

---

# **Adafruitrfm9x Library Documentation**

*Release 1.0*

**Tony DiCola**

**May 03, 2018**



---

## Contents

---

<b>1</b>	<b>Dependencies</b>	<b>3</b>
<b>2</b>	<b>Usage Example</b>	<b>5</b>
<b>3</b>	<b>Contributing</b>	<b>7</b>
<b>4</b>	<b>Building locally</b>	<b>9</b>
4.1	Zip release files . . . . .	9
4.2	Sphinx documentation . . . . .	9
<b>5</b>	<b>Table of Contents</b>	<b>11</b>
5.1	Simple test . . . . .	11
5.2	adafruit_rfm9x . . . . .	12
<b>6</b>	<b>Indices and tables</b>	<b>15</b>
	<b>Python Module Index</b>	<b>17</b>



CircuitPython module for the RFM95/6/7/8 LoRa 433/915mhz radio modules.



# CHAPTER 1

---

## Dependencies

---

This driver depends on:

- [Adafruit CircuitPython](#)
- [Bus Device](#)

Please ensure all dependencies are available on the CircuitPython filesystem. This is easily achieved by downloading the [Adafruit library and driver bundle](#).





## CHAPTER 2

---

### Usage Example

---

See `examples/simpletest.py` for a demo of the usage.



## CHAPTER 3

---

### Contributing

---

Contributions are welcome! Please read our [Code of Conduct](#) before contributing to help this project stay welcoming.



### 4.1 Zip release files

To build this library locally you'll need to install the `circuitpython-build-tools` package.

```
python3 -m venv .env
source .env/bin/activate
pip install circuitpython-build-tools
```

Once installed, make sure you are in the virtual environment:

```
source .env/bin/activate
```

Then run the build:

```
circuitpython-build-bundles --filename_prefix adafruit-circuitpython-rfm9x --library_
↪location .
```

### 4.2 Sphinx documentation

Sphinx is used to build the documentation based on rST files and comments in the code. First, install dependencies (feel free to reuse the virtual environment from above):

```
python3 -m venv .env
source .env/bin/activate
pip install Sphinx sphinx-rtd-theme
```

Now, once you have the virtual environment activated:

```
cd docs
sphinx-build -E -W -b html . _build/html
```

This will output the documentation to `docs/_build/html`. Open the `index.html` in your browser to view them. It will also (due to `-W`) error out on any warning like Travis will. This is a good way to locally verify it will pass.

## 5.1 Simple test

Ensure your device works with this simple test.

Listing 5.1: examples/simpletest.py

```
1  # Simple demo of sending and receiving data with the RFM95 LoRa radio.
2  # Author: Tony DiCola
3  import board
4  import busio
5  import digitalio
6
7  import adafruit_rfm9x
8
9
10 # Define radio parameters.
11 RADIO_FREQ_MHZ = 915.0 # Frequency of the radio in Mhz. Must match your
12                        # module! Can be a value like 915.0, 433.0, etc.
13
14 # Define pins connected to the chip, use these if wiring up the breakout according to
15 # the guide:
16 CS = digitalio.DigitalInOut(board.D5)
17 RESET = digitalio.DigitalInOut(board.D6)
18 # Or uncomment and instead use these if using a Feather M0 RFM9x board and the
19 # appropriate
20 # CircuitPython build:
21 #CS = digitalio.DigitalInOut(board.RFM9X_CS)
22 #RESET = digitalio.DigitalInOut(board.RFM9X_RST)
23
24 # Initialize SPI bus.
25 spi = busio.SPI(board.SCK, MOSI=board.MOSI, MISO=board.MISO)
26
27 # Initialize RFM radio
28 rfm9x = adafruit_rfm9x.RFM9x(spi, CS, RESET, RADIO_FREQ_MHZ)
```

```

27
28 # Note that the radio is configured in LoRa mode so you can't control sync
29 # word, encryption, frequency deviation, or other settings!
30
31 # You can however adjust the transmit power (in dB). The default is 13 dB but
32 # high power radios like the RFM95 can go up to 23 dB:
33 rfm9x.tx_power = 23
34
35 # Send a packet. Note you can only send a packet up to 252 bytes in length.
36 # This is a limitation of the radio packet size, so if you need to send larger
37 # amounts of data you will need to break it into smaller send calls. Each send
38 # call will wait for the previous one to finish before continuing.
39 rfm9x.send('Hello world!\r\n')
40 print('Sent hello world message!')
41
42 # Wait to receive packets. Note that this library can't receive data at a fast
43 # rate, in fact it can only receive and process one 252 byte packet at a time.
44 # This means you should only use this for low bandwidth scenarios, like sending
45 # and receiving a single message at a time.
46 print('Waiting for packets...')
47 while True:
48     packet = rfm9x.receive()
49     # Optionally change the receive timeout from its default of 0.5 seconds:
50     #packet = rfm9x.receive(timeout_s=5.0)
51     # If no packet was received during the timeout then None is returned.
52     if packet is None:
53         print('Received nothing! Listening again...')
54     else:
55         # Received a packet!
56         # Print out the raw bytes of the packet:
57         print('Received (raw bytes): {0}'.format(packet))
58         # And decode to ASCII text and print it too. Note that you always
59         # receive raw bytes and need to convert to a text format like ASCII
60         # if you intend to do string processing on your data. Make sure the
61         # sending side is sending ASCII data before you try to decode!
62         packet_text = str(packet, 'ascii')
63         print('Received (ASCII): {0}'.format(packet_text))
64         # Also read the RSSI (signal strength) of the last received message and
65         # print it.
66         rssi = rfm9x.rssi
67         print('Received signal strength: {0} dB'.format(rssi))

```

## 5.2 adafruit\_rfm9x

CircuitPython module for the RFM95/6/7/8 LoRa 433/915mhz radio modules. This is adapted from the Radiohead library RF95 code from: <http://www.airspayce.com/mikem/arduino/RadioHead/>

- Author(s): Tony DiCola

**class** adafruit\_rfm9x.**RFM9x**(spi, cs, reset, frequency, \*, preamble\_length=8, high\_power=True, baudrate=1000000)

Interface to a RFM95/6/7/8 LoRa radio module. Allows sending and receiving bytes of data in long range LoRa mode at a support board frequency (433/915mhz).

You must specify the following parameters: - spi: The SPI bus connected to the radio. - cs: The CS pin DigitalInOut connected to the radio. - reset: The reset/RST pin DigitalInOut connected to the radio. - frequency:



The frequency (in mhz) of the radio module (433/915mhz typically).

You can optionally specify: - `preamble_length`: The length in bytes of the packet preamble (default 8). - `high_power`: Boolean to indicate a high power board (RFM95, etc.). Default is True for high power. - `baudrate`: Baud rate of the SPI connection, default is 10mhz but you might choose to lower to 1mhz if using long wires or a breadboard.

Remember this library makes a best effort at receiving packets with pure Python code. Trying to receive packets too quickly will result in lost data so limit yourself to simple scenarios of sending and receiving single packets at a time.

Also note this library tries to be compatible with raw RadioHead Arduino library communication. This means the library sets up the radio modulation to match RadioHead's defaults. Features like addressing and guaranteed delivery need to be implemented at an application level.

#### **frequency\_mhz**

The frequency of the radio in Megahertz. Only the allowed values for your radio must be specified (i.e. 433 vs. 915 mhz)!

#### **idle ()**

Enter idle standby mode.

#### **listen ()**

Listen for packets to be received by the chip. Use `receive ()` to listen, wait and retrieve packets as they're available.

#### **preamble\_length**

The length of the preamble for sent and received packets, an unsigned 16-bit value. Received packets must match this length or they are ignored! Set to 8 to match the RadioHead RFM95 library.

#### **receive (timeout\_s=0.5, keep\_listening=True)**

Wait to receive a packet from the receiver. Will wait for up to `timeout_s` amount of seconds for a packet to be received and decoded. If a packet is found the payload bytes are returned, otherwise None is returned (which indicates the timeout elapsed with no reception). Note this assumes a 4-byte header is prepended to the data for compatibility with the RadioHead library (the header is not validated nor returned). If `keep_listening` is True (the default) the chip will immediately enter listening mode after reception of a packet, otherwise it will fall back to idle mode and ignore any future reception.

#### **reset ()**

Perform a reset of the chip.

#### **rss\_i**

The received strength indicator (in dBm) of the last received message.

#### **send (data)**

Send a string of data using the transmitter. You can only send 252 bytes at a time (limited by chip's FIFO size and appended headers). Note this appends a 4 byte header to be compatible with the RadioHead library.

#### **sleep ()**

Enter sleep mode.

#### **transmit ()**

Transmit a packet which is queued in the FIFO. This is a low level function for entering transmit mode and more. For generating and transmitting a packet of data use `send ()` instead.

#### **tx\_power**

The transmit power in dBm. Can be set to a value from 5 to 23 for high power devices (RFM95/96/97/98, `high_power=True`) or -1 to 14 for low power devices. Only integer power levels are actually set (i.e. 12.5 will result in a value of 12 dBm).



## CHAPTER 6

---

### Indices and tables

---

- `genindex`
- `modindex`
- `search`



**a**

[adafruit\\_rfm9x](#), 12



## A

adafruit\_rfm9x (module), 12

## F

frequency\_mhz (adafruit\_rfm9x.RFM9x attribute), 13

## I

idle() (adafruit\_rfm9x.RFM9x method), 13

## L

listen() (adafruit\_rfm9x.RFM9x method), 13

## P

preamble\_length (adafruit\_rfm9x.RFM9x attribute), 13

## R

receive() (adafruit\_rfm9x.RFM9x method), 13

reset() (adafruit\_rfm9x.RFM9x method), 13

RFM9x (class in adafruit\_rfm9x), 12

rsi (adafruit\_rfm9x.RFM9x attribute), 13

## S

send() (adafruit\_rfm9x.RFM9x method), 13

sleep() (adafruit\_rfm9x.RFM9x method), 13

## T

transmit() (adafruit\_rfm9x.RFM9x method), 13

tx\_power (adafruit\_rfm9x.RFM9x attribute), 13