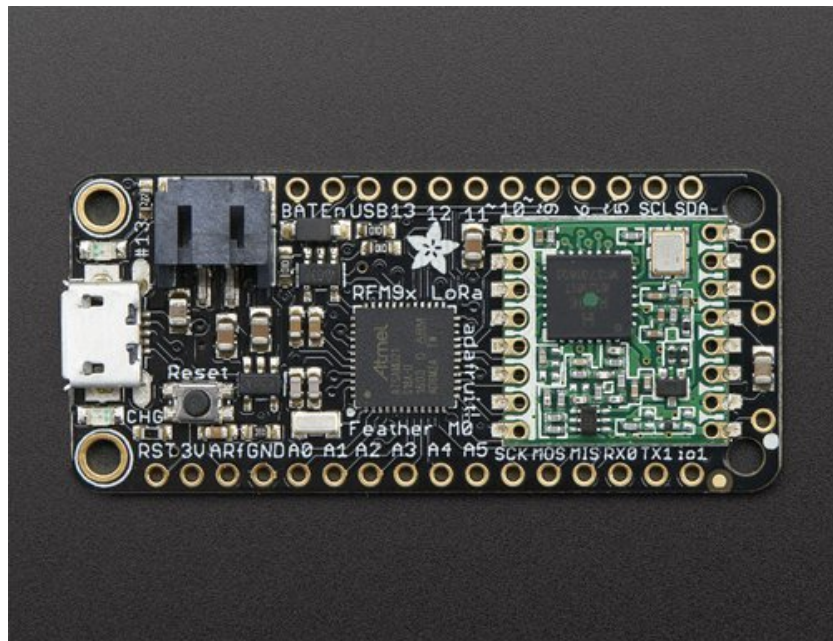


□

Adafruit Feather M0 Radio with LoRa Radio Module

Created by lady ada



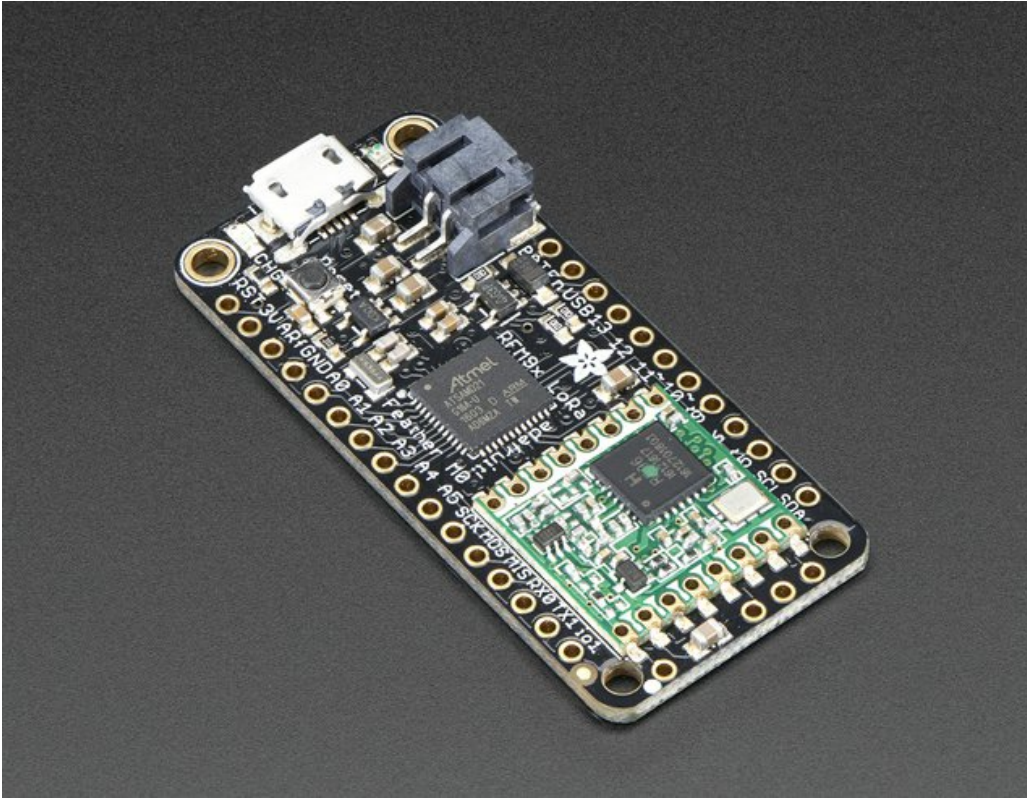
Last updated on 2016-11-07 07:51:33 PM UTC

Guide Contents

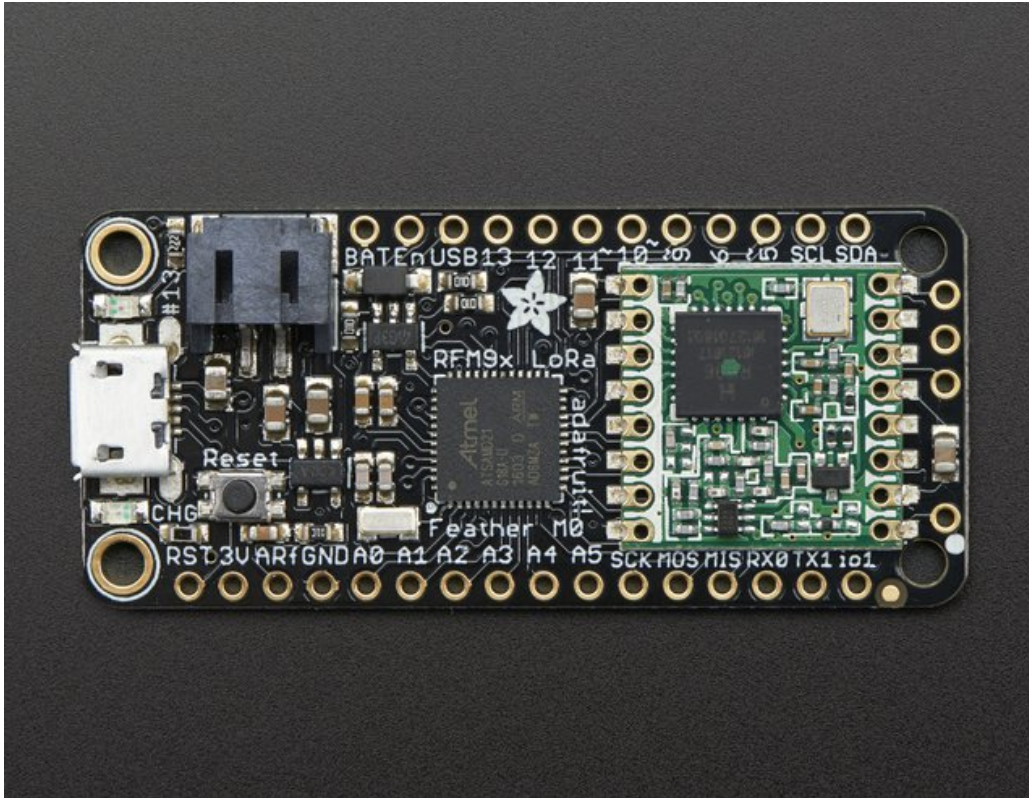
Guide Contents	2
Overview	5
Pinouts	9
Power Pins	9
Logic pins	10
RFM/SemTech Radio Module	10
Other Pins!	11
Assembly	13
Header Options!	13
Soldering in Plain Headers	15
Prepare the header strip:	15
Add the breakout board:	16
And Solder!	16
Soldering on Female Header	18
Tape In Place	18
Flip & Tack Solder	19
And Solder!	20
Antenna Options	22
Wire Antenna	22
uFL Antenna	23
Power Management	25
Battery + USB Power	25
Power supplies	26
Measuring Battery	26
ENable pin	27
Radio Power Draw	27
Arduino IDE Setup	30
https://adafruit.github.io/arduino-board-index/package_adafruit_index.json	31
Feather HELP!	33
Using with Arduino IDE	36
Install SAMD Support	36
Install Adafruit SAMD	37

Install Drivers (Windows Only)	38
Blink	39
Sucessful Upload	40
Compilation Issues	40
Manually bootloading	41
Ubuntu & Linux Issue Fix	41
Adapting Sketches to M0	42
Analog References	42
Pin Outputs & Pullups	42
Serial vs SerialUSB	42
AnalogWrite / PWM	43
Missing header files	43
Bootloader Launching	43
Aligned Memory Access	43
Floating Point Conversion	44
How Much RAM Available?	44
Storing data in FLASH	44
Using the RFM9X Radio	46
Arduino Library	46
RadioHead RFM9x Library example	46
Basic RX & TX example	47
Transmitter example code	47
Receiver example code	50
Feather Radio Pinout	53
Frequency	54
Setup	54
Initializing Radio	54
Transmission Code	55
Receiver Code	55
Radio Range F.A.Q.	57
Downloads	58
Datasheets & Files	58
Schematic	58

Overview

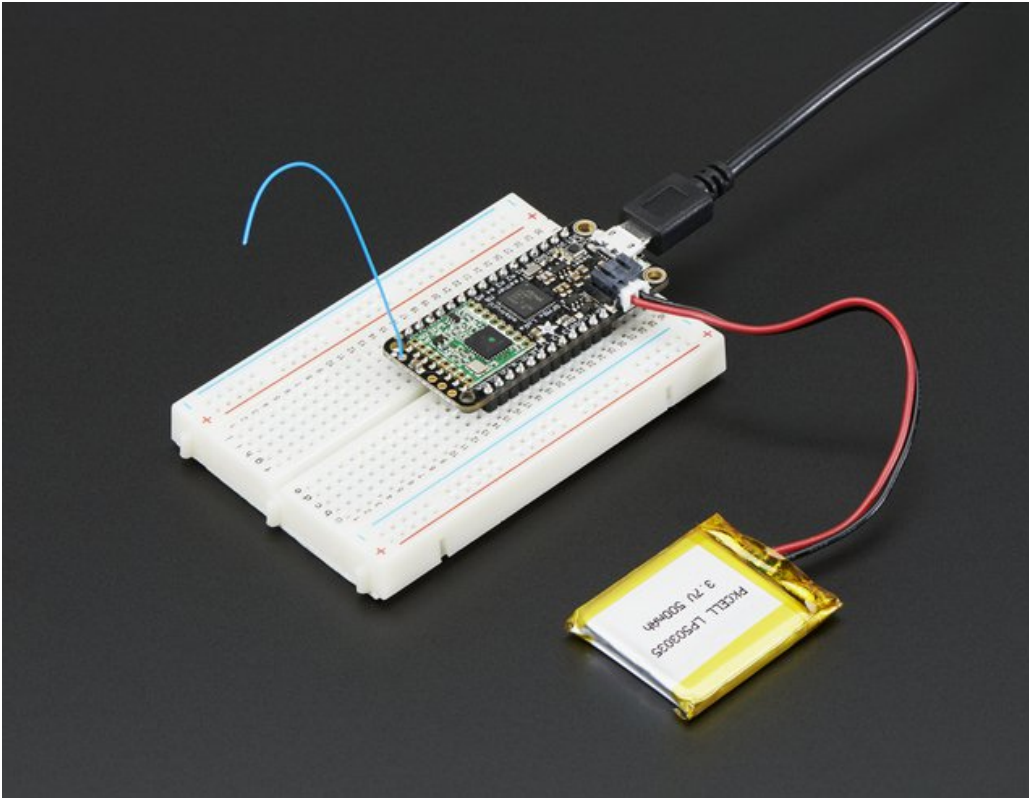


This is the **Adafruit Feather M0 RFM95 LoRa Radio (433 or 900 MHz)**- our take on an microcontroller with a "Long Range (LoRa) (<http://adafru.it/mFZ>)" packet radio transceiver with built in USB and battery charging. Its an Adafruit Feather M0 with a Long Range radio module cooked in! Great for making wireless networks that 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/I7B>).



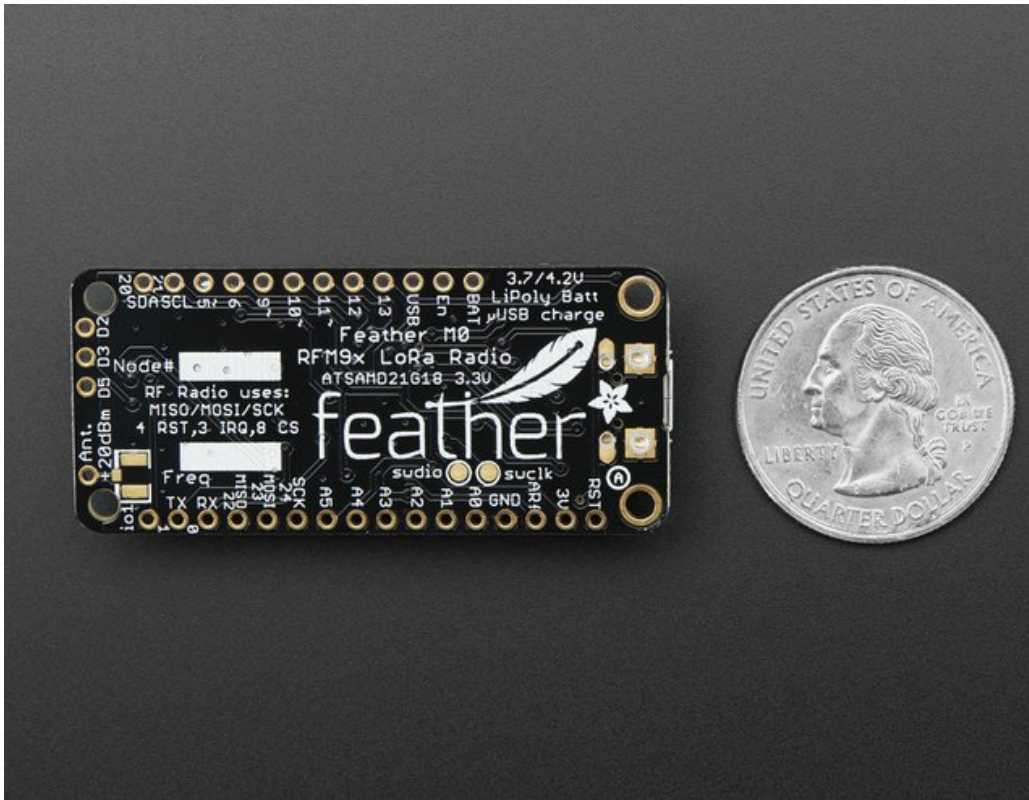
At the Feather M0's heart is an ATSAM21G18 ARM Cortex M0 processor, clocked at 48 MHz and at 3.3V logic, the same one used in the new [Arduino Zero](http://adafruit.it/2843) (<http://adafruit.it/2843>). This chip has a whopping 256K of FLASH (8x more than the Atmega328 or 32u4) and 32K of RAM (16x as 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
- ATSAM21G18 @ 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



This **Feather M0 LoRa Radio** uses the extra space left over to add an RFM9x LoRa 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 need more range than 2.4 GHz (BT, BLE, WiFi, ZigBee).

- SX1276 LoRa® based module with SPI interface
- Packet radio with ready-to-go Arduino libraries
- Uses the license-free ISM bands (ITU "Europe" @ 433MHz and ITU "Americas" @ 900MHz)
- +5 to +20 dBm up to 100 mW Power Output Capability (power output selectable in software)
- ~300uA during full sleep, ~120mA peak during +20dBm transmit, ~40mA during active radio listening.
- Simple wire antenna or spot for uFL connector

Our initial tests with default library settings: over 1.2mi/2Km line-of-sight with wire quarter-wave antennas. [With setting tweaking and directional antennas, 20Km is possible](http://adafruit.it/mGa) (<http://adafruit.it/mGa>).

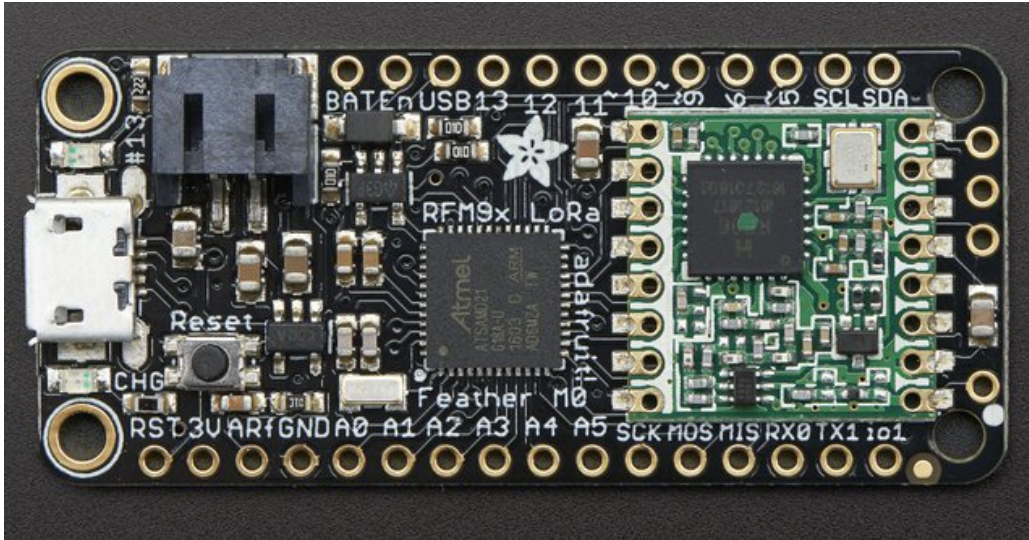
Comes fully assembled and tested, with a USB bootloader that lets you quickly use it with the Arduino IDE. We also toss in some headers so you can solder it in and plug into a solderless breadboard. You will need to cut and solder on a small piece of wire (any solid or stranded core is fine) in order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

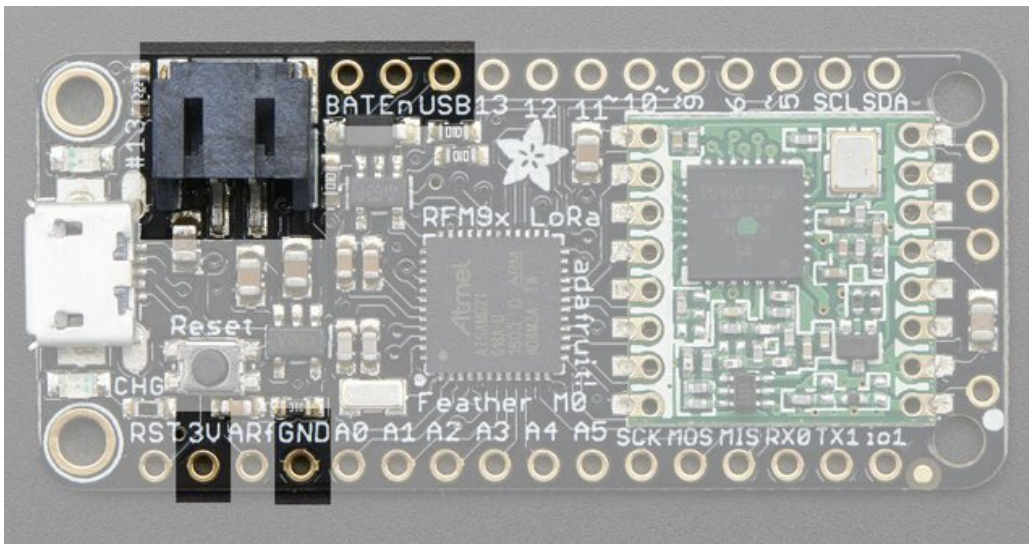
The Feather M0 Radio is chock-full of microcontroller goodness. There's also a lot of pins and ports. We'll take you a tour of them now!

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



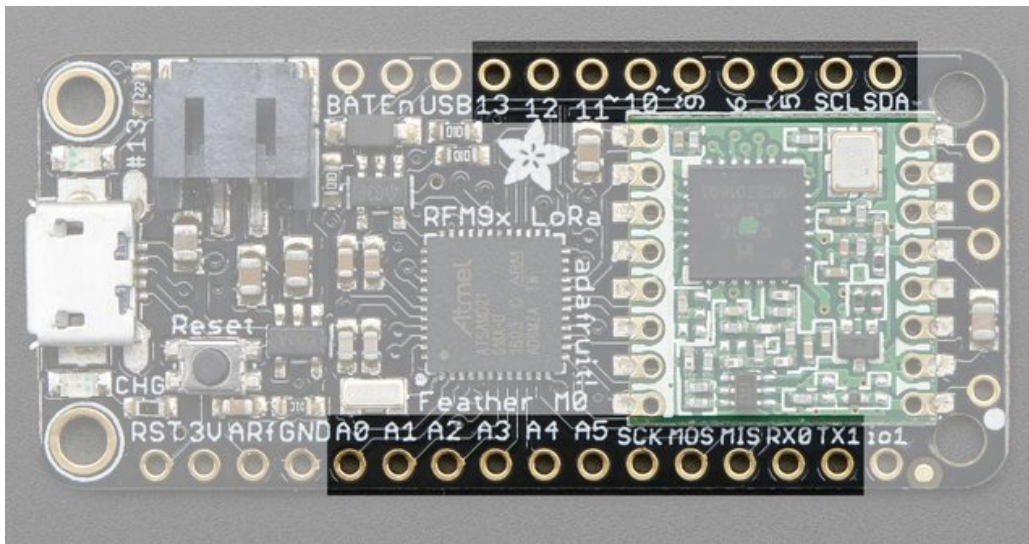
Power Pins



- **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

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

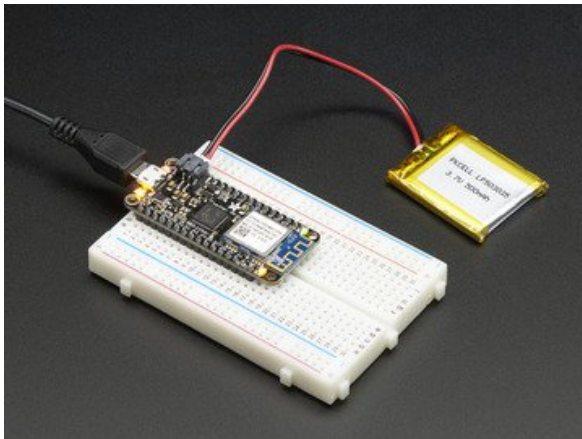
firmware. Can't go higher than 3.3V!

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

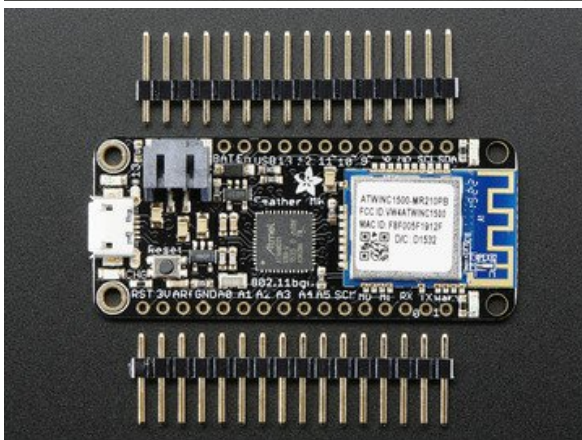
Header Options!

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

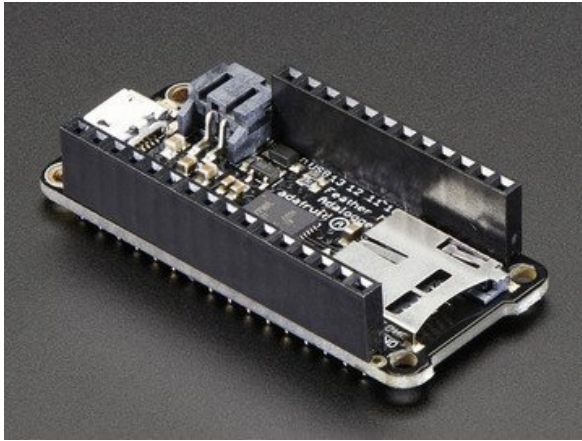


-

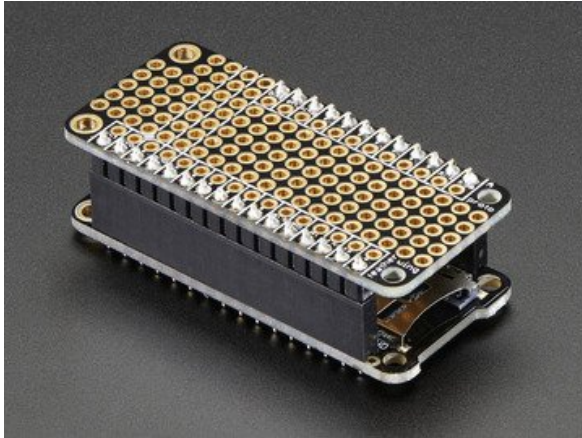
The first option is soldering in plain male headers, this lets you plug in the Feather into a solderless breadboard



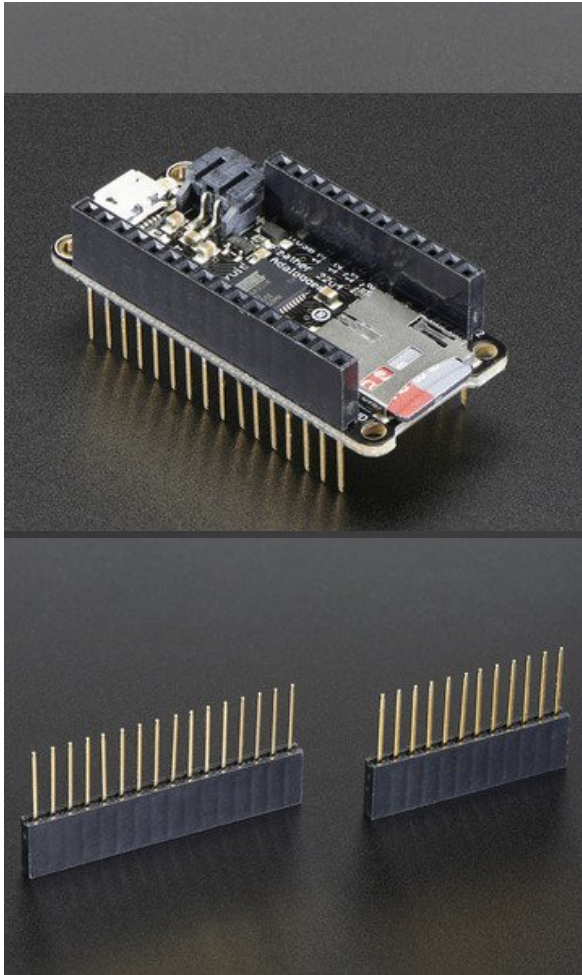
-



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

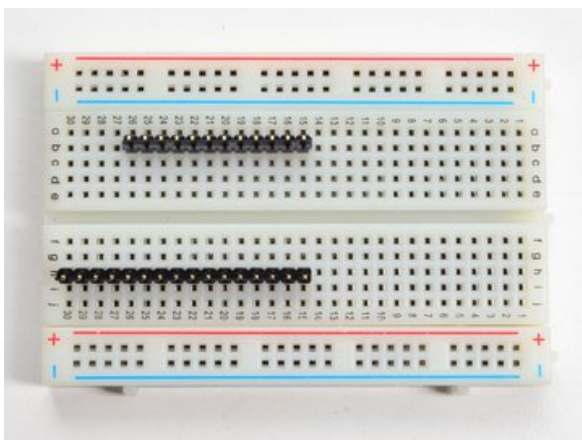


We also have 'slim' versions of the female headers, that are a little shorter and give a more compact shape



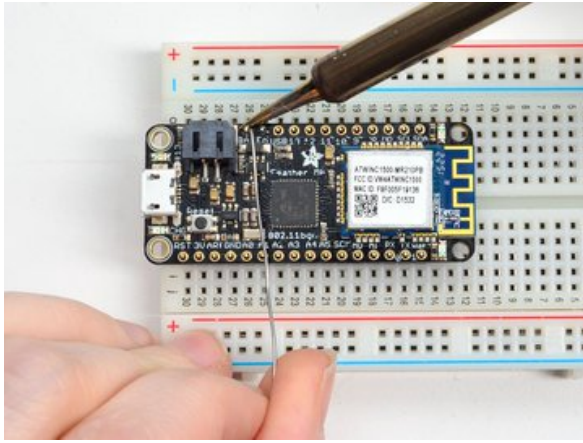
Finally, there's the "Stacking Header" option. This one is sort of the best-of-both-worlds. You get the ability to plug into a solderless breadboard *and* plug a featherwing on top. But its a little bulky

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**



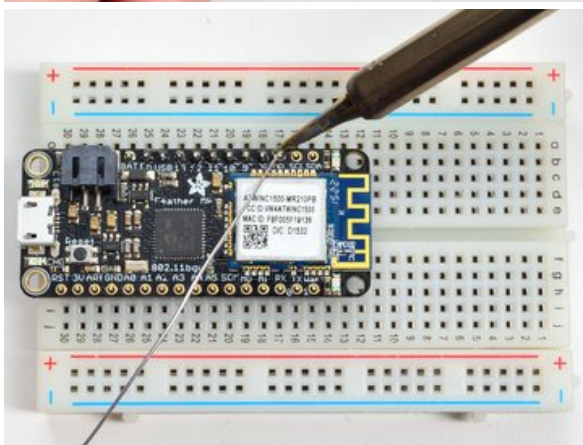
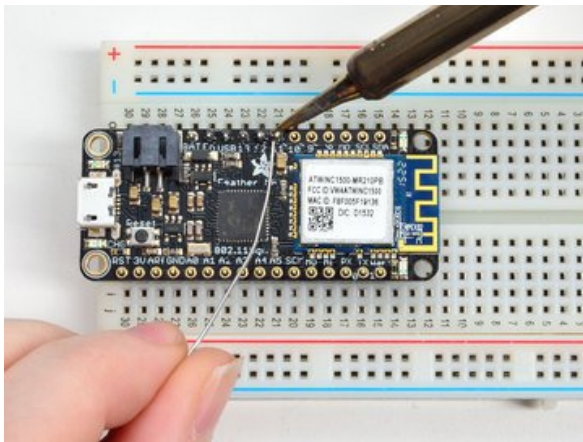
Add the breakout board:

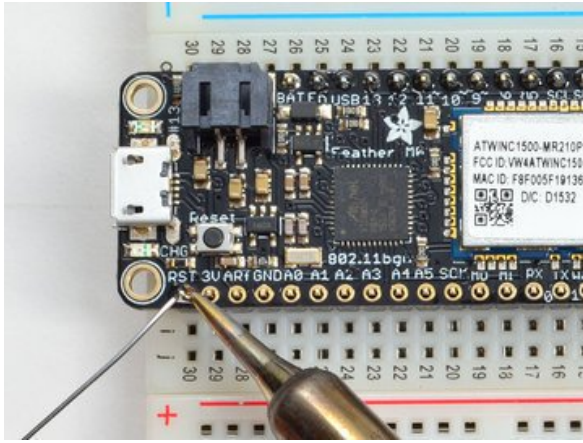
Place the breakout board over the pins so that the short pins poke through the breakout pads

And Solder!

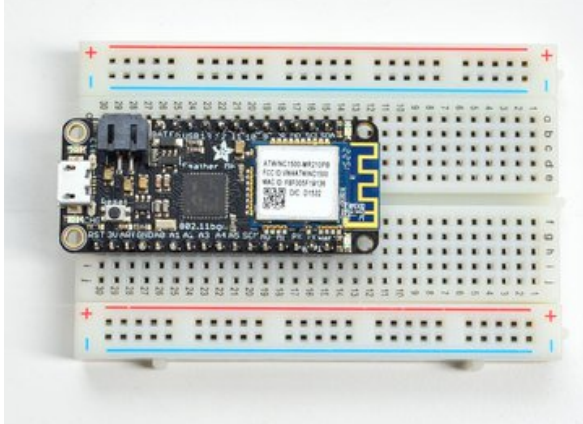
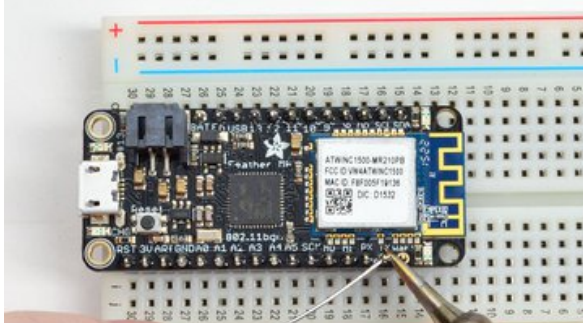
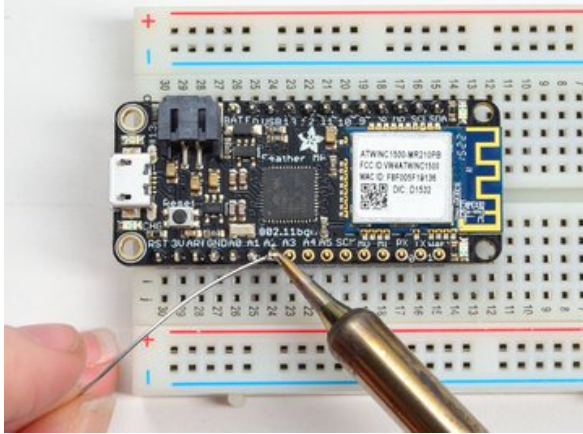
Be sure to solder all pins for reliable electrical contact.

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

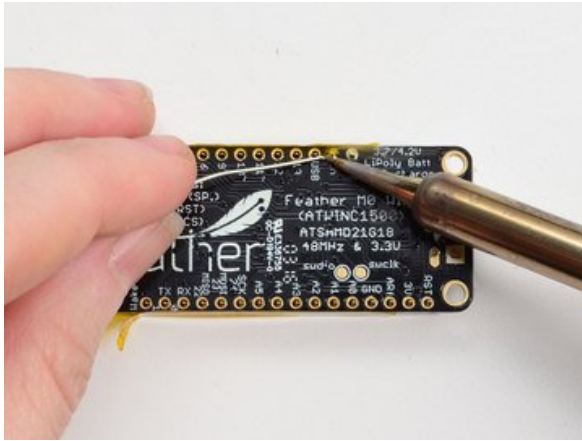


Tape In Place

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

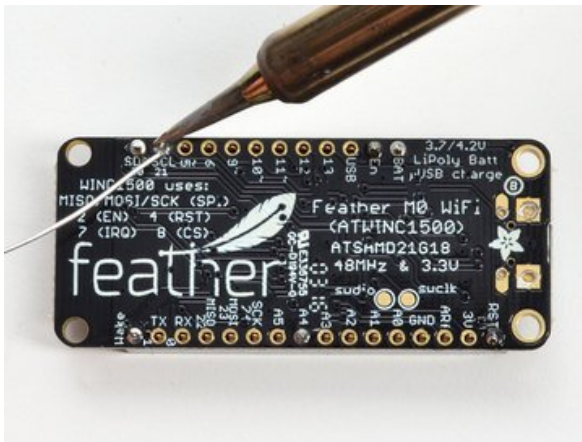
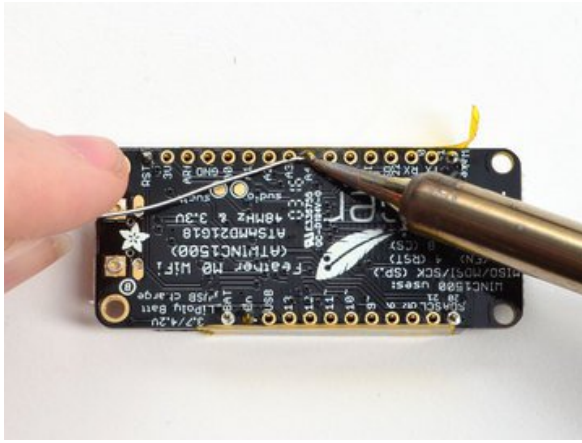
After flipping over, solder one or two points on each strip, to 'tack' the header in place



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://adafruit.it/aTk) (<http://adafruit.it/aTk>)).



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

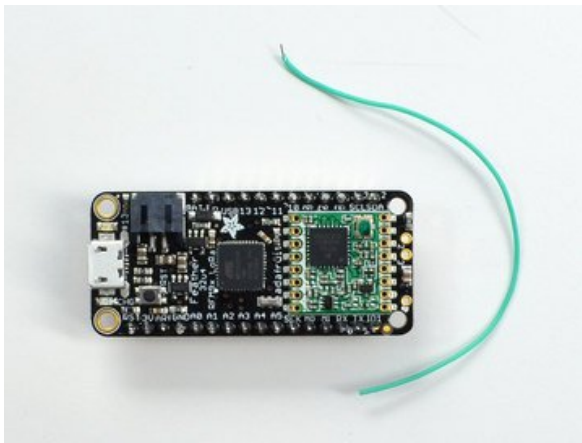


Antenna Options

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

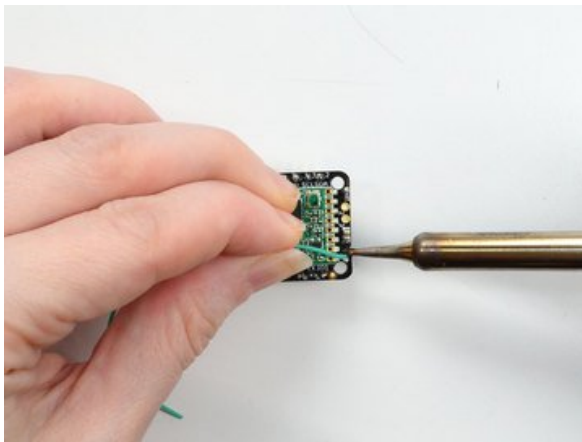
Wire Antenna

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

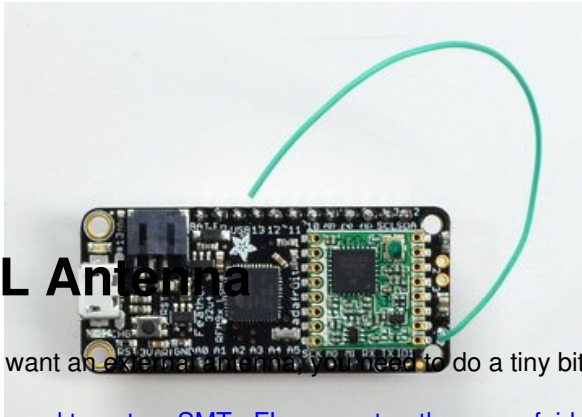
- **433 MHz** - 6.5 inches, or 16.5 cm
- **868 MHz** - 3.25 inches or 8.2 cm
- **915 MHz** - 3 inches or 7.8 cm



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



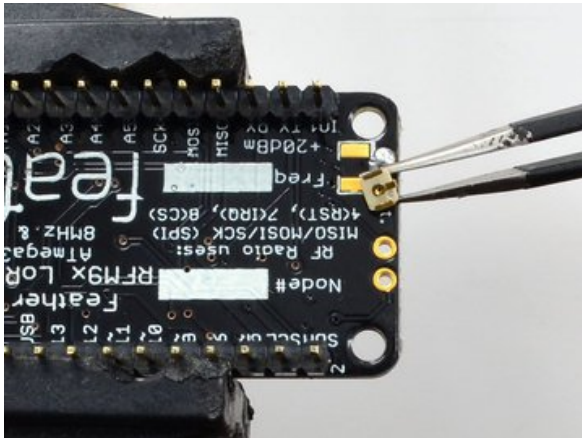
If you want an external antenna, you need to do a tiny bit more work but its not too difficult.

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

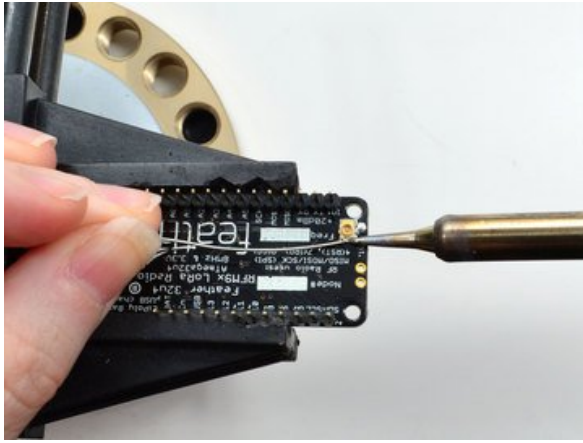
You'll also need a uFL to SMA adapter (<http://adafruit.it/851>) (or whatever adapter you need for the antenna you'll be using, SMA is the most common

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

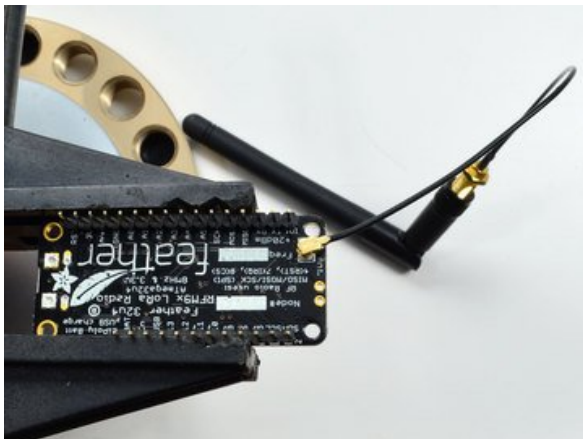
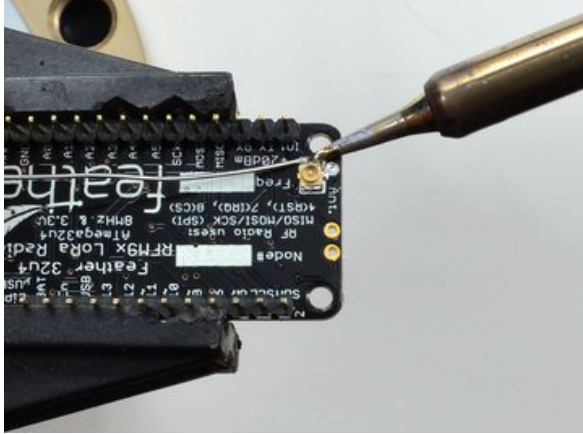
uFL connectors are rated for 30 connection cycles, but be careful when connecting/disconnecting to not rip the pads off the PCB. Once a uFL/SMA adapter is connected, use strain relief!



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!

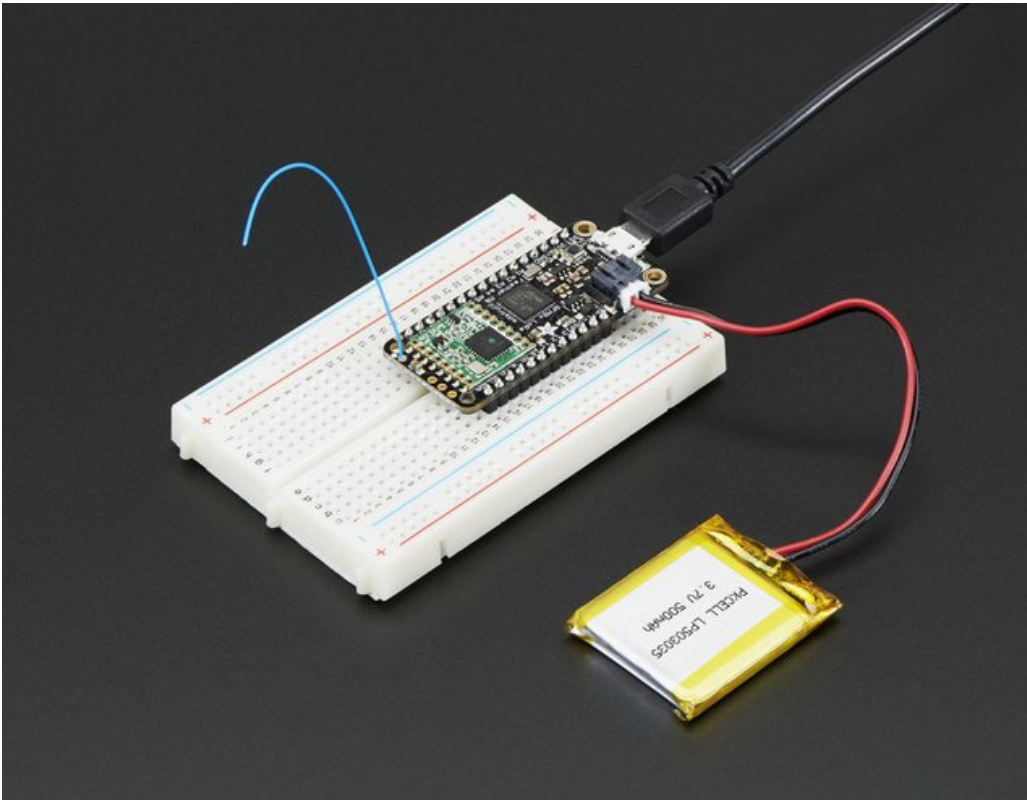


Solder all three pads to the bottom of the Feather



Once done attach your uFL adapter and antenna!

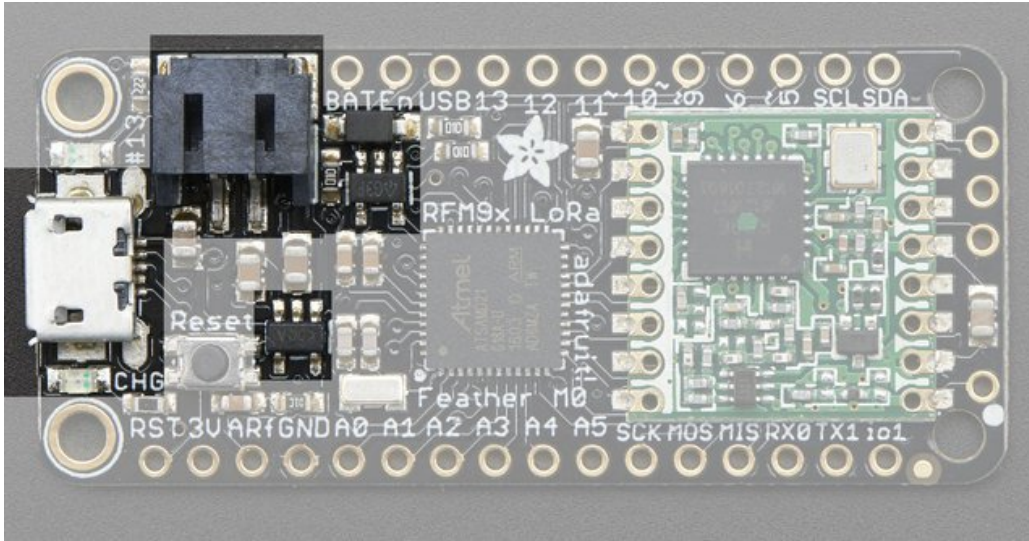
Power Management



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 (Lilon) battery to the JST jack. This will let the Feather run on a rechargeable battery. **When the USB power is powered, it will automatically switch over to USB for power, as well as start charging the battery (if attached) at 100mA.** This happens 'hotswap' style so you can always keep the Lipoly connected as a 'backup' power that will only get used when USB power is lost.

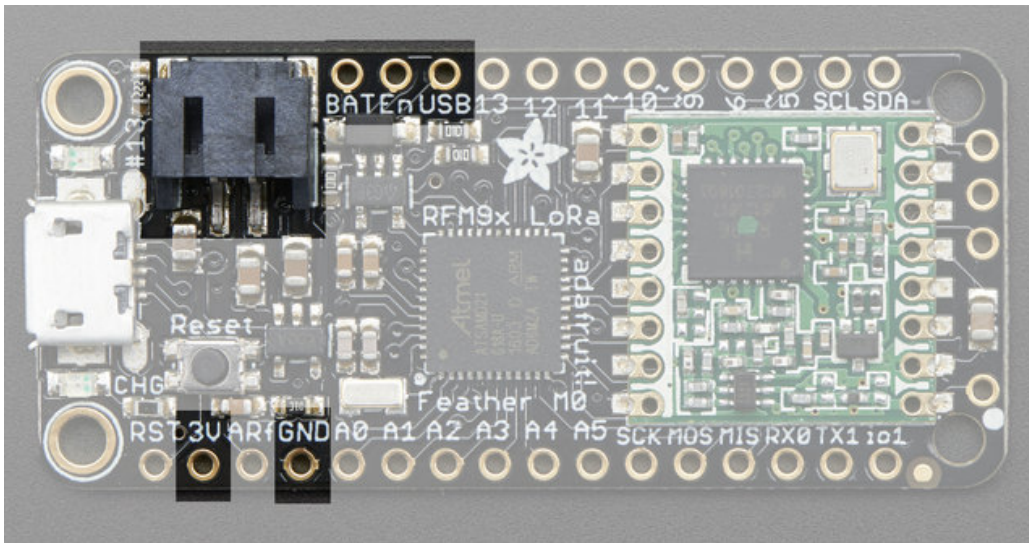
The JST connector polarity is matched to Adafruit LiPoly batteries. Using wrong polarity batteries can destroy your Feather



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

Power supplies

You have a lot of power supply options here! We bring out the **BAT** pin, which is tied to the lipoly JST connector, as well as **USB** which is the +5V from USB if connected. We also have the **3V** pin which has the output from the 3.3V regulator. We use a 500mA peak 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.



Measuring Battery

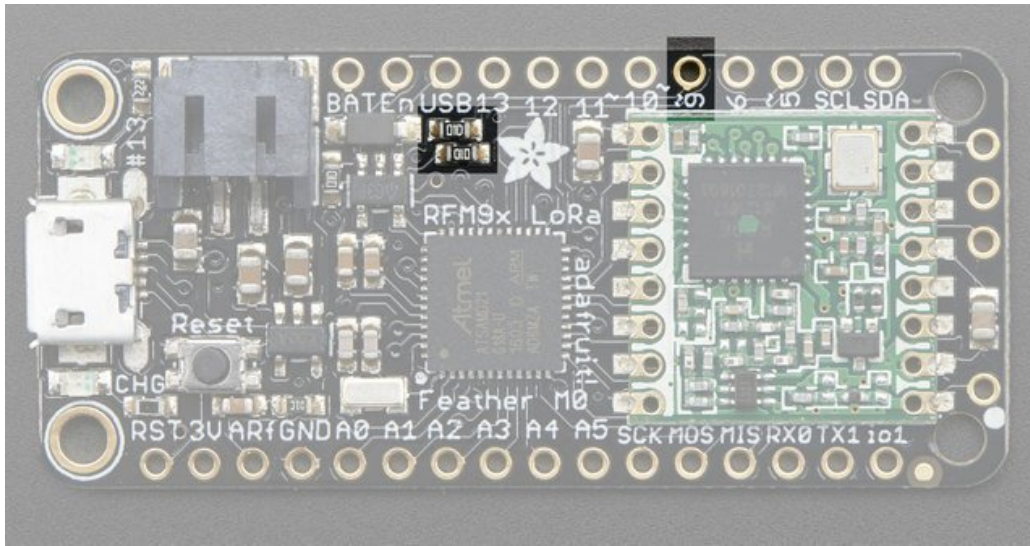
If you're running off of a battery, chances are you wanna know what the voltage is at! That way you can tell when the battery needs recharging. Lipoly batteries are 'maxed out' at 4.2V and stick around 3.7V for much of the battery

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.

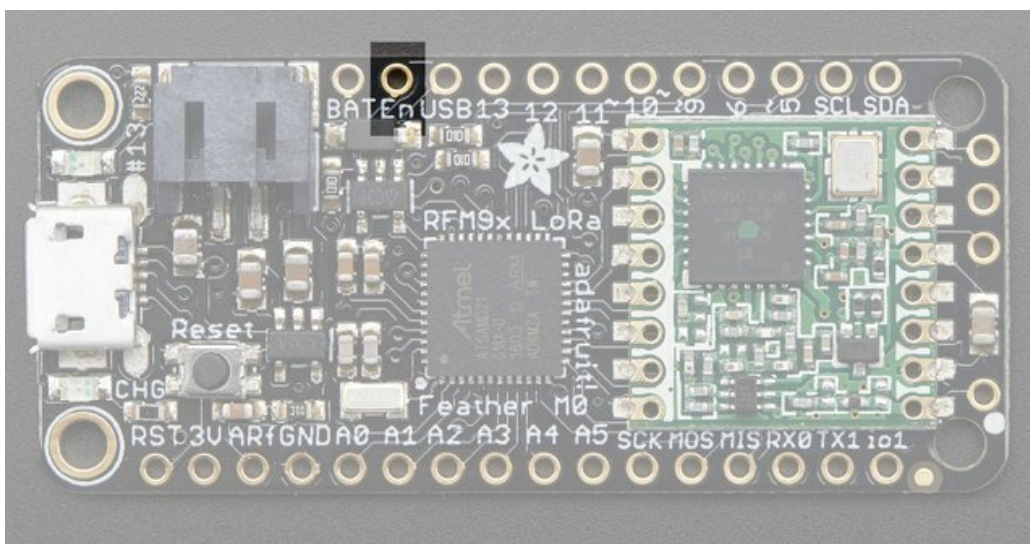
```
#define VBATPIN A7
```

```
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

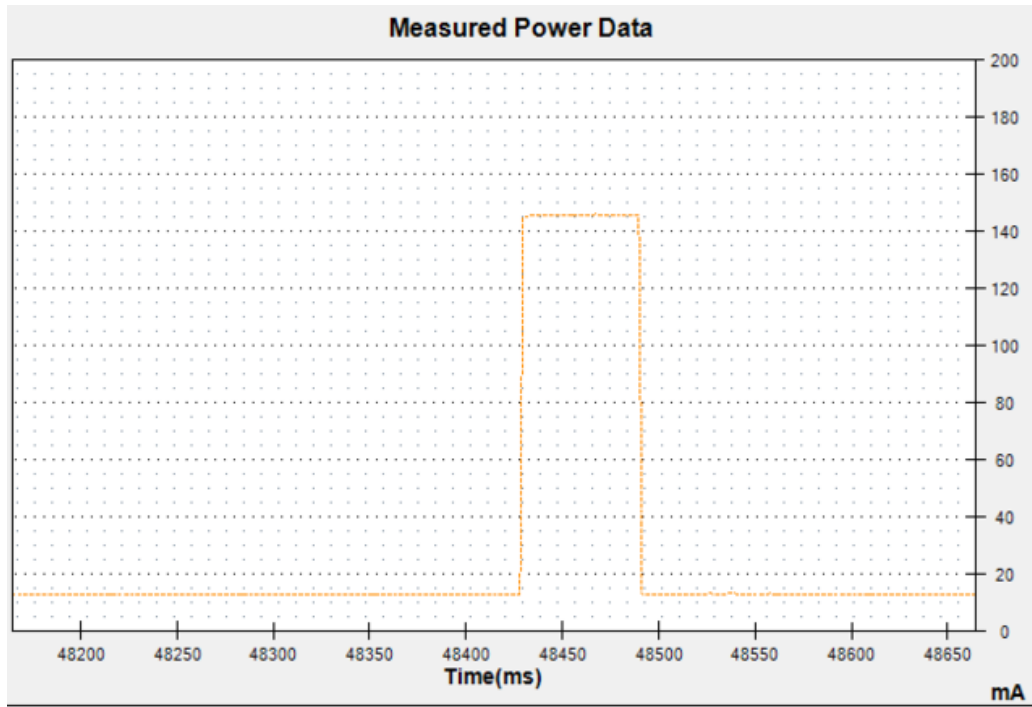


Radio Power Draw

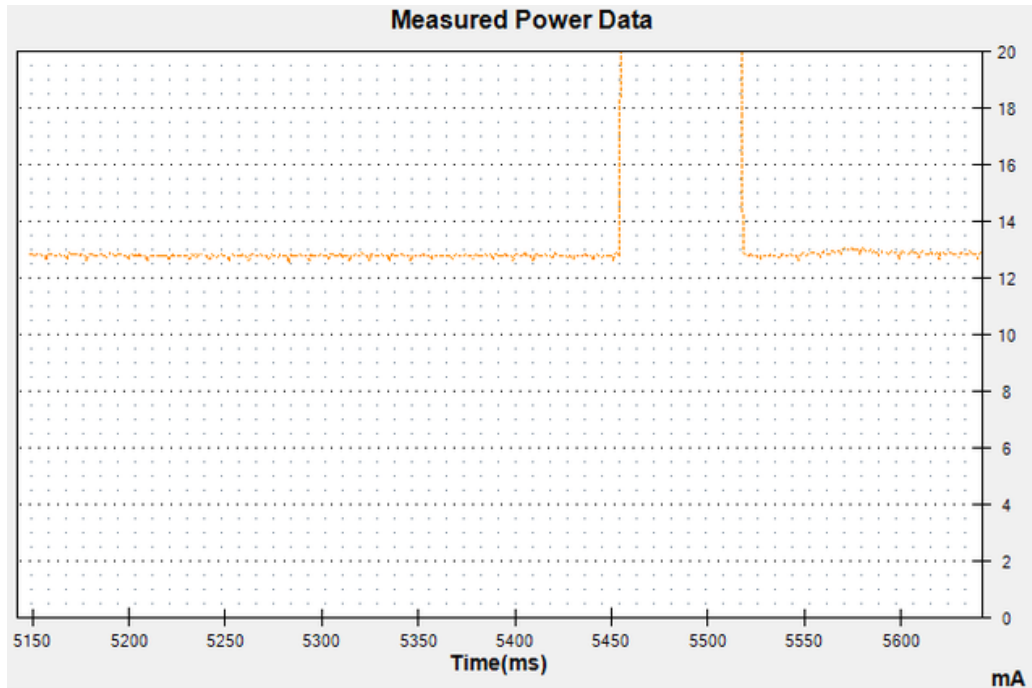
Transmitting Data

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

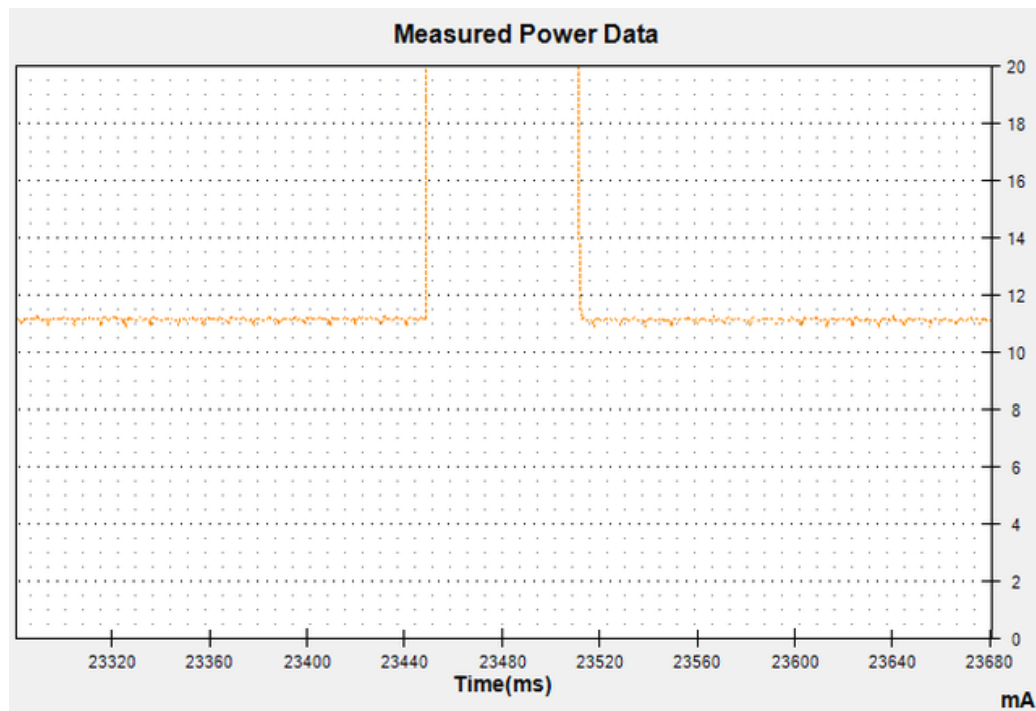
For example, here is the Feather with RFM9x 900MHz radio set up for +20dBm power, transmitting a data payload of 20 bytes. Transmits take about 130mA for 70ms



The ~13mA quiescent current is the current draw for listening (~2mA) plus ~11mA for the microcontroller. This can be reduced to almost nothing with proper sleep modes and not putting the module in active listen mode!



You can put the module into sleep mode by calling `radio.sleep()`; which will save you about 2mA

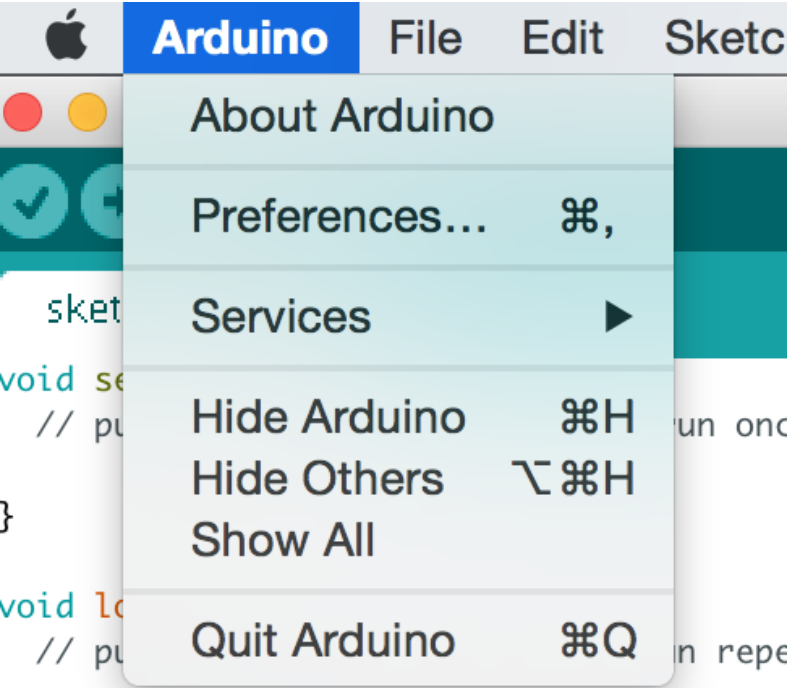


Arduino IDE Setup

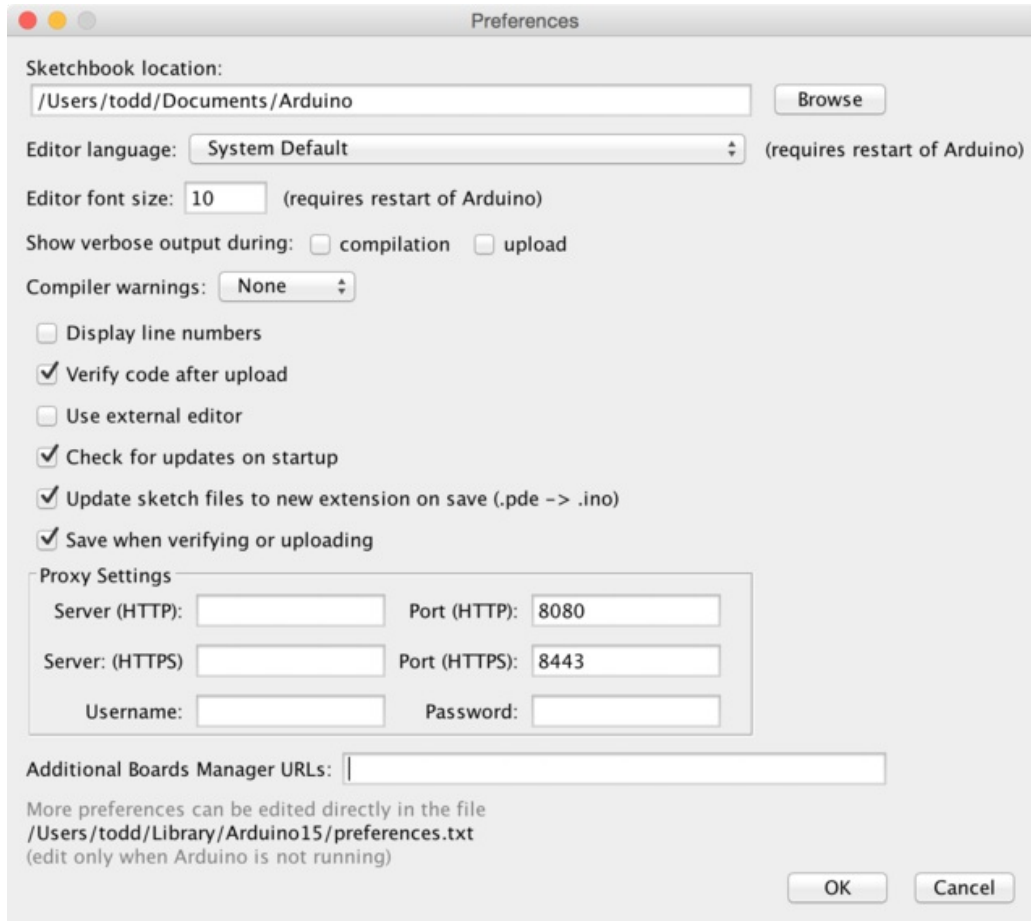
The first thing you will need to do is to download the latest release of the Arduino IDE. You will need to be using **version 1.6.4** or higher for this guide.

[Arduino IDE v1.6.4+ Download](http://adafru.it/f1P)
<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*.



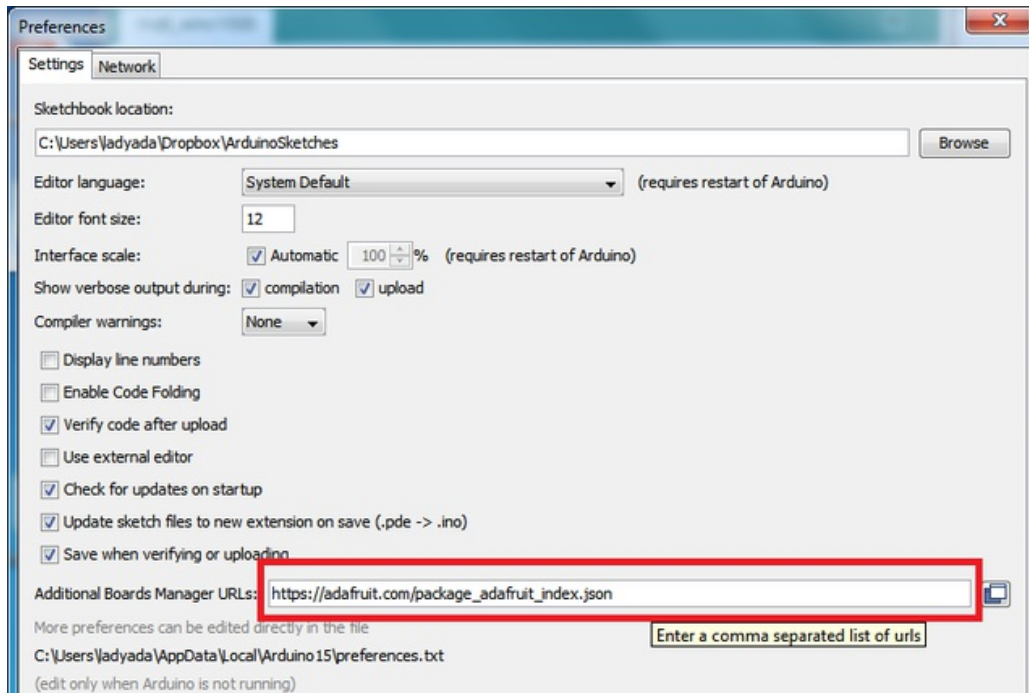
A dialog will pop up just like the one shown below.



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) (<http://adafru.it/f7U>). We will only need to add one URL to the IDE in this example, but ***you can add multiple URLs by separating them with commas***. Copy and paste the link below into the **Additional Boards Manager URLs** option in the Arduino IDE preferences.

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



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

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

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

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

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

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

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

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

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

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

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

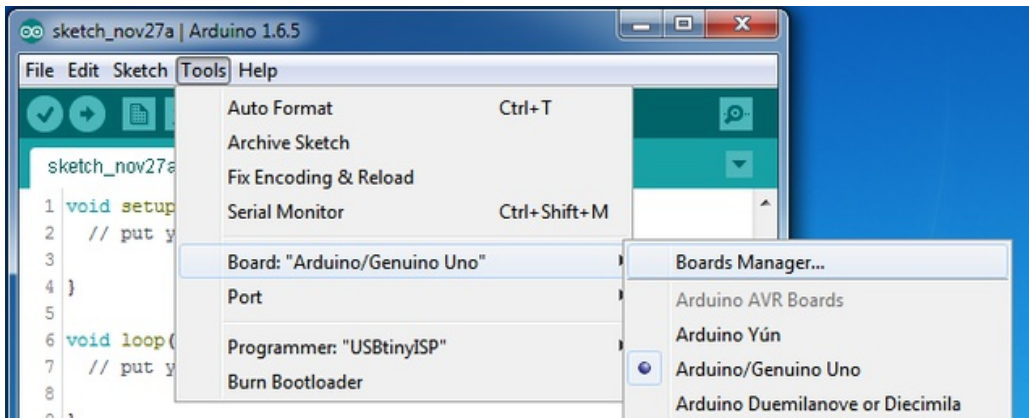
I'm trying to upload to my Feather M0, and I get this error "Connecting to programmer: .avrdude: butterfly_recv(): programmer is not responding"

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

Using with Arduino IDE

Since the Feather M0 uses an ATSAM21 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.

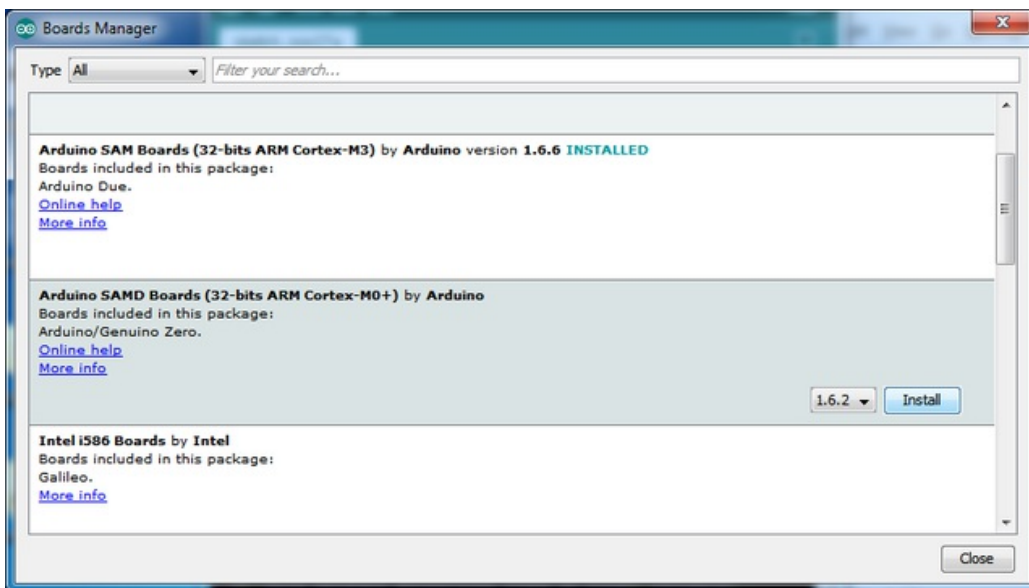
Now that you have added the appropriate URLs to the Arduino IDE preferences, you can open the **Boards Manager** by navigating to the **Tools->Board** menu.

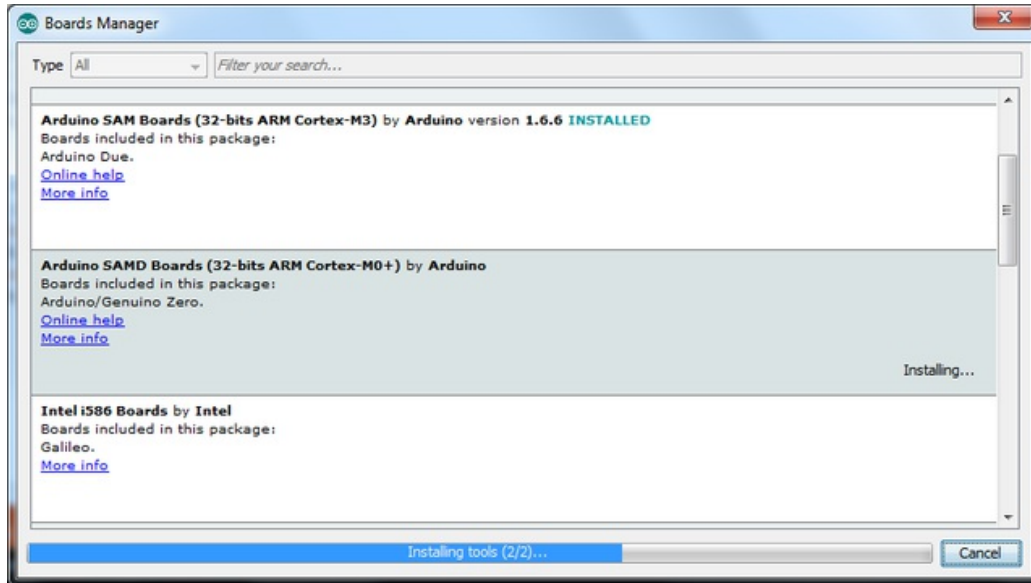


Once the Board Manager opens, click on the category drop down menu on the top left hand side of the window and select **Contributed**. You will then be able to select and install the boards supplied by the URLs added to the preferences.

Install SAMD Support

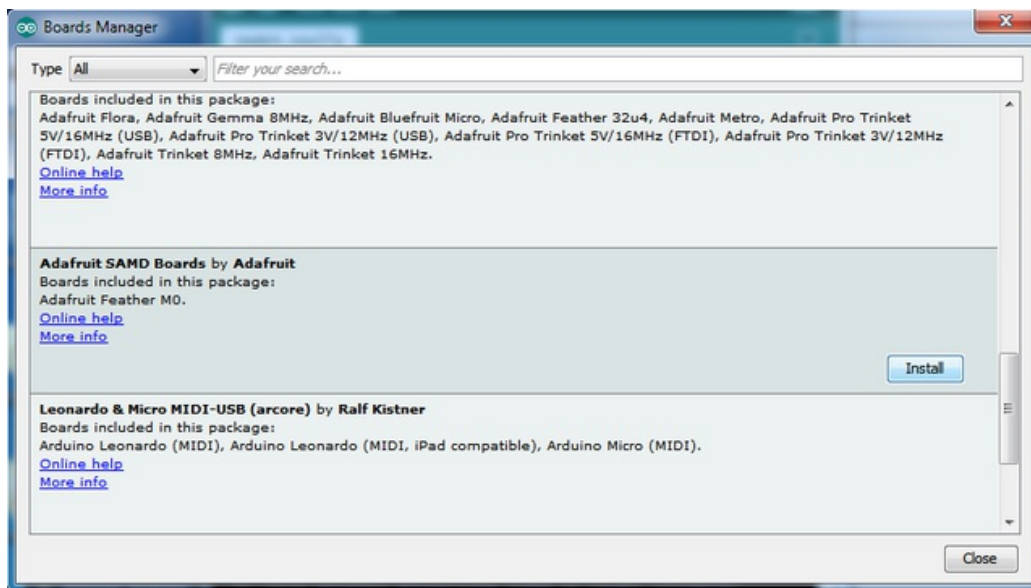
First up, install the **Arduino SAMD Boards** version **1.6.2** or later





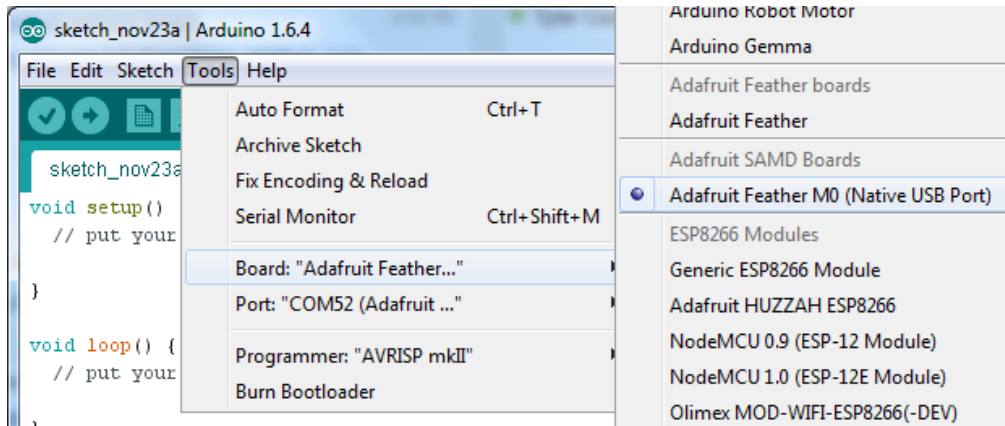
Install Adafruit SAMD

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



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.



Install Drivers (Windows Only)

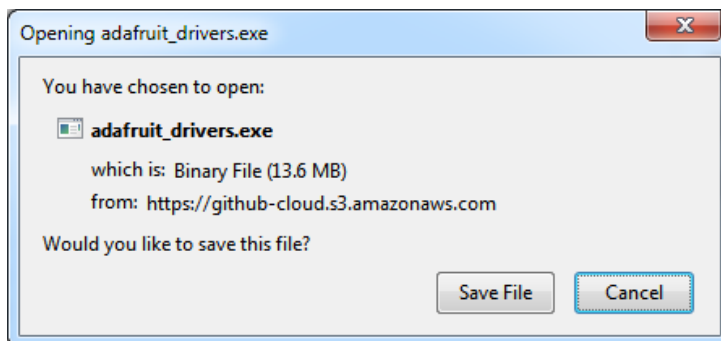
When you plug in the Feather, you'll need to possibly install a driver

Click below to download our Driver Installer

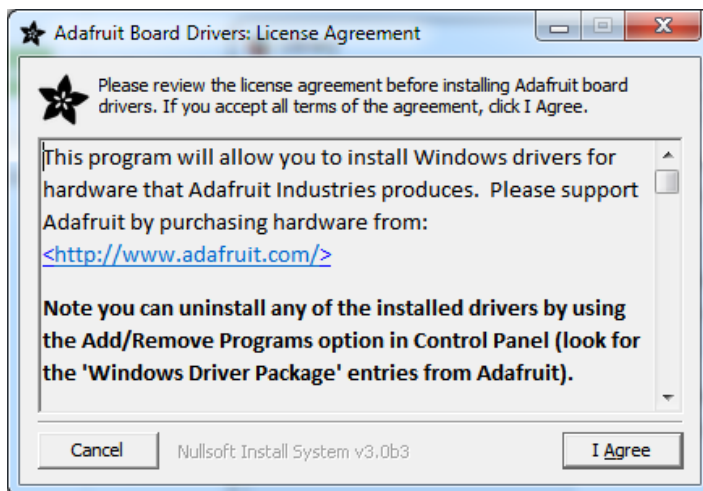
[Download Adafruit Driver Installer](http://adafru.it/mai)

<http://adafru.it/mai>

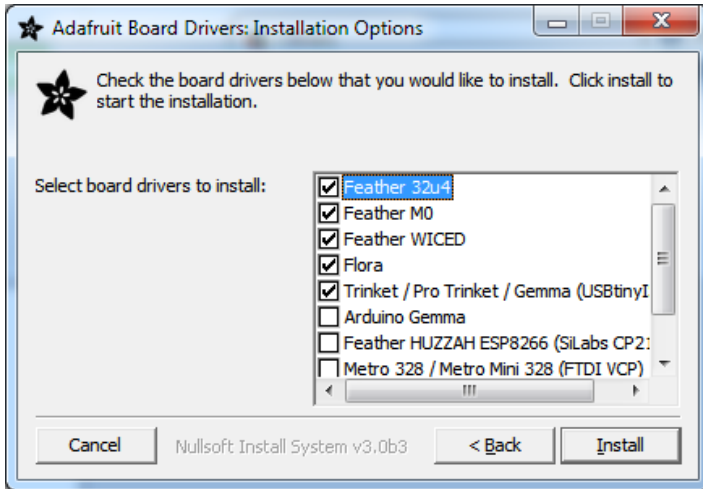
Download and run the installer



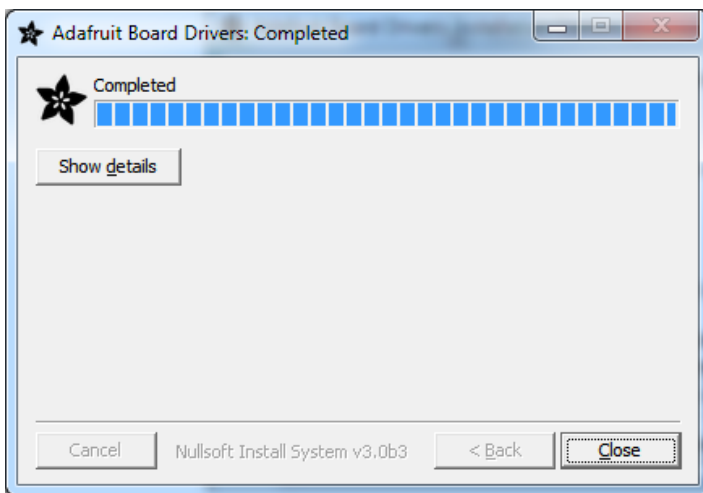
Run the installer! Since we bundle the SiLabs and FTDI drivers as well, you'll need to click through the license



Select which drivers you want to install:



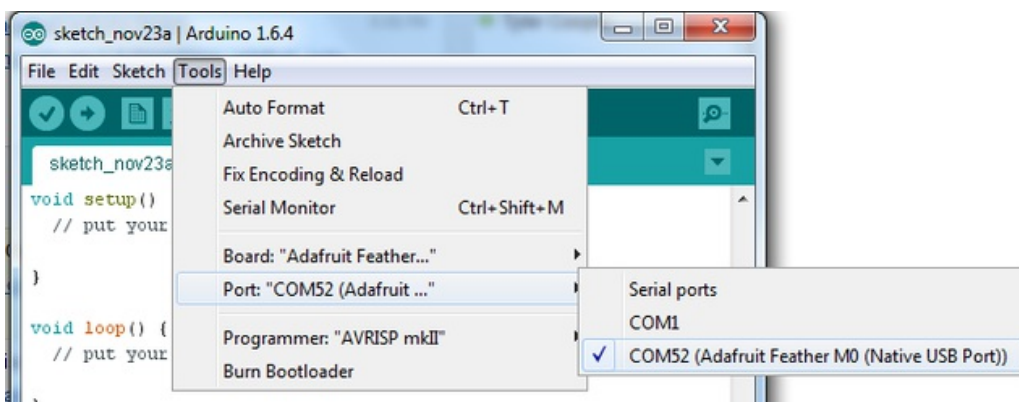
Click **Install** to do the installin'



Blink

Now you can upload your first blink sketch!

Plug in the Feather 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!



Now load up the Blink example

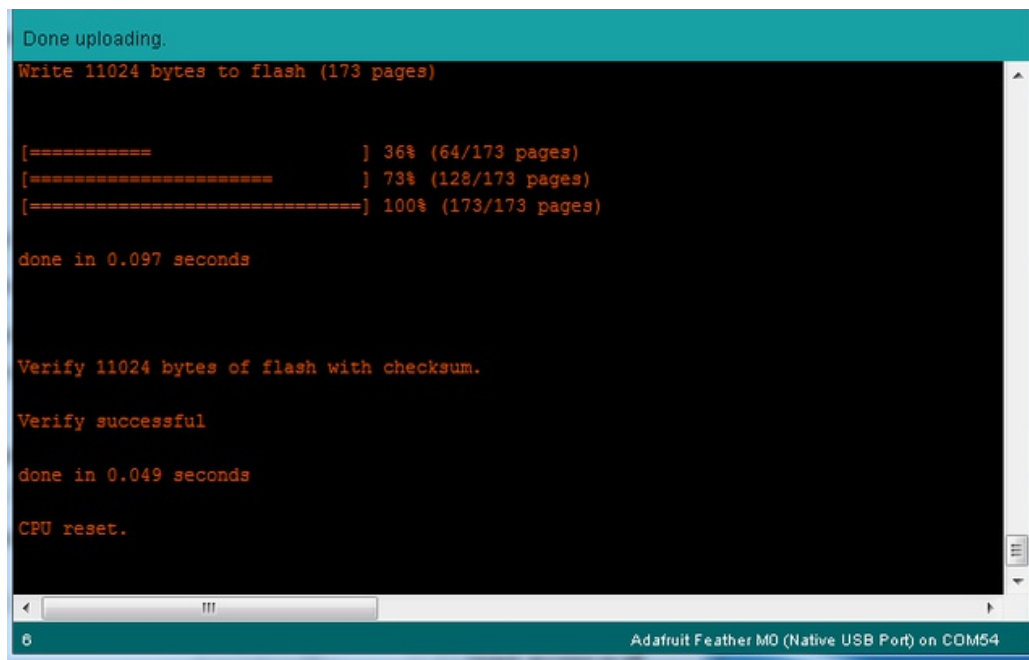
```
// the setup function runs once when you press reset or power the board
void setup() {
  // initialize digital pin 13 as an output.
  pinMode(13, OUTPUT);
}

// 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.

Successful 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

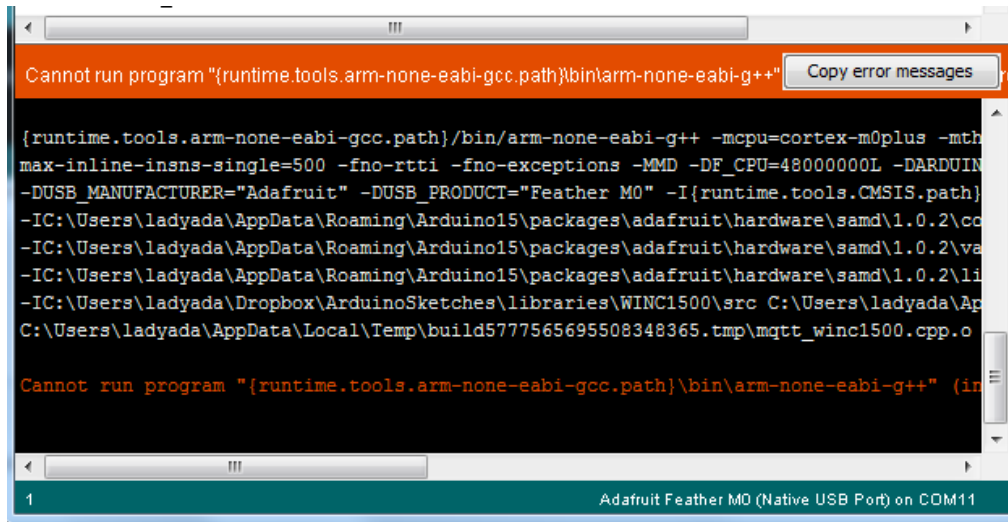


Compilation Issues

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

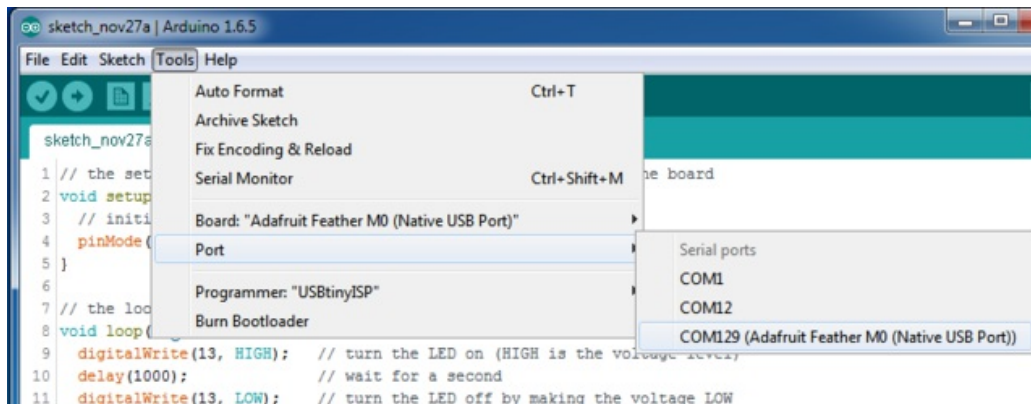


Manually bootloading

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.



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). (<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). (<http://adafru.it/iOE>)

Adapting Sketches to M0

The ATSAM21 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 `isAnalogReference(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:



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 AVR's, 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/IFS>)

How Much RAM Available?

The ATSAM21G18 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, it's a little easier, simply add **const** before the variable name:

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

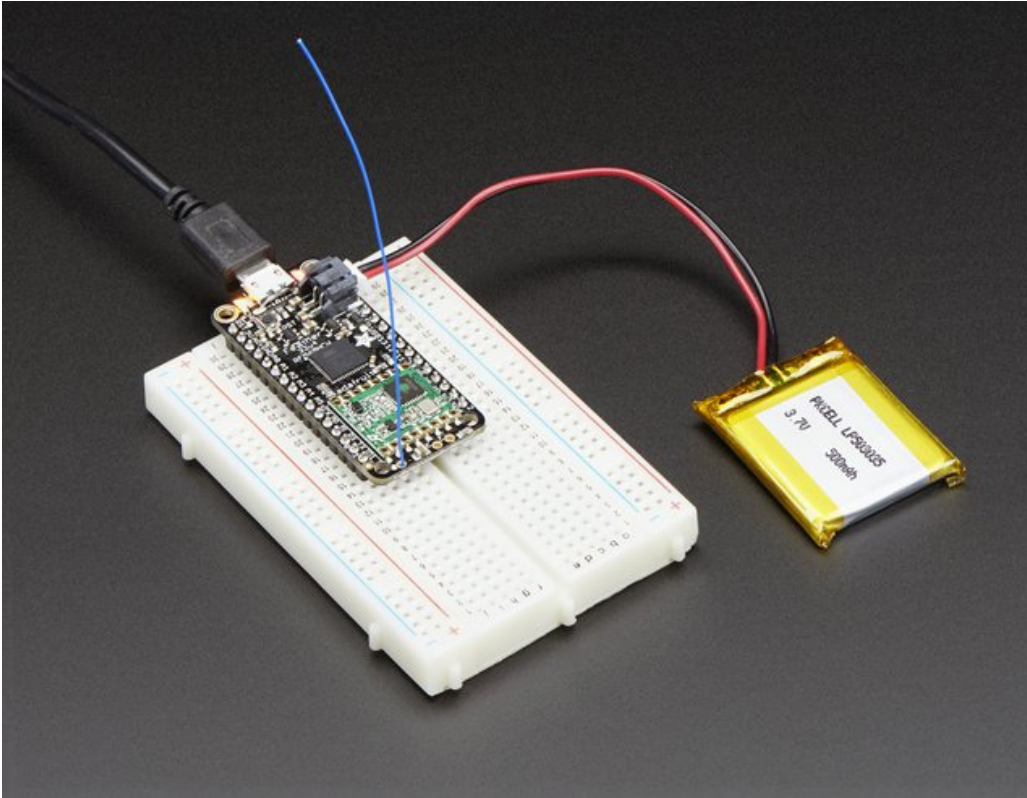
That string is now in FLASH. You can manipulate the string just like RAM data, the compiler will automatically read from FLASH so you don't need special progmem-knowledgeable functions.

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 RFM9X 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 adjustable but it's common to stick to around 19.2 Kbps (that's 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.

Arduino Library

These radios have really excellent code already written, so rather than coming up with a new standard we suggest using existing libraries such as [AirSpayce's Radiohead library](http://adafruit.it/mCA) (<http://adafruit.it/mCA>) which also supports a vast number of other radios.

This is a really great Arduino Library, so please support them in thanks for their efforts!

RadioHead RFM9x Library example

To begin talking to the radio, you will need to download the [RadioHead library](http://adafruit.it/mCA) (<http://adafruit.it/mCA>). You can do

that by visiting the github repo and manually downloading or, easier, just click this button to download the zip corresponding to version 1.62

Note that while all the code in the examples below are based on this version you can [visit the RadioHead documentation page to get the most recent version which may have bug-fixes or more functionality](http://adafru.it/mCA) (<http://adafru.it/mCA>)

[RadioHead-1.62.zip](http://adafru.it/q6f)

<http://adafru.it/q6f>

Uncompress the zip and find the folder named **RadioHead** and check that the **RadioHead** folder contains **RH_RF95.cpp** and **RH_RF95.h** (as well as a few dozen other files for radios that are supported)

Place the **RadioHead** library folder your **arduinorsketchfolder/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 RF95_FREQ 915.0` line and change that to 433.0 if you are using the 433MHz version of the LoRa radio!

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!

```
// Feather9x_TX
// -*- mode: C++ -*-
// Example sketch showing how to create a simple messaging client (transmitter)
// with the RH_RF95 class. RH_RF95 class does not provide for addressing or
// reliability, so you should only use RH_RF95 if you do not need the higher
// level messaging abilities.
// It is designed to work with the other example Feather9x_RX

#include <SPI.h>
#include <RH_RF95.h>

/* for feather32u4 */
#define RFM95_CS 8
#define RFM95_RST 4
#define RFM95_INT 7

/* for feather m0
#define RFM95_CS 8
#define RFM95_RST 4
#define RFM95_INT 3
*/
```

```

/* for shield
#define RFM95_CS 10
#define RFM95_RST 9
#define RFM95_INT 7
*/

/* for ESP w/featherwing
#define RFM95_CS 2 // "E"
#define RFM95_RST 16 // "D"
#define RFM95_INT 15 // "B"
*/

/* Feather 32u4 w/wing
#define RFM95_RST 11 // "A"
#define RFM95_CS 10 // "B"
#define RFM95_INT 2 // "SDA" (only SDA/SCL/RX/TX have IRQ!)
*/

/* Feather m0 w/wing
#define RFM95_RST 11 // "A"
#define RFM95_CS 10 // "B"
#define RFM95_INT 6 // "D"
*/

/* Teensy 3.x w/wing
#define RFM95_RST 9 // "A"
#define RFM95_CS 10 // "B"
#define RFM95_INT 4 // "C"
*/

// Change to 434.0 or other frequency, must match RX's freq!
#define RF95_FREQ 915.0

// Singleton instance of the radio driver
RH_RF95 rf95(RFM95_CS, RFM95_INT);

void setup()
{
  pinMode(RFM95_RST, OUTPUT);
  digitalWrite(RFM95_RST, HIGH);

  while (!Serial);
  Serial.begin(9600);
  delay(100);

  Serial.println("Feather LoRa TX Test!");

  // manual reset
  digitalWrite(RFM95_RST, LOW);
  delay(10);
  digitalWrite(RFM95_RST, HIGH);
  delay(10);

  while (!rf95.init()) {
    Serial.println("LoRa radio init failed");
    while (1);
  }
  Serial.println("LoRa radio init OK!");

  // Defaults after init are 434.0MHz, modulation GFSK_Rb250Fd250, +13dbM
  if (!rf95.setFrequency(RF95_FREQ)) {
    Serial.println("setFrequency failed");
    while (1);
  }
}

```

```

Serial.print("Set Freq to: "); Serial.println(RF95_FREQ);

// Defaults after init are 434.0MHz, 13dBm, Bw = 125 kHz, Cr = 4/5, Sf = 128chips/symbol, CRC on

// The default transmitter power is 13dBm, using PA_BOOST.
// If you are using RFM95/96/97/98 modules which uses the PA_BOOST transmitter pin, then
// you can set transmitter powers from 5 to 23 dBm:
rf95.setTxPower(23, false);
}

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

void loop()
{
  Serial.println("Sending to rf95_server");
  // Send a message to rf95_server

  char radiopacket[20] = "Hello World # ";
  itoa(packetnum++, radiopacket+13, 10);
  Serial.print("Sending "); Serial.println(radiopacket);
  radiopacket[19] = 0;

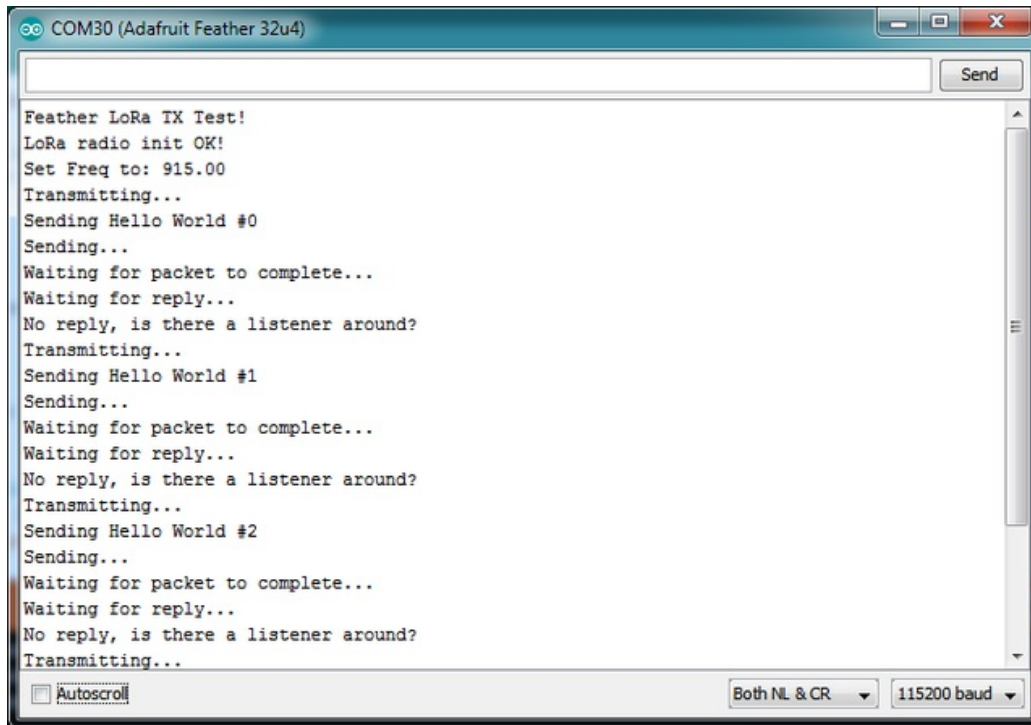
  Serial.println("Sending..."); delay(10);
  rf95.send((uint8_t *)radiopacket, 20);

  Serial.println("Waiting for packet to complete..."); delay(10);
  rf95.waitPacketSent();
  // Now wait for a reply
  uint8_t buf[RH_RF95_MAX_MESSAGE_LEN];
  uint8_t len = sizeof(buf);

  Serial.println("Waiting for reply..."); delay(10);
  if (rf95.waitAvailableTimeout(1000))
  {
    // Should be a reply message for us now
    if (rf95.recv(buf, &len))
    {
      Serial.print("Got reply: ");
      Serial.println((char*)buf);
      Serial.print("RSSI: ");
      Serial.println(rf95.lastRssi(), DEC);
    }
    else
    {
      Serial.println("Receive failed");
    }
  }
  else
  {
    Serial.println("No reply, is there a listener around?");
  }
  delay(1000);
}

```

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!

Make sure the `#define RF95_FREQ 915.0` matches your transmitter Feather!

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!

```
// Feather9x_RX
// -*- mode: C++ -*-
// Example sketch showing how to create a simple messaging client (receiver)
// with the RH_RF95 class. RH_RF95 class does not provide for addressing or
// reliability, so you should only use RH_RF95 if you do not need the higher
// level messaging abilities.
// It is designed to work with the other example Feather9x_TX

#include <SPI.h>
#include <RH_RF95.h>

/* for feather32u4 */
#define RFM95_CS 8
#define RFM95_RST 4
#define RFM95_INT 7

/* for feather m0
#define RFM95_CS 8
#define RFM95_RST 4
#define RFM95_INT 3
*/

/* for shield
```

```

#define RFM95_CS 10
#define RFM95_RST 9
#define RFM95_INT 7
*/

/* for ESP w/featherwing
#define RFM95_CS 2 // "E"
#define RFM95_RST 16 // "D"
#define RFM95_INT 15 // "B"
*/

/* Feather 32u4 w/wing
#define RFM95_RST 11 // "A"
#define RFM95_CS 10 // "B"
#define RFM95_INT 2 // "SDA" (only SDA/SCL/RX/TX have IRQ!)
*/

/* Feather m0 w/wing
#define RFM95_RST 11 // "A"
#define RFM95_CS 10 // "B"
#define RFM95_INT 6 // "D"
*/

/* Teensy 3.x w/wing
#define RFM95_RST 9 // "A"
#define RFM95_CS 10 // "B"
#define RFM95_INT 4 // "C"
*/

// Change to 434.0 or other frequency, must match RX's freq!
#define RF95_FREQ 915.0

// Singleton instance of the radio driver
RH_RF95 rf95(RFM95_CS, RFM95_INT);

// Blinky on receipt
#define LED 13

void setup()
{
  pinMode(LED, OUTPUT);
  pinMode(RFM95_RST, OUTPUT);
  digitalWrite(RFM95_RST, HIGH);

  while (!Serial);
  Serial.begin(9600);
  delay(100);

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

  // manual reset
  digitalWrite(RFM95_RST, LOW);
  delay(10);
  digitalWrite(RFM95_RST, HIGH);
  delay(10);

  while (!rf95.init()) {
    Serial.println("LoRa radio init failed");
    while (1);
  }
  Serial.println("LoRa radio init OK!");

  // Defaults after init are 434.0MHz, modulation GFSK_Rb250Fd250, +13dbM
  if (!rf95.setFrequency(RF95_FREQ)) {

```

```

    Serial.println("setFrequency failed");
    while (1);
}
Serial.print("Set Freq to: "); Serial.println(RF95_FREQ);

// Defaults after init are 434.0MHz, 13dBm, Bw = 125 kHz, Cr = 4/5, Sf = 128chips/symbol, CRC on

// The default transmitter power is 13dBm, using PA_BOOST.
// If you are using RFM95/96/97/98 modules which uses the PA_BOOST transmitter pin, then
// you can set transmitter powers from 5 to 23 dBm:
rf95.setTxPower(23, false);
}

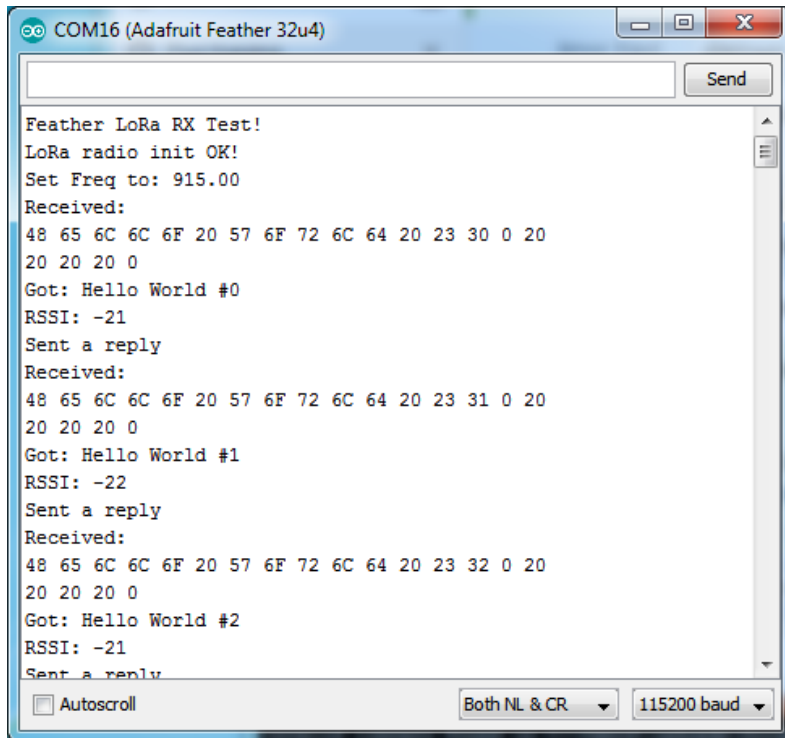
void loop()
{
  if (rf95.available())
  {
    // Should be a message for us now
    uint8_t buf[RH_RF95_MAX_MESSAGE_LEN];
    uint8_t len = sizeof(buf);

    if (rf95.recv(buf, &len))
    {
      digitalWrite(LED, HIGH);
      RH_RF95::printBuffer("Received: ", buf, len);
      Serial.print("Got: ");
      Serial.println((char*)buf);
      Serial.print("RSSI: ");
      Serial.println(rf95.lastRssi(), DEC);

      // Send a reply
      uint8_t data[] = "And hello back to you";
      rf95.send(data, sizeof(data));
      rf95.waitPacketSent();
      Serial.println("Sent a reply");
      digitalWrite(LED, LOW);
    }
    else
    {
      Serial.println("Receive failed");
    }
  }
}

```

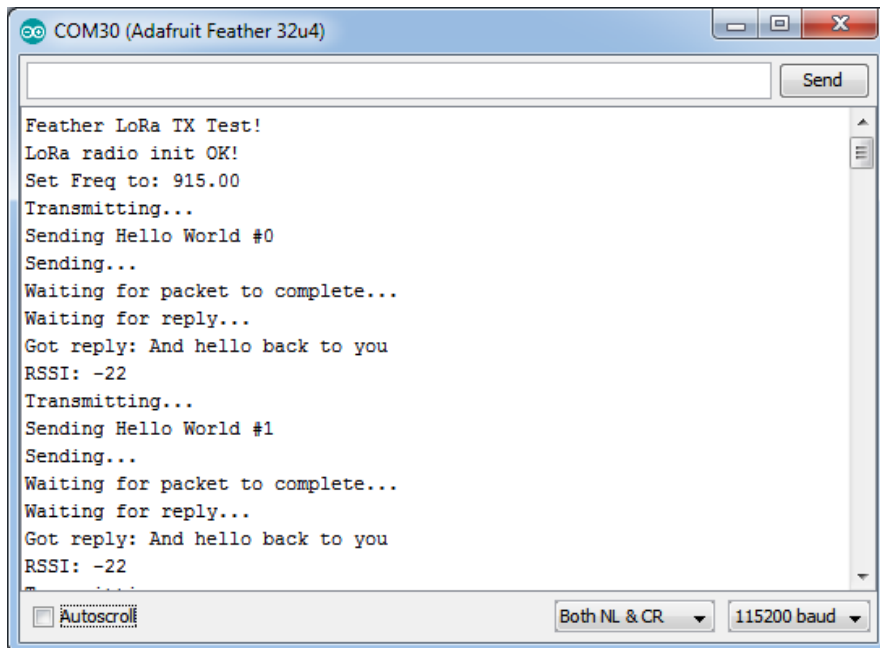
Now open up the Serial console on the receiver, while also checking in on the transmitter's serial console. You should see the receiver is...well, receiving packets



```
COM16 (Adafruit Feather 32u4)
Feather LoRa RX Test!
LoRa radio init OK!
Set Freq to: 915.00
Received:
48 65 6C 6C 6F 20 57 6F 72 6C 64 20 23 30 0 20
20 20 20 0
Got: Hello World #0
RSSI: -21
Sent a reply
Received:
48 65 6C 6C 6F 20 57 6F 72 6C 64 20 23 31 0 20
20 20 20 0
Got: Hello World #1
RSSI: -22
Sent a reply
Received:
48 65 6C 6C 6F 20 57 6F 72 6C 64 20 23 32 0 20
20 20 20 0
Got: Hello World #2
RSSI: -21
Sent a reply
Autoscroll Both NL & CR 115200 baud
```

You can see that the library example prints out the hex-bytes received 48 65 6C 6C 6F 20 57 6F 72 6C 64 20 23 30 0 20 20 20 0, as well as the ASCII 'string' Hello World. Then it will send a reply.

And, on the transmitter side, it is now printing that it got a reply after each transmission And hello back to you because it got a reply from the receiver



```
COM30 (Adafruit Feather 32u4)
Feather LoRa TX Test!
LoRa radio init OK!
Set Freq to: 915.00
Transmitting...
Sending Hello World #0
Sending...
Waiting for packet to complete...
Waiting for reply...
Got reply: And hello back to you
RSSI: -22
Transmitting...
Sending Hello World #1
Sending...
Waiting for packet to complete...
Waiting for reply...
Got reply: And hello back to you
RSSI: -22
Autoscroll Both NL & CR 115200 baud
```

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 setup

Feather Radio Pinout

This is the pinout setup for all **Feather 32u4** RFM9X's:

```
/* for feather32u4 */
#define RFM95_CS 8
#define RFM95_RST 4
#define RFM95_INT 7
```

This is the pinout for all **Feather M0** RFM9X's:

```
/* for feather m0 */
#define RFM95_CS 8
#define RFM95_RST 4
#define RFM95_INT 3
```

Frequency

You can dial in the frequency you want the radio to communicate on, such as 915.0, 434.0 or 868.0 or any number really. Different countries/ITU Zones have different ISM bands so make sure you're using those or if you are licensed, those frequencies you may use

```
// Change to 434.0 or other frequency, must match RX's freq!
#define RF95_FREQ 915.0
```

You can then instantiate the radio object with our custom pin numbers.

```
// Singleton instance of the radio driver
RH_RF95 rf95(RFM95_CS, RFM95_INT);
```

Setup

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

```
void setup()
{
  pinMode(LED, OUTPUT);
  pinMode(RFM95_RST, OUTPUT);
  digitalWrite(RFM95_RST, HIGH);

  while (!Serial); // wait until serial console is open, remove if not tethered to computer
  Serial.begin(9600);
  delay(100);
  Serial.println("Feather LoRa RX Test!");

  // manual reset
  digitalWrite(RFM95_RST, LOW);
  delay(10);
  digitalWrite(RFM95_RST, HIGH);
  delay(10);
```

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 a call to **init()**. Once initialized, you can set the frequency. You can also configure the output power level, the number ranges from 5 to 23. Start with the highest power level (23) and then scale down as necessary

```

while (!rf95.init()) {
  Serial.println("LoRa radio init failed");
  while (1);
}
Serial.println("LoRa radio init OK!");

// Defaults after init are 434.0MHz, modulation GFSK_Rb250Fd250, +13dBm
if (!rf95.setFrequency(RF95_FREQ)) {
  Serial.println("setFrequency failed");
  while (1);
}
Serial.print("Set Freq to: "); Serial.println(RF95_FREQ);

// Defaults after init are 434.0MHz, 13dBm, Bw = 125 kHz, Cr = 4/5, Sf = 128chips/symbol, CRC on

// The default transmitter power is 13dBm, using PA_BOOST.
// If you are using RFM95/96/97/98 modules which uses the PA_BOOST transmitter pin, then
// you can set transmitter powers from 5 to 23 dBm:
rf95.setTxPower(23, false);

```

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

```

void loop()
{
  delay(1000); // Wait 1 second between transmits, could also 'sleep' here!
  Serial.println("Transmitting..."); // Send a message to rf95_server

  char radiopacket[20] = "Hello World #    ";
  itoa(packetnum++, radiopacket+13, 10);
  Serial.print("Sending "); Serial.println(radiopacket);
  radiopacket[19] = 0;

  Serial.println("Sending..."); delay(10);
  rf95.send((uint8_t *)radiopacket, 20);

  Serial.println("Waiting for packet to complete..."); delay(10);
  rf95.waitPacketSent();
}

```

Its pretty simple, the delay does the waiting, you can replace that with low power sleep code. Then it generates the packet and appends a number that increases every tx. Then it simply calls **send** to transmit the data, and passes in the array of data and the length of the data.

Note that this does not any addressing or subnetworking - if you want to make sure the packet goes to a particular radio, you may have to add an identifier/address byte on your own!

Then you call **waitPacketSent()** to wait until the radio is done transmitting. You will not get an automatic acknowledgement, from the other radio unless it knows to send back a packet. Think of it like the 'UDP' of radio - the data is sent, but its not certain it was received! Also, there will not be any automatic retries.

Receiver Code

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

```

void loop()
{
  if (rf95.available())
  {
    // Should be a message for us now

```

```

uint8_t buf[RH_RF95_MAX_MESSAGE_LEN];
uint8_t len = sizeof(buf);

if (rf95.recv(buf, &len))
{
  digitalWrite(LED, HIGH);
  RH_RF95::printBuffer("Received: ", buf, len);
  Serial.print("Got: ");
  Serial.println((char*)buf);
  Serial.print("RSSI: ");
  Serial.println(rf95.lastRssi(), DEC);
}

```

Instead of transmitting, it is constantly checking if there's any data packets that have been received **available()** will return true if a packet with proper error-correction was received. If so, the receiver prints it out in hex and also as a 'character string'

It also prints out the RSSI which is the receiver signal strength indicator. This number will range from about -15 to about -100. The larger the number (-15 being the highest you'll likely see) the stronger the signal.

Once done it will automatically reply, which is a way for the radios to know that there was an acknowledgement

```

// Send a reply
uint8_t data[] = "And hello back to you";
rf95.send(data, sizeof(data));
rf95.waitPacketSent();
Serial.println("Sent a reply");

```

It simply sends back a string and waits till the reply is completely sent

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 advertised! 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 different 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 antenna book is recommended if you want to learn how to do the modeling and analysis \(http://adafruit.it/sdN\)](http://adafruit.it/sdN)

But nothing beats actual tests in your environment!

For the LoRa version:

- For the RFM69 version:

- For both:

- ## 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)

