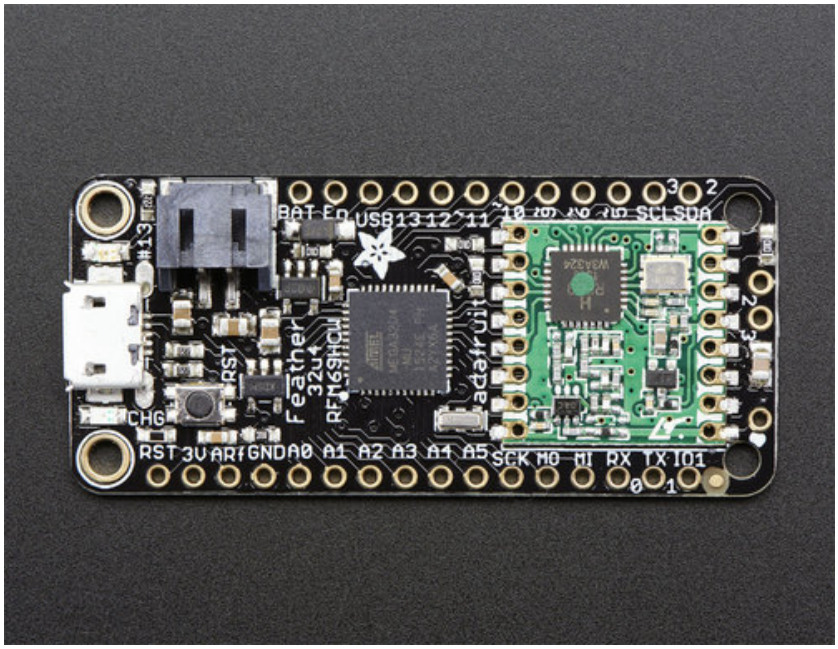
adafruit learning system

Adafruit Feather 32u4 Radio with RFM69HCW Module

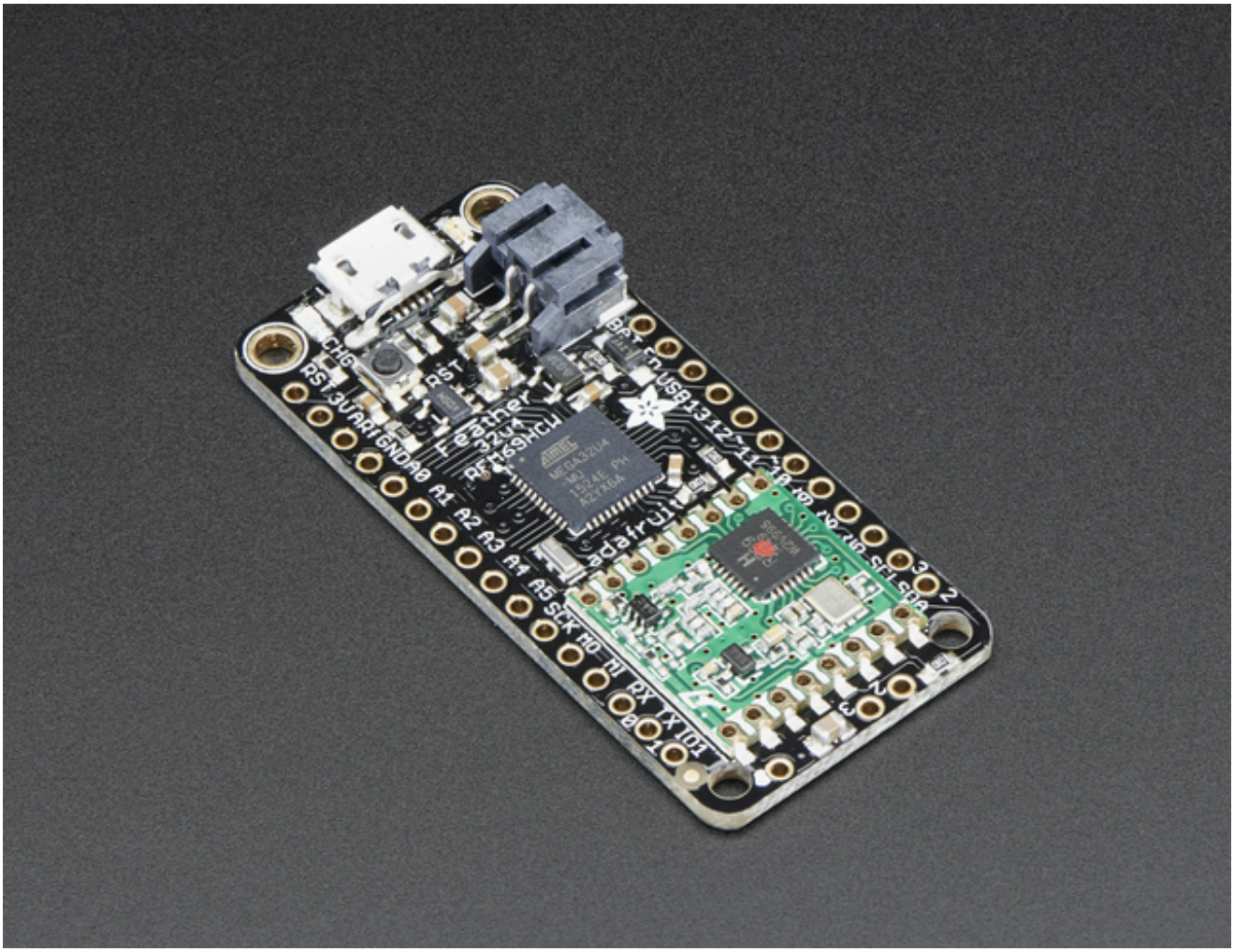
Created by lady ada



Last updated on 2016-04-15 05:22:07 PM EDT

Guide Contents

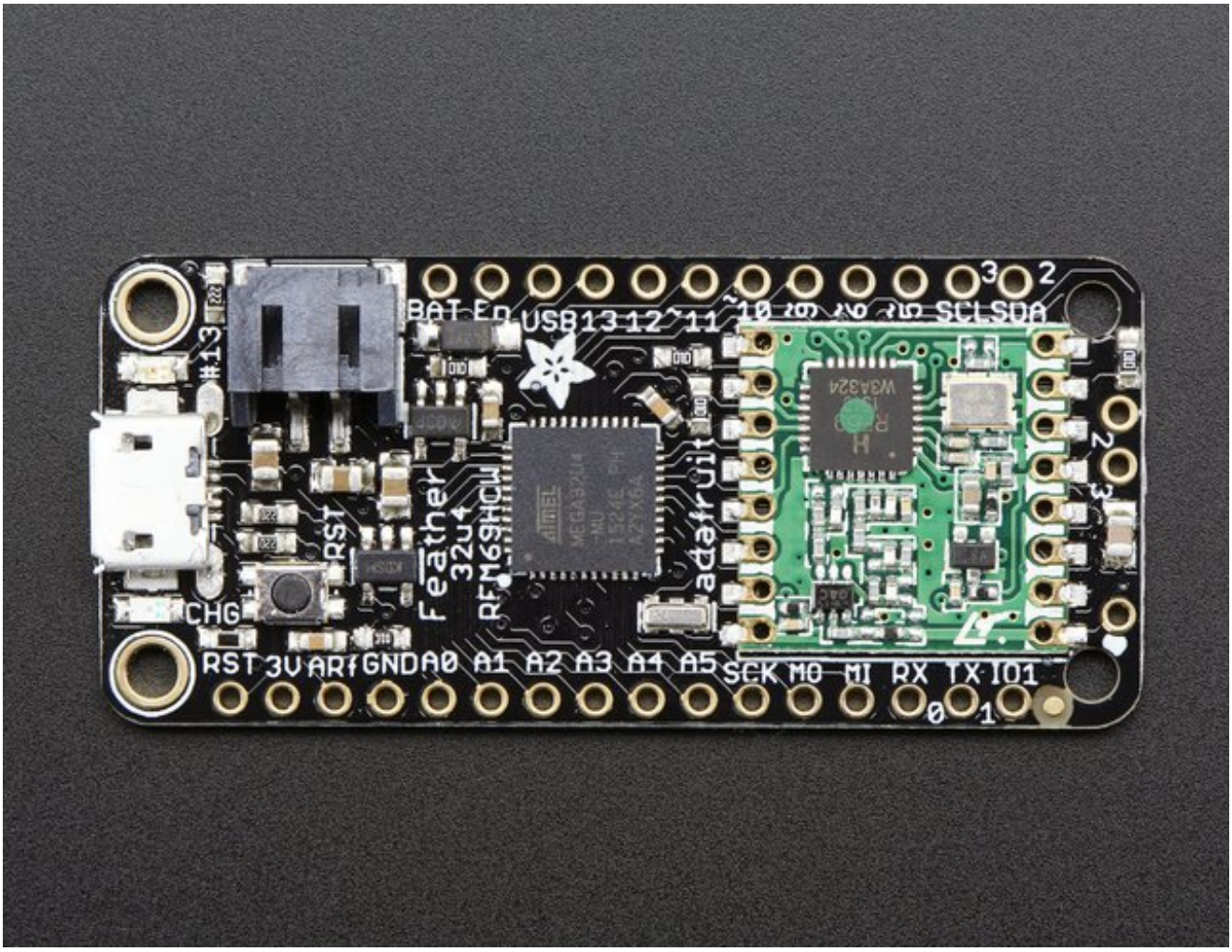
Overview



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.

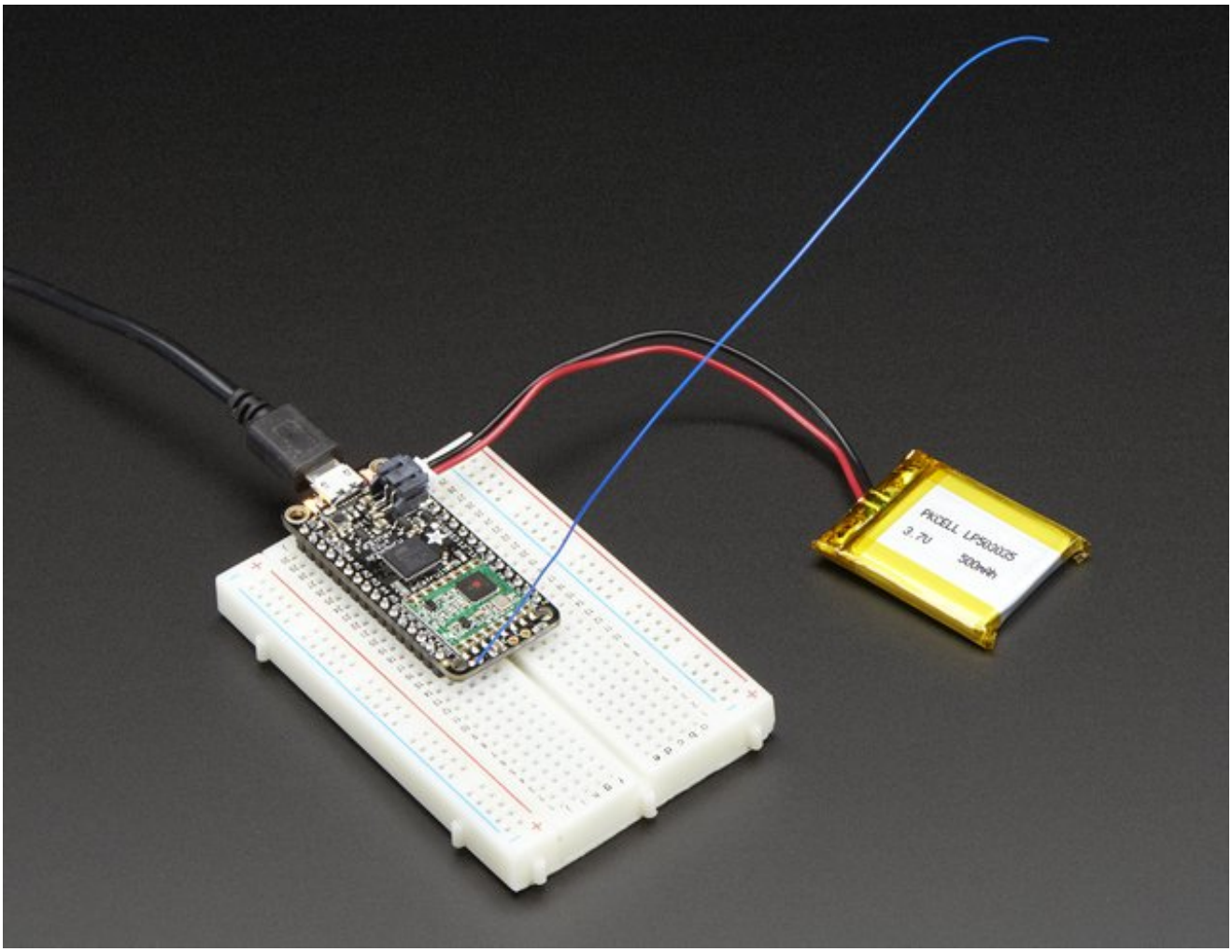


To make it easy to use for portable projects, we added a connector for any of our 3.7V Lithium polymer batteries and built in battery charging. You don't need a battery, it will run just fine straight from the micro USB connector. But, if you do have a battery, you can take it on the go, then plug in 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:

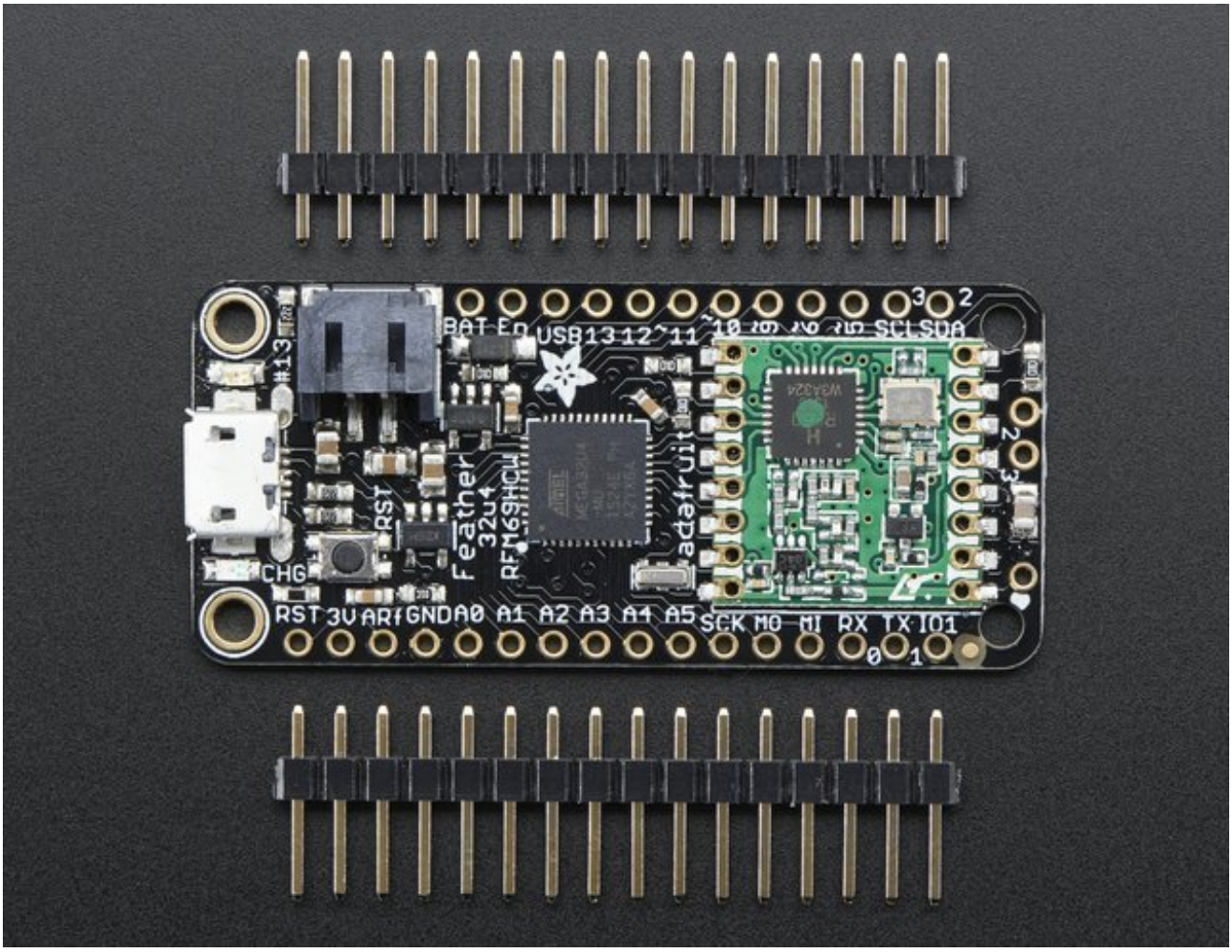
- Measures 2.0" x 0.9" x 0.28" (51mm x 23mm x 8mm) without headers soldered in
- Light as a (large?) feather - 5.5 grams
- ATmega32u4 @ 8MHz with 3.3V logic/power
- 3.3V regulator with 500mA peak current output
- USB native support, comes with USB bootloader and serial port debugging
- You also get tons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



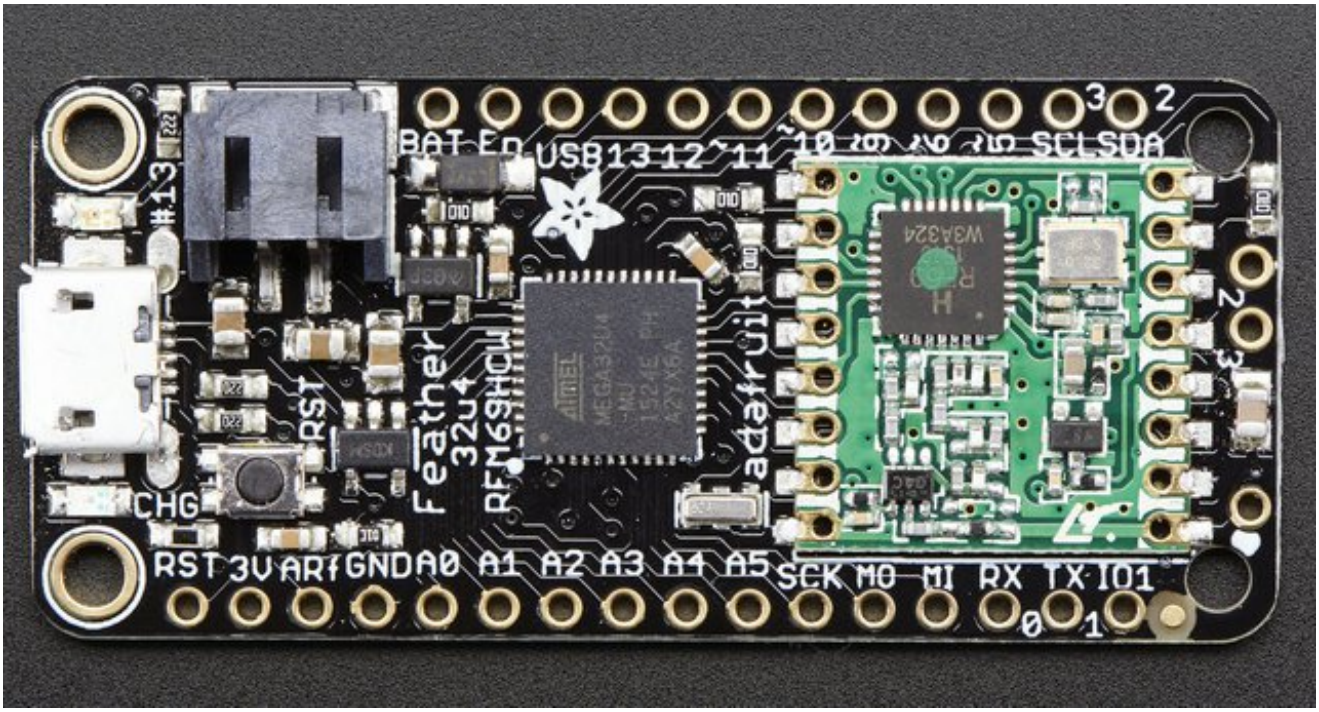
Comes fully assembled and tested, with a USB bootloader that lets you quickly use it with the Arduino IDE. We also toss in some header so you can solder it in and plug into a solderless breadboard. You will need to cut and solder on a small piece of wire (any solid or stranded core is fine) in order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

Pinouts

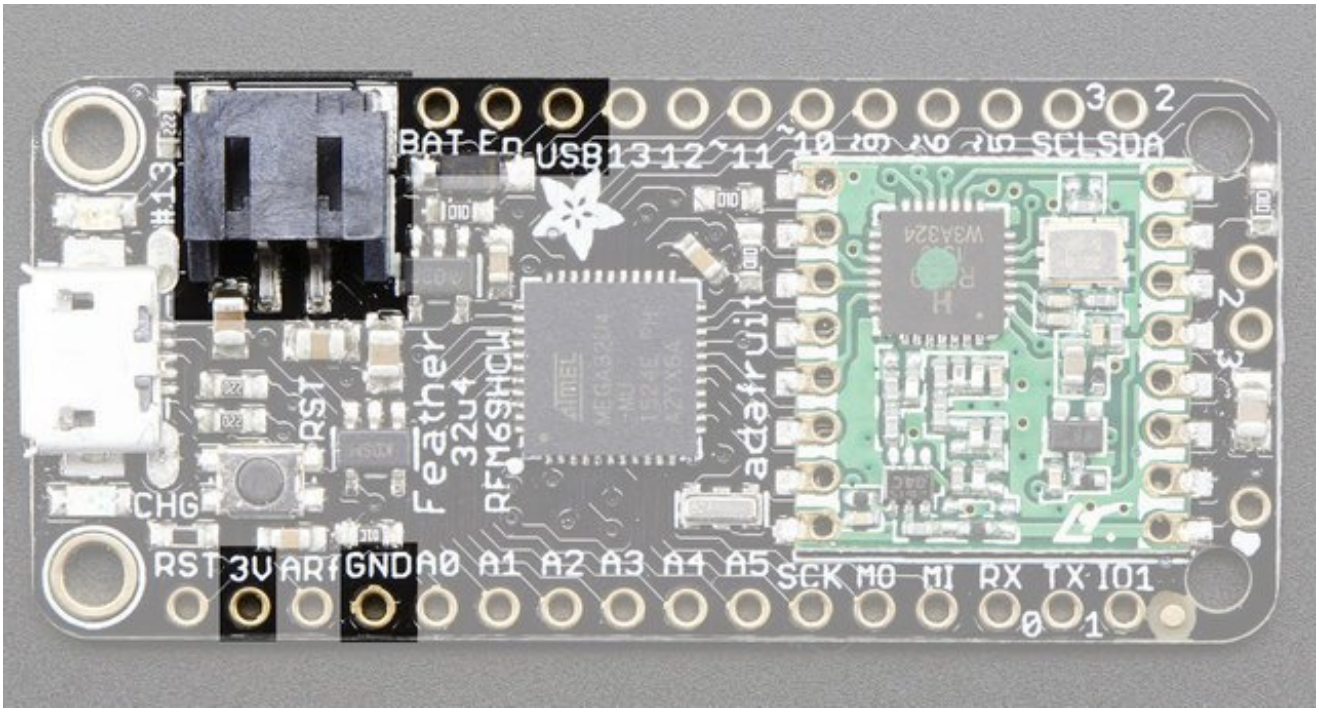
The Feather 32u4 Radio is chock-full of microcontroller goodness. There's also a lot of pins and ports. We'll take you a tour of them now!

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

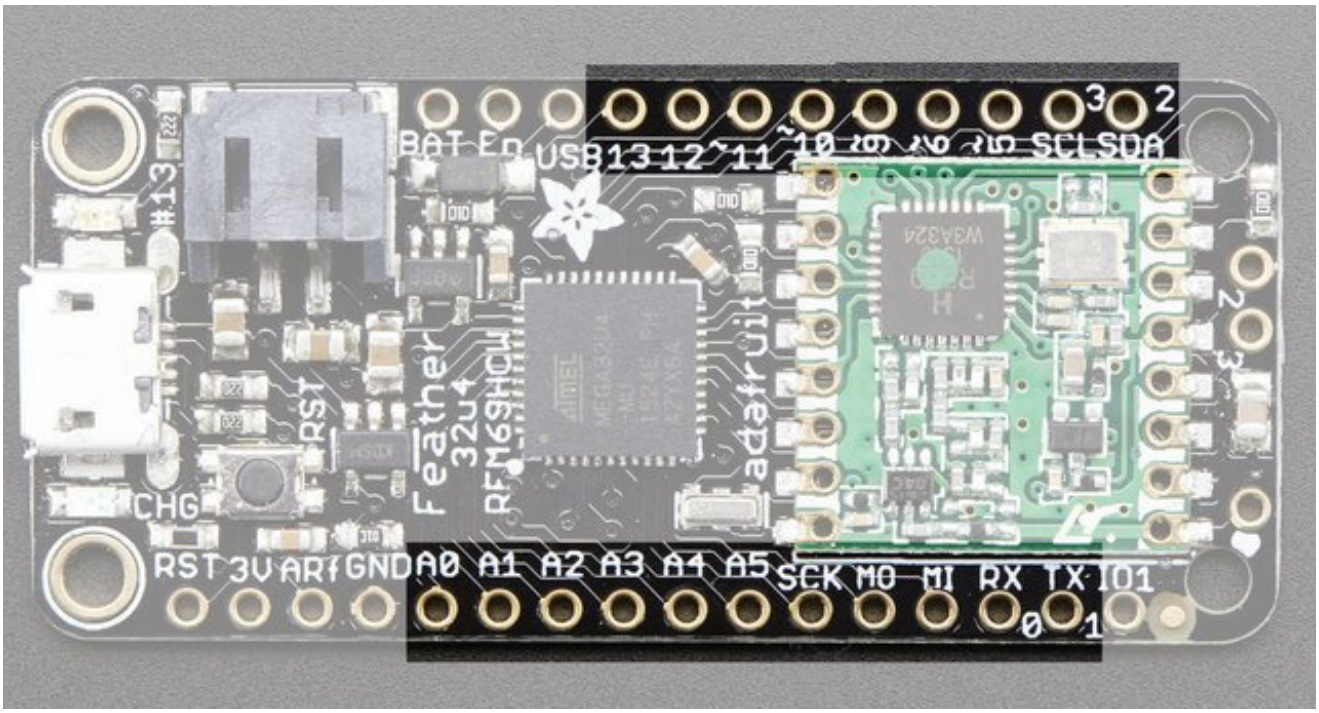


Power Pins



- **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

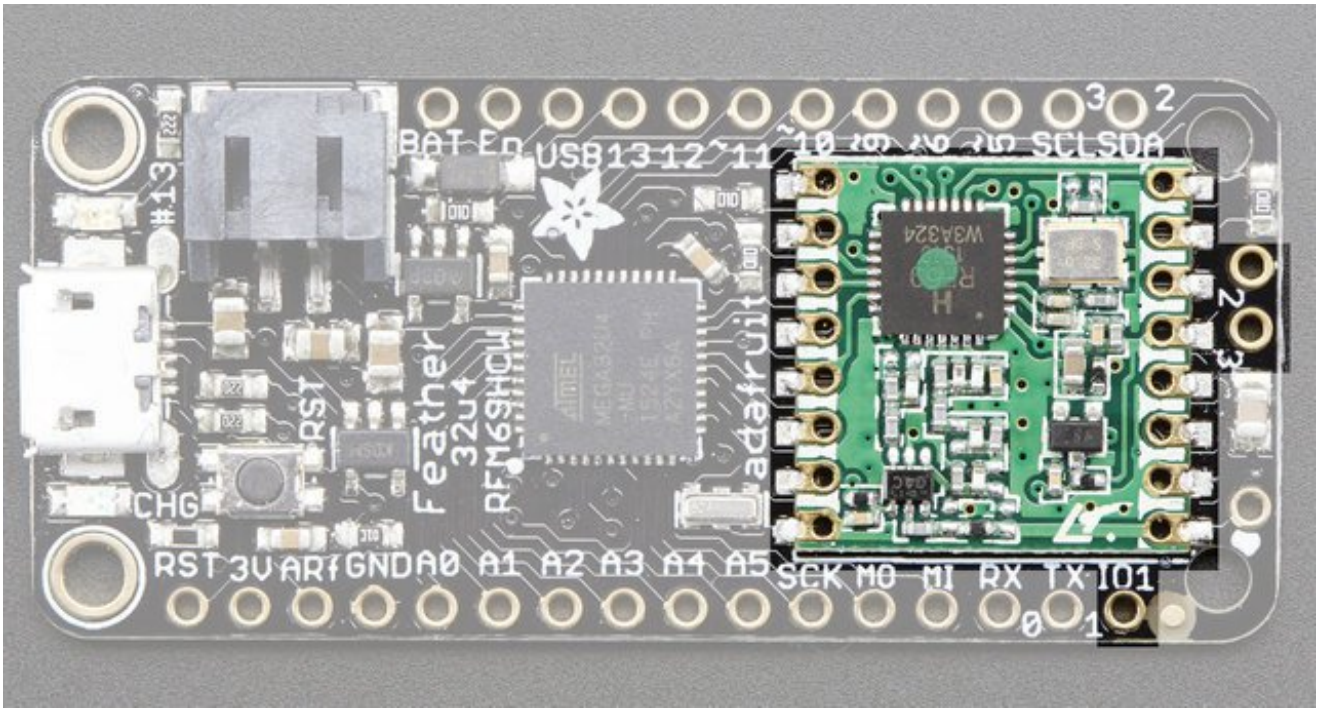
Logic pins



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

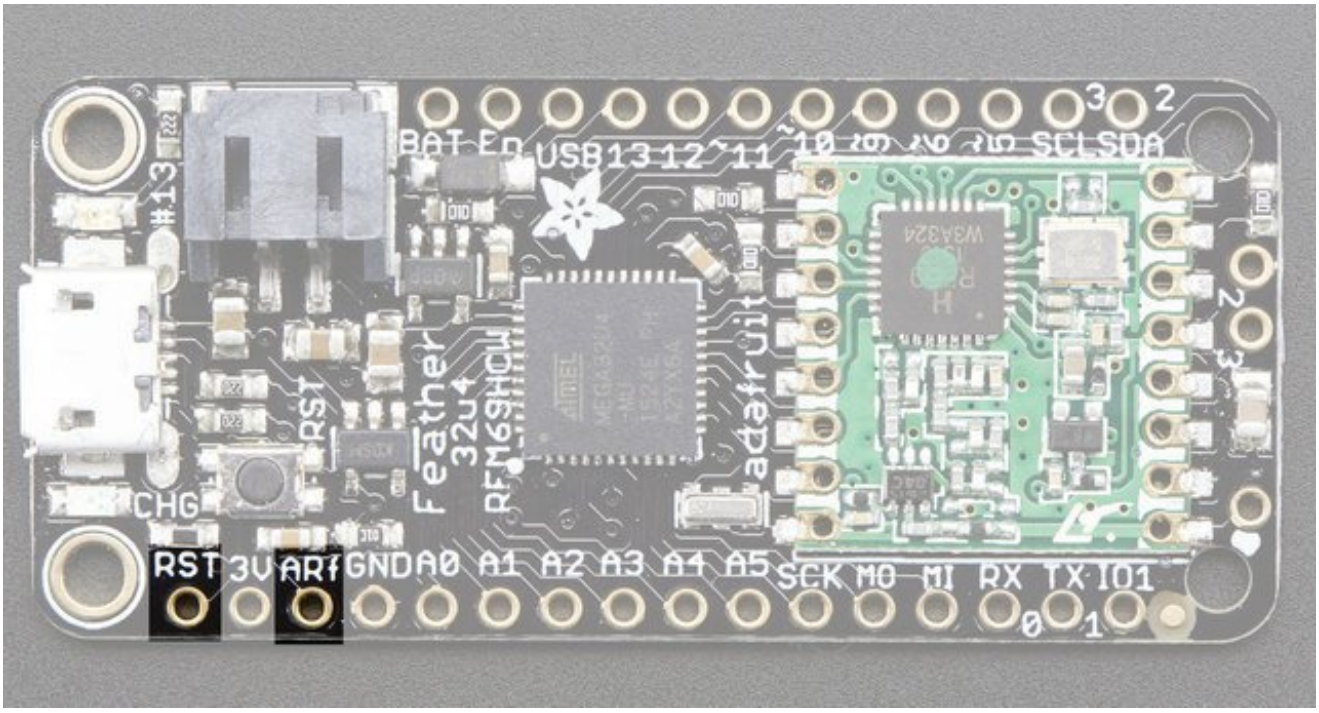
- **#8** - used as the radio **CS** (chip select) pin
- **#7** - used as the radio **GPIO0** / **IRQ** (interrupt request) pin.
- **#4** - used as the radio **Reset** pin

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!

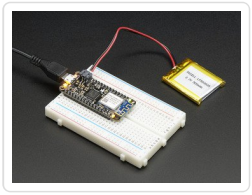
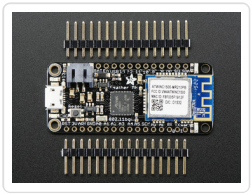
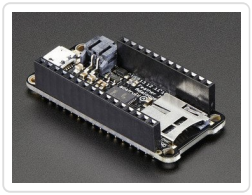
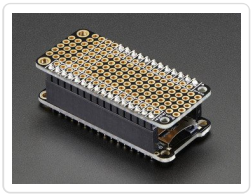

BAT EN USB 13 12 ~11 ~10 SCL SDA
#13
RST
CHG
Feather
32u4
RFM69HCW
adafruit
RST 3V AREF GND A0 A1 A2 A3 A4 A5 SCK MO MI RX TX IO1
0 1

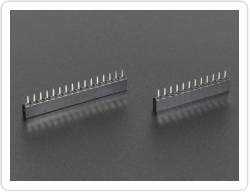
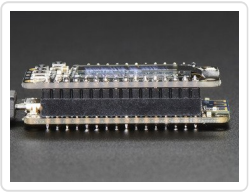
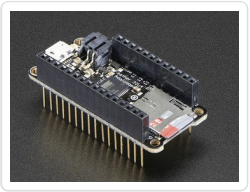
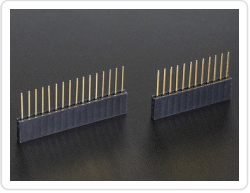
Assembly

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

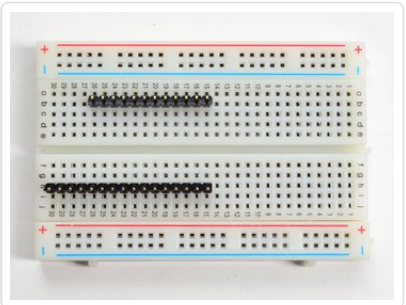
Header Options!

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
- 

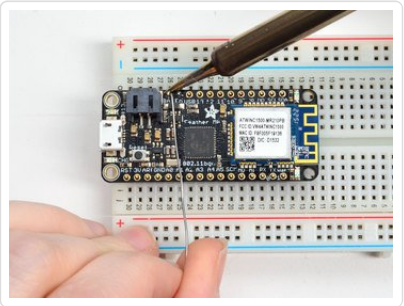
- 
  We also have 'slim' versions of the female headers, that are a little shorter and give a more compact shape
- 
- 
  Finally, there's the "Stacking Header" option. This one is sort of the best-of-both-worlds. You get the ability to plug into a solderless breadboard *and* plug a featherwing on top. But its a little bulky
- 

Soldering in Plain Headers

- 

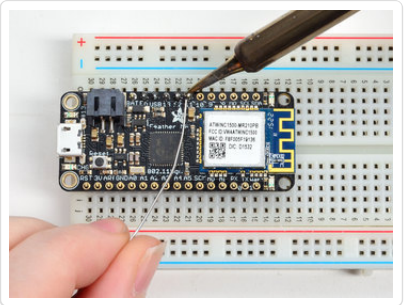
Prepare the header strip:

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:

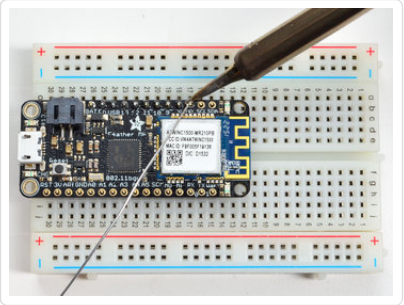
- Place the breakout board over the pins so that the short pins poke through the breakout pads

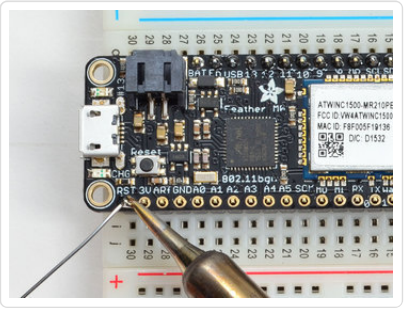
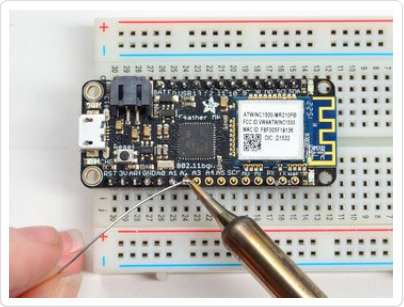
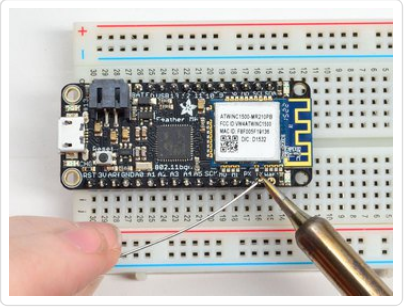
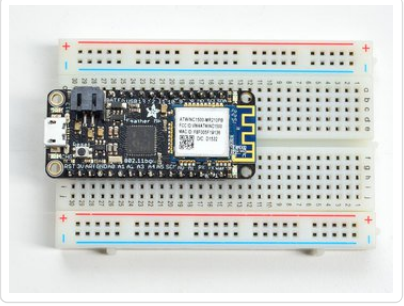


And Solder!

- 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).

- 

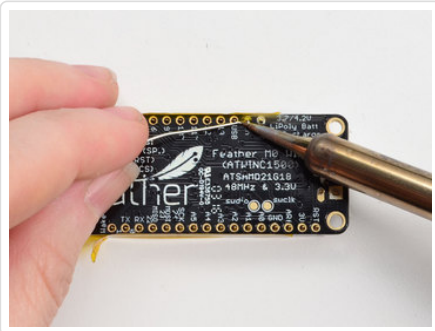
-  Solder the other strip as well.
- 
- 
-  You're done! Check your solder joints visually and continue onto the next steps

Soldering on Female Header

- 

Tape In Place

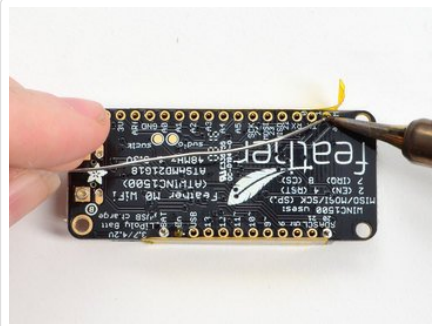
For sockets you'll want to tape them in place so when you flip over the board they don't fall out

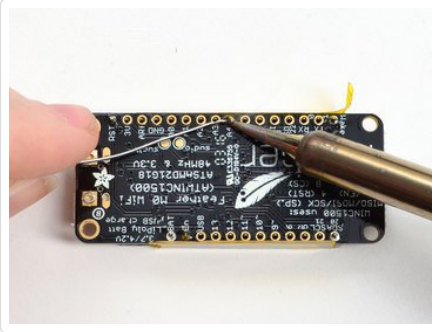
- 

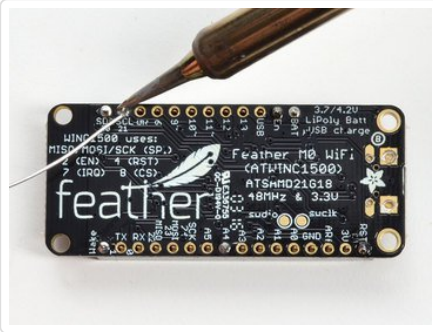
- 

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

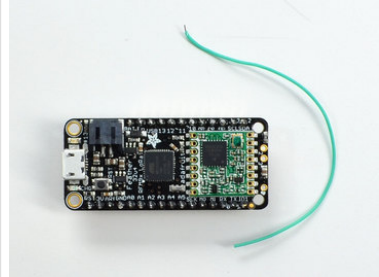
- 

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

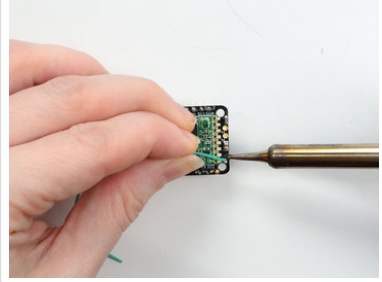
Wire Antenna

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.

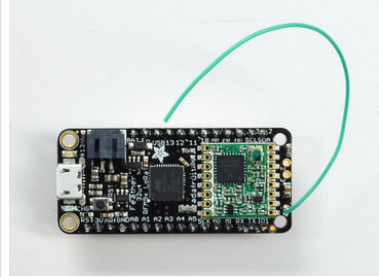
- 

  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

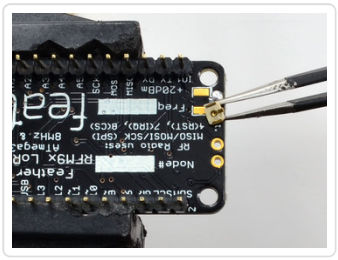
If you want an external antenna, you need to do a tiny bit more work but its not too difficult.

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

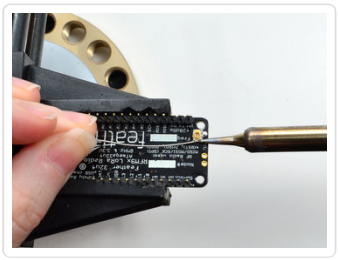
You'll also need a uFL to SMA adapter (http://adafru.it/851) (or whatever adapter you need for the antenna you'll be using, SMA is the most common

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

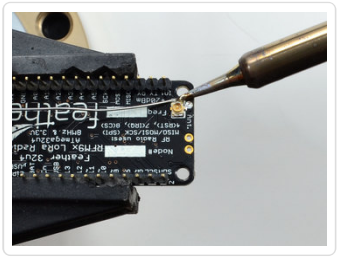
> uFL connectors are rated for 30 connection cycles, but be careful when connecting/disconnecting to not rip the pads off the PCB. Once a uFL/SMA adapter is connected, use strain relief!

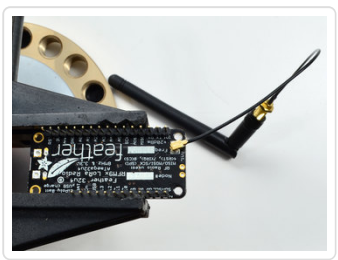
- 

  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!

- 

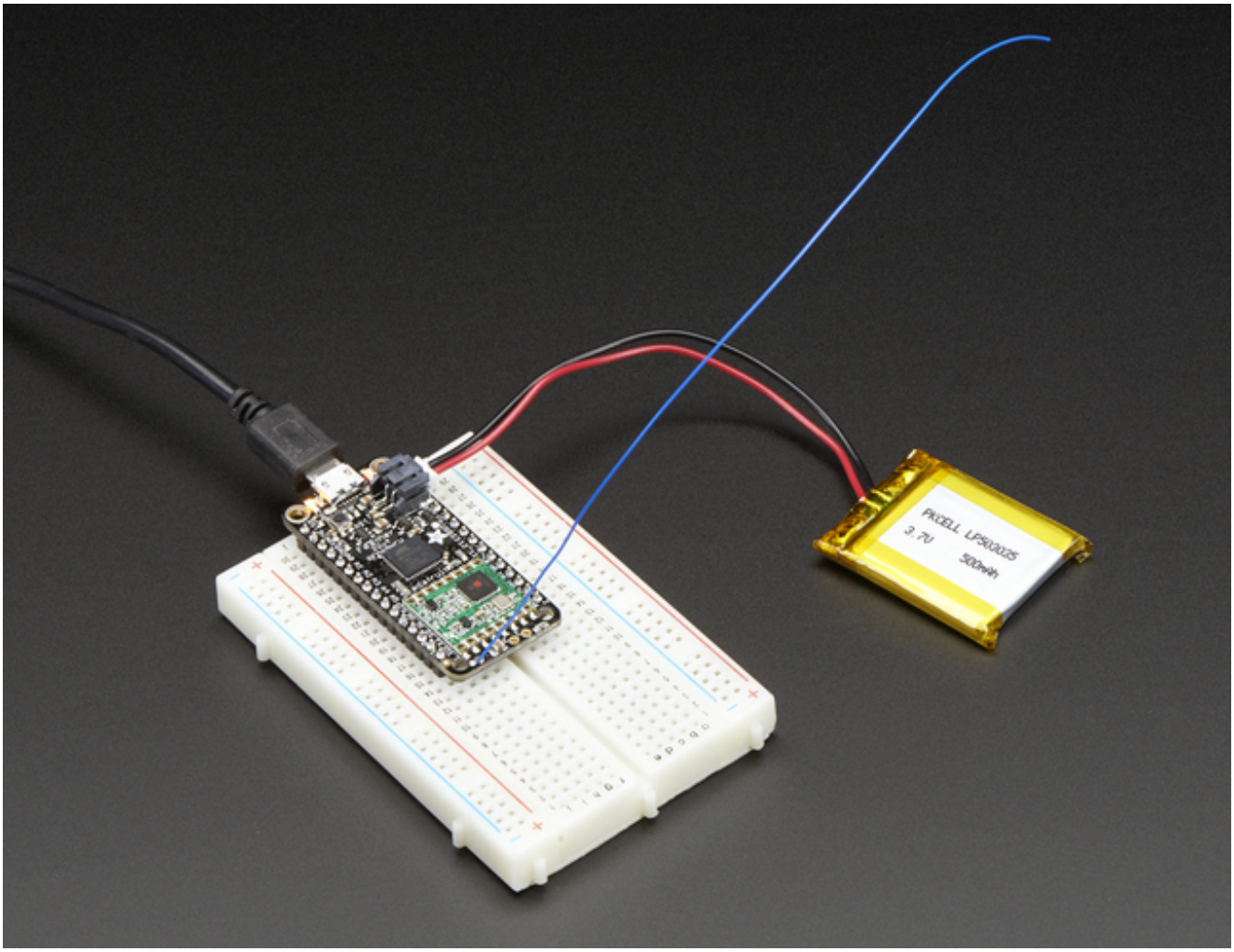
  Solder all three pads to the bottom of the Feather

- 

- 

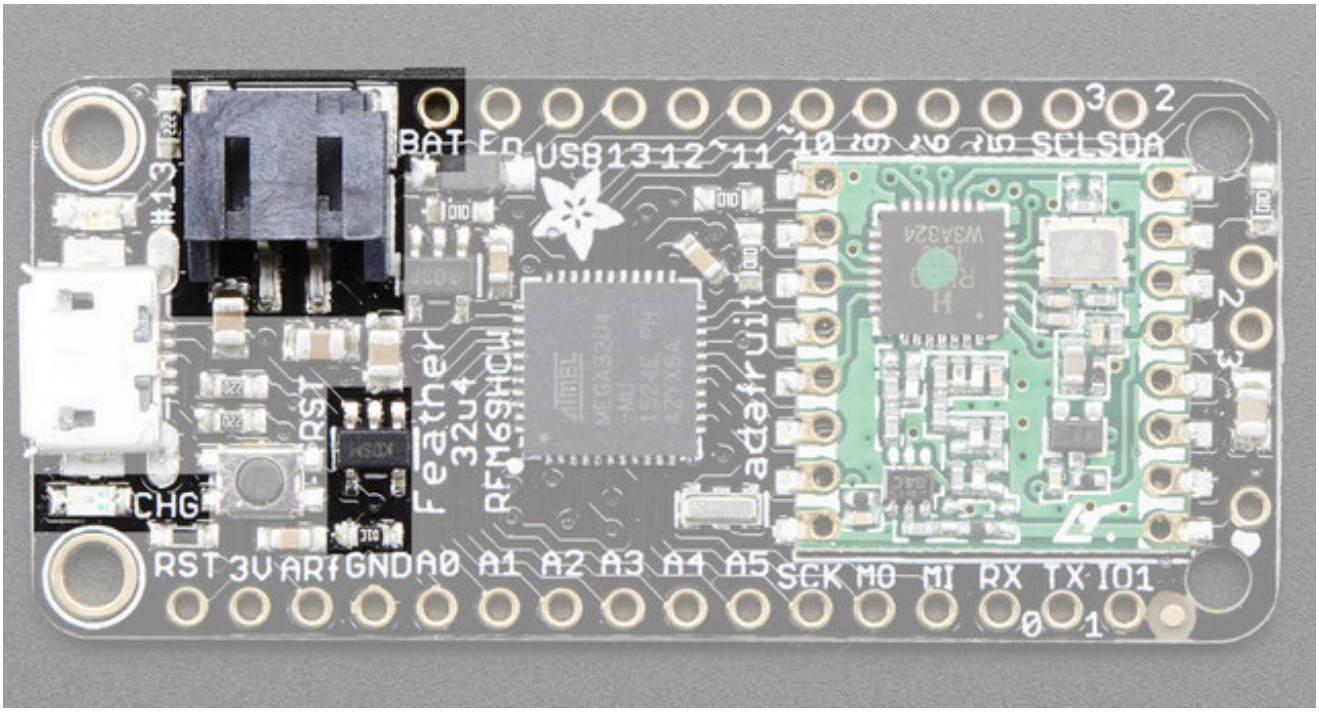
  Once done attach your uFL adapter and antenna!

Power Management



Battery + USB Power

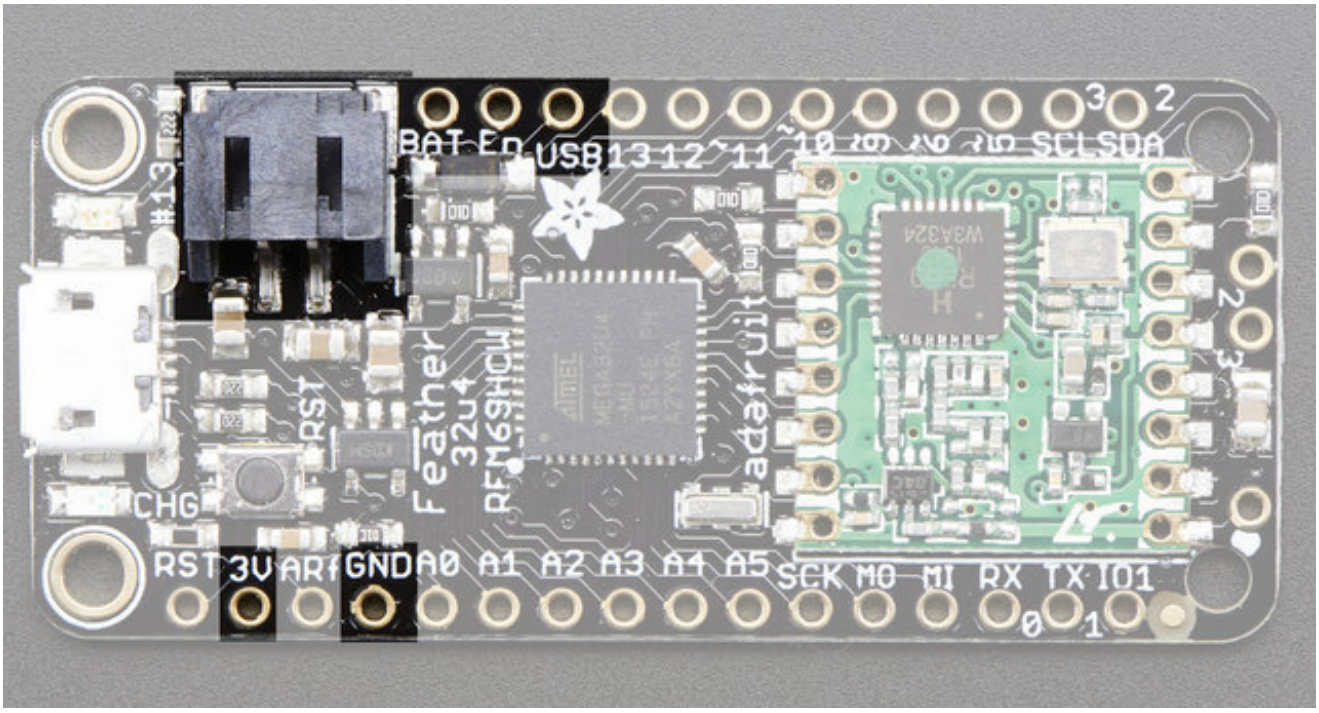
We wanted to make the Feather easy to power both when connected to a computer as well as via battery. There's **two ways to power** a Feather. You can connect with a MicroUSB cable (just plug into the jack) and the Feather will regulate the 5V USB down to 3.3V. You can also connect a 4.2/3.7V Lithium 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 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.

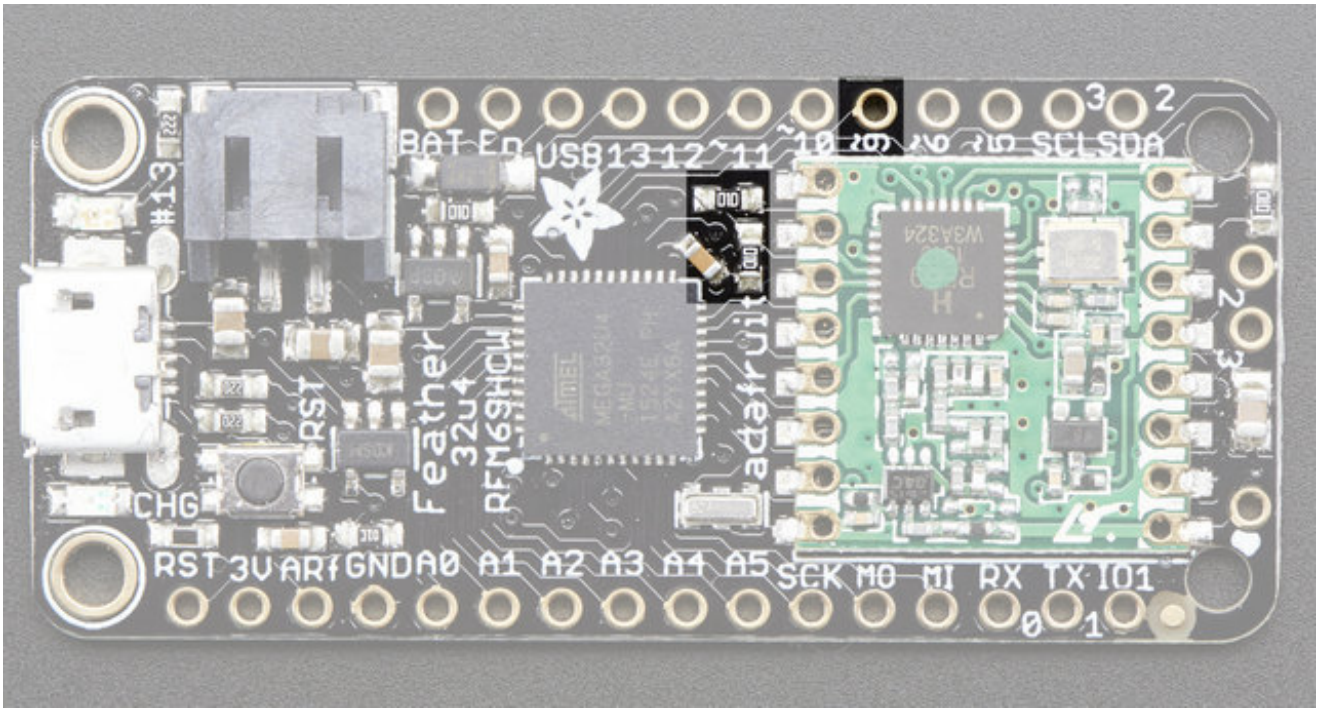


Measuring Battery

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);
```

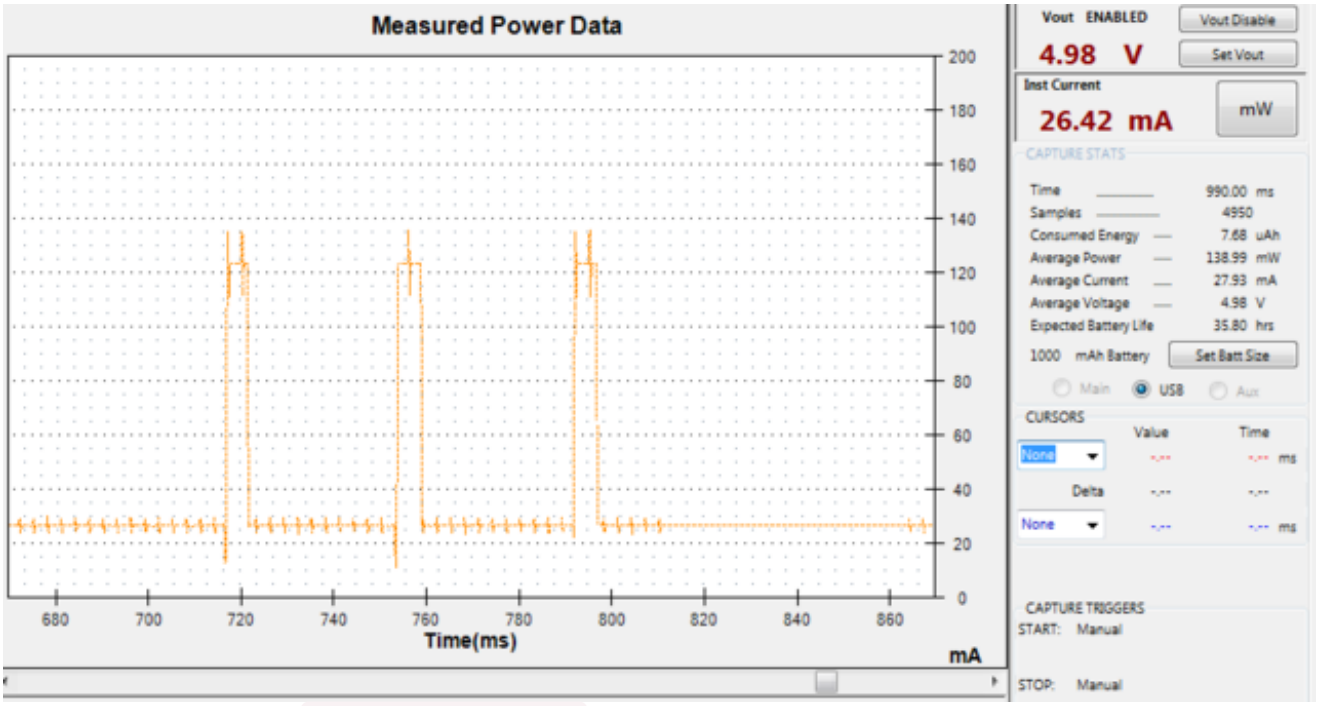


Radio Power Draw

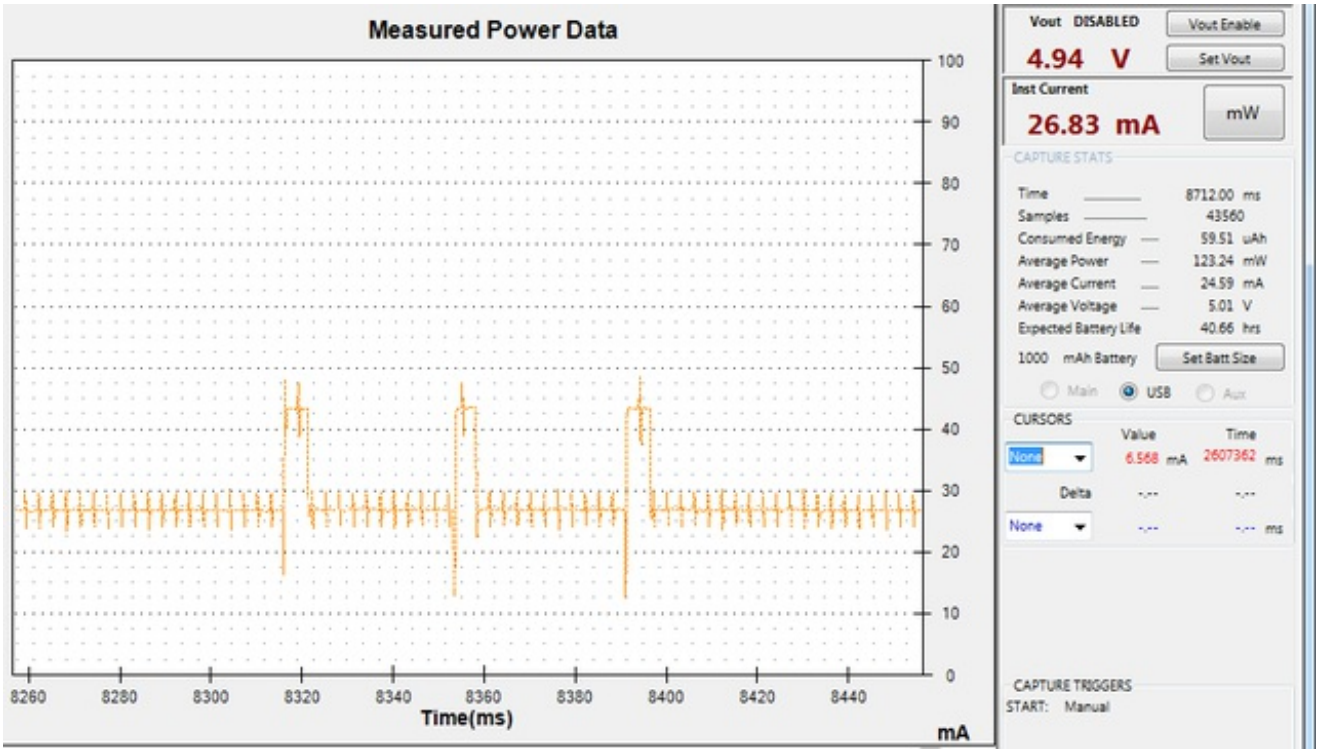
You can select the power output you want via software, more power equals more range but of course, uses more of your battery.

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!

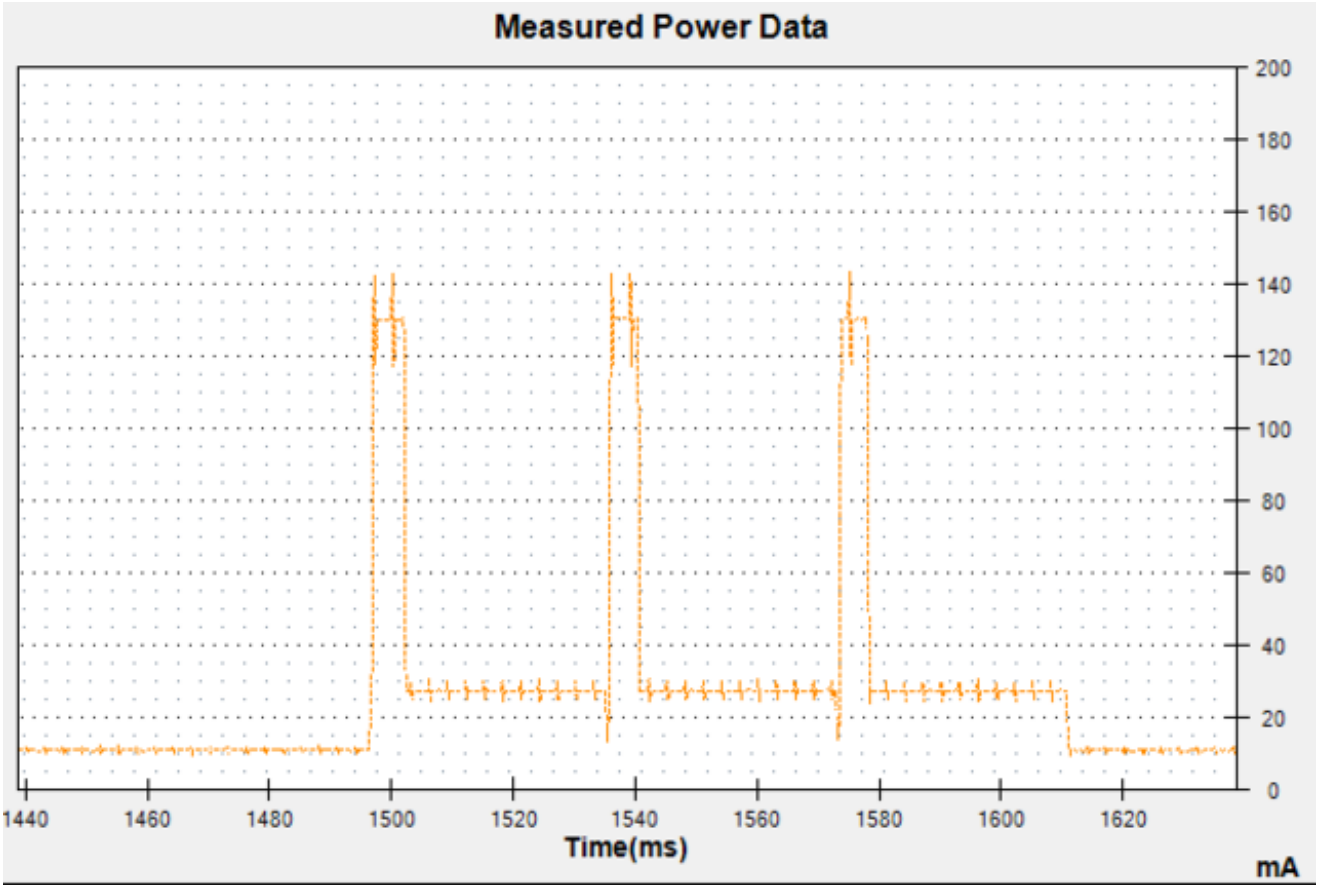


Here is a transmit with `radio.setPowerLevel(0)` to set +5dBm power



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



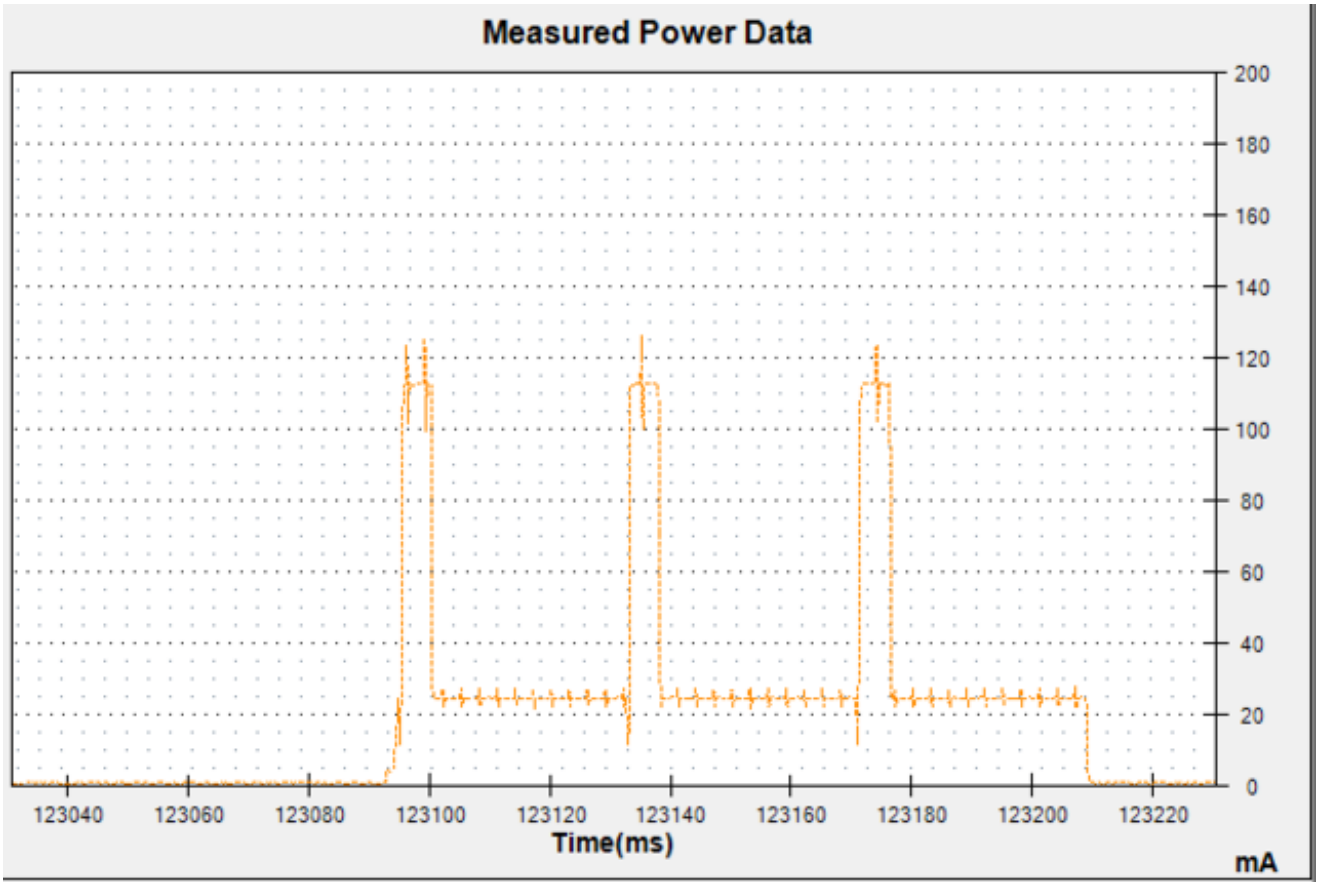
If you want to reduce even more power, use the Adafruit Sleepdog (http://adafru.it/fp8)library by installing and adding #include "Adafruit_SleepyDog.h" at the top of your sketch and replace

```
delay(1000);
```

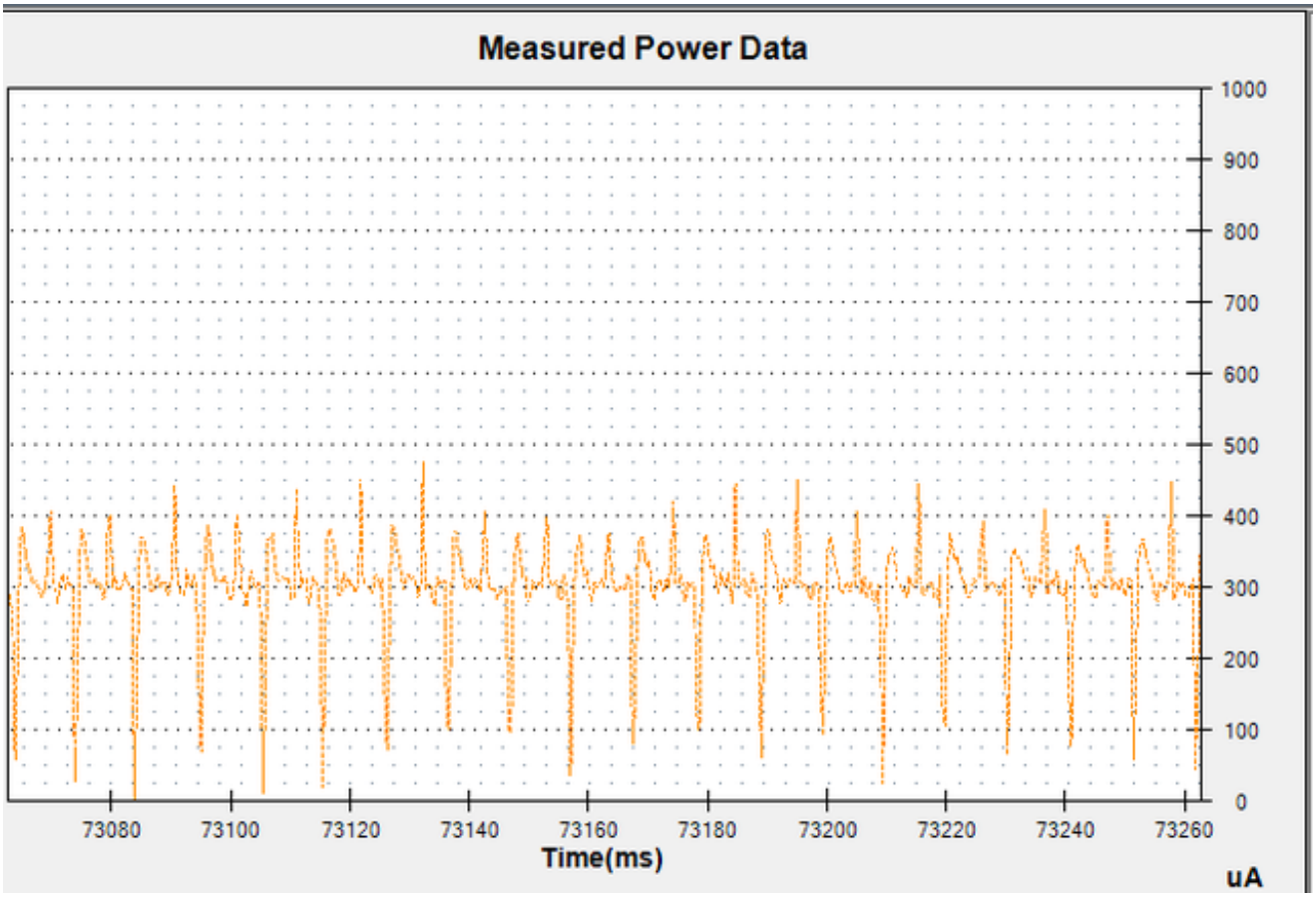
with

```
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)!

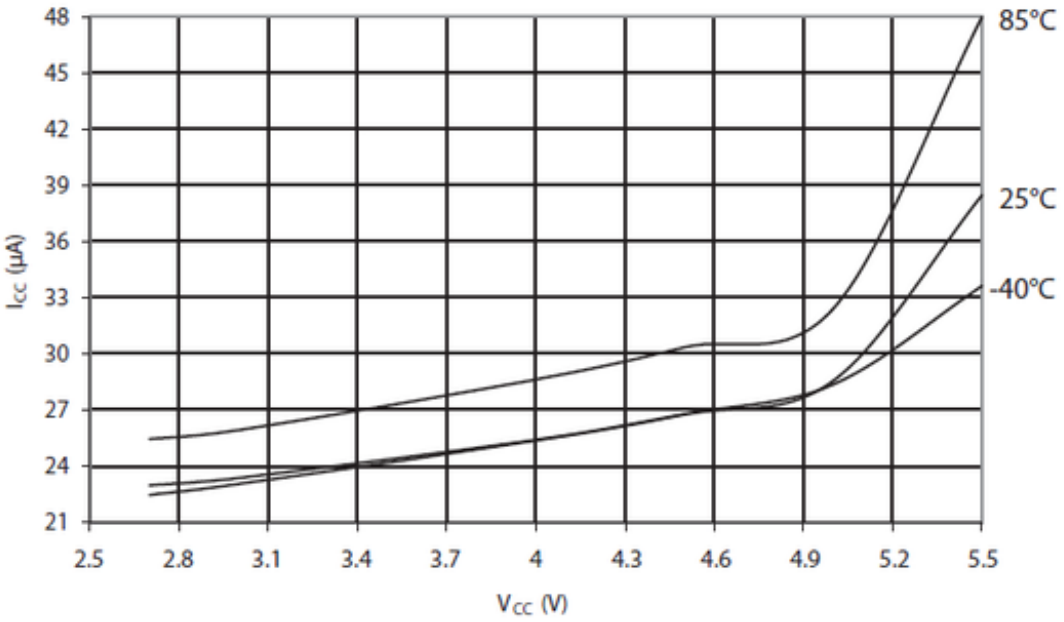


While its not easy to get the exact numbers for all of what comprise the 300uA there are a few quiescent current items on the Feather 32u4:

- 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

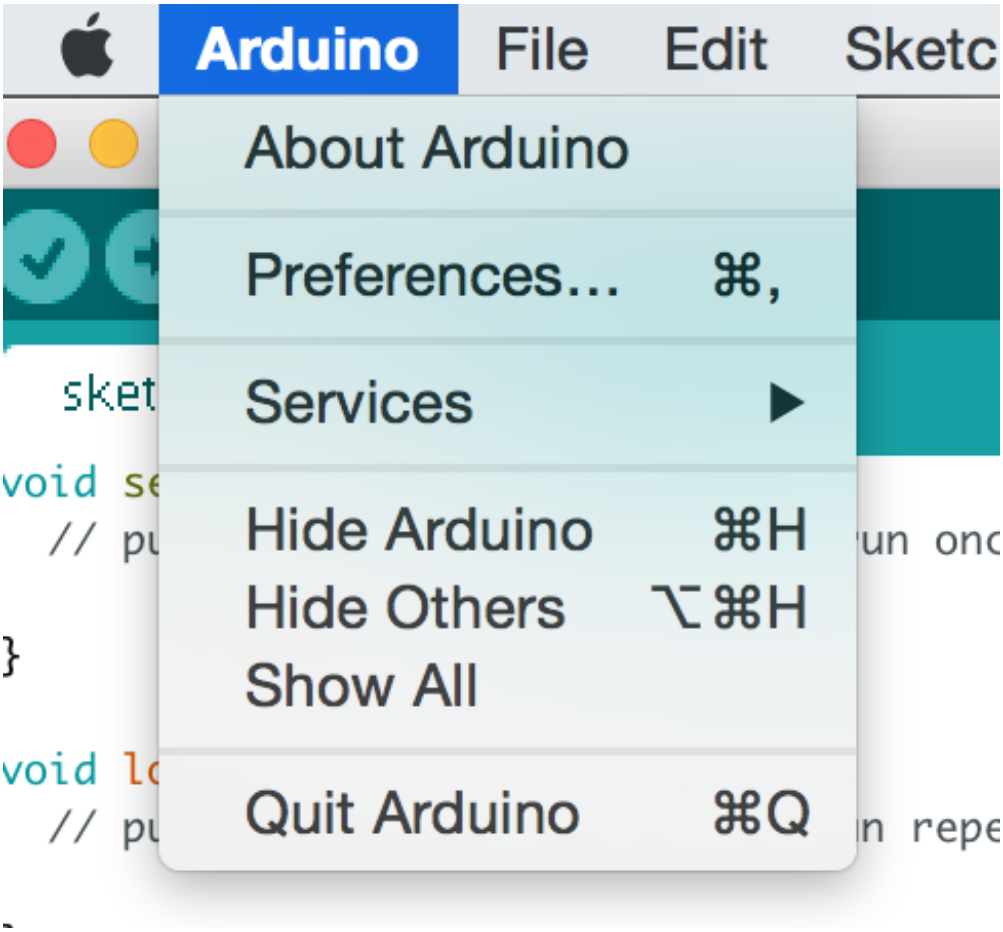
Arduino IDE Setup

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.

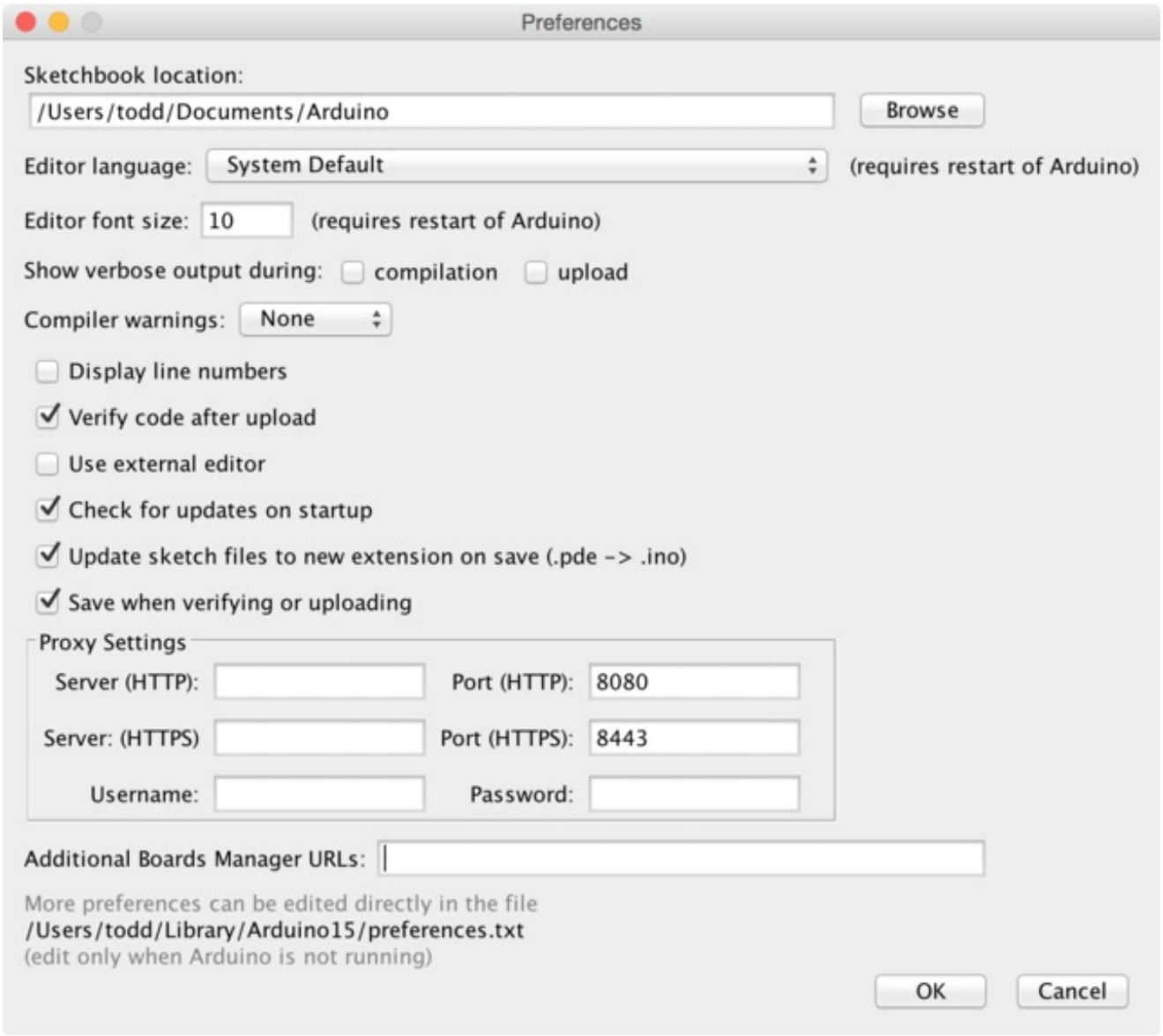
Arduino IDE v1.6.4+ Download

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*.



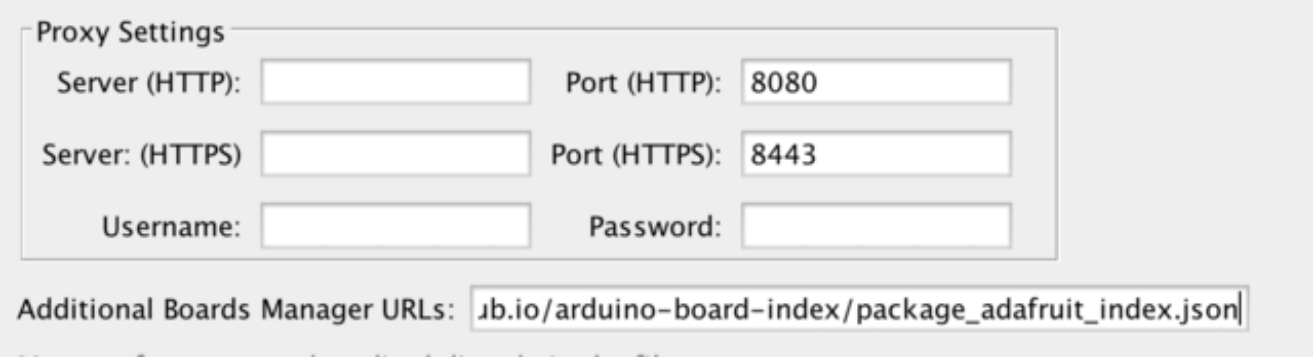
A dialog will pop up just like the one shown below.



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 of third 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
```



Make sure you remove the apt.adafruit.com proxy setting from the Arduino preferences if you have previously added it.

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:

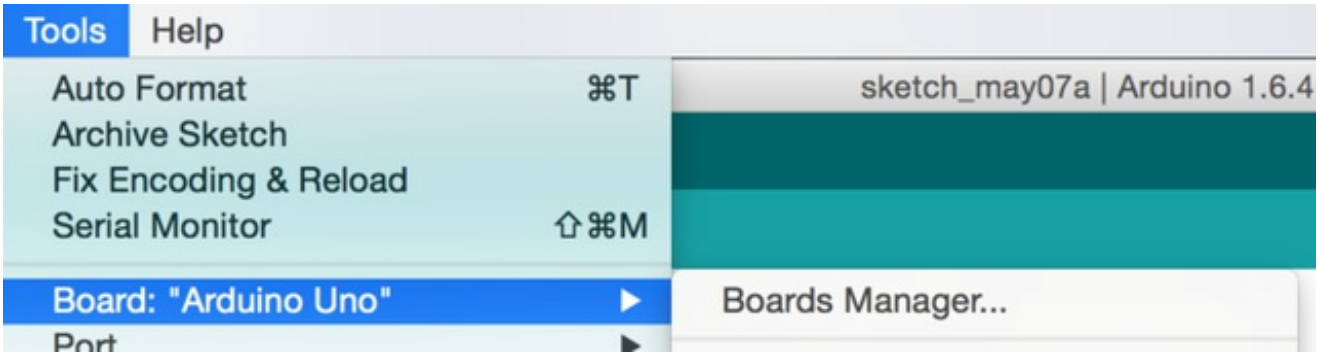
- **Adafruit AVR Boards** - Includes support for Flora, Gemma, Feather 32u4, Trinket, & Trinket Pro.
- **Adafruit SAMD Boards** - Includes support for Feather M0
- **Arduino Leonardo & Micro MIDI-USB** - This adds MIDI over USB support for the Flora, Feather 32u4, Micro and Leonardo using the arcore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

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

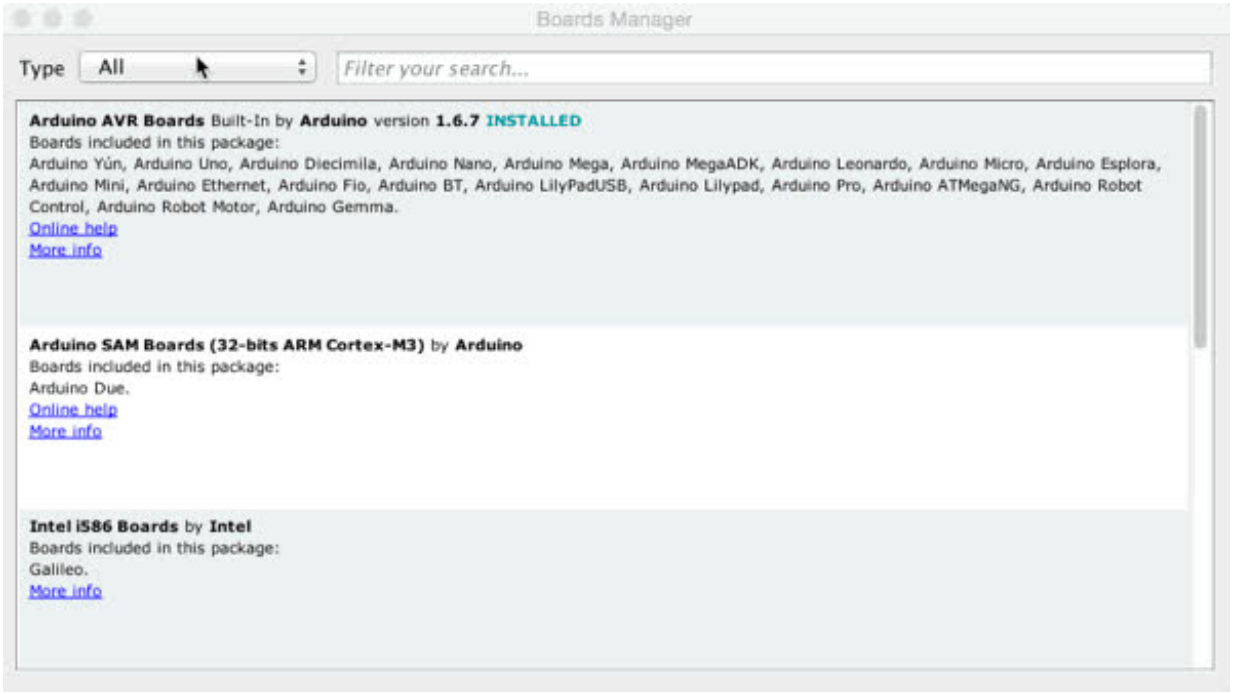
Using with Arduino IDE

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.

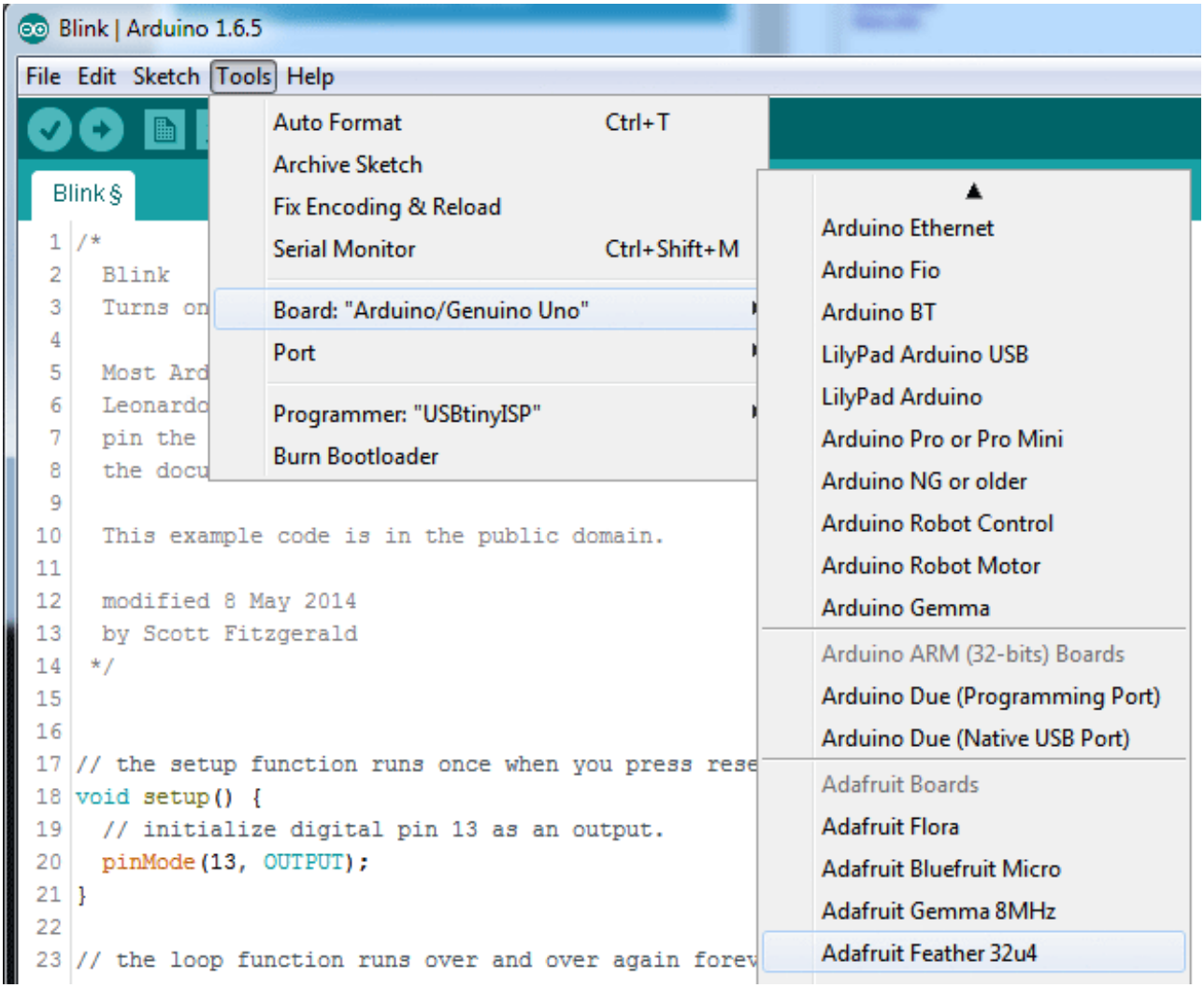
Now that you have added the appropriate URLs to the Arduino IDE preferences, you can open the **Boards Manager** by navigating to the **Tools->Board** menu.



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)

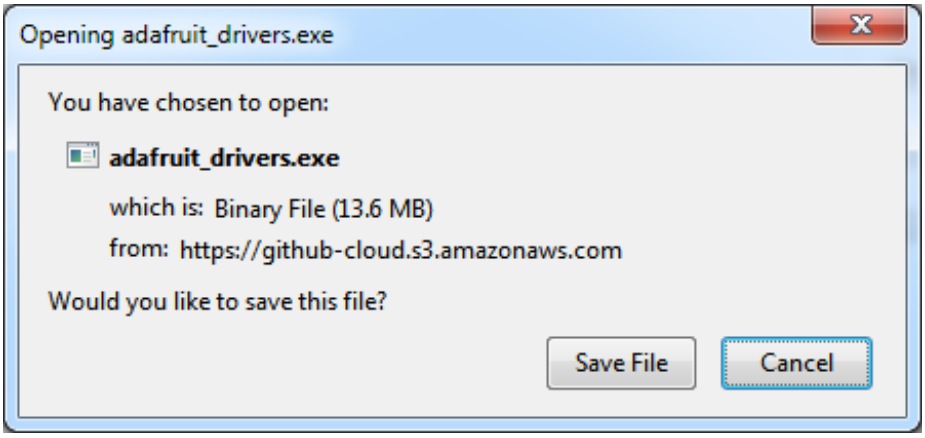
When you plug in the Feather, you'll need to possibly install a driver

Click below to download our Driver Installer

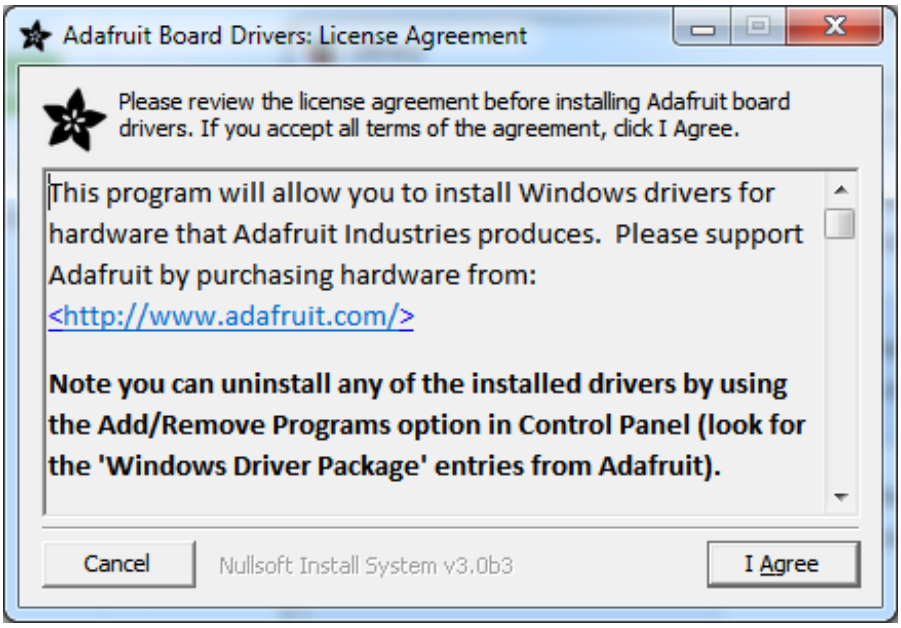
Download Adafruit Drivers Installer

http://adafru.it/mai

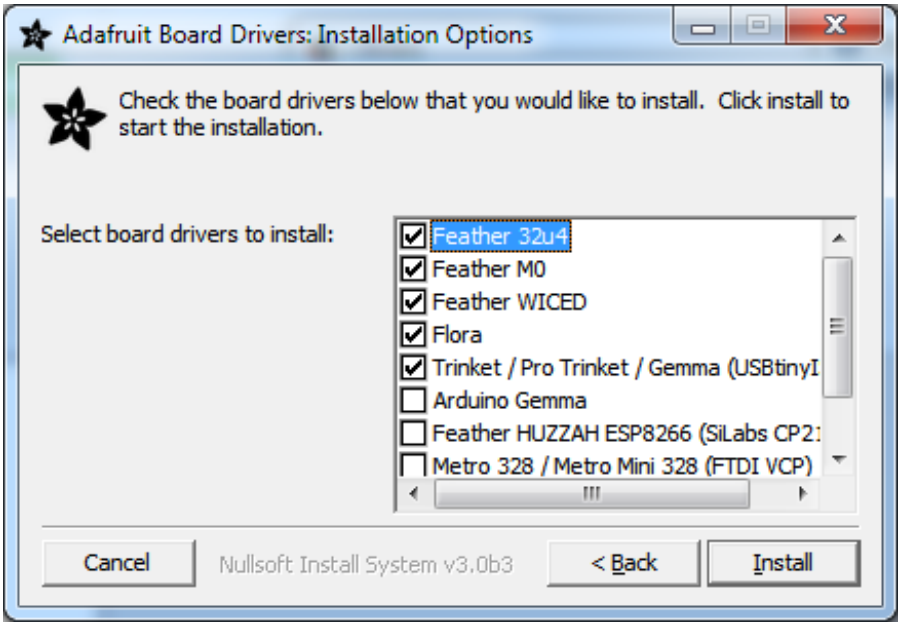
Download and run the installer



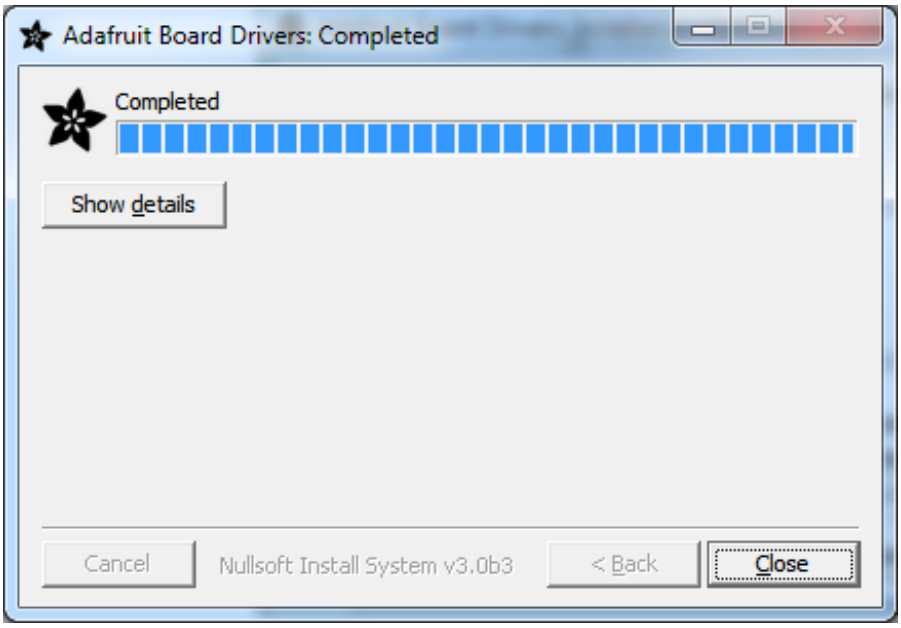
Run the installer! Since we bundle the SiLabs and FTDI drivers as well, you'll need to click through the license



Select which drivers you want to install:



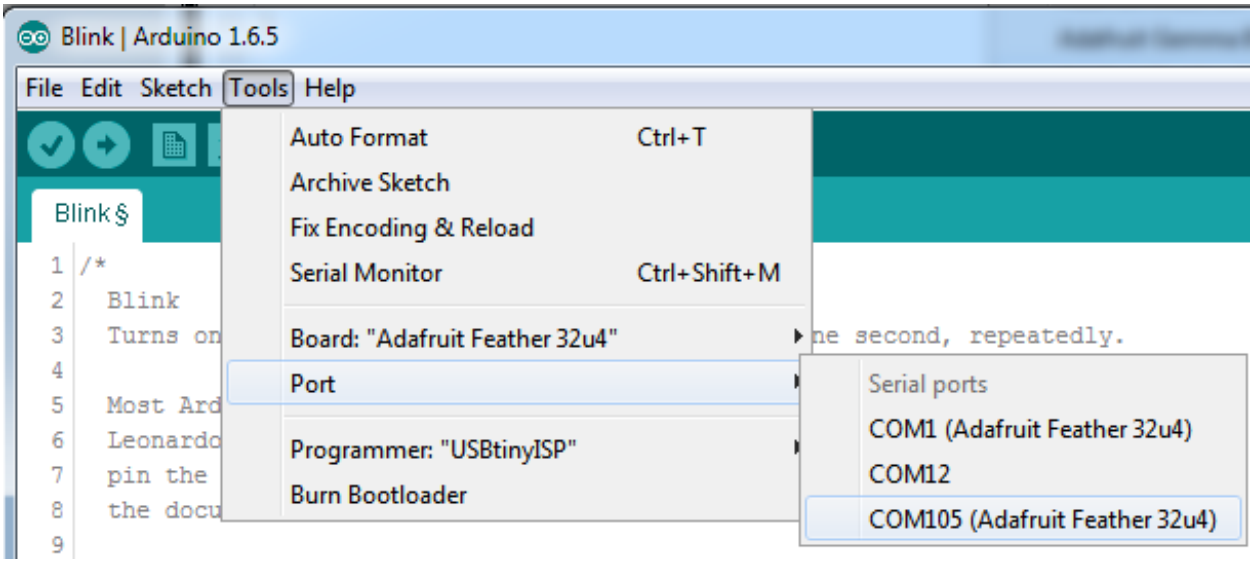
Click **Install** to do the installin'



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!



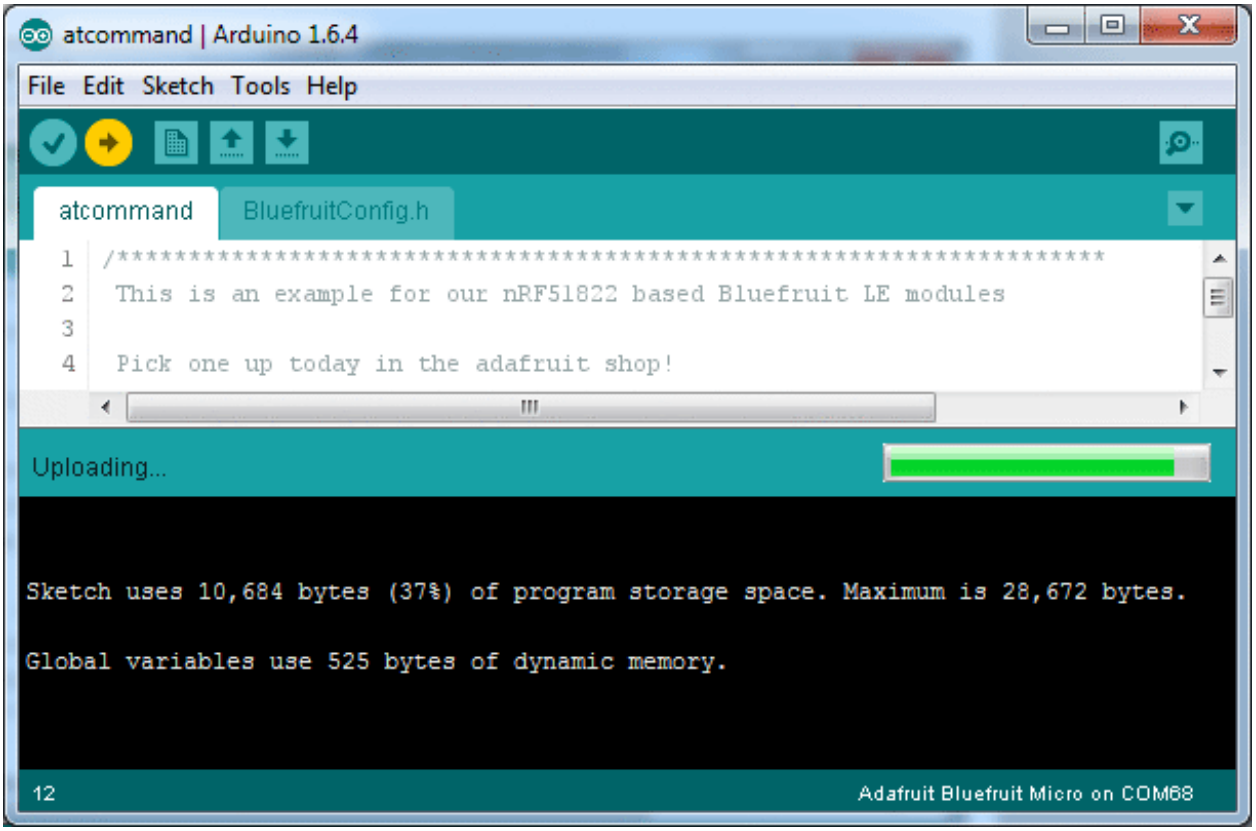
Now load up the Blink example

```
// 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/fP6)

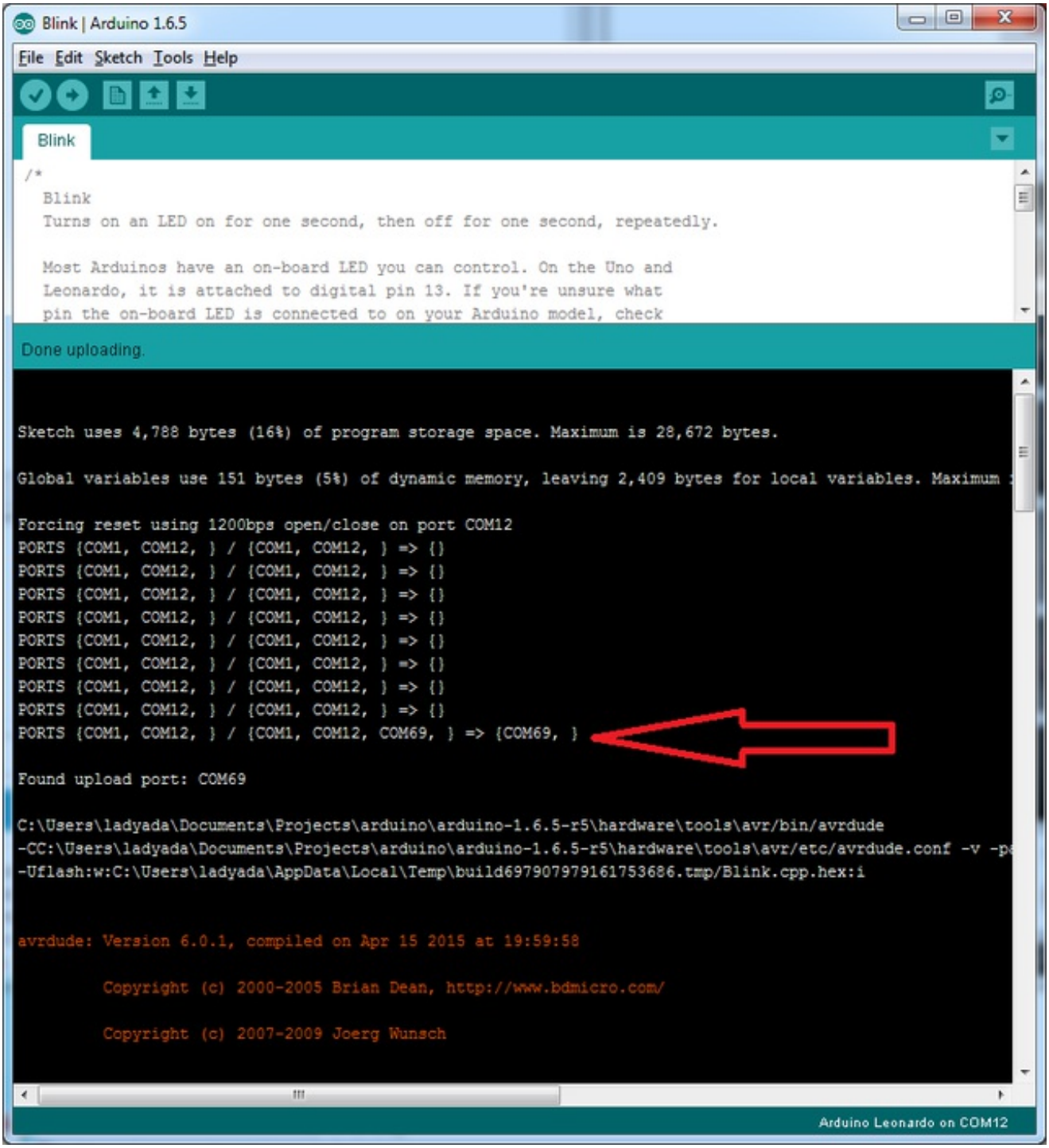
The fix for this issue is to make sure Adafruit's custom udev rules are applied to your system. One of these rules is made to configure modem manager not to touch the Bluefruit Micro board and will fix the programming difficulty issue. Follow the steps for installing Adafruit's udev rules on this page.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

HELP!

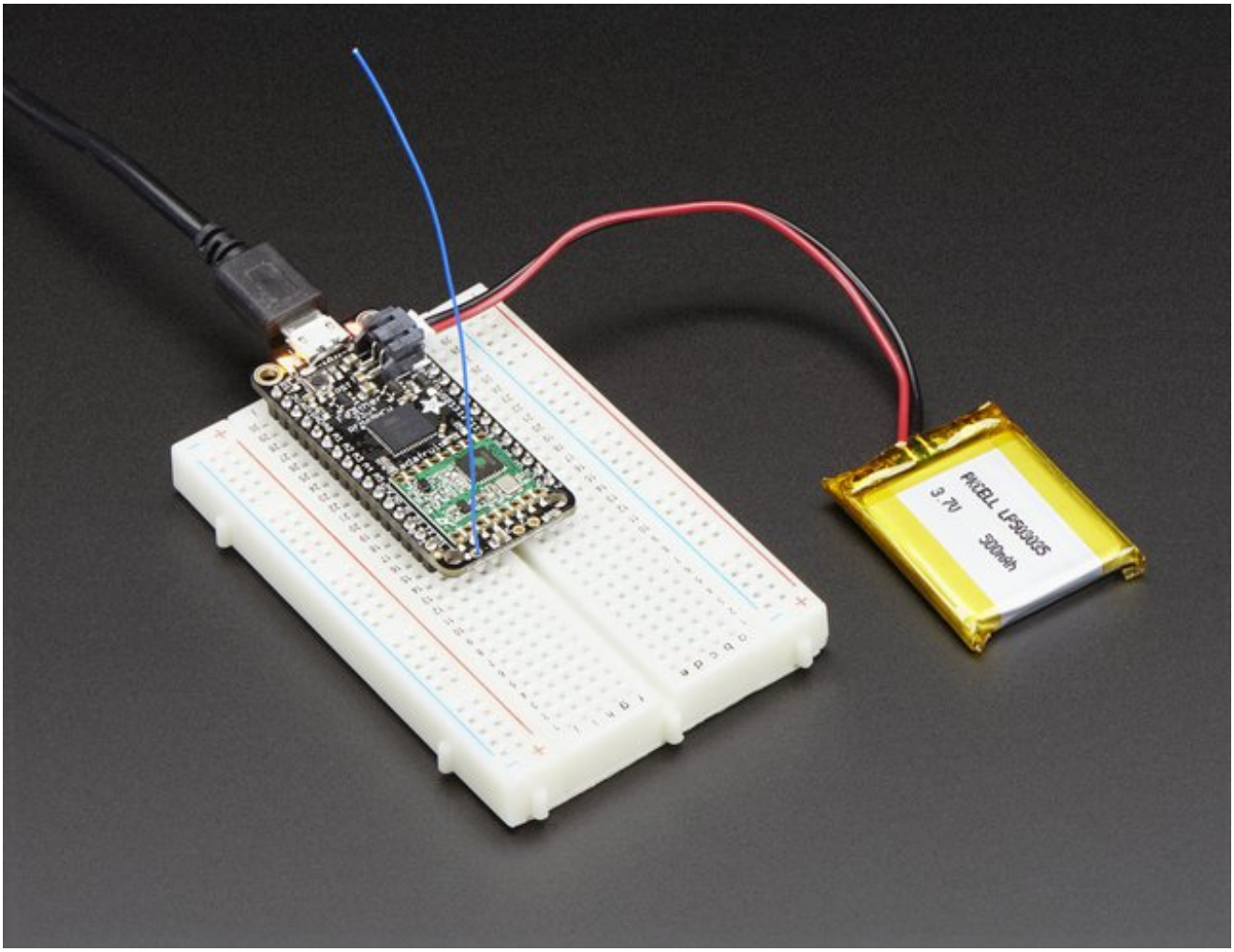
Ack! I "did something" and now when I plug in the Feather 32u4 it doesn't show up as a device anymore so I cant upload to it or fix it...
No problem! You can 'repair' a bad code upload easily. Note that this can happen if you set a watchdog timer or sleep mode that stops USB, or any sketch that 'crashes' your Feather

1. Turn on **verbose upload** in the Arduino IDE preferences
2. Plug in feather 32u4, 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
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, press and release the reset button, you'll see the red pulsing LED that tells you its now in bootloading mode**
8. The Feather 32u4 will show up as the Bootloader COM/Serial port
9. The IDE should see the bootloader COM/Serial port and upload properly

For Feather M0, same as above, but *double click* the reset button, you'll see the red pulsing LED that tells you its bootloading


Blink | Arduino 1.6.5
File Edit Sketch Tools Help
Blink
/*
Blink
Turns on an LED on for one second, then off for one second, repeatedly.
Most Arduinos have an on-board LED you can control. On the Uno and
Leonardo, it is attached to digital pin 13. If you're unsure what
pin the on-board LED is connected to on your Arduino model, check
Done uploading.
Sketch uses 4,788 bytes (16%) of program storage space. Maximum is 28,672 bytes.
Global variables use 151 bytes (5%) of dynamic memory, leaving 2,409 bytes for local variables. Maximum
Forcing reset using 1200bps open/close on port COM12
PORTS {COM1, COM12, } / {COM1, COM12, } => {}
PORTS {COM1, COM12, } / {COM1, COM12, } => {}
PORTS {COM1, COM12, } / {COM1, COM12, } => {}
PORTS {COM1, COM12, } / {COM1, COM12, } => {}
PORTS {COM1, COM12, } / {COM1, COM12, } => {}
PORTS {COM1, COM12, } / {COM1, COM12, } => {}
PORTS {COM1, COM12, } / {COM1, COM12, } => {}
PORTS {COM1, COM12, } / {COM1, COM12, } => {}
PORTS {COM1, COM12, } / {COM1, COM12, COM69, } => {COM69, }
Found upload port: COM69
C:\Users\ladyada\Documents\Projects\arduino\arduino-1.6.5-r5\hardware\tools\avr/bin/avrdude
-CC:\Users\ladyada\Documents\Projects\arduino\arduino-1.6.5-r5\hardware\tools\avr/etc/avrdude.conf -v -p
-Uflash:w:C:\Users\ladyada\AppData\Local\Temp\build6979079791617536 86.tmp/Blink.cpp.hex:i
avrdude: Version 6.0.1, compiled on Apr 15 2015 at 19:59:58
Copyright (c) 2000-2005 Brian Dean, http://www.bdmicro.com/
Copyright (c) 2007-2009 Joerg Wunsch
Arduino Leonardo on COM12

Using the Radio



Once you have the basic Feather 32u4 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 adjustbale 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 RFM96 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!

LowPowerLab RFM69 Library example

To begin talking to the radio, you will need to download RFM69 from their github repository (http://adafru.it/mCz). You can do that by visiting the github repo and manually downloading or, easier, just click this button to download the zip

Download RFM69 Library

http://adafru.it/mCB

Rename the uncompressed folder **RFM69** and check that the **RFM69** folder contains **RFM69.cpp** and **RFM69.h**

Place the **RFM69** library folder your **arduinosketchfolder/libraries/** folder.
You may need to create the **libraries** subfolder if its your first library. Restart the IDE.

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

Basic RX & TX example

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

Transmitter example code

This code will send a small packet of data once a second to node address #1

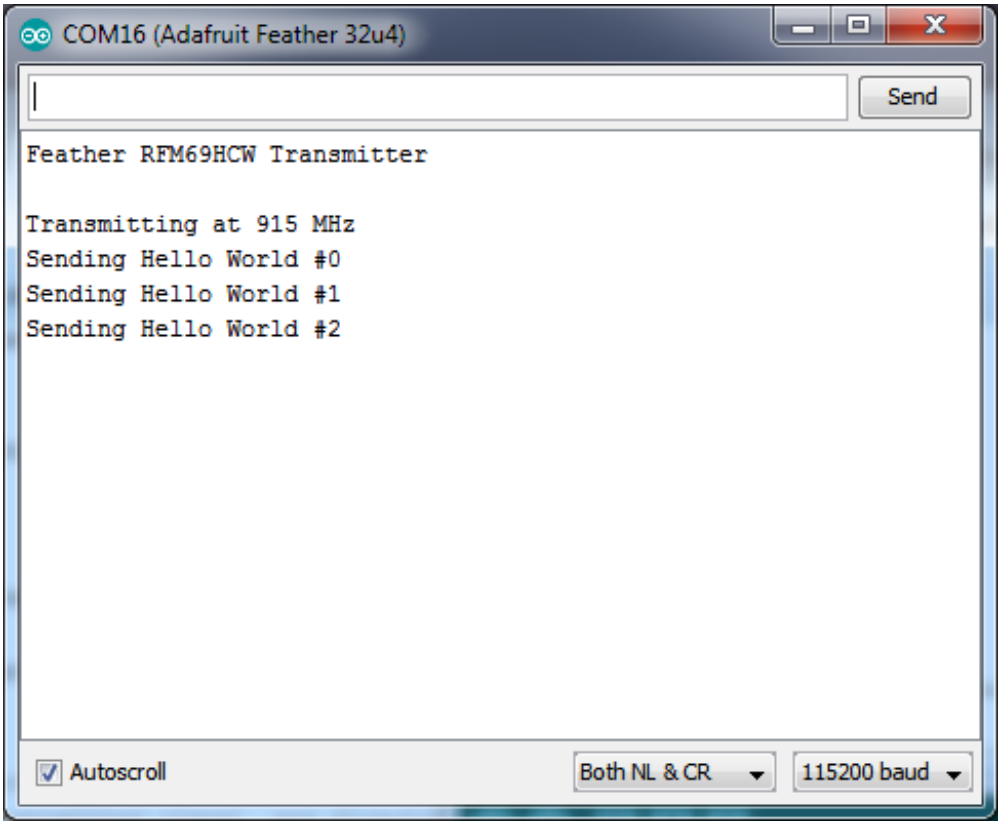
Load this code into your Transmitter Arduino/Feather!

```
/* RFM69 library and code by Felix Rusu - felix@lowpowerlab.com
// Get libraries at: https://github.com/LowPowerLab/
// Make sure you adjust the settings in the configuration section below !!!
// **********************************************************************************
// Copyright Felix Rusu, LowPowerLab.com
// Library and code by Felix Rusu - felix@lowpowerlab.com
// **********************************************************************************
// License
// **********************************************************************************
// This program is free software; you can redistribute it
// and/or modify it under the terms of the GNU General
// Public License as published by the Free Software
// Foundation; either version 3 of the License, or
// (at your option) any later version.
//
// This program is distributed in the hope that it will
// be useful, but WITHOUT ANY WARRANTY; without even the
// implied warranty of MERCHANTABILITY or FITNESS FOR A
// PARTICULAR PURPOSE. See the GNU General Public
// License for more details.
//
// You should have received a copy of the GNU General
// Public License along with this program.
// If not, see <http://www.gnu.org/licenses/>.
//
// Licence can be viewed at
// http://www.gnu.org/licenses/gpl-3.0.txt
//
// Please maintain this license information along with authorship
// and copyright notices in any redistribution of this code
// **********************************************************************************/

#include <RFM69.h>    //get it here: https://www.github.com/lowpowerlab/rfm69
#include <SPI.h>

//*****************************************************************************************
// ********** IMPORTANT SETTINGS - YOU MUST CHANGE/ONFIGURE TO FIT YOUR HARDWARE *************
//*****************************************************************************************
#define NETWORKID     100  // The same on all nodes that talk to each other
#define NODEID        2    // The unique identifier of this node
#define RECEIVER      1    // The recipient of packets

//Match frequency to the hardware version of the radio on your Feather
//#define FREQUENCY     RF69_433MHZ
//#define FREQUENCY     RF69_868MHZ
#define FREQUENCY     RF69_915MHZ
#define ENCRYPTKEY    "sampleEncryptKey" //exactly the same 16 characters/bytes on all nodes!
#define IS_RFM69HCW   true // set to 'true' if you are using an RFM69HCW module
```

```
//*********************************************************************************************
#define SERIAL_BAUD   115200

#define RFM69_CS      8
#define RFM69_IRQ     7
#define RFM69_IRQN    4  // Pin 7 is IRQ 4!
#define RFM69_RST     4

#define LED           13  // onboard blinky


int16_t packetnum = 0;  // packet counter, we increment per xmission

RFM69 radio = RFM69(RFM69_CS, RFM69_IRQ, IS_RFM69HCW, RFM69_IRQN);

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

  Serial.println("Feather RFM69HCW Transmitter");

  // Hard Reset the RFM module
  pinMode(RFM69_RST, OUTPUT);
  digitalWrite(RFM69_RST, HIGH);
  delay(100);
  digitalWrite(RFM69_RST, LOW);
  delay(100);

  // Initialize radio
  radio.initialize(FREQUENCY,NODEID,NETWORKID);
  if (IS_RFM69HCW) {
    radio.setHighPower();    // Only for RFM69HCW & HW!
  }
  radio.setPowerLevel(31); // power output ranges from 0 (5dBm) to 31 (20dBm)

  radio.encrypt(ENCRYPTKEY);

  pinMode(LED, OUTPUT);
  Serial.print("\nTransmitting at ");
  Serial.print(FREQUENCY==RF69_433MHZ ? 433 : FREQUENCY==RF69_868MHZ ? 868 : 915);
  Serial.println(" MHz");
}


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);

  if (radio.sendWithRetry(RECEIVER, radiopacket, strlen(radiopacket))) { //target node Id, message as string or byte
array, message length
    Serial.println("OK");
    Blink(LED, 50, 3); //blink LED 3 times, 50ms between blinks
  }

  radio.receiveDone(); //put radio in RX mode
  Serial.flush(); //make sure all serial data is clocked out before sleeping the MCU
}

void Blink(byte PIN, byte DELAY_MS, byte loops)
{
  for (byte i=0; i<loops; i++)
  {
    digitalWrite(PIN,HIGH);
    delay(DELAY_MS);
    digitalWrite(PIN,LOW);
    delay(DELAY_MS);
  }
}
```

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 Feather while you set up the RX Feather.

Receiver example code

This code will receive and acknowledge a small packet of data.

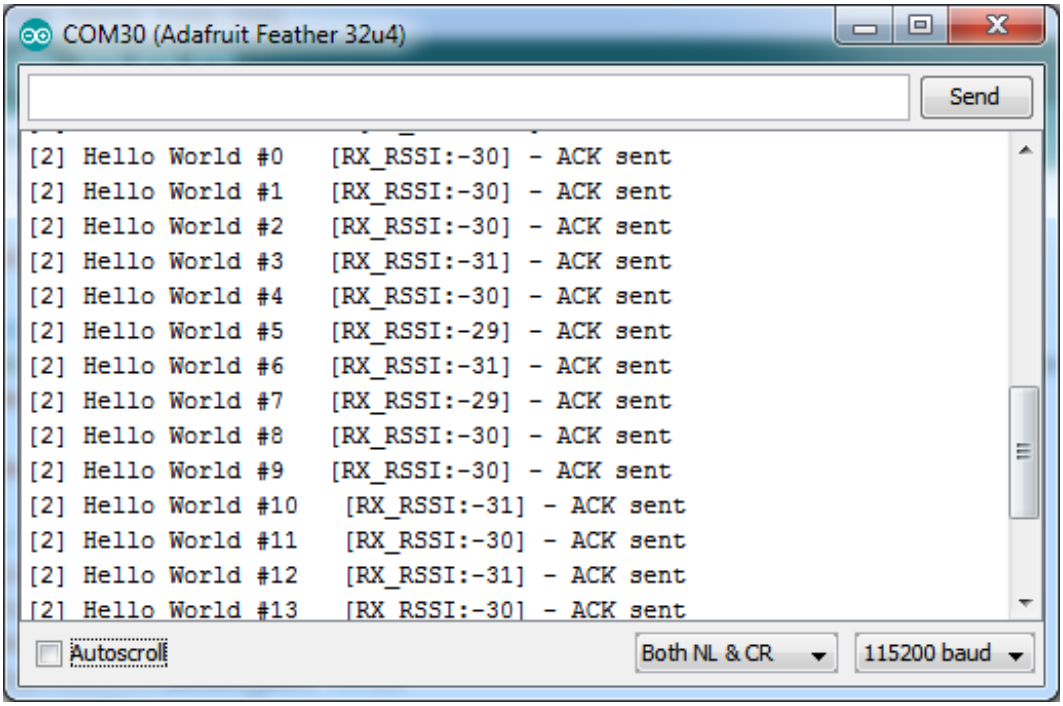
Load this code into your **Receiver** Arduino/Feather!

```
/* RFM69 library and code by Felix Rusu - felix@lowpowerlab.com
// Get libraries at: https://github.com/LowPowerLab/
// Make sure you adjust the settings in the configuration section below !!!
// **********************************************************************************
// Copyright Felix Rusu, LowPowerLab.com
// Library and code by Felix Rusu - felix@lowpowerlab.com
// **********************************************************************************
// License
// **********************************************************************************
// This program is free software; you can redistribute it
// and/or modify it under the terms of the GNU General
// Public License as published by the Free Software
// Foundation; either version 3 of the License, or
// (at your option) any later version.
//
// This program is distributed in the hope that it will
// be useful, but WITHOUT ANY WARRANTY; without even the
// implied warranty of MERCHANTABILITY or FITNESS FOR A
// PARTICULAR PURPOSE. See the GNU General Public
// License for more details.
//
// You should have received a copy of the GNU General
// Public License along with this program.
// If not, see <http://www.gnu.org/licenses></http:>.
//
// Licence can be viewed at
// http://www.gnu.org/licenses/gpl-3.0.txt
//
// Please maintain this license information along with authorship
// and copyright notices in any redistribution of this code
// **********************************************************************************/

#include <RFM69.h>    //get it here: https://www.github.com/lowpowerlab/rfm69
#include <SPI.h>

//*****************************************************************************************
// ********** IMPORTANT SETTINGS - YOU MUST CHANGE/ONFIGURE TO FIT YOUR HARDWARE *************
//*****************************************************************************************
#define NETWORKID     100  //the same on all nodes that talk to each other
#define NODEID        1
```

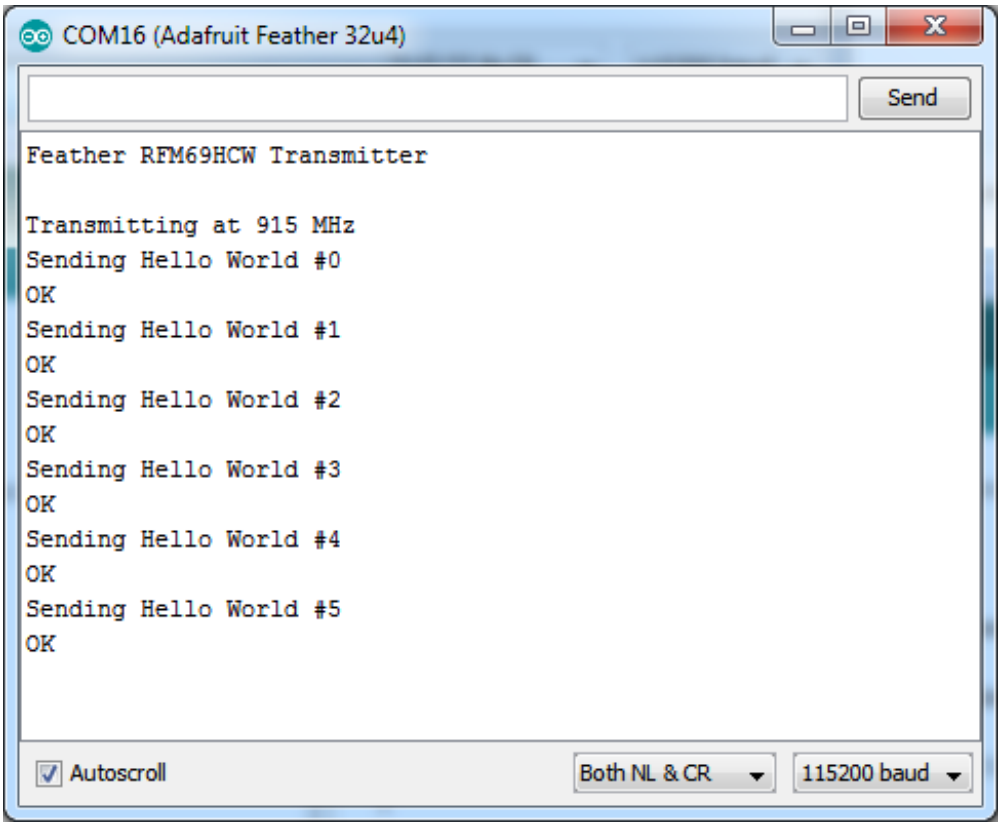
```
//Match frequency to the hardware version of the radio on your Feather
//#define FREQUENCY     RF69_433MHZ
//#define FREQUENCY     RF69_868MHZ
#define FREQUENCY     RF69_915MHZ
#define ENCRYPTKEY    "sampleEncryptKey" //exactly the same 16 characters/bytes on all nodes!
#define IS_RFM69HCW    true // set to 'true' if you are using an RFM69HCW module

//*****************************************************************************************
#define SERIAL_BAUD   115200

#define RFM69_CS      8
#define RFM69_IRQ     7
#define RFM69_IRQN    4  // Pin 7 is IRQ 4!
#define RFM69_RST     4

#define LED           13  // onboard blinky

int16_t packetnum = 0;  // packet counter, we increment per xmission

RFM69 radio = RFM69(RFM69_CS, RFM69_IRQ, IS_RFM69HCW, RFM69_IRQN);

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

  Serial.println("Feather RFM69HCW Receiver");

  // Hard Reset the RFM module
  pinMode(RFM69_RST, OUTPUT);
  digitalWrite(RFM69_RST, HIGH);
  delay(100);
  digitalWrite(RFM69_RST, LOW);
  delay(100);

  // Initialize radio
  radio.initialize(FREQUENCY,NODEID,NETWORKID);
  if (IS_RFM69HCW) {
    radio.setHighPower();    // Only for RFM69HCW & HW!
  }
  radio.setPowerLevel(31); // power output ranges from 0 (5dBm) to 31 (20dBm)

  radio.encrypt(ENCRYPTKEY);

  pinMode(LED, OUTPUT);

  Serial.print("\nListening at ");
  Serial.print(FREQUENCY==RF69_433MHZ ? 433 : FREQUENCY==RF69_868MHZ ? 868 : 915);
  Serial.println(" MHz");
```

```
}

void loop() {
  //check if something was received (could be an interrupt from the radio)
  if (radio.receiveDone())
  {
    //print message received to serial
    Serial.print('[');Serial.print(radio.SENDERID);Serial.print("] ");
    Serial.print((char*)radio.DATA);
    Serial.print("   [RX_RSSI:");Serial.print(radio.RSSI);Serial.print("]");

    //check if received message contains Hello World
    if (strstr((char *)radio.DATA, "Hello World"))
    {
      //check if sender wanted an ACK
      if (radio.ACKRequested())
      {
        radio.sendACK();
        Serial.println(" - ACK sent");
      }
      Blink(LED, 40, 3); //blink LED 3 times, 40ms between blinks
    }
  }

  radio.receiveDone(); //put radio in RX mode
  Serial.flush(); //make sure all serial data is clocked out before sleeping the MCU
}

void Blink(byte PIN, byte DELAY_MS, byte loops)
{
  for (byte i=0; i<loops; i++)
  {
    digitalWrite(PIN,HIGH);
    delay(DELAY_MS);
    digitalWrite(PIN,LOW);
    delay(DELAY_MS);
  }
}
```

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 **OK** after each transmisssion because it got an acknowledgement 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 Net & ID Configuration

The first thing you need to do for *all* radio nodes on your network is to configure the network ID (the identifier shared by all radio nodes you are using) and individual ID's for each node.

```
//*************************************************************************************
// ********** IMPORTANT SETTINGS - YOU MUST CHANGE/ONFIGURE TO FIT YOUR HARDWARE *************
//*************************************************************************************
#define NETWORKID     100  // The same on all nodes that talk to each other
#define NODEID        2    // The unique identifier of this node
```

Make sure all radios you are using are sharing the same network ID and all have different/unique Node ID's!

Radio Type Config

You will also have to set up type of radio you are using, an encryption key if you're using it, and the type of radio (high or low power)

```
//Match frequency to the hardware version of the radio on your Feather
//#define FREQUENCY     RF69_433MHZ
//#define FREQUENCY     RF69_868MHZ
#define FREQUENCY     RF69_915MHZ
#define ENCRYPTKEY    "sampleEncryptKey" //exactly the same 16 characters/bytes on all nodes!
#define IS_RFM69HCW   true // set to 'true' if you are using an RFM69HCW module
```

For all radios they will need to be on the same frequency. If you have a 433MHz radio you will have 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

The **ENCRYPTKEY** is 16 characters long and keeps your messages a little more secret than just plain text

IS_RFM69HCW is used for telling the library that we are using the high power version of the radio, make sure its set to **true**

Feather Radio Pinout

This is the pinout setup for all Feather 32u4 RFM69's so keep this the same

```
#define RFM69_CS      8
#define RFM69_IRQ     7
#define RFM69_IRQN    4  // Pin 7 is IRQ 4!
#define RFM69_RST     4
```

You can then instantiate the radio object with our custom pin numbers. Note that the IRQ is defined by the IRQ *number* not the pin.

```
RFM69 radio = RFM69(RFM69_CS, RFM69_IRQ, IS_RFM69HCW, RFM69_IRQN);
```

Setup

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

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

  Serial.println("Feather RFM69HCW Transmitter");

  // Hard Reset the RFM module
  pinMode(RFM69_RST, OUTPUT);
  digitalWrite(RFM69_RST, HIGH);
  delay(100);
  digitalWrite(RFM69_RST, LOW);
  delay(100);
```

Remove the **while (!Serial);** line if you are not tethering to a computer, as it will cause the Feather to halt until a USB connection is made!

Initializing Radio

The library gets initialized with the frequency, unique identifier and network identifier. For HCW type modules (which you are using) it will also turn on the amplifier. You can also configure the output power level, the number ranges from 0 to 31. Start with the highest power level (31) and then scale down as necessary

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

```
  // Initialize radio
  radio.initialize(FREQUENCY,NODEID,NETWORKID);
  if (IS_RFM69HCW) {
    radio.setHighPower();    // Only for RFM69HCW & HW!
  }
  radio.setPowerLevel(31); // power output ranges from 0 (5dBm) to 31 (20dBm)

  radio.encrypt(ENCRYPTKEY);
```

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 an acknowledgement

```
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);

  if (radio.sendWithRetry(RECEIVER, radiopacket, strlen(radiopacket))) { //target node Id, message as string or byte
array, message length
    Serial.println("OK");
    Blink(LED, 50, 3); //blink LED 3 times, 50ms between blinks
  }

  radio.receiveDone(); //put radio in RX mode
  Serial.flush(); //make sure all serial data is clocked out before sleeping the MCU
}
```

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 **sendWithRetry** to the address in the first argument (RECEIVER), and passes in the array of data and the length of the data.

If you receive a **true** from that call it means an acknowledgement was received and print **OK** - either way the transmitter will continue the loop and sleep for a second until the next TX.

Receiver Code

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

```
void loop() {
  //check if something was received (could be an interrupt from the radio)
  if (radio.receiveDone())
  {
    //print message received to serial
    Serial.print('[');Serial.print(radio.SENDERID);Serial.print("] ");
    Serial.print((char*)radio.DATA);
    Serial.print("   [RX_RSSI:");Serial.print(radio.RSSI);Serial.print("]");

    //check if received message contains Hello World
    if (strstr((char *)radio.DATA, "Hello World"))
    {
      //check if sender wanted an ACK
      if (radio.ACKRequested())
      {
        radio.sendACK();
        Serial.println(" - ACK sent");
      }
```

```
    Blink(LED, 40, 3); //blink LED 3 times, 40ms between blinks
  }
 }

 Serial.flush(); //make sure all serial data is clocked out before sleeping the MCU
}
```

Instead of transmitting, it is constantly checking if there's any data packets that have been received. **receiveDone()** will return true if a packet for the current node, in the right network and 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 acknowledge the packet.

Once done it will continue waiting for a new packet

Receiver/Transmiter Demo

OK once you have that going you can try this example, we're using the Feather with an OLED wing but 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 **but** change the following lines

```
#define NODEID        NODE1 // Swap these two for other Feather
#define RECEIVER      NODE2 // Swap these two for other Feather
```

swap the NODE1/NODE2 identifers for the second Feather (one is NODE1 and the receiver is NODE 2, the other is NODE2 and the receiver is NODE1)

```
/* RFM69 library and code by Felix Rusu - felix@lowpowerlab.com
// Get libraries at: https://github.com/LowPowerLab/
// Make sure you adjust the settings in the configuration section below !!!
// **********************************************************************************
// Copyright Felix Rusu, LowPowerLab.com
// Library and code by Felix Rusu - felix@lowpowerlab.com
// **********************************************************************************
// License
// **********************************************************************************
// This program is free software; you can redistribute it
// and/or modify it under the terms of the GNU General
// Public License as published by the Free Software
// Foundation; either version 3 of the License, or
// (at your option) any later version.
//
// This program is distributed in the hope that it will
// be useful, but WITHOUT ANY WARRANTY; without even the
```

```
// implied warranty of MERCHANTABILITY or FITNESS FOR A
// PARTICULAR PURPOSE. See the GNU General Public
// License for more details.
//
// You should have received a copy of the GNU General
// Public License along with this program.
// If not, see <http://www.gnu.org/licenses></http:>.
//
// Licence can be viewed at
// http://www.gnu.org/licenses/gpl-3.0.txt
//
// Please maintain this license information along with authorship
// and copyright notices in any redistribution of this code
// **********************************************************************************/

#include <RFM69.h>    //get it here: https://www.github.com/lowpowerlab/rfm69
#include <SPI.h>
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>

Adafruit_SSD1306 oled = Adafruit_SSD1306();
#define BUTTON_A 9
#define BUTTON_B 6
#define BUTTON_C 5

//************************************************************************************************
// ********** IMPORTANT SETTINGS - YOU MUST CHANGE/ONFIGURE TO FIT YOUR HARDWARE *************
//************************************************************************************************
#define NETWORKID     100  //the same on all nodes that talk to each other
#define NODE1        1
#define NODE2        2

#define NODEID       NODE1 // Swap these two for other Feather
#define RECEIVER     NODE2 // Swap these two for other Feather

//Match frequency to the hardware version of the radio on your Feather
//#define FREQUENCY     RF69_433MHZ
//#define FREQUENCY     RF69_868MHZ
#define FREQUENCY     RF69_915MHZ
#define ENCRYPTKEY    "sampleEncryptKey" //exactly the same 16 characters/bytes on all nodes!
#define IS_RFM69HCW    true // set to 'true' if you are using an RFM69HCW module

//************************************************************************************************
#define SERIAL_BAUD   115200

#define RFM69_CS      8
#define RFM69_IRQ     7
#define RFM69_IRQN    4  // Pin 7 is IRQ 4!
```

```
#define RFM69_RST     4

#define LED           13  // onboard blinky

int16_t packetnum = 0;  // packet counter, we increment per xmission

RFM69 radio = RFM69(RFM69_CS, RFM69_IRQ, IS_RFM69HCW, RFM69_IRQN);

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

  // Initialize OLED display
  oled.begin(SSD1306_SWITCHCAPVCC, 0x3C);  // initialize with the I2C addr 0x3C (for the 128x32)
  oled.display();
  delay(250);
  oled.clearDisplay();
  oled.display();

  // OLED text display tests
  oled.setTextSize(1);
  oled.setTextColor(WHITE);
  oled.setCursor(0,0);

  pinMode(BUTTON_A, INPUT_PULLUP);
  pinMode(BUTTON_B, INPUT_PULLUP);
  pinMode(BUTTON_C, INPUT_PULLUP);

  Serial.println("Feather RFM69HCW RX/TX OLED demo");

  // Hard Reset the RFM module
  pinMode(RFM69_RST, OUTPUT);
  digitalWrite(RFM69_RST, HIGH);
  delay(100);
  digitalWrite(RFM69_RST, LOW);
  delay(100);

  // Initialize radio
  radio.initialize(FREQUENCY,NODEID,NETWORKID);
  if (IS_RFM69HCW) {
    radio.setHighPower();    // Only for RFM69HCW & HW!
  }
  radio.setPowerLevel(31); // power output ranges from 0 (5dBm) to 31 (20dBm)

  radio.encrypt(ENCRYPTKEY);

  pinMode(LED, OUTPUT);

  Serial.print("Node #"); Serial.print(NODEID); Serial.print(" radio at ");
```

```
  Serial.print(FREQUENCY==RF69_433MHZ ? 433 : FREQUENCY==RF69_868MHZ ? 868 : 915);
  Serial.println(" MHz");

  oled.print("Node #"); oled.print(NODEID); oled.print(" radio at ");
  oled.print(FREQUENCY==RF69_433MHZ ? 433 : FREQUENCY==RF69_868MHZ ? 868 : 915);
  oled.println(" MHz");
  oled.display();
}

void loop() {
  //check if something was received (could be an interrupt from the radio)
  if (radio.receiveDone())
  {
    //print message received to serial
    Serial.print('[');Serial.print(radio.SENDERID);Serial.print("] ");
    Serial.print((char*)radio.DATA);
    Serial.print("   [RX_RSSI:");Serial.print(radio.RSSI);Serial.print("]");

    oled.clearDisplay();
    oled.setCursor(0,0);
    oled.print('[');oled.print(radio.SENDERID);oled.print("] ");
    oled.println((char*)radio.DATA);
    oled.print("[RX_RSSI:"); oled.print(radio.RSSI); oled.print("]");
    oled.display();

    //check if sender wanted an ACK
    if (radio.ACKRequested())
    {
      radio.sendACK();
      Serial.println(" - ACK sent");
    }
    Blink(LED, 40, 3); //blink LED 3 times, 40ms between blinks
  }

  if (!digitalRead(BUTTON_A) || !digitalRead(BUTTON_B) || !digitalRead(BUTTON_C))
  {
    Serial.println("Button pressed!");

    char radiopacket[20] = "Button #";
    if (!digitalRead(BUTTON_A)) radiopacket[8] = 'A';
    if (!digitalRead(BUTTON_B)) radiopacket[8] = 'B';
    if (!digitalRead(BUTTON_C)) radiopacket[8] = 'C';
    radiopacket[9] = 0;

    Serial.print("Sending "); Serial.println(radiopacket);

    if (radio.sendWithRetry(RECEIVER, radiopacket, strlen(radiopacket))) { //target node Id, message as string or byte
array, message length
      Serial.println("OK");
```

```
      Blink(LED, 50, 3); //blink LED 3 times, 50ms between blinks
    }
    delay(250); // delay for retransmission
  }

  radio.receiveDone(); //put radio in RX mode
  Serial.flush(); //make sure all serial data is clocked out before sleeping the MCU
}

void Blink(byte PIN, byte DELAY_MS, byte loops)
{
  for (byte i=0; i<loops; i++)
  {
    digitalWrite(PIN,HIGH);
    delay(DELAY_MS);
    digitalWrite(PIN,LOW);
    delay(DELAY_MS);
  }
}
```

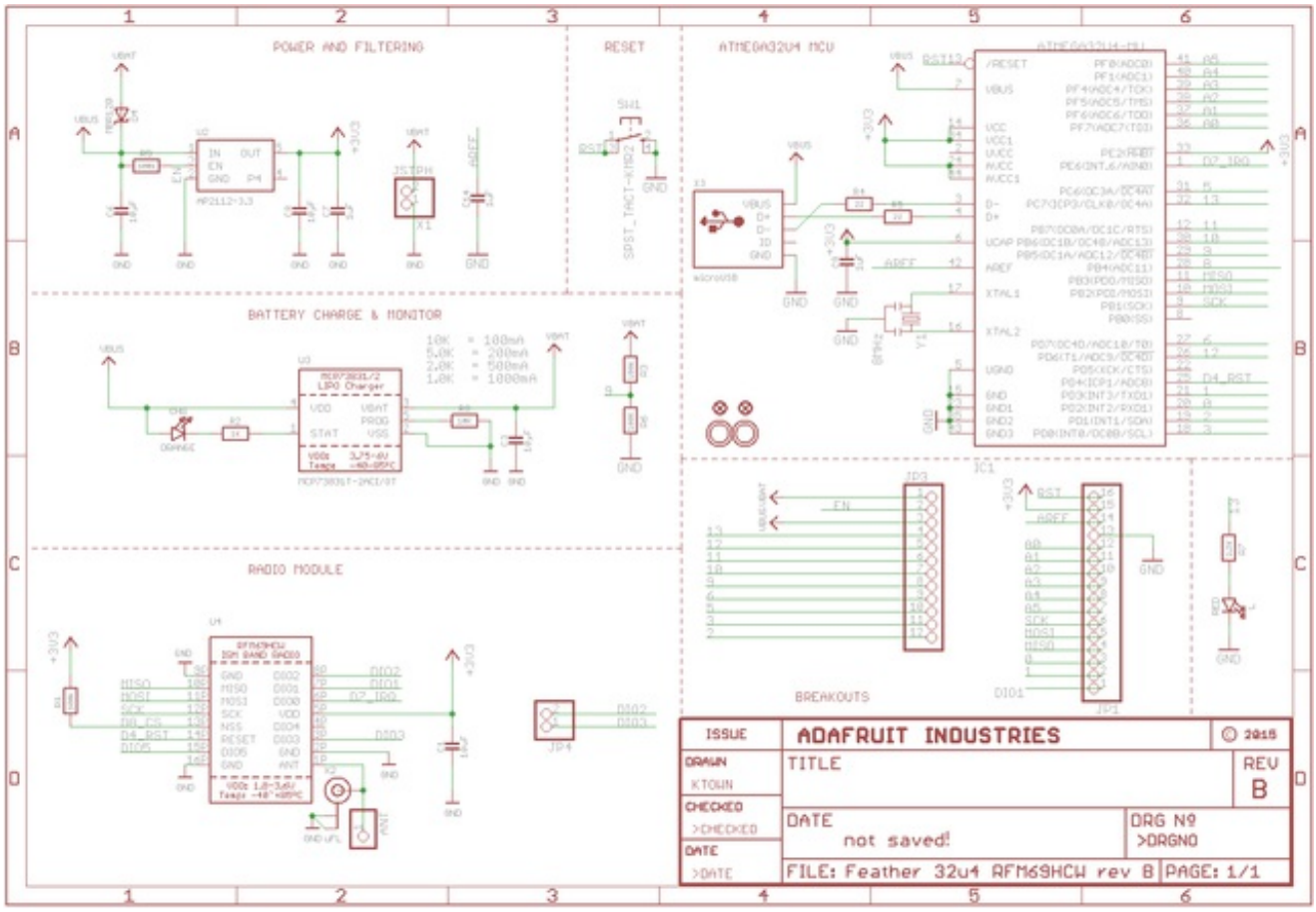
This demo code shows how you can listen for packets and also check for button presses (or sensor data or whatever you like) and send them back and forth between the two radios!

Downloads

Datasheets

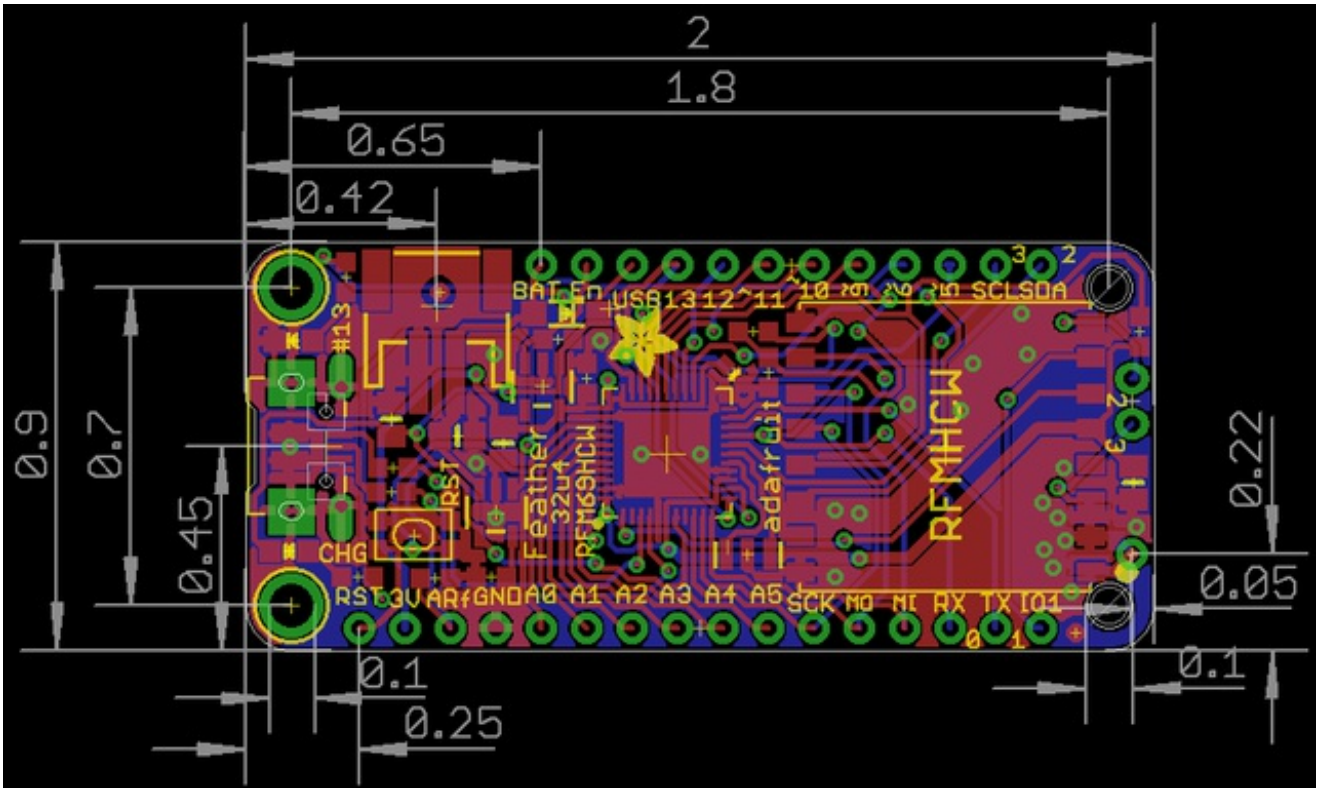
- RFM69HCW datasheet - contains the SX1231 datasheet plus details about the module (http://adafru.it/mCu)
- SX1231 (the actual radio chip used) datasheet (http://adafru.it/mCv)

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 13 12 11 10 SCL SDA
RST 3V AREF GND A0 A1 A2 A3 A4 A5 SCK MO MI RX TX 0 1
CHG
Feather
32u4
RFM69HCW
adafruit
RFMHCW