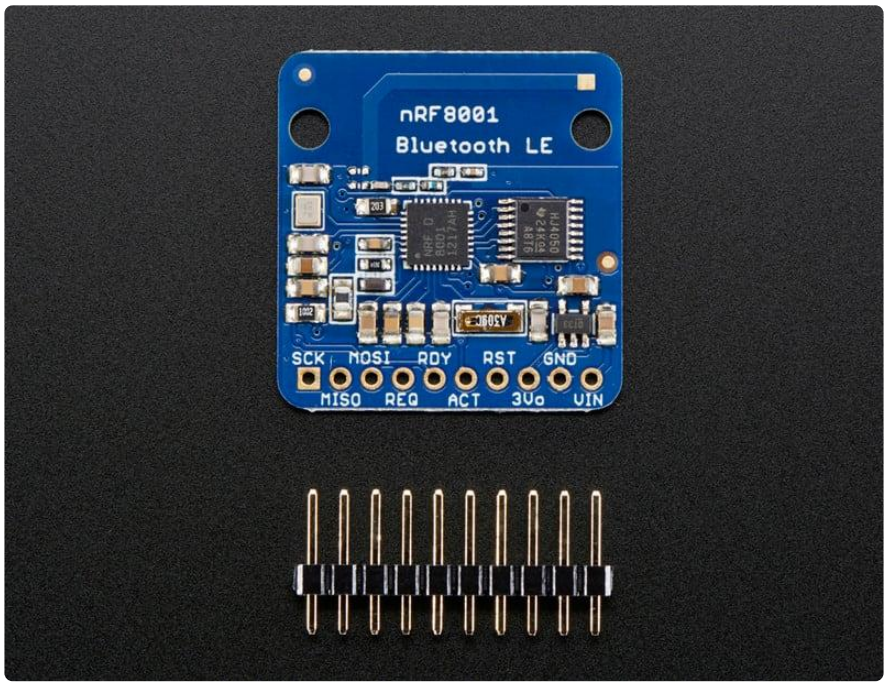
We can get you started super fast with this BLE module which can act like an 'every day' UART data link. Send and receive data up to 10 meters away, from your Arduino to an iOS device. We've even made it easy to get started with our very own BLE connect app that has a "serial console" for sending/receiving data and also an 'arduino pin i/o control station" (https://adafru.it/ddu) to let you set pins on your Arduino to inputs or outputs, high or low logic or even PWM output, as well as read button presses and analog inputs. You can start prototyping your accessory and then use our open source Objective C code to base your new app on! (https://adafru.it/ddv)

The nRF8001 library is not compatible with the Arduino Due at this time

Please note: At this time, we don't have an Android version of the Adafruit Bluefruit LE App available (our native BLE application), but you can use Nordic's Android nRF UART application with the nRF8001 Breakout on BLE capable Android devices (Nexus 4, Nexus 5, Nexus 7, etc.)

This guide will help you setup your nRF8001 Bluetooth Low Energy breakout, and start using some of the sample sketches we provide with it to connect to an iOS or Android device. If you're new to Bluetooth Low Energy, be sure to check out our Introduction to Bluetooth Low Energy (https://adafru.it/dd1) learning guide as well!

At this time, we don't have an Android version of the Adafruit Bluefruit LE App available (our native BLE application), but you can use Nordic's Android nRF UART application on BLE capable Android devices (Nexus 4, Nexus 5, Nexus 7, etc.), or have a look at this Android project by Tony Dicola: https://github.com/tdicola/BTLETest



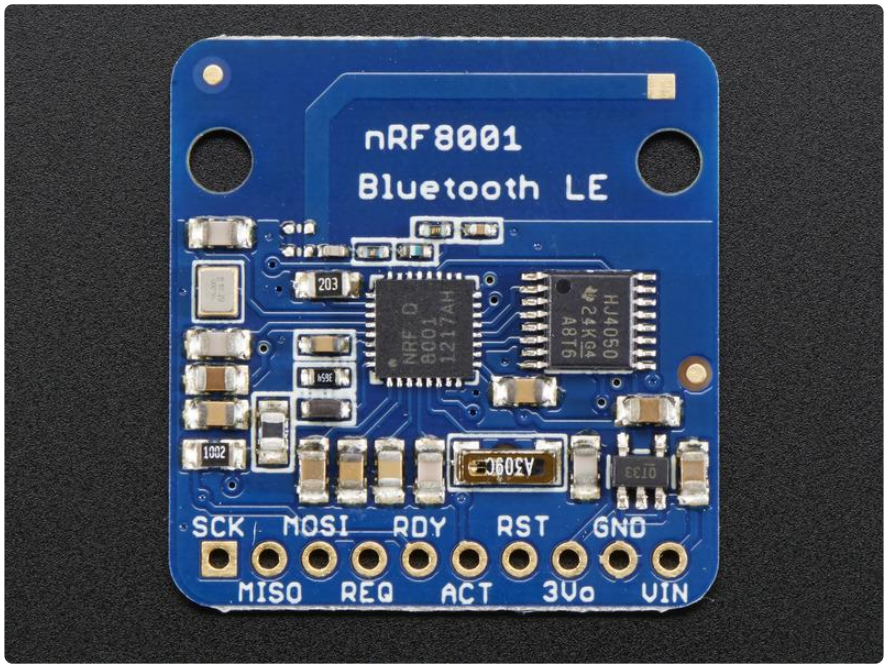
Requirements

- Adafruit nRF8001 Breakout
- A BLE enabled Android or iOS device to test with for nRF UART demos

- An iOS device running iOS 7 with Bluefruit (https://adafru.it/dd2) installed for the BlueFruit LE Firmata demos

---

Pinouts



The nRF8001 is nice because it handles all the BLE radio and low level work, and does it all over SPI which makes it easy to use with any kind of microcontroller. All pins you need are broken out on the bottom of the PCB and all are 5V compliant so you can use with 3V or 5V micros!

Starting from the left:

- **SCK** - this is the SPI data clock pin, connect to your SPI master clock out
- **MISO** - this the SPI data out pin, data is sent from the module on this pin. Data level is 3V but that is fine for 5V microcontrollers.
- **MOSI** - this is the SPI data in pin, data is sent to the module on this pin.
- **REQ** - this is basically what the nRF8001 considers the 'SPI Chip Select' pin, its an input
- **RDY** (ready) - this is the data-ready pin, an interrupt output from the breakout to the microcontroller letting it know that data is ready to read
- **ACT** (active) - this is an output from the module, it lets the host know when the nRF8001 is busy
- **RST** (reset) - this is the reset pin input.
- **3Vo** - this is the output from the onboard **3.3V** regulator, you can grab up to 100mA from this pin.
- **GND** - common ground for data and power

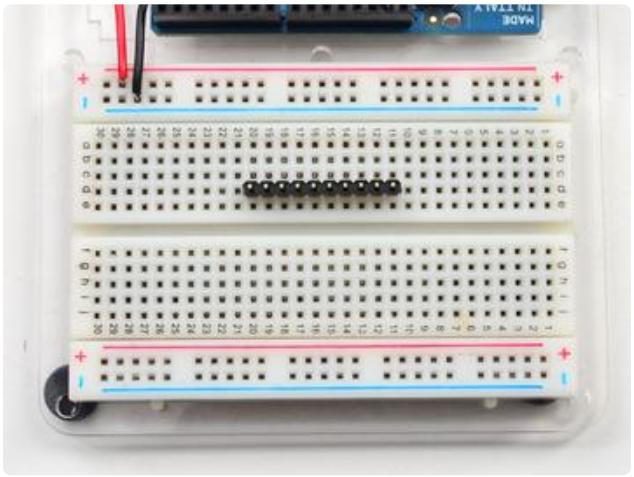
- **VIN** - 3-5 VDC input to power the breakout

Hooking Everything Up

The nRF8001 breakout has full level shifting to make it safe to use with 5V logic, and uses a custom SPI-type bus to talk to the Arduino.

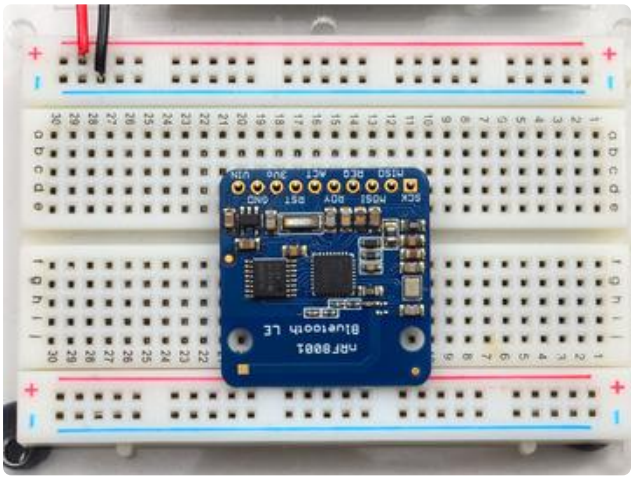
The SPI bus means that this breakout and library will work on any Arduino as long as you're using the hardware SPI pins.

We'll start by attaching headers. You can also solder wires directly but header makes it breadboard friendly!



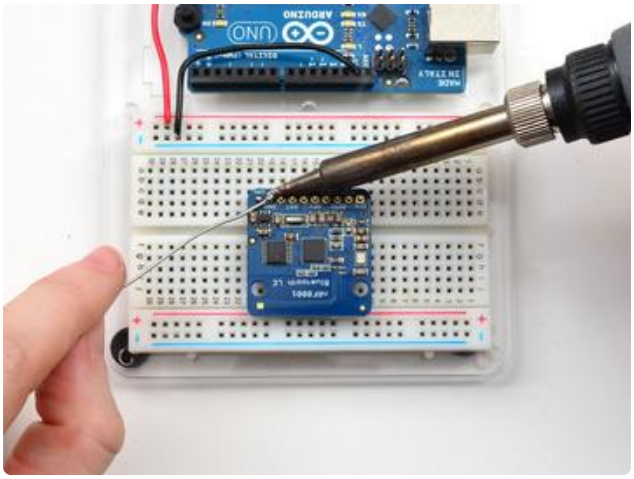
Prepare the header strip:

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



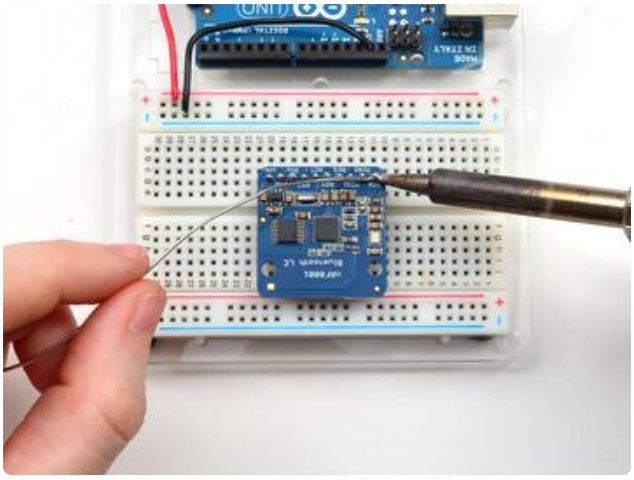
Add the breakout board:

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

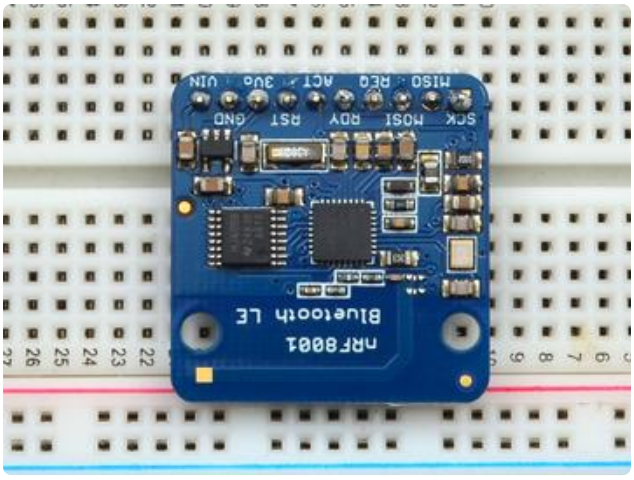


And Solder!

Be sure to solder all 10 pins for reliable electrical contact.



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



That's it! you are now ready to wire and test

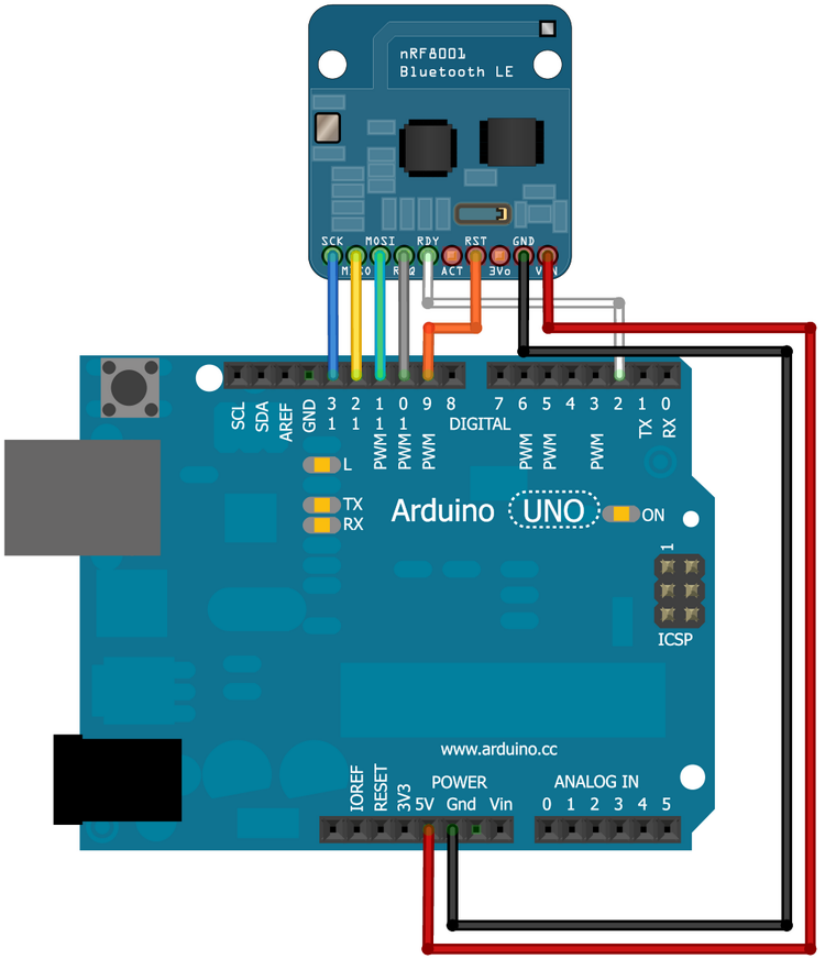
Wiring

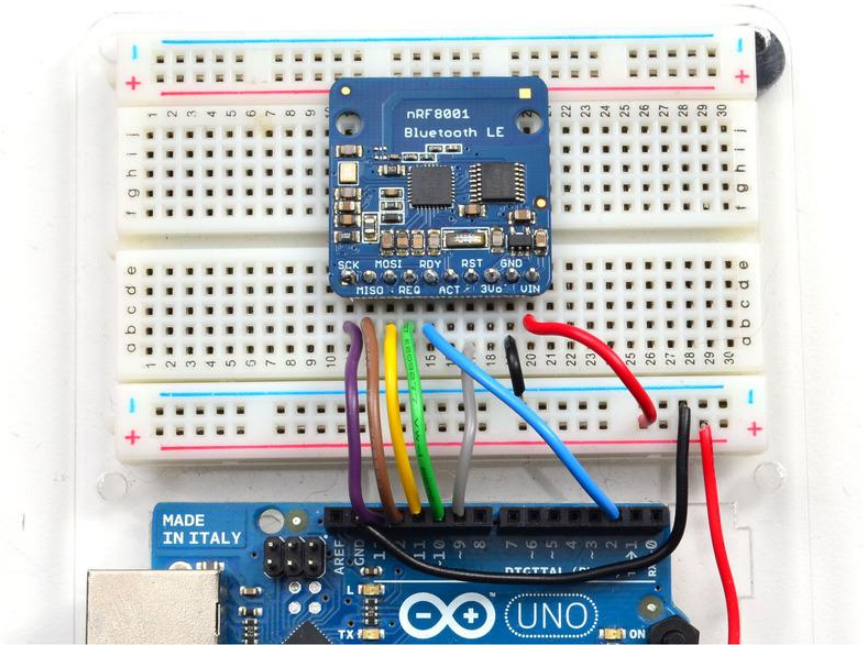
Now that we have headers attached we can easily wire it up to our Arduino

- **VIN** connects to the Arduino **5V** pin
- **GND** connects to Arduino ground

- **SCK** connects to SPI clock.
  On Arduino Uno/Duemilanove/328-based, thats **Digital 13**.
  On Mega's, its **Digital 52** and on
  Leonardo/Micro its **ICSP-3** (See SPI Connections for more details (https://adafru.it/d5h))
- **MISO** connects to SPI MISO.
  On Arduino Uno/Duemilanove/328-based, thats **Digital 12**.
  On Mega's, its **Digital 50** and on
  Leonardo/Micro its **ICSP-1** (See SPI Connections for more details (https://adafru.it/d5h))
- **MOSI** connects to SPI MOSI.
  On Arduino Uno/Duemilanove/328-based, thats **Digital 11**.
  On Mega's, its **Digital 51** and on
  Leonardo/Micro its **ICSP-4** (See SPI Connections for more details (https://adafru.it/d5h))
- **REQ** connects to our SPI Chip Select pin. We'll be using **Digital 10** but you can later change this to any pin
- **RST** connects to **Digital 9** - this is for resetting the board when we start up, you can later change this to any pin
- **RDY** is the interrupt out from the nRF8001, we'll connect to **Digital 2** but be aware that if you want to change it, it must connect to an interrupt capable pin (see this Arduino page for which pins are interrupt-capable (https://adafru.it/dd4). Digital 2 is OK on Uno/Leonardo/Micro/Mega/etc.)

Our code does not currently use the **ACT** pin so you can leave it disconnected





The nRF8001 differs from a classic SPI bus since CS is replaced by two pins, REQ and RDY, but you can still use HW SPI since CS is normally controlled purely in SW anyway.

By connecting 5.0V on the VIN pin, all of the signals will be level shifted between 5V for the Arduino and 3.3V for the nRF8001, meaning you don't need to worry about damaging the IC by providing logic levels that it can't safely handle.

If you are using 3.3V logic, simply connect 3.3V from your development board to the VIN pin on the nRF8001 breakout.

ACT is an optional pin that is not currently used in our sample sketches or low level drivers, but is broken out for future use if required.

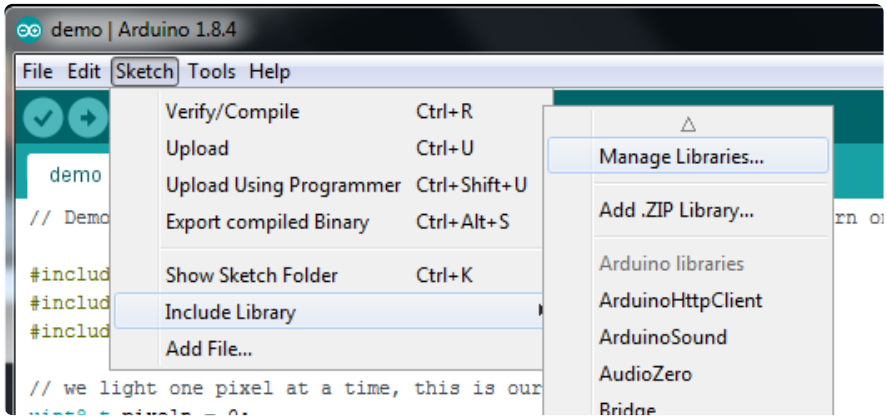
3Vo is the output of the on board 3.3V voltage regulator, and can be used if you need an additional 3.3V supply rail, but generally won't be required on an Uno.

Software: UART Service

Most people understand the basic concept behind UART (one channel to transmit data and one to receive it), so this felt like the easiest way to provide flexible, bi-directional communication between an Arduino and any BLE-enabled mobile platform, without painting people into the corner. BLE does have the capability to handle more complicated structured data, but for the vast majority of people doing projects, UART will get you very very far.

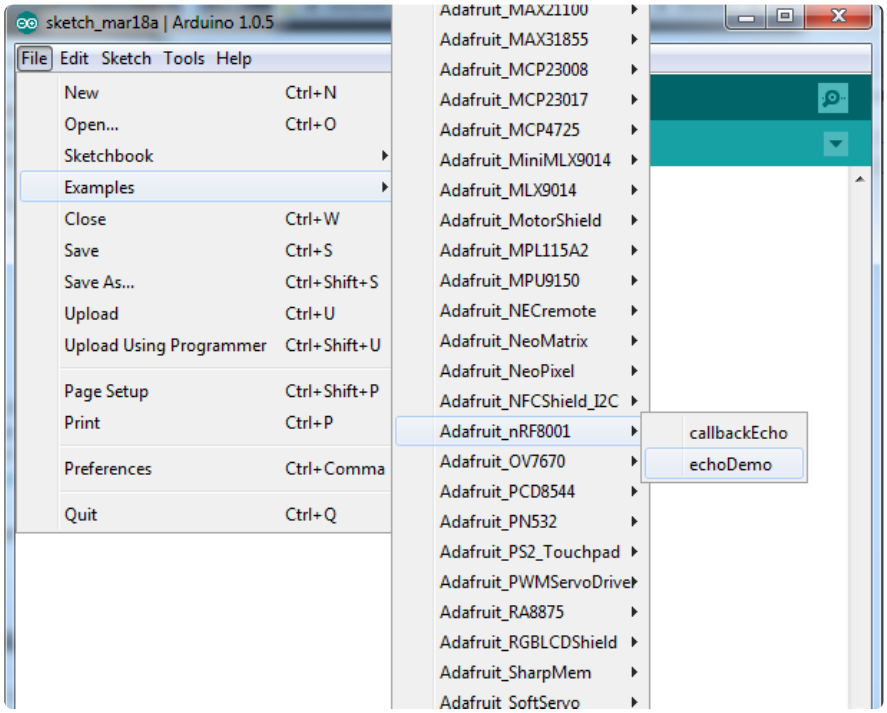
To save everyone the headache of defining and working with custom services, we've wrapped up all of the low level BLE code into a single, easy to use class called **Adafruit_BLE_UART**, available in the nRF8001 / Adafruit_BLE_UART repository on Github (https://adafru.it/dd8)

To install this library, first, open up the Arduino library manager:



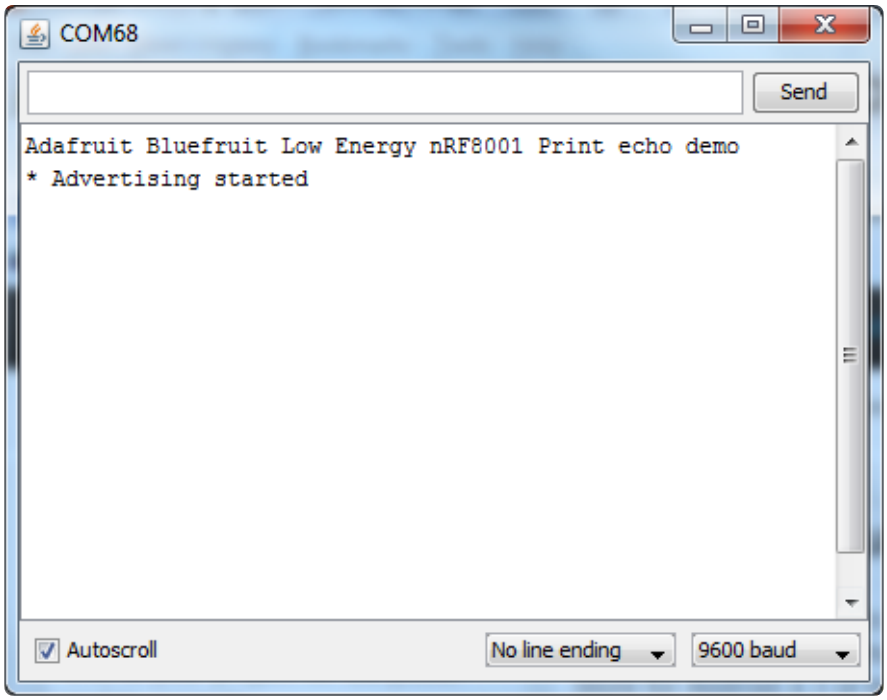
Search for the **Adafruit NRF8001** library and install it

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



Open the 'uart' example via the '**File** > **Examples** > **Adafruit_BLE_UART**> **echoDemo** ' menu item. (The library was renamed from Adafruit_nRF8001 to avoid confusion with the underlying library so the screenshot above is mismatched)

If you upload the demo to your wired-up Arduino and open the serial monitor you should see that it starts advertising BLE signal



Next up we will use our iOS or Android device to make the other side of the connection!

---

nRF UART In Detail

To better understand the BLE UART interface, lets take a look at the basic echo demo. This version is designed to make the BLE breakout be as effortless to use as **Serial**.

Behind the scenes, the library does much of the heavy lifting of managing the connection, sending and receiving data as well as buffering incoming data so you can grab it when the Arduino has time.

The following sketch should allow you to start bi-directional communication on BLE-enabled Android devices (4.3 or higher) or recent iOS devices. It waits for incoming data, and then echoes it back to the transmitting device.

```
// This version uses the internal data queing so you can treat it like Serial
(kinda)!

#include &lt;SPI.h&gt;
#include "Adafruit_BLE_UART.h"

// Connect CLK/MISO/MOSI to hardware SPI
// e.g. On UNO &amp; compatible: CLK = 13, MISO = 12, MOSI = 11
#define ADAFRUITBLE_REQ 10
#define ADAFRUITBLE_RDY 2     // This should be an interrupt pin, on Uno thats #2
or #3
#define ADAFRUITBLE_RST 9

Adafruit_BLE_UART BTLEserial = Adafruit_BLE_UART(ADAFRUITBLE_REQ, ADAFRUITBLE_RDY,
ADAFRUITBLE_RST);
/**************************************************************************/
```

```
/*!
    Configure the Arduino and start advertising with the radio
*/
/**************************************************************************/
void setup(void)
{
  Serial.begin(9600);
  Serial.println(F("Adafruit Bluefruit Low Energy nRF8001 Print echo demo"));

  BTLEserial.begin();
}

/**************************************************************************/
/*!
    Constantly checks for new events on the nRF8001
*/
/**************************************************************************/
aci_evt_opcode_t laststatus = ACI_EVT_DISCONNECTED;

void loop()
{
  // Tell the nRF8001 to do whatever it should be working on.
  BTLEserial.pollACI();

  // Ask what is our current status
  aci_evt_opcode_t status = BTLEserial.getState();
  // If the status changed....
  if (status != laststatus) {
    // print it out!
    if (status == ACI_EVT_DEVICE_STARTED) {
        Serial.println(F("* Advertising started"));
    }
    if (status == ACI_EVT_CONNECTED) {
        Serial.println(F("* Connected!"));
    }
    if (status == ACI_EVT_DISCONNECTED) {
        Serial.println(F("* Disconnected or advertising timed out"));
    }
    // OK set the last status change to this one
    laststatus = status;
  }

  if (status == ACI_EVT_CONNECTED) {
    // Lets see if there's any data for us!
    if (BTLEserial.available()) {
      Serial.print("* "); Serial.print(BTLEserial.available()); Serial.println(F("
bytes available from BTLE"));
    }
    // OK while we still have something to read, get a character and print it out
    while (BTLEserial.available()) {
      char c = BTLEserial.read();
      Serial.print(c);
    }

    // Next up, see if we have any data to get from the Serial console

    if (Serial.available()) {
      // Read a line from Serial
      Serial.setTimeout(100); // 100 millisecond timeout
      String s = Serial.readString();

      // We need to convert the line to bytes, no more than 20 at this time
      uint8_t sendbuffer[20];
      s.getBytes(sendbuffer, 20);
      char sendbuffersize = min(20, s.length());

      Serial.print(F("\n* Sending -&gt; \"")); Serial.print((char *)sendbuffer);
Serial.println("\"");
```

```
      // write the data
      BTLEserial.write(sendbuffer, sendbuffersize);
    }
  }
}
```

Initialization

Lets look at it section by section. Starting with initialization. You'll need to include the header files and define the pins used. Since we're using hardware SPI, the CLK/MOSI and MISO pins are fixed (see the hookup guide)

the **RDY** pin is the only pin that must be an interrupt pin. We'll use 2, most Arduino's can use 2 or 3.

Then create the **Adafruit_BLE_UART** object at the top.

```
#include &lt;SPI.h&gt;
#include "Adafruit_BLE_UART.h"

// Connect CLK/MISO/MOSI to hardware SPI
// e.g. On UNO &amp; compatible: CLK = 13, MISO = 12, MOSI = 11
#define ADAFRUITBLE_REQ 10
#define ADAFRUITBLE_RDY 2     // This should be an interrupt pin, on Uno thats #2
or #3
#define ADAFRUITBLE_RST 9

Adafruit_BLE_UART BTLEserial = Adafruit_BLE_UART(ADAFRUITBLE_REQ, ADAFRUITBLE_RDY,
ADAFRUITBLE_RST);
```

Setup

Setup is easy, just remember to call **begin();** in the setup procedure to begin talking to the **nrf8001**

Polling

During your working loop, you have to give some time to the nRF8001 and tell it to process data. So be sure to call

```
// Tell the nRF8001 to do whatever it should be working on.
BTLEserial.pollACI();
```

as often as possible - and if you're having issues where data rates seem slow, try speeding up your loop

It's important to constantly call pollACI if you want to efficiently handle data over BLE. Be sure to include this function at the top of your 'loop' function in your sketch.

Managing Status

BLE is very asynchronous, it can connect, disconnect, time out. Part of the niceness of BTLE compared to classic BT is that this is all much more stable. Reconnecting takes less than half a second instead of up to 20 seconds! Be sure to check in with the nRF8001 often to see if the see the state has changed. We suggest keeping a global variable for the last known status so you can see if its changed

```
aci_evt_opcode_t laststatus = ACI_EVT_DISCONNECTED;
```

and then calling **getState()** to query the latest state. If something's changed, you can notify the user:

```
// Ask what is our current status
aci_evt_opcode_t status = BTLEserial.getState();
// If the status changed....
if (status != laststatus) {
  // print it out!
  if (status == ACI_EVT_DEVICE_STARTED) {
      Serial.println(F("* Advertising started"));
  }
  if (status == ACI_EVT_CONNECTED) {
      Serial.println(F("* Connected!"));
  }
  if (status == ACI_EVT_DISCONNECTED) {
      Serial.println(F("* Disconnected or advertising timed out"));
  }
  // OK set the last status change to this one
  laststatus = status;
}
```

Valid events are:

- **ACI_EVT_DEVICE_STARTED**: The device has started advertising, and can be detected by other devices in listening range
- **ACI_EVT_CONNECTED**: A connection has been established with another devices (meaning that advertising will now stop)
- **ACI_EVT_DISCONNECTED**: The connection with the external device was closed or timed out

By detecting the event type, we can perform an action like enabling an LED when we are connected, or no longer reading sensor data when we are disconnected, etc.

Reading data

If data is available, you can query it with **available()** which will return the number of bytes waiting. You can then read one byte at a time with **read()** just like you would with **Serial**

Writing data

The nRF8001 sends out packets of data, 20 bytes at time. Keep this in mind if you want to send a lot of data it will be packetized into chunks of 20. You can of course send less than 20 bytes.

Much like **Serial** you can use the **.write** and **.print** functions allow us to send data out to the connected device:

(https://adafru.it/ddw) (https://adafru.it/ddx)uint16_t write (uint8_t singlebyte)

Writes a single byte to the connected device, and returns the number of bytes successfully written.

(https://adafru.it/ddw) (https://adafru.it/ddx)uint16_t write (uint8_t * buffer, uint8_t len)

Writes **len** bytes from **buffer** to the connection device, and returns the number of bytes successfully written.

(https://adafru.it/ddw) (https://adafru.it/ddx)uint16_t print("text here")

Prints the supplied string to the connected device, and returns the number of bytes successfully written. This is simple a helper function that points to **.write**, but may be easier to work with since it follows the same naming conventions as the familiar Serial class on Arduino.

(https://adafru.it/ddw) (https://adafru.it/ddx)uint16_t println("text here")

Similar to the print function above, but appends the string with new line characters at the end of the string, similar to the difference between **Serial.print** and **Serial.println** on Arduino.

Try to keep the buffers and strings under 20 bytes. The library will split up large

messages but often times the app on the other side wants to read the whole packet at once, and it can make your job a lot easier!

Software: nRF UART App

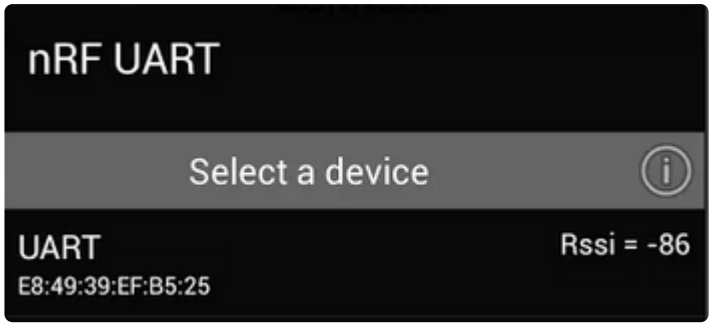
In order to test the sketch described on the previous page, you can use a free UART application from Nordic Semiconductors (https://adafru.it/dd5) that's available in Apple's app store for recent iOS devices or Android's Play Store for Android 4.3 or higher devices.

Android: nRFUART 2.0

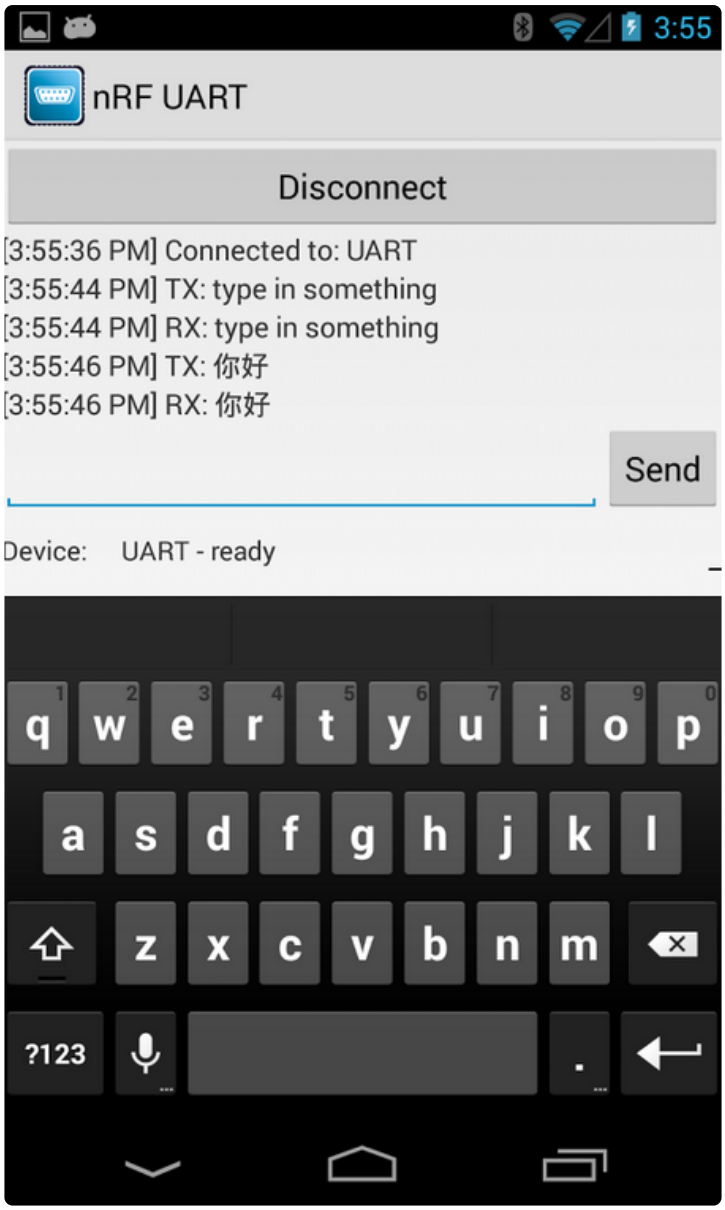
- Go to the Play Store and search for nRFUART 2.0 (https://adafru.it/dd6), then install the application. If you can't find this application, your Android device probably doesn't support BLE or isn't running Android 4.3+!
- Load the 'callbackEcho' sketch onto your Arduino (**File** > **Examples** > **Adafruit_nRF8001** > **callbackEcho**)
- Run the sketch and open the Serial Monitor (Baud Rate = 9600)

Be sure to use the 2.0 version of the app on Android. The earlier (non 2.0) version is based on a proprietary BLE stack for certain Samsung devices, which was created before Google added official support for BLE in Android 4.3.

Once the device starts advertising, you can open the nRFUART 2.0 application, and you should be able to connect to the 'UART' device, similar to the screenshot below:



Once you're connected, you can click on the 'send' textbox at the bottom, and any data you send out should show up in the Serial Monitor, and also get echoed back to the Android application, as seen below:



You will need an Android device running Android 4.3 or higher with BLE support to use this application. Nexus 4, Nexus 5 and Nexus 10 devices running the latest version of Android can all use this application, but other devices will need to be verified for BLE support.

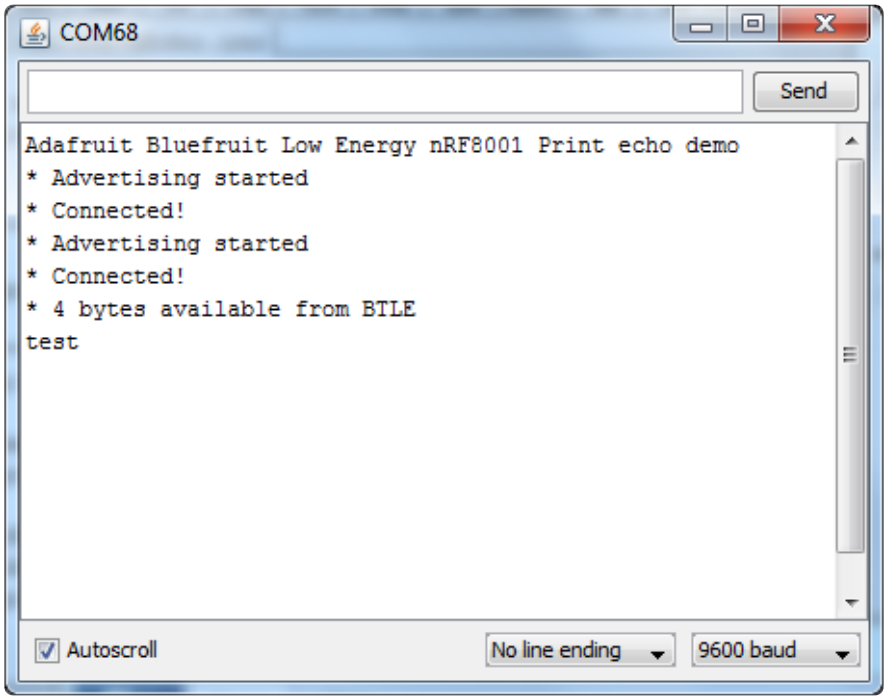
If you wish to create your own Android BLE UART project, you can have a look at some Android source code from Tony Dicola that works with our UART service here: https://github.com/tdicola/BTLETest

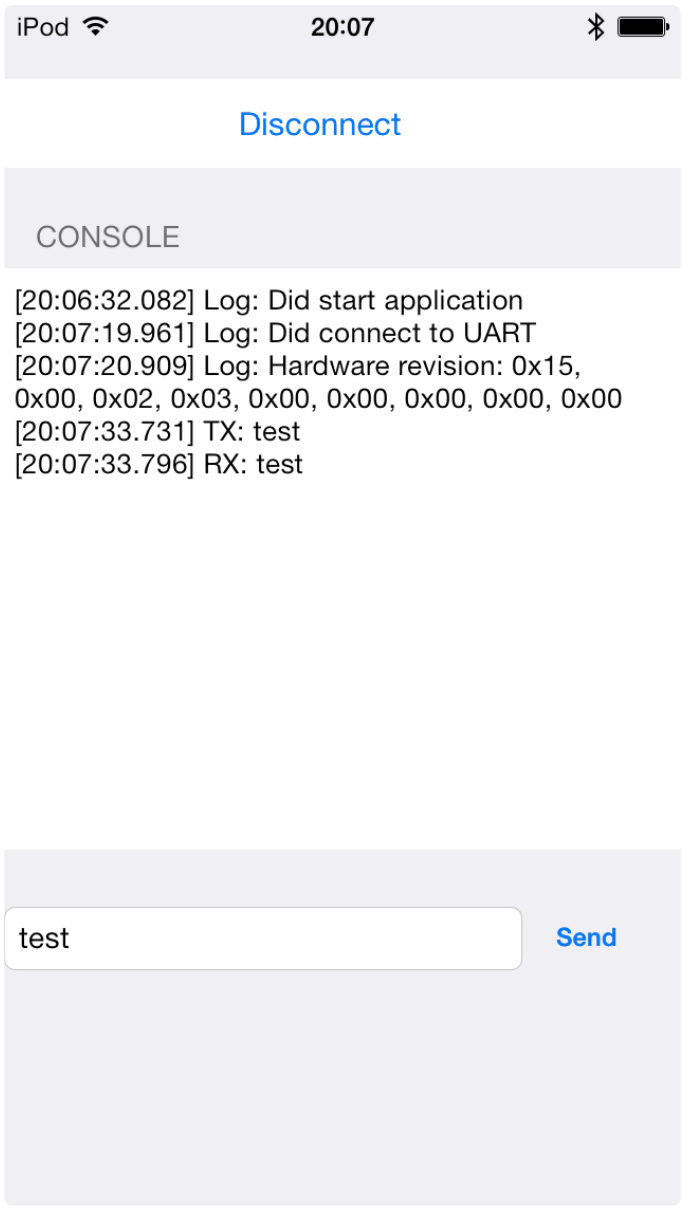
iOS: nRF UART

If you are using a BLE-enabled iOS device (recent iPhones, iPod Touch models, iPads, etc.), you can also test this on iOS.

- Download nRF UART (https://adafru.it/dd7) application from Apple's App Store.
- Load the 'callbackEcho' sketch onto your Arduino (**File** > **Examples** > **Adafruit_nRF8001** > **callbackEcho**)
- Once the sketch is running, open up the Serial Monitor at 9600 baud.

You should be able to connect to the board using the 'Connect' button in the iOS application now, and send and receive text via the textbox at the bottom of the app:

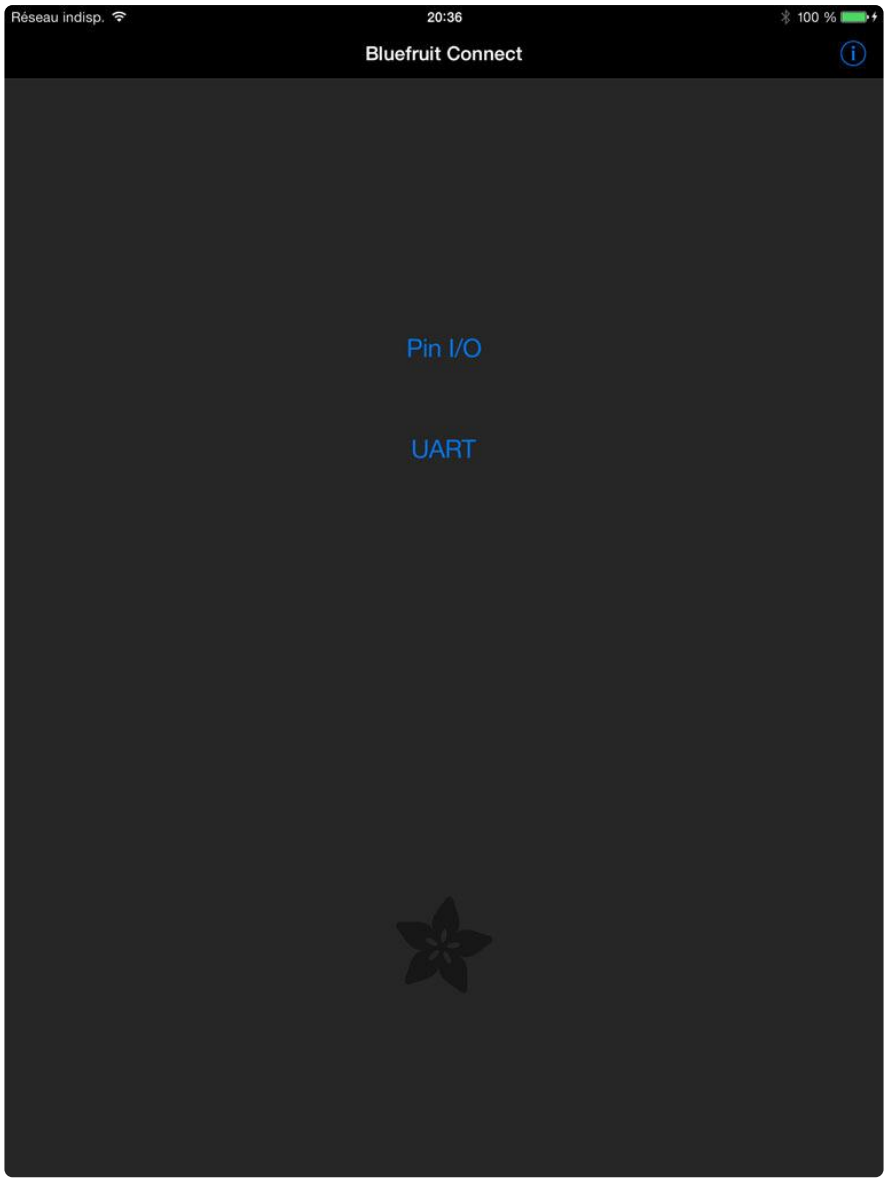




Software: BlueFruit UART App

If you're using an iOS based device, we've made your life easy with our BlueFruit application (https://adafru.it/dd2), which is available in Apple's App Store.

This free iOS application allows you to send or received UART messages between your iOS device and the nRF8001 (select **UART** on the home page), or toggle pins from the iOS UI setting them to input, output or as PWM (select **Pin I/O** discussed in the next page)



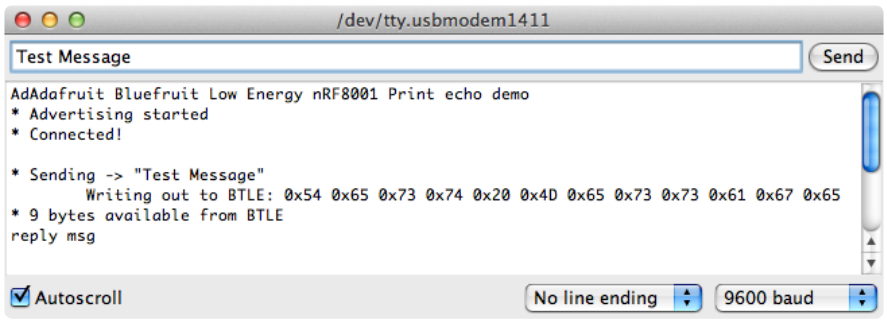
UART Echo Demo

This UART is basically the same as nRF's but its a little more like a terminal window instead of a timestamped log.

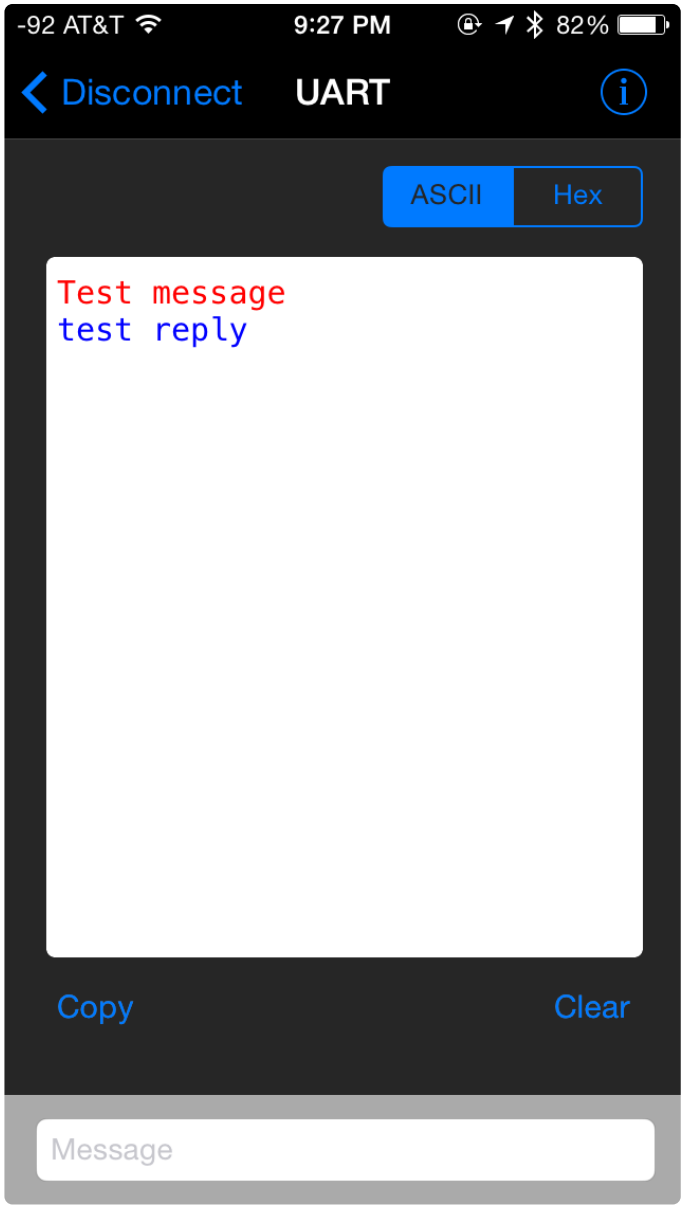
The **echoDemo** example sketch allows you to send and receive simple messages using **Serial-esque** style commands, and the data will be displayed on both BlueFruit on the iOS device and the Serial Monitor on the Uno.

After programming the Uno with the sketch, you can open up the Serial Monitor (make sure it's set to 9600 baud!), and then open up the BlueFruit application on your iOS device and select **UART** on the home screen. It should connect!

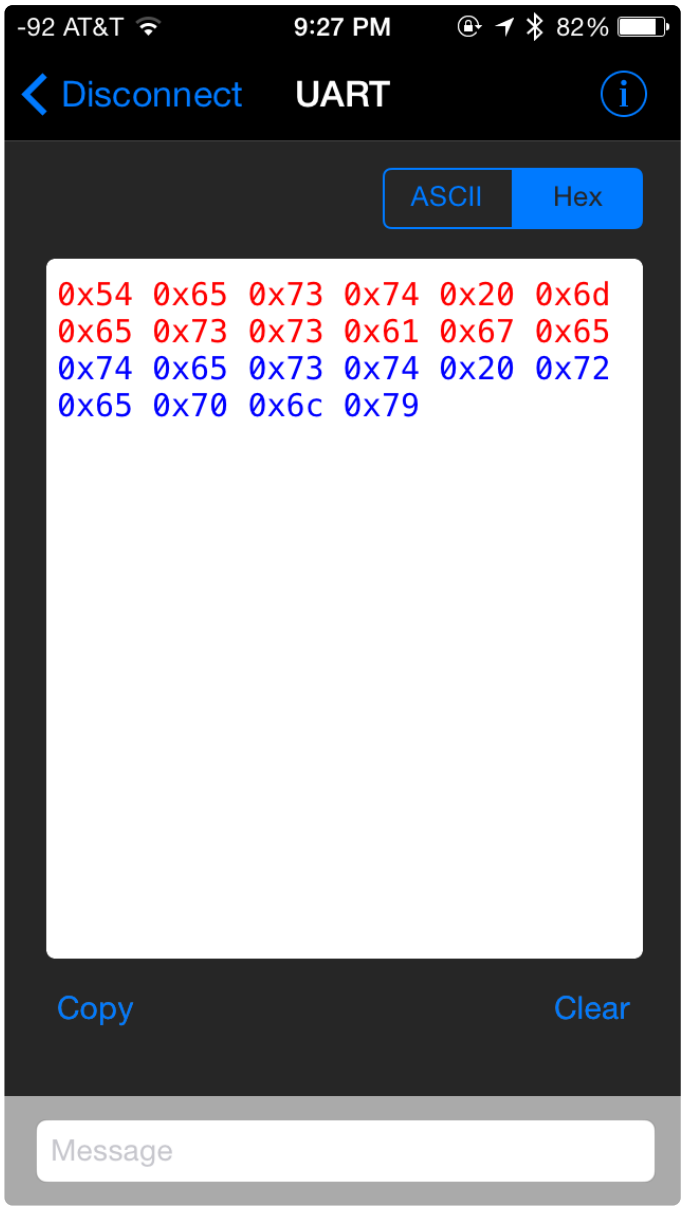
Now, any data that you enter on the iOS device or the Uno will be transmitted to the other device as long as the connection is open:



The corresponding BlueFruit output can be seen below, where the red message is incoming data and the blue message is outgoing data.



Click the **HEX** button in the top right to switch over to hex display mode instead of plain 'ascii' mode



Software: BlueFruit Pin I/O

In addition to the UART functionality in BlueFruit (https://adafru.it/dd2), you can also use Firmata to control the pins on your Uno.

Firmata (https://adafru.it/dda) is a light weight protocol that was designed to make it possible to control an Uno from a variety of external devices, such as you laptop using another programming language. We've ported Firmata over to BLE using our Adafruit_BLE_UART as the transport layer, and created an easy to use IDE to help you get started with it.

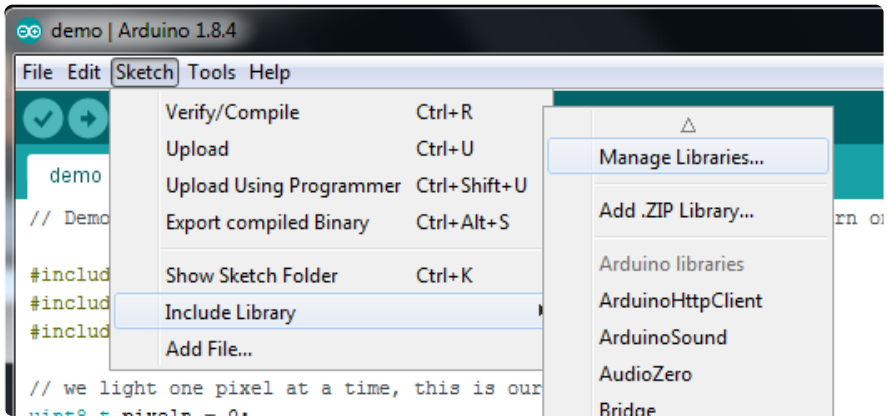
At this time, our Firmata sketch/App support is limited to iOS devices. BLE is relatively new to the Android ecosystem and there are only a handful of devices that support it today, and the stack itself is still in active development and has

some issues that will no doubt be resolved in future updates. For the moment, though, we have made the decision to concentrate our limited resources on iOS since this is the still statistically the most natural target plaform in the BLE world.

BLE StandardFirmata

The first thing you'll need to do is download the Adafruit_BLE_PinIO (https://adafru.it/fTO) repository from the Arduino library manager.

Open up the Arduino library manager:

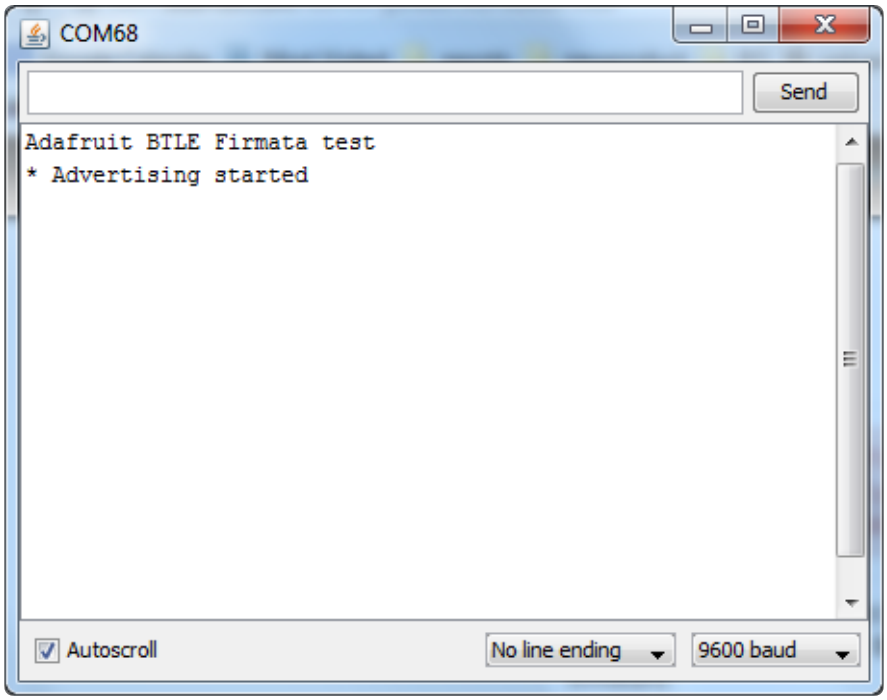


Search for the **Adafruit BLEFirmata** library and install it

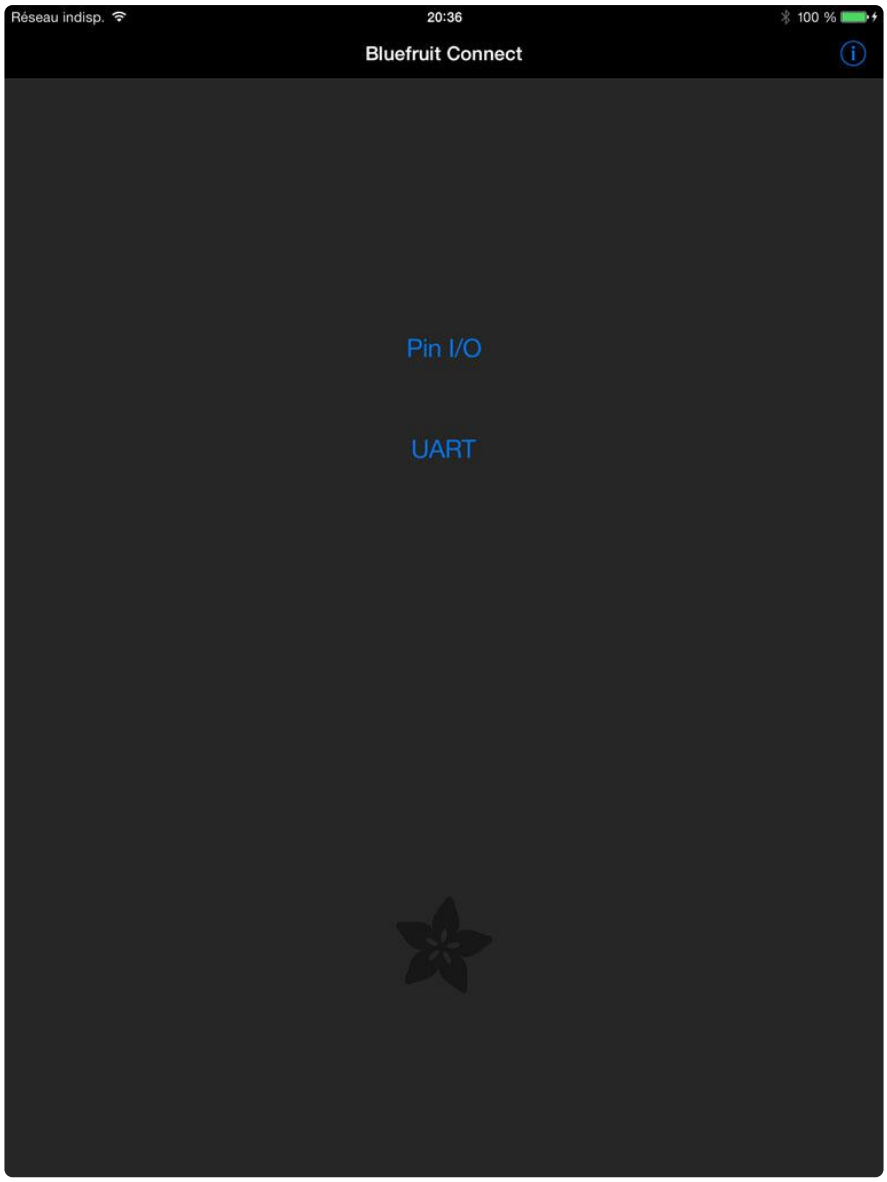
We also have a great tutorial on Arduino library installation at: http://learn.adafruit.com/adafruit-all-about-arduino-libraries-install-use (https://adafru.it/aYM)

The **Adafruit_PinIO** sketches also requires **Adafruit_nRF8001** to be present in your libraries folder but you already installed that so you should be good to go if you went through the UART echo tests.

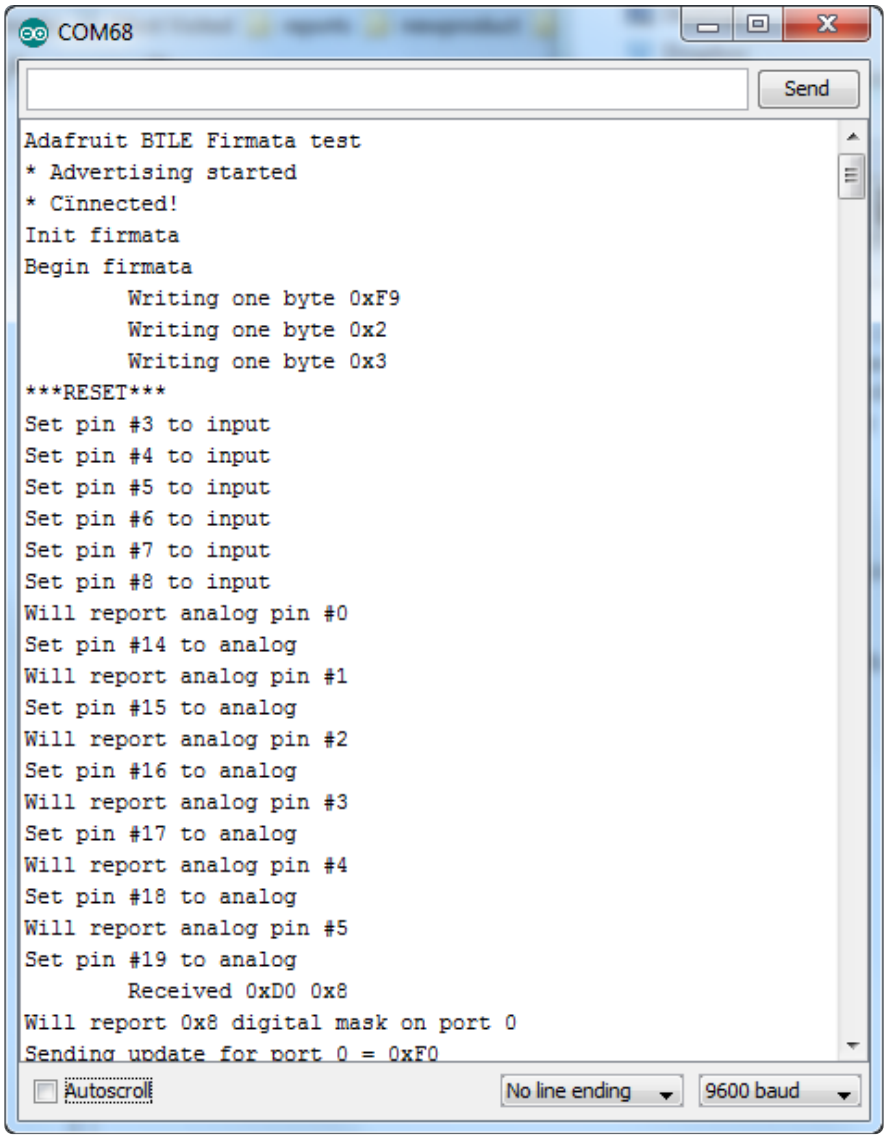
Once this library is installed, open up the **StandardFirmata** sketch (**File** > **Examples** > **Adafruit_BLEFirmata** > **StandardFirmata**), compile the sketch, and program the Uno with your firmware.



Next, open Adafruit Bluefruit LE Connect on your iOS device and select the **Pin I/O** option on the home page:



This will establish a connection between the nRF8001 and your iOS device, and you should see an I/O screen that allows you to select any available pin.



Wiring up for Firmata demo

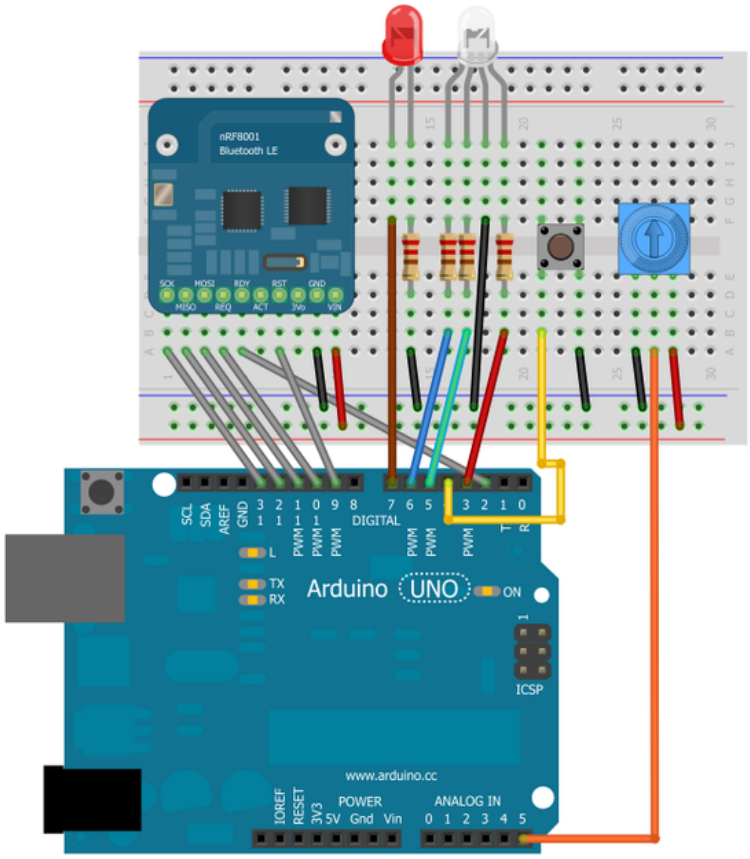
The Firmata BLE app demo allows you to some basic functionality with your Arduino, great for testing out ideas or sensors

- Digital Input (e.g. switches)
- Digital Output (e.g. relays)
- Analog Input (e.g. sensors)
- PWM Output (e.g. LED dimming)

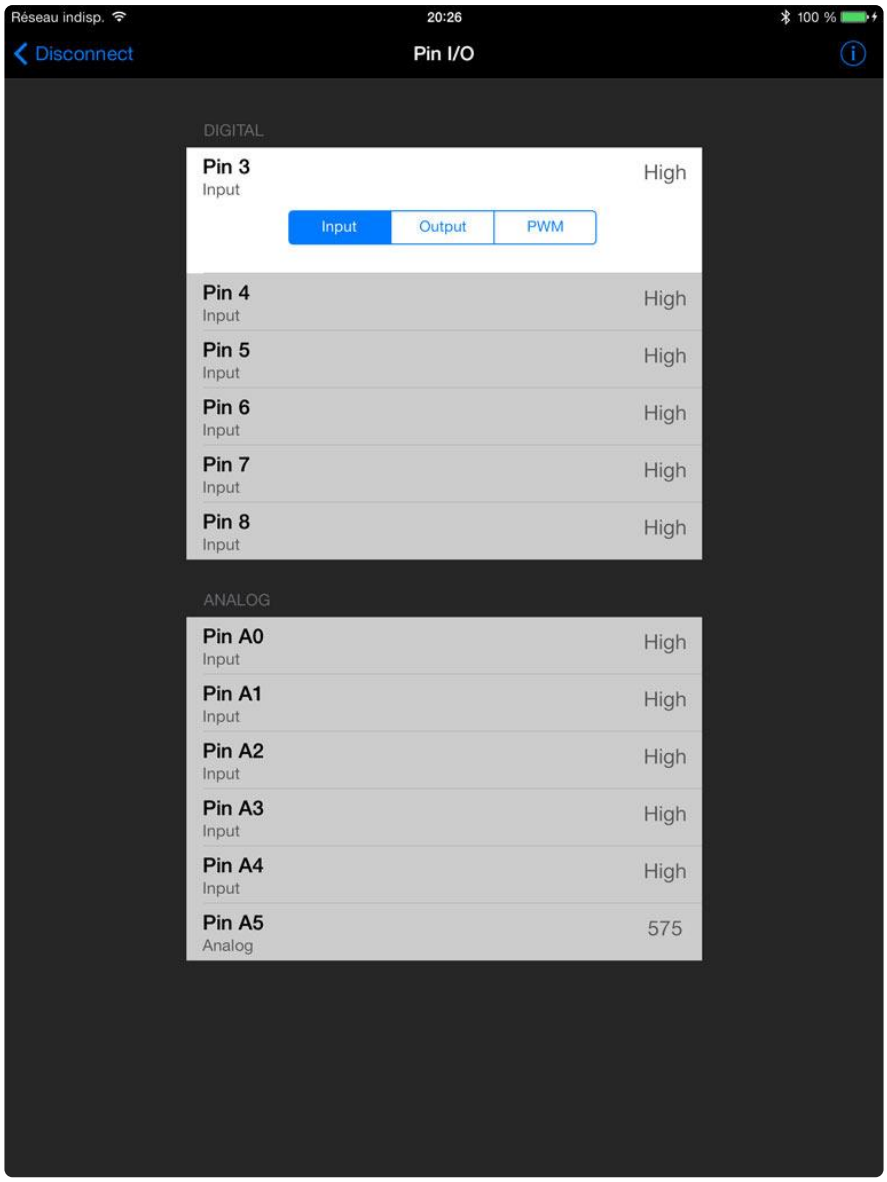
We'll demo all of these with the following wiring, grab some components from your parts bin and follow along!

- Connect a standard LED (any color) with a inline resistor (220-1K is fine) to **Digital 7**

- Connect an RGB LED (either common cathode or anode) so that the red, green and blue LED pins tie to **Digital 3 5** and **6** with inline resistors. If using common anode, connect the fourth pin to 5V. If using common cathode, connect it to GND.
- Connect a switch of some sort to **Digital 4** so that when pressed, it connects to ground. No pullup resistor is required
- Connect a potentiometer (any value 500 ohm to 1Mohm) so that the two outer legs connect to **5V** and **GND** and the middle pin connects to **Analog 5**



Simply click on the pin that you wish to manipulate (pin 3 is selected in the screenshot below), set one of the three pin modes (**Input, Output**, **PWM** or **Analog** mode), and adjust the settings accordingly:



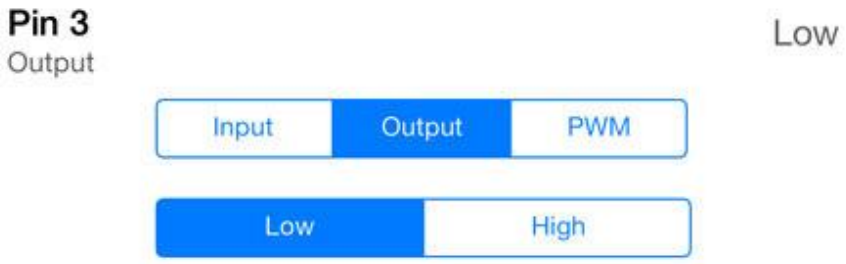
Some of the various options can be seen below, such as the ability to change the PWM rate when you select PWM mode, or whether to set output pins high or low, etc.:

Input Mode

This mode will setup the pin as an input, and the latest pin state will be displayed as **High** or **Low**:

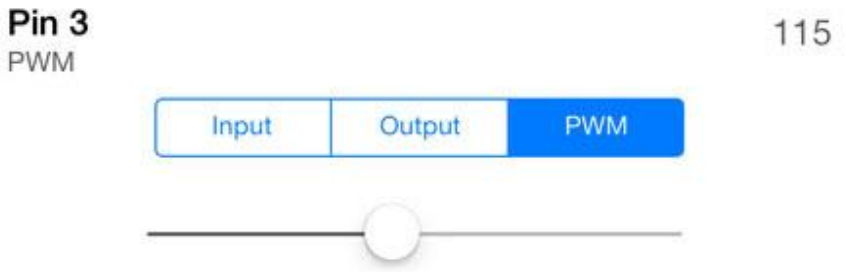
Output Mode

In Output Mode you can set the pin state yourself to **High** or **Low**, allowing you to manually toggle an LED, enable or disable a FET driving a heavy load, etc.:



PWM Mode

PWM Mode allows you to set adjust the PWM output on a pin between 0 and 255 using a convenient slider, as shown below:



Adding App Support

While we don't have a tutorial yet on creating your own custom applications on iOS, Android or any other BLE-enabled operating system, the following information will be useful to any application developers, and you're free to look at our open source code for our own iOS application (https://adafru.it/ddv).

Tony Dicola has also published some source code for Android around our BLE UART service, which you can consult on github (https://adafru.it/drl).

The UART Service

For reasons that are clearly beyond the comprehension of mere mortals like us, the Bluetooth SIG has decided not to include a UART-type service in the list of officially accepted BLE service definitions (https://adafru.it/ddl).

Without an equivalent to SPP in Bluetooth Classic, we only have one choice ...

defining and implementing a custom UART-esque service ourselves!

The custom UART service uses the following UUIDs, which are the values you need to know to make your application talk to the appropriate characteristic. There is one characteristic for TX and another for RX, similar to the way that UART uses two lines to send and receive data:

- **UART Service UUID**: 6E400001-B5A3-F393-E0A9-E50E24DCCA9E
- **TX Characteristic UUID**: 6E400002-B5A3-F393-E0A9-E50E24DCCA9E
- **RX Characteristic UUID**: 6E400003-B5A3-F393-E0A9-E50E24DCCA9E

These are the same UUID values used by Nordic Semiconductors in their test applications to stay compatible with their iOS and Android utilities

Using some sample code for your target OS (the Application Accelerator (https://adafru.it/ddJ) code from Bluetooth is a good start for iOS, Android or Windows), you can connect to the nRF8001 Breakout, find the UART service via the service UUID above, and then transfer data back and forth over the two available characteristics.

If you're new to Bluetooth Low Energy and don't know what characteristics and services are, have a look at our helpful Introduction to Bluetooth Low Energy (https://adafru.it/dd1) learning guide as well, which lists some useful development resources at the end!

Related Links

The following links may be useful to you working with the nRF8001 Breakout:

Adafruit Resources

- Adafruit_nRF8001 (https://adafru.it/dd8) drivers and samples sketches
- Adafruit BlueFruit LE Connect (https://adafru.it/dd2) iOS Application
- Adafruit's Introduction to Bluetooth Low Energy (https://adafru.it/dd1) learning guide

General Resources

- Bluetooth Core Specification (https://adafru.it/ddd) (BLE was introduced as part of the 4.0 core spec)
- Bluetooth Development Portal (https://adafru.it/dde)

- Nordic Semiconductor's nRF8001 (https://adafru.it/ddf) product page

If you have any specific problems with the Adafruit nRF8001 breakout, fee free to visit our actively moderated support forums (https://adafru.it/forums), though be sure to check for the latest code on github (https://adafru.it/dd8) first since that's the first place new features and bug fixes will be introduced!

F.A.Q.

I'm having connection dropouts in Android, whats up with that?

Android devices have some incompatibilities with 5GHz wifi on at the same time as BTLE, try disabling 5GHz wifi!

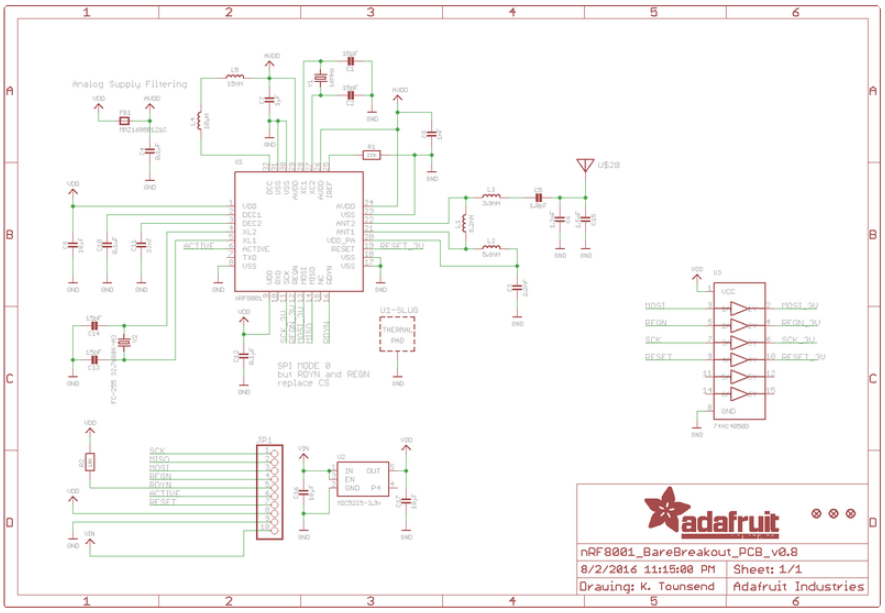
See for more details: https://code.google.com/p/android/issues/detail?id=63056 (https://adafru.it/eUJ)

Downloads

Datasheets & Files

- Nordic Semiconductor's nRF8001 (https://adafru.it/ddf) product page
- Fritzing object in Adafruit Fritzing library (https://adafru.it/aP3)
- EagleCAD PCB files in GitHub (https://adafru.it/rqE)

Schematic



Fabrication Print