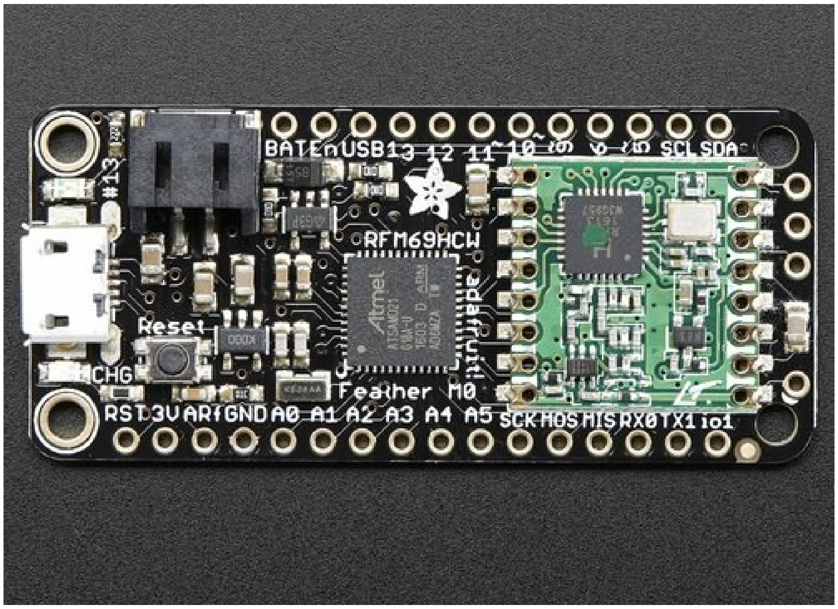
Adafruit Feather M0 Radio with RFM69 Packet Radio

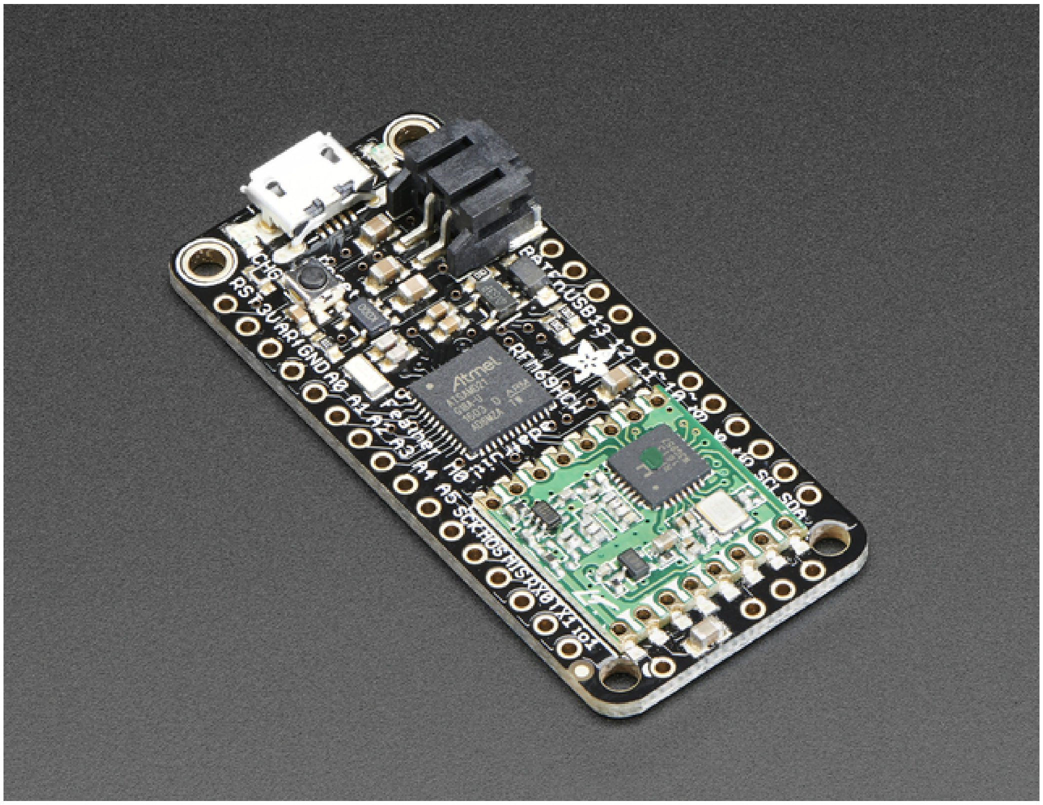
Created by lady ada


BAT En USB 13 12 11 10 9 6 5 SCL SDA
#13
RFM69HCW
Reset
CHG
Feather M0
RST 3V ARf GND A0 A1 A2 A3 A4 A5 SCK MOS MIS RX0 TX1 io1

Last updated on 2016-11-07 07:53:15 PM UTC

Guide Contents

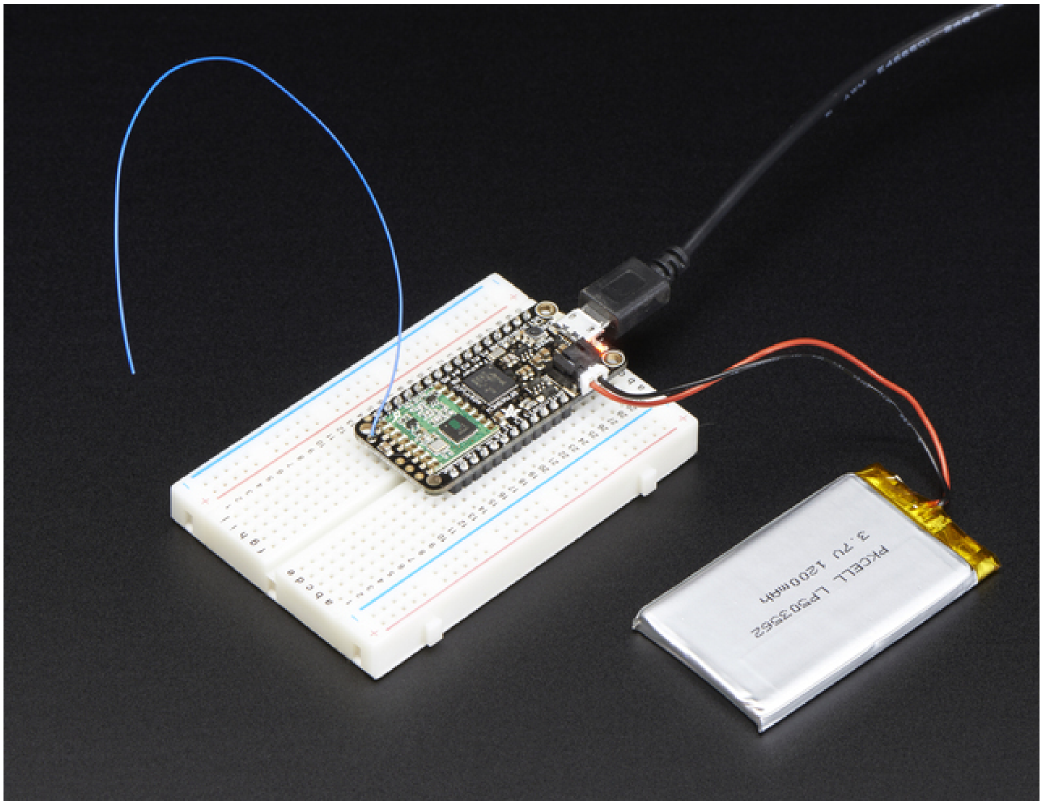
Overview



This is the **Adafruit Feather M0 RFM69 Packet Radio (433 or 900 MHz)**- our take on an microcontroller 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 M0's heart is an ATSAMD21G18 ARM Cortex M0 processor, clocked at 48 MHz and at 3.3V logic, the same one used in the new Arduino Zero (http://adafru.it/2843). This chip has a whopping 256K of FLASH (8x more than the Atmega328 or 32u4) and 32K of RAM (16x as much)! This chip comes with built in USB so it has USB-to-Serial program & debug capability built in with no need for an FTDI-like chip.

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.

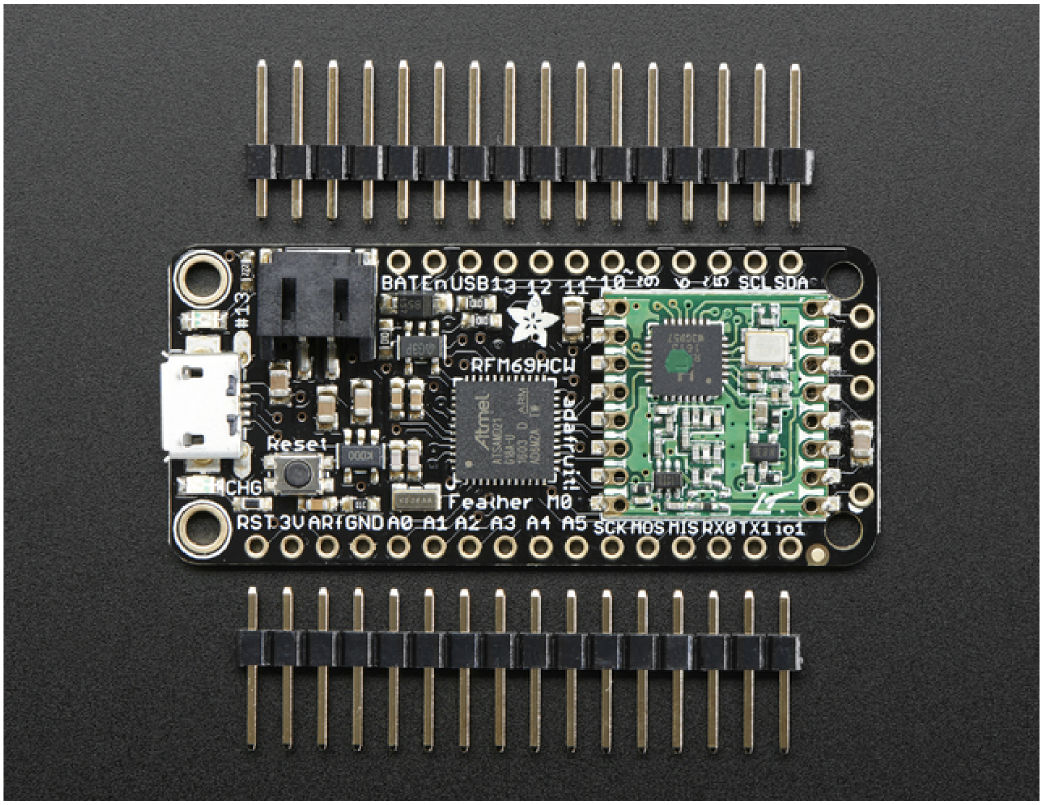


Here's some handy specs! Like all Feather M0's you get:

- Measures 2.0" x 0.9" x 0.3" (51mm x 23mm x 8mm) without headers soldered in
- Light as a (large?) feather - 5.8 grams
- ATSAMD21G18 @ 48MHz with 3.3V logic/power
- No EEPROM
- 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
- 1 x analog output
- 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 M0 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


BAT En USB 13 12 11 10 9 6 5 SCL SDA
#13
RFM69HCW
Reset
CHG
Feather M0
RST 3V ARf GND A0 A1 A2 A3 A4 A5 SCK MOS MIS RX0 TX1 io1

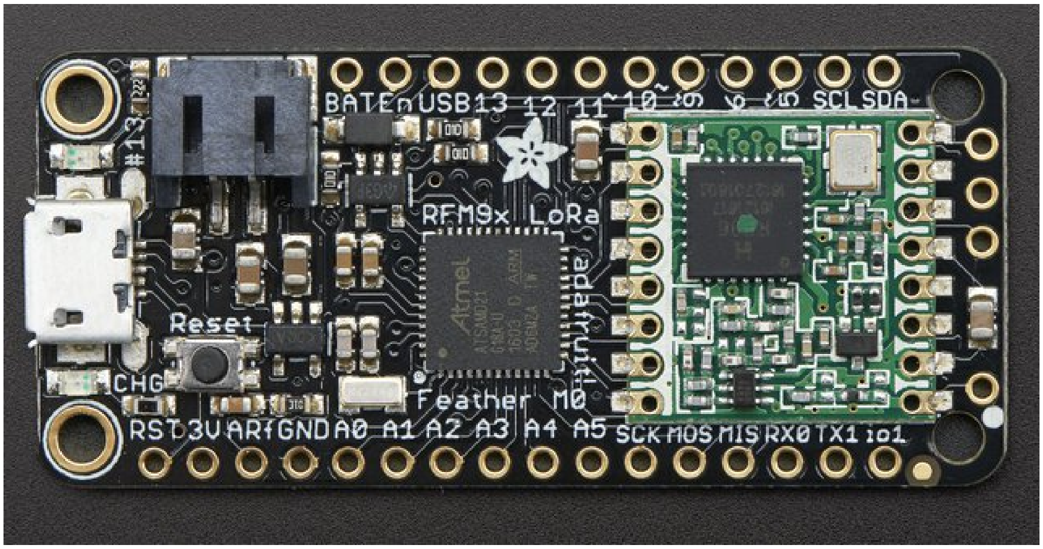
Comes fully assembled and tested, with a USB bootloader that lets you quickly use it with the Arduino IDE. We also toss in some header so you can solder it in and plug into a solderless breadboard. You will need to cut and solder on a small piece of wire (any solid or stranded core is fine) in order 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

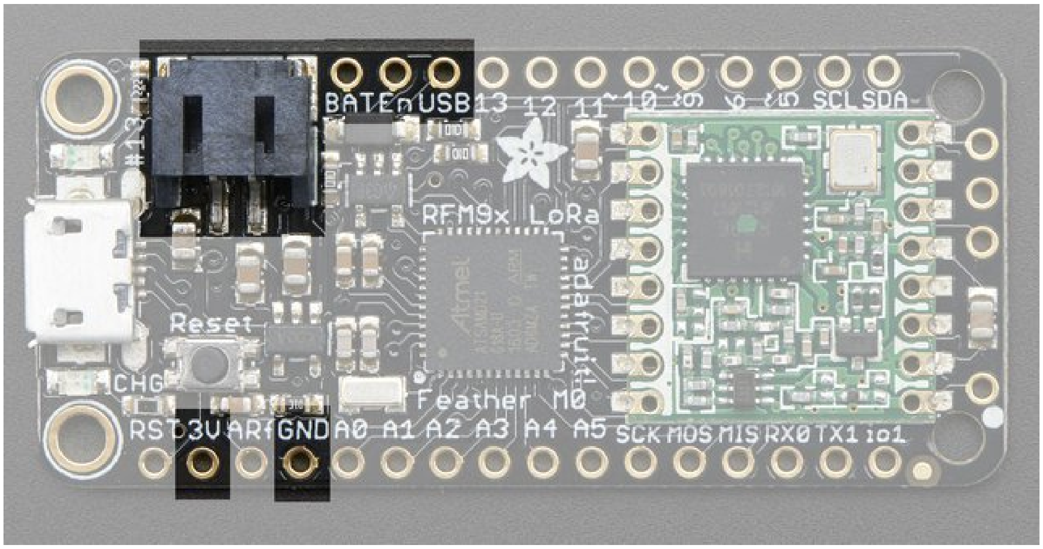
The Feather M0 Radio is chock-full of microcontroller goodness. There's also a lot of pins and ports. We'll take you a tour of them now!

Note that the pinouts are identical for both the Feather M0 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


BAT En USB 13 12 11 10 9 6 5 SCL SDA
#13
RFM9x LoRa
adafruit
Reset
CHG
Feather M0
RST 3V ARf GND A0 A1 A2 A3 A4 A5 SCK MOS MIS RX0 TX1 io1

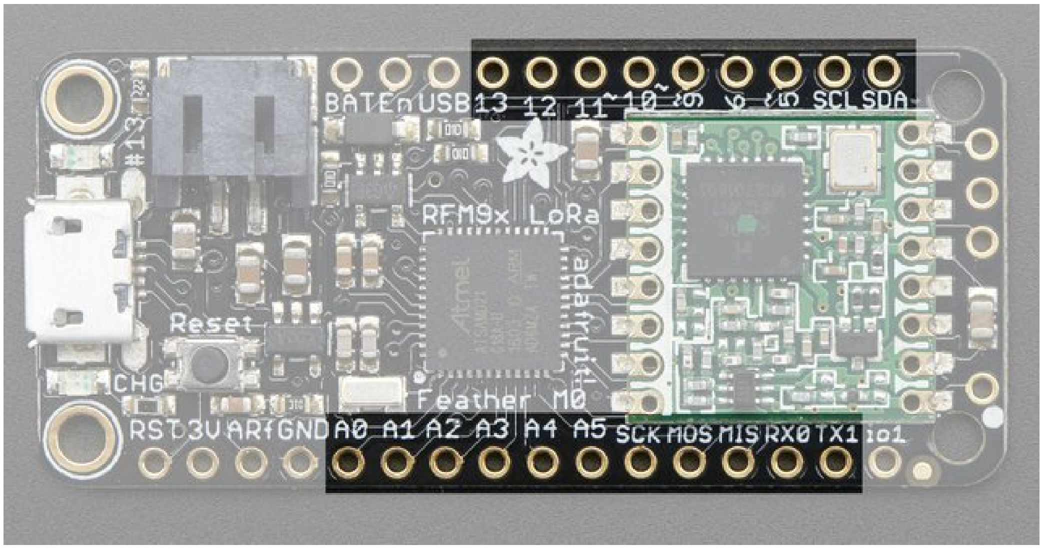
Power Pins


BAT En USB 13 12 11 10 9 6 5 SCL SDA
#13
RFM9x LoRa
adafruit
Reset
CHG
Feather M0
RST 3V ARf GND A0 A1 A2 A3 A4 A5 SCK MOS MIS RX0 TX1 io1

- **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

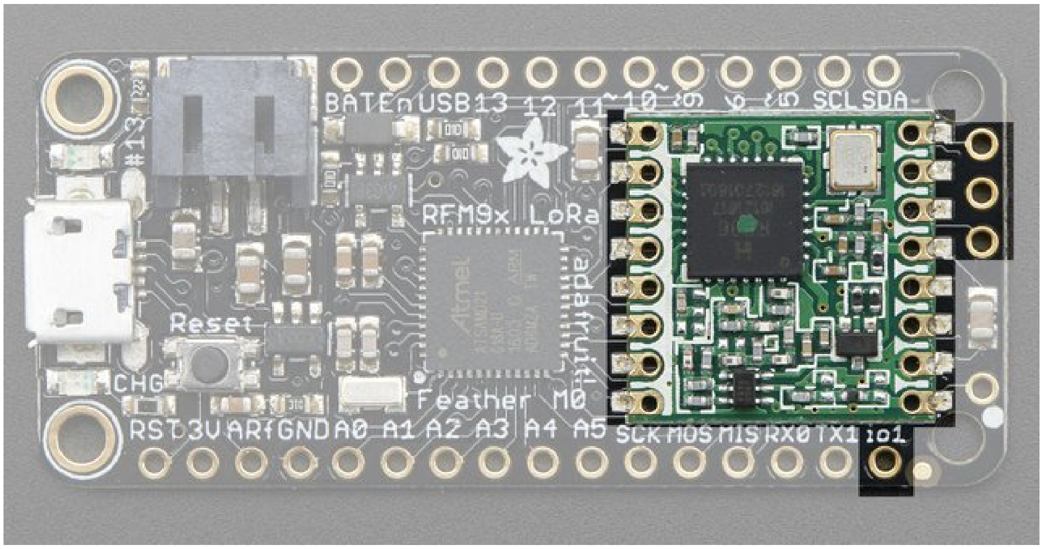
Logic pins



This is the general purpose I/O pin set for the microcontroller.
All logic is 3.3V
All pins can do PWM output
All pins can be interrupt inputs

- **#0** / **RX** - GPIO #0, also receive (input) pin for**Serial1** (hardware UART), also can be analog input
- **#1** / **TX** - GPIO #1, also transmit (output) pin for**Serial1**, also can be analog input
- **#20** / **SDA** - GPIO #20, 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.
- **#21** / **SCL** - GPIO #21, 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.
- **#5** - GPIO #5
- **#6** - GPIO #6
- **#9** - GPIO #9, also analog input**A7.** 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
- **#11** - GPIO #11
- **#12** - GPIO #12
- **#13** - GPIO #13 and is connected to the**red LED** next to the USB jack
- **A0** - This pin is analog *input* **A0** but is also an analog *output* due to having a DAC (digital-to-analog converter). You can set the raw voltage to anything from 0 to 3.3V, unlike PWM outputs this is a true analog output
- **A1 thru A5** - These are each analog input as well as digital I/O pins.
- **SCK**/**MOSI**/**MISO** (GPIO **24**/**23**/**22**)- These are the hardware SPI pins, you can use them as everyday GPIO pins (but recommend keeping them free as they are best used for hardware SPI connections for high speed.

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

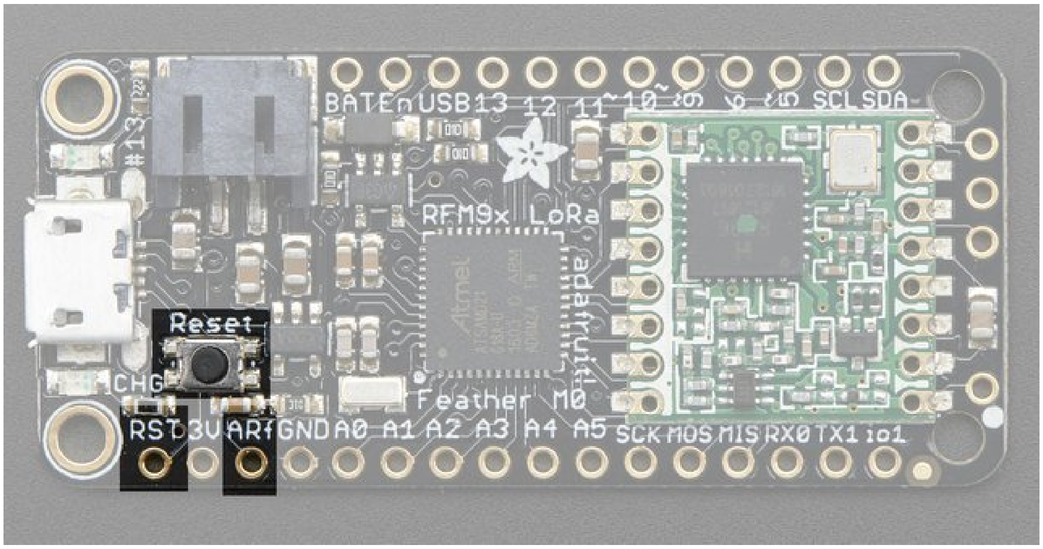
- **#8** - used as the radio **CS** (chip select) pin
- **#3** - used as the radio **GPIO0** / **IRQ** (interrupt request) pin.
- **#4** - used as the radio **Reset** pin

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, IO3** and **IO5**). You probably wont need these for most uses of the Feather but they are available in case you need 'em!

The CS pin (#8) does not have a pullup built in so be sure to set this pin HIGH when not using the radio!

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

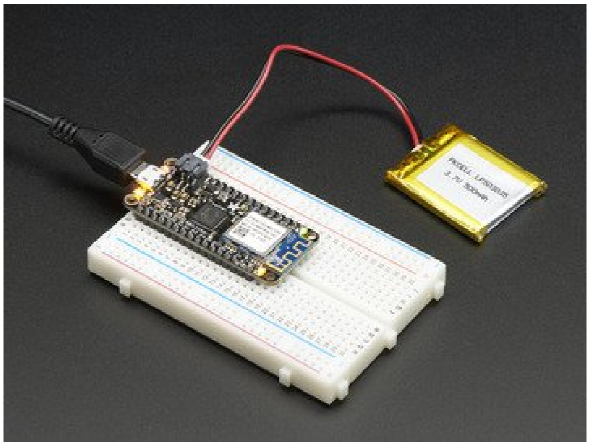
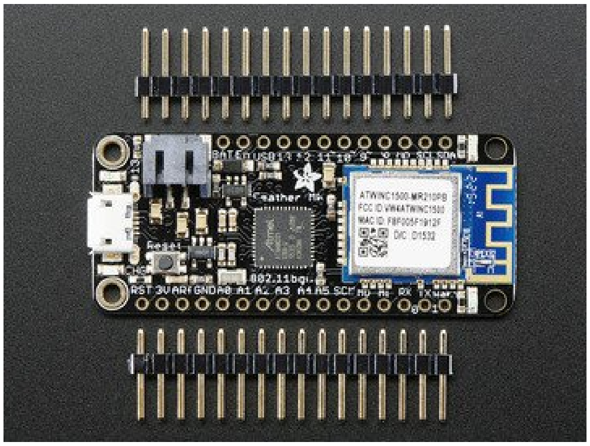
firmware. Can't go higher than 3.3V!

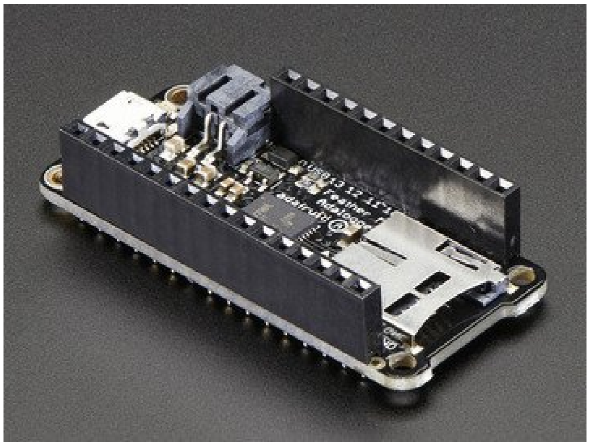
Assembly

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

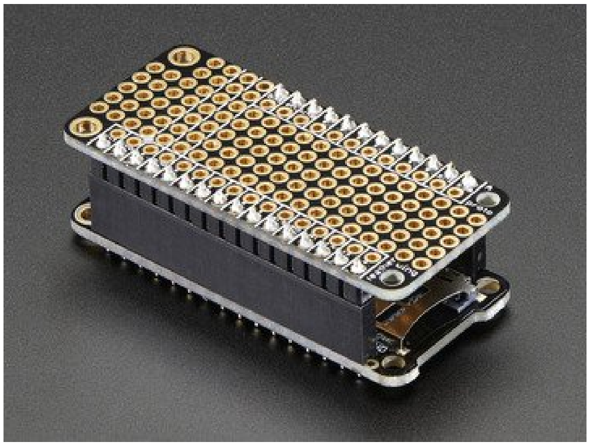
Header Options!

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

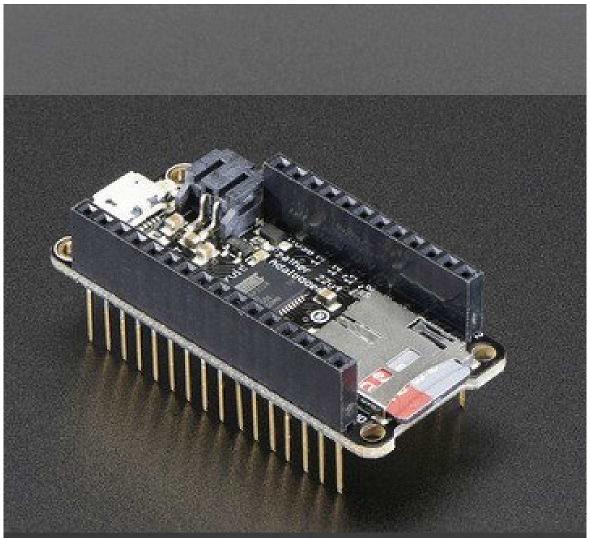
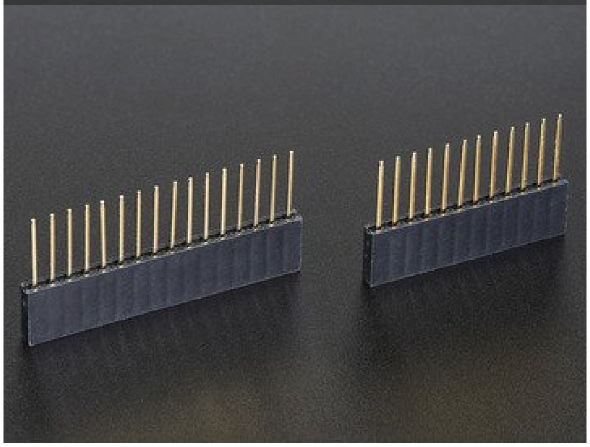
-  The first option is soldering in plain male headers, this lets you plug in the Feather into a solderless breadboard
- 

- 

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

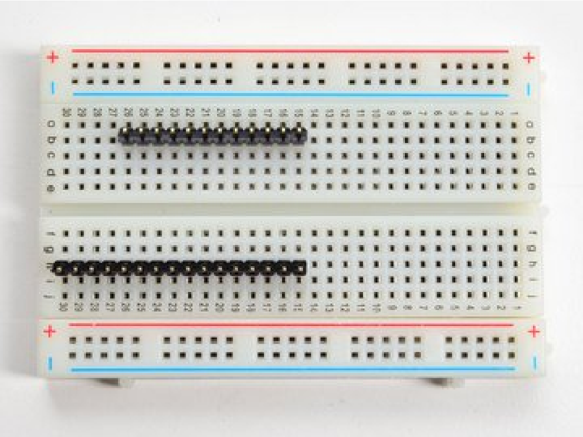
- 

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

- 
- 

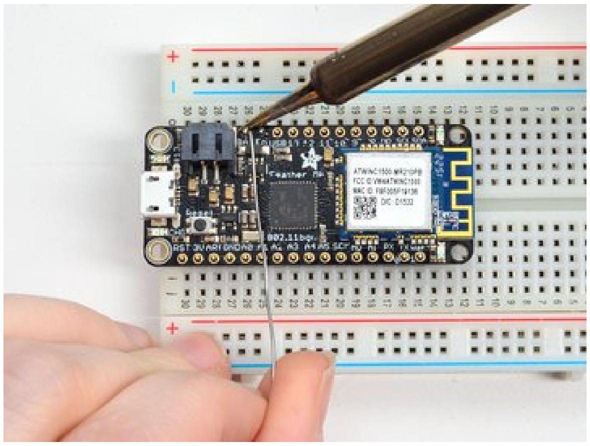
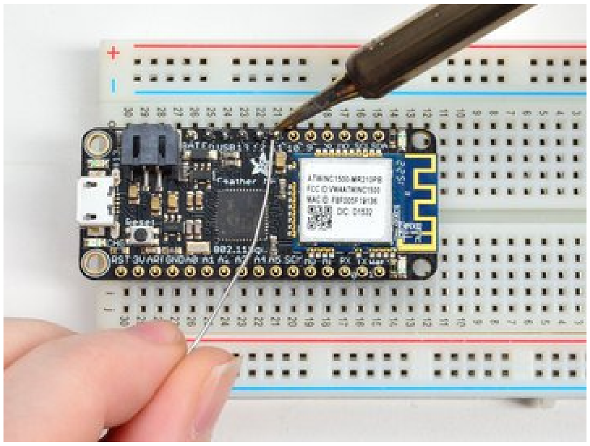
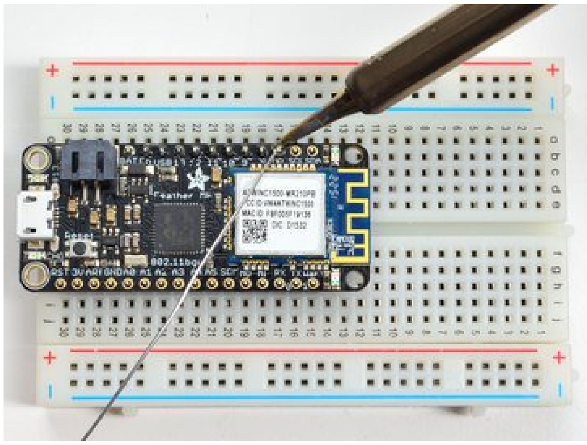
Finally, there's the "Stacking Header" option. This one is sort of the best-of-both-worlds. You get the ability to plug into a solderless breadboard *and* plug a featherwing on top. But its a little bulky

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**

- 
- 
- 

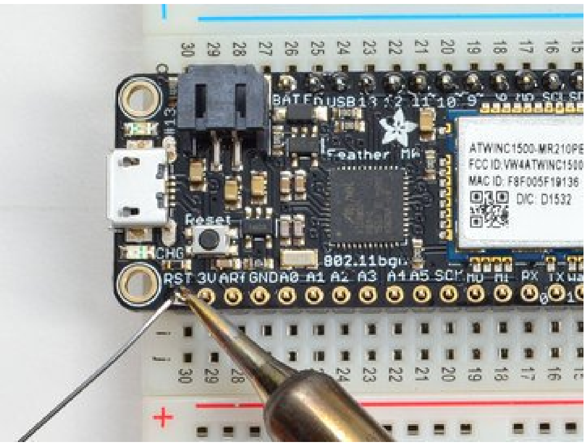
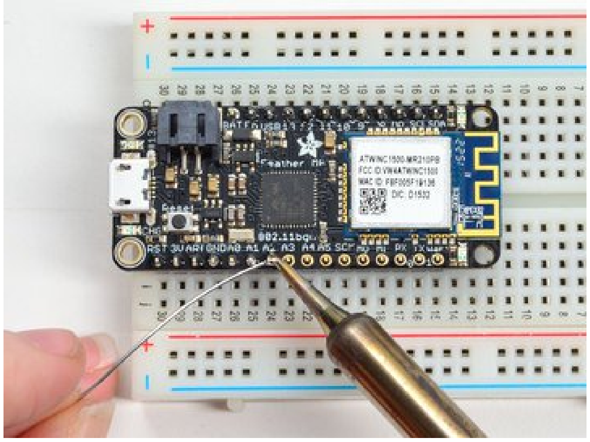
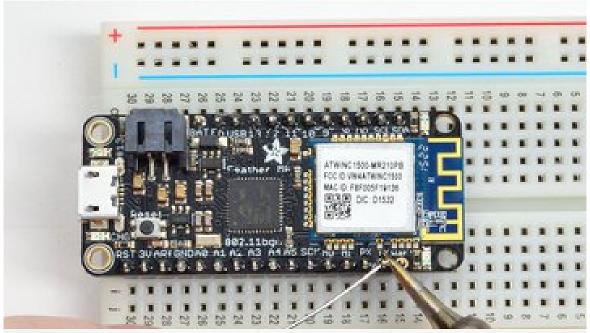
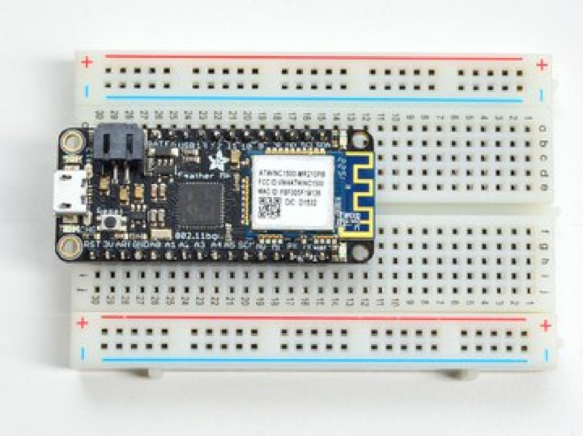
Add the breakout board:

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

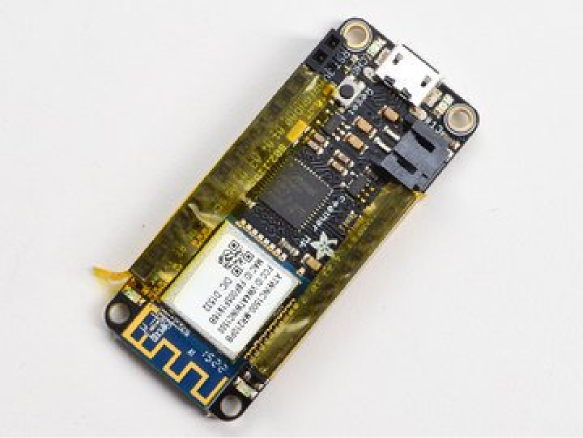
And Solder!

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) (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

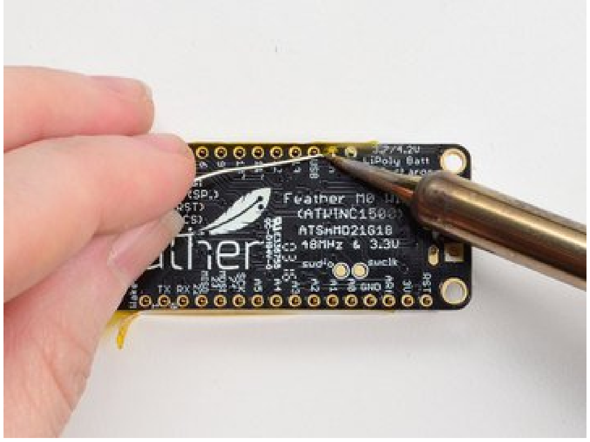
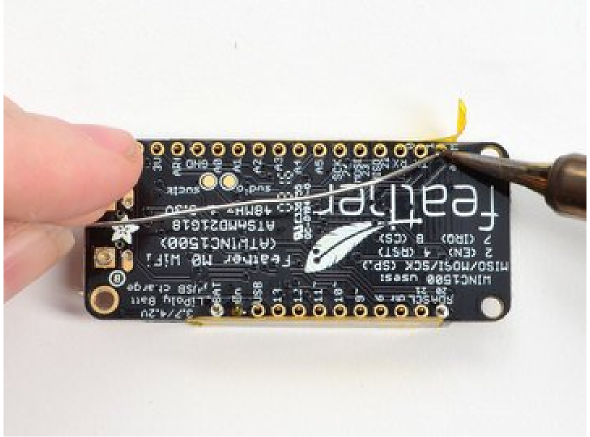
- 

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

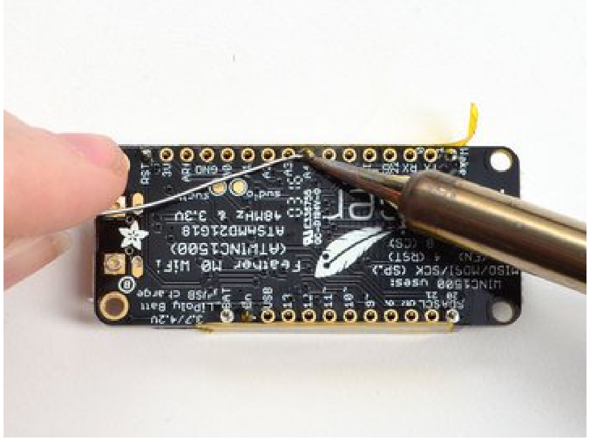
After flipping over, solder one or two points on each strip, to 'tack' the header in place

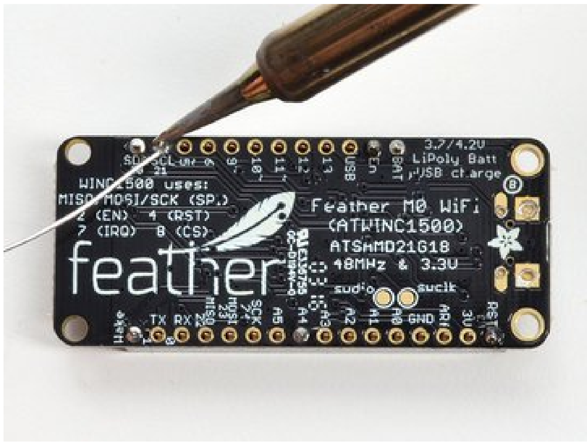
- 
-
- 

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) (http://adafru.it/aTk)).

- 

- 

-

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

-
WINC1500 uses:
MISO/MOSI/SCK (SPI)
2 (EN) 4 (RST)
7 (IRQ) 8 (CS)
feather
Feather M0 WiFi
(ATWINC1500)
ATSAMD21G18
48MHz & 3.3V
3.7/4.2V
LiPoly Batt
USB charge
-

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

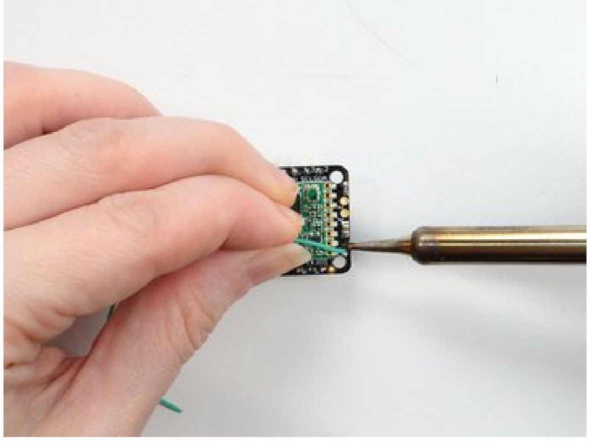
Wire Antenna

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.

-

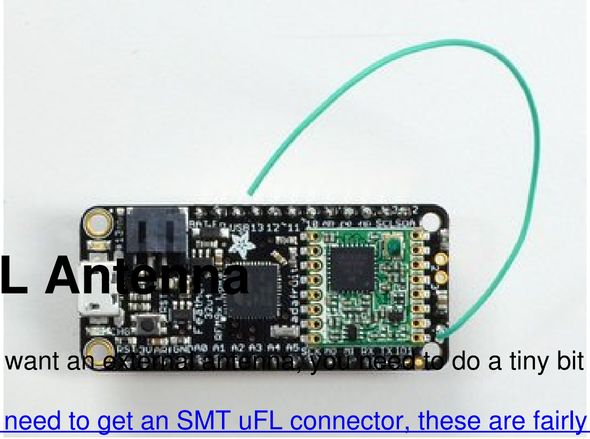
  Cut a stranded or solid core wire the the proper length for the module/frequency

  - **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

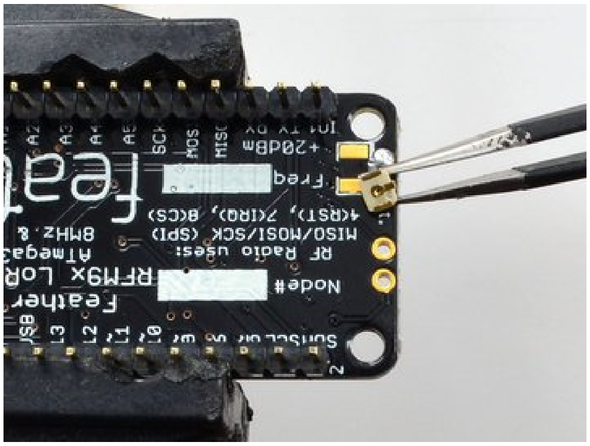
If you want an external antenna, you need to do a tiny bit more work but its not too difficult.

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

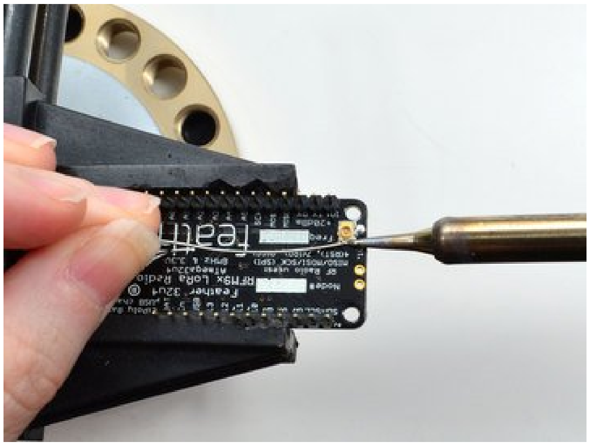
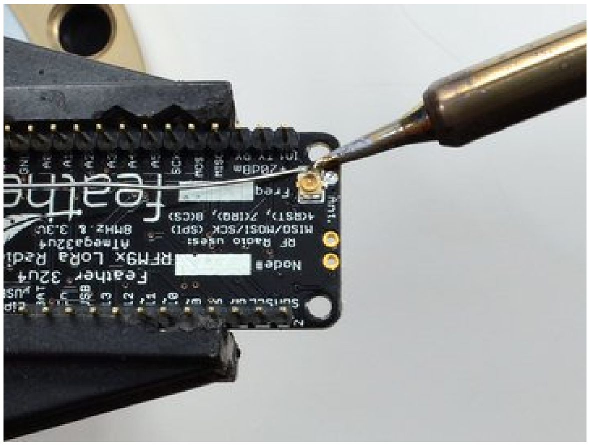
You'll also need a uFL to SMA adapter (http://adafru.it/851) (or whatever adapter you need for the antenna you'll be using, SMA is the most common

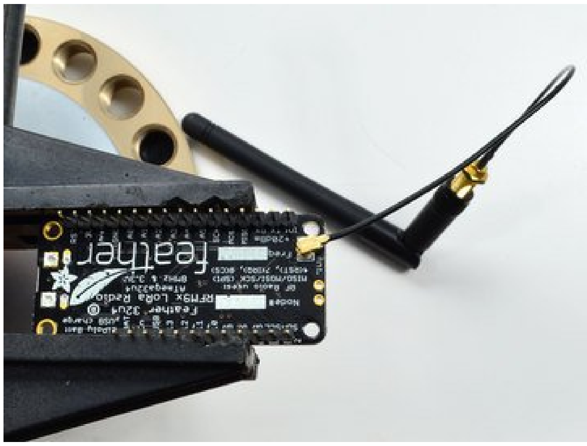
Of course, you will also need an antenna of some sort, that matches your radio frequency

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!

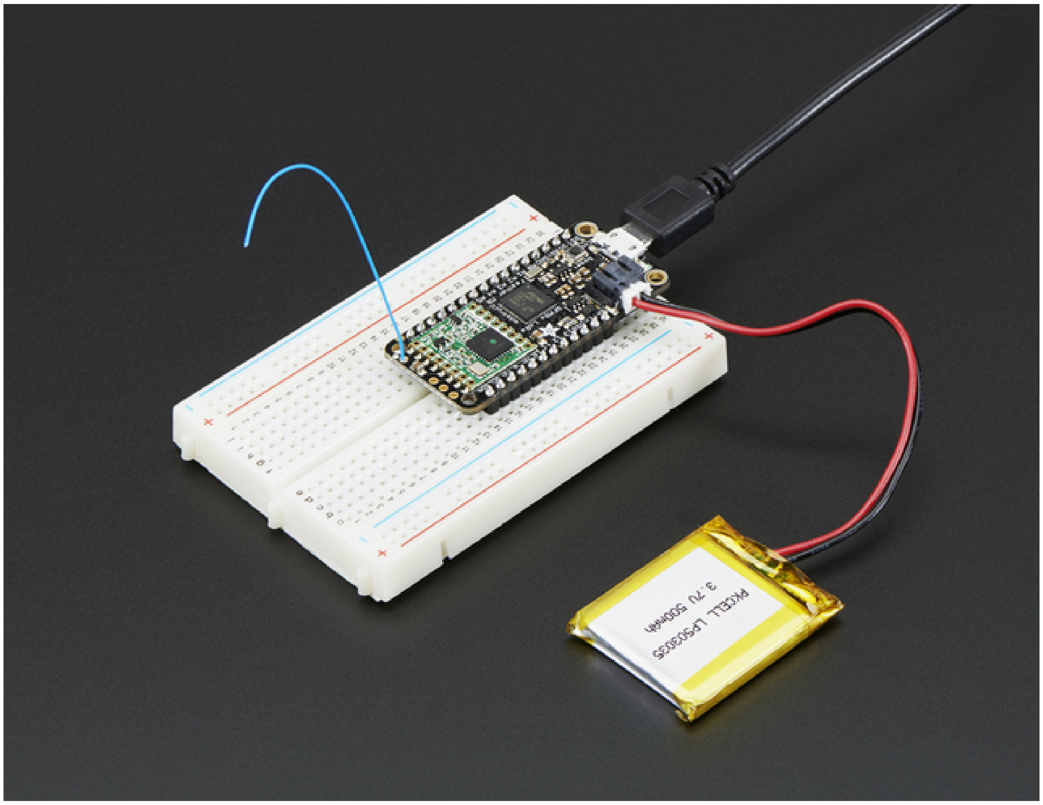
- 

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!

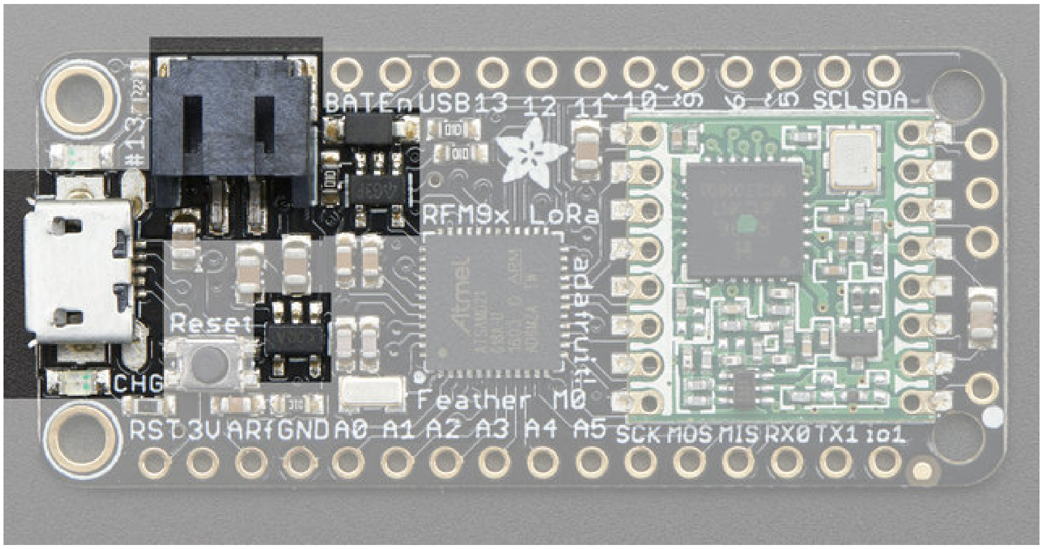
Power Management



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.

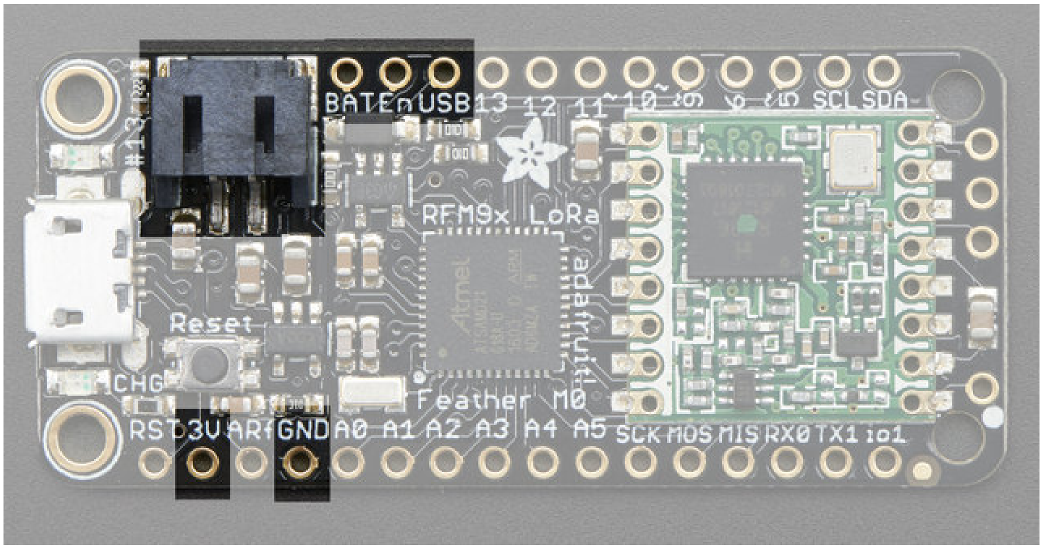
The JST connector polarity is matched to Adafruit LiPoly batteries. Using wrong polarity batteries can destroy your Feather


BAT En USB 13 12 11~ 10~ ~9 ~6 ~5 SCL SDA
#13
RFM9x LoRa
adafruit
Reset
CHG
Feather M0
RST 3V ARf GND A0 A1 A2 A3 A4 A5 SCK MOS MIS RX0 TX1 io1

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 regulator. 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 'spikey' & sporadic.

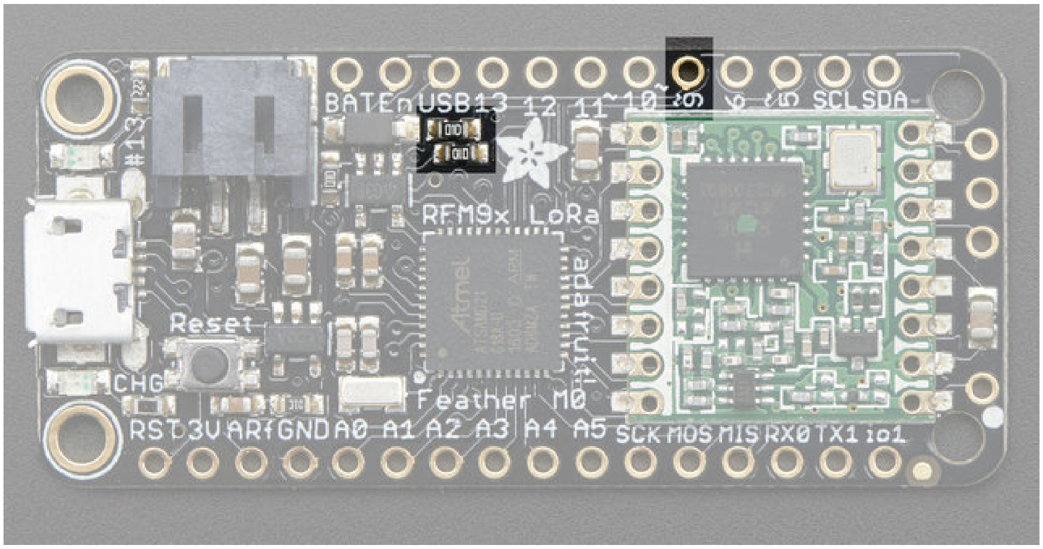

BAT En USB 13 12 11~ 10~ ~9 ~6 ~5 SCL SDA
#13
RFM9x LoRa
adafruit
Reset
CHG
Feather M0
RST 3V ARf GND A0 A1 A2 A3 A4 A5 SCK MOS MIS RX0 TX1 io1

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

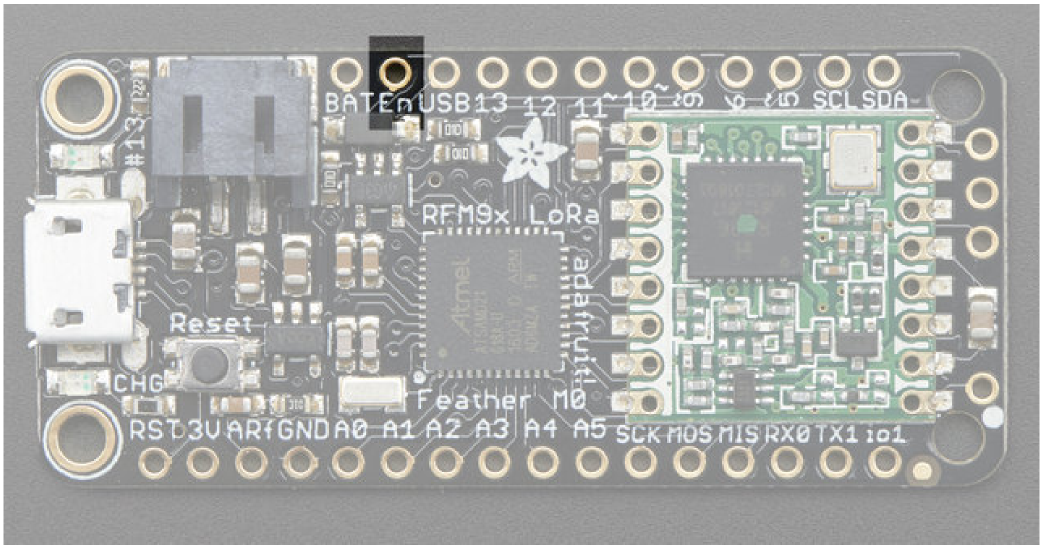
To make this easy we stuck a double-100K resistor divider on the**BAT** pin, and connected it to**D9** (a.k.a analog #7 **A7**). You can read this pin's voltage, then double it, to get the battery voltage.

```
#define VBATPIN A7

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);
```



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



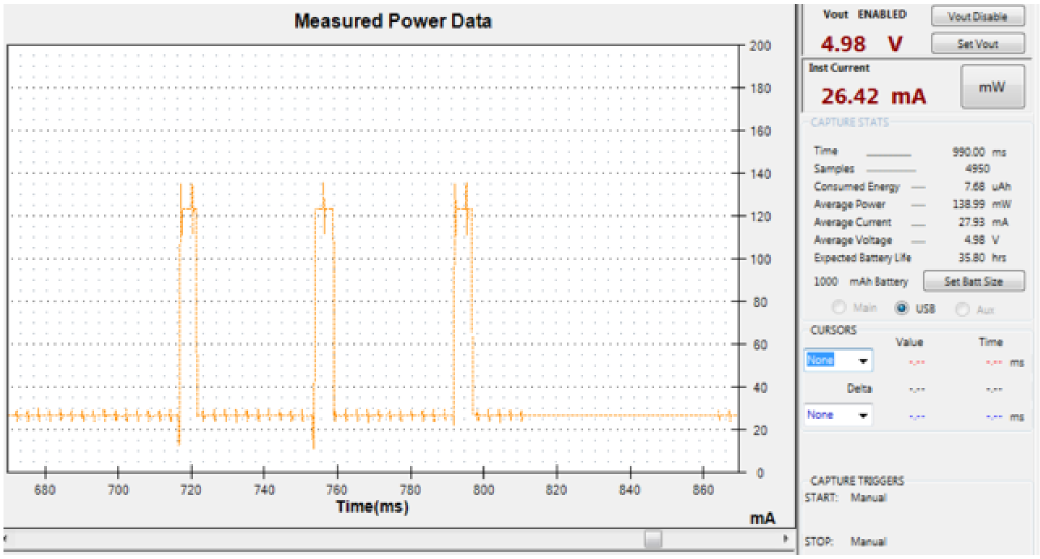
Radio Power Draw

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 M0 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!



Vout ENABLED
4.98 V
Inst Current
26.42 mA

CAPTURE STATS

```
Time                    990.00  ms
Samples                 4950
Consumed Energy         7.68    uAh
Average Power           138.99  mW
Average Current         27.93   mA
Average Voltage         4.98    V
Expected Battery Life   35.80   hrs
1000 mAh Battery
```

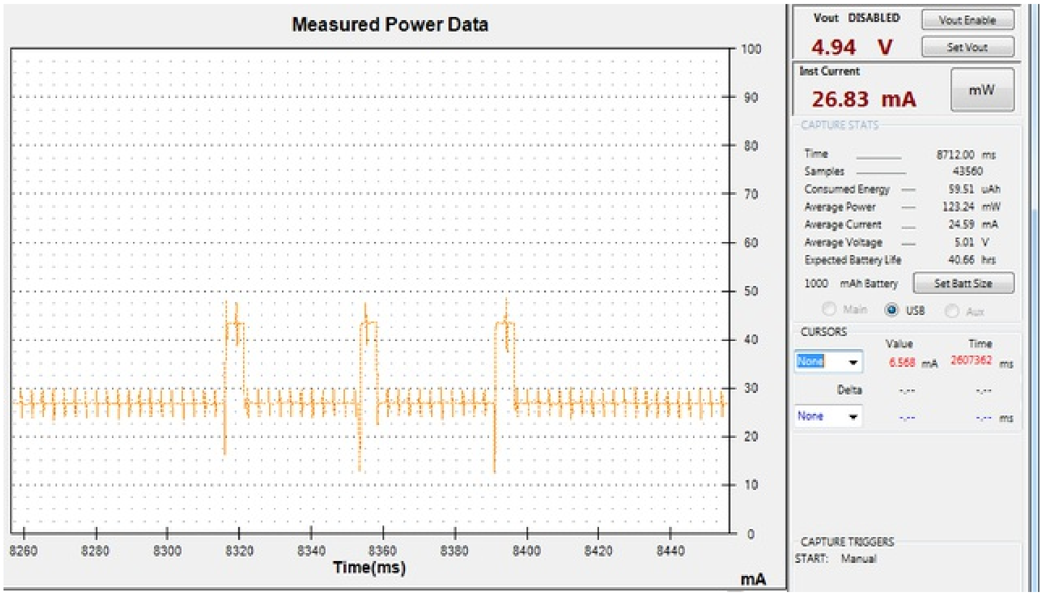
○ Main ● USB ○ Aux

CURSORS

	Value	Time
None	-.--	-.-- ms
Delta	-.--	-.--
None	-.--	-.-- ms

CAPTURE TRIGGERS
START: Manual
STOP: Manual

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



Vout DISABLED
4.94 V
Inst Current
26.83 mA

CAPTURE STATS

```
Time                    8712.00 ms
Samples                 43560
Consumed Energy         59.51   uAh
Average Power           123.24  mW
Average Current         24.59   mA
Average Voltage         5.01    V
Expected Battery Life   40.66   hrs
1000 mAh Battery
```

○ Main ● USB ○ Aux

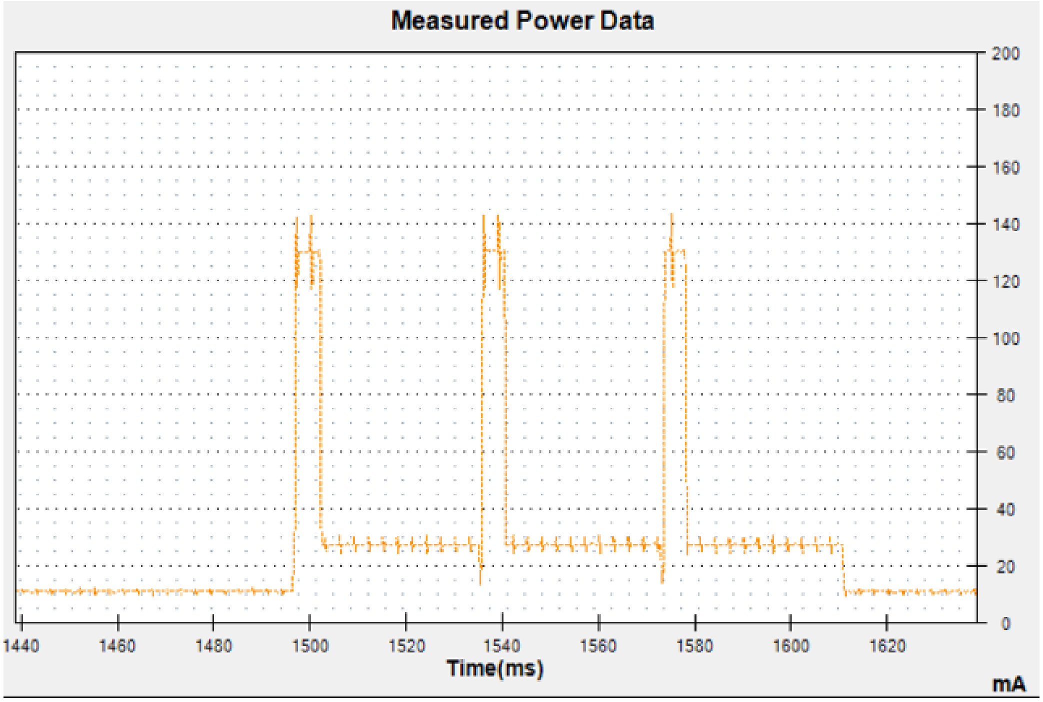
CURSORS

	Value	Time
None	6.568 mA	2607362 ms
Delta	-.--	-.--
None	-.--	-.-- ms

CAPTURE TRIGGERS
START: Manual

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 withradio.sleep(); 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

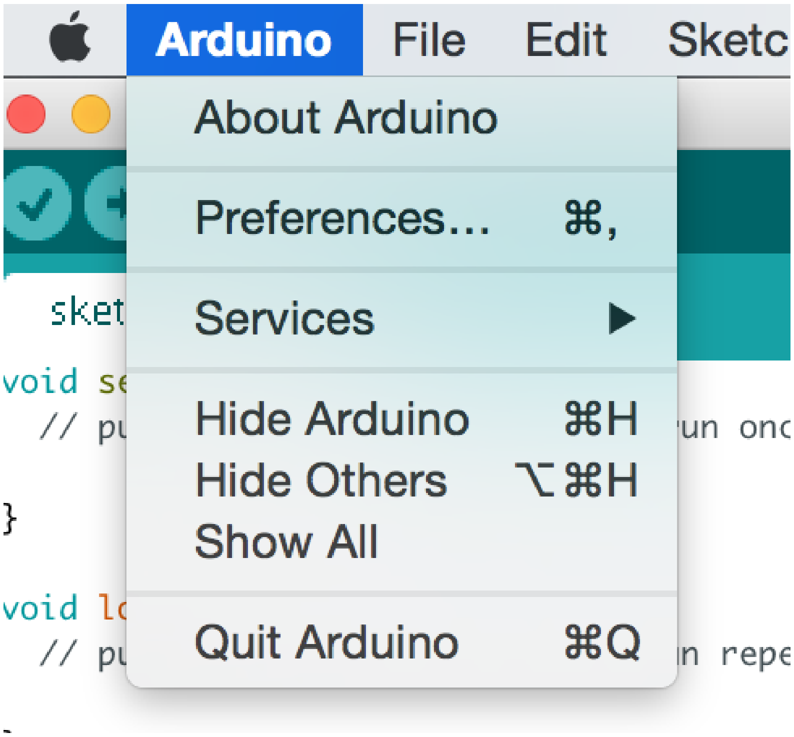

Measured Power Data
200
180
160
140
120
100
80
60
40
20
0
1440
1460
1480
1500
1520
1540
1560
1580
1600
1620
Time(ms)
mA

Arduino IDE Setup

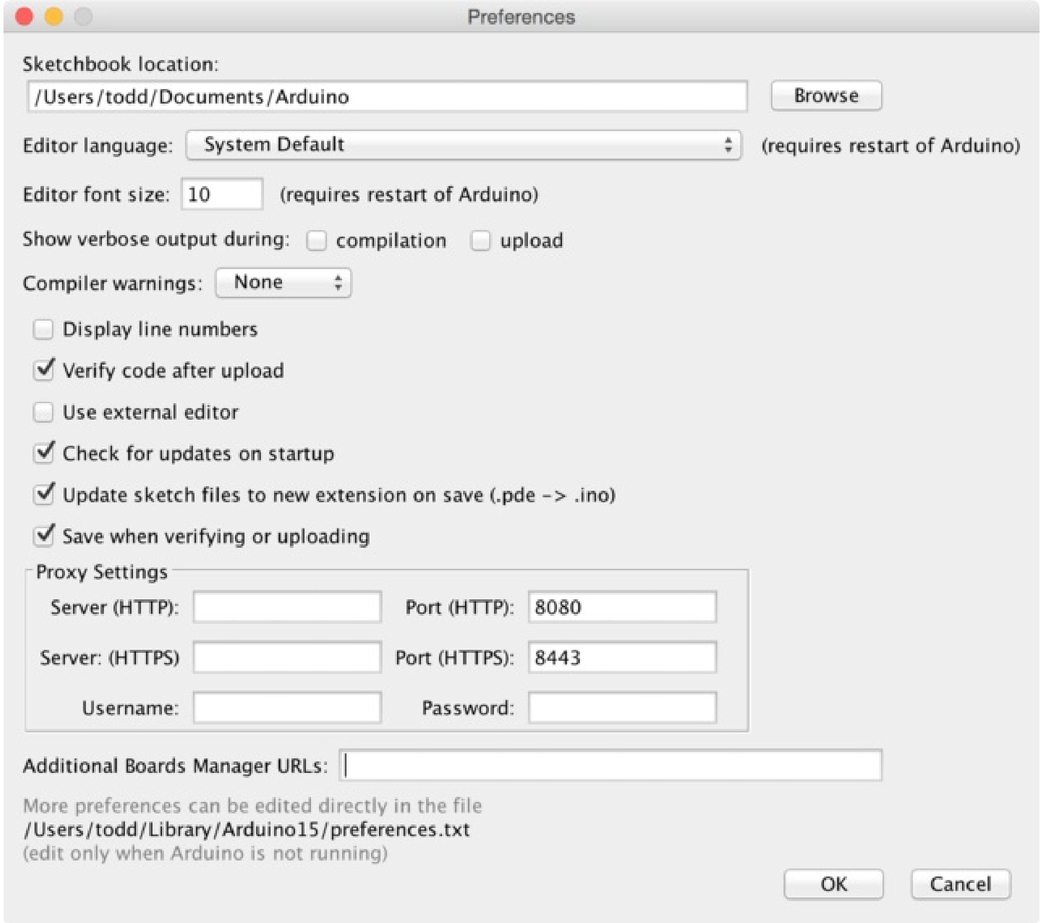
The first thing you will need to do is to download the latest release of the Arduino IDE. You will need to be using **version 1.6.4** or higher for this guide.

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.



Sketchbook location:
/Users/todd/Documents/Arduino

Editor language: System Default (requires restart of Arduino)

Editor font size: 10 (requires restart of Arduino)

Show verbose output during: ☐ compilation ☐ upload

Compiler warnings: None

☐ Display line numbers

☑ Verify code after upload

☐ Use external editor

☑ Check for updates on startup

☑ Update sketch files to new extension on save (.pde -> .ino)

☑ Save when verifying or uploading

Proxy Settings

Server (HTTP): Port (HTTP): 8080

Server: (HTTPS) Port (HTTPS): 8443

Username: Password:

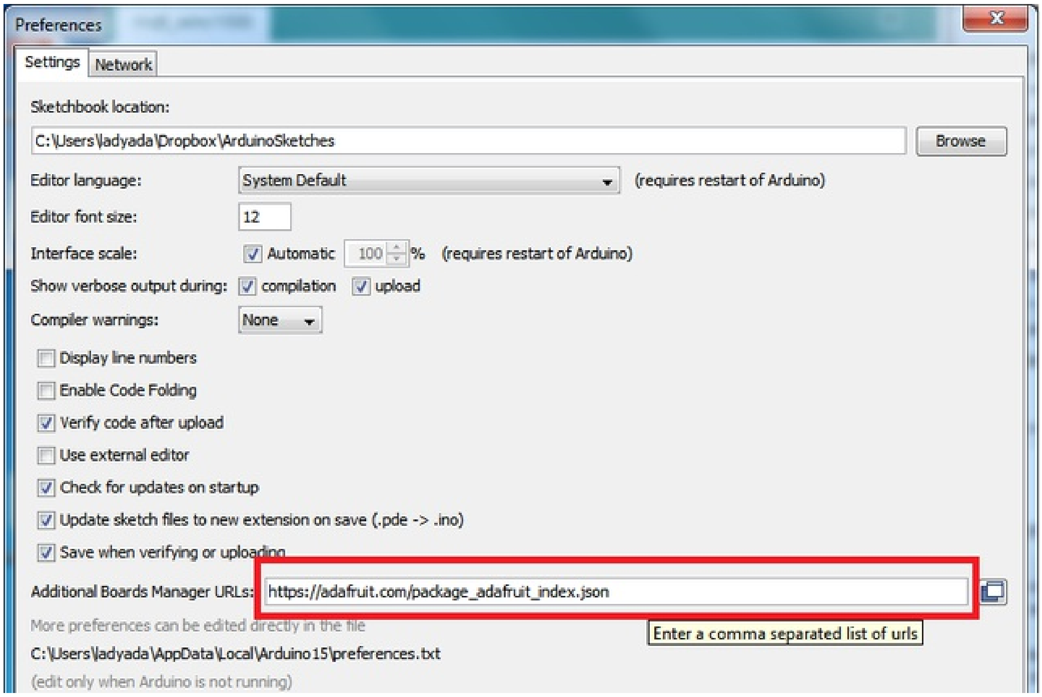
Additional Boards Manager URLs:

More preferences can be edited directly in the file
/Users/todd/Library/Arduino15/preferences.txt
(edit only when Arduino is not running)

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



Sketchbook location: C:\Users\ladyada\Dropbox\ArduinoSketches

Editor language: System Default (requires restart of Arduino)

Editor font size: 12

Interface scale: ☑ Automatic 100 % (requires restart of Arduino)

Show verbose output during: ☑ compilation ☑ upload

Compiler warnings: None

□ Display line numbers

□ Enable Code Folding

☑ Verify code after upload

□ Use external editor

☑ Check for updates on startup

☑ Update sketch files to new extension on save (.pde -> .ino)

☑ Save when verifying or uploading

Additional Boards Manager URLs: https://adafruit.com/package_adafruit_index.json

Enter a comma separated list of urls

More preferences can be edited directly in the file

C:\Users\ladyada\AppData\Local\Arduino15\preferences.txt

(edit only when Arduino is not running)

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).

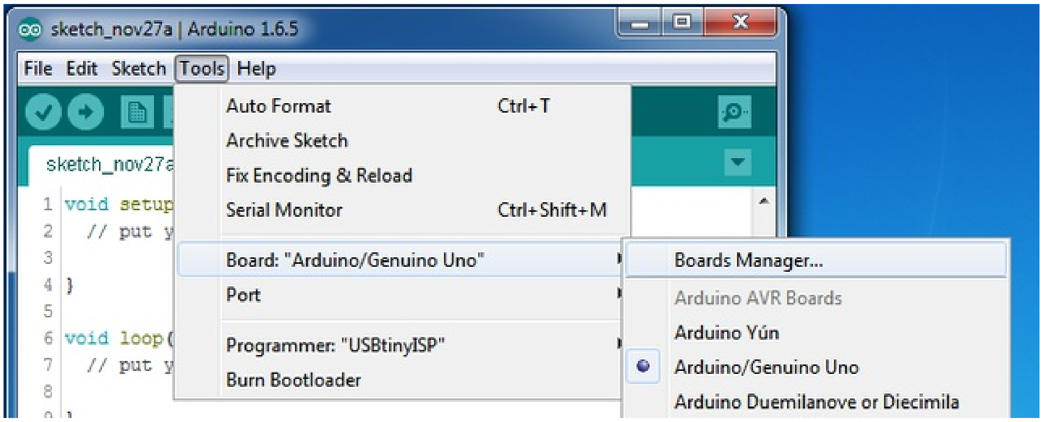
If you have multiple boards you want to support, say ESP8266 and Adafruit, have both URLs in the text box separated by a comma (,)

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 M0 uses an ATSAMD21 chip running at 48 MHz, you can pretty easily get it working with the Arduino IDE. Most libraries (including the popular ones like NeoPixels and display) will work with the M0, especially devices & sensors that use i2c or SPI.

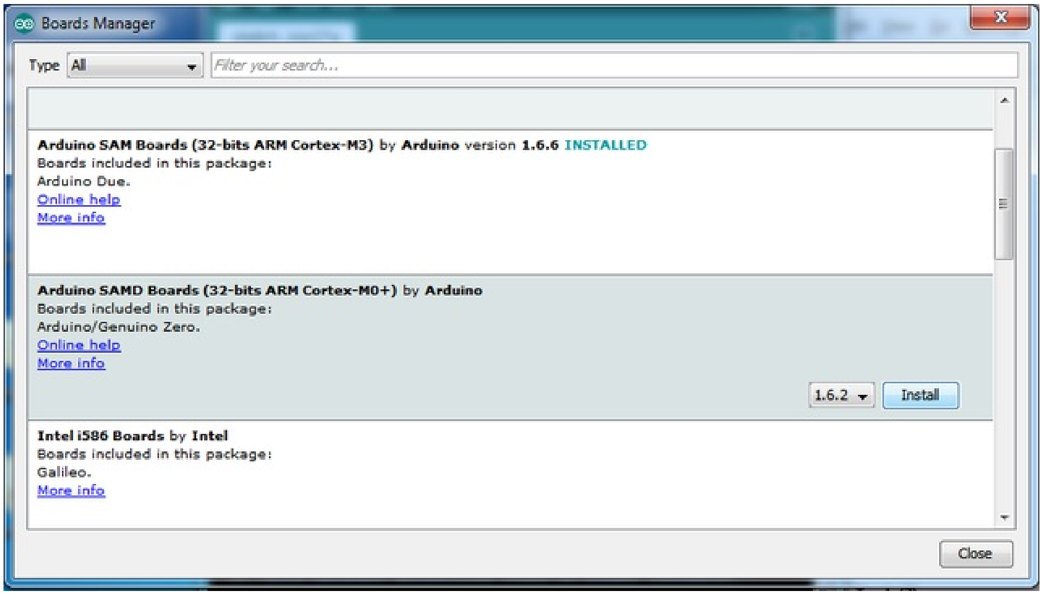
Now that you have added the appropriate URLs to the Arduino IDE preferences, you can open the**Boards Manager** by navigating to the **Tools->Board** menu.



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.

Install SAMD Support

First up, install the **Arduino SAMD Boards** version **1.6.2**
or later



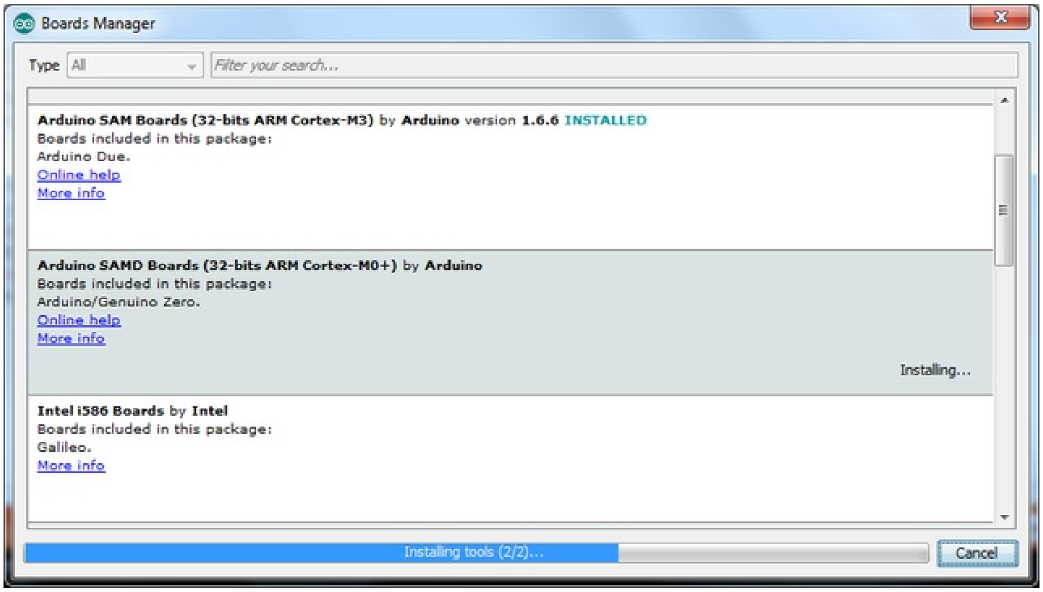
Boards Manager

Type All | Filter your search...

Arduino SAM Boards (32-bits ARM Cortex-M3) by **Arduino** version **1.6.6** INSTALLED
Boards included in this package:
Arduino Due.
Online help
More info

Arduino SAMD Boards (32-bits ARM Cortex-M0+) by **Arduino**
Boards included in this package:
Arduino/Genuino Zero.
Online help
More info

Intel i586 Boards by **Intel**
Boards included in this package:
Galileo.
More info



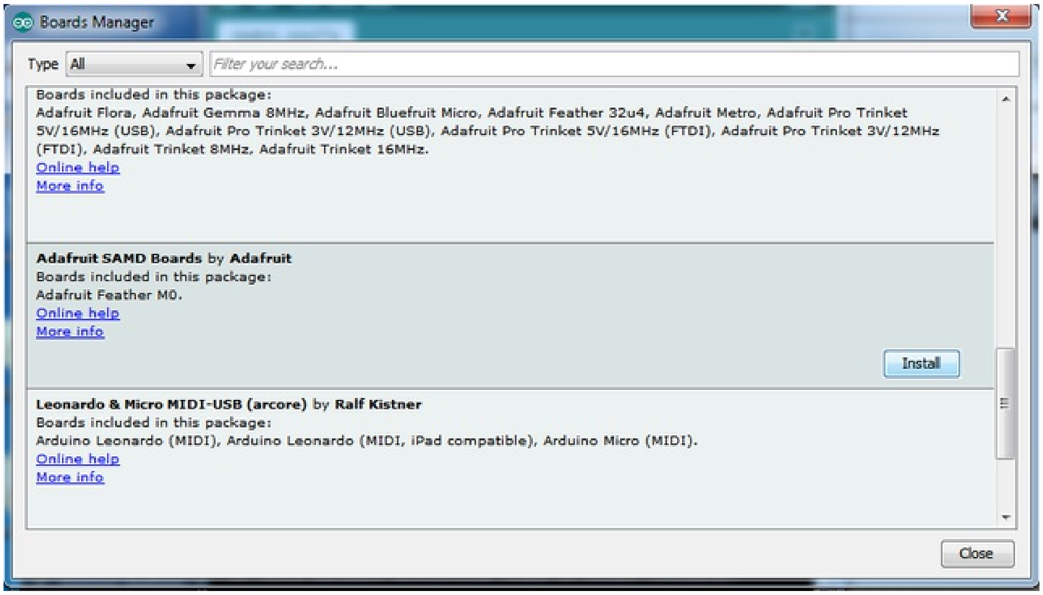
Arduino SAM Boards (32-bits ARM Cortex-M3) by **Arduino** version **1.6.6** INSTALLED
Boards included in this package:
Arduino Due.
Online help
More info

Arduino SAMD Boards (32-bits ARM Cortex-M0+) by **Arduino**
Boards included in this package:
Arduino/Genuino Zero.
Online help
More info

Intel i586 Boards by **Intel**
Boards included in this package:
Galileo.
More info

Install Adafruit SAMD

Next you can install the Adafruit SAMD package to add the board file definitions



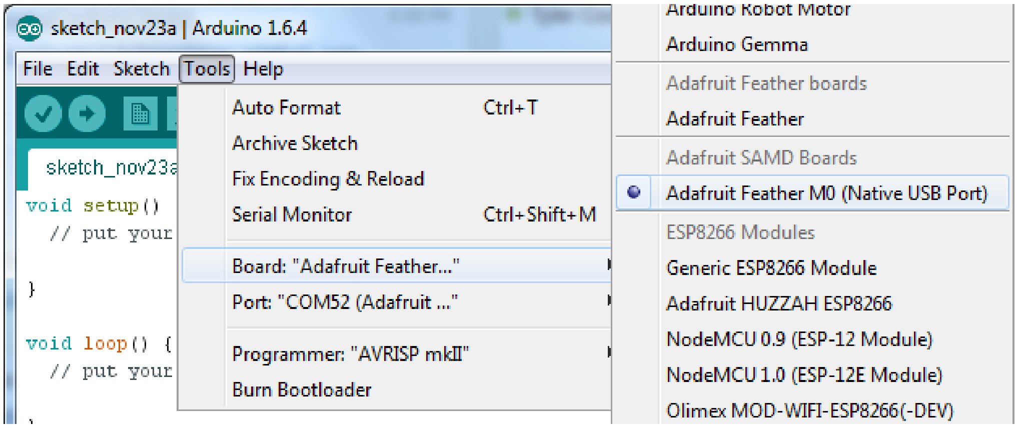
Boards included in this package:
Adafruit Flora, Adafruit Gemma 8MHz, Adafruit Bluefruit Micro, Adafruit Feather 32u4, Adafruit Metro, Adafruit Pro Trinket 5V/16MHz (USB), Adafruit Pro Trinket 3V/12MHz (USB), Adafruit Pro Trinket 5V/16MHz (FTDI), Adafruit Pro Trinket 3V/12MHz (FTDI), Adafruit Trinket 8MHz, Adafruit Trinket 16MHz.
Online help
More info

Adafruit SAMD Boards by **Adafruit**
Boards included in this package:
Adafruit Feather M0.
Online help
More info

Leonardo & Micro MIDI-USB (arcore) by **Ralf Kistner**
Boards included in this package:
Arduino Leonardo (MIDI), Arduino Leonardo (MIDI, iPad compatible), Arduino Micro (MIDI).
Online help
More info

Even though in theory you don't need to - I recommend rebooting the IDE

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.


sketch_nov23a | Arduino 1.6.4
File
Edit
Sketch
Tools
Help
sketch_nov23a
Auto Format Ctrl+T
Archive Sketch
Fix Encoding & Reload
Serial Monitor Ctrl+Shift+M
Board: "Adafruit Feather..."
Port: "COM52 (Adafruit ..."
Programmer: "AVRISP mkII"
Burn Bootloader
Arduino Robot Motor
Arduino Gemma
Adafruit Feather boards
Adafruit Feather
Adafruit SAMD Boards
● Adafruit Feather M0 (Native USB Port)
ESP8266 Modules
Generic ESP8266 Module
Adafruit HUZZAH ESP8266
NodeMCU 0.9 (ESP-12 Module)
NodeMCU 1.0 (ESP-12E Module)
Olimex MOD-WIFI-ESP8266(-DEV)

```
void setup()
  // put your
}

void loop() {
  // put your
```

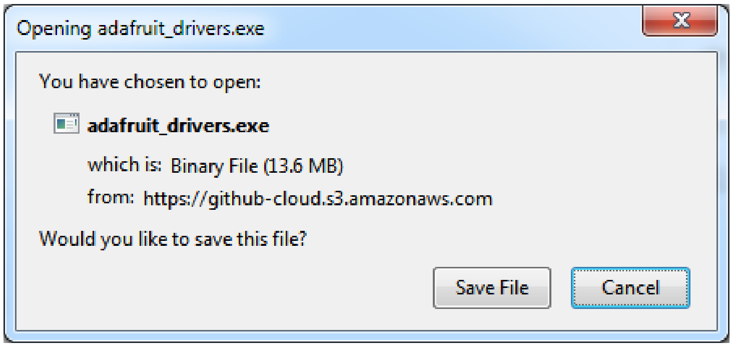
Install Drivers (Windows Only)

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

Click below to download our Driver Installer

Download Adafruit Driver Installer
http://adafru.it/mai

Download and run the installer



Opening adafruit_drivers.exe

You have chosen to open:

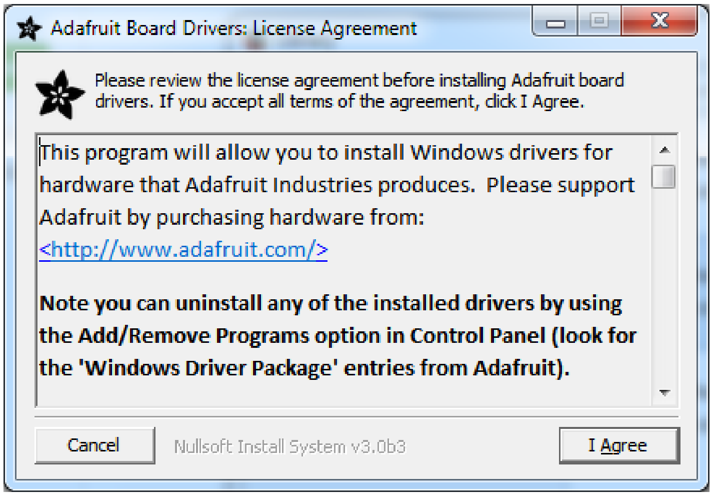
adafruit_drivers.exe

which is: Binary File (13.6 MB)

from: https://github-cloud.s3.amazonaws.com

Would you like to save this file?

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



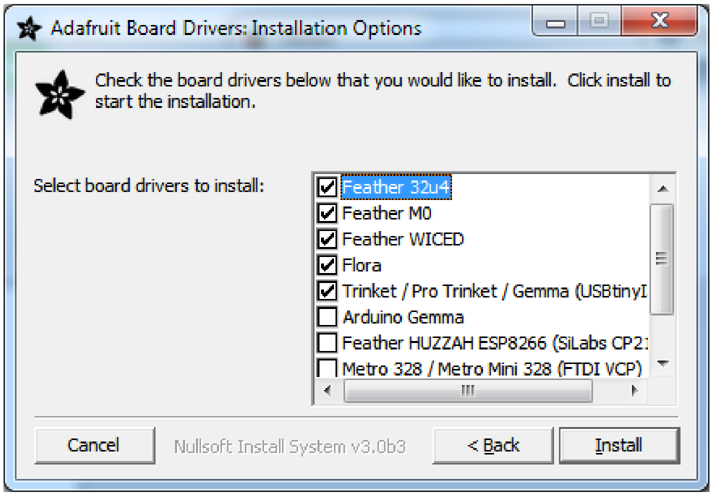
Adafruit Board Drivers: License Agreement

Please review the license agreement before installing Adafruit board drivers. If you accept all terms of the agreement, click I Agree.

This program will allow you to install Windows drivers for hardware that Adafruit Industries produces. Please support Adafruit by purchasing hardware from: <http://www.adafruit.com/>

Note you can uninstall any of the installed drivers by using the Add/Remove Programs option in Control Panel (look for the 'Windows Driver Package' entries from Adafruit).

Select which drivers you want to install:



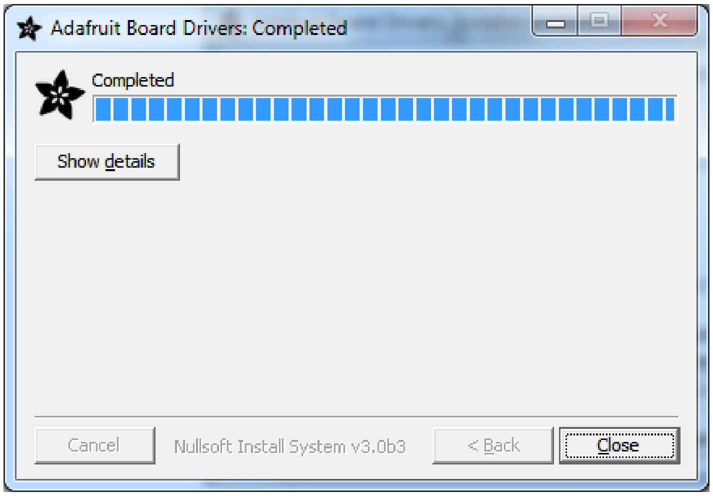
Adafruit Board Drivers: Installation Options

Check the board drivers below that you would like to install. Click install to start the installation.

Select board drivers to install:

- ☑ Feather 32u4
- ☑ Feather M0
- ☑ Feather WICED
- ☑ Flora
- ☑ Trinket / Pro Trinket / Gemma (USBtinyI
- ☐ Arduino Gemma
- ☐ Feather HUZZAH ESP8266 (SiLabs CP21
- ☐ Metro 328 / Metro Mini 328 (FTDI VCP)

Click **Install** to do the installin'



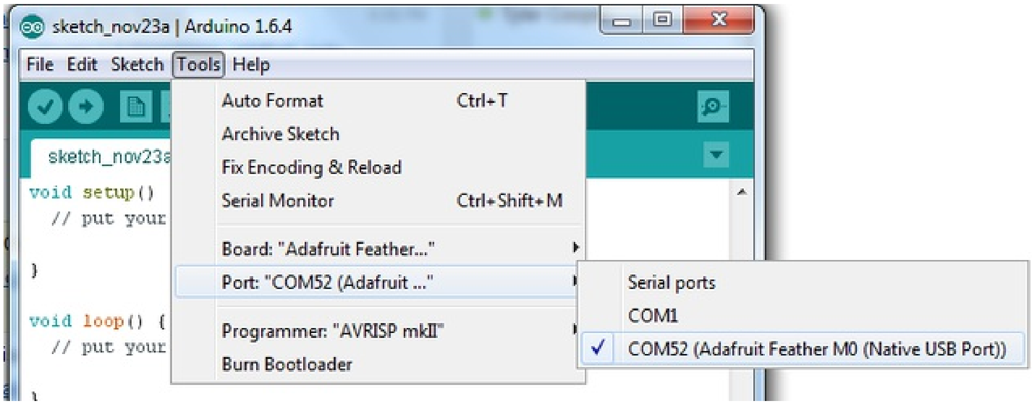
Adafruit Board Drivers: Completed

Completed

Blink

Now you can upload your first blink sketch!

Plug in the Feather M0 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 M0!



sketch_nov23a | Arduino 1.6.4

```
void setup()
  // put your
}

void loop() {
  // put your
}
```

File Edit Sketch Tools Help
sketch_nov23a
Auto Format Ctrl+T
Archive Sketch
Fix Encoding & Reload
Serial Monitor Ctrl+Shift+M
Board: "Adafruit Feather..."
Port: "COM52 (Adafruit ..."
Programmer: "AVRISP mkII"
Burn Bootloader
Serial ports
COM1
✓ COM52 (Adafruit Feather M0 (Native USB Port))

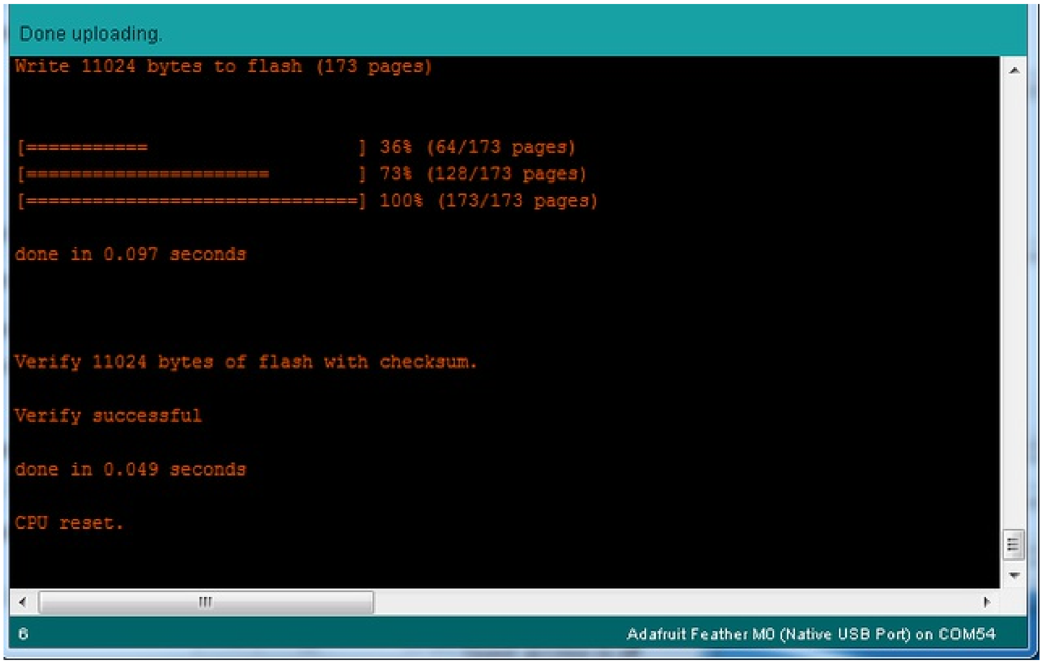
Now load up the Blink example

```
// 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.

Sucessful Upload

If you have a successful upload, you'll get a bunch of red text that tells you that the device was found and it was programmed, verified & reset



Done uploading.

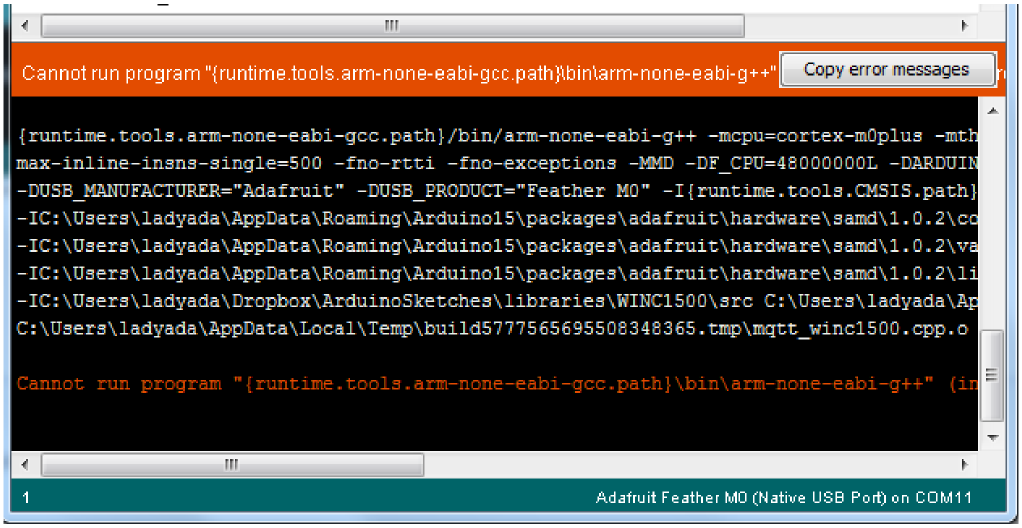
```
Write 11024 bytes to flash (173 pages)


[===========                   ] 36% (64/173 pages)
[=====================         ] 73% (128/173 pages)
[==============================] 100% (173/173 pages)

done in 0.097 seconds



Verify 11024 bytes of flash with checksum.

Verify successful

done in 0.049 seconds

CPU reset.
```

Compilation Issues

If you get an alert that looks like

Cannot run program "{runtime.tools.arm-none-eabi-gcc.path}\bin\arm-non-eabi-g++"

Make sure you have installed the **Arduino SAMD** boards package, you need *both* Arduino & Adafruit SAMD board packages



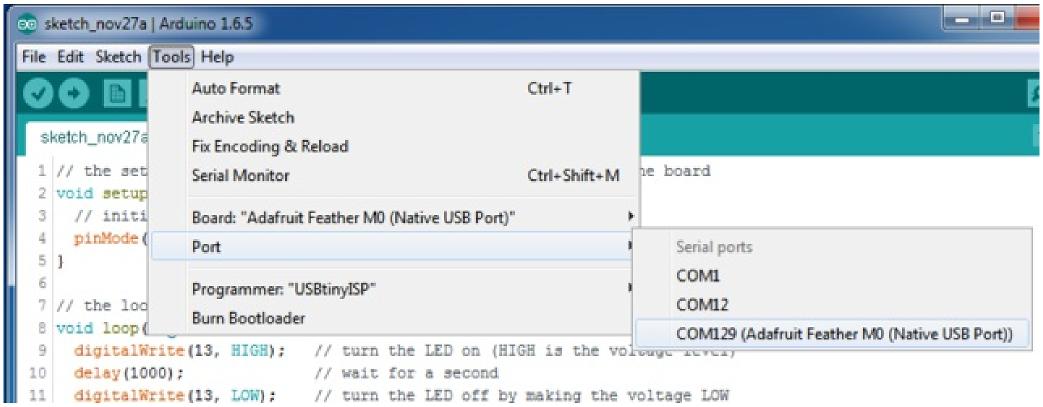
```
Cannot run program "{runtime.tools.arm-none-eabi-gcc.path}\bin\arm-none-eabi-g++"

{runtime.tools.arm-none-eabi-gcc.path}/bin/arm-none-eabi-g++ -mcpu=cortex-m0plus -mth
max-inline-insns-single=500 -fno-rtti -fno-exceptions -MMD -DF_CPU=48000000L -DARDUIN
-DUSB_MANUFACTURER="Adafruit" -DUSB_PRODUCT="Feather M0" -I{runtime.tools.CMSIS.path}
-IC:\Users\ladyada\AppData\Roaming\Arduino15\packages\adafruit\hardware\samd\1.0.2\co
-IC:\Users\ladyada\AppData\Roaming\Arduino15\packages\adafruit\hardware\samd\1.0.2\va
-IC:\Users\ladyada\AppData\Roaming\Arduino15\packages\adafruit\hardware\samd\1.0.2\li
-IC:\Users\ladyada\Dropbox\ArduinoSketches\libraries\WINC1500\src C:\Users\ladyada\Ap
C:\Users\ladyada\AppData\Local\Temp\build5777565695508348365.tmp\mqtt_winc1500.cpp.o

Cannot run program "{runtime.tools.arm-none-eabi-gcc.path}\bin\arm-none-eabi-g++" (in
```

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 **twice** (like a double-click)to get back into the bootloader.

The red LED will pulse, so you know that its in bootloader mode.

Once it is in bootloader mode, you can select the newly created COM/Serial port and re-try uploading.



sketch_nov27a | Arduino 1.6.5
File Edit Sketch Tools Help
sketch_nov27a
Auto Format Ctrl+T
Archive Sketch
Fix Encoding & Reload
Serial Monitor Ctrl+Shift+M
Board: "Adafruit Feather M0 (Native USB Port)"
Port
Programmer: "USBtinyISP"
Burn Bootloader
Serial ports
COM1
COM12
COM129 (Adafruit Feather M0 (Native USB Port))

```
// the set                                                   he board
void setup
  // initi
  pinMode(
}

// the loo
void loop(
  digitalWrite(13, HIGH);   // turn the LED on (HIGH is the vo
  delay(1000);              // wait for a second
  digitalWrite(13, LOW);    // turn the LED off by making the voltage LOW
```

You may need to go back and reselect the 'normal' USB serial port next time you want to use the normal 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 Feather 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!

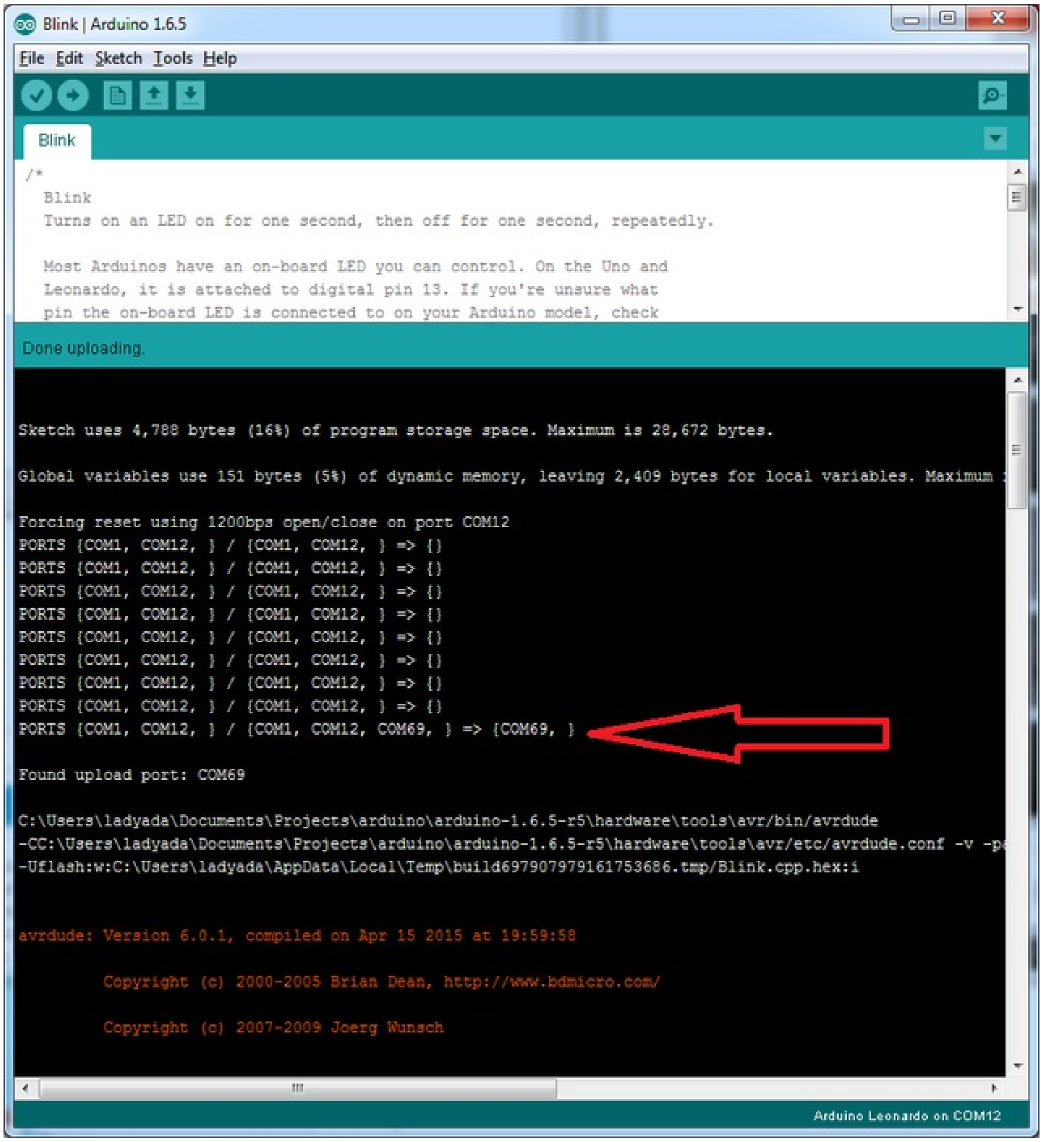
Use only a cable that you **know** is for data syncing

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



```
/*
  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
```

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 selec**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 thewrong 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 selected Feather M0.

Adapting Sketches to M0

The ATSAMD21 is a very nice little chip but its fairly new as Arduino-compatible cores go**Most** sketches & libraries will work but here's a few things we noticed!

Analog References

If you'd like to use the **ARef** pin for a non-3.3V analog reference, the code to use is`analogReference(AR_EXTERNAL)` (it's AR_EXTERNAL not EXTERNAL)

Pin Outputs & Pullups

The old-style way of turning on a pin as an input with a pullup is to use

```
pinMode(pin, INPUT)
digitalWrite(pin, HIGH)
```

This is because the pullup-selection register is the same as the output-selection register.

For the M0, you can't do this anymore! Instead, use

```
pinMode(pin, INPUT_PULLUP)
```

which has the benefit of being backwards compatible with AVR.

Serial vs SerialUSB

99.9% of your existing Arduino sketches use**Serial.print** to debug and give output. For the Official Arduino SAMD/M0 core, this goes to the Serial5 port, which isn't exposed on the Feather. The USB port for the Official Arduino M0 core, is called **SerialUSB** instead.

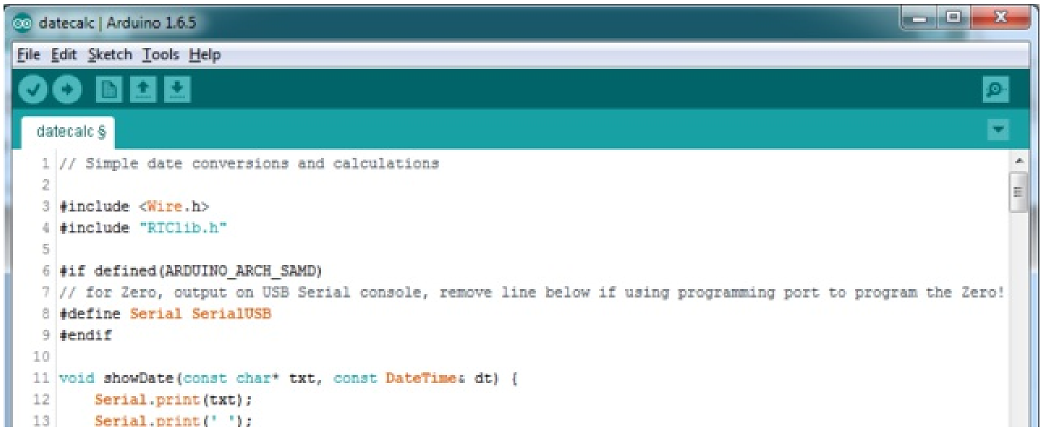
In the Adafruit M0 Core, we fixed it so that Serial goes to USB when you use a Feather M0 so it will automatically work just fine.

However, on the off chance you are using the official Arduino SAMD core & you want your Serial prints and reads to use the USB port, use **SerialUSB** instead of Serial in your sketch

If you have existing sketches and code and you want them to work with the M0 without a huge find-replace, put

```
#if defined(ARDUINO_SAMD_ZERO) && defined(SERIAL_PORT_USBVIRTUAL)
  // Required for Serial on Zero based boards
  #define Serial SERIAL_PORT_USBVIRTUAL
#endif
```

right above the first function definition in your code. For example:



```
// Simple date conversions and calculations

#include <Wire.h>
#include "RTClib.h"

#if defined(ARDUINO_ARCH_SAMD)
// for Zero, output on USB Serial console, remove line below if using programming port to program the Zero!
#define Serial SerialUSB
#endif

void showDate(const char* txt, const DateTime& dt) {
    Serial.print(txt);
    Serial.print(' ');
```

AnalogWrite / PWM

We've noticed that some PWM outputs are not working with the current SAMD core, its something that is being worked on!

Missing header files

there might be code that uses libraries that are not supported by the M0 core. For example if you have a line with

```
#include <util/delay.h>
```

you'll get an error that says

```
fatal error: util/delay.h: No such file or directory
  #include <util/delay.h>
                        ^
compilation terminated.
Error compiling.
```

In which case you can simply locate where the line is (the error will give you the file name and line number) and 'wrap it' with #ifdef's so it looks like:

```
#if !defined(ARDUINO_ARCH_SAM) && !defined(ARDUINO_ARCH_SAMD) && !defined(ESP8266) && !defined(ARDUINO_ARCH_STM32F2)
 #include <util/delay.h>
#endif
```

The above will also make sure that header file isn't included for other architectures

If the #include is in the arduino sketch itself, you can try just removing the line.

Bootloader Launching

For most other AVRs, clicking **reset** while plugged into USB will launch the bootloader manually, the bootloader will time out after a few seconds. For the M0, you'll need to *double click* the button. You will see a pulsing red LED to let you know you're in bootloader mode. Once in that mode, it wont time out! Click reset again if you want to go back to launching code

Aligned Memory Access

This is a little less likely to happen to you but it happened to me! If you're used to 8-bit platforms, you can do this nice thing where you can typecast variables around. e.g.

```
uint8_t mybuffer[4];
float f = (float)mybuffer;
```

You can't be guaranteed that this will work on a 32-bit platform because**mybuffer** might not be aligned to a 2 or 4-byte boundary. The ARM Cortex-M0 can only directly access data on 16-bit boundaries (every 2 or 4 bytes). Trying to access an odd-boundary byte (on a 1 or 3 byte location) will cause a Hard Fault and stop the MCU. Thankfully, there's an easy work around ... just use memcpy!

```
uint8_t mybuffer[4];
float f;
memcpy(f, mybuffer, 4)
```

Floating Point Conversion

Like the AVR Arduinos, the M0 library does not have full support for converting floating point numbers to ASCII strings. Functions like sprintf will not convert floating point. Fortunately, the standard AVR-LIBC library includes the dtostrf function which can handle the conversion for you.

Unfortunately, the M0 run-time library does not have dtostrf. You may see some references to using**#include <avr/dtostrf.h>** to get dtostrf in your code. And while it will compile, it does**not** work.

Instead, check out this thread to find a working dtostrf function you can include in your code:

http://forum.arduino.cc/index.php?topic=368720.0 (http://adafru.it/lFS)

How Much RAM Available?

The ATSAMD21G18 has 32K of RAM, but you still might need to track it for some reason. You can do so with this handy function:

```
extern "C" char *sbrk(int i);

int FreeRam () {
  char stack_dummy = 0;
  return &stack_dummy - sbrk(0);
}
```

Thx to http://forum.arduino.cc/index.php?topic=365830.msg2542879#msg2542879 (http://adafru.it/m6D) for the tip!

Storing data in FLASH

If you're used to AVR, you've probably used**PROGMEM** to let the compiler know you'd like to put a variable or string in flash memory to save on RAM. On the ARM, its a little easier, simply add **const** before the variable name:

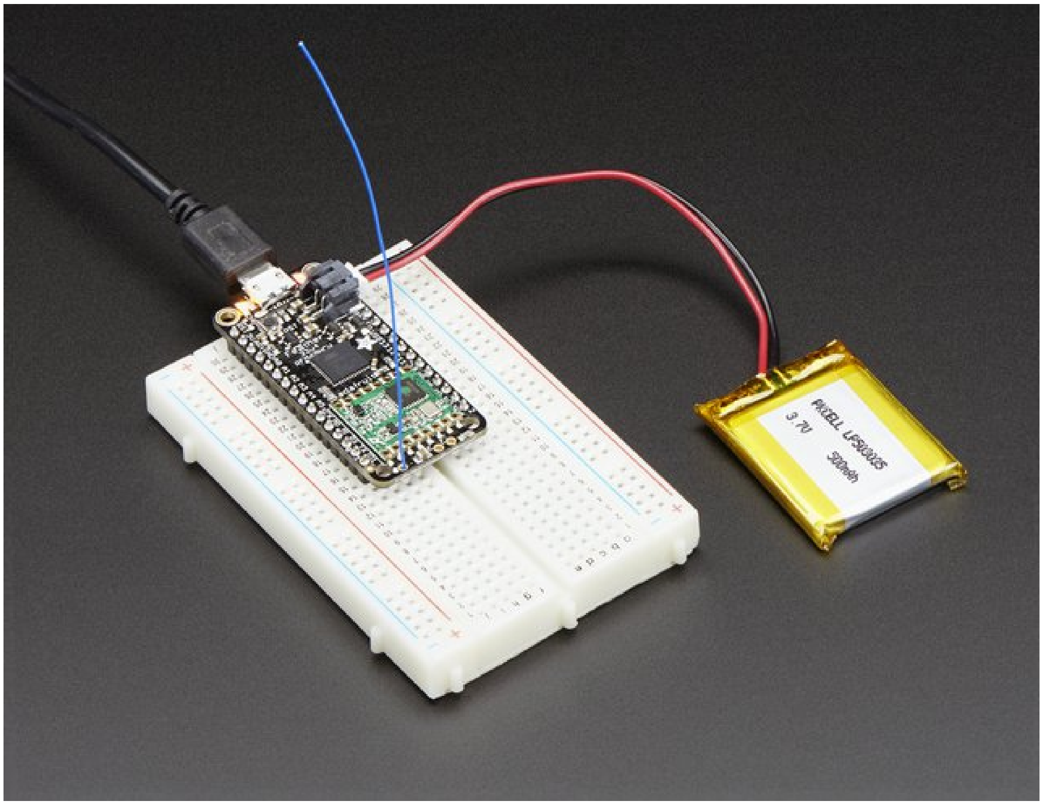
const char str[] = "My very long string";

That string is now in FLASH. You can manipulate the string just like RAM data, the compiler will automatically read from FLASH so you dont need special progmem-knowledgeable functions.

You can verify where data is stored by printing out the address:
Serial.print("Address of str $"); Serial.println((int)&str, HEX);

If the address is $2000000 or larger, its in SRAM. If the address is between $0000 and $3FFFF Then it is in FLASH

Using the RFM69 Radio



Before beginning make sure you have your Feather working smoothly, it will make this part a lot easier. Once you have the basic Feather 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 radiosto 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!

LowPowerLab RFM69 Library example

To begin talking to the radio, you will need todownload 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

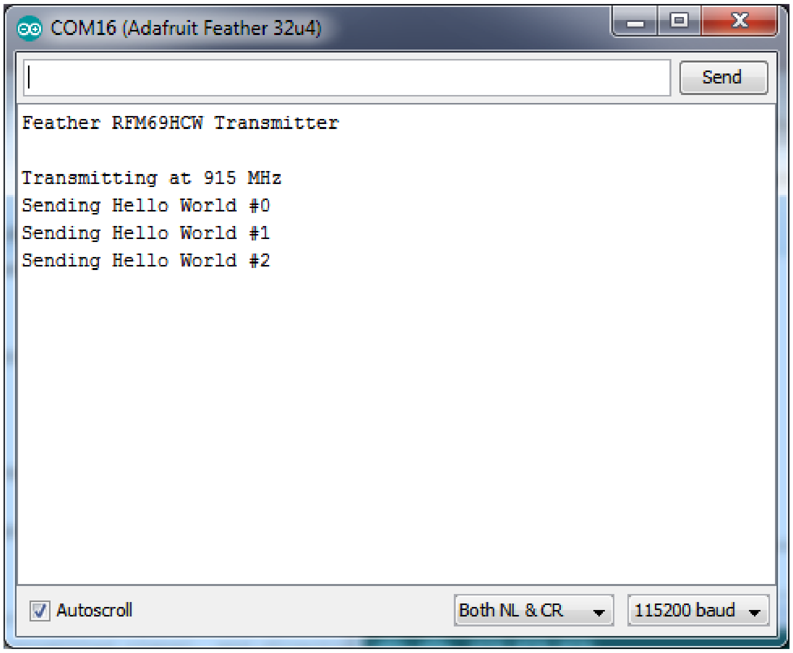
Load this code into your Transmitter Arduino/Feather!

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
Uncomment/comment the sections defining the pins for Feather 32u4, Feather M0, etc depending on which chipset and wiring you are using! The pins used will vary depending on your setup!
On the ESP8266, change the LED pin from 13 to 0

```
/* 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        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

/* for Feather 32u4 Radio
#define RFM69_CS      8
#define RFM69_IRQ     7
#define RFM69_IRQN    4  // Pin 7 is IRQ 4!
#define RFM69_RST     4
*/

/* for Feather M0 Radio
#define RFM69_CS      8
#define RFM69_IRQ     3
#define RFM69_IRQN    3  // Pin 3 is IRQ 3!
#define RFM69_RST     4
*/

/* ESP8266 feather w/wing
#define RFM69_CS      2
#define RFM69_IRQ     15
#define RFM69_IRQN    digitalPinToInterrupt(RFM69_IRQ )
#define RFM69_RST     16
*/

/* Feather 32u4 w/wing
#define RFM69_RST     11   // "A"
#define RFM69_CS      10   // "B"
#define RFM69_IRQ     2    // "SDA" (only SDA/SCL/RX/TX have IRQ!)
```

```
#define RFM69_IRQN    digitalPinToInterrupt(RFM69_IRQ )
*/

/* Feather m0 w/wing 
#define RFM69_RST     11   // "A"
#define RFM69_CS      10   // "B"
#define RFM69_IRQ     6    // "D"
#define RFM69_IRQN    digitalPinToInterrupt(RFM69_IRQ )
*/

/* Teensy 3.x w/wing 
#define RFM69_RST     9   // "A"
#define RFM69_CS      10   // "B"
#define RFM69_IRQ     4    // "C"
#define RFM69_IRQN    digitalPinToInterrupt(RFM69_IRQ )
*/

/* WICED Feather w/wing 
#define RFM69_RST     PA4     // "A"
#define RFM69_CS      PB4     // "B"
#define RFM69_IRQ     PA15    // "C"
#define RFM69_IRQN    RFM69_IRQ
*/

#define LED           13  // onboard blinky
//#define LED           0 //use 0 on ESP8266

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. Delete this line on ESP8266
  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



```
Feather RFM69HCW Transmitter

Transmitting at 915 MHz
Sending Hello World #0
Sending Hello World #1
Sending Hello World #2
```

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.

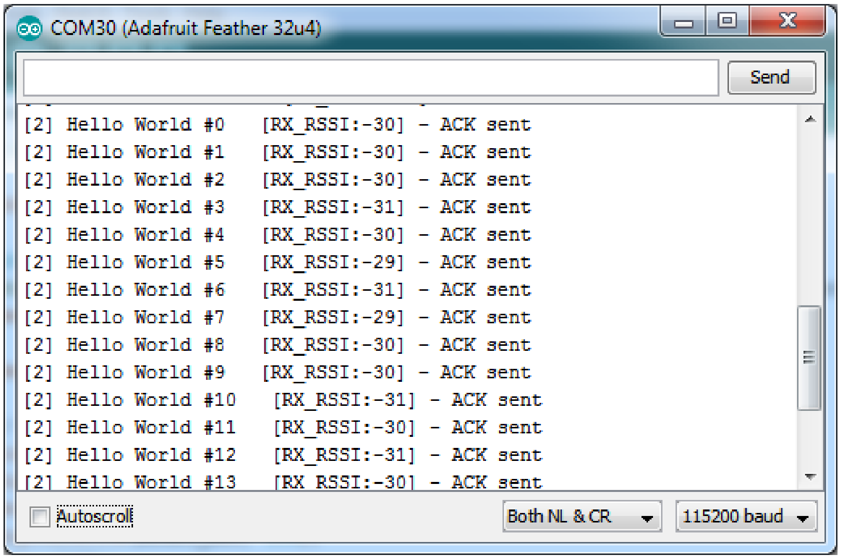
Load this code into your **Receiver** Arduino/Feather!

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
Uncomment/comment the sections defining the pins for Feather 32u4, Feather M0, etc depending on which chipset and wiring you are using! The pins used will vary depending on your setup!
On the ESP8266, change the LED pin from 13 to 0

```
/* 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

/* for Feather 32u4 */
#define RFM69_CS      8
#define RFM69_IRQ     7
#define RFM69_IRQN    4  // Pin 7 is IRQ 4!
#define RFM69_RST     4

/* for Feather M0
#define RFM69_CS      8
#define RFM69_IRQ     3
#define RFM69_IRQN    3  // Pin 3 is IRQ 3!
#define RFM69_RST     4
*/

/* ESP8266 feather w/wing
```

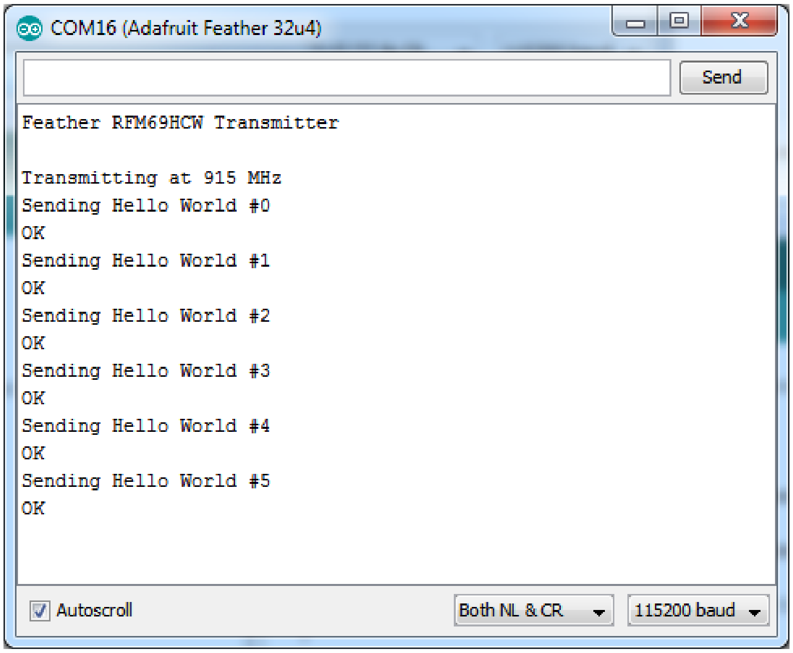
```
#define RFM69_CS      2
#define RFM69_IRQ     15
#define RFM69_IRQN    digitalPinToInterrupt(RFM69_IRQ )
#define RFM69_RST     16
*/

/* Feather 32u4 w/wing
#define RFM69_CS      10   // "B"
#define RFM69_RST     11   // "A"
#define RFM69_IRQ     2    // "SDA" (only SDA/SCL/RX/TX have IRQ!)
#define RFM69_IRQN    digitalPinToInterrupt(RFM69_IRQ )
*/

/* Feather m0 w/wing
#define RFM69_CS      10   // "B"
#define RFM69_RST     11   // "A"
#define RFM69_IRQ     6    // "D"
#define RFM69_IRQN    digitalPinToInterrupt(RFM69_IRQ )
*/

/* Teensy 3.x w/wing
#define RFM69_RST     9   // "A"
#define RFM69_CS      10   // "B"
#define RFM69_IRQ     4    // "C"
#define RFM69_IRQN    digitalPinToInterrupt(RFM69_IRQ )
*/

/* WICED Feather w/wing
#define RFM69_RST     PA4     // "A"
#define RFM69_CS      PB4     // "B"
#define RFM69_IRQ     PA15    // "C"
#define RFM69_IRQN    RFM69_IRQ
*/

#define LED           13  // onboard blinky
//#define LED           0 //use 0 on ESP8266

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. Delete this line on ESP8266
  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



```
[2] Hello World #0   [RX_RSSI:-30] - ACK sent
[2] Hello World #1   [RX_RSSI:-30] - ACK sent
[2] Hello World #2   [RX_RSSI:-30] - ACK sent
[2] Hello World #3   [RX_RSSI:-31] - ACK sent
[2] Hello World #4   [RX_RSSI:-30] - ACK sent
[2] Hello World #5   [RX_RSSI:-29] - ACK sent
[2] Hello World #6   [RX_RSSI:-31] - ACK sent
[2] Hello World #7   [RX_RSSI:-29] - ACK sent
[2] Hello World #8   [RX_RSSI:-30] - ACK sent
[2] Hello World #9   [RX_RSSI:-30] - ACK sent
[2] Hello World #10   [RX_RSSI:-31] - ACK sent
[2] Hello World #11   [RX_RSSI:-30] - ACK sent
[2] Hello World #12   [RX_RSSI:-31] - ACK sent
[2] Hello World #13   [RX_RSSI:-30] - ACK sent
```

And, on the transmitter side, it is now printing**OK** after each transmisssion because it got an acknowledgement from the receiver



```
Feather RFM69HCW Transmitter

Transmitting at 915 MHz
Sending Hello World #0
OK
Sending Hello World #1
OK
Sending Hello World #2
OK
Sending Hello World #3
OK
Sending Hello World #4
OK
Sending Hello World #5
OK
```

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

```
/* for Feather 32u4 */
#define RFM69_CS      8
#define RFM69_IRQ     7
#define RFM69_IRQN    4  // Pin 7 is IRQ 4!
#define RFM69_RST     4
```

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

```
/* for Feather M0 */
#define RFM69_CS      8
#define RFM69_IRQ     3
#define RFM69_IRQN    3  // Pin 3 is IRQ 3!
#define RFM69_RST     4
```

If you're using the FeatherWings, you'll have to set 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 *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.  Delete this line on ESP8266
  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/Transmitter 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)

On the ESP8266, change the LED pin from 13 to 0

```
/* 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

/* for Feather 32u4 */
#define RFM69_CS      8
#define RFM69_IRQ     7
#define RFM69_IRQN    4  // Pin 7 is IRQ 4!
#define RFM69_RST     4

/* for Feather M0
#define RFM69_CS      8
#define RFM69_IRQ     3
#define RFM69_IRQN    3  // Pin 3 is IRQ 3!
#define RFM69_RST     4
*/
/* ESP8266 feather w/wing
#define RFM69_CS      2
#define RFM69_IRQ     15
#define RFM69_IRQN    digitalPinToInterrupt(RFM69_IRQ )
#define RFM69_RST     16
*/

/* Feather 32u4 w/wing
#define RFM69_CS      10   // "B"
#define RFM69_RST     11   // "A"
#define RFM69_IRQ     2    // "SDA" (only SDA/SCL/RX/TX have IRQ!)
#define RFM69_IRQN    digitalPinToInterrupt(RFM69_IRQ )
*/

/* Feather m0 w/wing
#define RFM69_CS      10   // "B"
#define RFM69_RST     11   // "A"
#define RFM69_IRQ     6    // "D"
#define RFM69_IRQN    digitalPinToInterrupt(RFM69_IRQ )
*/

/* Teensy 3.x w/wing
```

```
#define RFM69_RST     9   // "A"
#define RFM69_CS      10   // "B"
#define RFM69_IRQ     4    // "C"
#define RFM69_IRQN    digitalPinToInterrupt(RFM69_IRQ )
*/

/* WICED Feather w/wing
#define RFM69_RST     PA4     // "A"
#define RFM69_CS      PB4     // "B"
#define RFM69_IRQ     PA15    // "C"
#define RFM69_IRQN    RFM69_IRQ
*/

#define LED           13  // onboard blinky
//#define LED           0 //use 0 on ESP8266

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!

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

For the LoRa version:

- SX127x Datasheet (http://adafru.it/oBm)- The RFM9X LoRa radio chip itself
- RFM9X (http://adafru.it/mFX) - The radio module, which contains the SX1272 chipset

For the RFM69 version:

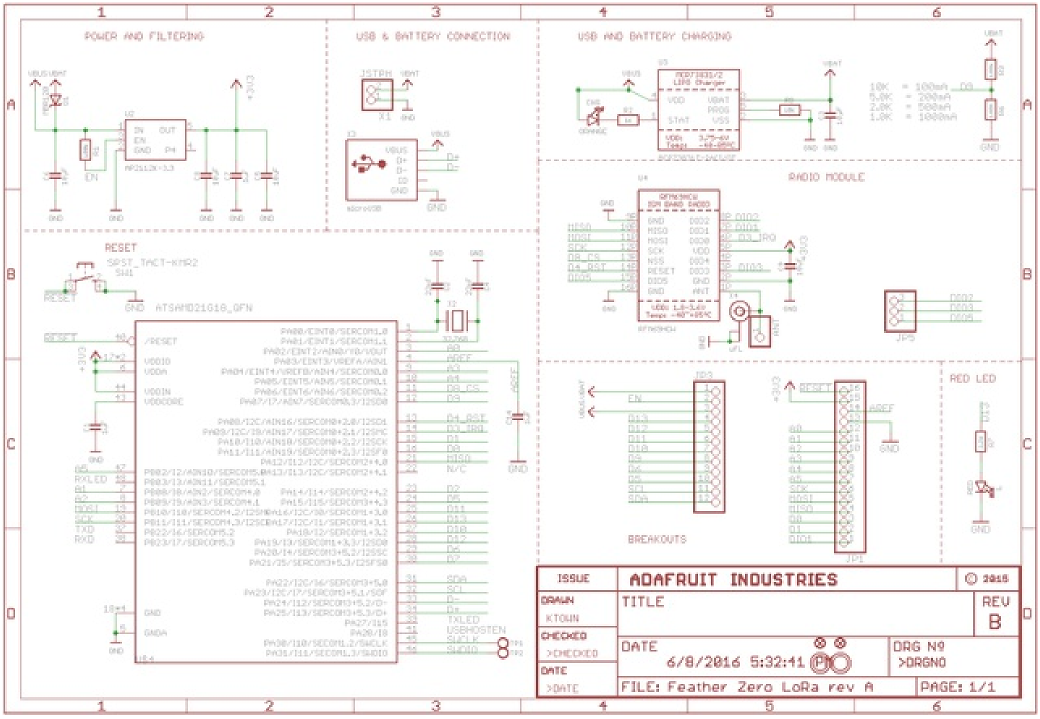
- SX1231 Transceiver Datasheet (http://adafru.it/mCv)
- RFM69HCW Datasheet (http://adafru.it/oC0)
- FCC Test Report (http://adafru.it/oC0)
- RoHS Test Report (http://adafru.it/oC1)
- RoHS Test Report (http://adafru.it/oC2)
- REACH Test Report (http://adafru.it/oC3)

For both:

- PCB files on GitHub (http://adafru.it/mCu)
- Fritzing Library (http://adafru.it/aP3)

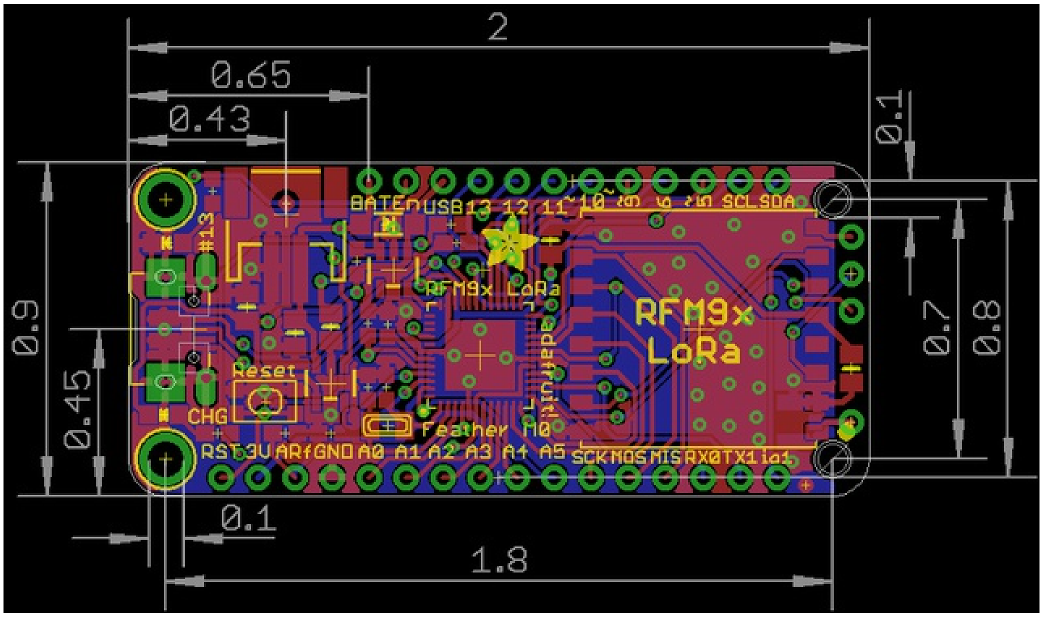
Schematic

Click to embiggen. Same schematic for both LoRa and RFM69 (the modules have the same pinout)



Fabrication Print

Dimensions in Inches. Same PCB design for both LoRa and RFM69 (the modules have the same pinout)


2
0.65
0.43
0.1
0.9
0.45
0.7
0.8
0.1
1.8
BAT EN USB 13 12 11 10 9 6 5 SCL SDA
#13
RFM9x LoRa
RFM9x
LoRa
Reset
CHG
Feather M0
adafruit!
RST 3V ARf GND A0 A1 A2 A3 A4 A5 SCK MOS MIS RX0 TX1 io1