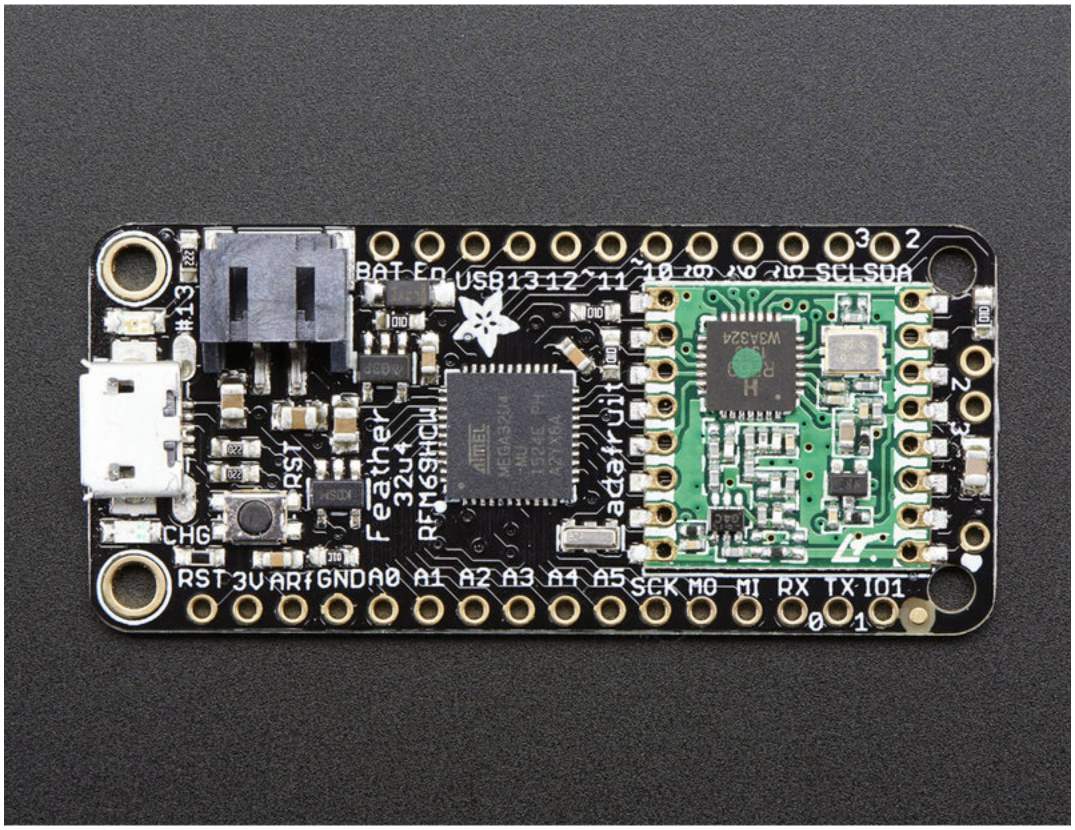


Adafruit Feather 32u4 Radio with RFM69HCW Module

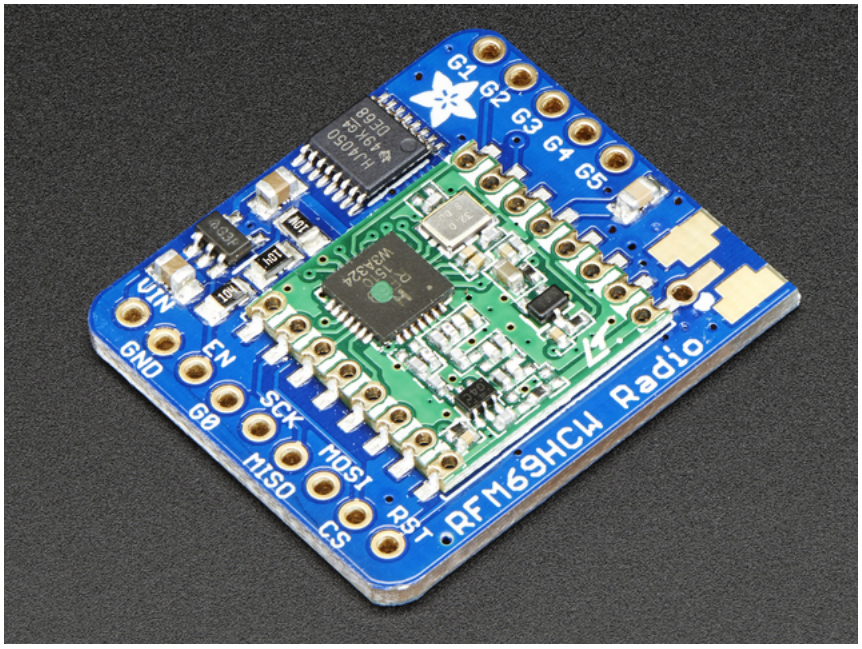
Send your message far and wide

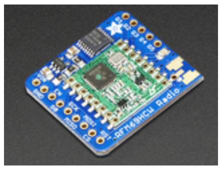
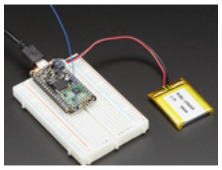
Contributors

lady ada
Feedback? Corrections?
FEATHER / FEATHER BOARDS

Using the RFM69 Radio

by lady ada

- 

- 
- 

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

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

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

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

"Raw" vs Packetized

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

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

Arduino Libraries

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

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

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

RadioHead Library example

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

Download RadioHead Library

Rename the uncompressed folder **RadioHead** and check that the **RadioHead** folder contains files like **RH_RFM69.cpp** and **RH_RFM69.h** (and many others!)

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

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

Basic RX & TX example

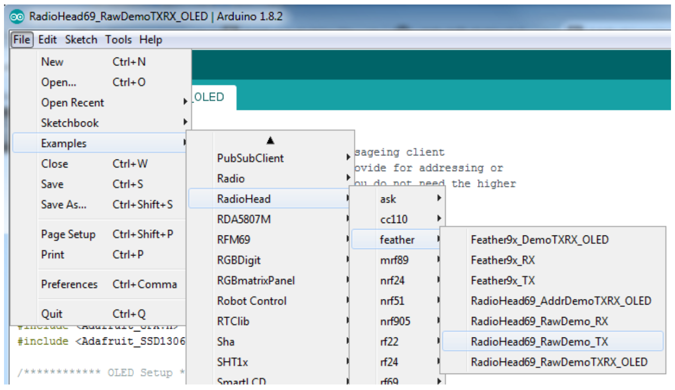
Lets get a basic demo going, where one radio transmits and the other receives. We'll start by setting up the transmitter

Basic Transmitter example code

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

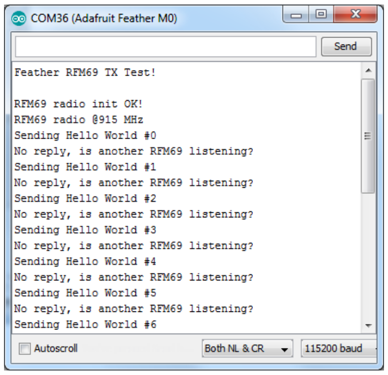
Open up the example **RadioHead -> feather -> RadioHead69_RawDemo_TX**

Load this code into your Transmitter Arduino or Feather!



Before uploading, check for the #define FREQUENCY RF69_915MHZ line and comment that out (and uncomment the line above) to match the frequency of the hardware you're using
These examples are optimized for the Feather 32u4/M0. If you're using differnet wiring, uncomment/comment/edit the sections defining the pins depending on which chipset and wiring you are using! The pins used will vary depending on your setup!

Once uploaded you should see the following on the serial console



```
Feather RFM69 TX Test!

RFM69 radio init OK!
RFM69 radio @915 MHz
Sending Hello World #0
No reply, is another RFM69 listening?
Sending Hello World #1
No reply, is another RFM69 listening?
Sending Hello World #2
No reply, is another RFM69 listening?
Sending Hello World #3
No reply, is another RFM69 listening?
Sending Hello World #4
No reply, is another RFM69 listening?
Sending Hello World #5
No reply, is another RFM69 listening?
Sending Hello World #6
```

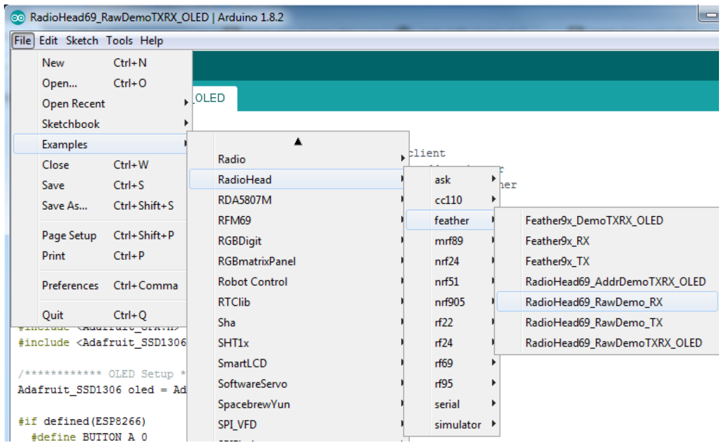
Now open up another instance of the Arduino IDE - this is so you can see the serial console output from the TX device while you set up the RX device.

Basic receiver example code

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

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

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



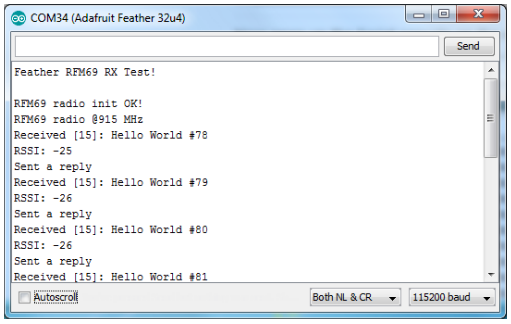
```
#include <Adafruit_SSD1306

/************ OLED Setup *
Adafruit_SSD1306 oled = Ad

#if defined(ESP8266)
  #define BUTTON_A 0
```

Before uploading, check for the #define FREQUENCY RF69_915MHZ line and comment that out (and uncomment the line above) to match the frequency of the hardware you're using
These examples are optimized for the Feather 32u4/M0. If you're using differnet wiring, uncomment/comment/edit the sections defining the pins depending on which chipset and wiring you are using! The pins used will vary depending on your setup!

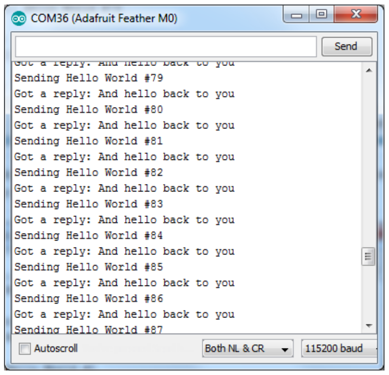
Now open up the Serial console on the receiver, while also checking in on the transmitter's serial console. You should see the receiver is...well, receiving packets



```
Feather RFM69 RX Test!

RFM69 radio init OK!
RFM69 radio @915 MHz
Received [15]: Hello World #78
RSSI: -25
Sent a reply
Received [15]: Hello World #79
RSSI: -26
Sent a reply
Received [15]: Hello World #80
RSSI: -26
Sent a reply
Received [15]: Hello World #81
```

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



```
Got a reply: And hello back to you
Sending Hello World #79
Got a reply: And hello back to you
Sending Hello World #80
Got a reply: And hello back to you
Sending Hello World #81
Got a reply: And hello back to you
Sending Hello World #82
Got a reply: And hello back to you
Sending Hello World #83
Got a reply: And hello back to you
Sending Hello World #84
Got a reply: And hello back to you
Sending Hello World #85
Got a reply: And hello back to you
Sending Hello World #86
Got a reply: And hello back to you
Sending Hello World #87
```

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

Radio Freq. Config

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

Download file
Copy Code

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

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

Configuring Radio Pinout

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

Also, an LED is defined.

For example, here is the Feather 32u4 pinout

Download file
Copy Code

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

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

Download file
Copy Code

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

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

Download file
Copy Code

```
#if defined (__AVR_ATmega328P__)  // UNO or Feather 328P w/wing
  #define RFM69_INT     3  //
  #define RFM69_CS      4  //
  #define RFM69_RST     2  // "A"
  #define LED           13
#endif
```

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

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

Download file
Copy Code

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

Setup

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

Download file
Copy Code

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

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

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

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

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

Initializing Radio

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

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

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

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

Download file
Copy Code

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

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

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

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

Basic Transmission Code

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

Download file
Copy Code

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

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

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

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

  if (rf69.waitAvailableTimeout(500))  {
    // Should be a reply message for us now
    if (rf69.recv(buf, &len)) {
      Serial.print("Got a reply: ");
      Serial.println((char*)buf);
      Blink(LED, 50, 3); //blink LED 3 times, 50ms between blinks
    } else {
      Serial.println("Receive failed");
    }
  } else {
    Serial.println("No reply, is another RFM69 listening?");
  }
}
```

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

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

Basic Receiver Code

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

Download file
Copy Code

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

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

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

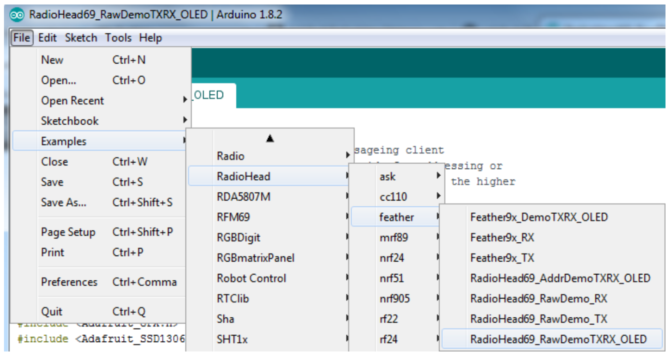
It also prints out the RSSI which is the receiver signal strength indicator. This number will range from about -15 to -80. The larger the number (-15 being the highest you'll likely see) the stronger the signal.

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

Once done it will continue waiting for a new packet

Basic Receiver/Transmitter Demo w/OLED

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



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

Addressed RX and TX Demo

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

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

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

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

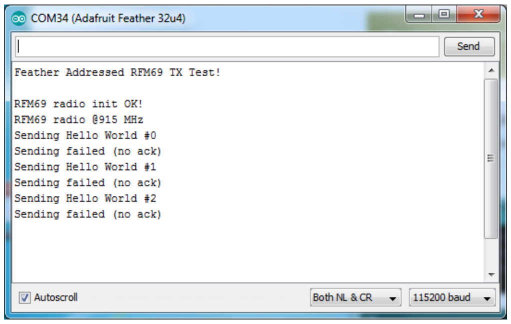
Each client can have its own address set, as well as the server address. See this code at the beginning:

Download file
Copy Code

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

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

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

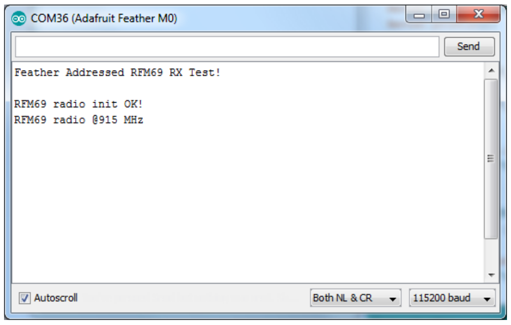


```
Feather Addressed RFM69 TX Test!

RFM69 radio init OK!
RFM69 radio @915 MHz
Sending Hello World #0
Sending failed (no ack)
Sending Hello World #1
Sending failed (no ack)
Sending Hello World #2
Sending failed (no ack)
```

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

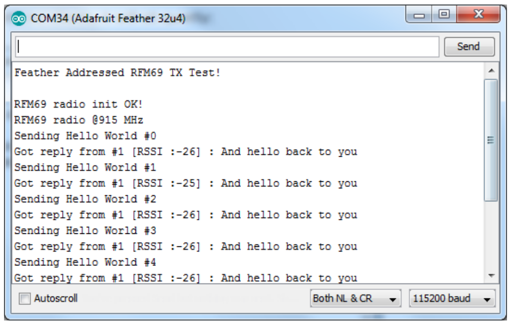
If you have the server running, with no clients, it will sit quietly:



```
Feather Addressed RFM69 RX Test!

RFM69 radio init OK!
RFM69 radio @915 MHz
```

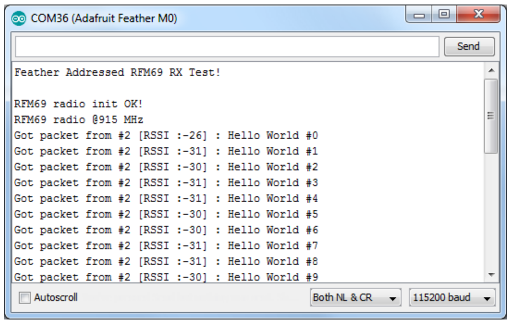
Turn on the client and you'll see acknowledged packets!



```
Feather Addressed RFM69 TX Test!

RFM69 radio init OK!
RFM69 radio @915 MHz
Sending Hello World #0
Got reply from #1 [RSSI :-26] : And hello back to you
Sending Hello World #1
Got reply from #1 [RSSI :-25] : And hello back to you
Sending Hello World #2
Got reply from #1 [RSSI :-26] : And hello back to you
Sending Hello World #3
Got reply from #1 [RSSI :-26] : And hello back to you
Sending Hello World #4
Got reply from #1 [RSSI :-26] : And hello back to you
```

And the server is also pretty happy



```
Feather Addressed RFM69 RX Test!

RFM69 radio init OK!
RFM69 radio @915 MHz
Got packet from #2 [RSSI :-26] : Hello World #0
Got packet from #2 [RSSI :-31] : Hello World #1
Got packet from #2 [RSSI :-30] : Hello World #2
Got packet from #2 [RSSI :-31] : Hello World #3
Got packet from #2 [RSSI :-31] : Hello World #4
Got packet from #2 [RSSI :-30] : Hello World #5
Got packet from #2 [RSSI :-30] : Hello World #6
Got packet from #2 [RSSI :-31] : Hello World #7
Got packet from #2 [RSSI :-31] : Hello World #8
Got packet from #2 [RSSI :-30] : Hello World #9
```

The secret sauce is the addition of this new object:

Download file
Copy Code

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

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

Download file
Copy Code

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

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

Download file
Copy Code

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

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

Download file
Copy Code

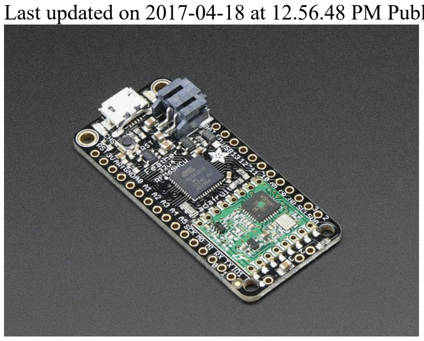
```
  // Wait for a message addressed to us from the client
  uint8_t len = sizeof(buf);
  uint8_t from;
  if (rf69_manager.recvfromAck(buf, &len, &from)) {
```

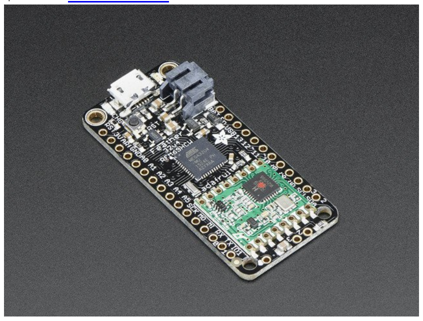
That function will wait forever. If you'd like to timeout while waiting for a packet, use **recvfromAckTimeout** which will wait an indicated # of milliseconds

Download file
Copy Code

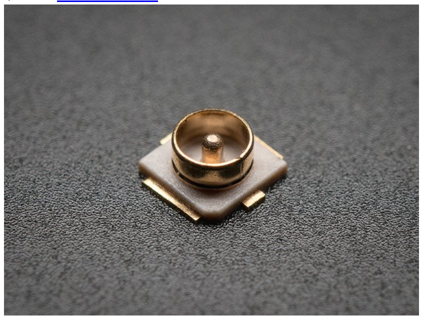
```
if (rf69_manager.recvfromAckTimeout(buf, &len, 2000, &from))
```

FEATHER HELP! RADIO RANGE F.A.Q.
Last updated on 2017-04-18 at 12.56.48 PM Published on 2016-04-06 at 05.09.04 PM

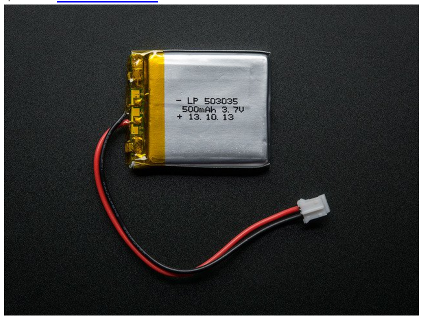

Adafruit Feather 32u4 RFM69HCW Packet Radio - 868 or 915 MHz
$24.95 Add to Cart


Adafruit Feather 32u4 with RFM69HCW Packet Radio - 433MHz
$24.95 Add to Cart

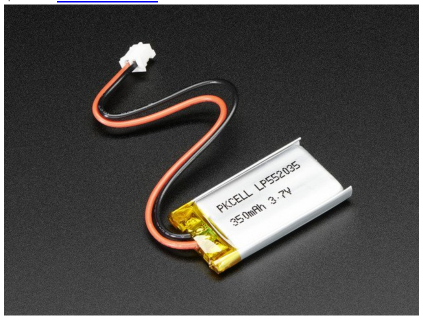
SMA to uFL/u.FL/IPX/IPEX RF Adapter Cable
$3.95 Add to Cart


uFL SMT Antenna Connector

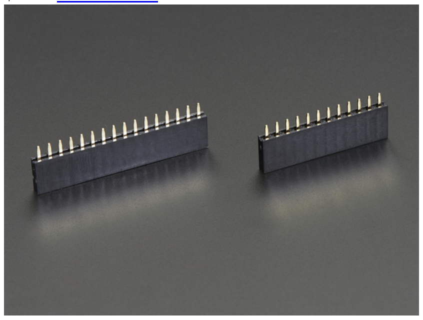
$0.75 Add to Cart



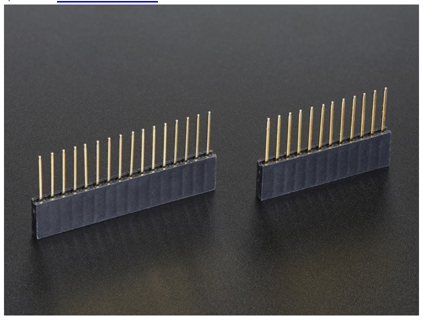
Lithium Ion Polymer Battery - 3.7v 500mAh
$7.95 Add to Cart



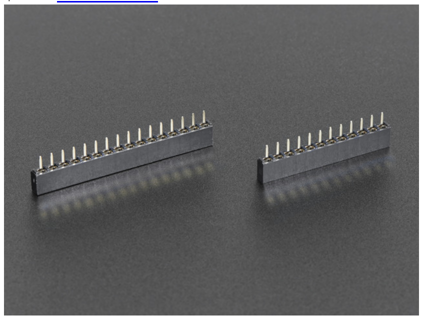
Lithium Ion Polymer Battery - 3.7v 350mAh
$6.95 Add to Cart



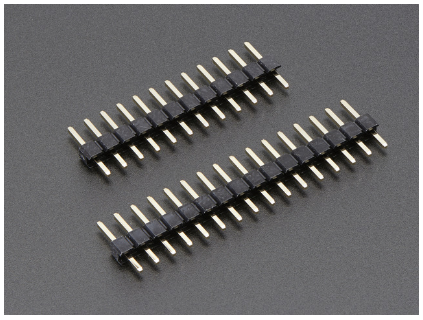
Feather Header Kit - 12-pin and 16-pin Female Header Set
$0.95 Add to Cart



Feather Stacking Headers - 12-pin and 16-pin female headers
$1.25 Add to Cart



Short Feather Headers Kit - 12-pin and 16-pin Female Header Set
$1.50 Add to Cart



Short Feather Male Headers - 12-pin and 16-pin Male Header Set
$0.50 Add to Cart
Add All To Cart

RELATED GUIDES

POPULAR

TTL Serial Camera

Snap, Snap!
by lady ada

This guide is for our new TTL serial camera module with NTSC video output. These modules are a nice addition to a microcontroller project when you want to take a photo or control a video stream. The modules have a few features built in, such as the ability to change the brightness/saturation/hue of images, auto-contrast and auto-brightness adjustment, and motion detection.
POPULAR

EL Wire

Working with Electroluminescent Wire
by lady ada

EL Wire, also known as Electroluminescent wire, is a stiff wire core coated with phosphor and then covered with a protective PVC sheath. When an AC signal is applied to it, it glows a cool neon color. Find out how to solder, power, and work with EL Wire in your next project.
POPULAR

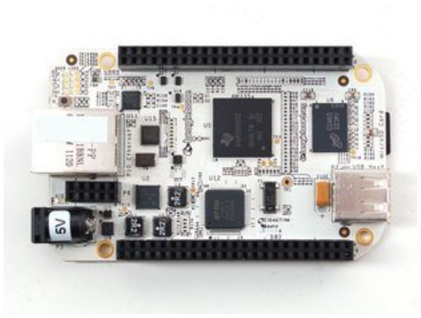
Hacking the Kinect

Reverse engineering the Microsoft Kinect
by lady ada

Here's a step by step guide on how you can reverse engineer a Microsoft Kinect for the Xbox 360.
POPULAR

BeagleBone

Tutorials for the TI embedded Linux board
by lady ada



New from the fine people who have brought us the Beagle Board, we now have a smaller, lighter, but powerful single board linux computer, Beagle Bone! We like this move to a more compact and integrated SBC. For example, there is onboard Ethernet and USB host, as well as a USB client interface (a FTDI chip for shell access). It even comes preloaded with Angstrom Linux on the 4 GB microSD card! Here are some tips and tricks to get your BeagleBone up and running.
×

OUT OF STOCK NOTIFICATION

YOUR NAME
YOUR EMAIL
NOTIFY ME

- CONTACT
- SUPPORT
- DISTRIBUTORS
- EDUCATORS
- JOBS
- FAQ
- SHIPPING & RETURNS
- TERMS OF SERVICE
- PRIVACY & LEGAL
- ABOUT US

ENGINEERED IN NYC Adafruit ®
"To achieve great things, two things are needed: a plan and not quite enough time" - Leonard Bernstein