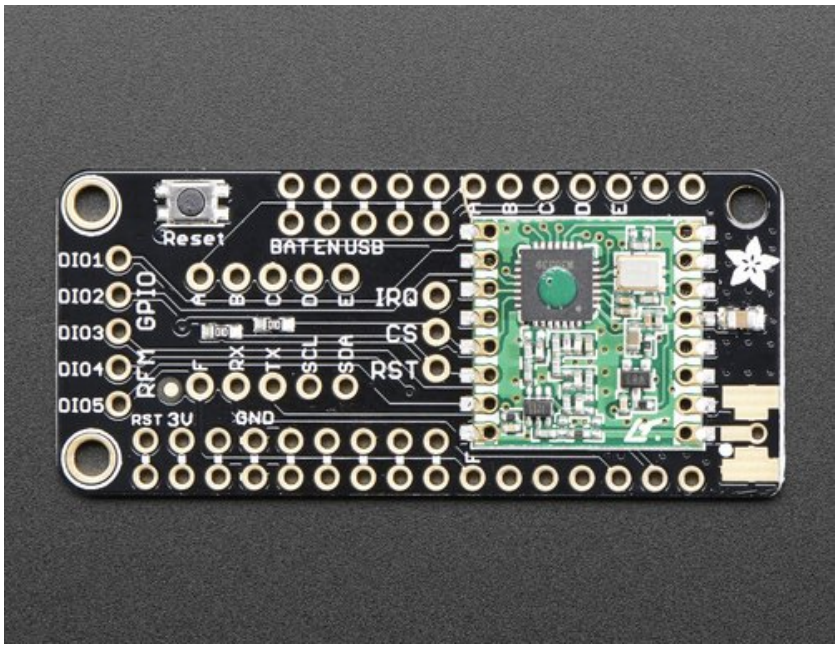
Radio FeatherWing

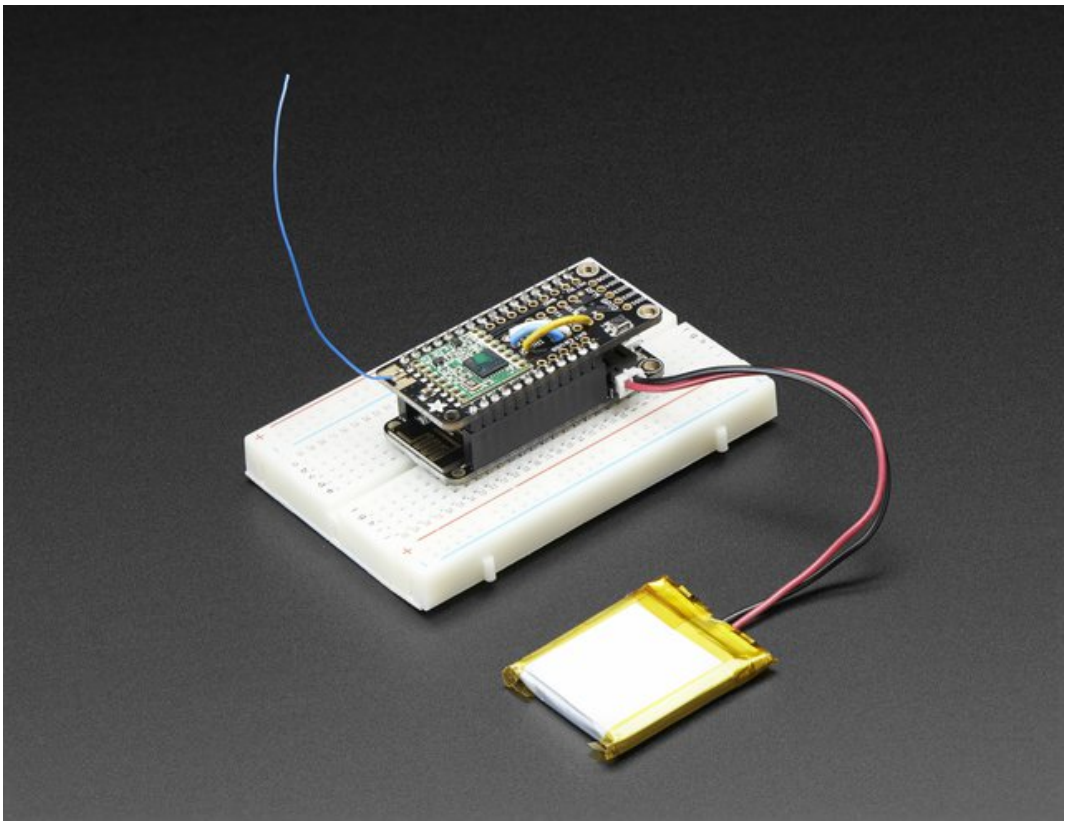
Created by lady ada



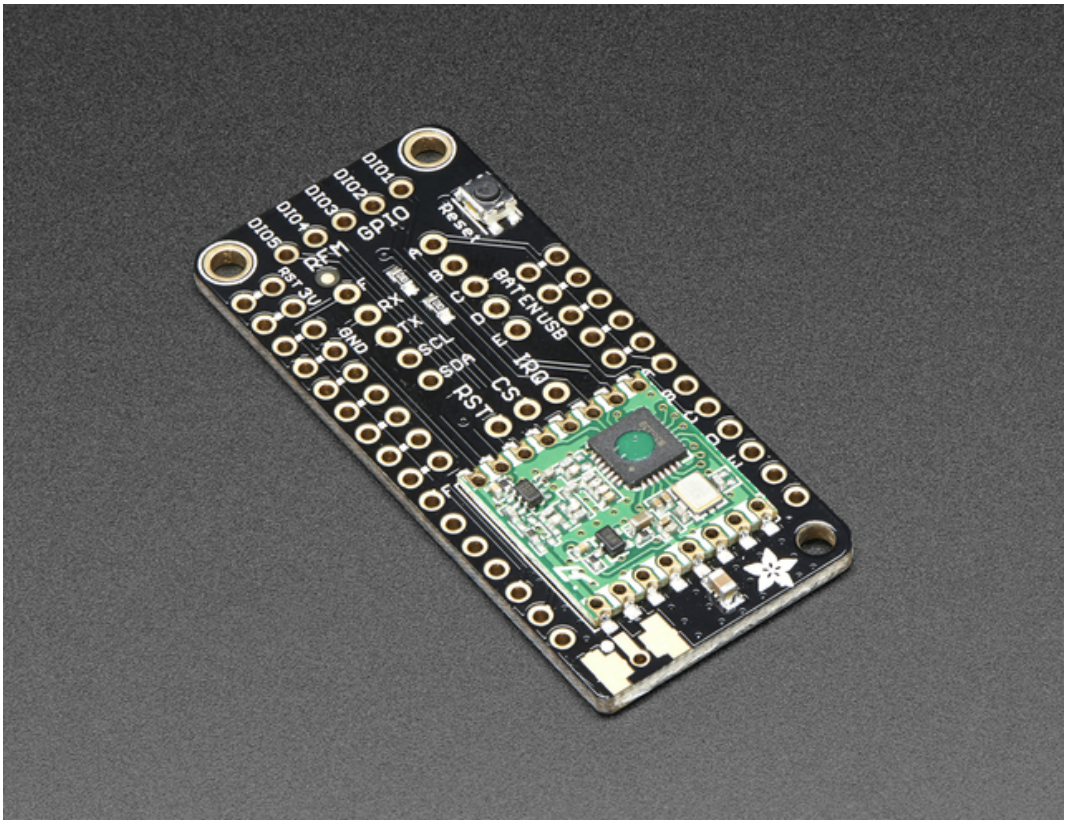
Last updated on 2016-11-03 03:18:57 PM UTC

Guide Contents

Overview



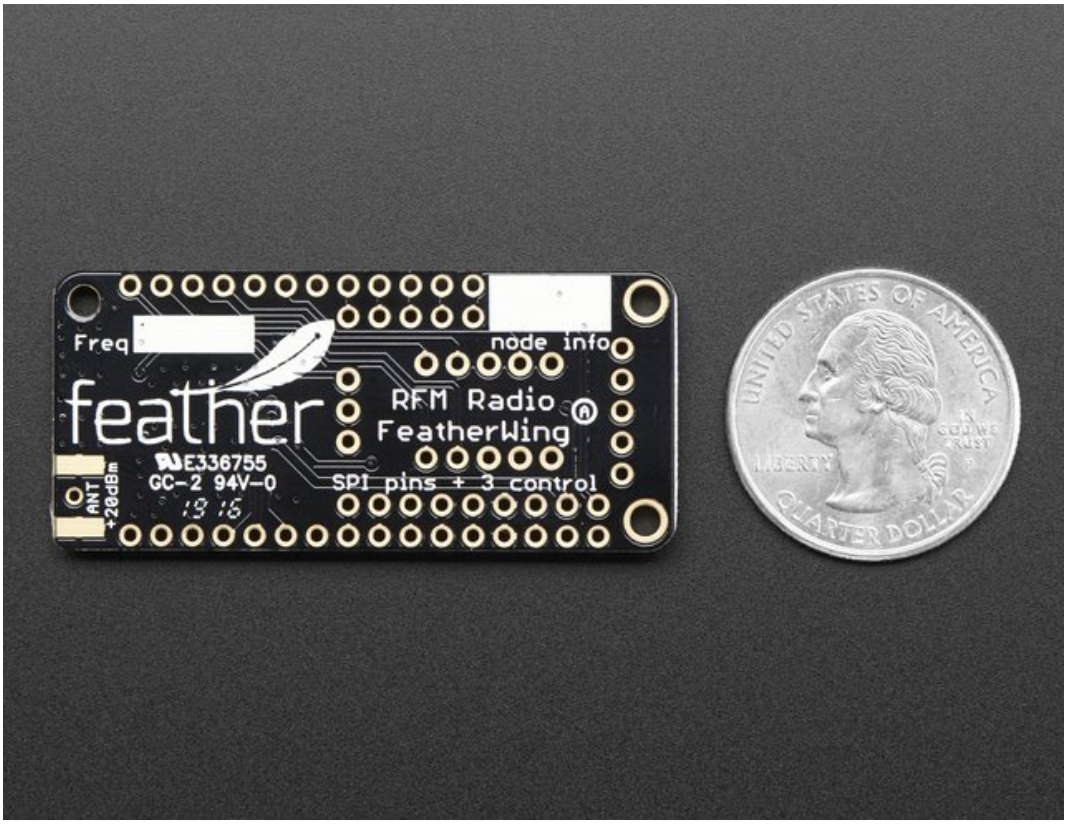
Add short-hop wireless to your Feather with these Radio Featherwings. These add-ons for any Feather board will let you integrate packetized radio (with the RFM69 radio) or LoRa radio (with the RFM9x's). These radios are good options for kilometer-range radio, and paired with one of our WiFi, cellular or Bluetooth Feathers, will let you bridge from 433/900 MHz to the Internet or your mobile device.



These radio modules come in four variants (two modulation types and two frequencies) The RFM69's are easiest to work with, and are well known and understood. The LoRa radios are exciting, longer-range and more powerful but also more expensive.

- **RFM69 @ 433 MHz** - basic packetized FSK/GFSK/MSK/GMSK/OOK radio at 433 MHz for use in Europe ITU 1 license-free ISM, or for amateur use with restrictions (check your local amateur regulations!)
- **RFM69 @ 900 MHz** - basic packetized FSK/GFSK/MSK/GMSK/OOK radio at 868 or 915 MHz for use in Americas ITU 2 license-free ISM, or for amateur use with restrictions (check your amateur regulations!)
- **RFM98 @ 433 MHz** - LoRa capable radio at 433 MHz for use in Europe ITU 1 license-free ISM, or for amateur use with restrictions (check your local amateur regulations!)
- **RFM95 @ 900 MHz** - LoRa capable radio at 868 or 915 MHz for use in Americas ITU 2 license-free ISM, or for amateur use with restrictions (check your local amateur regulations!)

The radio modules themselves have the same pinout so the PCB is the same, but the library usage and wiring may vary. All use SPI for interfacing, and there are great Arduino libraries available for both.

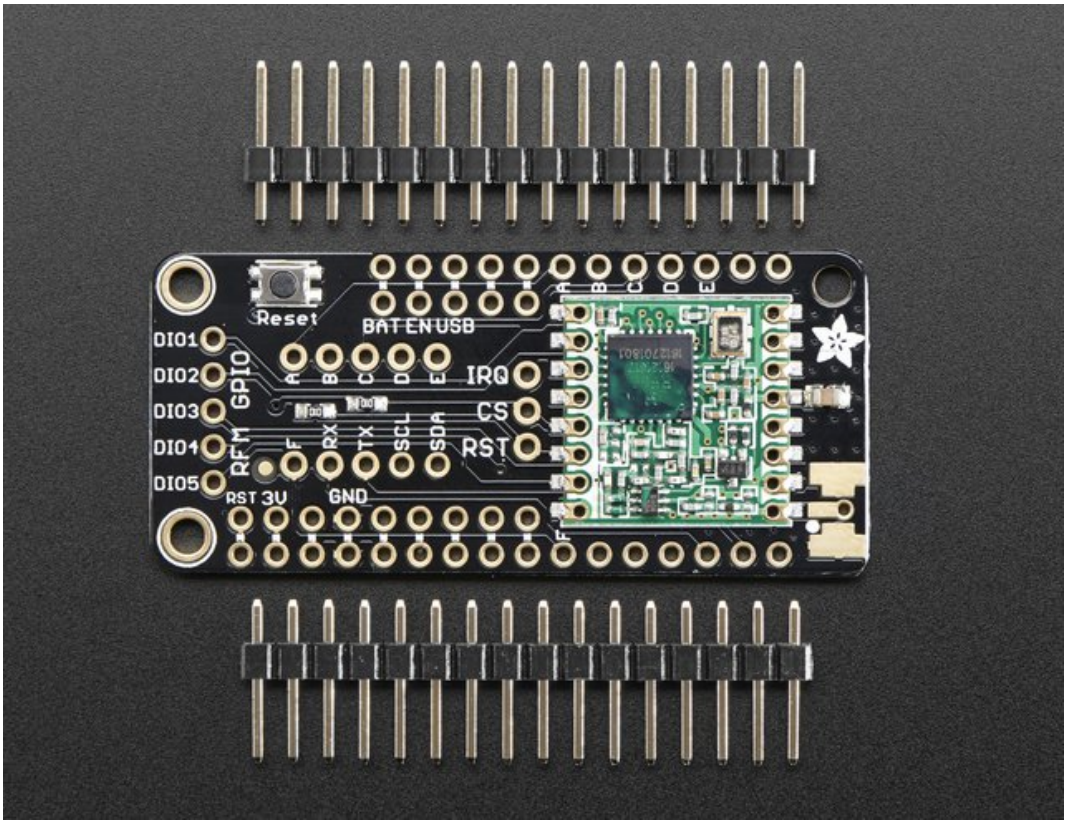


RFM69 Specs

- SX1231 based module with SPI interface
- Packet radio with ready-to-go Arduino libraries
- Uses the license-free ISM bands
- +13 to +20 dBm up to 100 mW Power Output Capability (power output selectable in software)
- 50mA (+13 dBm) to 150mA (+20dBm) current draw for transmissions
- Range of approx. 350 meters, depending on obstructions, frequency, antenna and power output
- Create multipoint networks with individual node addresses
- Encrypted packet engine with AES-128

RFM9x Specs

- SX127x LoRa® based module with SPI interface
- Packet radio with ready-to-go Arduino libraries
- Uses the license-free ISM bands
- +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.
- 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://adafru.it/mGa)).

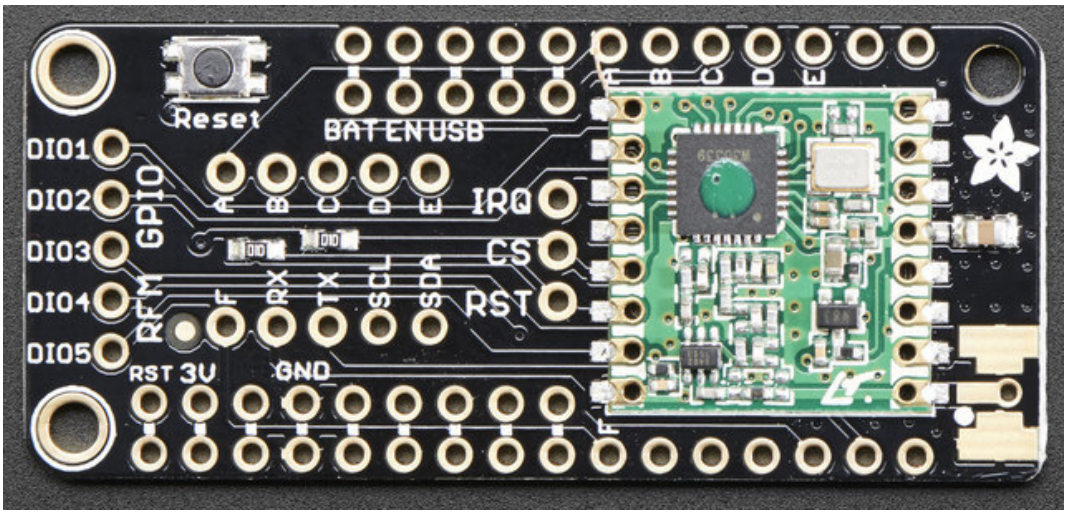


Currently tested to work with the **Feather ESP8266**, **Feather 32u4**, **Feather M0, WICED Feather** (RFM69 library only) and **Teensy 3 Feather** series, some wiring is required to configure the FeatherWing for the chipset you plan to use.

All radios are sold individually and can only talk to radios of the same part number. E.g. RFM69 900 MHz can only talk to RFM69 900 MHz, LoRa 433 MHz can only talk to LoRa 433, etc.

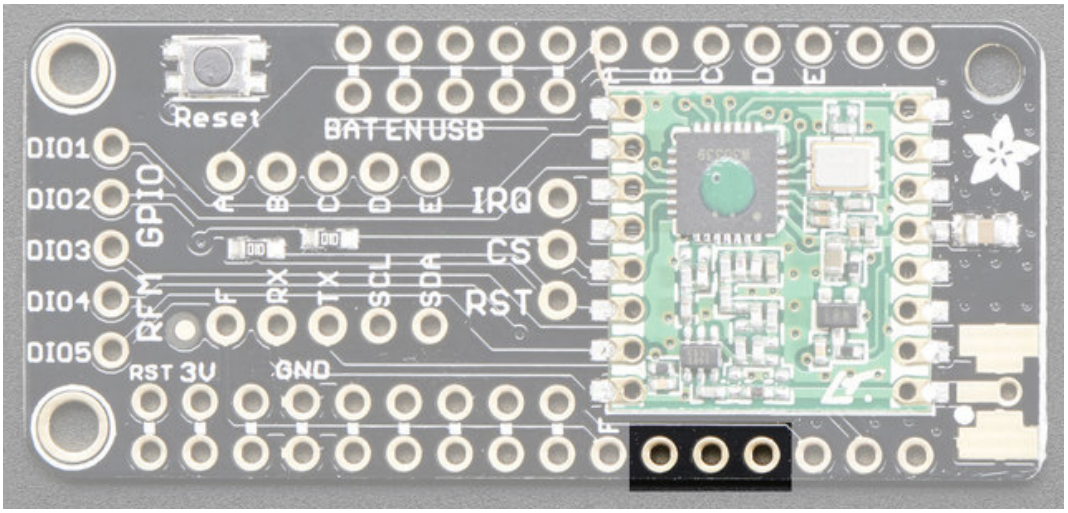
Each radio 'Wing comes with some header. Some soldering is required to attach the header. You will need to cut and solder on a small piece of wire (any solid or stranded core is fine) in order to create your antenna. Optionally you can pick up a uFL or SMA edge-mount connector and attach an external duck.

Pinouts



SPI Data Pins (Fixed)

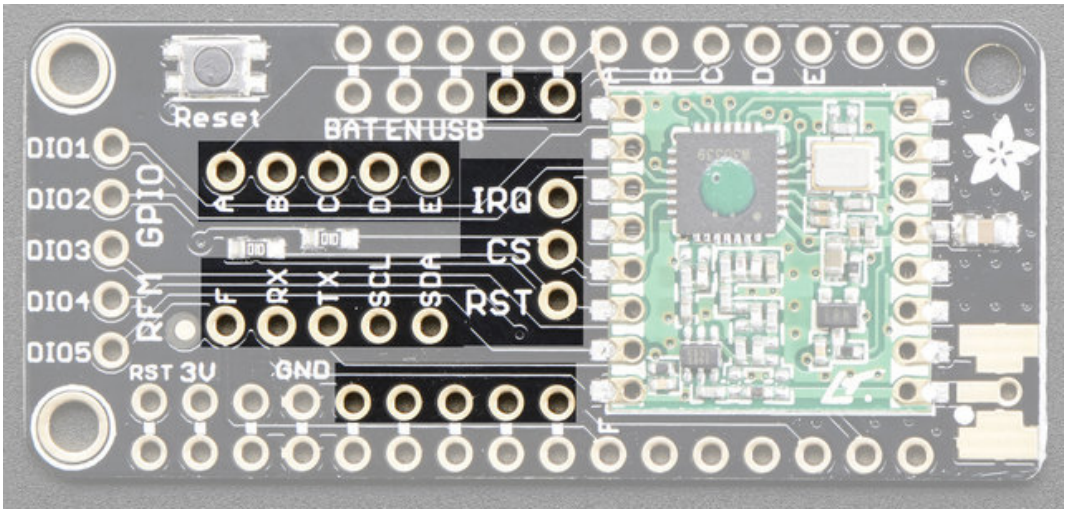
The three SPI data pins (MOSI/MISO/SCK) are hardwired to these three pads, which are use for the default SPI interface on all Feathers:



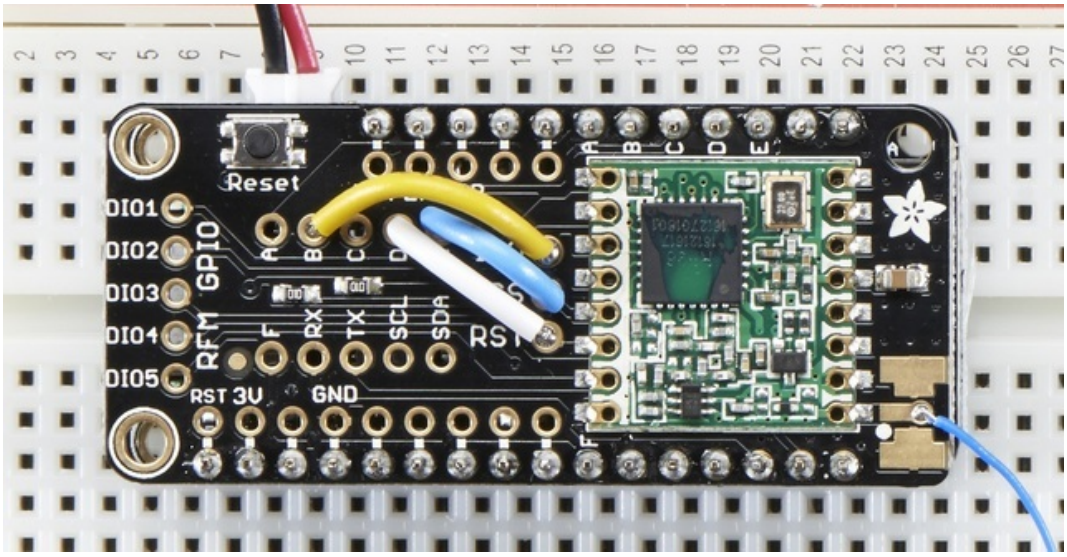
SPI Control Pins (Flexible)

You also need three more pins to control the radio: **CS, RST** and **IRQ**

Since there is no guaranteed Feather pin that is interrupt-capable, we set it up so you can fly-wire these three to any three pins available. For the non-Serial/IC pins on the right, we name them **A** thru **F**. We also indicate the **RX/TX/SDA/SCL** pins if you need to use those:

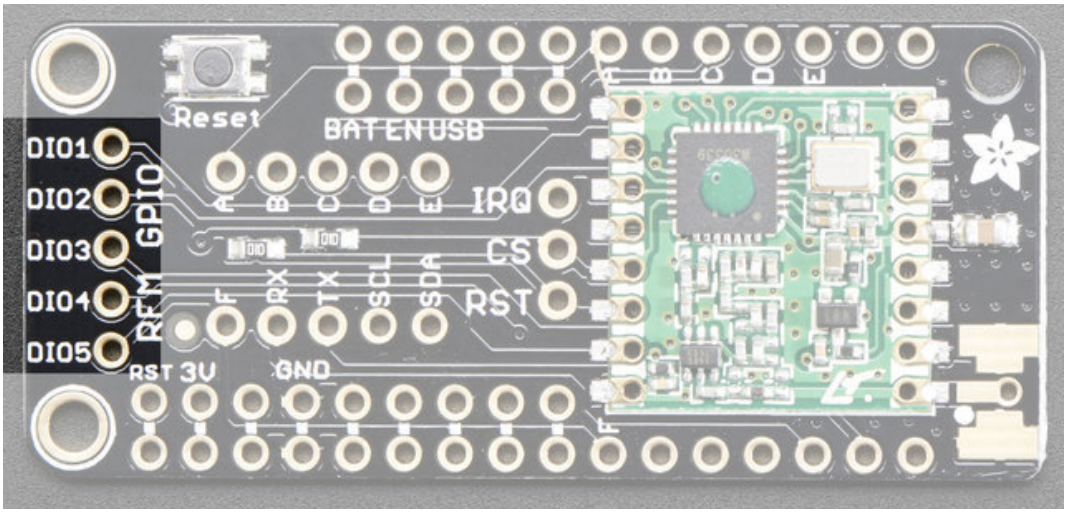


Wire them with three short jumpers like so:



RFM GPIO

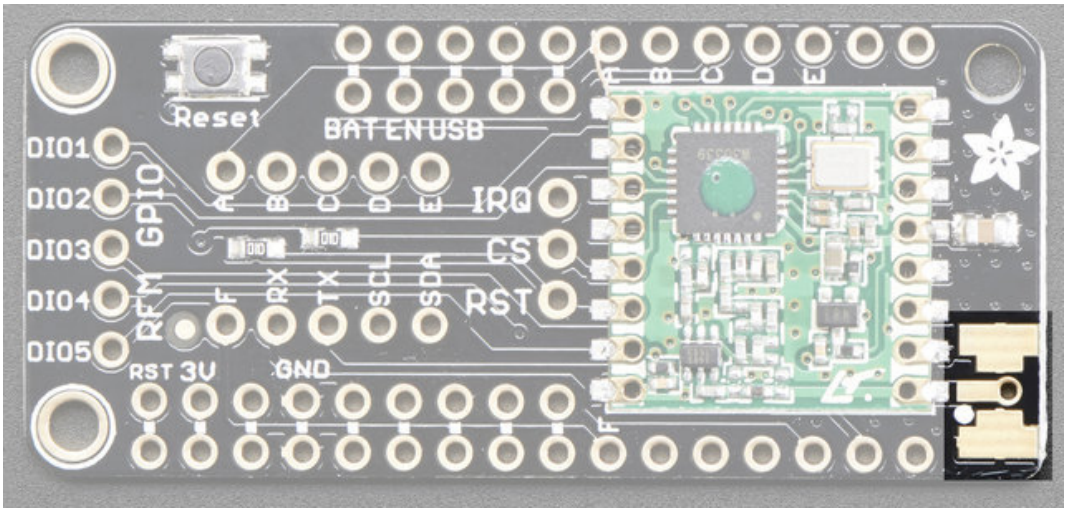
There's some other GPIO pins that you may want to use - they can be configured to give you notice of things like packet completion or incoming data. They're all on the left. **DIO0** is also known as **IRQ** so we don't have that duplicated on the left breakouts



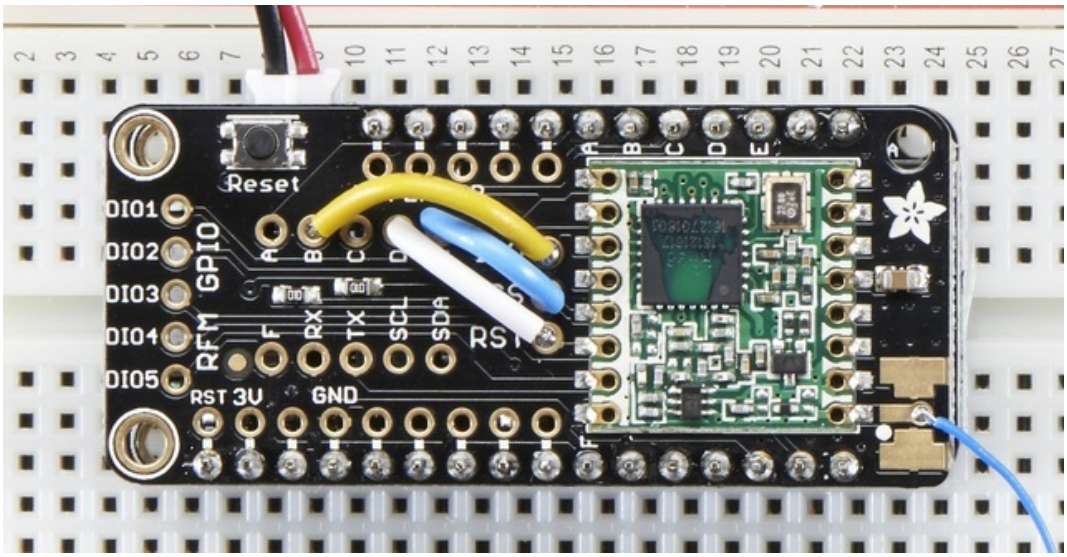
Antenna

For an antenna, you have **three** options:

- Plain wire antenna (cut a quarter-wavelength piece and solder it into the pad/hole
- uFL connector (not included) (http://adafru.it/1661), which can be soldered and then used to attach a uFL antenna or adapter
- SMA edge-launch (not included) (http://adafru.it/1865), for use with any SMA connector



Wiring



Because each Feather uses a differet processor, there is some light wiring that needs to be done to configure the radio pins. In particular, an interrupt-capable pin is required for **IRQ** but there is no one irq pin that is the same on all the Feathers!

So, while MOSI/MISO/SCK are fixed, you will want to solder three short wires for **CS, RST** and **IRQ**

Here is our tested/suggested wiring configurations and code snippets for defining the pins

ESP8266 Wiring

The ESP does not have a lot of spare pins, and the SPI pins are taken, so here's what we've tested that works:

```
#define RFM95_CS  2    // "E"
#define RFM95_RST 16   // "D"
#define RFM95_INT 15   // "B"

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

This leaves the I2C default pins (4 and 5) available

Feather 32u4

The 32u4 doesn't have a lot of IRQs and the only ones available are on pins 0, 1, 2, 3 which are also the Serial RX/TX and I2C pins. So it's not great because you have to give up one of those pins.

```
#define RFM95_CS  10   // "B"
#define RFM95_RST 11   // "A"
#define RFM95_INT 2    // "SDA" (only SDA/SCL/RX/TX have IRQ!)
```

```
#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

The Feather M0 is really easy to use, a ton of interrupts so wiring is easy

```
#define RFM95_CS  10   // "B"
#define RFM95_RST 11   // "A"
#define RFM95_INT  6   // "D"
```

```
#define RFM69_CS      10   // "B"
#define RFM69_RST     11   // "A"
#define RFM69_IRQ     6    // "D"
#define RFM69_IRQN    digitalPinToInterrupt(RFM69_IRQ )
```

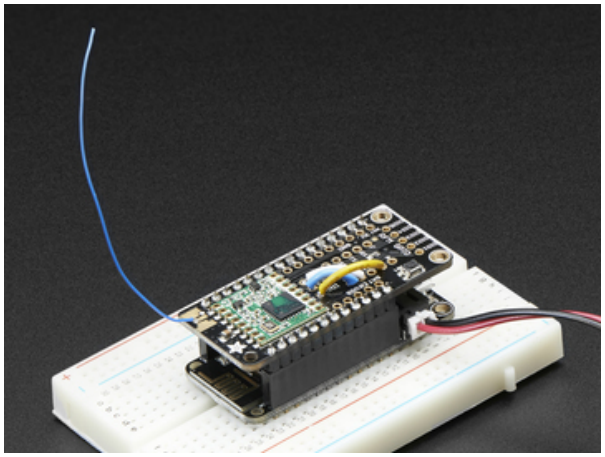
Assembly

Antenna Options

These radio Wings do not have a built-in antenna. Instead, you have three options for attaching an antenna. For most low cost radio nodes, a wire works great. If you need to put the radio into an enclosure, soldering in uFL and using a uFL to SMA adapter will let you attach an external antenna. You can also solder an SMA edge-mount connector directly

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.

uFL Connector

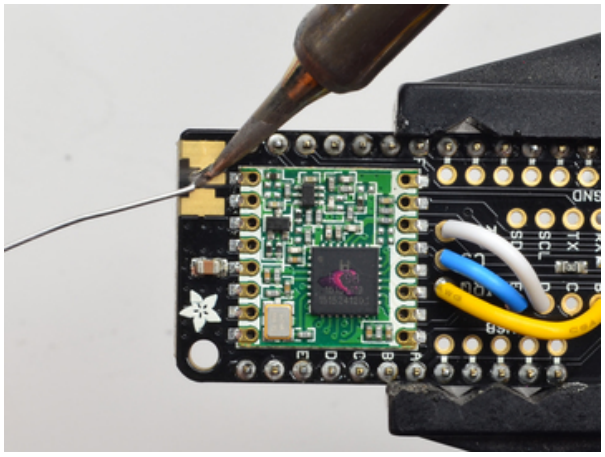
If you want an external antenna that is a few inches away from the radio, you need to do a tiny bit more work but its not too difficult.

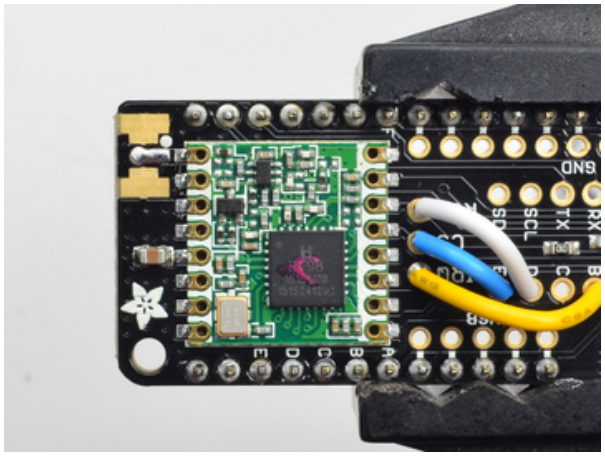
You'll need to get an SMT uFL connector, these are fairly standard (http://adafru.it/1661)

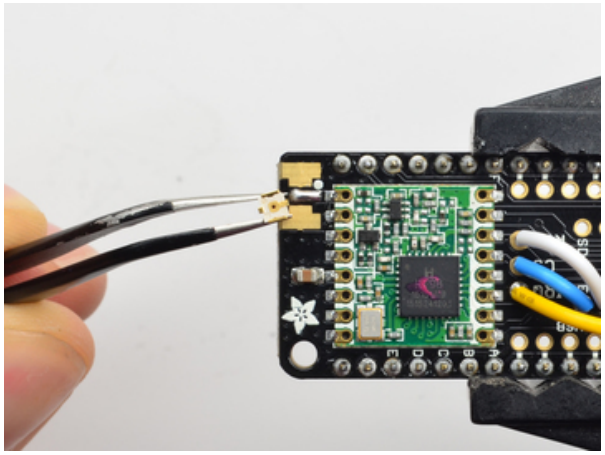
You'll also need a uFL to SMA adapter (http://adafru.it/851) (or whatever adapter you need for the antenna you'll be using, SMA is the most common

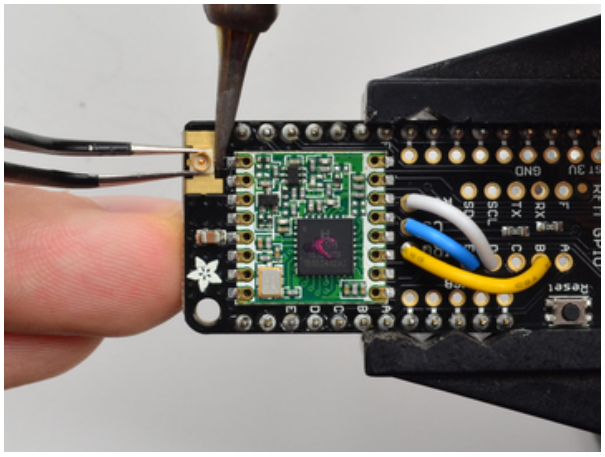
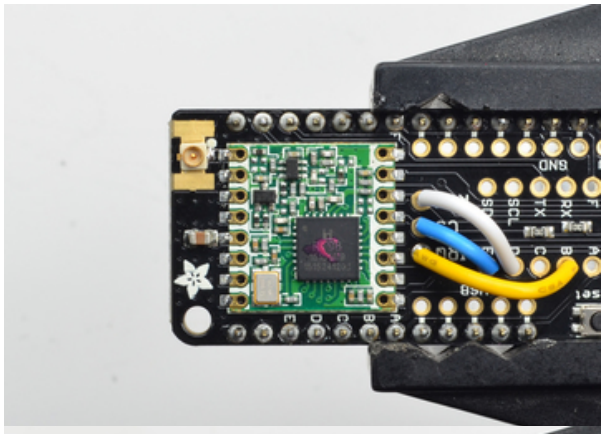
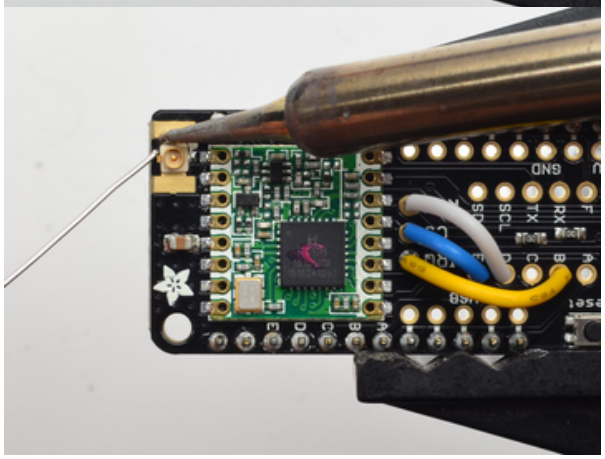
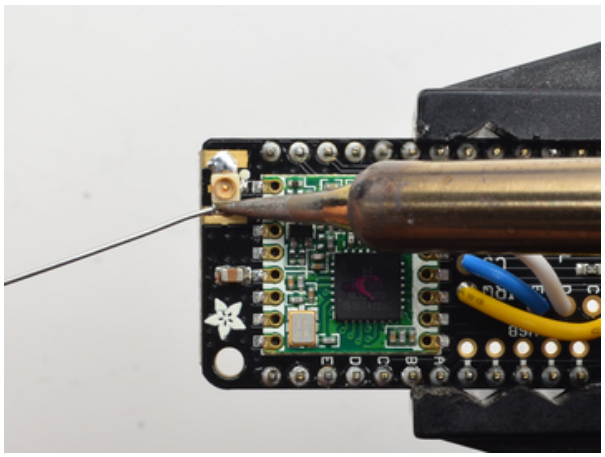
Of course, you will also need an antenna of some sort, that matches your radio frequency

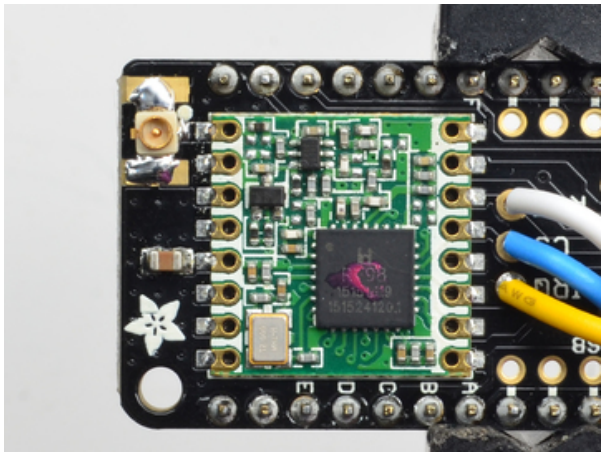
uFL connectors are rated for 30 connection cycles, but be careful when connecting/disconnecting to not rip the pads off the PCB. Once a uFL/SMA adapter is connected, use strain relief!

- 

Start by melting solder onto the center signal pad
- 

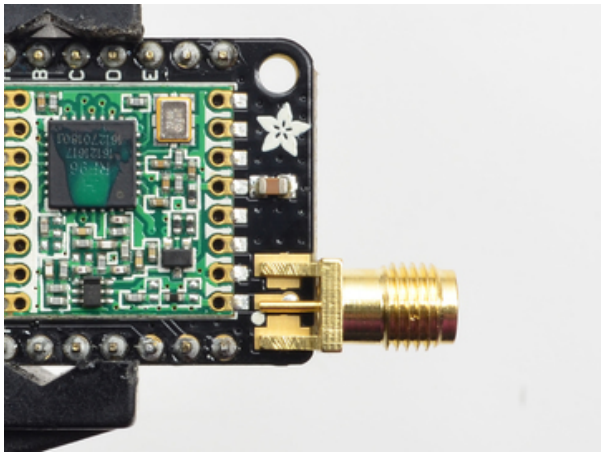
- 
Check the bottom of the uFL connector, note that there's two large side pads (ground) and a little inlet pad. The other small pad is not used!

- 
  Solder in the first pad while holding the uFL steady
- 
- 
  Solder in the two side pads, they are used for signal and mechanical connectivity so make sure there's plenty of solder
- 

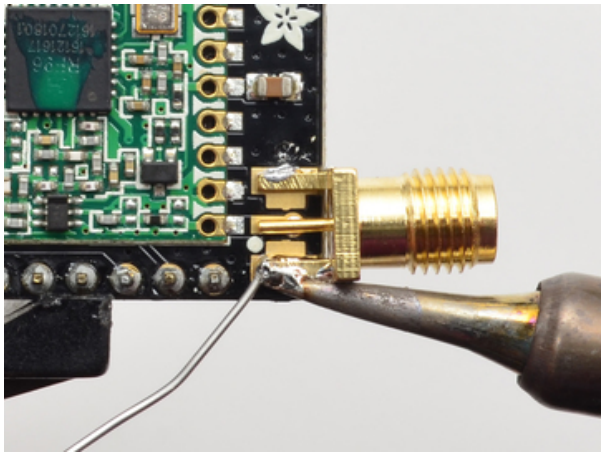
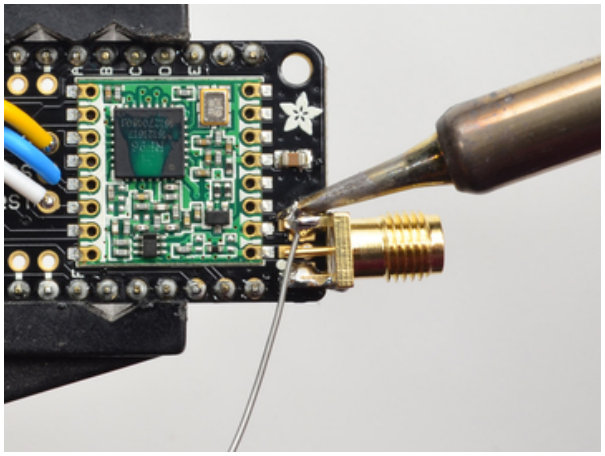
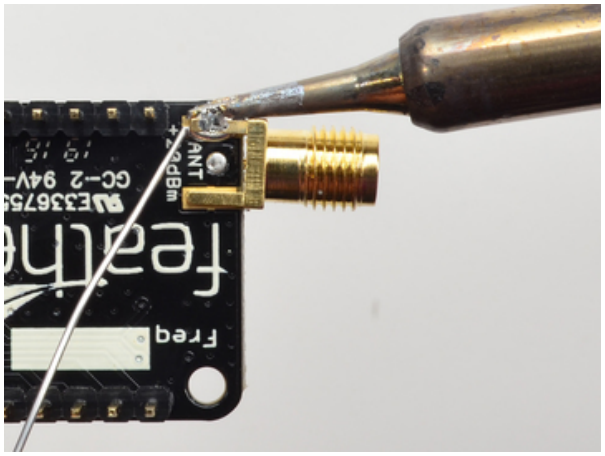
- 
Once done, check your work visually

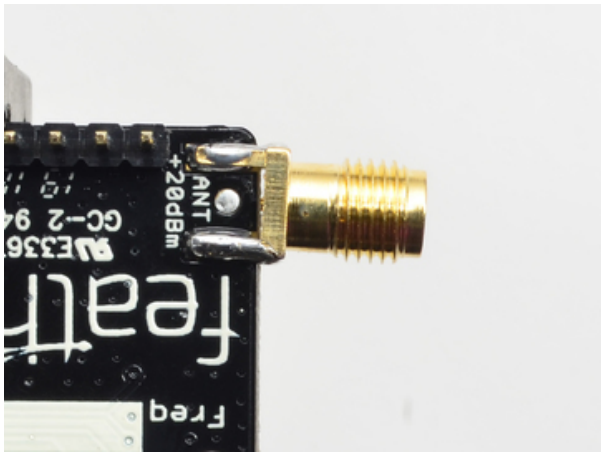
SMA Edge-Mount Connector

These strong edge connectors are used for many 'duck' antennas, and can also be panel mounted

- 
You'll need an SMA (or, if you need RP-SMA for some reason) Edge-Mount connector with 1.6mm spacing

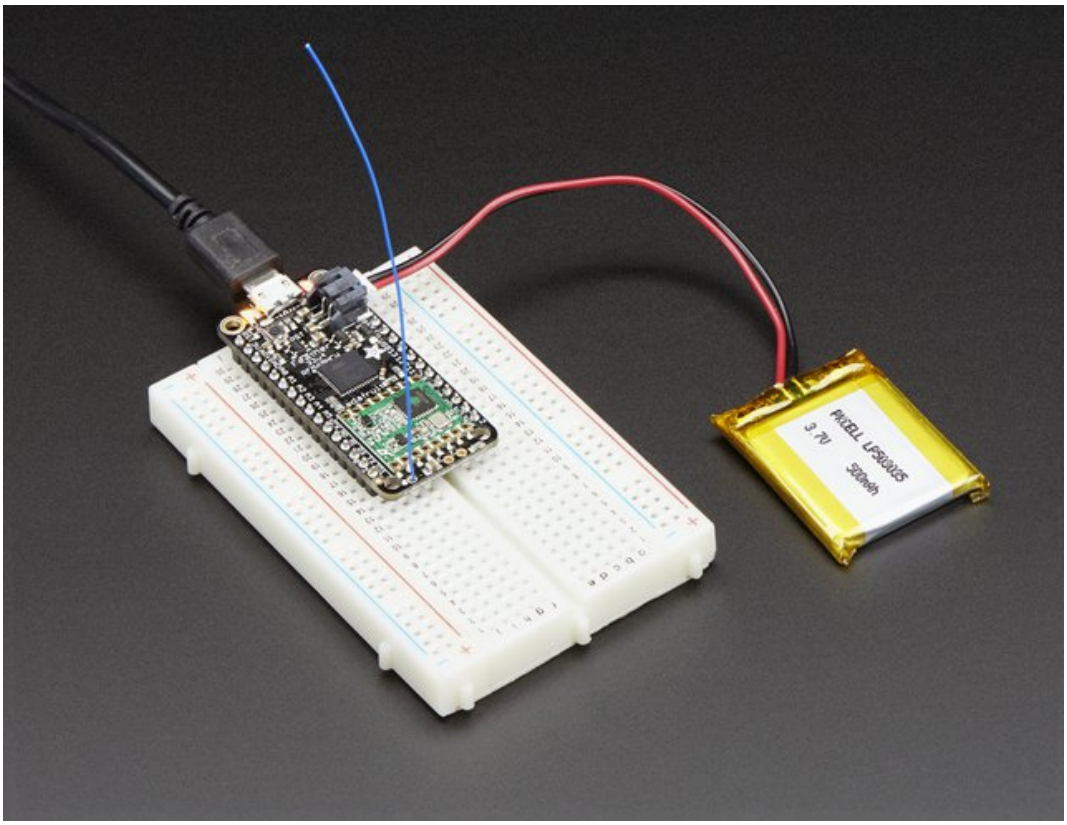
The SMA connector 'slides on' the top of the PCB

-  Solder all 5 connections (4 ground/mechanical and 1 signal)
- 
- 

- 

Use plenty of solder to make sure you have a good strong mechanical connection. The duck antennas are long and make great levers, so they could pry apart the solder joints if not soldered well

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

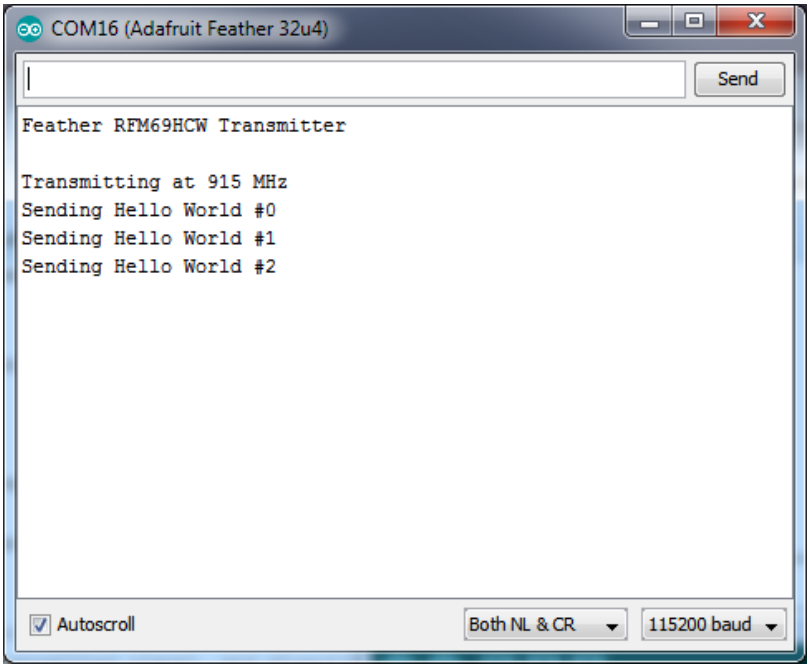
```
/* RFM69 library and code by Felix Rusu - felix@lowpowerlab.com
// Get libraries at: https://github.com/LowPowerLab/
// Make sure you adjust the settings in the configuration section below !!!
// **********************************************************************************
// Copyright Felix 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



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!

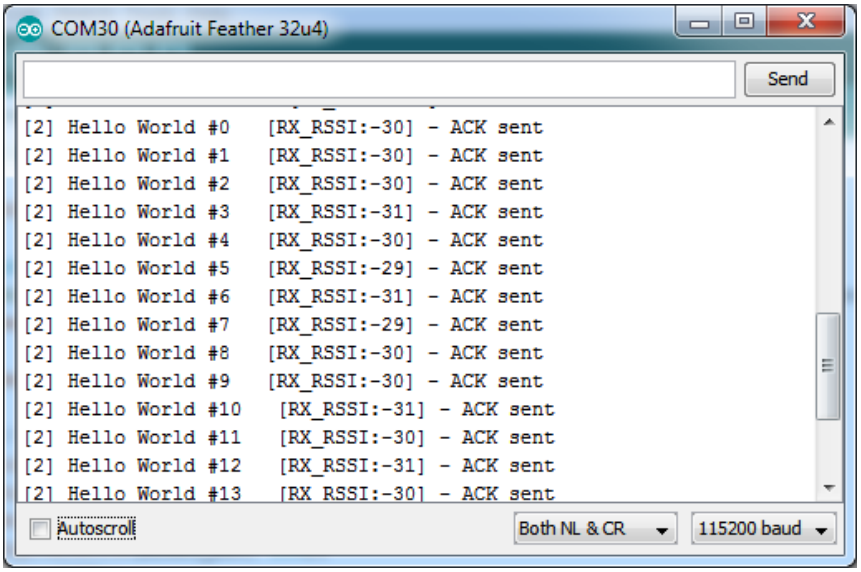
Before uploading, check for the #define FREQUENCY RF69_915MHZ line and comment that out (and uncomment the line above) to match the frequency of the hardware you're using
Uncomment/comment the sections defining the pins for Feather 32u4, Feather M0, etc depending on which chipset and wiring you are using! 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
```

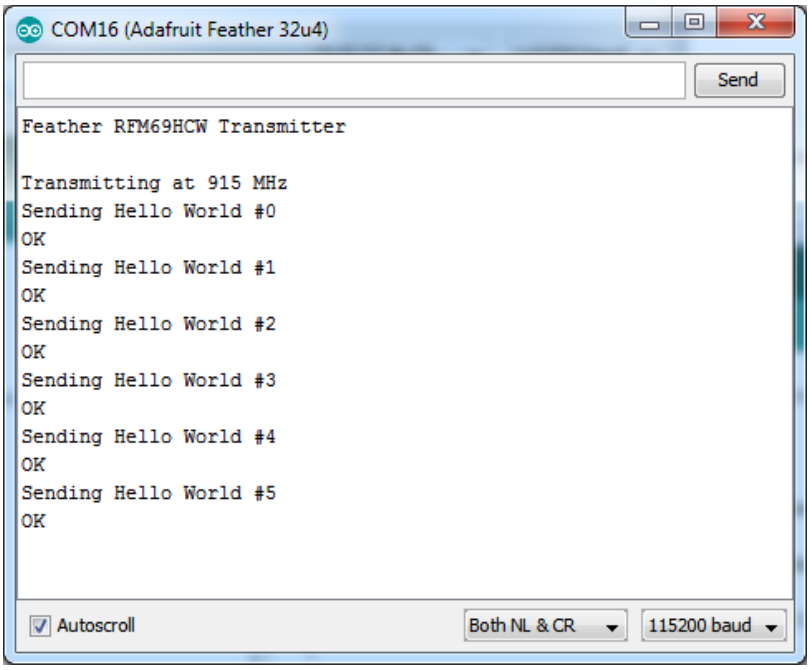
```
/* for Feather M0
#define RFM69_CS      8
#define RFM69_IRQ     3
#define RFM69_IRQN    3  // Pin 3 is IRQ 3!
#define RFM69_RST     4
*/

/* 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



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



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

Radio Net & ID Configuration

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

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

```
#define NODEID        2    // The unique identifier of this node
```

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

Radio Type Config

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

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

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

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

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

Feather Radio Pinout

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

```
/* 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
```

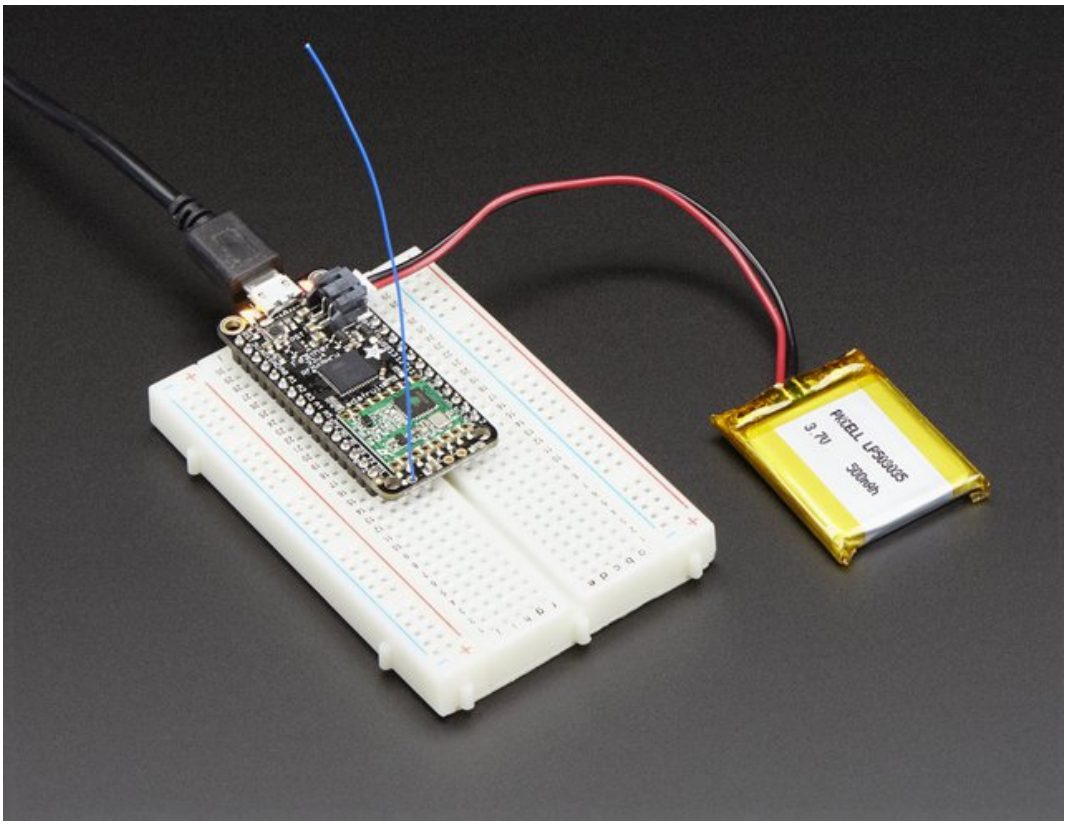
```
/* for Feather M0
#define RFM69_CS      8
#define RFM69_IRQ     3
#define RFM69_IRQN    3  // Pin 3 is IRQ 3!
#define RFM69_RST     4
*/
/* ESP8266 feather w/wing
#define RFM69_CS      2
#define RFM69_IRQ     15
#define RFM69_IRQN    digitalPinToInterrupt(RFM69_IRQ )
#define RFM69_RST     16
*/

/* 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!

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 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 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://adafru.it/mCA) which also suppors 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 theRadioHead library (http://adafru.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 canvisit the RadioHead documentation page to get the most recent version which may have bug-fixes or more functionality (http://adafru.it/mCA)

RadioHead-1.62.zip
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 **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!

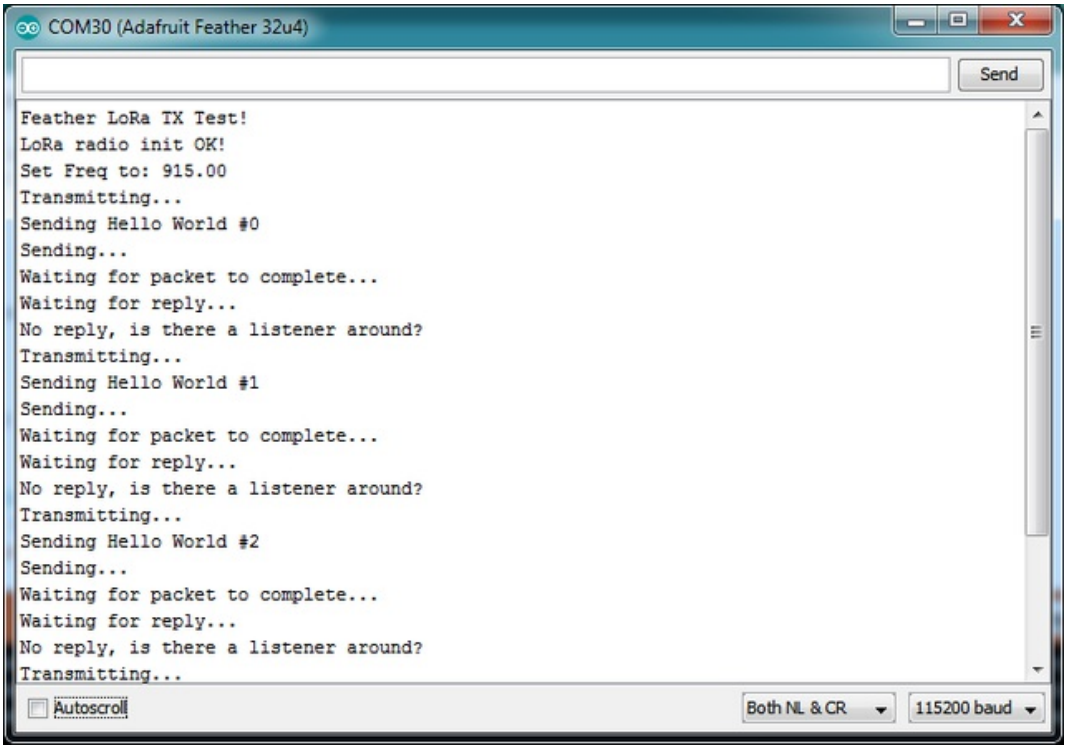
Before uploading, check for the #define RF95_FREQ 915.0 line and change that to 433.0 if you are using the 433MHz version of the LoRa radio!
Uncomment/comment the sections defining the pins for Feather 32u4, Feather M0, etc depending on which chipset and wiring you are using! The pins used will vary 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!

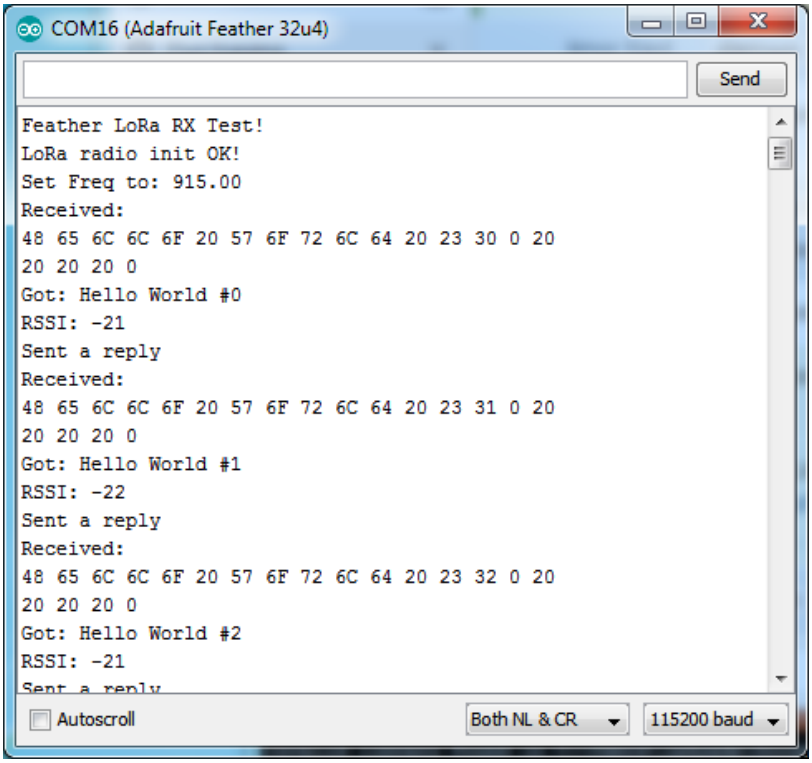
Make sure the #define RF95_FREQ 915.0 matches your transmitter Feather!
Uncomment/comment the sections defining the pins for Feather 32u4, Feather M0, etc depending on which chipset and wiring you are using! The pins used will vary depending on your setup!

```
// 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!");
```

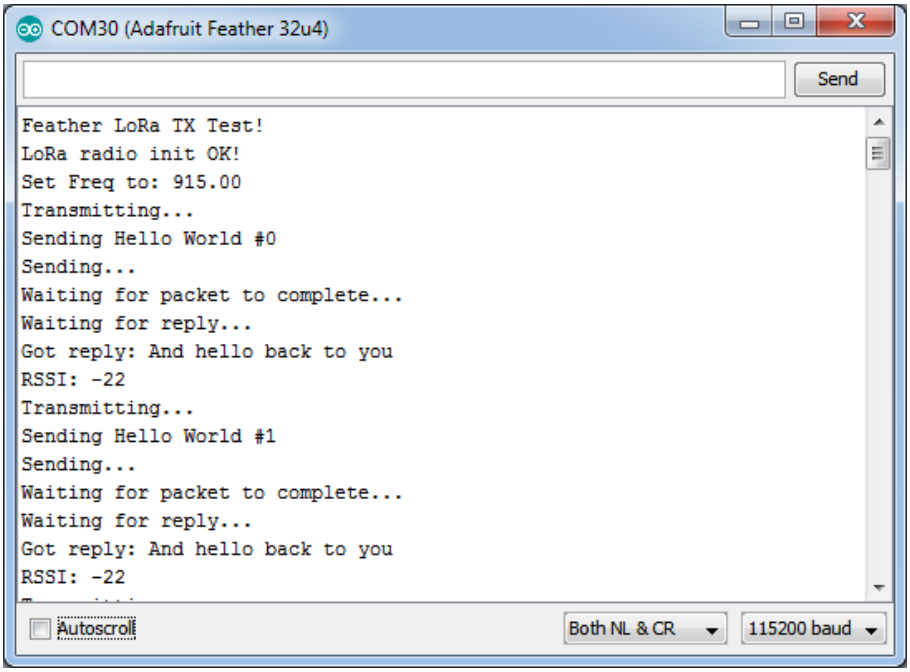
```
  // Defaults after init are 434.0MHz, modulation GFSK_Rb250Fd250, +13dbM
  if (!rf95.setFrequency(RF95_FREQ)) {
    Serial.println("setFrequency failed");
    while (1);
  }
  Serial.print("Set Freq to: "); Serial.println(RF95_FREQ);

  // 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



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 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 transmisssion `And hello back to you` because it got a reply from the receiver



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

RFM9x

- 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
- FCC Test Report (http://adafru.it/q6A)
- ETSI Test Report (http://adafru.it/r6a)
- RoHS Report (http://adafru.it/r6b)

RFM69

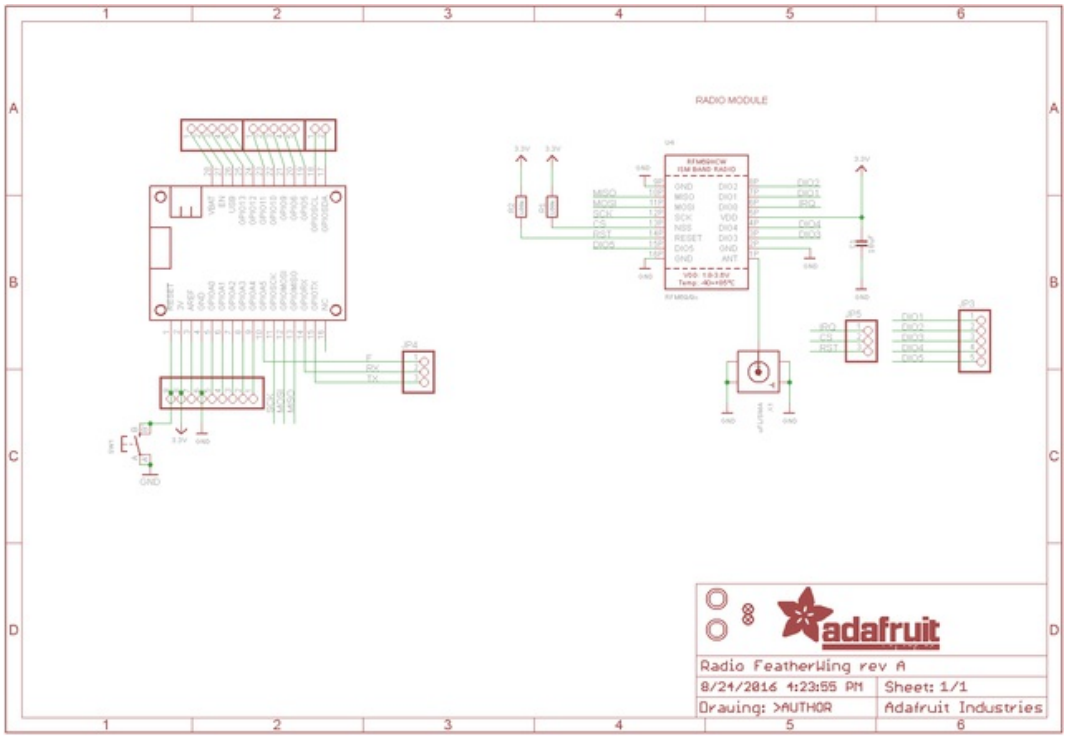
- SX1231 Datasheet (http://adafru.it/mCv) - The RFM69 radio chip itself
- RFM69HCW datasheet (http://adafru.it/mCu)- contains the SX1231 datasheet plus details about the module (http://adafru.it/mFX)
- RoHS Test Report (http://adafru.it/oC1)
- RoHS Test Report (http://adafru.it/oC2)
- REACH Test Report (http://adafru.it/oC3)
- ETSI Test Report (http://adafru.it/r6c)
- FCC Test Report (http://adafru.it/r6d)

EagleCAD PCB files on GitHub (http://adafru.it/r6e)

Fritzing object in Adafruit Fritzing library (http://adafru.it/aP3)

Schematic

(Pinouts are the same for all four radio versions)



Fabrication Print

Dimensions in inches