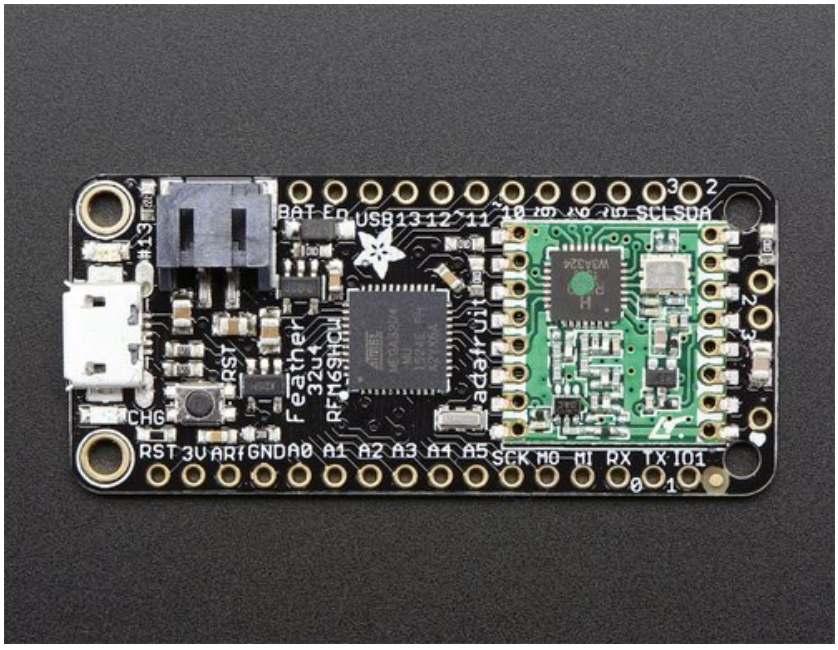
Adafruit Feather 32u4 Radio with RFM69HCW Module

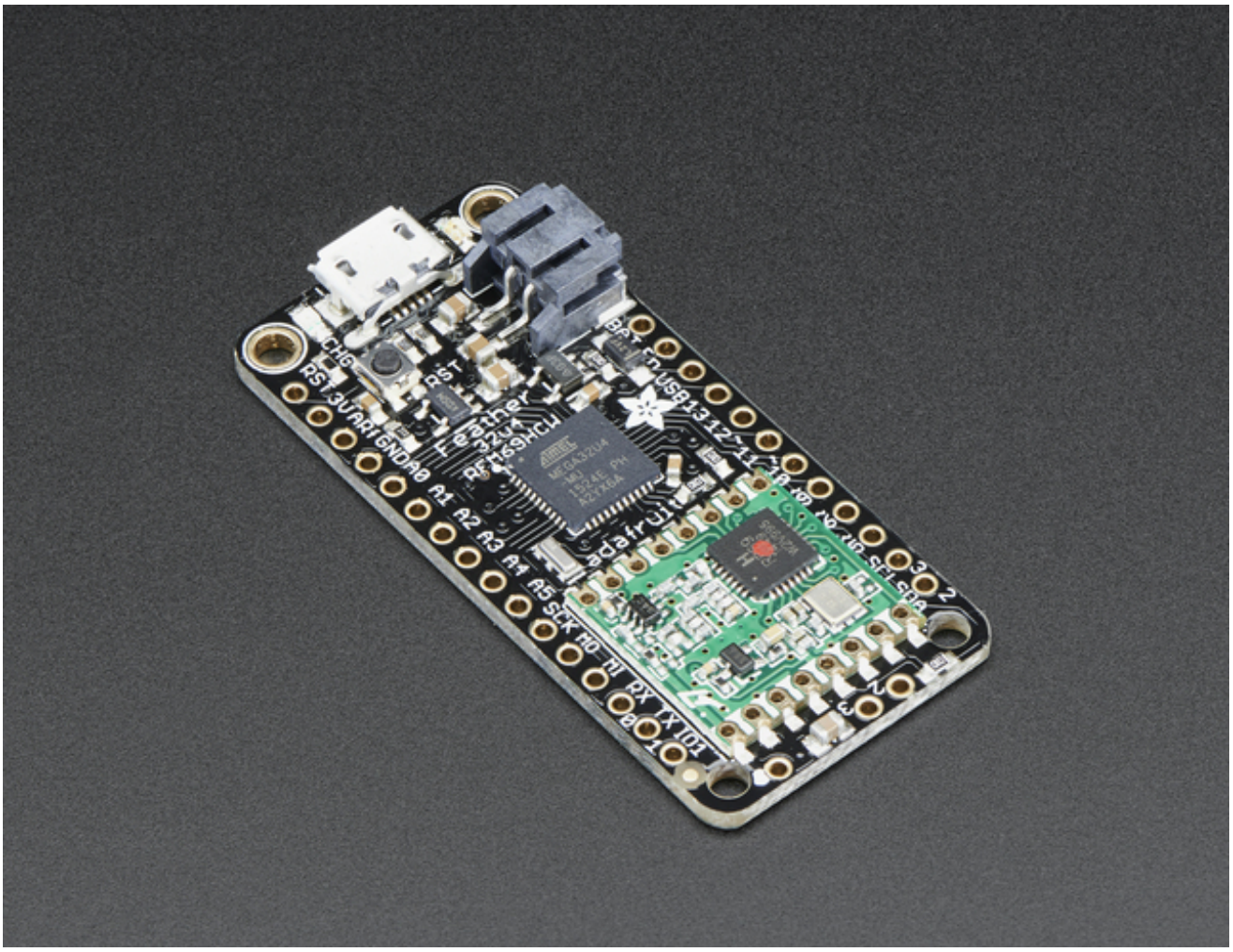
Created by lady ada



Last updated on 2017-05-01 09:13:25 PM UTC

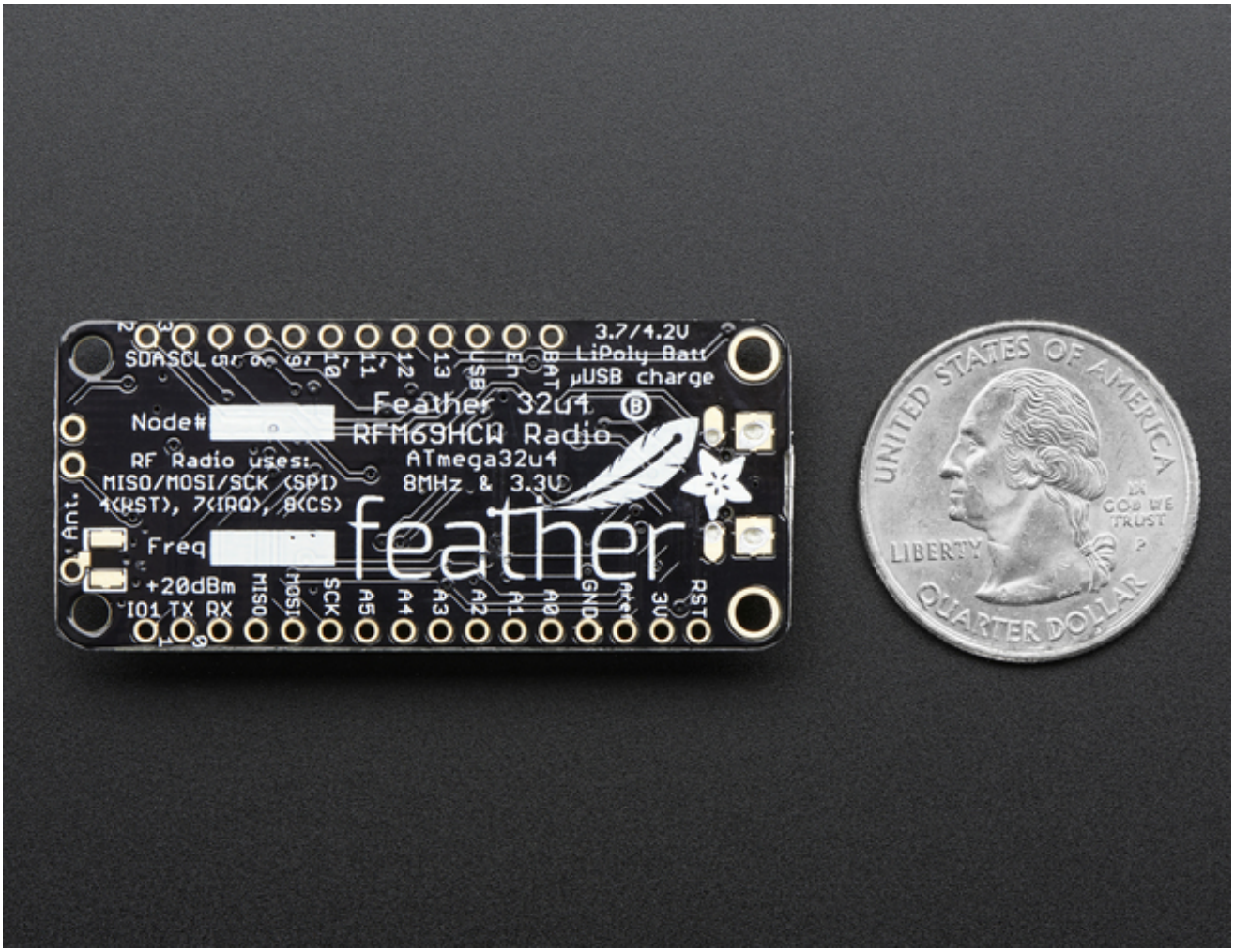
Guide Contents

Overview

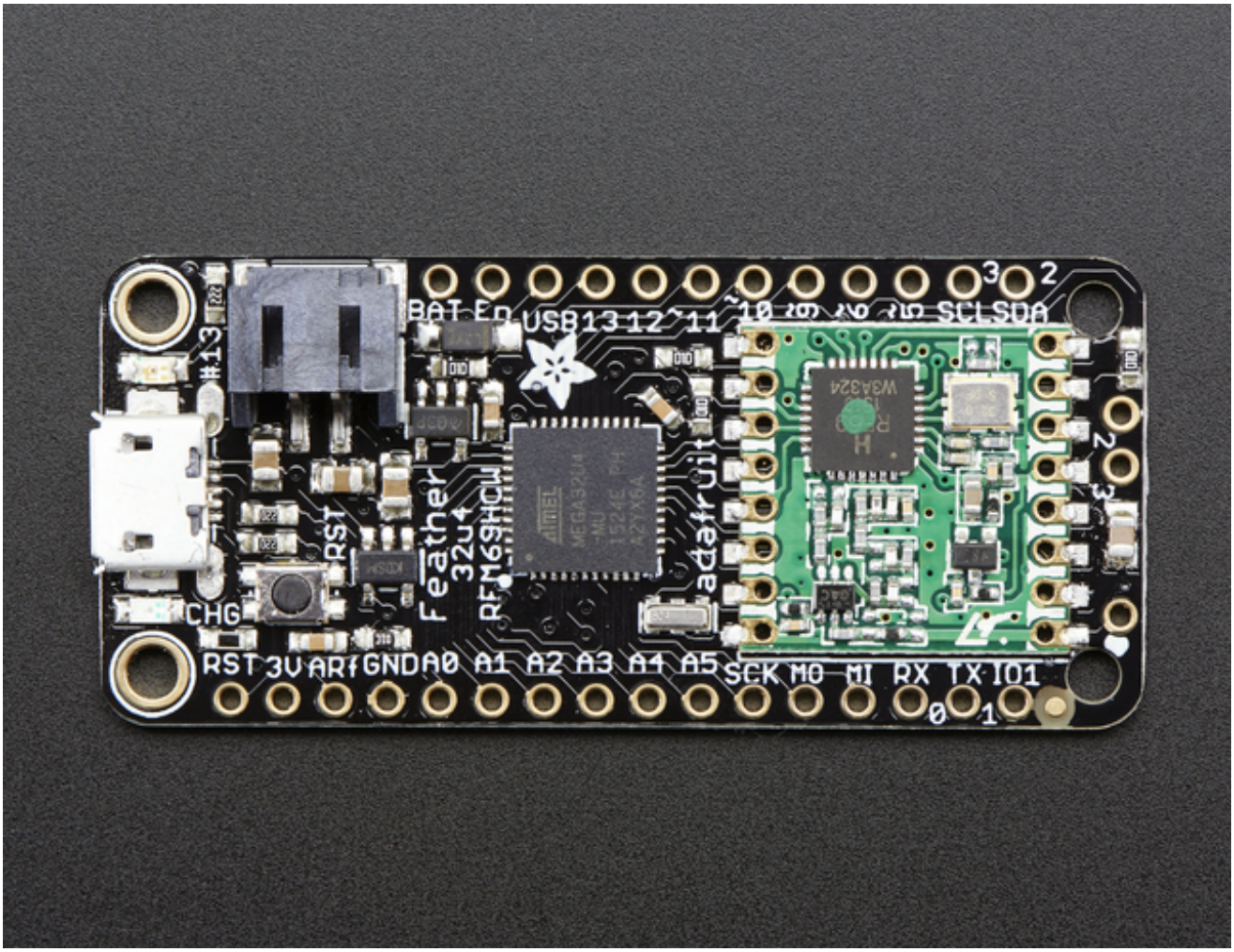


Feather is the new development board from Adafruit, and like its namesake it is thin, light, and lets you fly! We designed Feather to be a new standard for portable microcontroller cores.

This is the **Adafruit Feather 32u4 Radio (RFM69HCW)** - our take on an microcontroller packet radio transceiver with built in USB and battery charging. Its an Adafruit Feather 32u4 with a 433 or 868/915 MHz radio module cooked in! Great for making wireless networks that can go further than 2.4GHz 802.15.4 and similar, are more flexible than Bluetooth LE and without the high power requirements of WiFi. We have other boards in the Feather family, check'em out here (http://adafru.it/l7B)



At the Feather 32u4's heart is at ATmega32u4 clocked at 8 MHz and at 3.3V logic, a chip setup we've had tons of experience with as [it's the same as the Flora](http://adafru.it/dVl). This chip has 32K of flash and 2K of RAM, with built in USB so not only does it have a USB-to-Serial program & debug capability built in with no need for an FTDI-like chip, it can also act like a mouse, keyboard, USB MIDI device, etc.

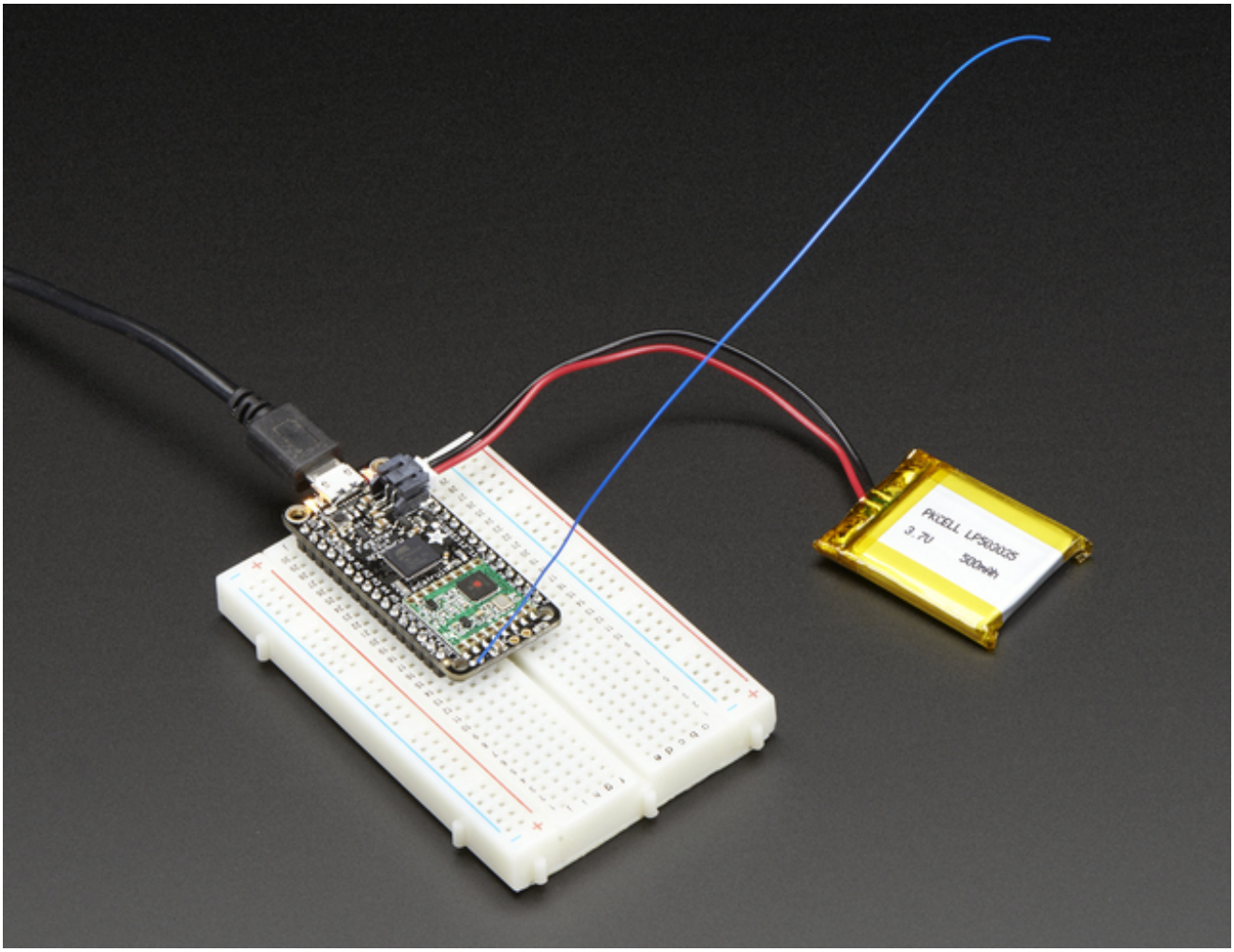


To make it easy to use for portable projects, we added a connector for any of our 3.7V Lithium polymer batteries and built in battery charging. You don't need a battery, it will run just fine straight from the micro USB connector. But, if you do have a battery, you can take it on the go, then plug in the USB to recharge. The Feather will automatically switch over to USB power when its available. We also tied the battery thru a divider to an analog pin, so you can measure and monitor the battery voltage to detect when you need a recharge.

Here's some handy specs! Like all Feather 32u4's you get:

- Measures 2.0" x 0.9" x 0.28" (51mm x 23mm x 8mm) without headers soldered in
- Light as a (large?) feather - 5.5 grams
- ATmega32u4 @ 8MHz with 3.3V logic/power
- 3.3V regulator with 500mA peak current output
- USB native support, comes with USB bootloader and serial port debugging
- You also get tons of pins - 20 GPIO pins
- Hardware Serial, hardware I2C, hardware SPI support
- 8 x PWM pins
- 10 x analog inputs

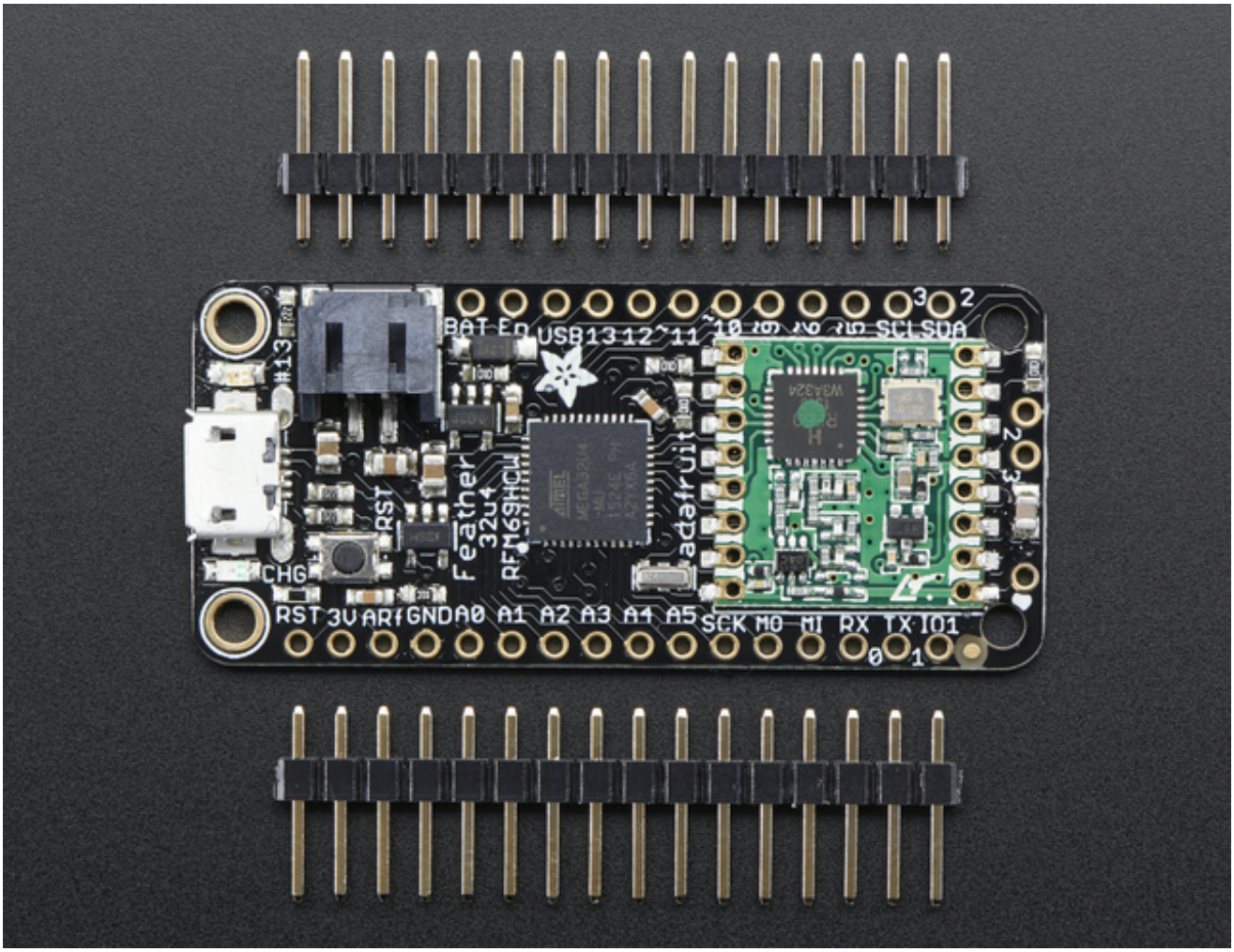
- Built in 100mA lipoly charger with charging status indicator LED
- Pin #13 red LED for general purpose blinking
- Power/enable pin
- 4 mounting holes
- Reset button



The **Feather 32u4 Radio** uses the extra space left over to add an RFM69HCW 433 or 868/915 MHz radio module. These radios are not good for transmitting audio or video, but they do work quite well for small data packet transmission when you ned more range than 2.4 GHz (BT, BLE, WiFi, ZigBee)

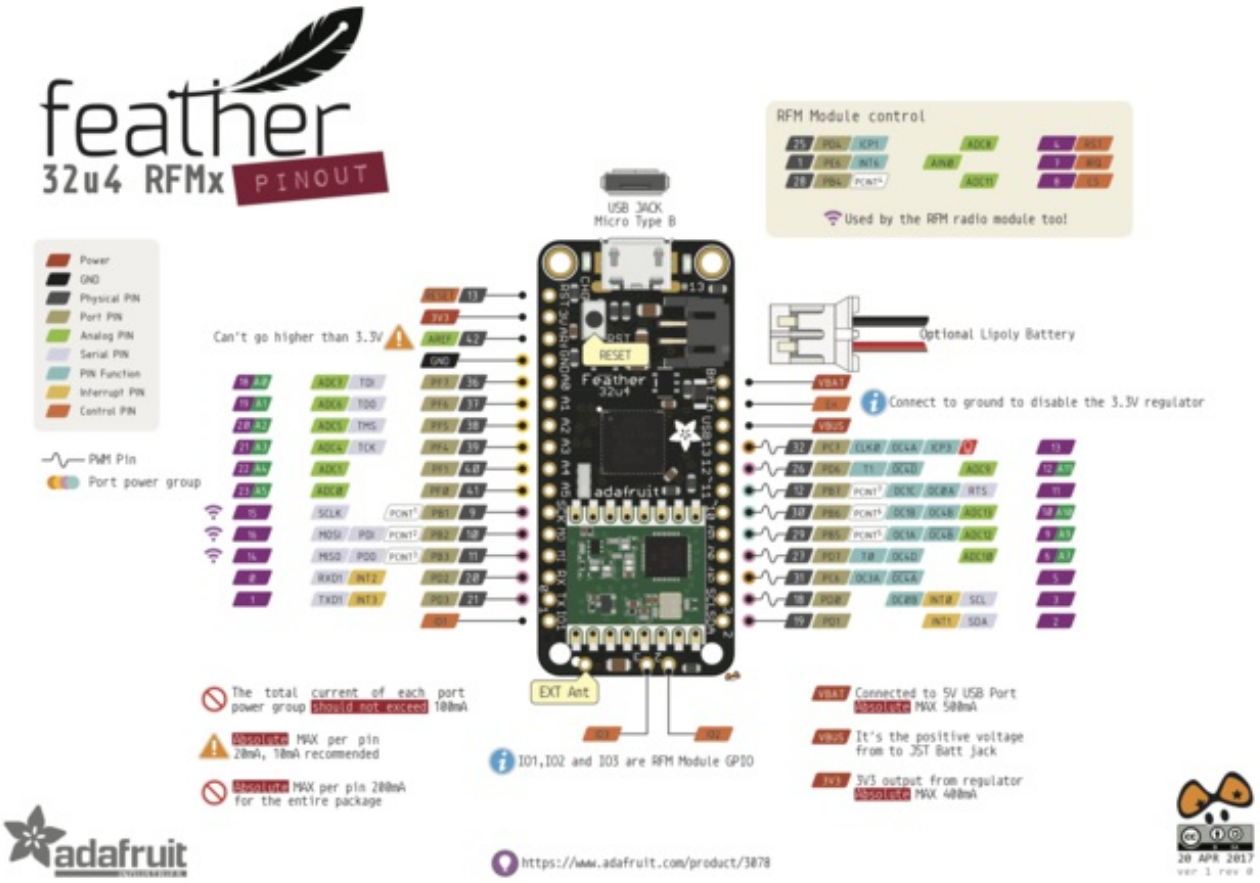
- SX1231 based module with SPI interface
- Packet radio with ready-to-go Arduino libraries
- Uses the amateur or license-free ISM bands (http://adafru.it/mOE): 433MHz is ITU "Europe" license-free ISM or ITU "American" amateur with limitations. 900MHz is license-free ISM for ITU "Americas"
- +13 to +20 dBm up to 100 mW Power Output Capability (power output selectable in software)

- Create multipoint networks with individual node addresses
- Encrypted packet engine with AES-128
- Simple wire antenna or spot for uFL connector



Comes fully assembled and tested, with a USB bootloader that lets you quickly use it with the Arduino IDE. We also toss in some header so you can solder it in and plug into a solderless breadboard. You will need to cut and solder on a small piece of wire (any solid or stranded core is fine) in order to create your antenna. **Lipoly battery and USB cable not included** but we do have lots of options in the shop if you'd like!

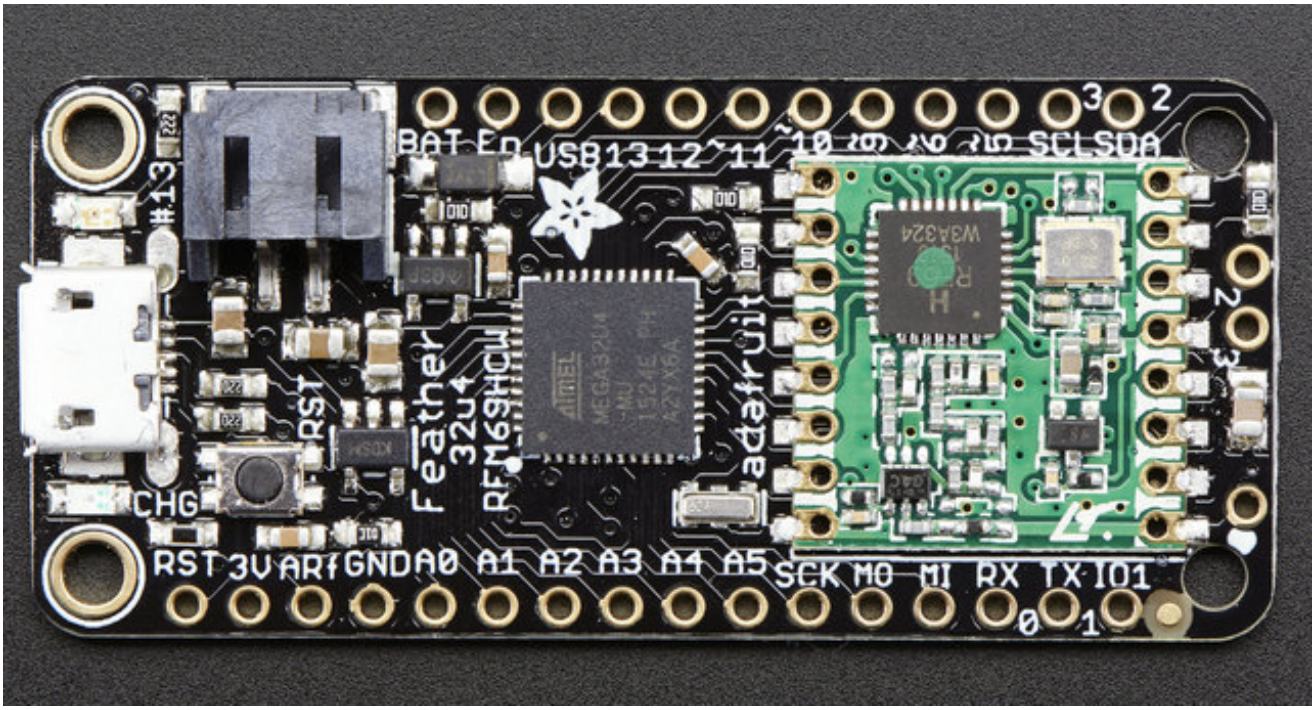
Pinouts



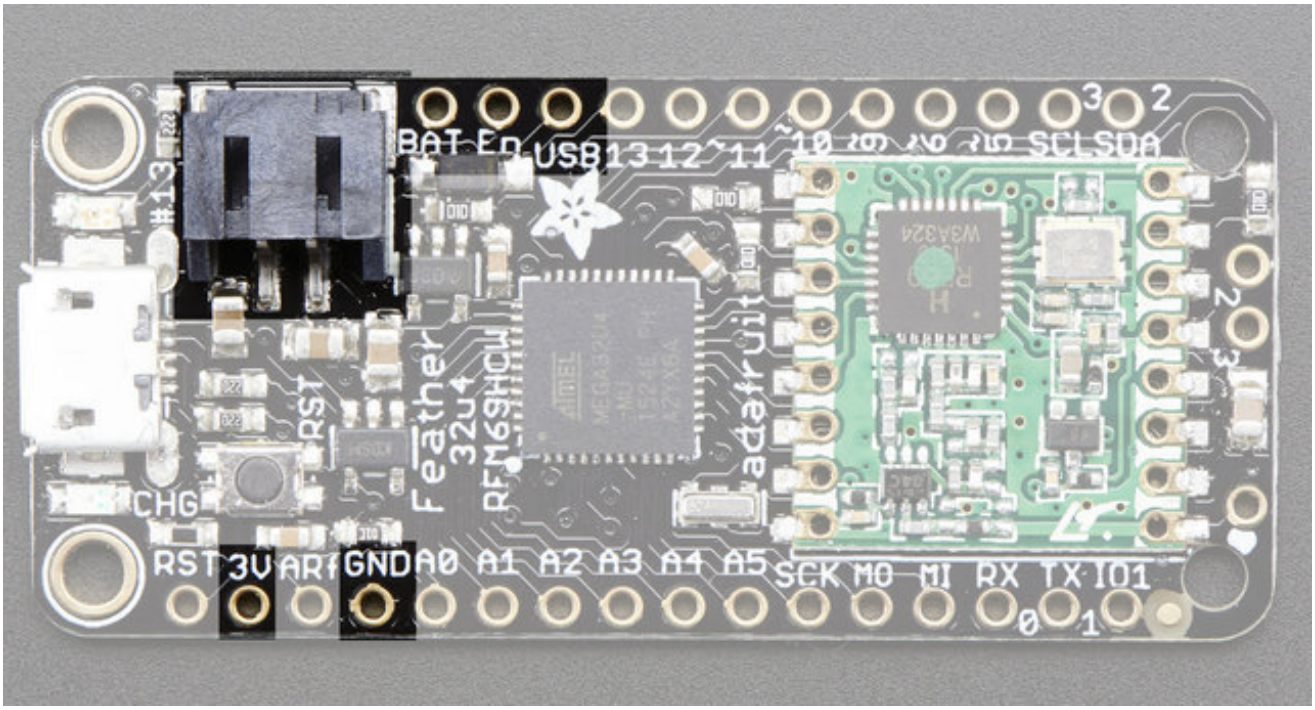
The Feather 32u4 Radio is chock-full of microcontroller goodness. There's also a lot of pins and ports. We'll take you a tour of them now!

Note that the pinouts are identical for both the Feather 32u4 RFM69 and LoRa radios - you can look at the silkscreen of the Feather to see it says "RFM69" or "LoRa"

Pinouts are also the same for both 433MHz and 900Mhz. You can tell the difference by looking for a colored dot on the chip or crystal of the radio, green/blue is 900MHz & red is 433MHz



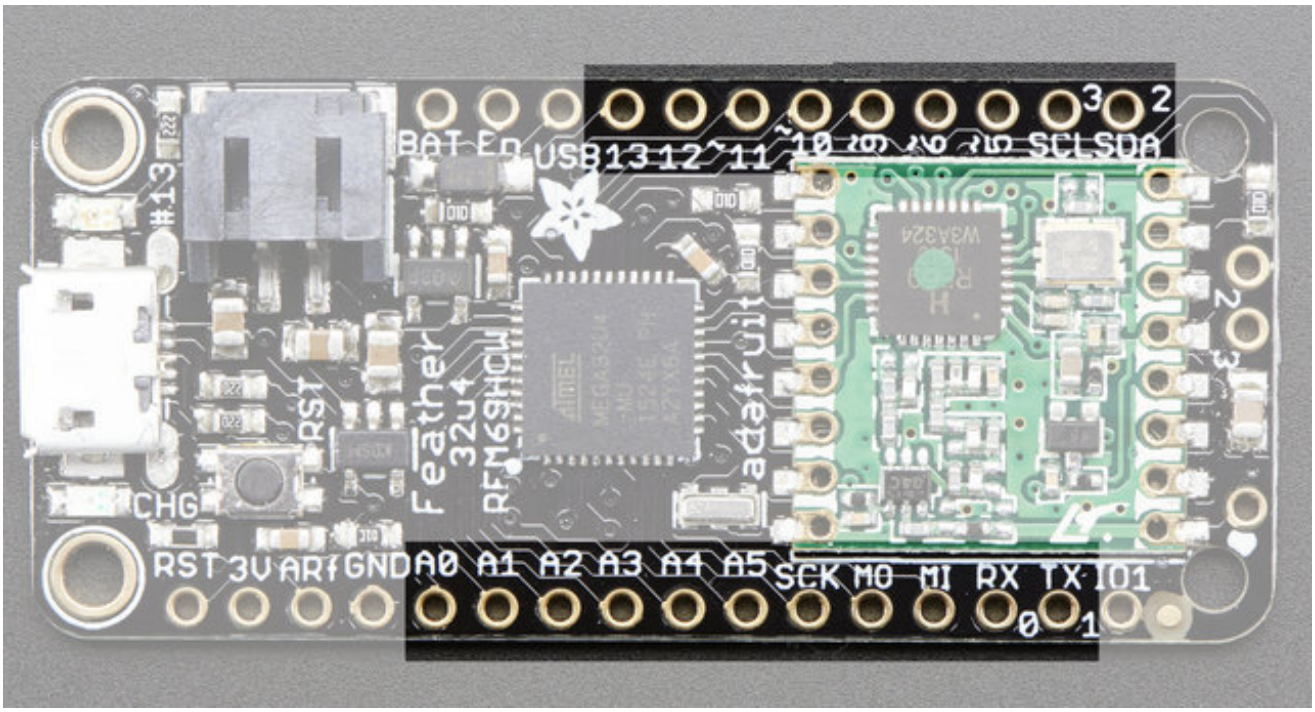
Power Pins



- **GND** - this is the common ground for all power and logic
- **BAT** - this is the positive voltage to/from the JST jack for the optional Lipoly battery
- **USB** - this is the positive voltage to/from the micro USB jack if connected
- **EN** - this is the 3.3V regulator's enable pin. It's pulled up, so connect to ground to

disable the 3.3V regulator
- **3V** - this is the output from the 3.3V regulator, it can supply 500mA peak

Logic pins

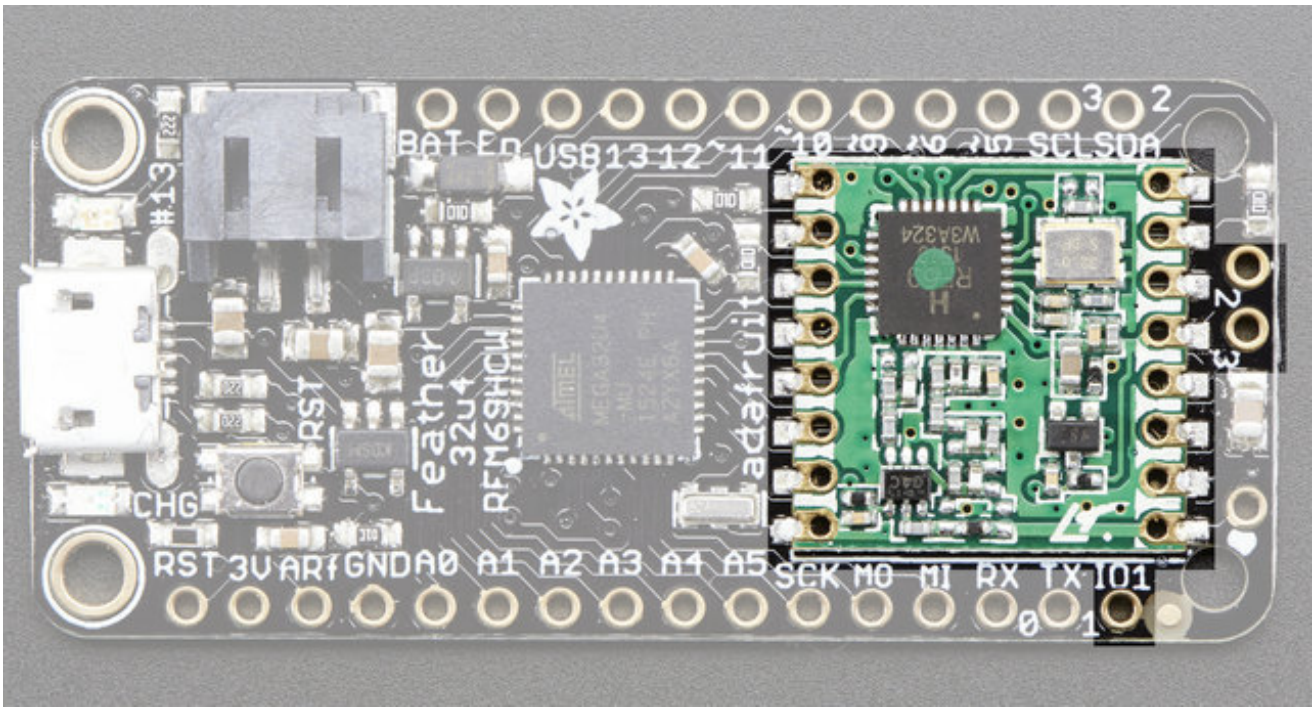


This is the general purpose I/O pin set for the microcontroller. All logic is 3.3V

- **#0** / **RX** - GPIO #0, also receive (input) pin for **Serial1** and Interrupt #2
- **#1** / **TX** - GPIO #1, also transmit (output) pin for **Serial1** and Interrupt #3
- **#2** / **SDA** - GPIO #2, also the I2C (Wire) data pin. There's no pull up on this pin by default so when using with I2C, you may need a 2.2K-10K pullup. Also Interrupt #1
- **#3** / **SCL** - GPIO #3, also the I2C (Wire) clock pin. There's no pull up on this pin by default so when using with I2C, you may need a 2.2K-10K pullup. Can also do PWM output and act as Interrupt #0.
- **#5** - GPIO #5, can also do PWM output
- **#6** - GPIO #6, can also do PWM output and analog input **A7**
- **#9** - GPIO #9, also analog input **A9** and can do PWM output. This analog input is connected to a voltage divider for the lipoly battery so be aware that this pin naturally 'sits' at around 2VDC due to the resistor divider
- **#10** - GPIO #10, also analog input **A10** and can do PWM output.
- **#11** - GPIO #11, can do PWM output.
- **#12** - GPIO #12, also analog input **A11** and can do PWM output.
- **#13** - GPIO #13, can do PWM output and is connected to the **red LED** next to the USB jack

- **A0 thru A5** - These are each analog input as well as digital I/O pins.
- **SCK/MOSI/MISO** - These are the hardware SPI pins,**used by the RFM radio module too!** You can use them as everyday GPIO pins if you don't activate the radio and keep the RFM CS pin high. However, we really recommend keeping them free as they should be kept available for the radio module. If they are used, make sure its with a device that will kindly share the SPI bus! Also used to reprogram the chip with an AVR programmer if you need.

RFM/SemTech Radio Module



Since not all pins can be brought out to breakouts, due to the small size of the Feather, we use these to control the radio module

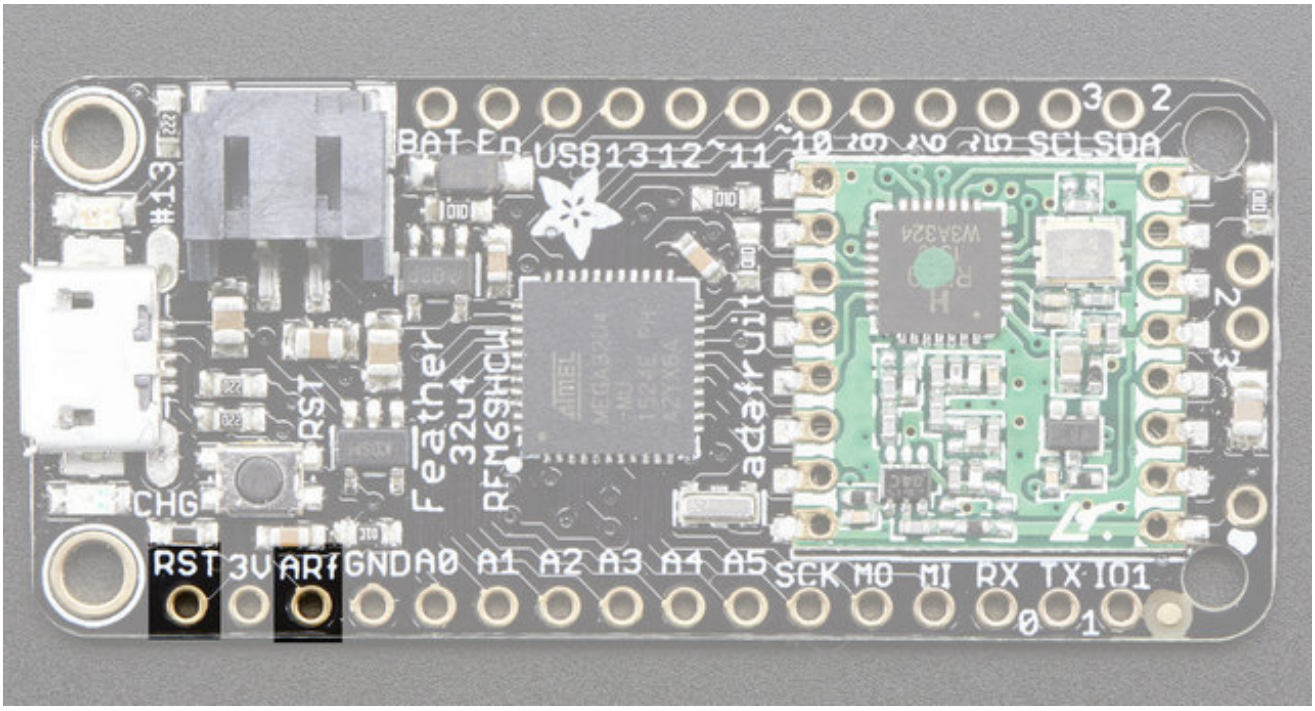
- **#8** - used as the radio **CS** (chip select) pin
- **#7** - used as the radio **GPIO0** / **IRQ** (interrupt request) pin.
- **#4** - used as the radio **Reset** pin

Since these are not brought out there should be no risk of using them by accident!

There are also breakouts for 3 of the RFM's GPIO pins (**IO1, IO2** and **IO3**). You probably wont need these for most uses of the Feather but they are available in case you need 'em!

Other Pins!

- **RST** - this is the Reset pin, tie to ground to manually reset the AVR, as well as launch the bootloader manually
- **ARef** - the analog reference pin. Normally the reference voltage is the same as the chip logic voltage (3.3V) but if you need an alternative analog reference, connect it to this pin and select the external AREF in your firmware. Can't go higher than 3.3V!

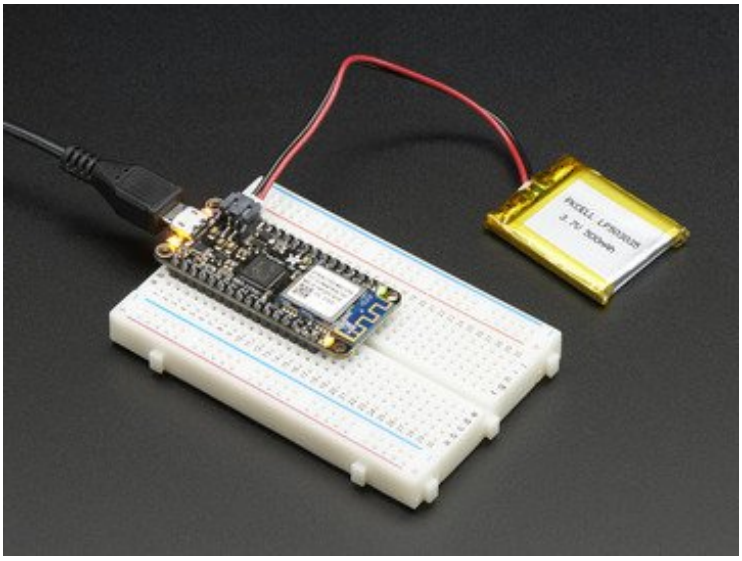
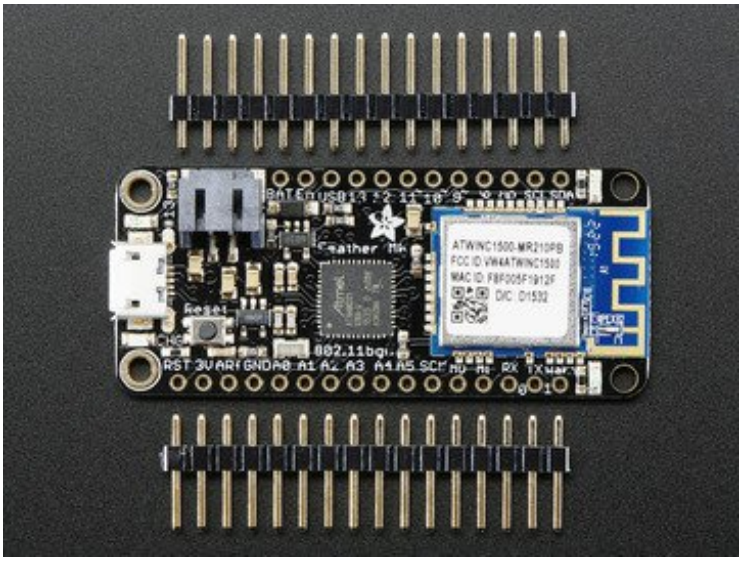


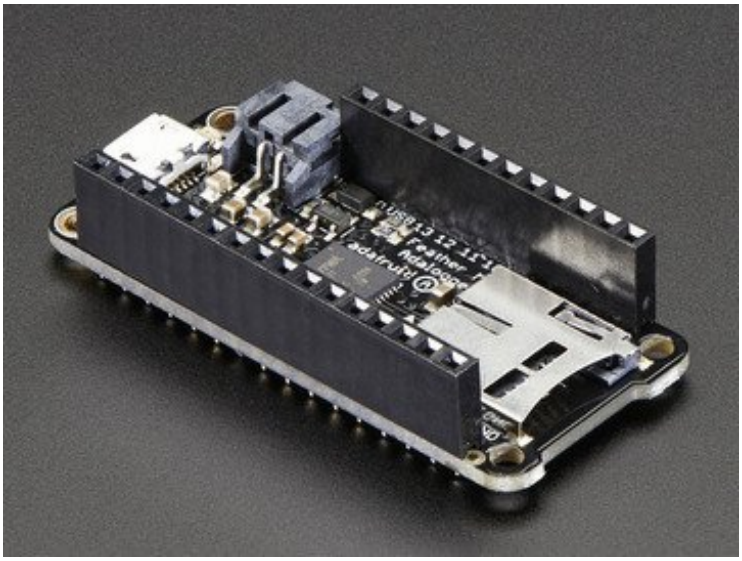
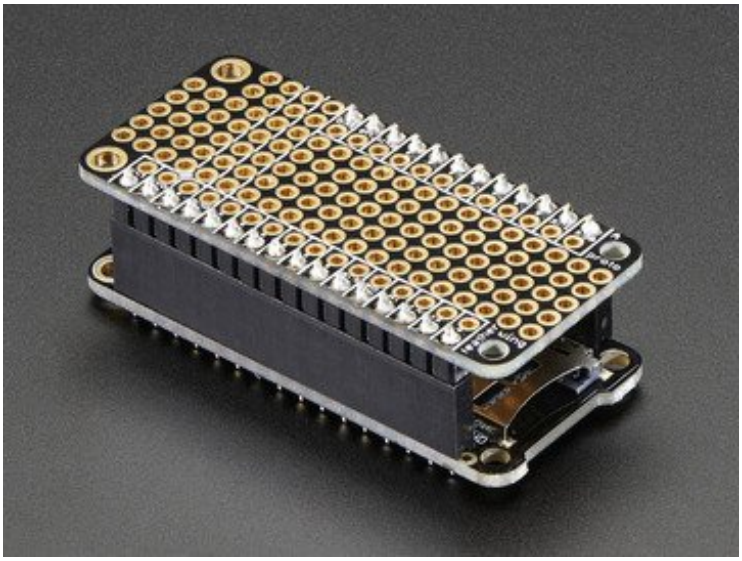
Assembly

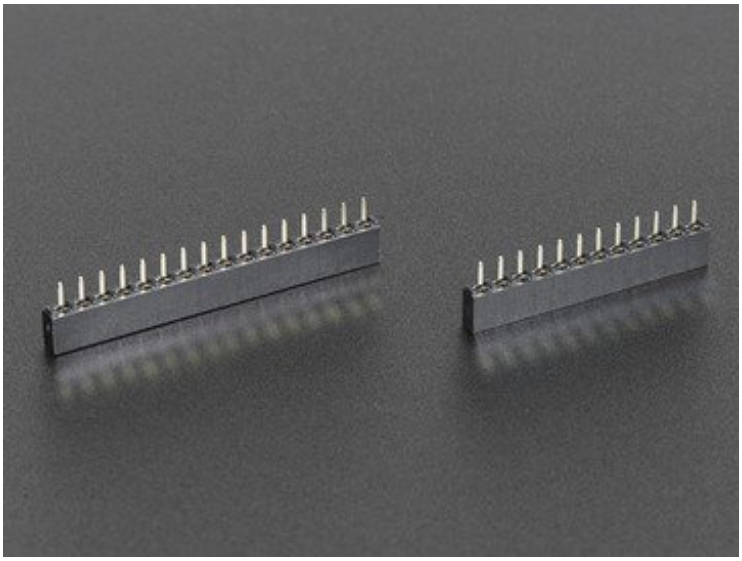
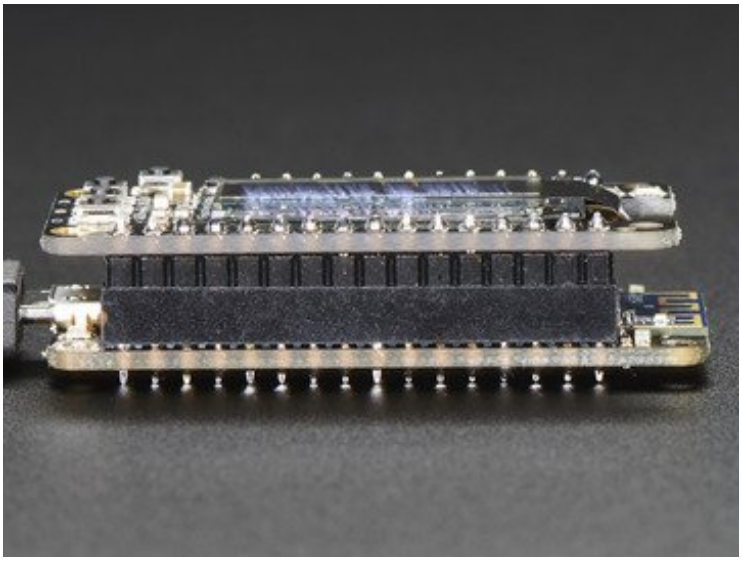
We ship Feathers fully tested but without headers attached - this gives you the most flexibility on choosing how to use and configure your Feather

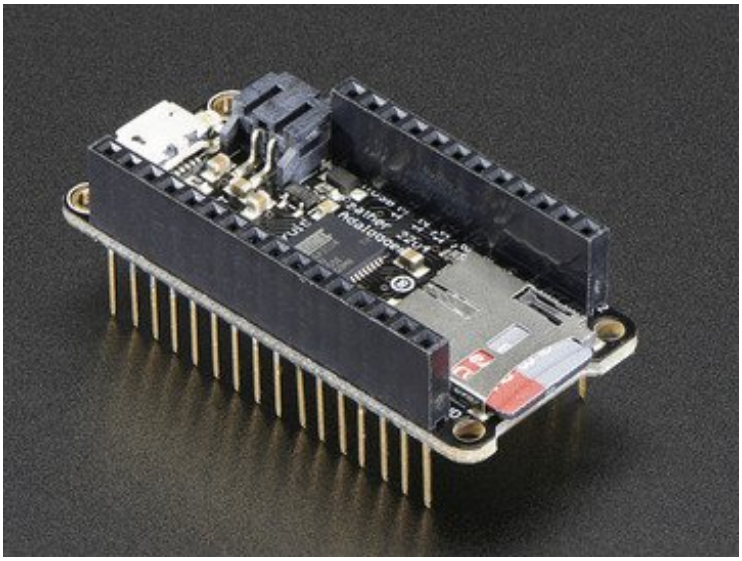
Header Options!

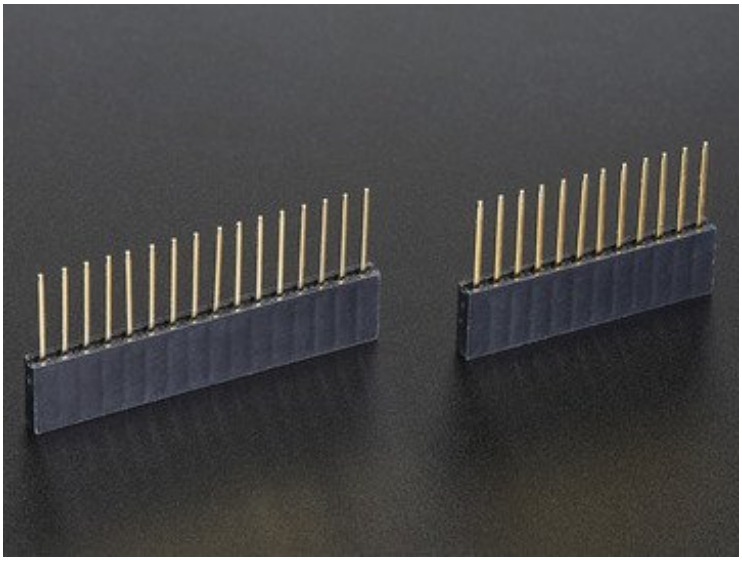
Before you go gung-ho on soldering, there's a few options to consider!

- 
The first option is soldering in plain male headers, this lets you plug in the Feather into a solderless breadboard
- 

- 
  Another option is to go with socket female headers. This won't let you plug the Feather into a breadboard but it will let you attach featherwings very easily
- 

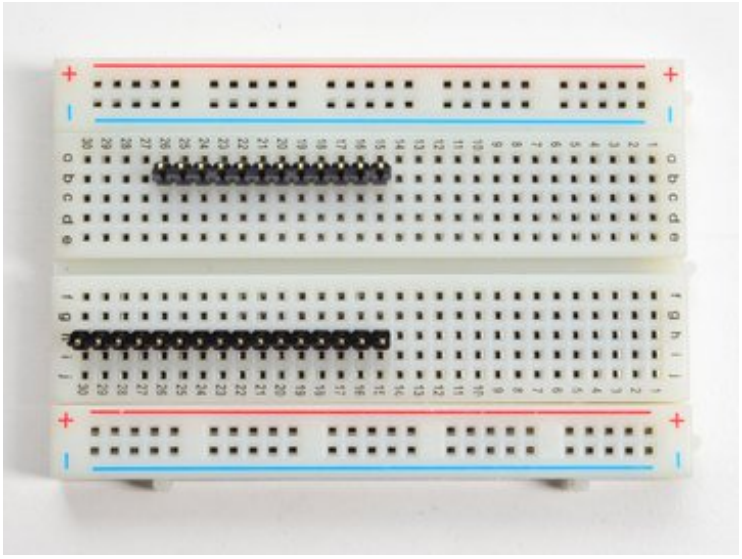
- 
  We also have 'slim' versions of the female headers, that are a little shorter and give a more compact shape
- 

- 

  Finally, there's the "Stacking Header" option. This one is sort of the best-of-both-worlds. You get the ability to plug into a solderless breadboard *and* plug a featherwing on top. But its a little bulky
- 

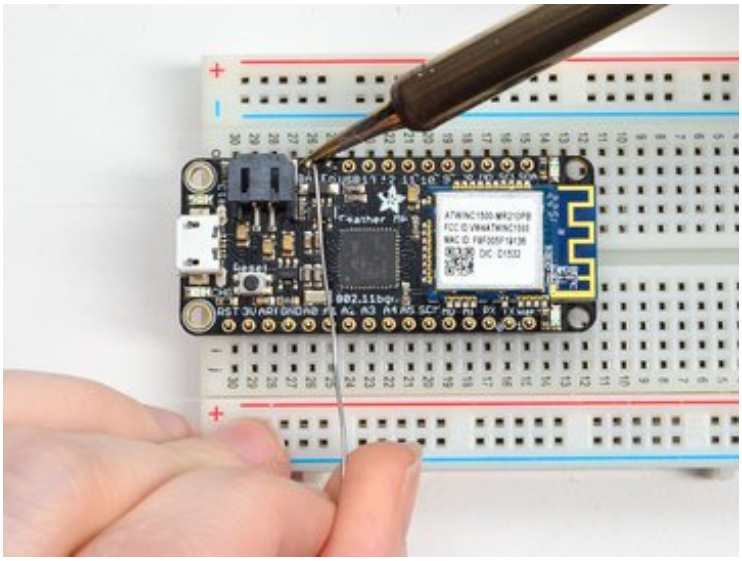
Soldering in Plain Headers

Prepare the header

- 

strip:

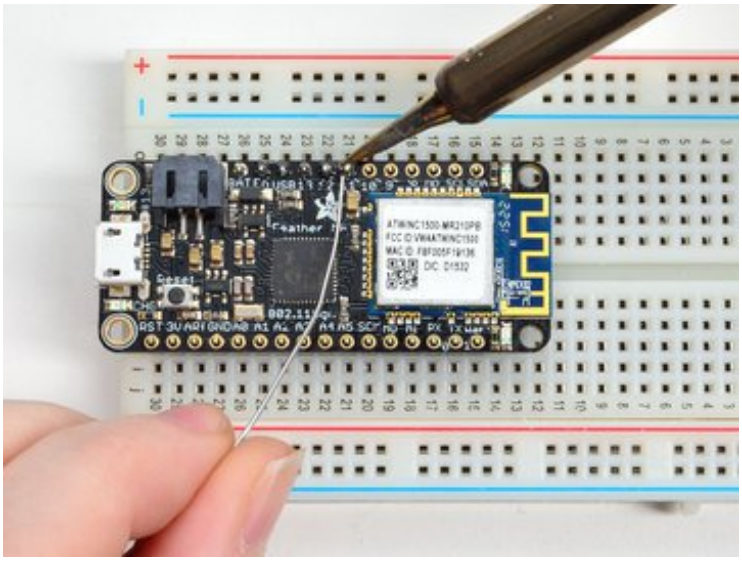
Cut the strip to length if necessary. It will be easier to solder if you insert it into a breadboard - **long pins down**

- 

Add the breakout board:

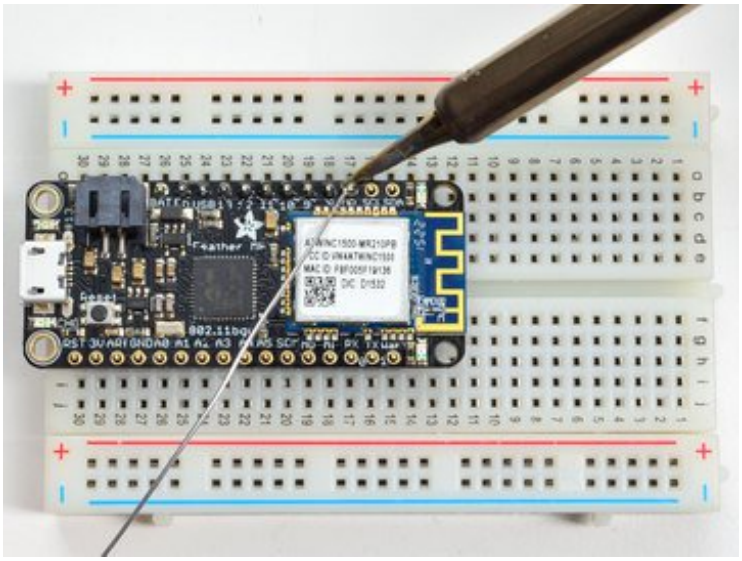
Place the breakout board over the pins so that the short pins poke through the breakout pads

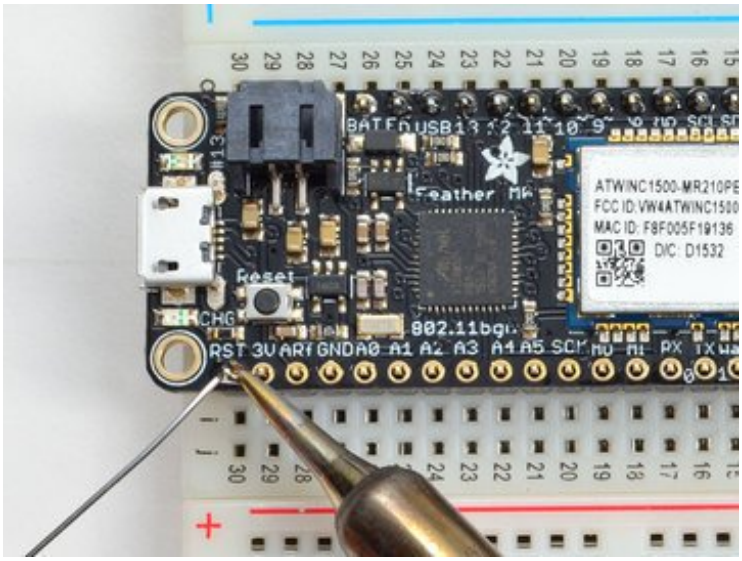
And Solder!

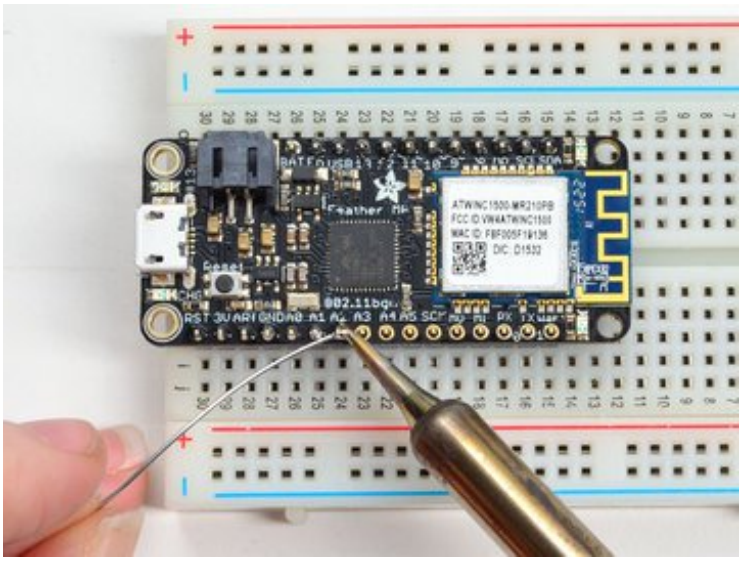
- 

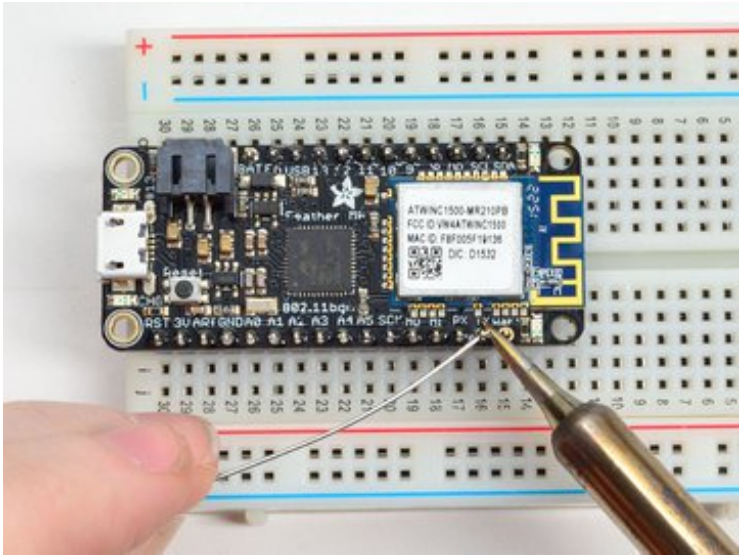
Be sure to solder all pins for reliable electrical contact.

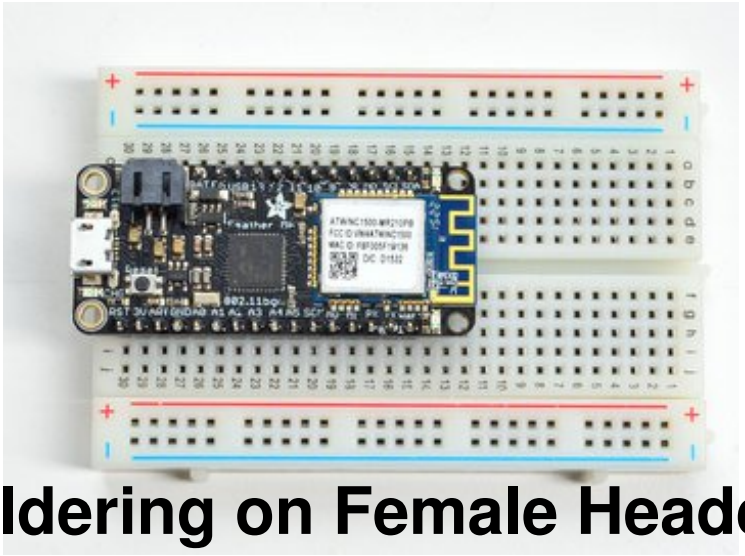
(For tips on soldering, be sure to check out our [Guide to Excellent Soldering](http://adafru.it/aTk)).

- 

- 

Solder the other strip as well.
- 

- 



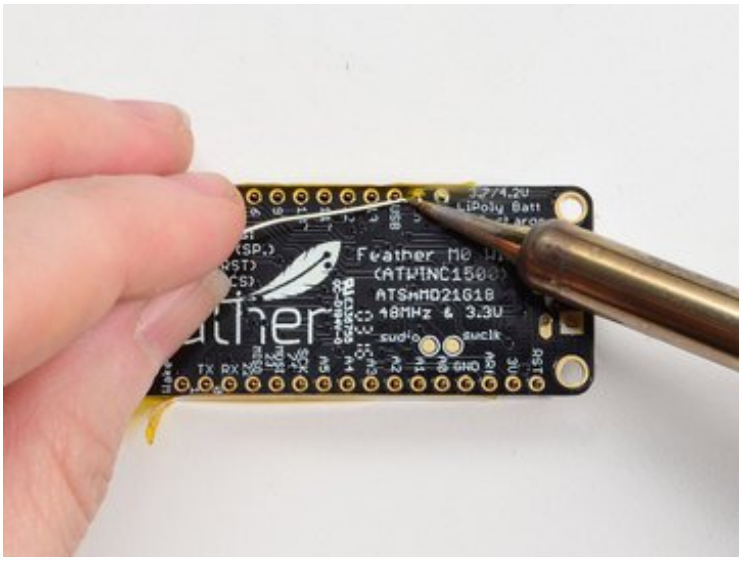
You're done! Check your solder joints visually and continue onto the next steps

Soldering on Female Header

- 

Tape In Place

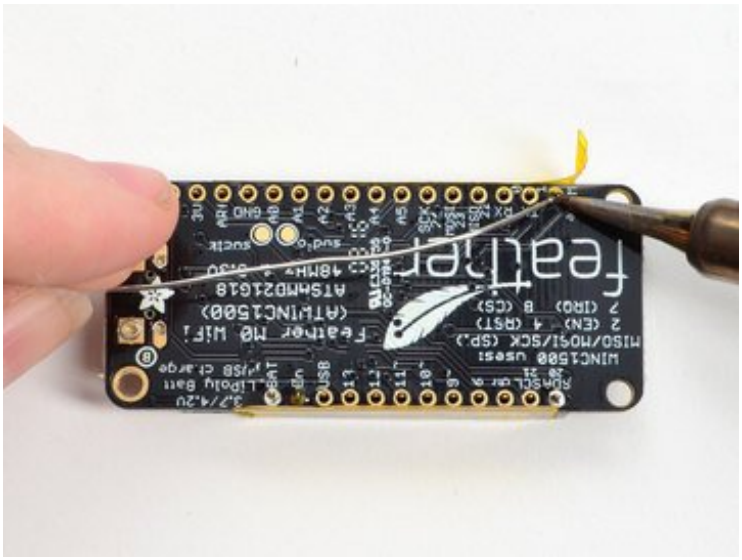
For sockets you'll want to tape them in place so when you flip over the board they don't fall out

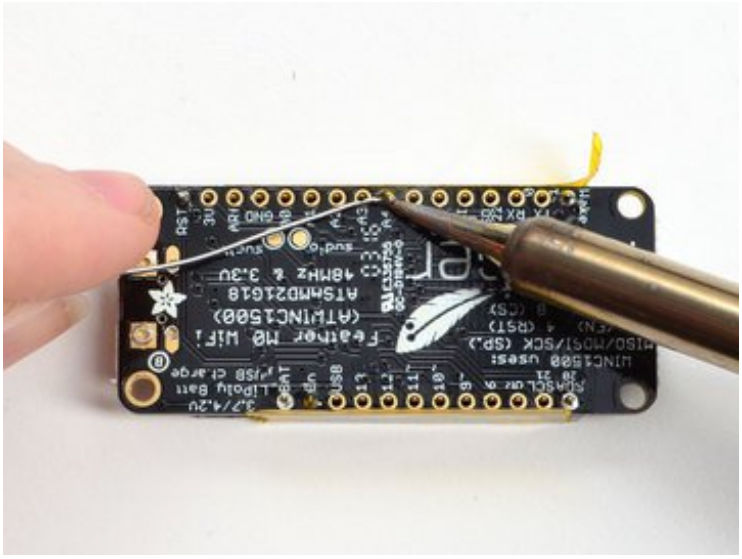
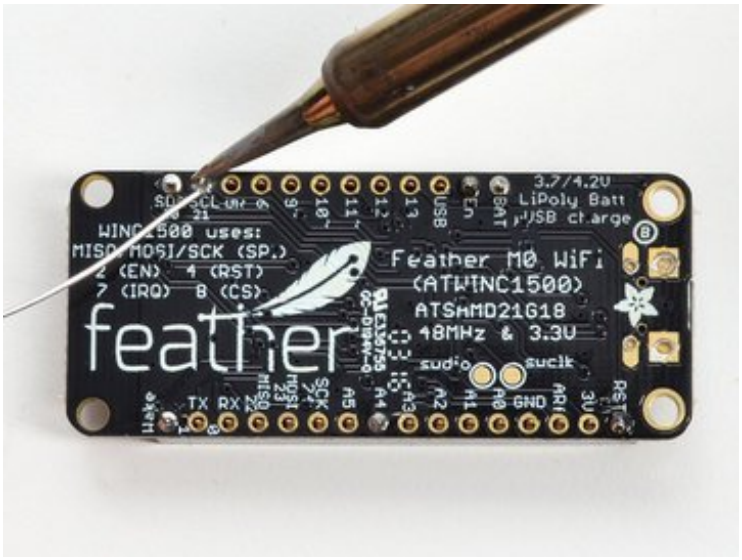
- 

Flip & Tack Solder

After flipping over, solder one or two points on each strip, to 'tack' the header in place

- 

- 

- 
- 

And Solder!

Be sure to solder all pins for reliable electrical contact.

(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

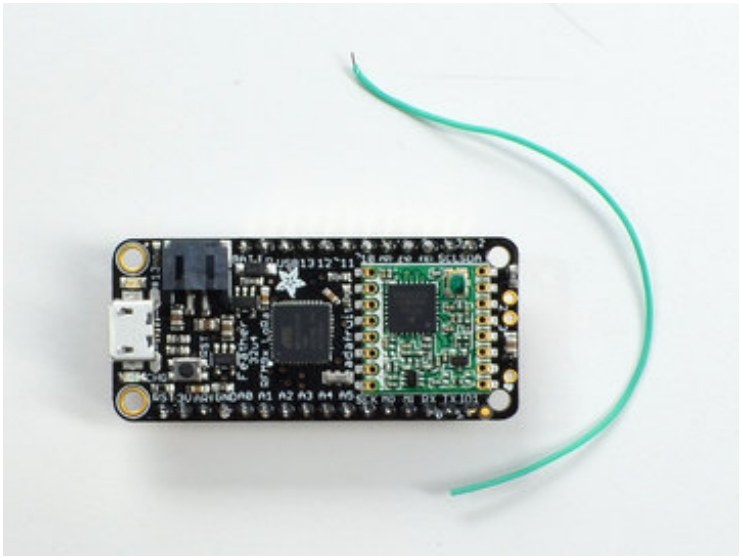
- 

Antenna Options

Your Feather Radio does not have a built-in antenna. Instead, you have two options for attaching an antenna. For most low cost radio nodes, a wire works great. If you need to put the Feather into an enclosure, soldering in uFL and using a uFL to SMA adapter will let you attach an external antenna

Wire Antenna

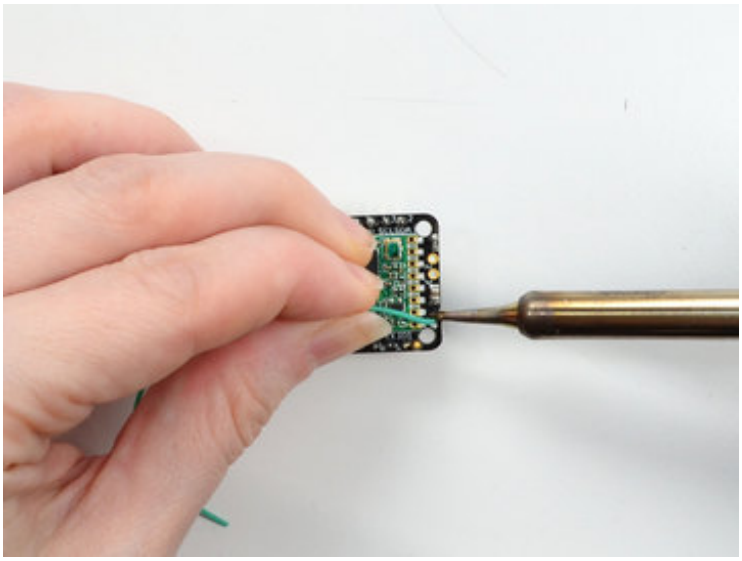
A wire antenna, aka "quarter wave whip antenna" is low cost and works very well! You just have to cut the wire down to the right length.

- 

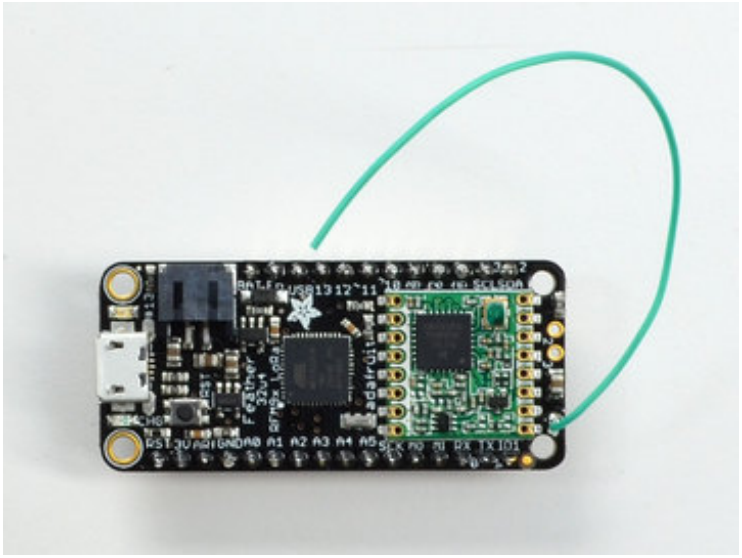
Cut a stranded or solid core wire the the proper length for the module/frequency

- **433 MHz** - 6.5 inches, or 16.5 cm
- **868 MHz** - 3.25 inches or 8.2 cm
- **915 MHz** - 3 inches or 7.8 cm

Strip a mm or two off the end of

- 

  the wire, tin and solder into the **ANT** pad on the very right hand edge of the Feather

- 

  That's pretty much it, you're done!

uFL Antenna

If you want an external antenna, you need to do a tiny bit more work but its not too difficult.

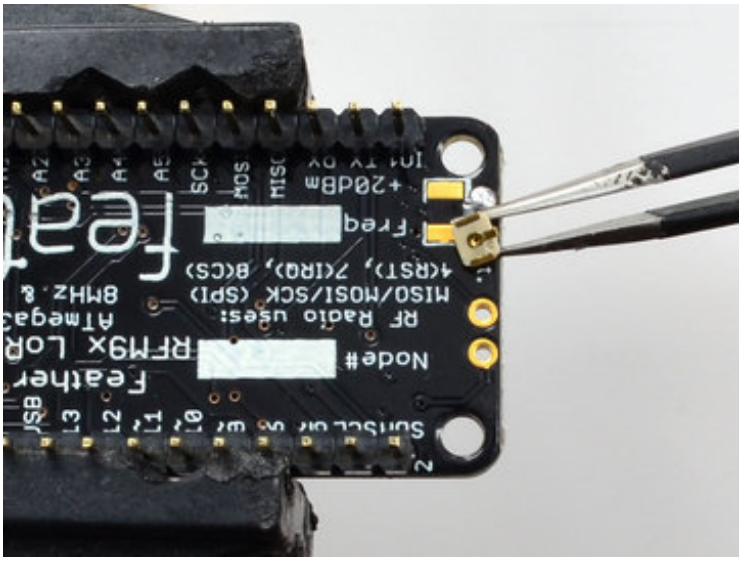
You'll need to get an SMT uFL connector, these are fairly standard (http://adafru.it/1661)

You'll also need a uFL to SMA adapter (http://adafru.it/851) (or whatever adapter you need for the antenna you'll be using, SMA is the most common

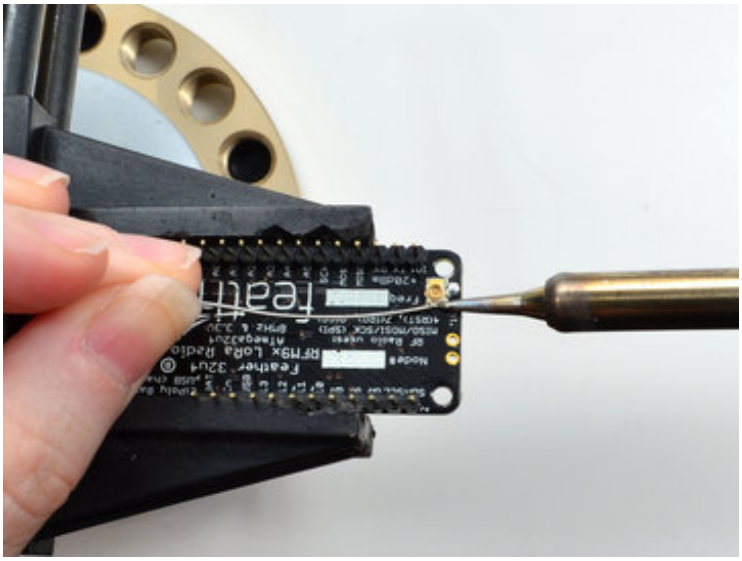
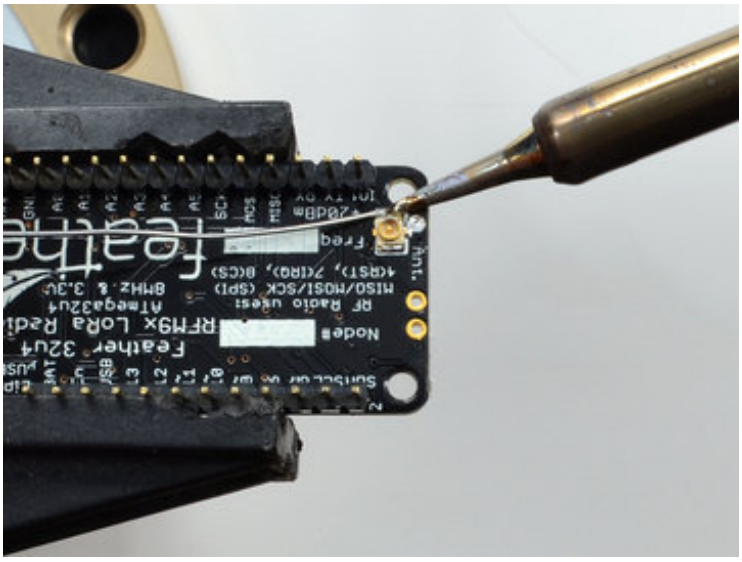
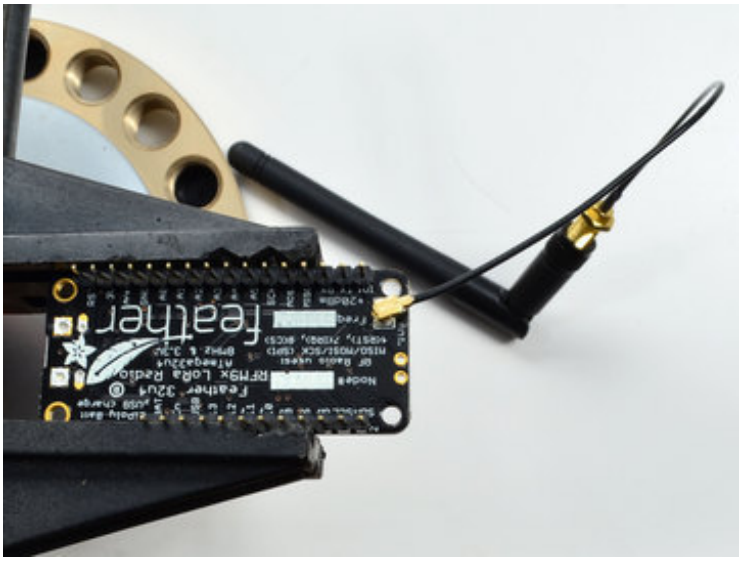
Of course, you will also need an antenna of some sort, that matches your radio frequency

uFL connectors are rated for 30 connection cycles, but be careful when

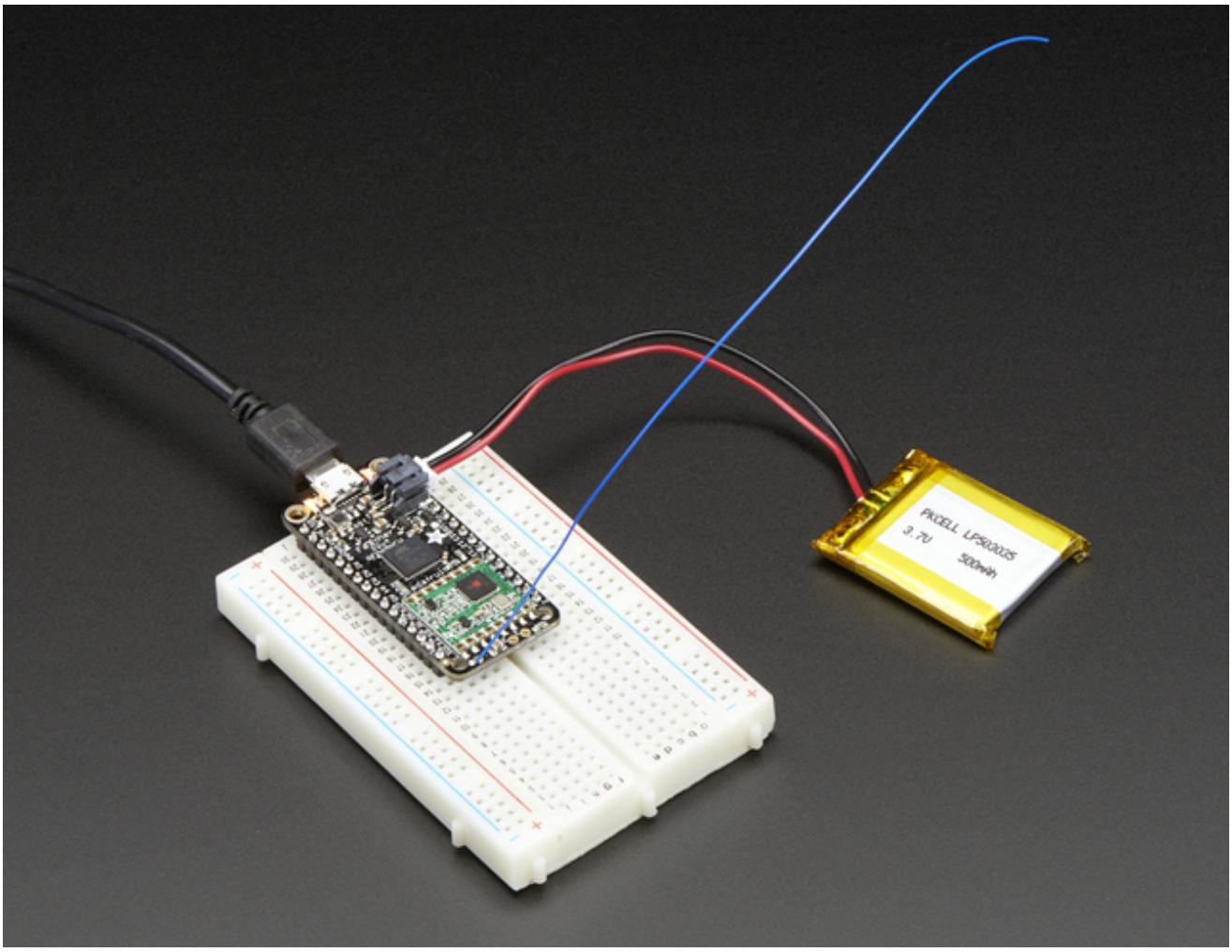
connecting/disconnecting to not rip the pads off the PCB. Once a uFL/SMA adapter is connected, use strain relief!

- 

Check the bottom of the uFL connector, note that there's two large side pads (ground) and a little inlet pad. The other small pad is not used!

- 
  Solder all three pads to the bottom of the Feather
- 
- 
  Once done attach your uFL adapter and antenna!

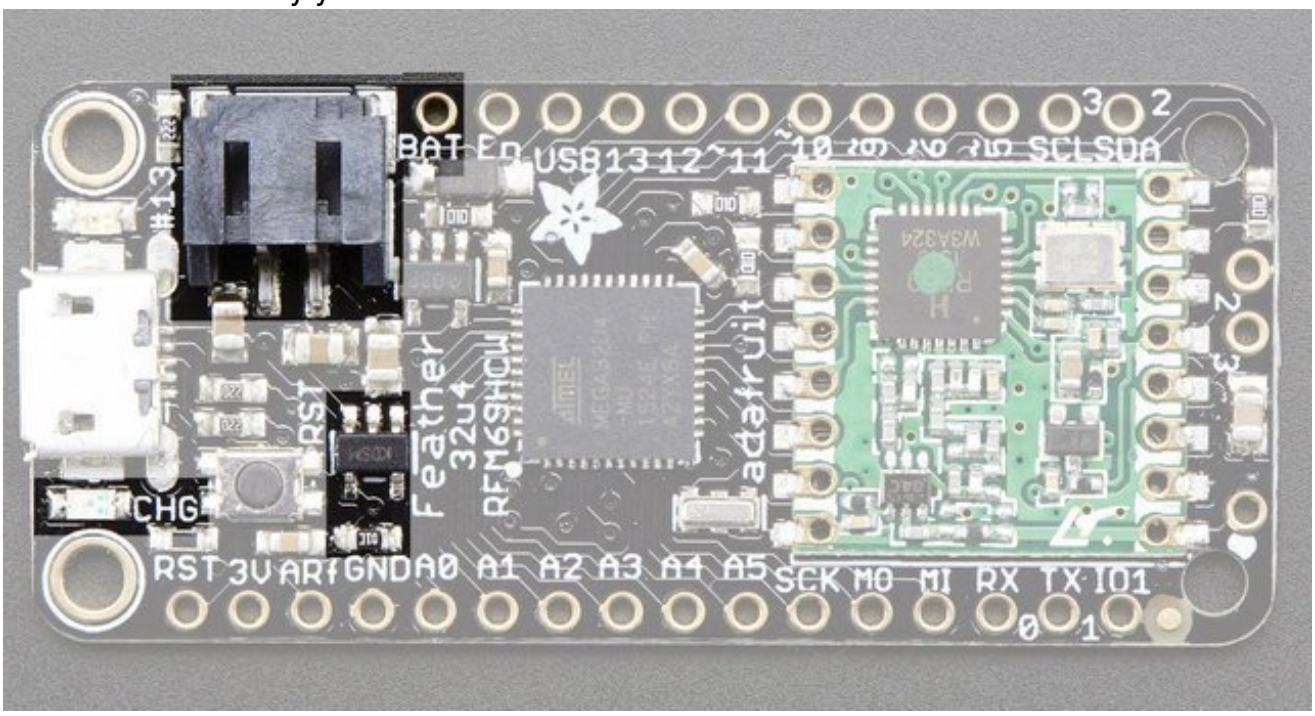
Power Management



Battery + USB Power

We wanted to make the Feather easy to power both when connected to a computer as well as via battery. There's **two ways to power** a Feather. You can connect with a MicroUSB cable (just plug into the jack) and the Feather will regulate the 5V USB down to 3.3V. You can also connect a 4.2/3.7V Lithium Polymer (Lipo/Lipoly) or Lithium Ion (LiIon) battery to the JST jack. This will let the Feather run on a rechargable battery. **When the USB power is powered, it will automatically switch over to USB for power, as well as start charging the battery (if attached) at 100mA.** This happens 'hotswap' style so you can always keep the Lipoly connected as a 'backup' power that will only get used when USB power is lost.

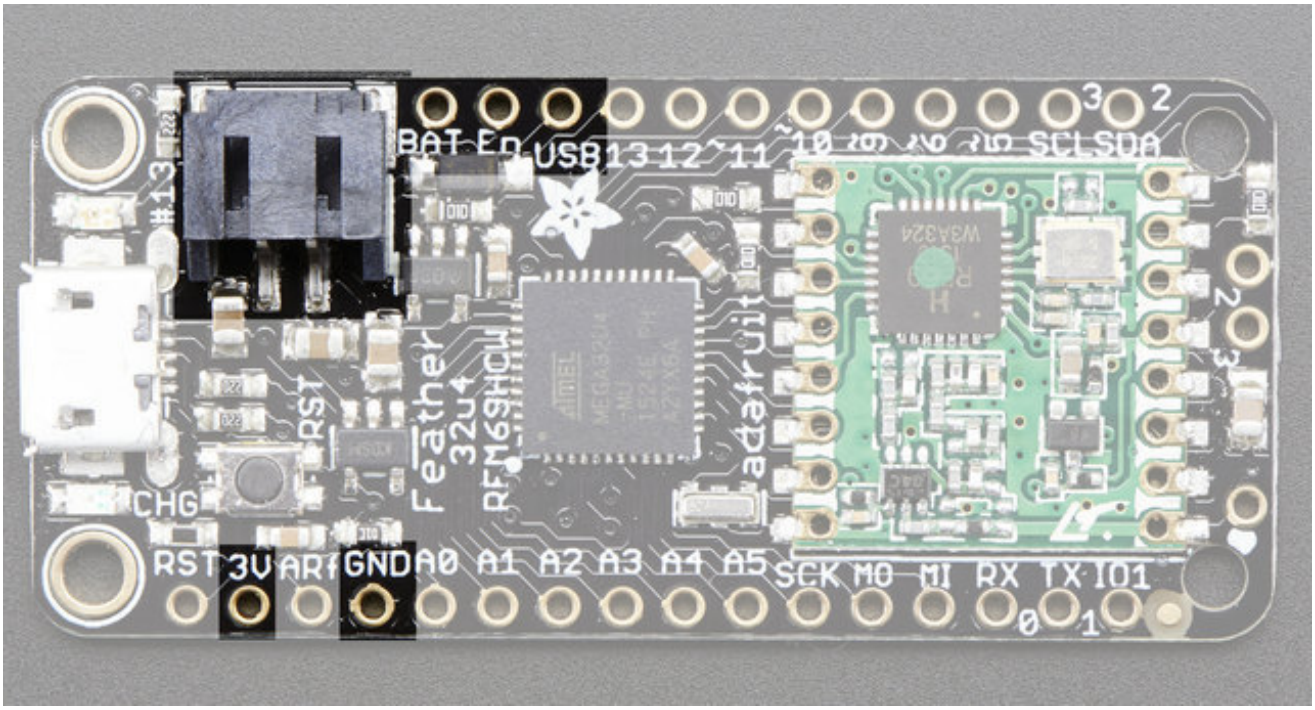
The JST connector polarity is matched to Adafruit LiPoly batteries. Using wrong polarity batteries can destroy your Feather



The above shows the Micro USB jack (left), Lipoly JST jack (top left), as well as the 3.3V regulator and changeover diode (just to the right of the JST jack) and the Lipoly charging circuitry (to the right of the Reset button). There's also a **CHG** LED, which will light up while the battery is charging. This LED might also flicker if the battery is not connected.

Power supplies

You have a lot of power supply options here! We bring out the**BAT** pin, which is tied to the lipoly JST connector, as well as **USB** which is the +5V from USB if connected. We also have the **3V** pin which has the output from the 3.3V regulator. We use a 500mA peak AP2112. While you can get 500mA from it, you can't do it continuously from 5V as it will overheat the regulator. It's fine for, say, powering an ESP8266 WiFi chip or XBee radio though, since the current draw is 'spiky' & sporadic.



Measuring Battery

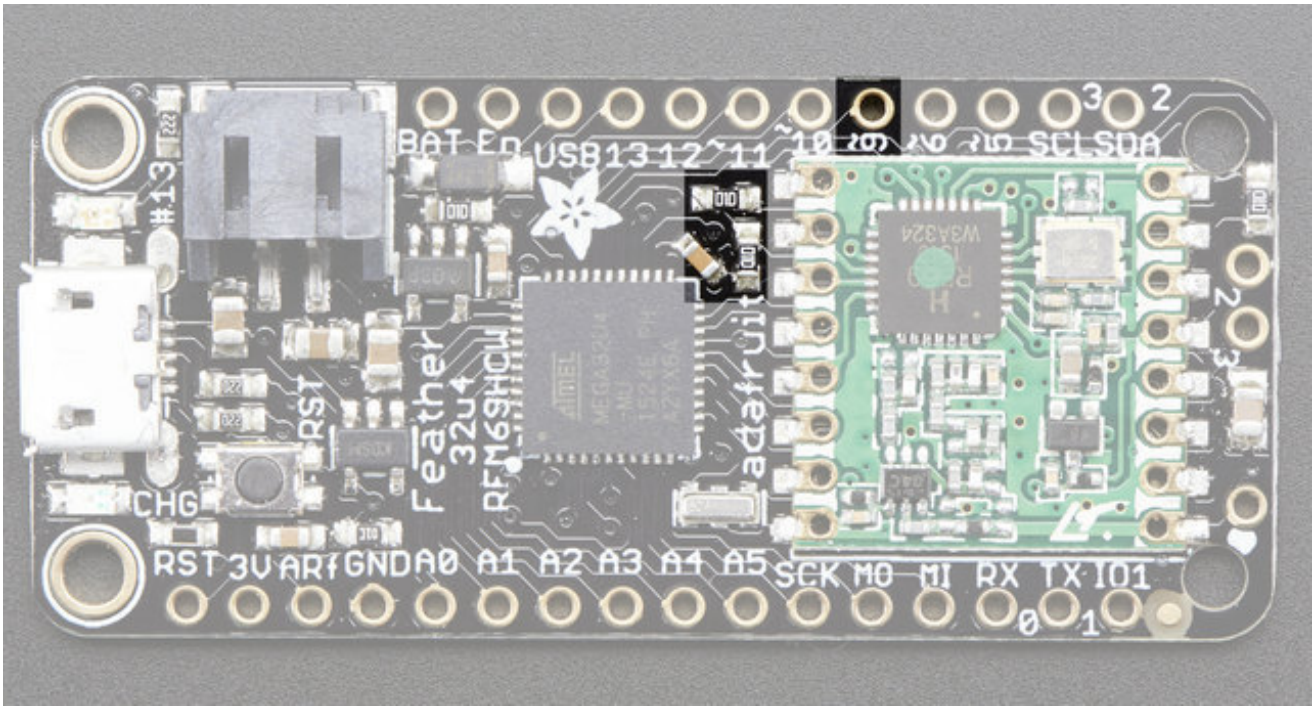
If you're running off of a battery, chances are you wanna know what the voltage is at! That way you can tell when the battery needs recharging. Lipoly batteries are 'maxed out' at 4.2V and stick around 3.7V for much of the battery life, then slowly sink down to 3.2V or so before the protection circuitry cuts it off. By measuring the voltage you can quickly tell when you're heading below 3.7V

To make this easy we stuck a double-100K resistor divider on the **BAT** pin, and connected it to **D9** (a.k.a analog #7 **A7**). You can read this pin's voltage, then double it, to get the battery voltage.

```
#define VBATPIN A9

float measuredvbat = analogRead(VBATPIN);
measuredvbat *= 2;    // we divided by 2, so multiply back
measuredvbat *= 3.3;  // Multiply by 3.3V, our reference voltage
measuredvbat /= 1024; // convert to voltage
Serial.print("VBat: " ); Serial.println(measuredvbat);
```

This voltage will 'float' at 4.2V when no battery is plugged in, due to the lipoly charger output, so its not a good way to detect if a battery is plugged in or not (there is no simple way to detect if a battery is plugged in)

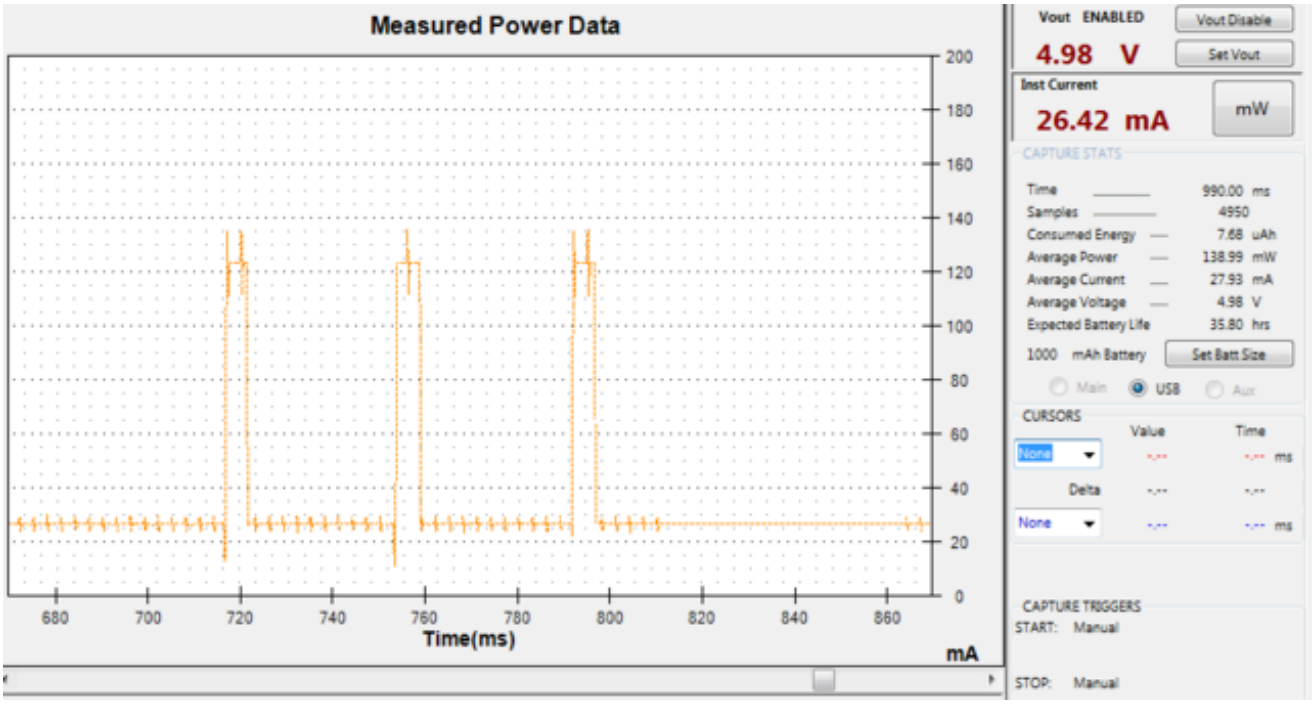


Radio Power Draw

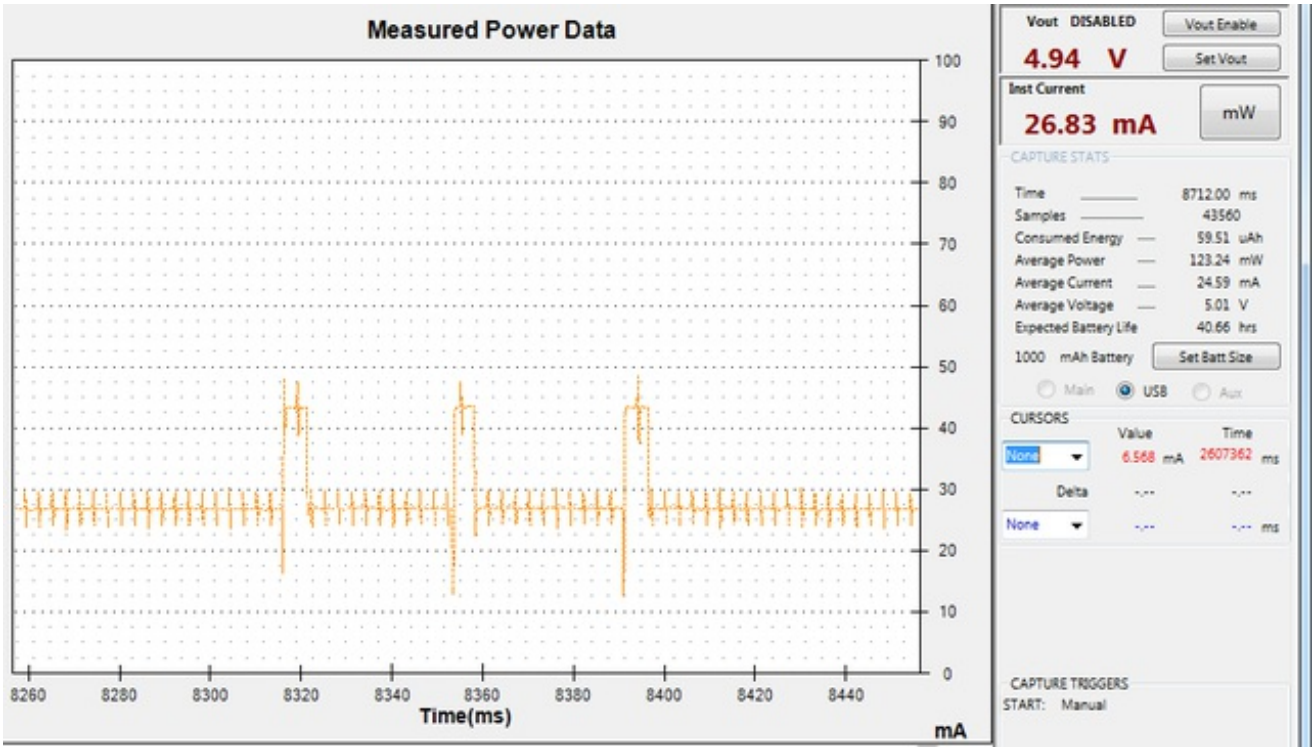
You can select the power output you want via software, more power equals more range but of course, uses more of your battery.

For example, here is the feather 32u4 with RFM69HCW set up for +20dBm power, transmitting a data payload of 20 bytes

The ~25mA quiescent current is the current draw for listening (~15mA) plus ~10mA for the microcontroller. This can be reduce to amost nothing with proper sleep modes and not putting the module in active listen mode!

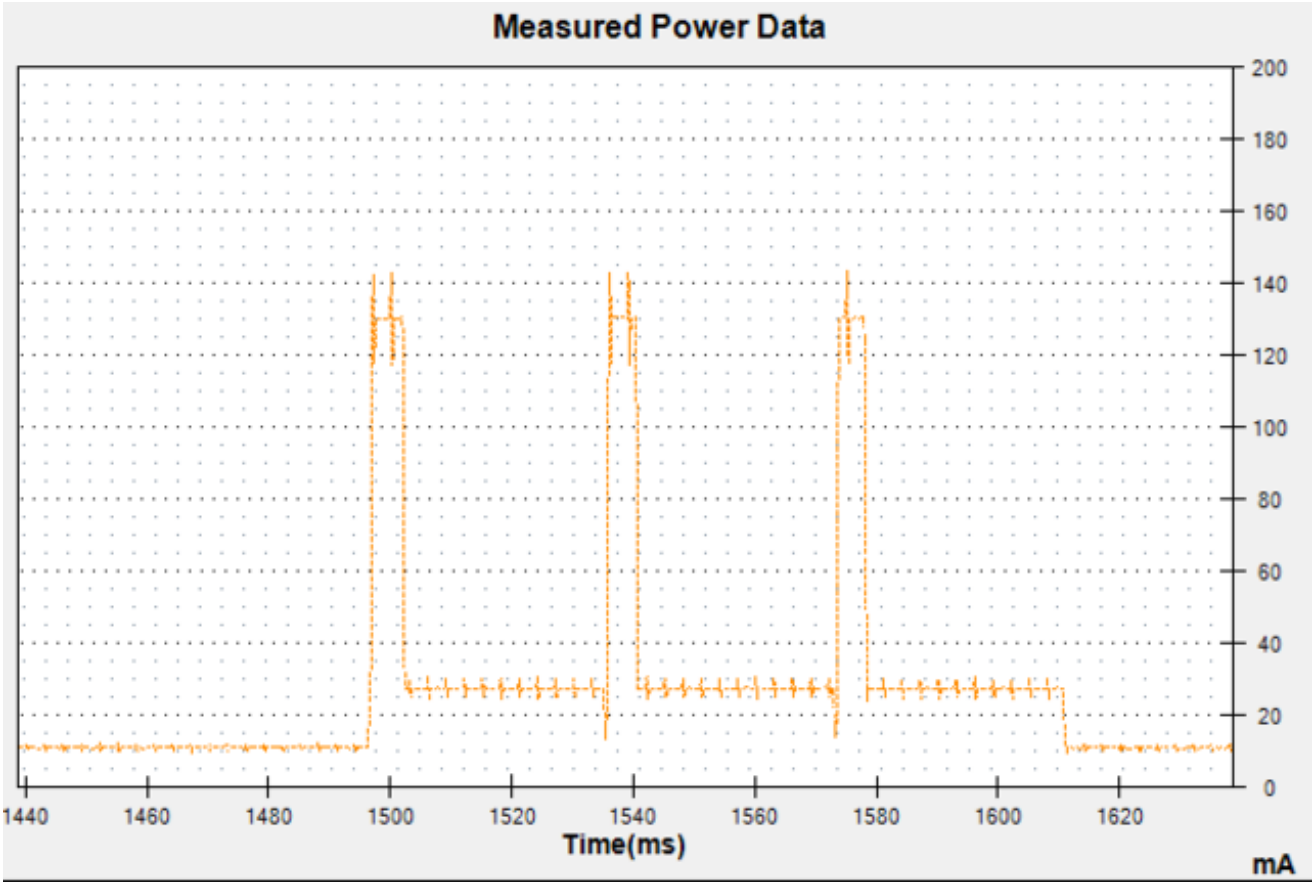


Here is a transmit with radio.setPowerLevel(0) to set +5dBm power



You still have the 25mA average, but during transmit, it only uses another 20mA not 100mA

If you put the radio to sleep after transmitting, rather than just sitting in receive mode, you can save more current, after transmit is complete, the average current drops to ~10mA which is just for the microcontroller



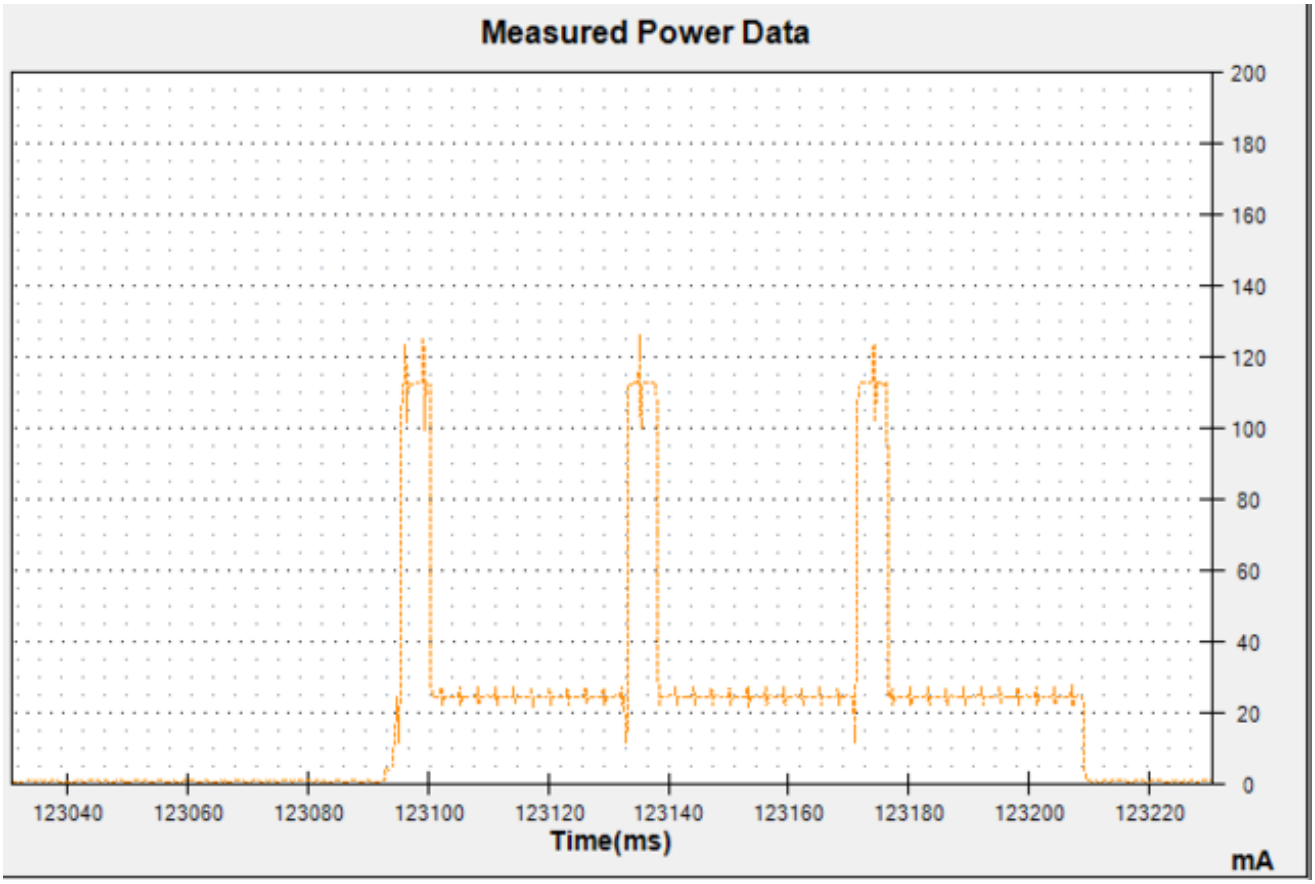
If you want to reduce even more power, use the[Adafruit Sleepdog (http://adafru.it/fp8)](http://adafru.it/fp8)library by installing and adding #include "Adafruit_SleepyDog.h" at the top of your sketch and replace

```
delay(1000);
```

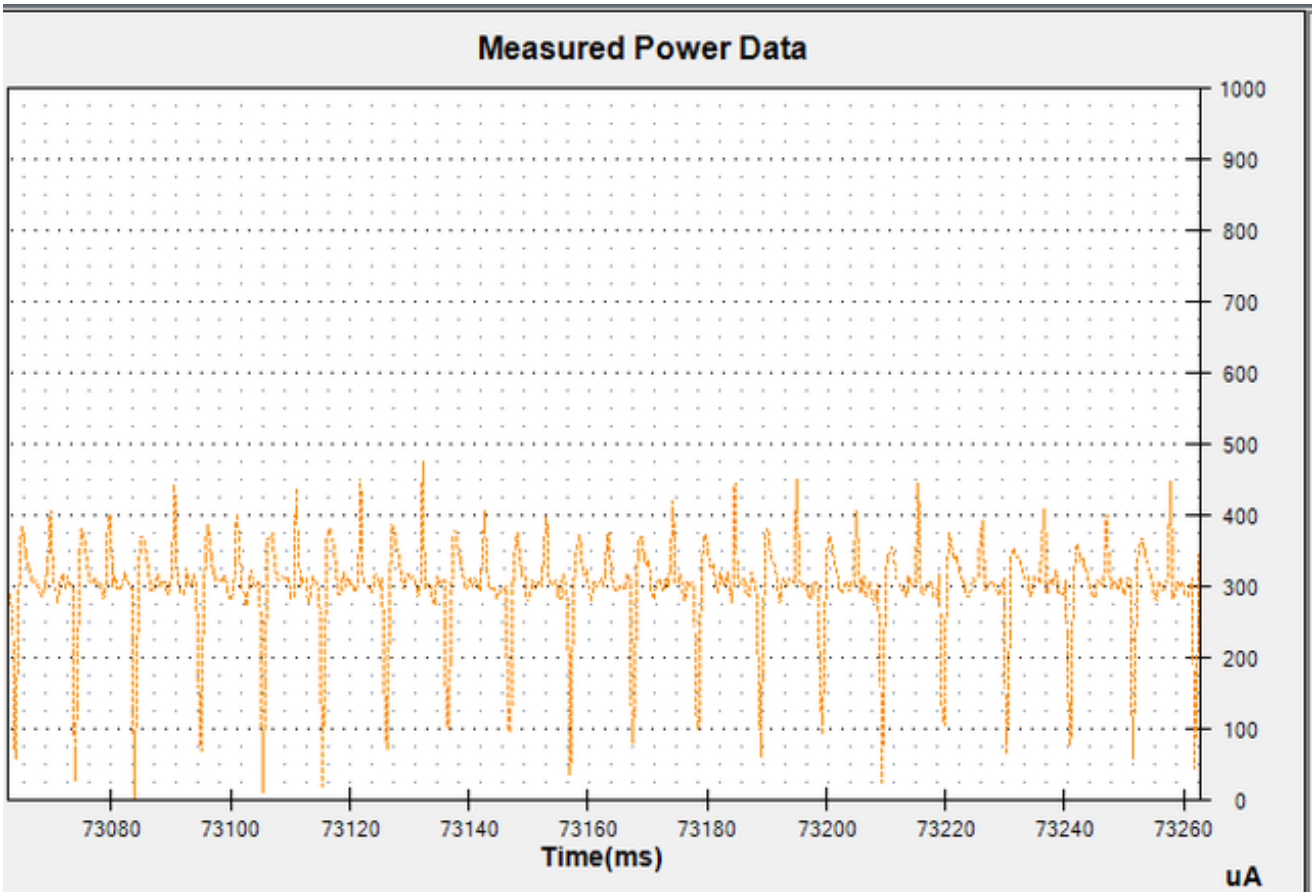
with

```
radio.sleep();
Watchdog.sleep(1000);
```

To put the chip into ultra-low-power mode. Note that USB will disconnect so do this after you have done all your debugging!



During the super sleepy mode you're using only 300uA (0.3mA)!

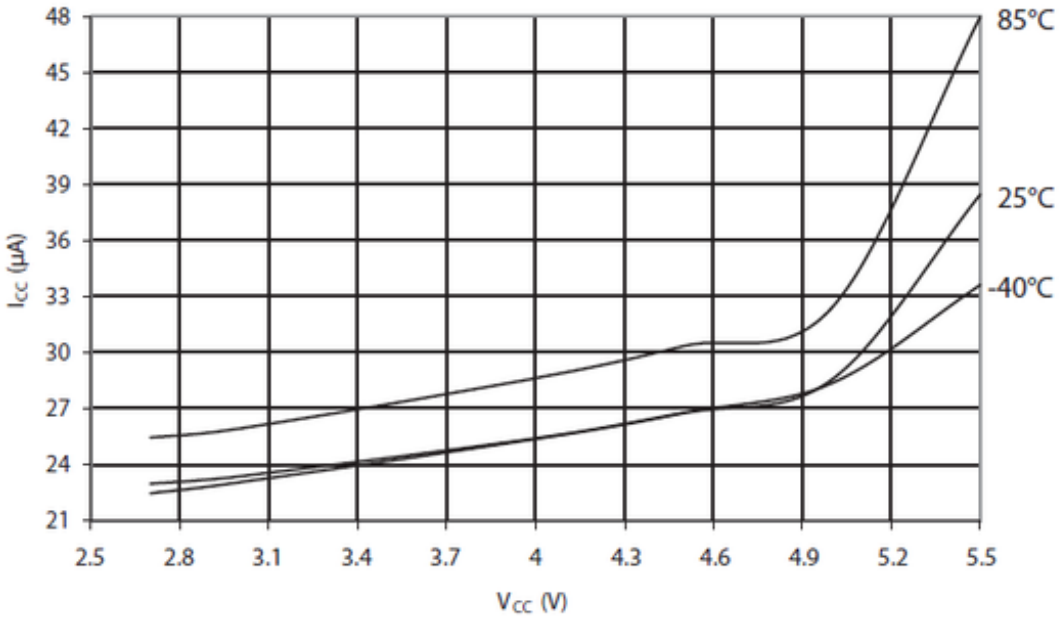


While its not easy to get the exact numbers for all of what comprise the 300uA there are a few quiescent current items on the Feather 32u4:

- 2 x 100K resistors for VBAT measurement =**25uA**
- AP2112K 3.3V regulator = **55uA**
- MCP73871 batt charger = **up to 100uA** even when no battery is connected

The rest is probably the Atmega32u4 peripherals including the brown-out detect and bandgap circuitry, ceramic oscillator, etc. According to the datasheet, with the watchdog and BrownOutDetect enabled, the lowest possible current is ~**30uA** (at 5V which is what we're testing at)

Figure 30-12. Power-down Supply Current vs. V_{CC} (WDT Enabled, BOD EN)



ENable pin

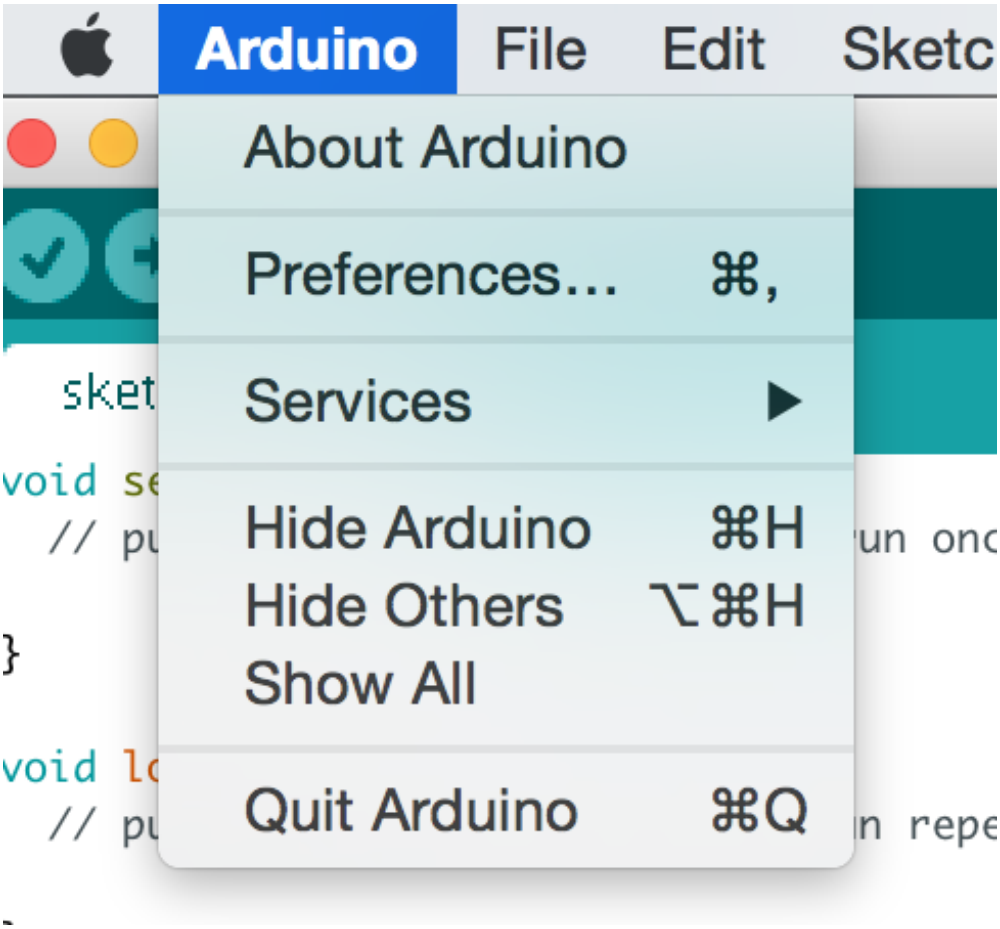
If you'd like to turn off the 3.3V regulator, you can do that with the**EN**(able) pin. Simply tie this pin to **Ground** and it will disable the 3V regulator. The **BAT** and **USB** pins will still be powered

Arduino IDE Setup

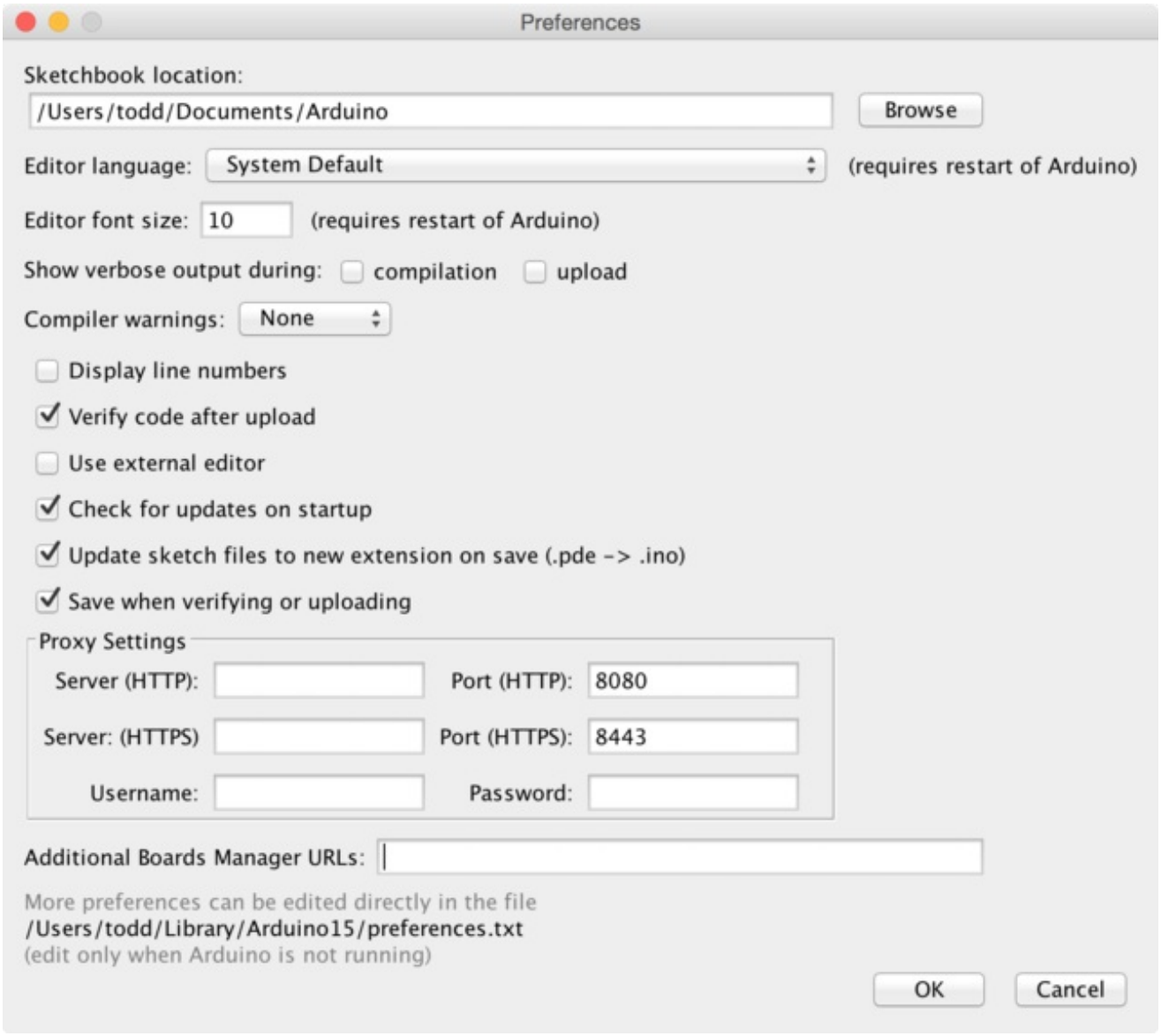
The first thing you will need to do is to download the latest release of the Arduino IDE. You will need to be using **version 1.6.4** or higher for this guide.

Arduino IDE v1.6.4+ Download
http://adafru.it/f1P

After you have downloaded and installed **v1.6.4**, you will need to start the IDE and navigate to the **Preferences** menu. You can access it from the**File** menu in *Windows* or *Linux*, or the **Arduino** menu on *OS X*.



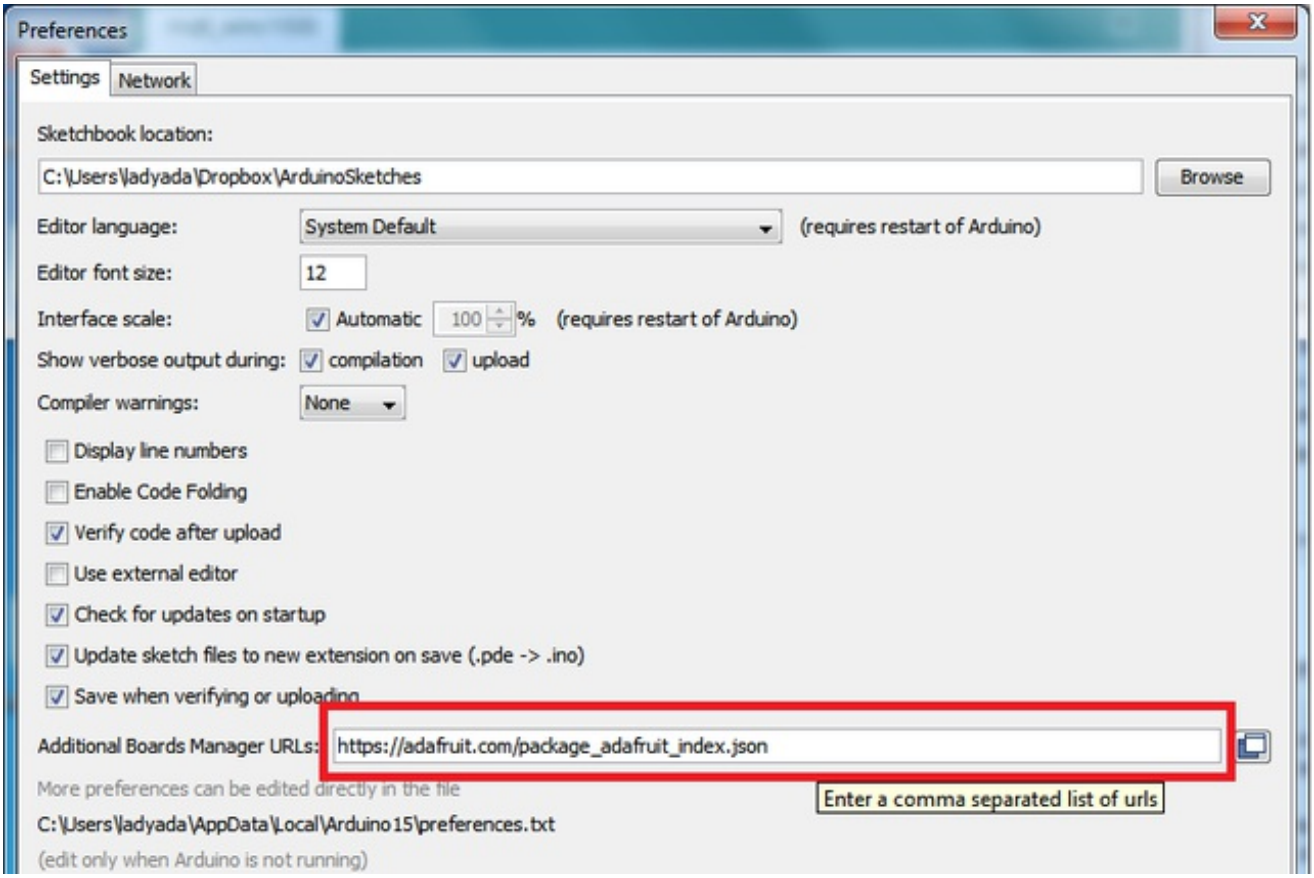
A dialog will pop up just like the one shown below.



We will be adding a URL to the new**Additional Boards Manager URLs** option. The list of URLs is comma separated, and *you will only have to add each URL once.*New Adafruit boards and updates to existing boards will automatically be picked up by the Board Manager each time it is opened. The URLs point to index files that the Board Manager uses to build the list of available & installed boards.

To find the most up to date list of URLs you can add, you can visit the list ofthird party board URLs on the Arduino IDE wiki (http://adafru.it/f7U). We will only need to add one URL to the IDE in this example, but ***you can add multiple URLS by separating them with commas***. Copy and paste the link below into the**Additional Boards Manager URLs** option in the Arduino IDE preferences.

https://adafruit.github.io/arduino-board-index/package_adafruit_index.json



Here's a short description of each of the Adafruit supplied packages that will be available in the Board Manager when you add the URL:

- **Adafruit AVR Boards** - Includes support for Flora, Gemma, Feather 32u4, Trinket, & Trinket Pro.
- **Adafruit SAMD Boards** - Includes support for Feather M0
- **Arduino Leonardo & Micro MIDI-USB** - This adds MIDI over USB support for the Flora, Feather 32u4, Micro and Leonardo using the arcore project (http://adafru.it/eSI).

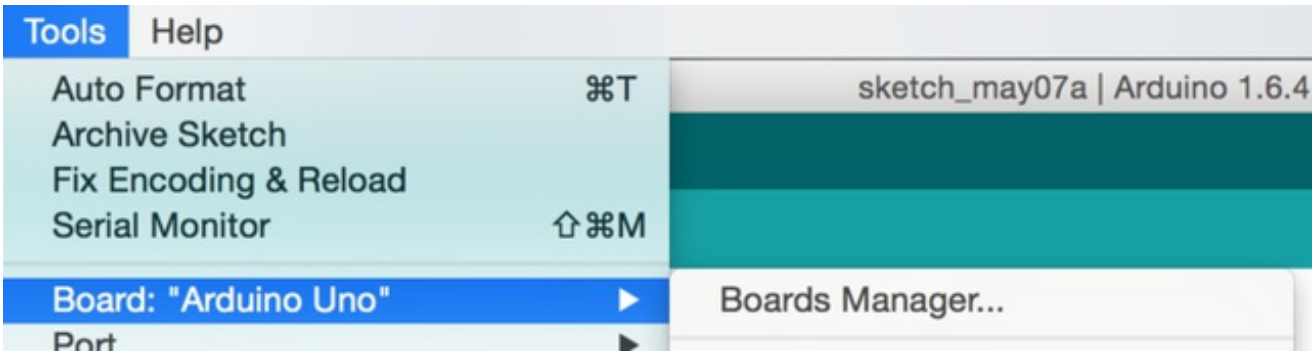
If you have multiple boards you want to support, say ESP8266 and Adafruit, have both URLs in the text box separated by a comma (,)

Once done click **OK** to save the new preference settings. Next we will look at installing boards with the Board Manager.

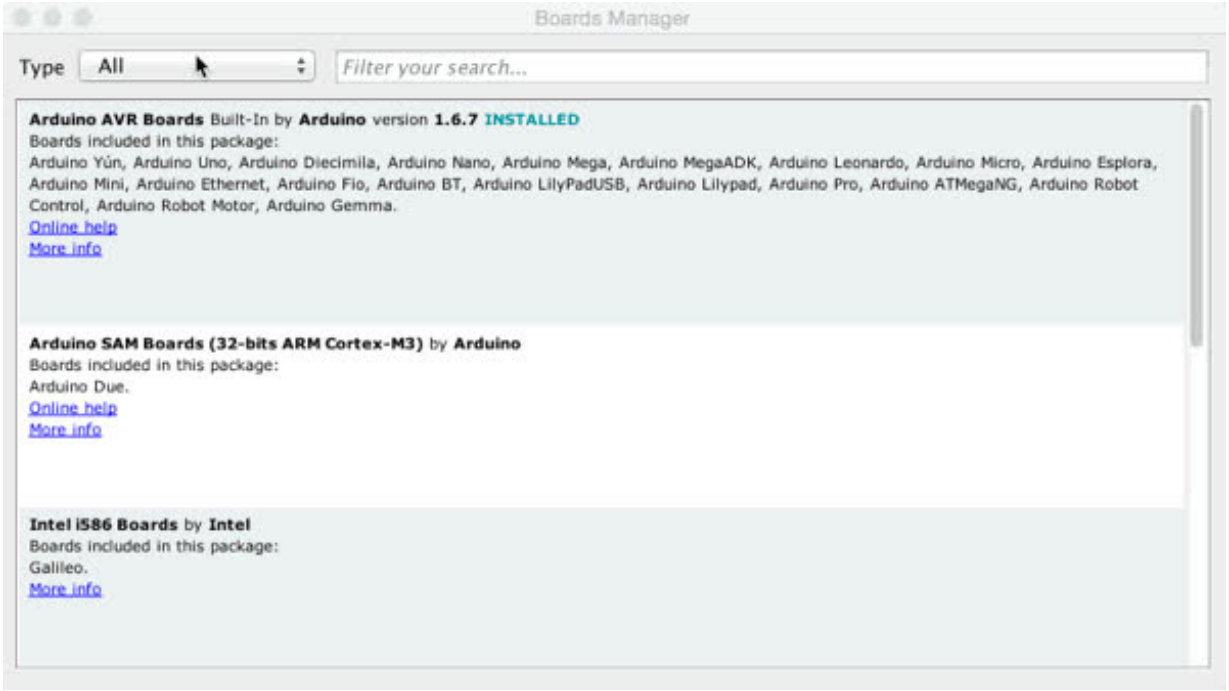
Using with Arduino IDE

Since the Feather 32u4 uses an ATmega32u4 chip running at 8 MHz, you can pretty easily get it working with the Arduino IDE. Many libraries (including the popular ones like NeoPixels and display) work great with the '32u4 and 8 MHz clock speed.

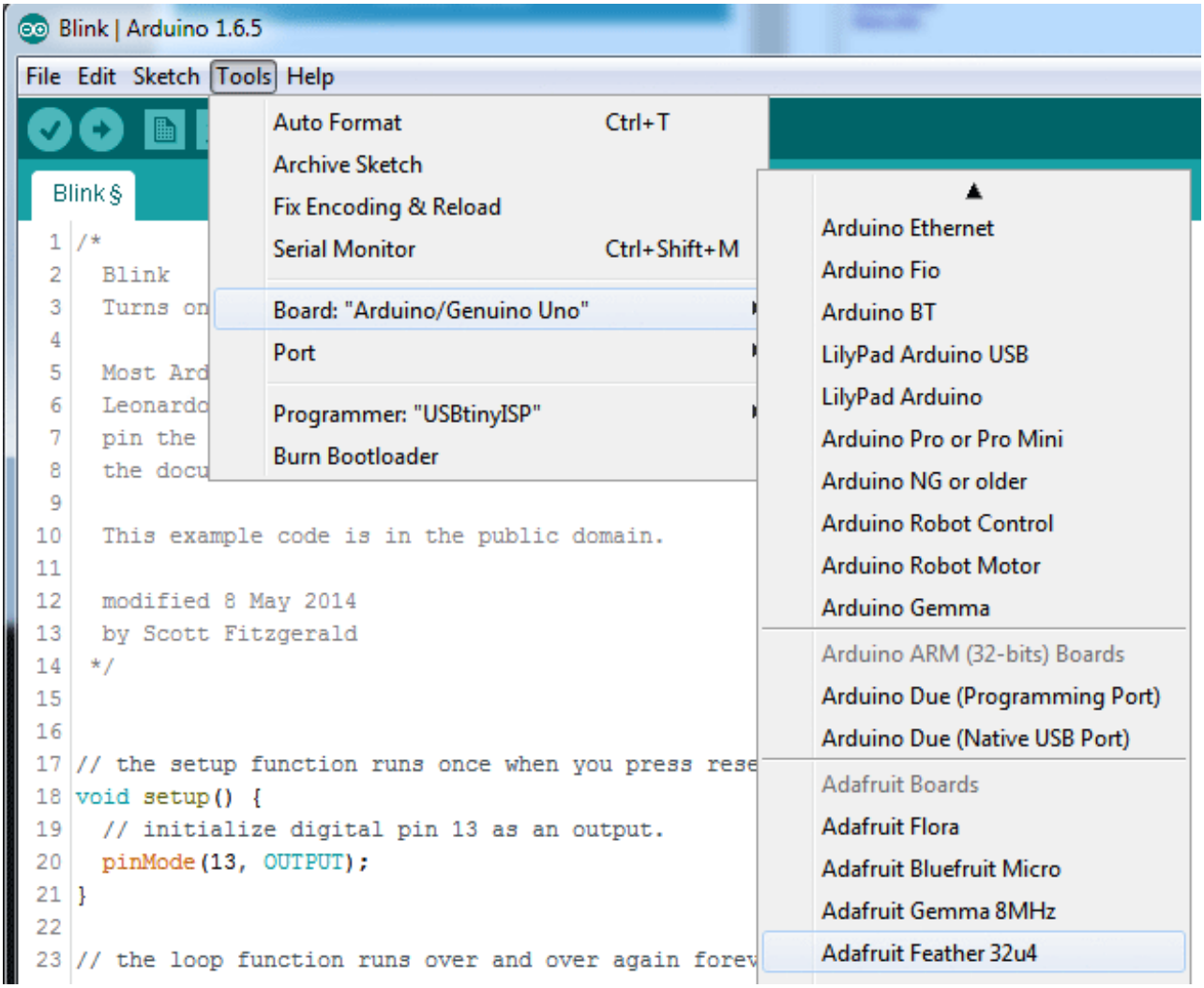
Now that you have added the appropriate URLs to the Arduino IDE preferences, you can open the **Boards Manager** by navigating to the **Tools->Board** menu.



Once the Board Manager opens, click on the category drop down menu on the top left hand side of the window and select **Contributed**. You will then be able to select and install the boards supplied by the URLs added to the prefrences. In the example below, we are installing support for ***Adafruit AVR Boards***, but the same applies to all boards installed with the Board Manager.



Next, **quit and reopen the Arduino IDE** to ensure that all of the boards are properly installed. You should now be able to select and upload to the new boards listed in the **Tools->Board** menu.



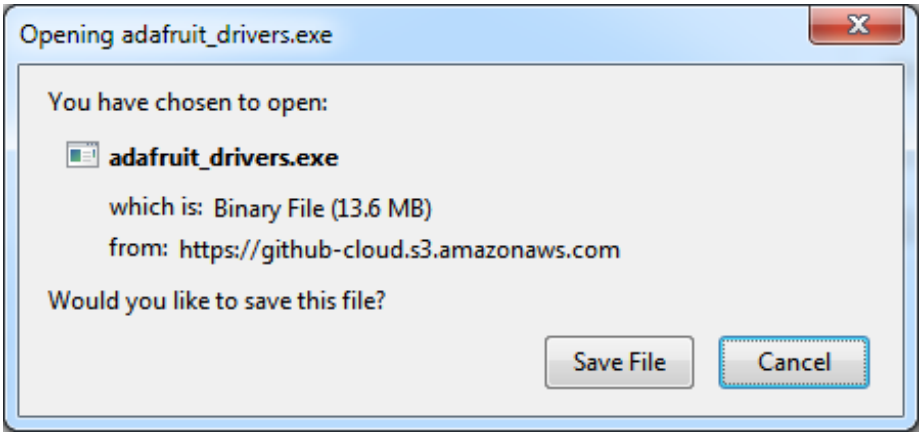
Install Drivers (Windows Only)

When you plug in the Feather, you'll need to possibly install a driver

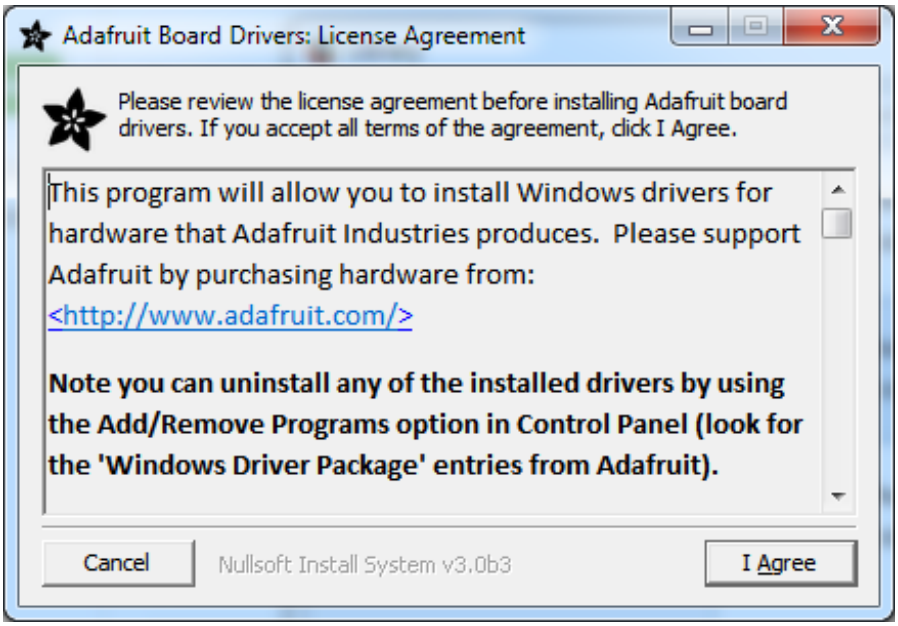
Click below to download our Driver Installer

[Download Adafruit Drivers Installer]
http://adafru.it/mai

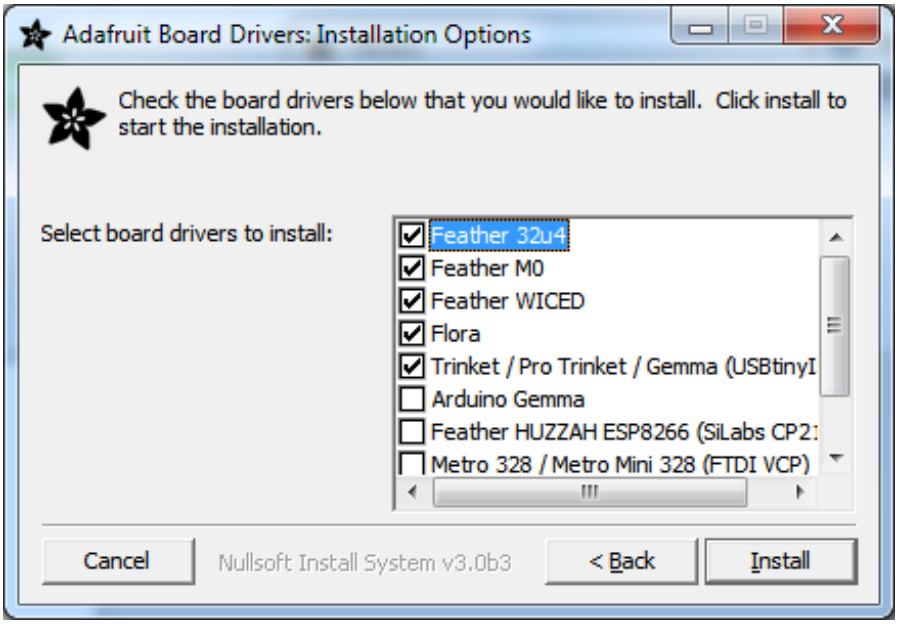
Download and run the installer



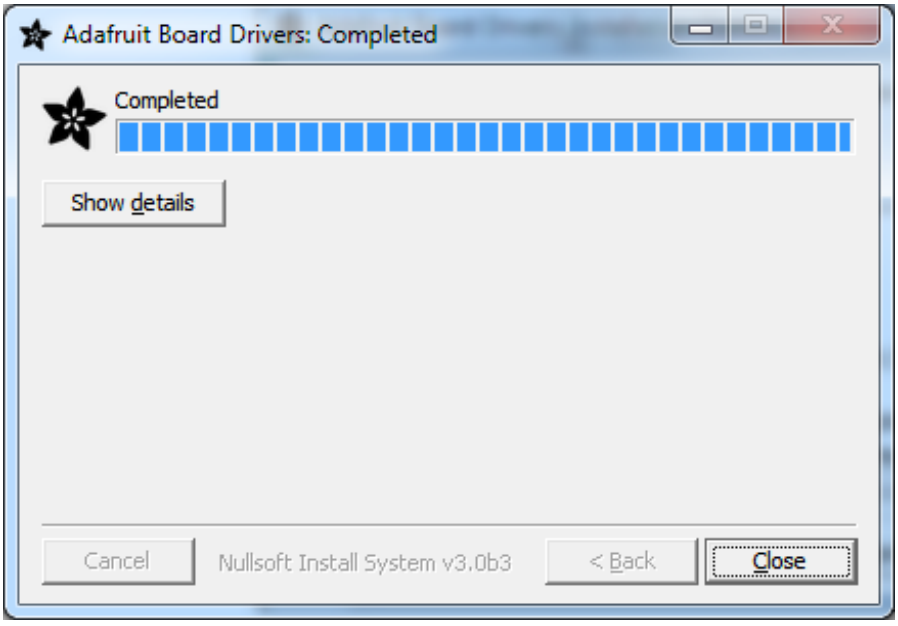
Run the installer! Since we bundle the SiLabs and FTDI drivers as well, you'll need to click through the license



Select which drivers you want to install:



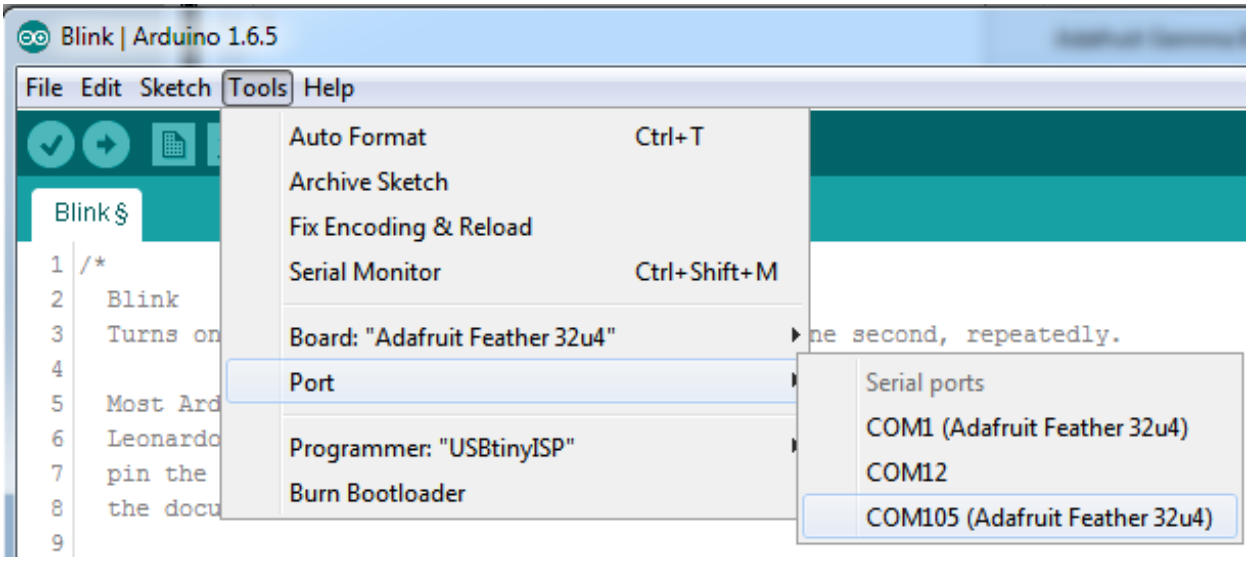
Click **Install** to do the installin'



Blink

Now you can upload your first blink sketch!

Plug in the Feather 32u4 and wait for it to be recognized by the OS (just takes a few seconds). It will create a serial/COM port, you can now select it from the dropdown, it'll even be 'indicated' as Feather 32u4!



Now load up the Blink example

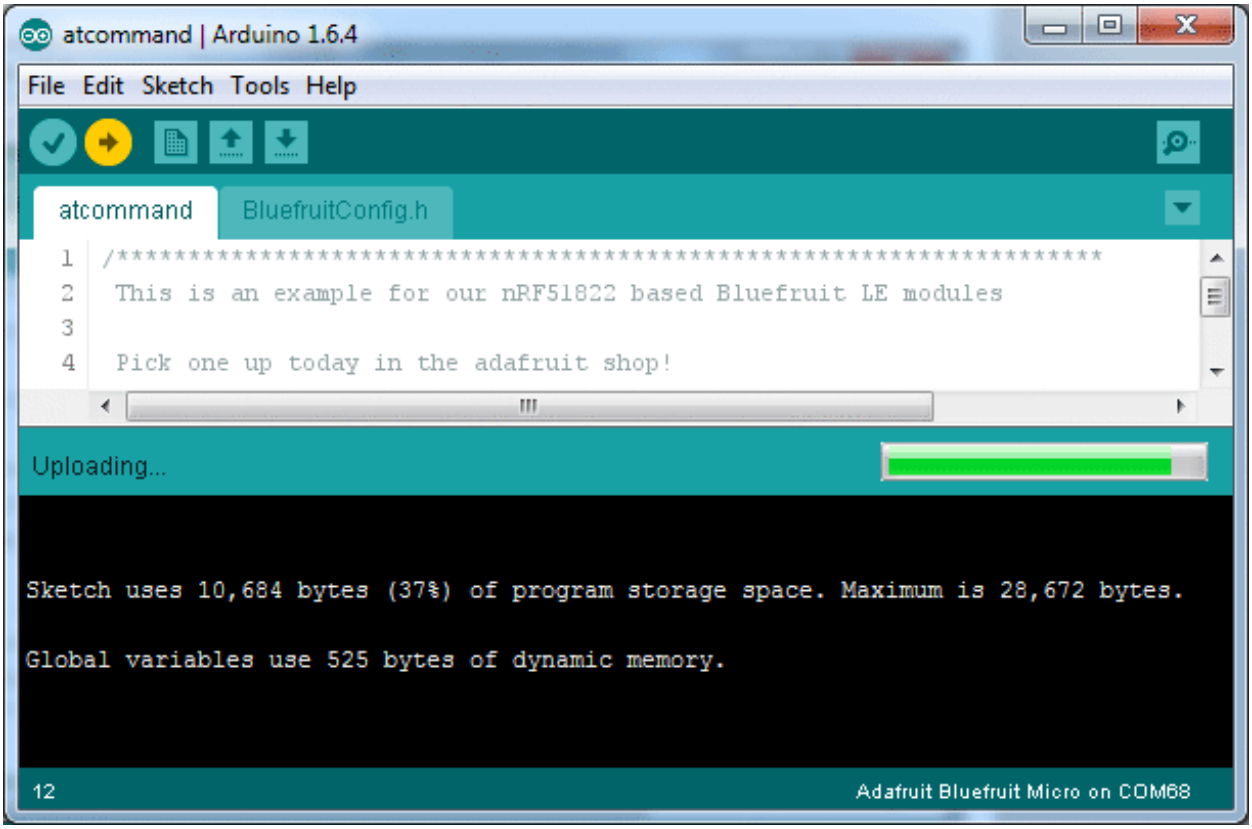
```
// the setup function runs once when you press reset or power the board
void setup() {
  // initialize digital pin 13 as an output.
  pinMode(13, OUTPUT);
}

// the loop function runs over and over again forever
void loop() {
  digitalWrite(13, HIGH);   // turn the LED on (HIGH is the voltage level)
  delay(1000);              // wait for a second
  digitalWrite(13, LOW);    // turn the LED off by making the voltage LOW
  delay(1000);              // wait for a second
}
```

And click upload! That's it, you will be able to see the LED blink rate change as you adapt the **delay()** calls.

Manually bootloading

If you ever get in a 'weird' spot with the bootloader, or you have uploaded code that crashes and doesn't auto-reboot into the bootloader, click the **RST** button to get back into the bootloader. The red LED will pulse, so you know that its in bootloader mode. Do the reset button press right as the Arduino IDE says its attempting to upload the sketch, when you see the Yellow Arrow lit and the **Uploading...** text in the status bar.



Don't click the reset button **before** uploading, unlike other bootloaders you want this one to run at the time Arduino is trying to upload

Ubuntu & Linux Issue Fix

Note if you're using Ubuntu 15.04 (or perhaps other more recent Linux distributions) there is an issue with the modem manager service which causes the Bluefruit LE micro to be difficult to program. If you run into errors like "device or resource busy", "bad file descriptor", or "port is busy" when attempting to program then [you are hitting this issue.](http://adafru.it/sHE)

The fix for this issue is to make sure Adafruit's custom udev rules are applied to your system. One of these rules is made to configure modem manager not to touch the Bluefruit Micro board and will fix the programming difficulty issue. [Follow the steps for installing Adafruit's udev rules on this page.](http://adafru.it/iOE)

Feather HELP!

My Feather stopped working when I unplugged the USB!

A lot of our example sketches have a

```
while (!Serial);
```

line in setup(), to keep the board waiting until the USB is opened. This makes it a lot easier to debug a program because you get to see all the USB data output. If you want to run your Feather without USB connectivity, delete or comment out that line

My Feather never shows up as a COM or Serial port in the Arduino IDE

A vast number of Feather 'failures' are due to charge-only USB cables

We get upwards of 5 complaints a day that turn out to be due to charge-only cables!

Use only a cable that you **know** is for data syncing

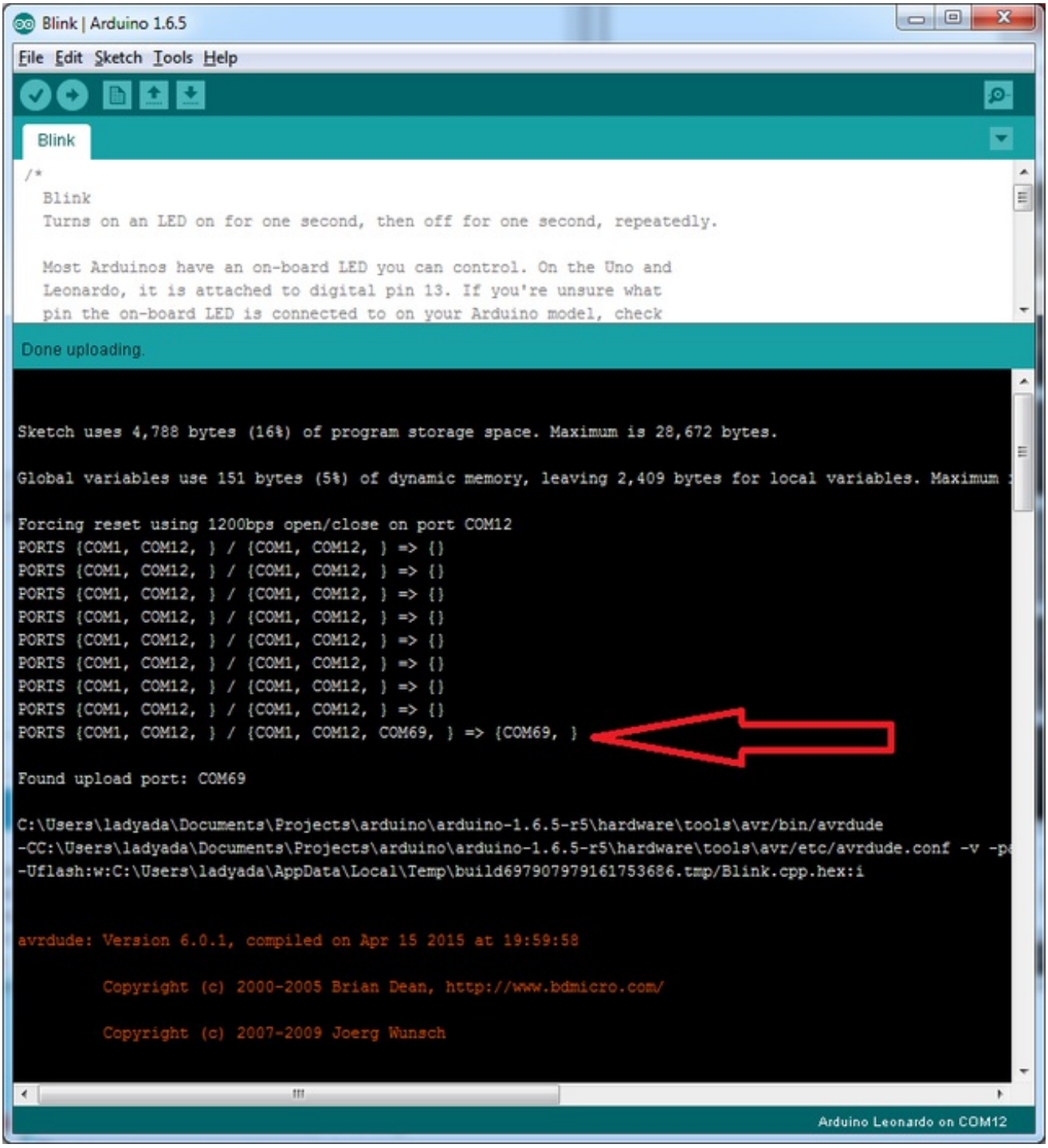
If you have any charge-only cables, cut them in half throw them out. We are serious! They tend to be low quality in general, and will only confuse you and others later, just get a good data+charge USB cable

Ack! I "did something" and now when I plug in the Feather, it doesn't show up as a device anymore so I cant upload to it or fix it...

No problem! You can 'repair' a bad code upload easily. Note that this can happen if you set a watchdog timer or sleep mode that stops USB, or any sketch that 'crashes' your Feather

1. Turn on **verbose upload** in the Arduino IDE preferences
2. Plug in feather 32u4/M0, it won't show up as a COM/serial port that's ok
3. Open up the Blink example (Examples->Basics->Blink)
4. Select the correct board in the Tools menu, e.g. Feather 32u4 or Feather M0 (check your board to make sure you have the right one selected!)
5. Compile it (make sure that works)
6. Click Upload to attempt to upload the code
7. The IDE will print out a bunch of COM Ports as it tries to upload. **During this time, double-click the reset button, you'll see the red pulsing LED that tells you its now in bootloading mode**

8. The Feather will show up as the Bootloader COM/Serial port
9. The IDE should see the bootloader COM/Serial port and upload properly



I can't get the Feather USB device to show up - I get "USB Device Malfunctioning" errors!

This seems to happen when people select the wrong board from the Arduino Boards menu.

If you have a Feather 32u4 (look on the board to read what it is you have) Make sure you select **Feather 32u4** for ATMega32u4 based boards! Do not use anything else, do not use

the 32u4 breakout board line.

If you have a Feather M0 (look on the board to read what it is you have) Make sure you select **Feather M0** - do not use 32u4 or Arduino Zero

I'm having problems with COM ports and my Feather 32u4/M0

Theres **two** COM ports you can have with the 32u4/M0, one is the**user port** and one is the **bootloader port**. They are not the same COM port number!

When you upload a new user program it will come up with a user com port, particularly if you use Serial in your user program.

If you crash your user program, or have a program that halts or otherwise fails, the user com port can disappear.

When the user COM port disappears, Arduino will not be able to automatically start the bootloader and upload new software.

So you will need to help it by performing the click-during upload procedure to re-start the bootloader, and upload something that is known working like "Blink"

I don't understand why the COM port disappears, this does not happen on my Arduino UNO!

UNO-type Arduinos have a *seperate* serial port chip (aka "FTDI chip" or "Prolific PL2303" etc etc) which handles all serial port capability seperately than the main chip. This way if the main chip fails, you can always use the COM port.

M0 and 32u4-based Arduinos do not have a seperate chip, instead the main processor performs this task for you. It allows for a lower cost, higher power setup...but requires a little more effort since you will need to 'kick' into the bootloader manually once in a while

I'm trying to upload to my 32u4, getting "avrdude: butterfly_recv(): programmer is not responding" errors

This is likely because the bootloader is not kicking in and you are accidentally**trying to upload to the wrong COM port**

The best solution is what is detailed above: manually upload Blink or a similar working sketch by hand by manually launching the bootloader

I'm trying to upload to my Feather M0, and I get this error "Connecting to programmer: .avrdude: butterfly_recv(): programmer is not responding"

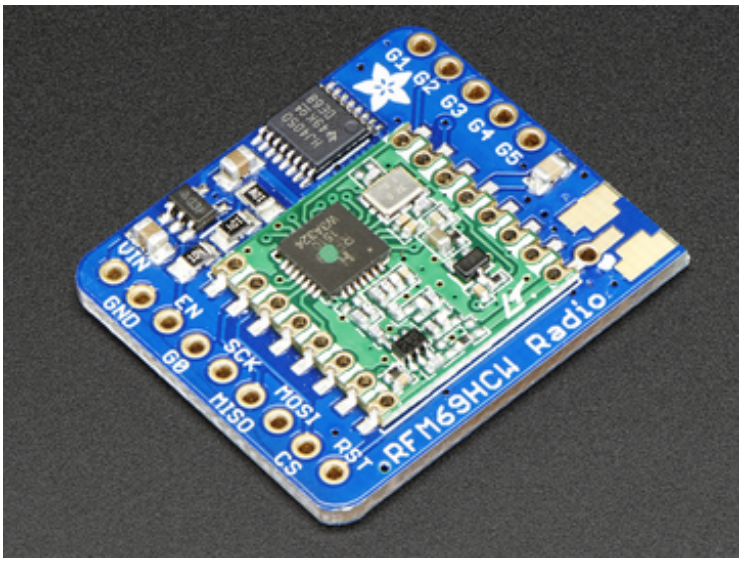
You probably don't have Feather M0 selected in the boards drop-down. Make sure you

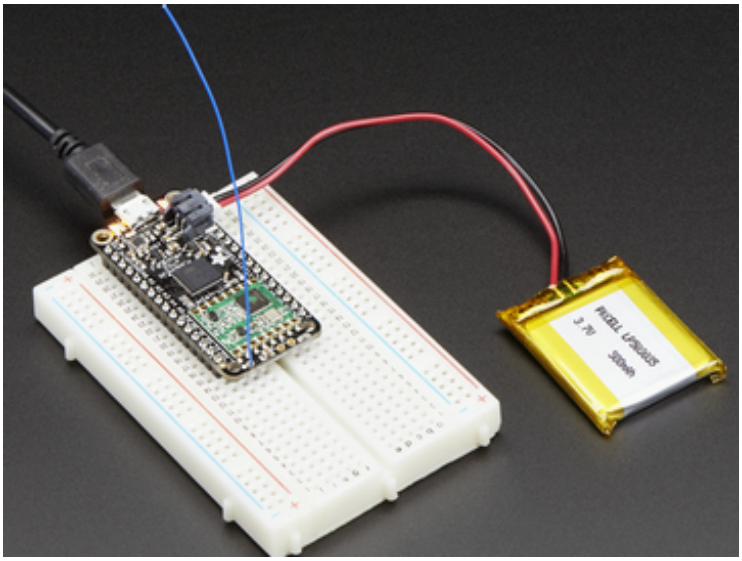
You probably don't have Feather M0 selected in the boards drop-down. Make sure you selected Feather M0.

I'm trying to upload to my Feather and i get this error "avrdude: ser_recv(): programmer is not responding"

You probably don't have Feather M0 / Feather 32u4 selected in the boards drop-down. Make sure you selected Feather M0 (or Feather 32u4).

Using the RFM69 Radio

- 

- 

This page is shared between the RFM69 breakout and the all-in-one Feather RFM69's. The example code and overall functionality is the same, only the pinouts used may differ! Just make sure the example code is using the pins you have wired up.

Before beginning make sure you have your Arduino or Feather working smoothly, it will make this part a lot easier. Once you have the basic functionality going - you can upload code, blink an LED, use the serial output, etc. you can then upgrade to using the radio itself.

Note that the sub-GHz radio is not designed for streaming audio or video! It's best used for small packets of data. The data rate is adjustable but its common to stick to around 19.2 Kbps (thats bits per second). Lower data rates will be more successful in their transmissions

You will, of course, need at least two paired radios to do any testing! The radios must be matched in frequency (e.g. 900 MHz & 900 MHz are ok, 900 MHz & 433 MHz are not). They also must use the same encoding schemes, you cannot have a 900 MHz RFM69 packet radio talk to a 900 MHz RFM9x LoRa radio.

"Raw" vs Packetized

The SX1231 can be used in a 'raw rx/tx' mode where it just modulates incoming bits from pin #2 and sends them on the radio, however there's no error correction or addressing so we wont be covering that technique.

Instead, 99% of cases are best off using packetized mode. This means you can set up a recipient for your data, error correction so you can be sure the whole data set was transmitted correctly, automatic re-transmit retries and return-receipt when the packet was delivered. Basically, you get the transparency of a data pipe without the annoyances of radio transmission unreliability

Arduino Libraries

These radios have really great libraries already written, so rather than coming up with a new standard we suggest using existing libraries such as LowPowerLab's RFM69 Library (http://adafru.it/mCz) and AirSpayce's Radiohead library (http://adafru.it/mCA) which also suppors a vast number of other radios

These are really great Arduino Libraries, so please support both companies in thanks for their efforts!

We recommend using the **Radiohead library** - it is very cross-platform friendly and used a lot in the community!

RadioHead Library example

To begin talking to the radio, you will need to download our small fork of the Radiohead from our github repository (http://adafru.it/vgE). You can do that by visiting the github repo and manually downloading or, easier, just click this button to download the zip

Download RadioHead Library
http://adafru.it/vgF

Rename the uncompressed folder **RadioHead** and check that the **RadioHead** folder

contains files like **RH_RFM69.cpp** and **RH_RFM69.h** (and many others!)

Place the **RadioHead** library folder in your ***arduinosketchfolder*/libraries/** folder.
You may need to create the**libraries** subfolder if it's 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)

Basic RX & TX example

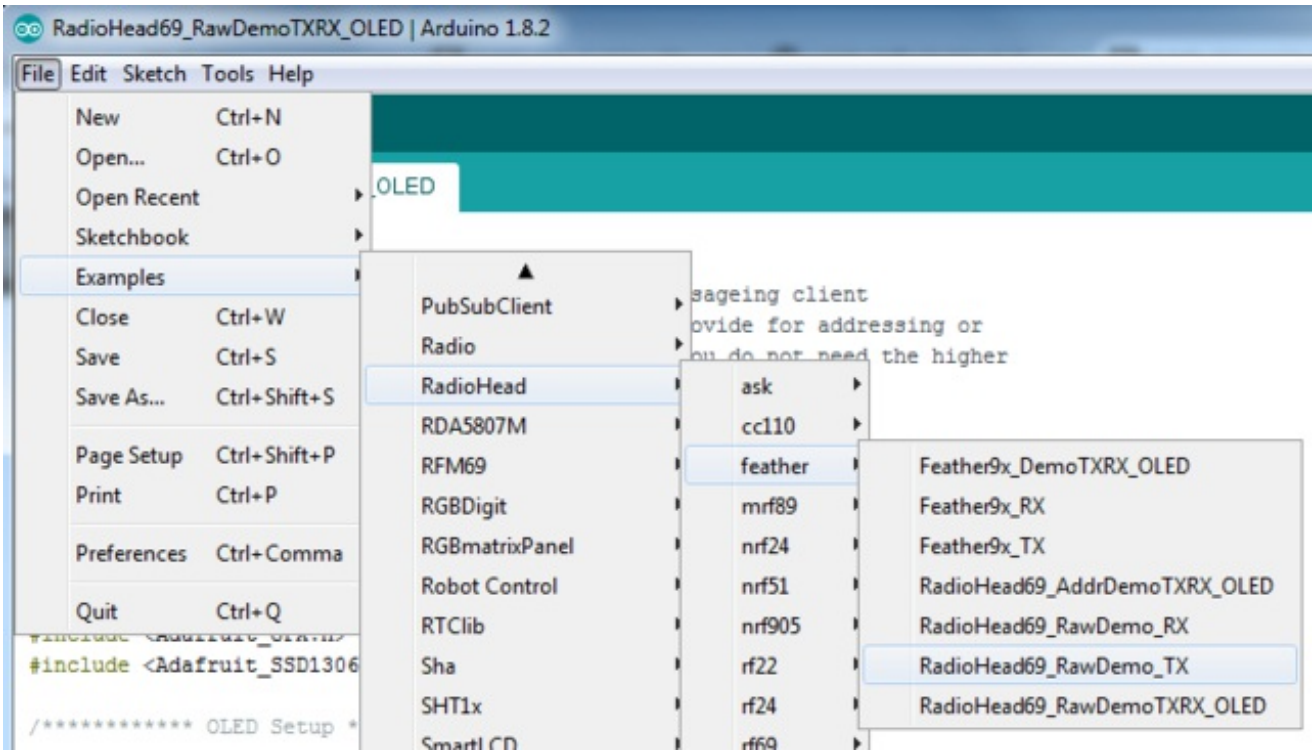
Lets get a basic demo going, where one radio transmits and the other receives. We'll start by setting up the transmitter

Basic Transmitter example code

This code will send a small packet of data once a second to another RFM69 radio, without any addressing.

Open up the example **RadioHead -> feather -> RadioHead69_RawDemo_TX**

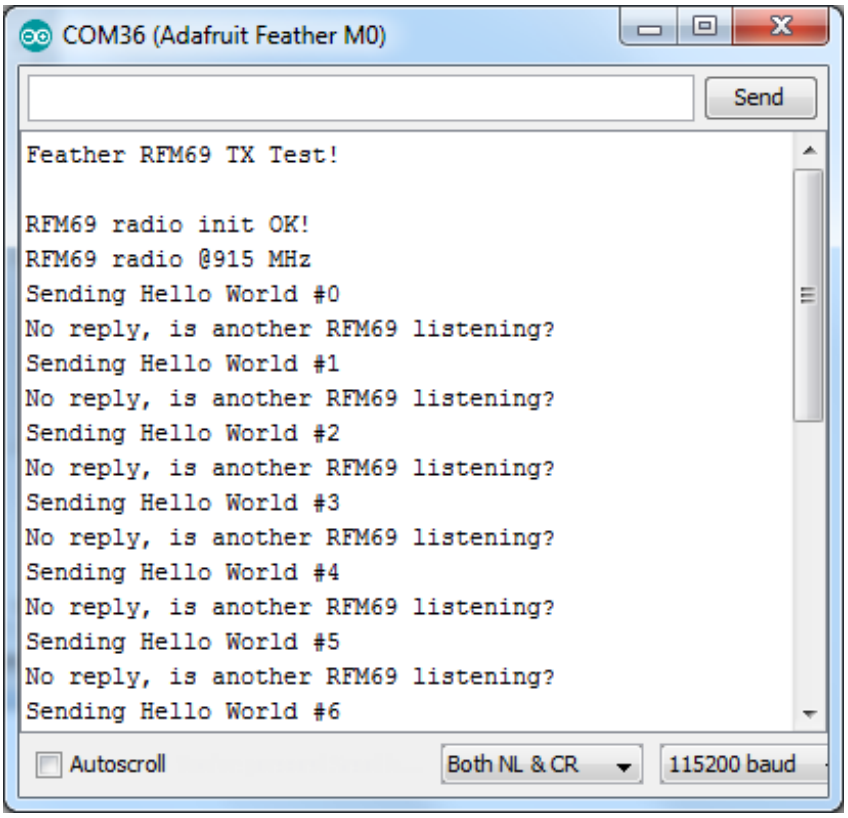
Load this code into your Transmitter Arduino or Feather!



Before uploading, check for the #define FREQUENCY RF69_915MHZ line and comment that out (and uncomment the line above) to match the frequency of the hardware you're

using
These examples are optimized for the Feather 32u4/M0. If you're using differnet wiring, uncomment/comment/edit the sections defining the pins depending on which chipset and wiring you are using! The pins used will vary depending on your setup!

Once uploaded you should see the following on the serial console



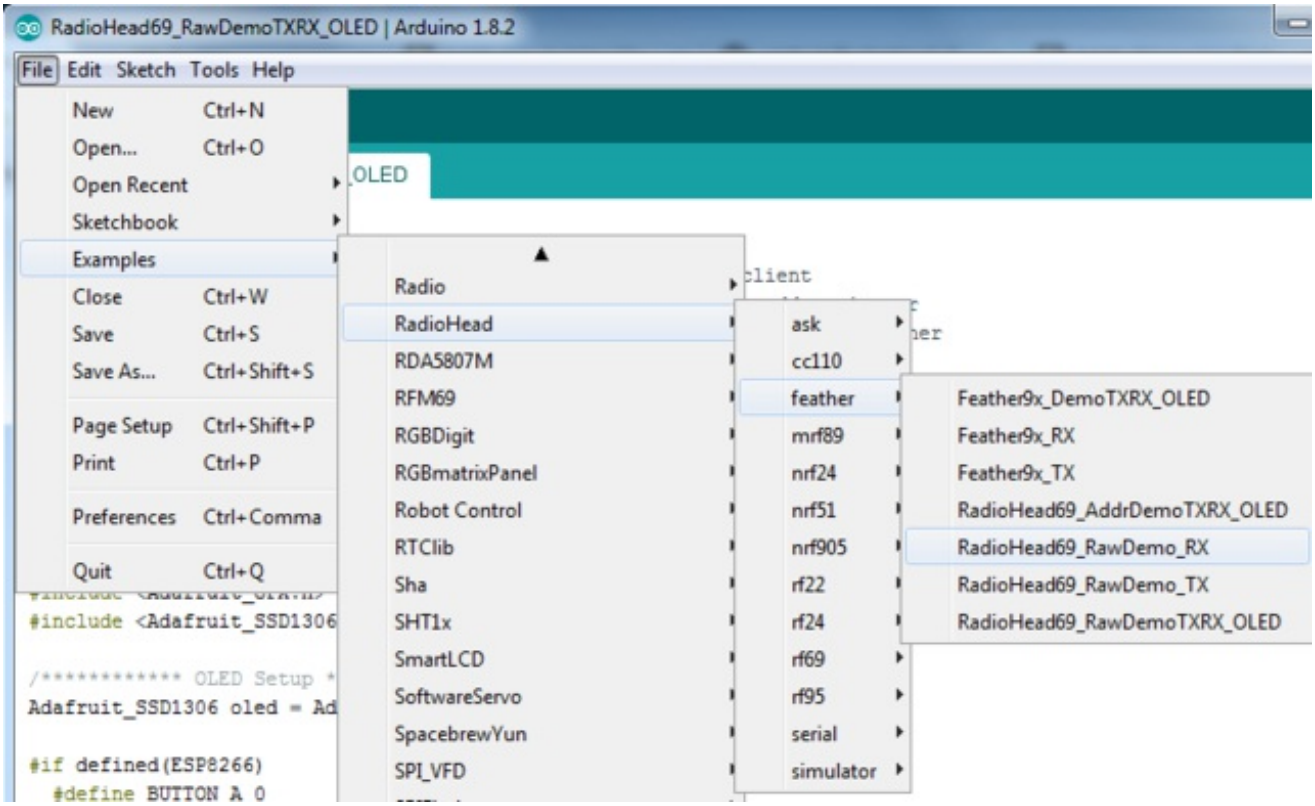
Now open up another instance of the Arduino IDE - this is so you can see the serial console output from the TX device while you set up the RX device.

Basic receiver example code

This code will receive and reply with a small packet of data.

Open up the example **RadioHead -> feather -> RadioHead69_RawDemo_RX**

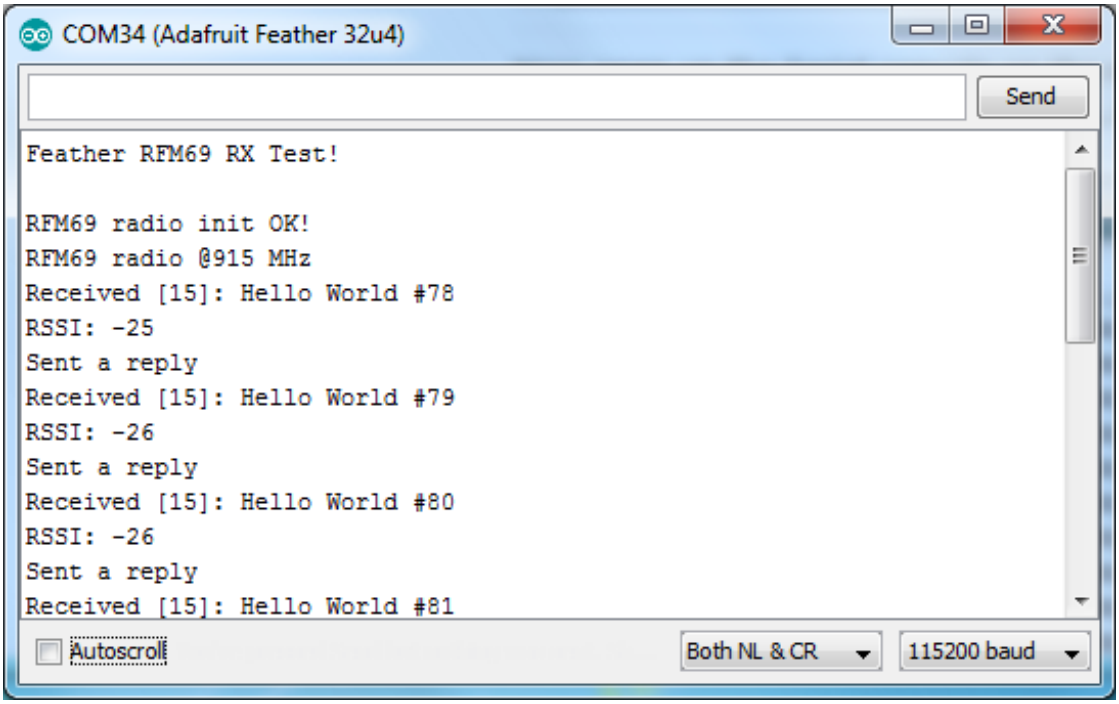
Load this code into your **Receiver** Arduino/Feather!



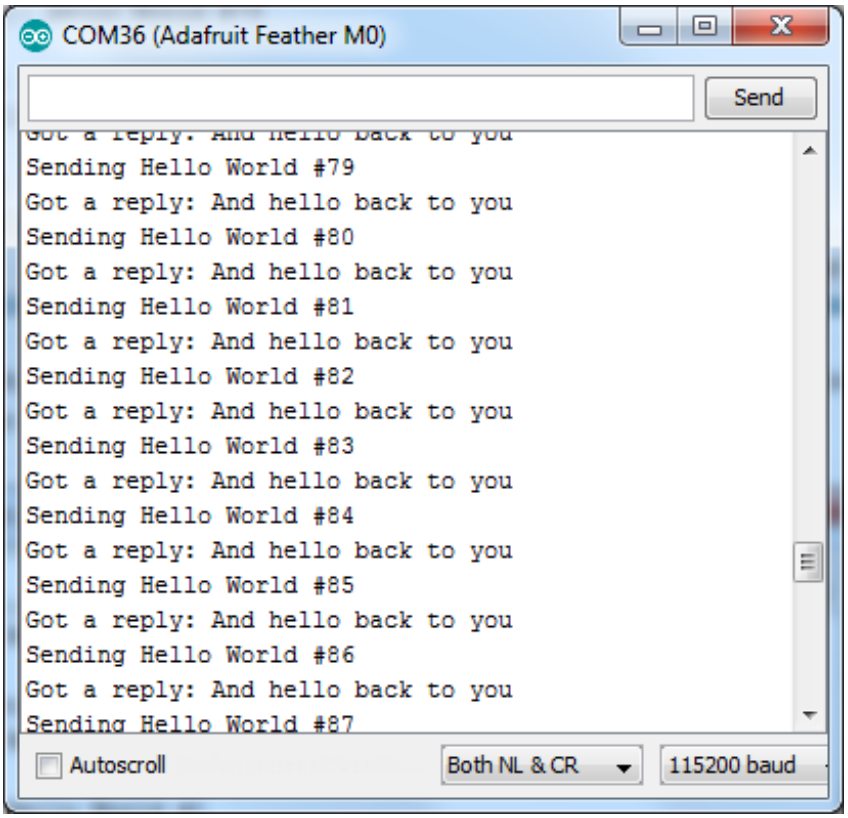
Before uploading, check for the #define FREQUENCY RF69_915MHZ line and comment that out (and uncomment the line above) to match the frequency of the hardware you're using

These examples are optimized for the Feather 32u4/M0. If you're using differnet wiring, uncomment/comment/edit the sections defining the pins depending on which chipset and wiring you are using! The pins used will vary depending on your setup!

Now open up the Serial console on the receiver, while also checking in on the transmitter's serial console. You should see the receiver is...well, receiving packets



And, on the transmitter side, it is now printing**Got Reply** after each transmisssion because it got a reply from the receiver



That's pretty much the basics of it! Lets take a look at the examples so you know how to adapt to your own radio network

Radio Freq. Config

Each radio has a frequency that is configurable in software. You can actually tune outside the recommended frequency, but the range won't be good. 900 MHz can be tuned from about 850-950MHz with good performance. 433 MHz radios can be tuned from 400-460 MHz or so.

```
// Change to 434.0 or other frequency, must match RX's freq!
#define RF69_FREQ 915.0
```

For all radios they will need to be on the same frequency. If you have a 433MHz radio you will want to stick to 433. If you have a 900 Mhz radio, go with 868 or 915MHz, just make sure all radios are on the same frequency

Configuring Radio Pinout

At the top of the sketch you can also set the pinout. The radios will use hardware SPI, but you can select any pins for **RFM69_CS** (an output), **RFM_IRQ** (an input) and **RFM_RST** (an output). RFM_RST is manually used to reset the radio at the beginning of the sketch. **RFM_IRQ** must be an interrupt-capable pin. Check your board to determine which pins you can use!

Also, an LED is defined.

For example, here is the Feather 32u4 pinout

```
#if defined (__AVR_ATmega32U4__) // Feather 32u4 w/Radio
  #define RFM69_CS      8
  #define RFM69_INT     7
  #define RFM69_RST     4
  #define LED           13
#endif
```

If you're using a Feather M0, the pinout is slightly different:

```
#if defined(ARDUINO_SAMD_FEATHER_M0) // Feather M0 w/Radio
  #define RFM69_CS      8
  #define RFM69_INT     3
  #define RFM69_RST     4
  #define LED           13
#endif
```

If you're using an Arduino UNO or compatible, we recommend:

```
#if defined (__AVR_ATmega328P__)  // UNO or Feather 328P w/wing
  #define RFM69_INT     3  //
```

```
  #define RFM69_CS      4  //
  #define RFM69_RST     2  // "A"
  #define LED           13
#endif
```

If you're using a FeatherWing or different setup, you'll have to set up the`#define` statements to match your wiring

You can then instantiate the radio object with our custom pin numbers. Note that the IRQ is defined by the IRQ pin not number (sometimes they differ).

```
// Singleton instance of the radio driver
RH_RF69 rf69(RFM69_CS, RFM69_INT);
```

Setup

We begin by setting up the serial console and hard-resetting the RFM69

```
void setup()
{
  Serial.begin(115200);
  //while (!Serial) { delay(1); } // wait until serial console is open, remove if not tethered to computer

  pinMode(LED, OUTPUT);
  pinMode(RFM69_RST, OUTPUT);
  digitalWrite(RFM69_RST, LOW);

  Serial.println("Feather RFM69 RX Test!");
  Serial.println();

  // manual reset
  digitalWrite(RFM69_RST, HIGH);
  delay(10);
  digitalWrite(RFM69_RST, LOW);
  delay(10);
```

If you are using a board with 'native USB' make sure the**while (!Serial)** line is commented out if you are not tethering to a computer, as it will cause the microcontroller to halt until a USB connection is made!

Initializing Radio

Once initialized, you can set up the frequency, transmission power, radio type and encryption key.

For the **frequency**, we set it already at the top of the sketch

For **transmission power** you can select from 14 to 20 dBi. Lower numbers use less power, but have less range. The second argument to the function is whether it is an HCW type radio, with extra amplifier. This should *always* be set to **true**!

Finally, if you are **encrypting** data transmission, set up the encryption key

```
 if (!rf69.init()) {
   Serial.println("RFM69 radio init failed");
   while (1);
 }
 Serial.println("RFM69 radio init OK!");

 // Defaults after init are 434.0MHz, modulation GFSK_Rb250Fd250, +13dbM (for low power module)
 // No encryption
 if (!rf69.setFrequency(RF69_FREQ)) {
   Serial.println("setFrequency failed");
 }

 // If you are using a high power RF69 eg RFM69HW, you *must* set a Tx power with the
 // ishighpowermodule flag set like this:
 rf69.setTxPower(20, true);  // range from 14-20 for power, 2nd arg must be true for 69HCW

 // The encryption key has to be the same as the one in the server
 uint8_t key[] = { 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08,
                   0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08};
 rf69.setEncryptionKey(key);
```

Basic Transmission Code

If you are using the transmitter, this code will wait 1 second, then transmit a packet with "Hello World #" and an incrementing packet number, then check for a reply

```
void loop() {
  delay(1000);  // Wait 1 second between transmits, could also 'sleep' here!

  char radiopacket[20] = "Hello World #";
  itoa(packetnum++, radiopacket+13, 10);
  Serial.print("Sending "); Serial.println(radiopacket);

  // Send a message!
  rf69.send((uint8_t *)radiopacket, strlen(radiopacket));
  rf69.waitPacketSent();

  // Now wait for a reply
  uint8_t buf[RH_RF69_MAX_MESSAGE_LEN];
  uint8_t len = sizeof(buf);

  if (rf69.waitAvailableTimeout(500))  {
    // Should be a reply message for us now
```

```
    if (rf69.recv(buf, &len)) {
      Serial.print("Got a reply: ");
      Serial.println((char*)buf);
      Blink(LED, 50, 3); //blink LED 3 times, 50ms between blinks
    } else {
      Serial.println("Receive failed");
    }
  } else {
    Serial.println("No reply, is another RFM69 listening?");
  }
}
```

Its pretty simple, the delay does the waiting, you can replace that with low power sleep code. Then it generates the packet and appends a number that increases every tx. Then it simply calls send() waitPacketSent() to wait until is is done transmitting.

It will then wait up to 500 milliseconds for a reply from the receiver with waitAvailableTimeout(500). If there is a reply, it will print it out. If not, it will complain nothing was received. Either way the transmitter will continue the loop and sleep for a second until the next TX.

Basic Receiver Code

The Receiver has the same exact setup code, but the loop is different

```
void loop() {
 if (rf69.available()) {
    // Should be a message for us now
    uint8_t buf[RH_RF69_MAX_MESSAGE_LEN];
    uint8_t len = sizeof(buf);
    if (rf69.recv(buf, &len)) {
      if (!len) return;
      buf[len] = 0;
      Serial.print("Received [");
      Serial.print(len);
      Serial.print("]: ");
      Serial.println((char*)buf);
      Serial.print("RSSI: ");
      Serial.println(rf69.lastRssi(), DEC);

      if (strstr((char *)buf, "Hello World")) {
        // Send a reply!
        uint8_t data[] = "And hello back to you";
        rf69.send(data, sizeof(data));
        rf69.waitPacketSent();
        Serial.println("Sent a reply");
        Blink(LED, 40, 3); //blink LED 3 times, 40ms between blinks
      }
    } else {
```

```
      Serial.println("Receive failed");
    }
  }
}
```

Instead of transmitting, it is constantly checking if there's any data packets that have been received. available() will return true if a packet with the proper encryption has been received. If so, the receiver prints it out.

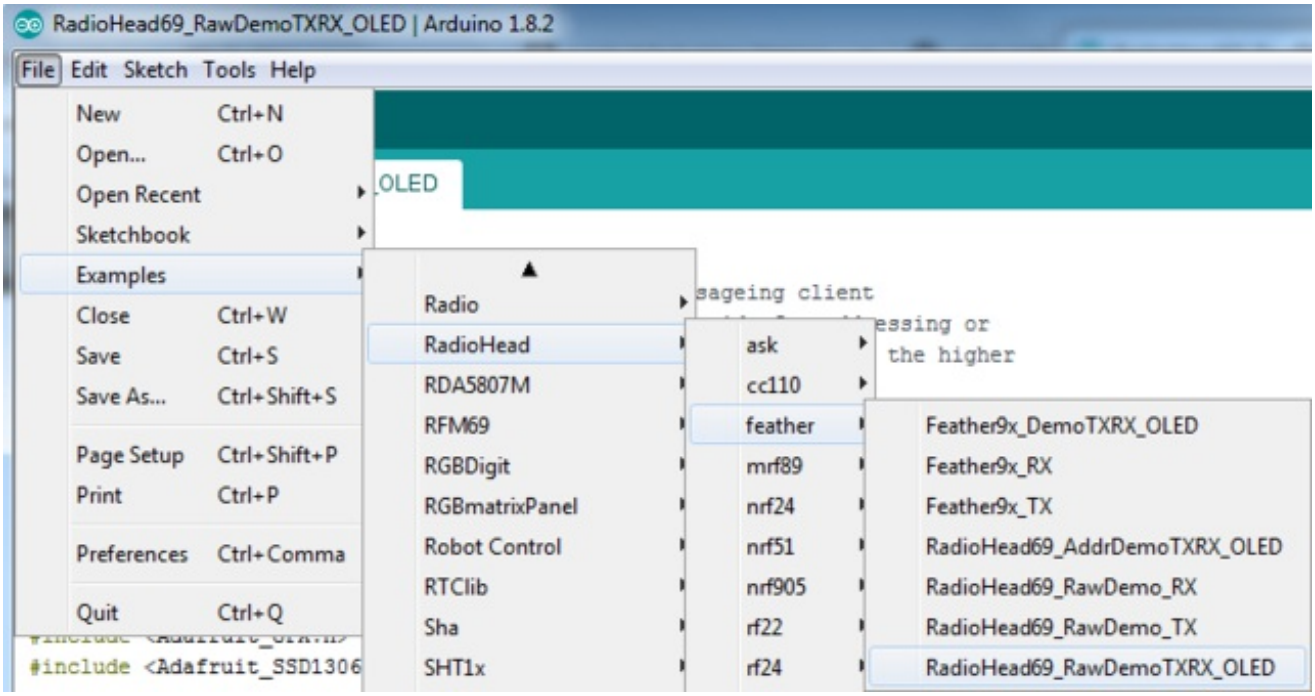
It also prints out the RSSI which is the receiver signal strength indicator. This number will range from about -15 to -80. The larger the number (-15 being the highest you'll likely see) the stronger the signal.

If the data contains the text "Hello World" it will also reply to the packet.

Once done it will continue waiting for a new packet

Basic Receiver/Transmitter Demo w/OLED

OK once you have that going you can try this example, **RadioHead69_RawDemoTXRX_OLED**. We're using the Feather with an OLED wing but in theory you can run the code without the OLED and connect three buttons to GPIO #9, 6, and 5 on the Feathers. Upload the same code to each Feather. When you press buttons on one Feather they will be printed out on the other one, and vice versa. Very handy for testing bi-directional communication!



This demo code shows how you can listen for packets and also check for button presses (or sensor data or whatever you like) and send them back and forth between the two radios!

Addressed RX and TX Demo

OK so the basic demo is well and good but you have to do a lot of*management* of the connection to make sure packets were received. Instead of manually sending acknowledgements, you can have the RFM69 and library do it for you! Thus the **Reliable Datagram** part of the **RadioHead** library.

Load up the **RadioHead69_AddrDemo_RX** and **RadioHead69_AddrDemo_TX** sketches to each of your boards

Don't forget to check the frequency set in the example, and that the pinouts match your wiring!!!

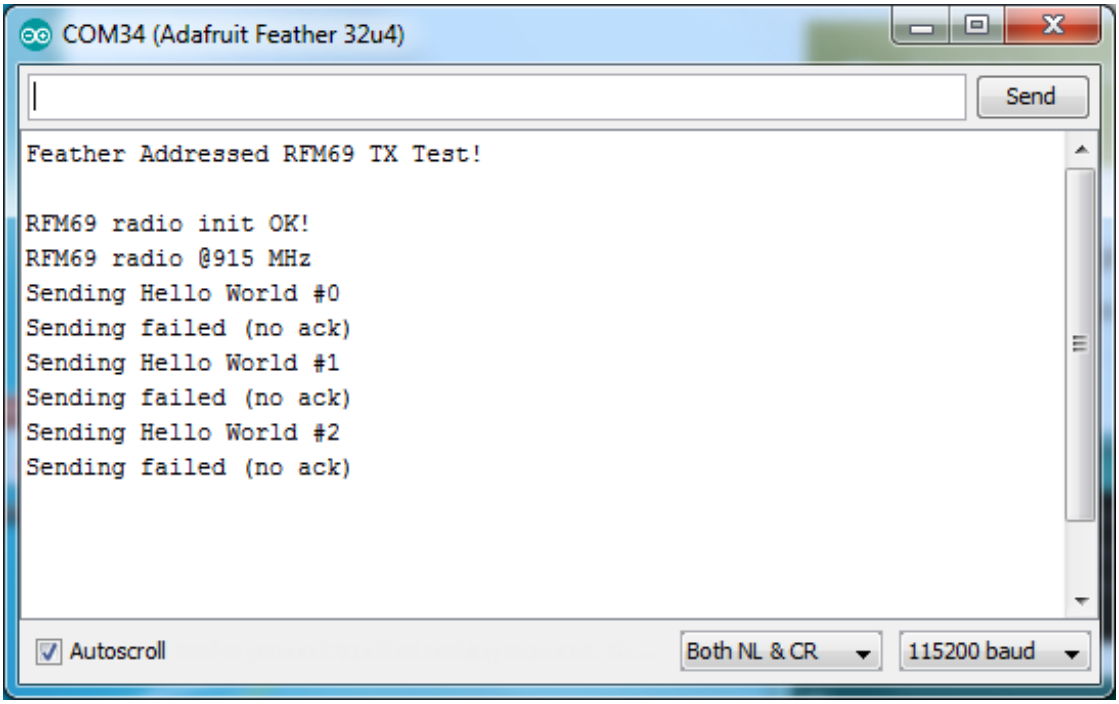
This example lets you have many 'client' RFM69's all sending data to one 'server'

Each client can have its own address set, as well as the server address. See this code at the beginning:

```
// Where to send packets to!
#define DEST_ADDRESS   1
// change addresses for each client board, any number :)
#define MY_ADDRESS     2
```

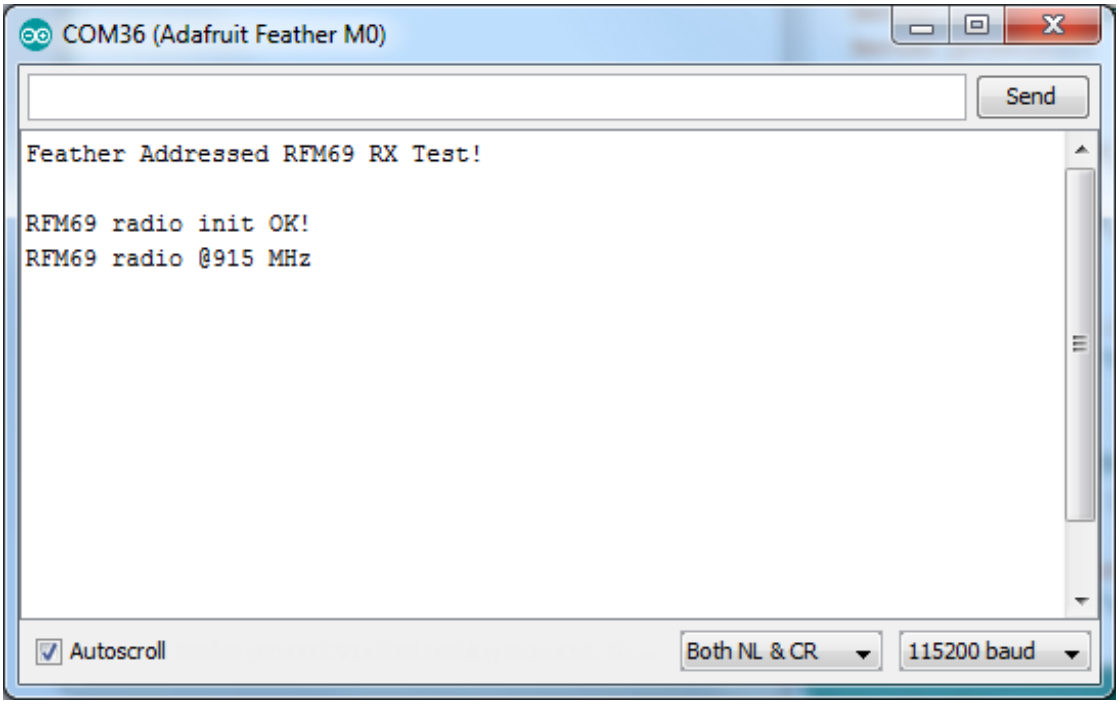
For each client, have a unique **MY_ADDRESS**. Then pick one server that will be address #1

Once you upload the code to a client, you'll see the following in the serial console:

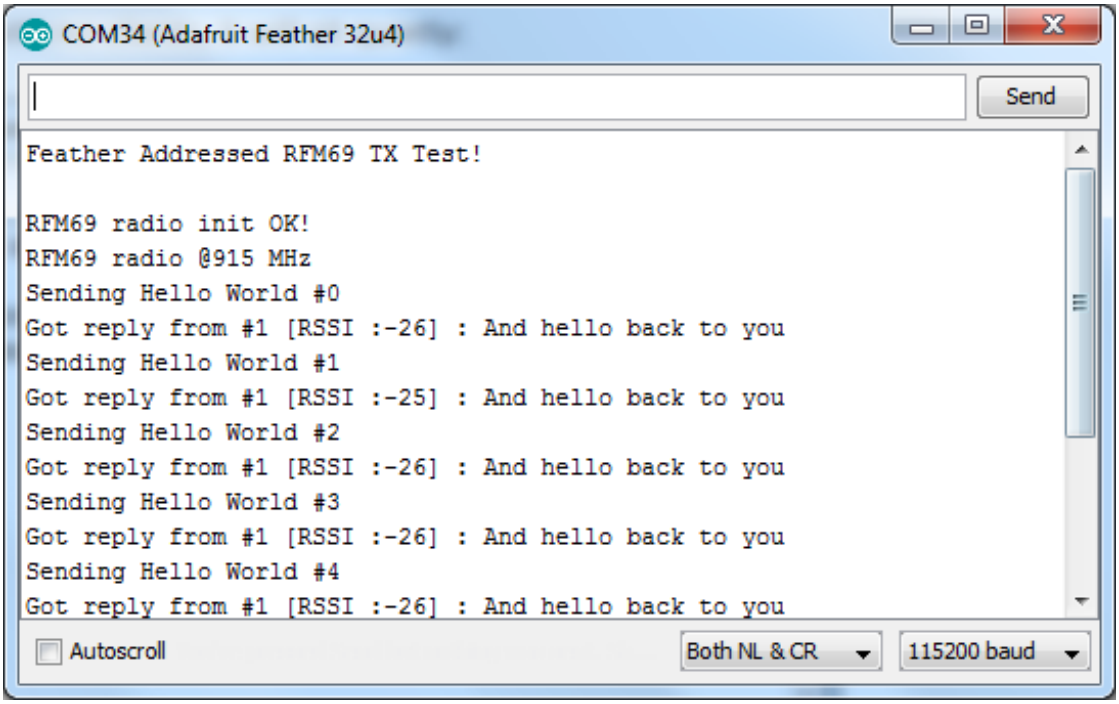


Because the data is being sent to address #1, but #1 is not acknowledging that data.

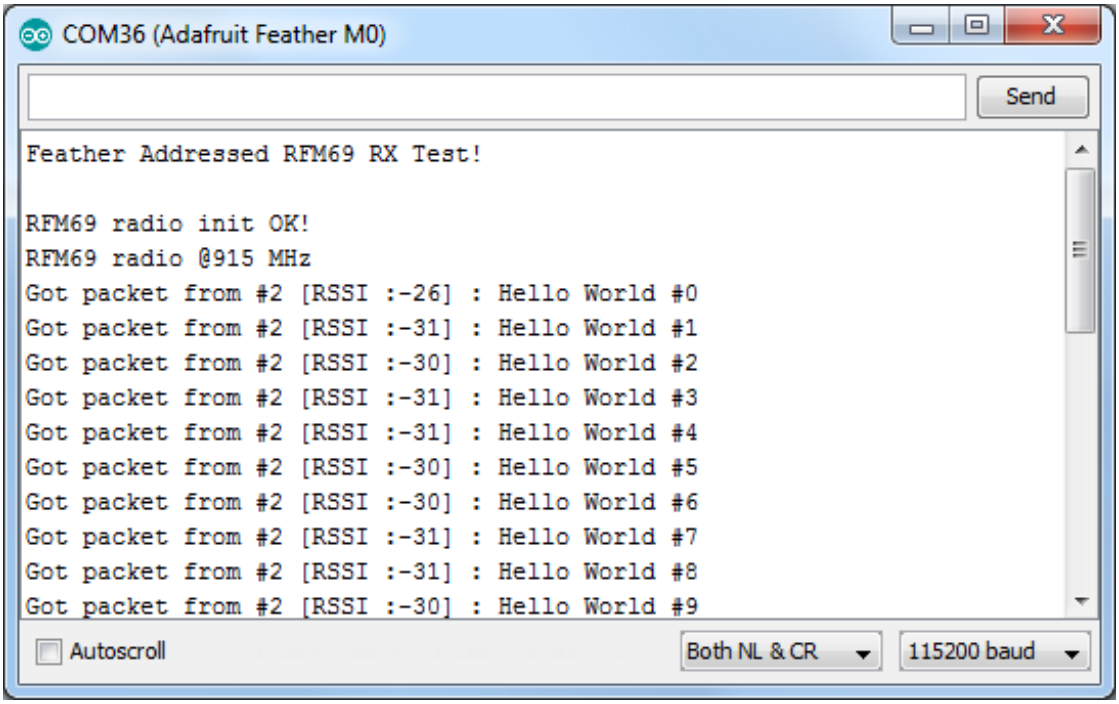
If you have the server running, with no clients, it will sit quietly:



Turn on the client and you'll see acknowledged packets!



And the server is also pretty happy



The secret sauce is the addition of this new object:

```
// Class to manage message delivery and receipt, using the driver declared above
RHReliableDatagram rf69_manager(rf69, MY_ADDRESS);
```

Which as you can see, is the manager for the RFM69. In **setup()** you'll need to init it, although you still configure the underlying rfm69 like before:

```
 if (!rf69_manager.init()) {
   Serial.println("RFM69 radio init failed");
   while (1);
 }
```

And when transmitting, use **sendToWait** which will wait for an ack from the recepient (at DEST_ADDRESS)

```
 if (rf69_manager.sendtoWait((uint8_t *)radiopacket, strlen(radiopacket), DEST_ADDRESS)) {
```

on the 'other side' use the **recvFromAck** which will receive and acknowledge a packet

```
   // Wait for a message addressed to us from the client
   uint8_t len = sizeof(buf);
   uint8_t from;
   if (rf69_manager.recvfromAck(buf, &len, &from)) {
```

That function will wait forever. If you'd like to timeout while waiting for a packet, use **recvfromAckTimeout** which will wait an indicated # of milliseconds

```
if (rf69_manager.recvfromAckTimeout(buf, &len, 2000, &from))
```

Radio Range F.A.Q.

Which gives better range, LoRa or RFM69?

All other things being equal (antenna, power output, location) you will get better range with LoRa than with RFM69 modules. We've found 50% to 100% range improvement is common.

What ranges can I expect for RFM69 radios?

The RFM69 radios have a range of approx. 500 meters **line of sight** with tuned uni-directional antennas. Depending on obstructions, frequency, antenna and power output, you will get lower ranges - *especially* if you are not line of sight.

What ranges can I expect for RFM9X LoRa radios?

The RFM9x radios have a range of up to 2 km **line of sight** with tuned uni-directional antennas. Depending on obstructions, frequency, antenna and power output, you will get lower ranges - *especially* if you are not line of sight.

I don't seem to be getting the range advetised! Is my module broken?

Your module is probably *not* broken. Radio range is dependant on *a lot of things* and all must be attended to to make sure you get the best performance!

1. Tuned antenna for your frequency - getting a well tuned antenna is incredibly important. Your antenna must be tuned for the exact frequency you are using
2. Matching frequency - make sure all modules are on the same exact frequency
3. Matching settings - all radios must have the same settings so they can communicate
4. Directional vs non-directional antennas - for the best range, *directional* antennas like Yagi will direct your energy in one path instead of all around
5. Good power supply - a nice steady power supply will keep your transmissions clean and strong
6. Max power settings on the radios - they can be set for higher/lower power! Don't forget to set them to max.
7. Line of sight - No obstructions, walls, trees, towers, buildings, mountains, etc can be in the way of your radio path. Likewise, outdoors is way better than indoors because its very hard to bounce radio paths around a building
8. Radio transmission speed - trying to transmit more data faster will be hard. Go for small packets, with lots of retransmissions. Lowering the baud rate on the radio (see the libraries for how to do this) will give you better reliability

How do I pick/design the right antenna?

Various antennas will cost diferent amounts and give you different directional gain. In general, spending a lot on a large fixed antenna can give you better power transfer if the antenna is well tuned. For most simple uses, a wire works pretty well

The ARRL antena book is recommended if you want to learn how to do the modeling and analysis (http://adafru.it/sdN)

But nothing beats actual tests in your environment!

Downloads

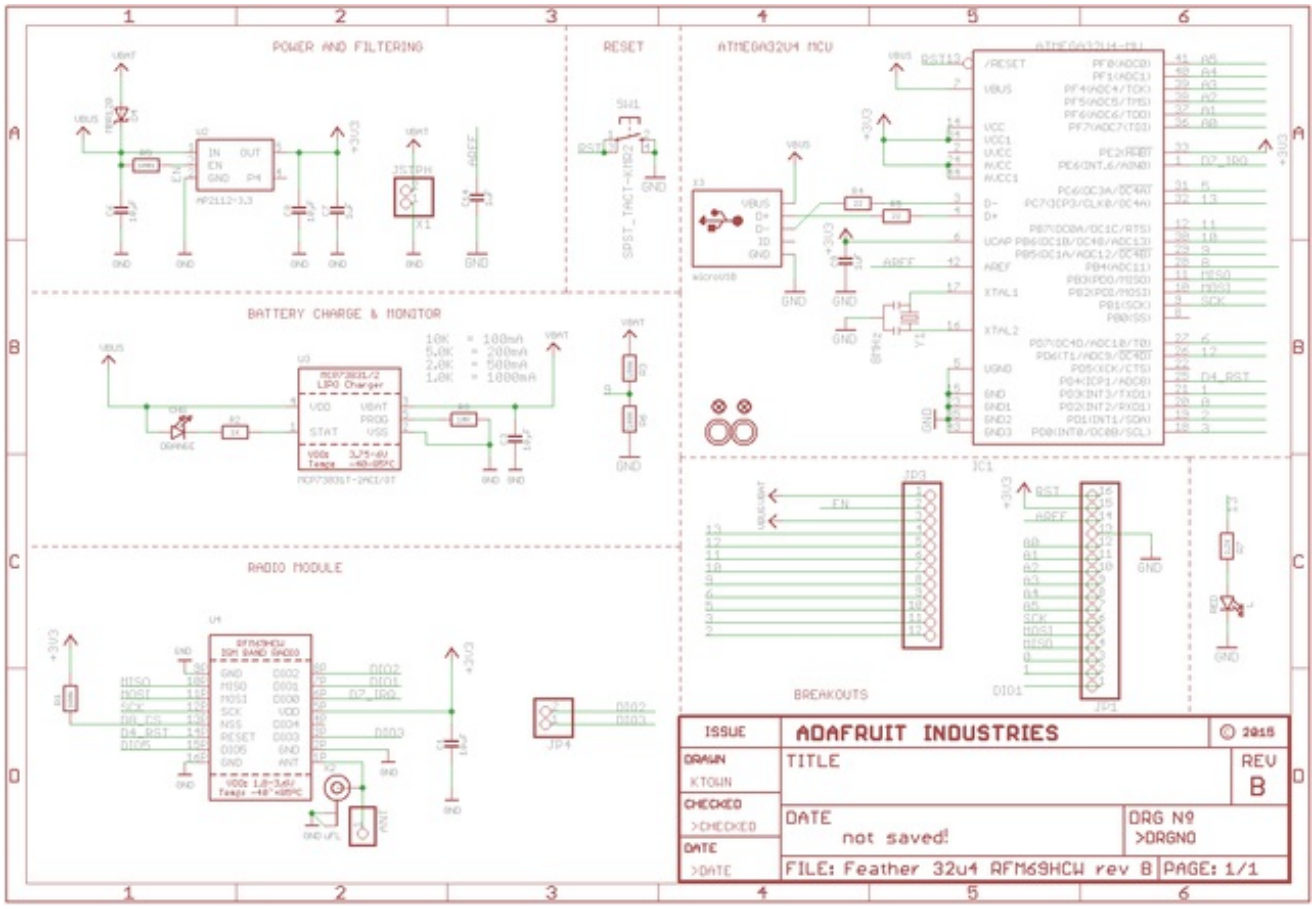
Datasheets & Files

- FCC Test Report (http://adafru.it/oC0)
- RoHS Test Report (http://adafru.it/oC1)
- RoHS Test Report (http://adafru.it/oC2)
- REACH Test Report (http://adafru.it/oC3)
- RFM69HCW datasheet - contains the SX1231 datasheet plus details about the module (http://adafru.it/mCu)
- SX1231 Transceiver Datasheet (http://adafru.it/mCv)

- EagleCAD PCB Files on GitHub (http://adafru.it/obt)
- Fritzing object in Adafruit Fritzing Library (http://adafru.it/aP3)

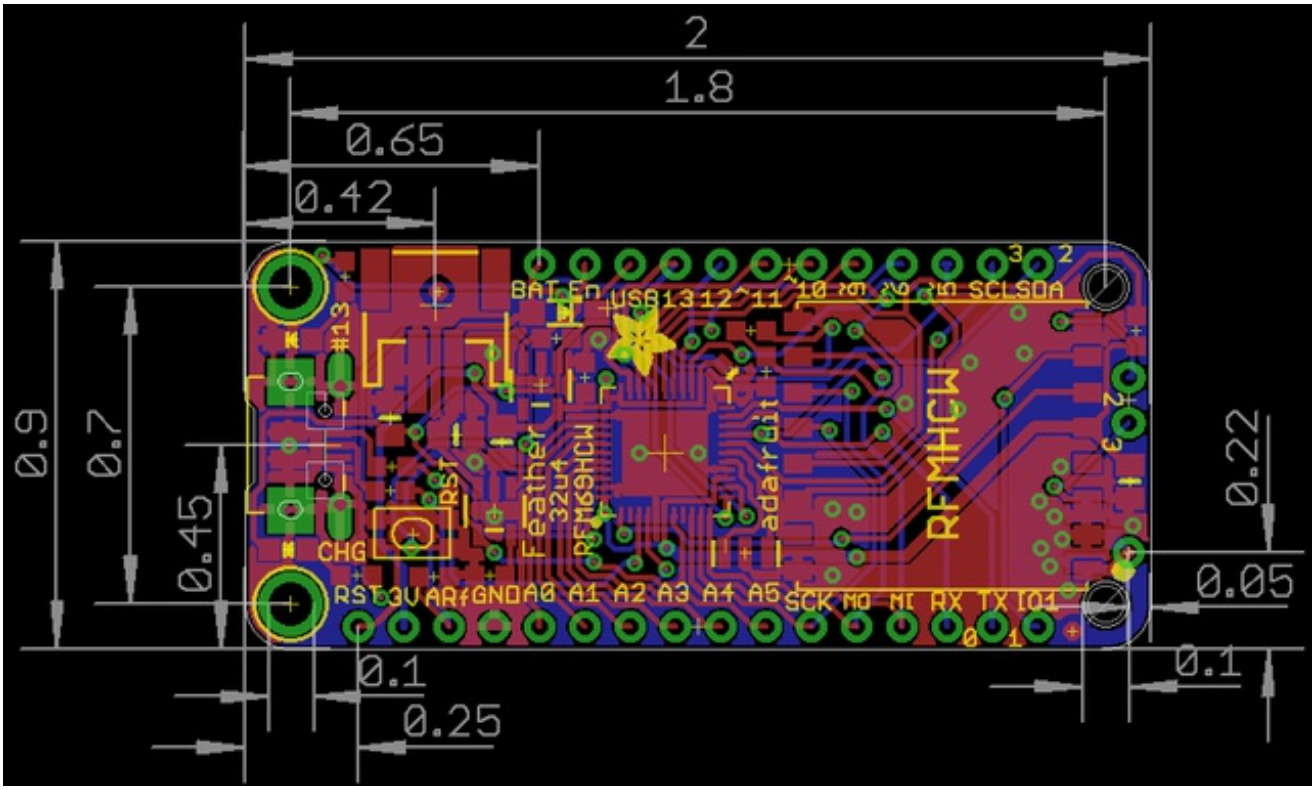
Feather 32u4 RFMxx Pinout Diagram
http://adafru.it/vWf

Schematic



Fabrication Print

Dimensions in Inches


2
1.8
0.65
0.42
0.9
0.7
0.45
0.22
0.05
0.1
0.25
0.1
BAT EN
USB
SCL SDA
CHG
RST
Feather
32u4
RFM69HCW
adafruit
RFMHCW
RST 3V AREF GND A0 A1 A2 A3 A4 A5
SCK MO MI RX TX IO1