# Using LoraWAN and The Things Network with CircuitPython

Created by Brent Rubell

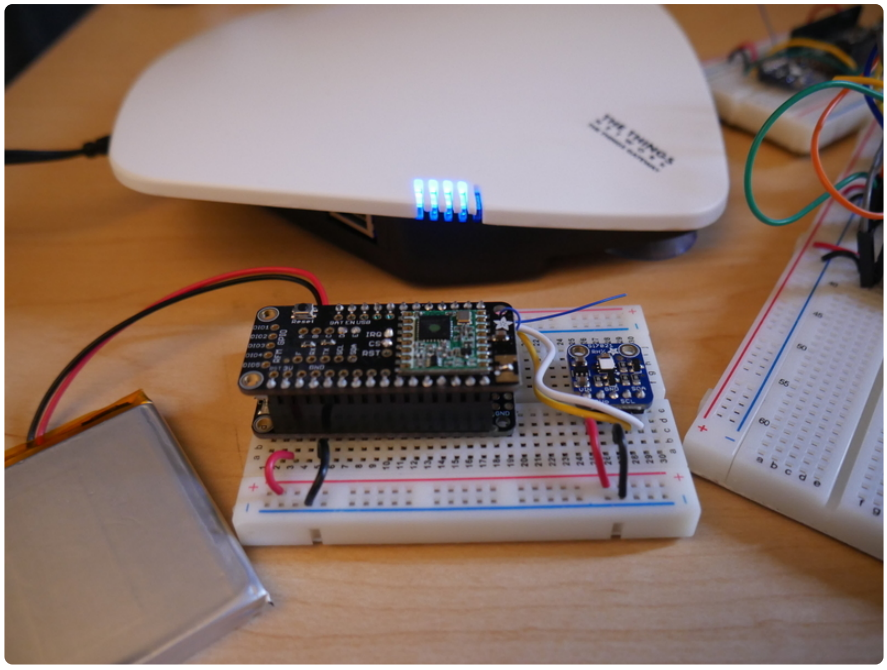

https://learn.adafruit.com/using-lorawan-and-the-things-network-with-circuitpython

Last updated on 2024-06-03 02:34:29 PM EDT

# Table of Contents

# Overview

This guide is deprecated and is no longer possible! Single-channel packet forwarders no longer work after the Things Network migration to The Things Stack v3. For more information about this decision, visit: https://www.thethingsnetwork.org/forum/t/single-channel-packet-forwarders-scpf-are-deprecated-and-not-supported/31117



In this project, we're going to build a small weather-logging node using a Feather and a temperature sensor. The captured data will then be sent over LoRaWAN to The Things Network.

Wait, this sounds similar to the a previous guide we have (https://adafru.it/Dei). What's different?

Good catch - the difference is that **this guide is for use with** CircuitPython **(https://adafru.it/CgS)**, meaning **you can get your project up and running on the Things Network quicker than ever**. Also, since we're using CircuitPython, this guide is compatible with Python Linux boards, like the Raspberry Pi family.

## What is The Things Network?

The Things Network (https://adafru.it/BsB) is a project dedicated to building a network (https://adafru.it/Cz8) for the Internet of Things. While WiFi is used in most Internet of Things devices, The Things Network uses a protocol called **LoRaWAN**

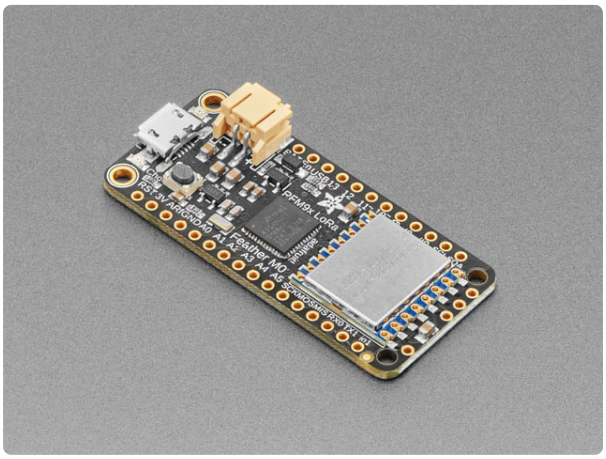
which allows devices to talk to the internet without cellular or WiFi connectivity. This means you don't need to worry about protected wireless hotspots, cellular data plans, or spotty WiFi connectivity.

It's **ideal** for most internet of things projects, and unlike cellular plans or WiFi - it's **free to use**.

There are many gateways available to connect your CircuitPython device to - if you'd like to find a gateway in your area, check the Gateway Map. (https://adafru.it/Cz9)

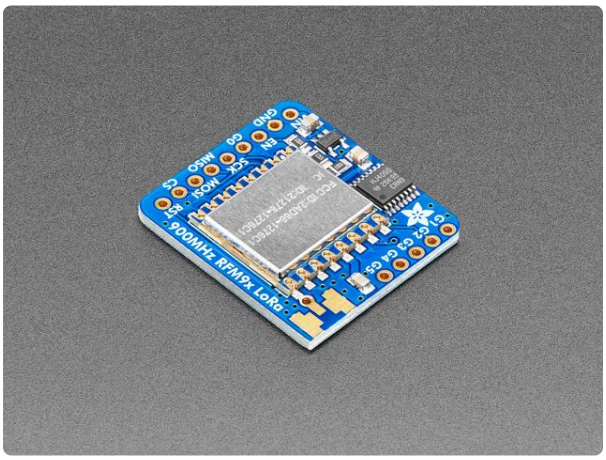
## Parts

Our **Feather M0 RFM9x** is an all-in-one Feather with an onboard RFM9x radio module cooked in, built-in USB, and battery charging (highly useful for deploying LoRa nodes).

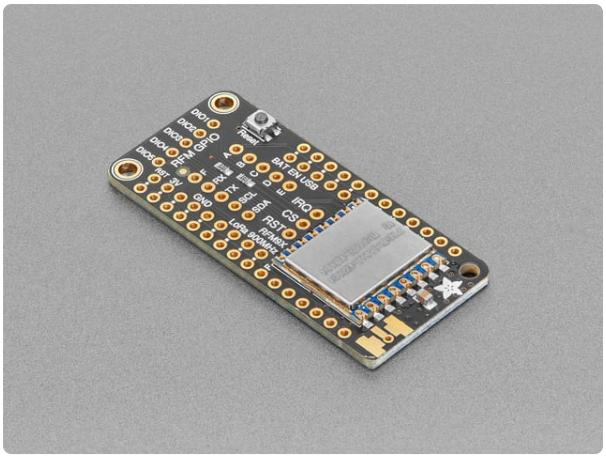


Adafruit Feather M0 with RFM95 LoRa Radio - 900MHz
This is the Adafruit Feather M0 RFM95 LoRa Radio (900MHz). We call these RadioFruits, our take on an microcontroller with a...
https://www.adafruit.com/product/3178

**Already have a CircuitPython-Compatible board (or a Raspberry Pi)?** There are **multiple** ways to connect - the all-in-one Feather M0 LoRa and the stack-able Radio FeatherWing:



Adafruit RFM95W LoRa Radio Transceiver Breakout - 868 or 915 MHz
"You see, wire telegraph is a kind of a very, very long cat.  You pull his tail in New York and his head is meowing in Los Angeles.  Do you understand this? And...
https://www.adafruit.com/product/3072

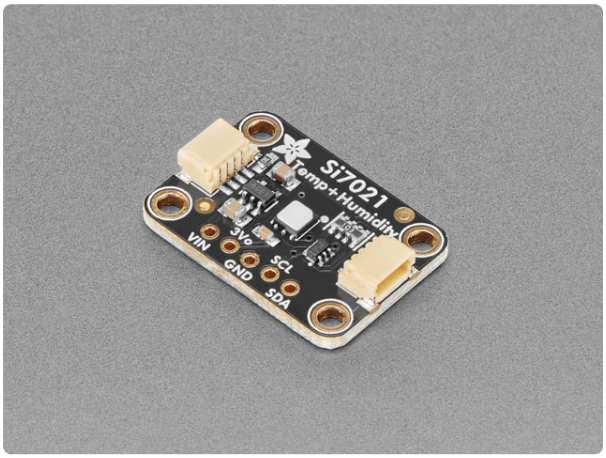


Adafruit LoRa Radio FeatherWing - RFM95W 900 MHz

Add short-hop wireless to your Feather with these RadioFruit Featherwings. These add-ons for any Feather board will let you integrate packetized radio (with the RFM69 radio) or LoRa...

https://www.adafruit.com/product/3231

We'll also use a Si7021 temperature and humidity sensor to log and record whether it's sweltering or freezing.




Adafruit Si7021 Temperature & Humidity Sensor Breakout Board

It's summer and you're sweating and your hair's all frizzy and all you really want to know is why the weatherman said this morning that today's relative humidity would...

https://www.adafruit.com/product/3251

## Materials

You'll want to pick these up from the Adafruit Store if you don't have them on-hand already:

**1 x** Half-size breadboard — https://www.adafruit.com/product/64

Hook up breakouts easily

---

**1 x** Breadboard Wire Bundle — https://www.adafruit.com/product/153

Lots of wires for use with any breadboard

---

If you're running a weather logger, you'll want to include a LiPo Battery in your build. LoRa has less power-draw than WiFi, so your node should be able to run unattended for a while.

**1 x** LiPo Battery — https://www.adafruit.com/product/2011

Ideal for Feather

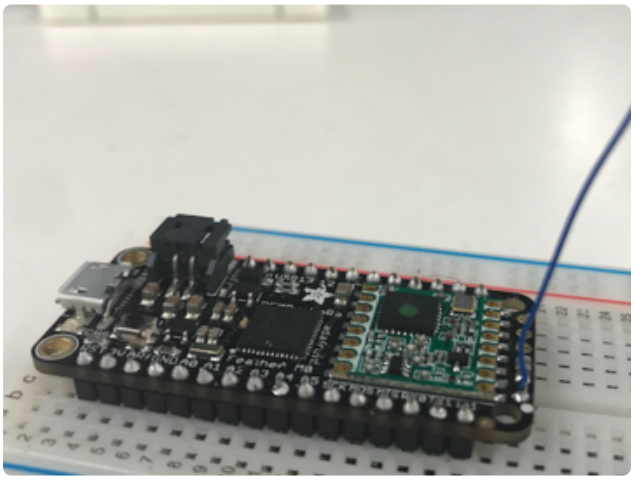
---

# Wiring

There's quite a few ways to connect to The Things Network with CircuitPython. You can use a Feather RFM9x (an all-in-one), a breakout or FeatherWing and a CircuitPython-compatible board, or a Python Linux computer/board.

## Wiring with a Feather M0 RFM9x

### Attaching an Antenna

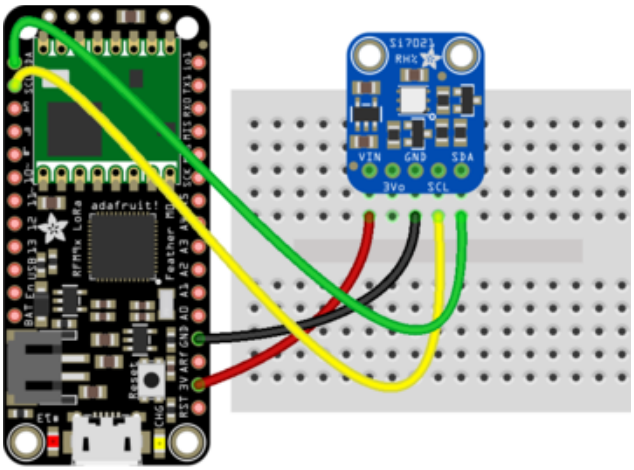


Your Feather M0 does not come with an antenna, but there are two ways of wiring one up.

For this guide, and to keep the build cost-effective, we soldered a small, 82mm wire to the **ANT** pad.

Antenna Options and installation instructions are detailed on this product's learn guide's **antenna options** page. (https://adafru.it/CyV)
**Note:** Antenna length differs between regions, **make sure you cut the antenna to the correct length for your region.**

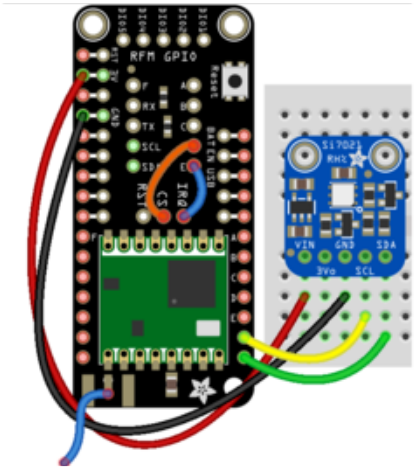
### Wiring




Make the following connections between the **SI7021** and the **Feather RFM9x**:

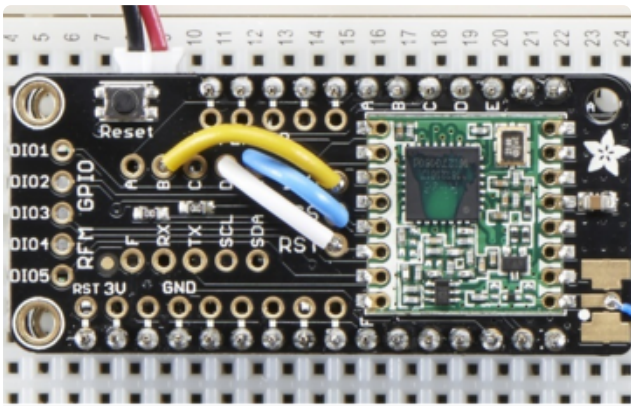
**Feather 3V** to **SI7021 VIN**
**Feather GND** to **SI7021 GND**
**Feather SCL** to **SI7021 SCL**
**Feather SDA** to **SI7021 SDA**

## Wiring with a Radio Featherwing



Make the following connections between the **Radio FeatherWing** and the **Si7021**:

**Wing 3V** to **SI7021 VIN**
**Wing GND** to **SI7021 GND**
**Wing SCL** to **SI7021 SCL**
**Wing SDA** to **SI7021 SDA**

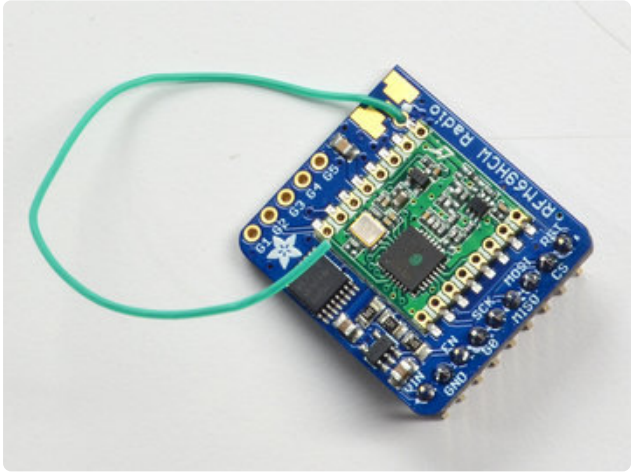



The pins for **IRQ** (RFM9x's DIO0 Pin) and **CS** are configurable. We provide a few lettered pins. Solder them together like so with some short jumper wires:

**Wing CS** to **Wing D**
**Wing IRQ** to **Wing E**

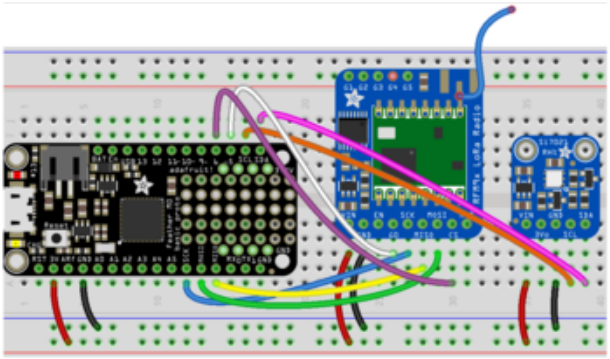
## Wiring a LoRa Radio Transceiver Breakout

### Attaching an Antenna



These radio breakouts do not have a built-in antenna. Instead, you have three options for attaching an antenna. For most low cost radio nodes, a Wire antenna (https://adafru.it/Dej) works great. If you need to put the radio into an enclosure, soldering in a UFL connector (https://adafru.it/Dej). and using a uFL to SMA adapter will let you attach an external antenna. You can also solder an SMA edge-mount connector directly.

## Wiring a CircuitPython-Compatible Board with a LoRa Breakout



Make the following connections between the **RFM Breakout** and the **board**:

**Board 3V** to **Radio VIN**
**Board GND** to **Radio GND**
**Board SCK** to **Radio SCK**
**Board MOSI** to **Radio MOSI**
**Board MISO** to **Radio MISO**
**Board D5** to **Radio G0**
**Board D6** to **Radio CS**

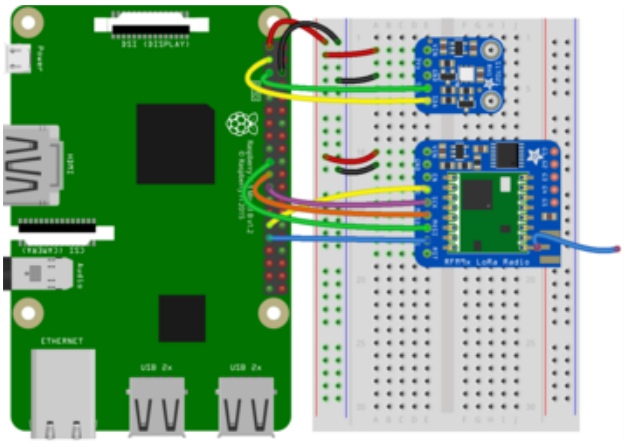
Make the following connections between the **Si7021** and the **board**:

**Board 3V** to **Si7021 VIN**
**Board GND** to **Si7021 GND**
**Board SCL** to **Si7021 SCL**
**Board SDA** to **Si7021 SDA**

## Python Computer Wiring with a LoRa Breakout

Since there's dozens of Linux computers/boards you can use we will show wiring for Raspberry Pi. For other platforms, please visit the guide for CircuitPython on Linux to see whether your platform is supported (https://adafru.it/BSN).

Here's the Raspberry Pi wired with a RFM9x Breakout and a SI7021:



Connect the Raspberry Pi to the SI7021 Sensor

**Pi 3V3** to **SI7021 VIN**
**Pi GND** to **SI7021 GND**
**Pi SCL** to **SI7021 SCL**
**Pi SDA** to **SI7021 SDA**
Make the following connections between the RFM breakout and the Raspberry Pi:

**Pi 3V** to **Radio VIN**
**Pi GND** to **Radio GND**
**Pi SCK** to **Radio SCK**
**Pi MOSI** to **Radio MOSI**
**Pi MISO** to **Radio MISO**
**Board D5** to **Radio G0**
**Board D6** to **Radio CS**

All wired up? Let's move on to setting up The Things Network for TinyLoRa.

---

# TinyLoRa TTN Setup

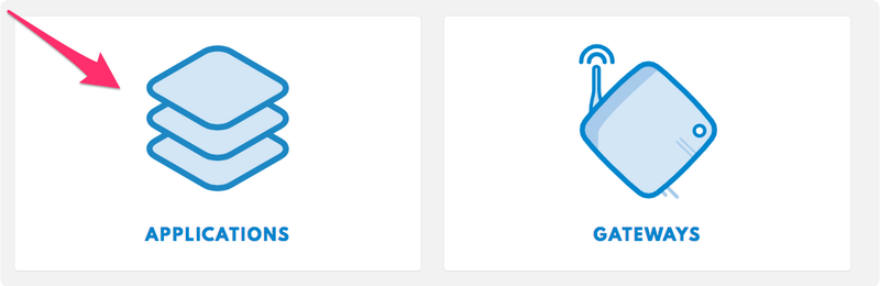
This guide is deprecated and is no longer possible! Single-channel packet forwarders no longer work after the Things Network migration to The Things Stack v3. For more information about this decision, visit: https://www.thethingsnetwork.org/forum/t/single-channel-packet-forwarders-scpf-are-deprecated-and-not-supported/31117

Feathers running TinyLoRa require a different setup process. Be sure to follow these steps closely if you haven't set up an ABP device before.

Before your Feather can communicate with The Things Network, you'll need to create an application.

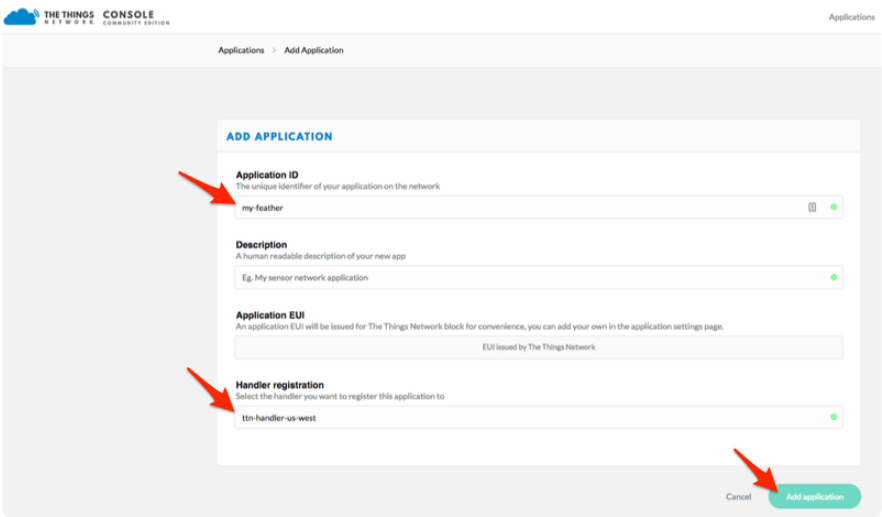
First, we're going to register an account with TTN. Navigate to their account registration page (https://adafru.it/CyW) to set up an account.

Once logged in, navigate to the The Things Network Console (https://adafru.it/CyX). This page is where you can register applications and add new devices or gateways. Click **Applications**



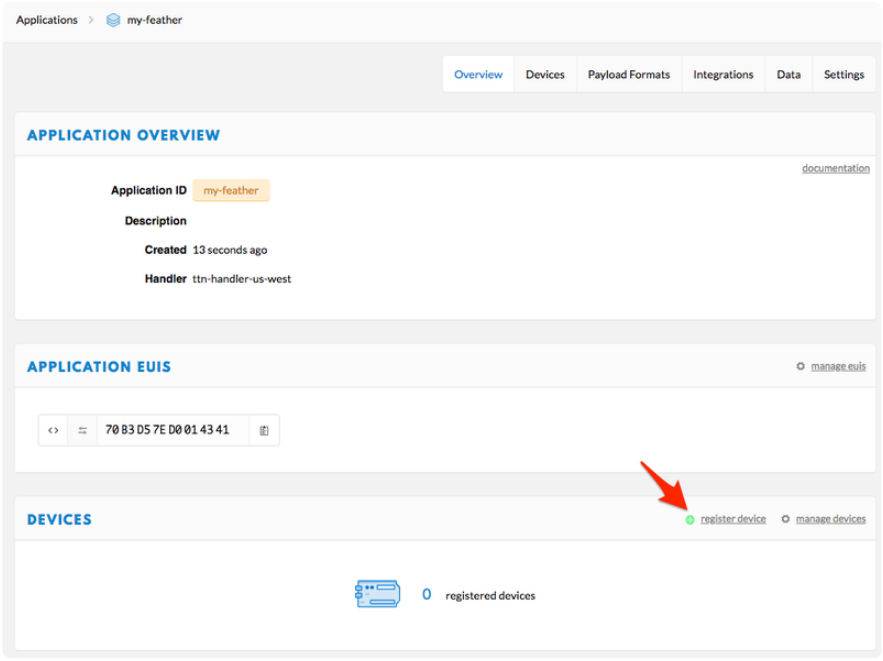

Click **Add Application**.

Fill out an **Application ID** to identify the application, and a **description** of what the application is. We set our **Handler Registration** to match our region, us-west. If you're not located in the U.S., TTN provides multiple regions for handler registration.




Once created, you'll be directed to the Application Overview. From here, you can add devices, view data coming into (and out of) the application, add integrations for external services, and more. We'll come back to this section later in the guide.

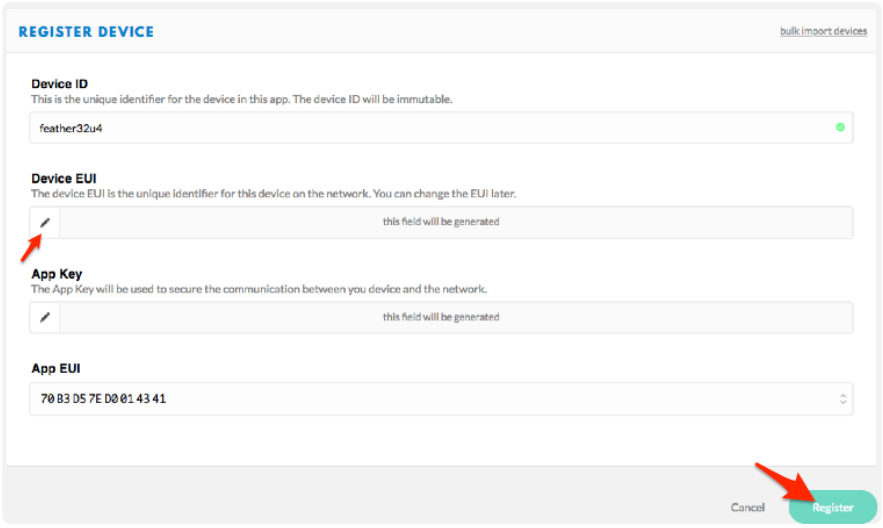
### Click Register Device




On the **Register Device Page**, The **Device ID** should be a unique string to identify the device.

The **Device EUI** is the unique identifier which came in the same bag as your Radiofruit Feather. We'll pad the middle of the string with four zeroes. Lost your key? No worries - you can also click the "mix" icon to switch it to auto-generate.

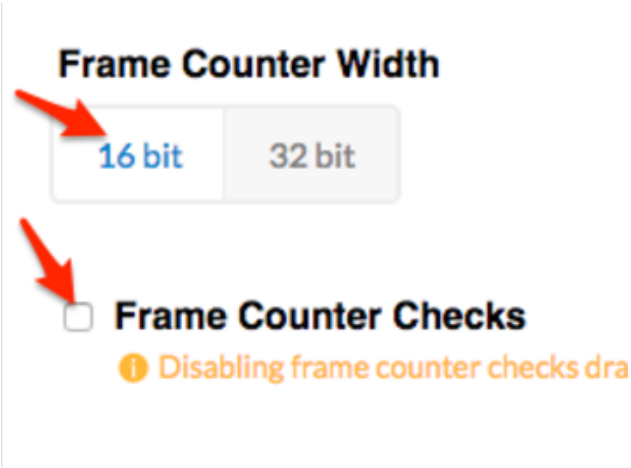
The **App Key** will be randomly generated for you by TTN. Select the **App EUI** (used to identify the application) from the list.




Next, we're going to switch the device settings from Over-the-Air-Activation to Activation-by-Personalization. From the Device Overview, **click Settings**



On the settings screen, **change the Activation Method from OTAA to ABP**.




Then, **switch the Frame Counter Width from 32b to 16b** and **disable frame counter checks**. TTN will display a warning, ignore it, and **click Save**.

Make sure you have disabled Frame Counter Checks

Now that the application is set up, and the device is registered to the application, let's move on to setting up Arduino with TinyLoRa.

---

## Why are we disabling Frame Counter Checks if The Things Network Console doesn't recommend it?

Disabling frame counter checks allows you to transmit data to The Things Network without requiring a match between your device's frame counter and the console's frame counter.

If you're making a project and doing a lot of prototyping/iteration to the code, disabling these checks is okay as it'll let you reset the device and not re-register it to the application (it'll also prevent counter overflows).

If you're deploying a project, you'll want to re-activate the frame counter for security purposes. With the frame counter disabled, one could re-transmit the messages sent to TTN using a replay attack.

The Things Network's documentation page has a full explanation about the role of the Frame Counter (https://adafru.it/Deu).

---

# CircuitPython Installation

## Installing CircuitPython on a Feather RFM9x

Some of the CircuitPython compatible boards come with CircuitPython installed. Others are CircuitPython-ready, but need to have it installed.

**If you're using a Feather M0 RFM9x (http://adafru.it/3178), you'll need to install CircuitPython on your board.**

- **If you're using a RFM9x breakout and a Feather Express or a Python Linux board, you can ignore this page** and continue to the library installation (https://adafru.it/Dek).

If you have Arduino IDE downloaded and installed (https://adafru.it/Cto), this process is as simple as uploading a sketch to your board then dragging and dropping a file.

First, we need to flash the UF2 bootloader onto the Feather.

Navigate to the latest release page for the UF2 bootloader on the Adafruit GitHub.

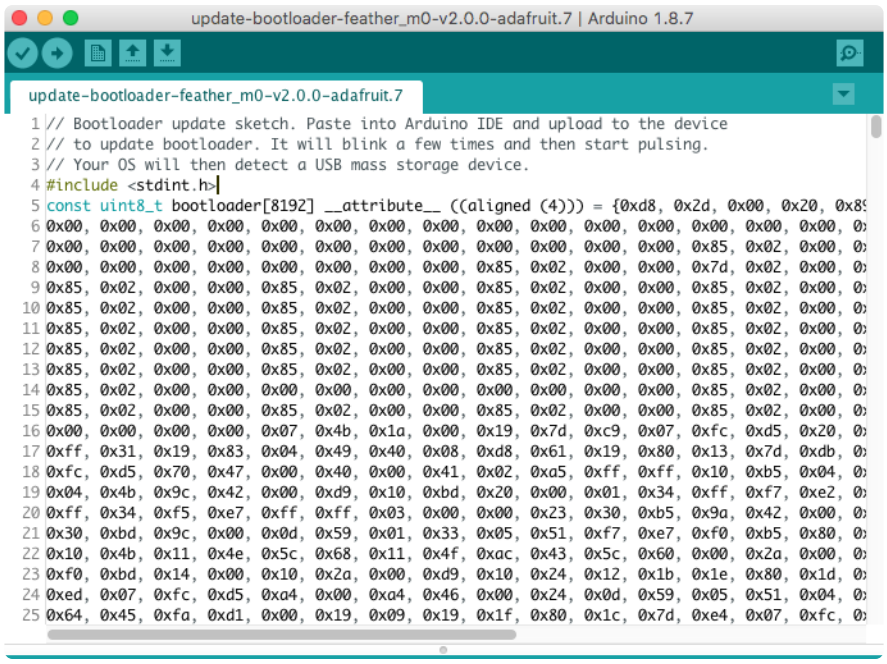
**Click here to see the latest UF2 bootloader releases.**

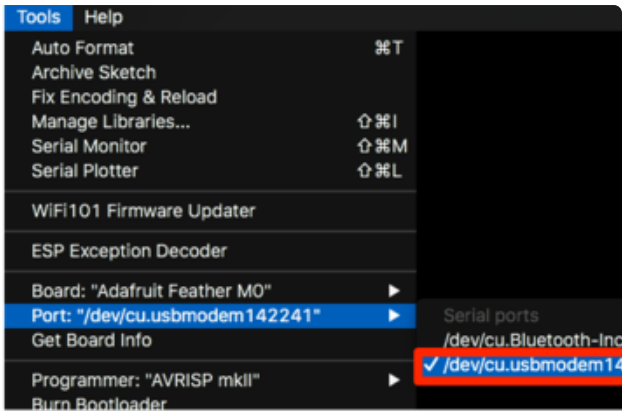
https://adafru.it/D3C

**Click update-bootloader-feather_m0-v2.x.x-adafruit.7.ino** to download the file.

**Double click** the file to open it in the Arduino IDE.

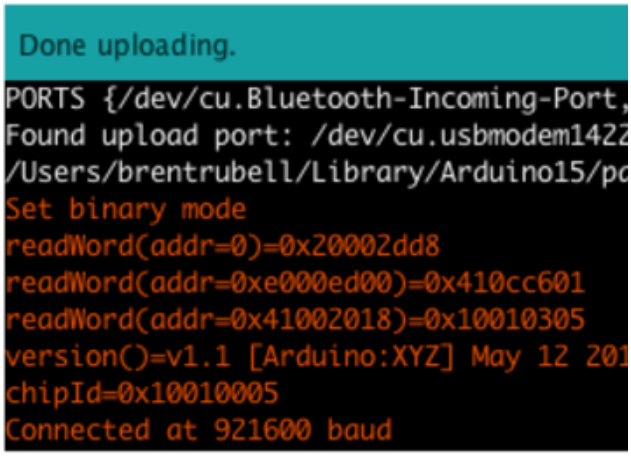
The **.ino** sketch should download to your computer. Open the file in the Arduino IDE by **double-clicking the file**. The Arduino IDE should load and the sketch should appear. The sketch will have a lot of hexadecimal numbers and might look daunting if you scroll down - don't worry, we're not touching any of it.



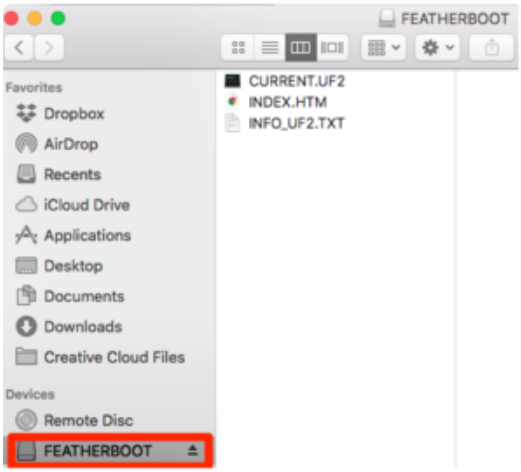




From the Arduino IDE, select the **Port** associated with your Feather.

Not seeing a port displayed? Ensure your Feather M0 RFM9x (https://adafru.it/Del) has been properly set up for the Arduino IDE.




Click **Upload** (**CTRL/CMD+U**) to upload the bootloader to your Feather. The sketch will flash the bootloader onto the board.




The board should reset. After resetting, the board should connect to your computer and appear as a volume on your computer named **FEATHERBOOT**. If you see this volume, you've successfully loaded the UF2 bootloader onto your Feather.

We're almost done, but not yet - we still need to load CircuitPython on the board.

## Download the latest CircuitPython version

Navigate to the CircuitPython release page (https://adafru.it/tBa) and select the latest build for the **Feather M0 RFM9x**.

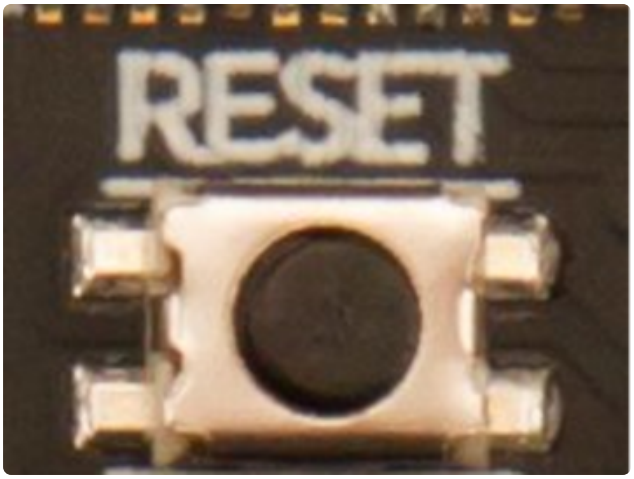
Click on the build for your region and Feather to download it to your computer. Put it somewhere you can remember, like your Desktop.


**Make sure you select a .UF2 file, not a .BIN file.**

For example, if I'm using a Feather M0 RFM9x in the United States, I'll select the build: **adafruit-circuitpython-feather_m0_rfm9x-en_US-4.0.0.uf2**

Next, we're going to load CircuitPython **(UF2 file)** onto our board.

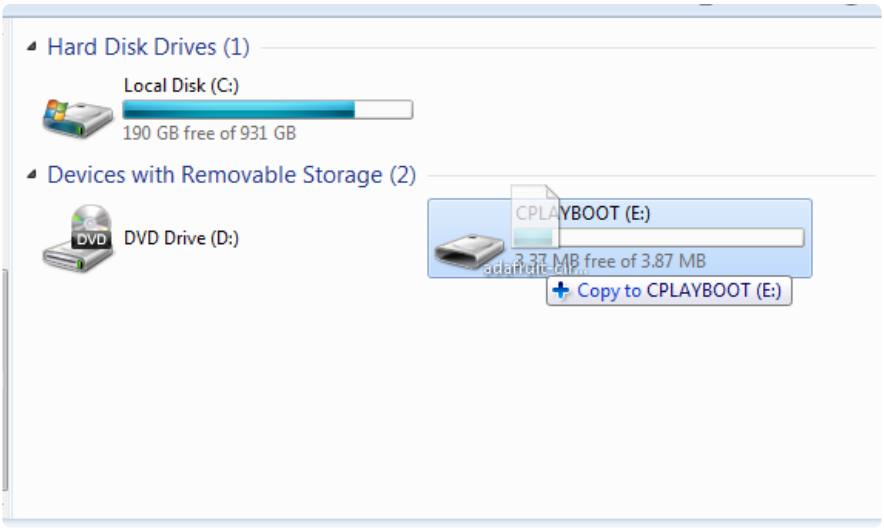
## Start the UF2 Bootloader



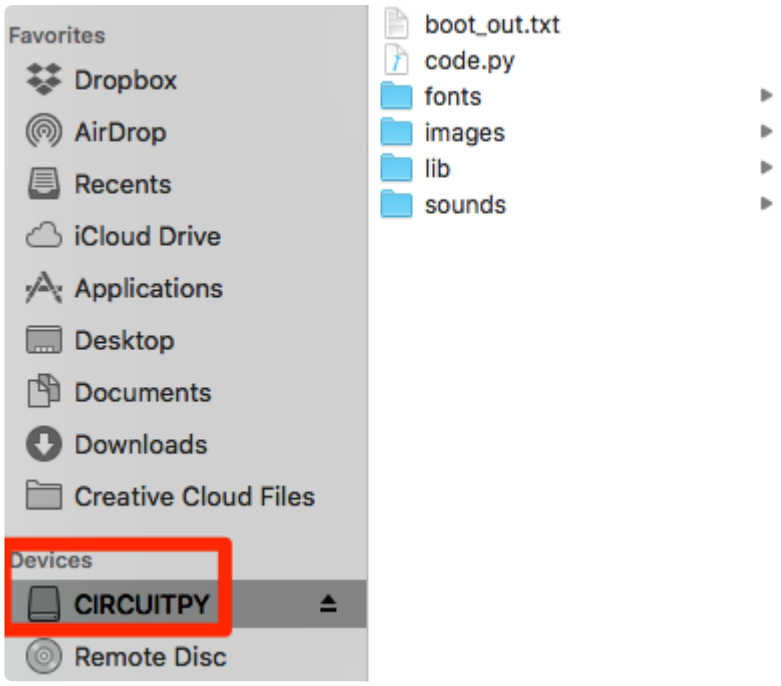

If you see the board on your computer as a volume, `FEATHERBOOT`, you can proceed with these steps.

Don't see `FEATHERBOOT`? **Tap the small, black, reset button on your board twice** to enter the bootloader. If it doesn't work on the first try, don't be discouraged. The rhythm of the taps needs to be correct and sometimes it takes a few tries.

Now find the **UF2 file** you downloaded. Drag that file to the `FEATHERBOOT` drive on your computer.




The lights should flash again, `FEATHERBOOT` will disappear and a **new** drive will show up on your computer called `CIRCUITPY`.




Congratulations! You've successfully installed CircuitPython!

---

# Library Installation

## Bundle Installation

We're constantly building/updating/improving our libraries, so we don't ship all the CircuitPython libraries with your board.

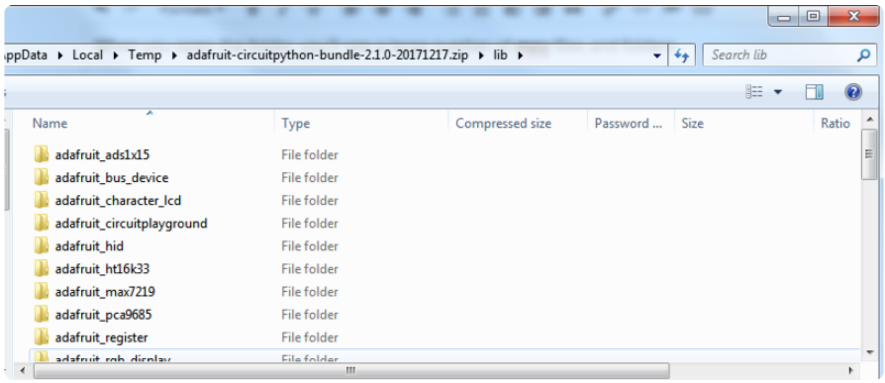
You'll want to grab a copy of the CircuitPython Library Bundle (https://adafru.it/zdx) - a .zip file containing a bunch of useful CircuitPython libraries. The TinyLoRa library is separate repository (https://adafru.it/Dem), but it's easier to install everything at once using the bundle.

Grab the latest version of the library bundle by clicking this button and **make sure you download the version that matches your CircuitPython library version**:

**Click for the latest CircuitPython Library Bundle**

https://adafru.it/y8E

Unzip the library. You'll see a two folders: `lib` and `examples`. Double click the `lib` folder to see a listing of all the CircuitPython libraries.




## If you're using an Express board

On **Express** boards (the board will say 'EXPRESS' on it),
the `lib` and `examples` directories can be copied directly to the CIRCUITPY drive.

Just drag the entire `lib` and `examples` (optional) folders into the CIRCUITPY drive, and 'replace' any old files if your operating system prompts you.

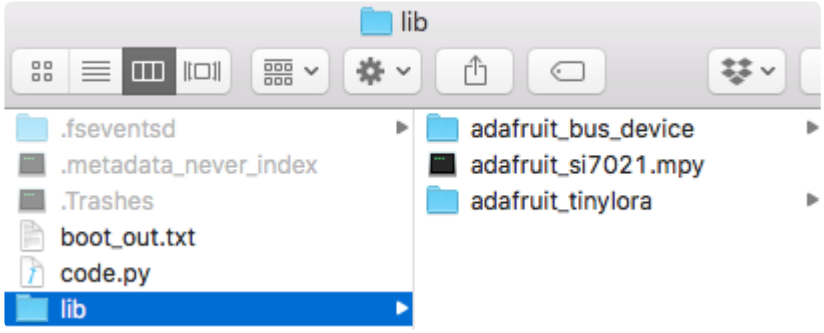
## If you're using a non-Express board

The RFM9x Feather M0 (http://adafru.it/3178) (all-in-one) has less space available compared to express boards. To save space, we're only going to install the libraries required for this guide.

From the bundle folder, navigate to the `lib` folder and find the required folders: `adafruit_bus_device`, `adafruit_tinylora` and `adafruit_si7021`.

Copy these three folders/files over to the `CIRCUITPY` drive's `lib` directory.

Before continuing make sure your board's `lib` folder has at the `adafruit_tinylora`, `adafruit_bus_device`, and `adafruit_si7021` libraries copied.



## Running out of space on the CIRCUITPY Volume?

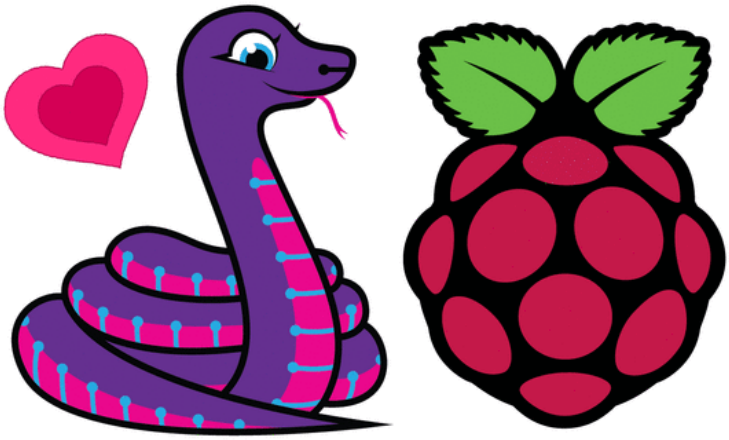
If you find yourself running out of space on your non-express board and are a computer running MacOS - there's a chance that your mac is creating hidden files which are taking up space on the board.

Make sure you're running the latest version of CircuitPython for your board, and follow the guide here for information on how to prevent these files from being created (https://adafru.it/Den).

For further space savings- remove the TTN configurations for any country other than your own (you'll only need `ttn_usa.mpy` if you're in the United States).

---

## Installation on Python Linux Computers



If you're following along with a Raspberry Pi, Beaglebone or any other supported small Linux computer, we'll use a special library called **adafruit_blinka** (https://adafru.it/BJS) (named after Blinka, the CircuitPython mascot (https://adafru.it/BJT)) to provide the layer that translates the CircuitPython hardware API to whatever library the Linux board provides. It's CircuitPython, on Pi!

If you haven't set up Blinka and the Adafruit IO Python Library yet on your Raspberry Pi, follow our guide:

- Blinka and Adafruit IO Setup (https://adafru.it/Deo)

## Enable I2C

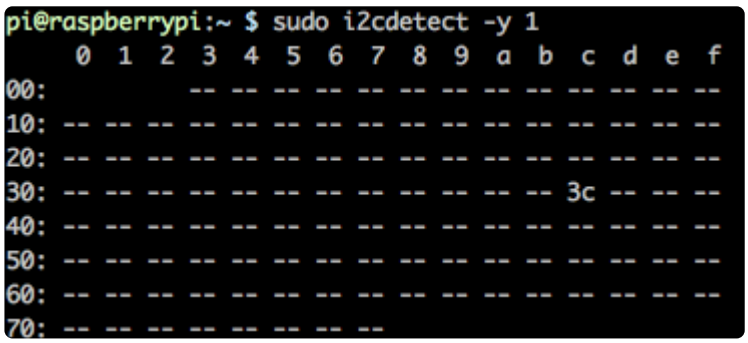
We use two pins on the Pi (SDA/SCL) to communicate over I2C with the Si7021. You only have to do this step once per Raspberry Pi, the I2C interface is disabled by default.

- Enabling I2C (https://adafru.it/dEO)

Once you're done with this and have rebooted, verify you have the I2C devices with the command:

```
sudo i2cdetect -y 1
```

The output from running this command should look like the following:




## Enable SPI

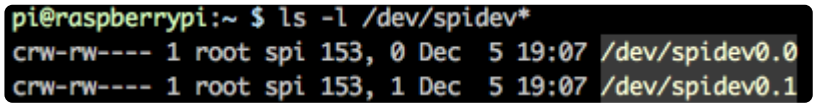
You'll also need to enable SPI to communicate over SPI with the RFM breakout. You only have to do this step once per Raspberry Pi, the SPI interface is disabled by default.

- Enabling SPI (https://adafru.it/BJZ)

Once you're done enabling SPI and have rebooted, verify you have the SPI interface by typing the following command into the terminal:

```
ls -l /dev/spidev*
```

You should see the following output:



## Installing required CircuitPython Libraries

Next, you'll need to install libraries to communicate with the RFM9x breakout and the Si7021. Since we're using Adafruit Blinka (CircuitPython), we can install CircuitPython libraries straight to our Raspberry Pi, instead of installing from separate files.

Run the following command from your terminal to **install the** Adafruit_CircuitPython_Si7021 Library **(https://adafru.it/BfV)**:

```
sudo pip3 install adafruit-circuitpython-si7021
```

Then, **install the** Adafruit_CircuitPython_TinyLoRa **(https://adafru.it/Bn-) library**

```
sudo pip3 install adafruit-circuitpython-tinylora
```

---

# Using TinyLoRa

## Writing code With Mu

Adafruit recommends the Mu editor to type your code into and interact with your project. You can find out how to install Mu on mac, PC, and Linux in this guide page (https://adafru.it/ANO) on the Adafruit Learning System.

The advantage to using Mu is it has an editor and added features like direct save to the board's REPL interactive command line and print output. Mu can even plot values for you.

## The Code

The code below will use the `adafruit_tinylora` library which provides high-level access to your radio transceiver's features.

The code for sending data using a Si7021 sensor to a Things Network Gateway is shown below:

```
# SPDX-FileCopyrightText: 2021 ladyada for Adafruit Industries
# SPDX-License-Identifier: MIT

"""Using TinyLoRa with a Si7021 Sensor.
"""
import time

import adafruit_si7021
```

```
import board
import busio
import digitalio

from adafruit_tinylora.adafruit_tinylora import TTN, TinyLoRa

try:  # typing
    from typing import Annotated, TypeAlias

    bytearray4: TypeAlias = Annotated[bytearray, 4]
    bytearray16: TypeAlias = Annotated[bytearray, 16]
except ImportError:
    pass

# Board LED
led = digitalio.DigitalInOut(board.D13)
led.direction = digitalio.Direction.OUTPUT

# Create library object using our bus i2c port for si7021
i2c = busio.I2C(board.SCL, board.SDA)
sensor = adafruit_si7021.SI7021(i2c)

# Create library object using our bus SPI port for radio
spi = busio.SPI(board.SCK, MOSI=board.MOSI, MISO=board.MISO)

# RFM9x Breakout Pinouts
cs = digitalio.DigitalInOut(board.D5)
irq = digitalio.DigitalInOut(board.D6)
rst = digitalio.DigitalInOut(board.D4)

# Feather M0 RFM9x Pinouts
# cs = digitalio.DigitalInOut(board.RFM9X_CS)
# irq = digitalio.DigitalInOut(board.RFM9X_D0)
# rst = digitalio.DigitalInOut(board.RFM9X_RST)

# TTN Device Address, 4 Bytes, MSB
devaddr: bytearray4 = bytearray([0x00, 0x00, 0x00, 0x00])

# TTN Network Key, 16 Bytes, MSB
nwkey: bytearray16 = bytearray(
    [
        0x00,
        0x00,
        0x00,
        0x00,
        0x00,
        0x00,
        0x00,
        0x00,
        0x00,
        0x00,
        0x00,
        0x00,
        0x00,
        0x00,
        0x00,
        0x00,
    ]
)

# TTN Application Key, 16 Bytess, MSB
app: bytearray16 = bytearray(
    [
        0x00,
        0x00,
        0x00,
        0x00,
        0x00,
        0x00,
```

```
        0x00,
        0x00,
        0x00,
        0x00,
        0x00,
        0x00,
        0x00,
        0x00,
        0x00,
        0x00,
    ]
)

ttn_config = TTN(devaddr, nwkey, app, country="US")

lora = TinyLoRa(spi, cs, irq, rst, ttn_config)

# Data Packet to send to TTN
data: bytearray4 = bytearray(4)

while True:
    temp_val = sensor.temperature
    humid_val = sensor.relative_humidity
    print("Temperature: %0.2f C" % temp_val)
    print("relative humidity: %0.1f %%" % humid_val)

    # Encode float as int
    temp_val = int(temp_val * 100)
    humid_val = int(humid_val * 100)

    # Encode payload as bytes
    data[0] = (temp_val >> 8) & 0xFF
    data[1] = temp_val & 0xFF
    data[2] = (humid_val >> 8) & 0xFF
    data[3] = humid_val & 0xFF

    # Send data packet
    print("Sending packet...")
    lora.send_data(data, len(data), lora.frame_counter)
    print("Packet Sent!")
    led.value = True
    lora.frame_counter += 1
    time.sleep(2)
    led.value = False
```

Using Mu (or a text editor), copy this code and save it to your computer, naming it `code.py`.

Note: The code is configured to use a RFM9x breakout by default. If you are using a Feather M0 RFM9x, delete the following lines:

```
# RFM9x Breakout Pinouts
```

```
cs = digitalio.DigitalInOut(board.D5)
```

```
irq = digitalio.DigitalInOut(board.D6)
```

```
rst = digitalio.DigitalInOut(board.D4)
```

Then, uncomment (remove the `#`) the following lines to enable the RFM9x's builtin D0 and CS pins

```
Feather M0 RFM9x Pinouts
```

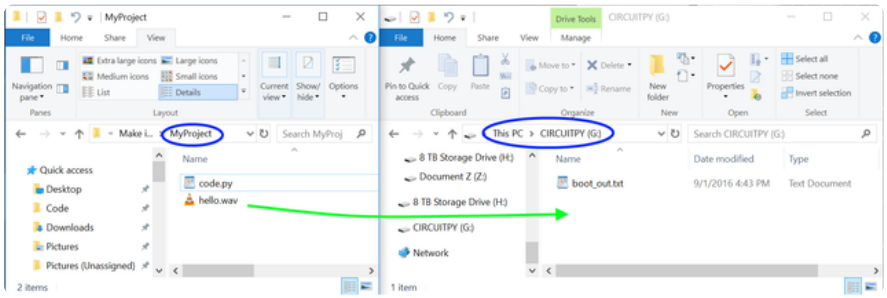
```
irq = digitalio.DigitalInOut(board.RFM9X_D0)
```

```
cs = digitalio.DigitalInOut(board.RFM9X_CS)
```

```
rst = digitalio.DigitalInOut(board.RFM9X_RST)
```

## Installing Code on CircuitPython

Navigate to where you saved your **code.py** file. Copy (by dragging the file and dropping it) the file to the **CIRCUITPY** drive.




While the code will appear to run, it is not yet set up for your Application or Device. We'll do that next.

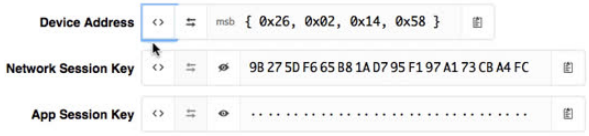
## Setting up the code for The Things Network

While we can send data to The Things Network, and our gateway might notice it, it isn't registered to a device yet. To register your device with The Things Network **using ABP**, you'll need to set three unique identifiers in the `code.py` file: the `Network Session Key`, the `Device Address`, and the `Application Session Key`.


Navigate to the **Device Overview** page for your Feather device.

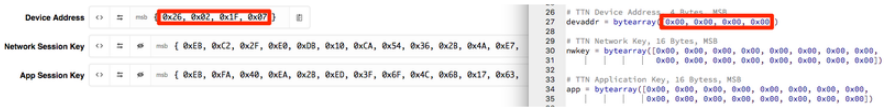
Make sure the **Activation Method** is set to **ABP**.

Before adding the unique identifiers to our sketch, we'll need to first expand them by **clicking the** <> icon.




These are your keys. We're going to enter them into our `code.py`, but we need to **be careful** - the keys on the Things Network console use **parentheses or curly braces { } instead of brackets [ ]**.

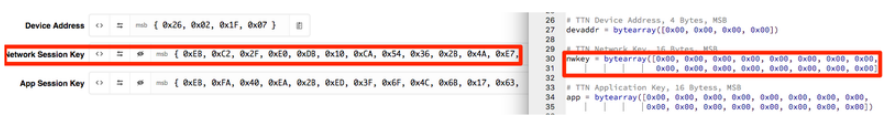
First, copy the **Device Address** from the TTN console to the `devaddr` variable in the code.



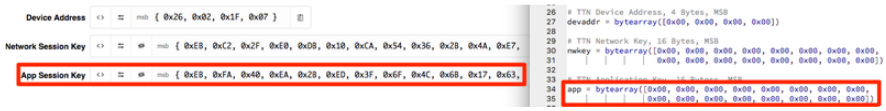

Then, **remove the braces { }** from the device address.

A device address copied from The Things Network console would look like: `{ 0x26, 0x02, 0x1F, 0x07 }`. In the code.py, it'd look like: `devaddr = bytearray([0x26, 0x02, 0x1F, 0x07])`.


Then, copy the **Network Session Key** from the TTN console to the `NwkSkey` variable in the code. Make sure to modify the code to remove the parentheses/curly braces { }.



Finally, copy the **Application Session Key** from the TTN console to the `AppSkey` variable in the code. Make sure to modify the code to remove the parentheses/curly braces { }.




That's all for now - we're ready to run our code!

## Running the Code using Mu Editor

Upon saving (**ctrl/cmd +s**) your code, the board will refresh.

But where is the output? The serial monitor is hidden by default in Mu Editor.

In the Mu Editor, click the **Serial** button on the top icon-bar to bring up the Serial REPL.


You should see the temperature and humidity values printed to the console. The packet's status (sending, or timed out and unable to send) should appear shortly after. The LED on your Feather should also blink the onboard LED when it successfully sends a packet to The Things Network.

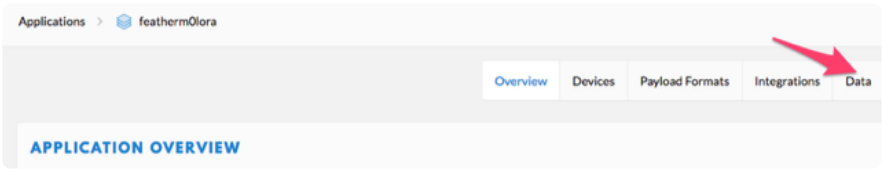
If you see Packet Sent!, rejoice - your packet has sent to the Things Network. But, while it's been sent, we want to check and make sure it's been received.

**If you're using a Raspberry Pi:** The Pi does not have an onboard LED located at pin D13, so it will not blink when the packet is sent. In the code, comment out every line which uses the LED, like the following:

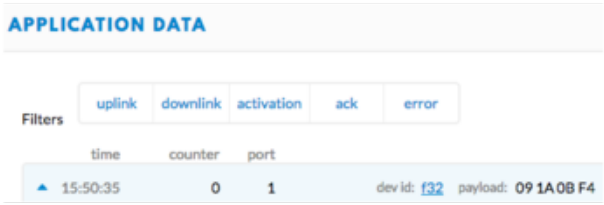
`led = digitalio.DigitalInOut(board.D13)` to `# led = digitalio.DigitalInOut(board.D13)`

## Checking Data on The Things Network Console

We want to make sure the data has been received on the other end. To do this, we'll navigate to the The Things Network Console and select the **application**. From the menu on the right hand side of the page, **Click Data**.




If everything worked correctly, you'll see the payload from your device streaming into the page, in real time.



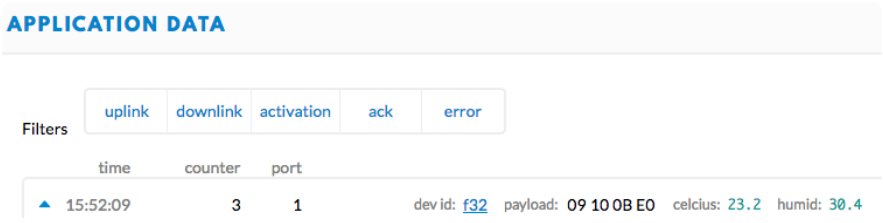

Neat, **right**? But while we received a payload, we still don't understand what it means. It's been sent to The Things Network and decoded on the client (Gateway) side, so it's not an AES-encrypted payload anymore. It's just not read-able by humans.

## Decoding the Payload

If you're sending packets in strange formats or encodings (like we are!), The Things Network Console has a programmable data decoder (https://adafru.it/CWk) to decode the packets, and assign useful labels to the data.

**Copy and paste the decoder script** below into the decoder's integrated text editor and **click save**.


Then, **click the data tab**. Next to the raw payload data, you should see the decoded data for humidity and temperature.




```
// TinyLoRa - Si7021 Decoder
function Decoder(bytes, port) {
  var decoded = {};

  // Decode bytes to int
  var celciusInt = (bytes[0] << 8) | bytes[1];
  var humidInt = (bytes[2] << 8) | bytes[3];

  // Decode int to float
  decoded.celcius = celciusInt / 100;
  decoded.humid = humidInt / 100;

  return decoded;
}
```

---

# TinyLoRa CircuitPython FAQ

## I'm located in a region other than the United States

All the software modifications you'd need to make are within one line of **code.py**. The country defaults to the **US** (adafruit is located in this region), but you can change it to any of the four supported regions:

- **US** - USA, Canada, and South America
- **EU** - Europe
- **AS** - Asia
- **AU** - Australia

If you wanted to switch to an European frequency plan, you'd configure the `ttn_config` object with the following parameters:

```
ttn_config = TTN(devaddr, nwkey, app, country = 'EU')
```

We've supported a subset of the major frequency plans in the initial release of TinyLoRa for CircuitPython. If you don't see your region (https://adafru.it/Dep) listed here, you can submit an issue (https://adafru.it/Deq) and we'll add it to the library.

## If you'd like to use a specific channel...

By default, TinyLoRa for CircuitPython broadcasts on multiple channels, randomly hopping between them. Since most gateways only listen for a transmission on a few channels at a time, this is a way to increase the probability of the gateway "seeing" the packet.

You can specify a channel by feeding the LoRa object constructor a channel as a keyword argument. For example, **if you'd like to send data on channel 3,** the object would be created like the following:

```
lora = TinyLoRa(spi, cs, irq, ttn_config, channel=3)
```

Want to change channels during the event loop, between sending different data packets? We included a `set_channel` function which will set up the radio to use the specified channel:

```
lora.set_channel(3)
```

## If you'd like to use a specific data rate...

By default, TinyLoRa uses the datarate **SF7BW125**. That is, a spreading factor of 7 and a bandwidth of 125kHz. Similar to setting a channel, TinyLoRa can set the datarate given a specific bandwidth and spreading factor. If you'd like to set the transmission datarate to a spreading factor of 10 and a bandwidth of 125kHz, call `set_datarate()` like the following:

```
lora.set_datarate("SF10BW125")
```

TinyLoRa has the following datarates available for use by this method:

- SF7BW125, SF7BW250, SF8BW125, SF9BW125, SF10BW125, SF11BW125, SF12BW125

Want to learn more about datarates and LoRa? Check out this blog post here.. (https://adafru.it/Der)

## My packet never successfully sends

If you're hitting an error, `RuntimeError: Timeout during packet send` , it's likely a gateway is not in range, or connected properly. Ensure that there are gateways near you and in-range (click here for a map of all the registered gateways (https://adafru.it/Des)).

If you're a Things Network Gateway operator, check from the Things Network Console if the gateway has been seen recently. If the Last Seen value is greater than a few minutes, reset your gateway.

## My project has disconnected/been reset and I don't see any data in my application

If you haven't disabled frame counter checks from within your device (instructions on how do this are on the TinyLoRa TTN Setup page (https://adafru.it/Det)), you'll encounter this issue when the device is reset/loses power.

Why would this happen? There's a mismatch between the frame counter on your device and the frame counter on The Things Network. Most likely, the frame counter on your device is reset to zero, and the frame counter on your device's console is set to the last known value.

There are ways around this issue, such as using a different activation method such as OTAA (currently unsupported by TinyLoRa).

However, we can still reset both the node and the frame counter on the application.

To do this, From your Things Network Console, navigate to your device (**Applications->Application->Device**).


Click the **reset frame counters** button next to the **Frames up** label. This will reset the frame counter of the device (on The Things Network's Console) to zero.

We'll also need to reset the frame counter on our device to zero. Press the **RST** button on the board.

## What's the deal with the frame counter? Do I need it in my code?

Disabling frame counter checks allows you to transmit data to The Things Network without requiring a match between your device's frame counter and the console's frame counter.

If you're making a project and doing a lot of prototyping/iteration to the code, disabling these checks is okay as it'll let you reset the device and not re-register it to the application (it'll also prevent counter overflows).

If you're deploying a project, you'll want to re-activate the frame counter for security purposes. With the frame counter disabled, one could re-transmit the messages sent to TTN using a replay attack.

The Things Network's documentation page has a full explanation about the role of the Frame Counter (https://adafru.it/Deu).