

[Skip to main content](#)

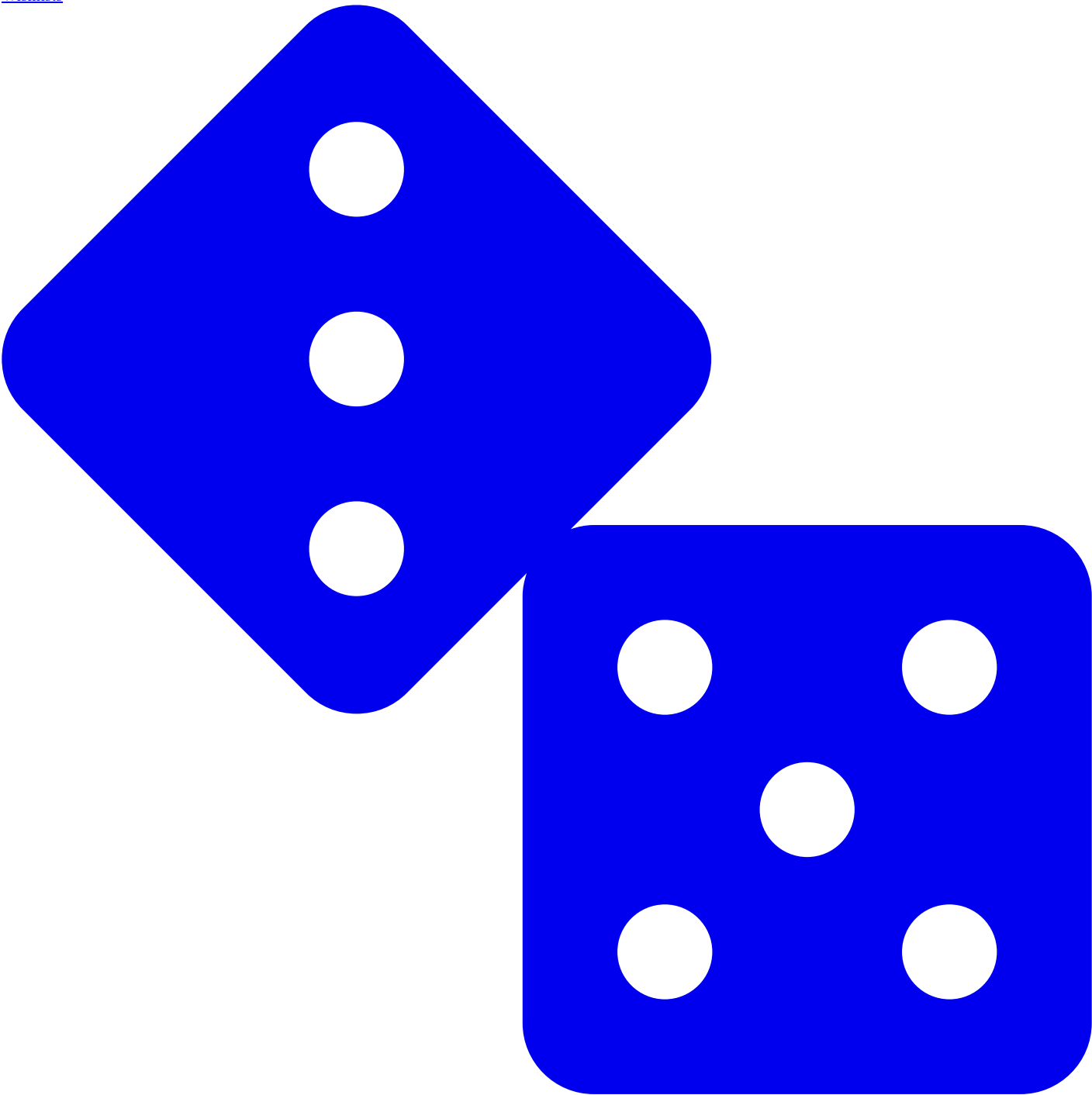
- [Shop](#)
- [Learn](#)
- [Blog](#)
- [Forums](#)
- [LIVE!](#)
- [AdaBox](#)
- [IO](#)



toggle menu

0

- [Sign In](#) | [Create Account](#)
- [New Guides](#)
- [Series](#)
- [Wishlists](#)



.

- [Shop](#)
- [Learn](#)
- [Blog](#)
- [Forums](#)
- [LIVE!](#)
- [AdaBox](#)
- [IO](#)

[Sign In](#)  
0



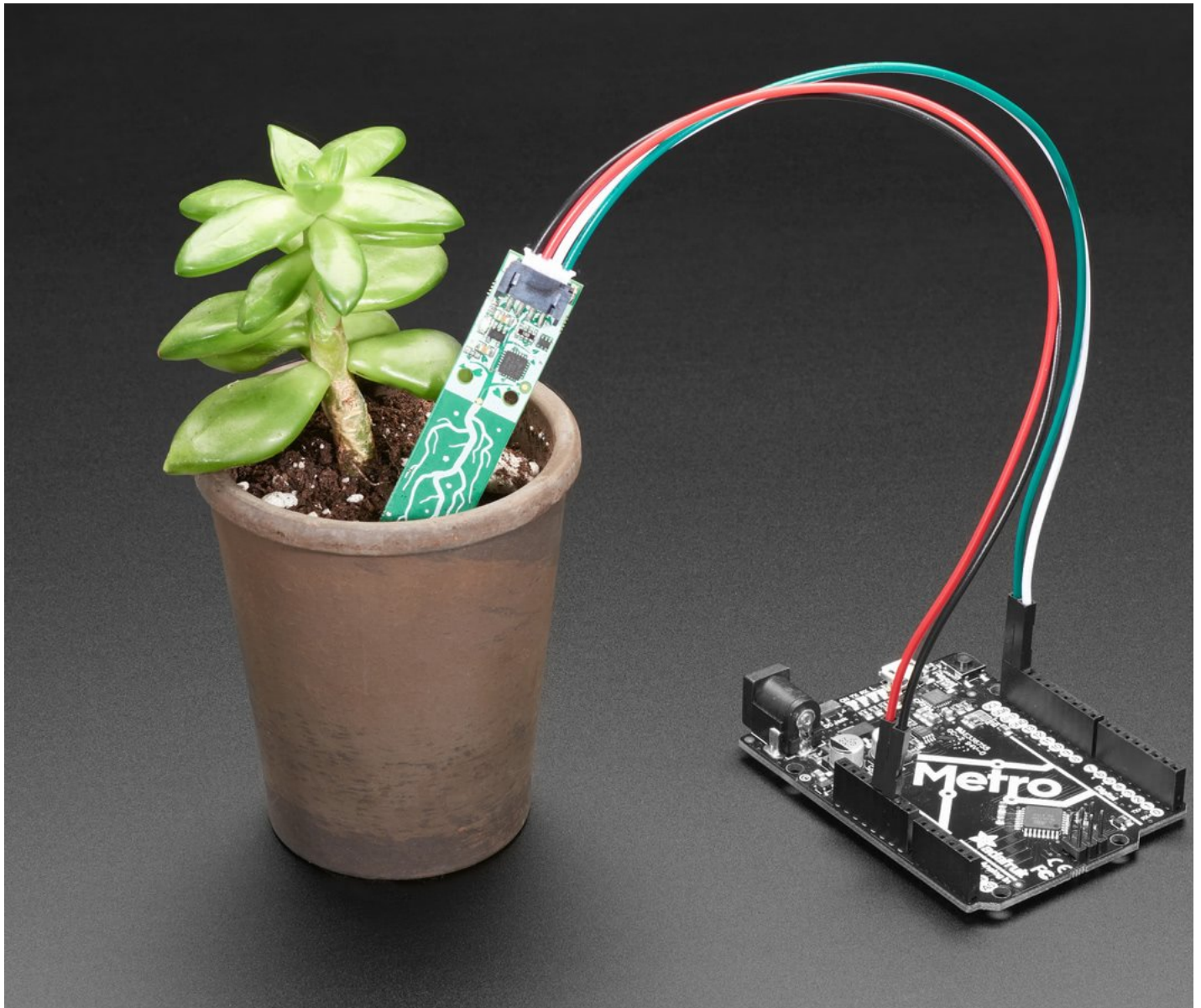
- [Explore & Learn](#)

Learn Categories[view all](#)

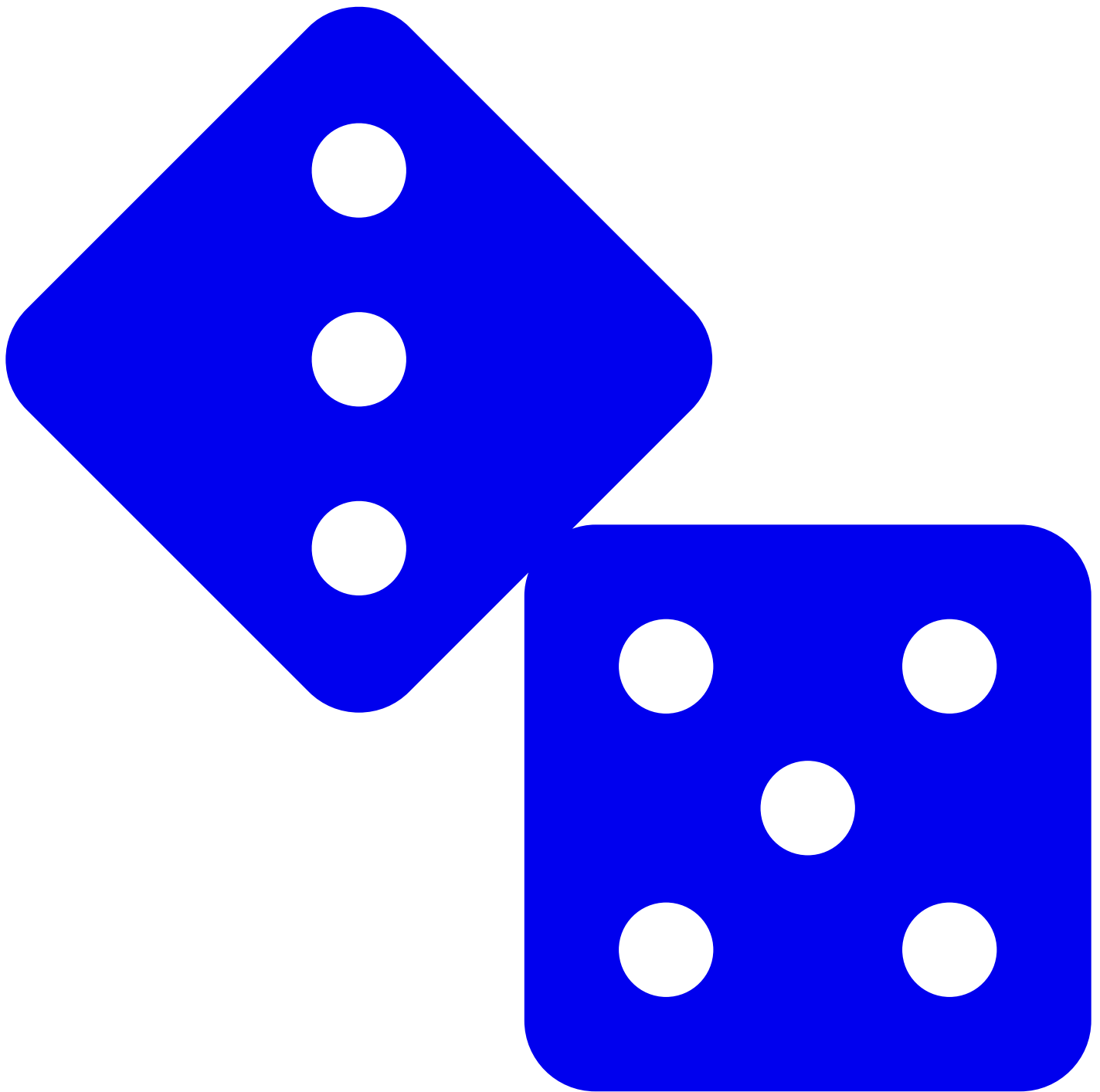
- [3D Printing](#)
- [AdaBox](#)
- [Adafruit Products](#)
- [Arduino Compatibles](#)
- [Breakout Boards](#)
- [Circuit Playground](#)
- [CircuitPython](#)
- [CLUE](#)
- [Community Support](#)
- [Components](#)
- [Crickit](#)
- [Customer & Partner Projects](#)
- [Development Boards](#)
- [Educators](#)
- [EL Wire/Tape/Panel](#)
- [Feather](#)
- [Gaming](#)
- [Hacks](#)
- [Internet of Things - IOT](#)
- [LCDs & Displays](#)
- [LEDs](#)
- [Machine Learning](#)
- [MakeCode](#)
- [Maker Business](#)
- [micro:bit](#)
- [Microcontrollers](#)
- [Programming](#)
- [Projects](#)
- [Raspberry Pi](#)
- [Robotics & CNC](#)
- [Sensors](#)
- [STEMMA](#)
- [Tools](#)
- [Trellis](#)
- [Wearables](#)

Series[view all](#)

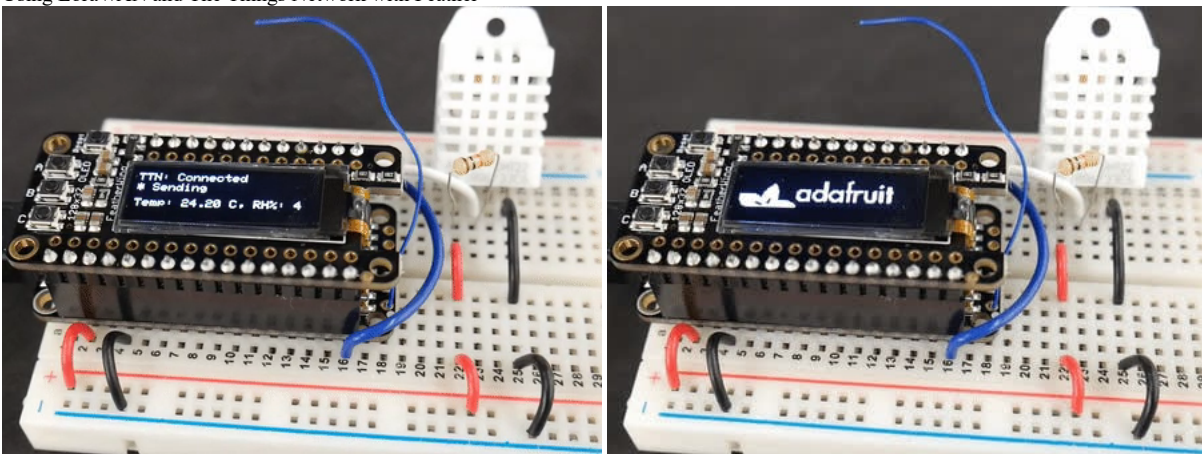
- [Circuit Playground](#)
- [Adafruit IO Basics](#)
- [Collin's Lab](#)



STEMMA  
Plug-n-play components  
[Get connected](#)  
• [New Guides](#)



Using LoRaWAN and The Things Network with Feather





## Using LoRaWAN and The Things Network with Feather

By [Brent Rubell](#)

Get connected to The Things Network by building a small weather node using a Feather M0 LoRa

- [Overview](#)
- [Arduino Wiring](#)
- [Registering a Feather with The Things Network](#)
- [Arduino Setup](#)
  - [Region Configuration](#)
- [Arduino Code](#)
- [Payload Decoding](#)
- [Add an OLED](#)
- [Using a Feather 32u4](#)
  - [TinyLoRa TTN Setup](#)
  - [TinyLoRa Arduino Setup](#)
  - [TinyLoRa Usage](#)
- [Featured Products](#)
- [Multiple pages](#)
- [Download PDF](#)

[Feedback? Corrections?](#)

## Overview

[Save](#) [Subscribe](#)

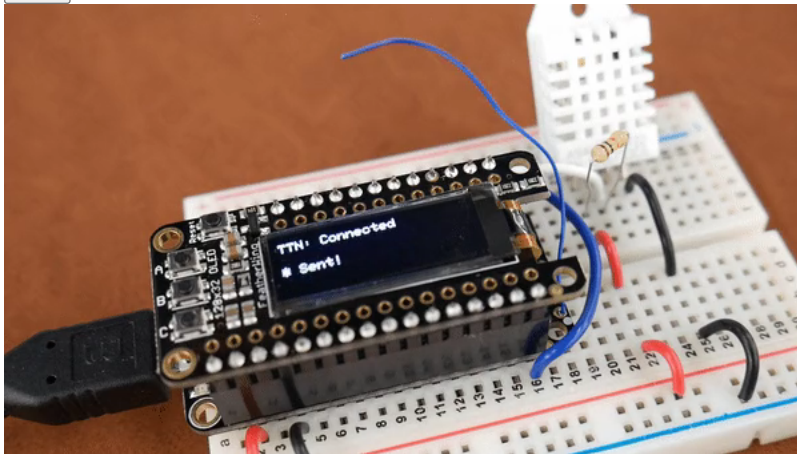


### New Subscription

Please [sign in](#) to subscribe to this guide.

You will be redirected back to this guide once you [sign in](#), and can then subscribe to this guide.

Close



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 to The Things Network.

## What is The Things Network?

The Things Network



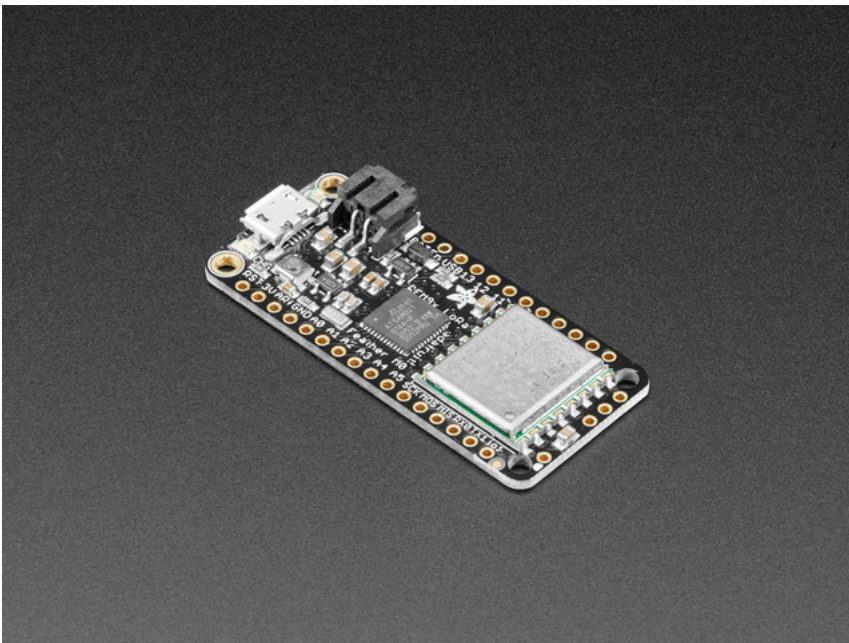
The [Things Network](#) is a project dedicated to building a [network](#) for the Internet of Things. While WiFi is used in most Internet of Things devices, The Things Network uses a protocol called **LoRaWAN** which allows devices to talk to the internet *without cellular or WiFi connectivity*. This 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**.

Also, there are plenty of gateways available to connect your feather to - [if you'd like to find a gateway in your area, check the Gateway Map](#).

## Parts

We're going to use one of our RadioFruits, the Feather M0 LoRa. In The The Things Network terms, this will be used as our [device](#).



### [Adafruit Feather M0 with RFM95 LoRa Radio - 900MHz](#)

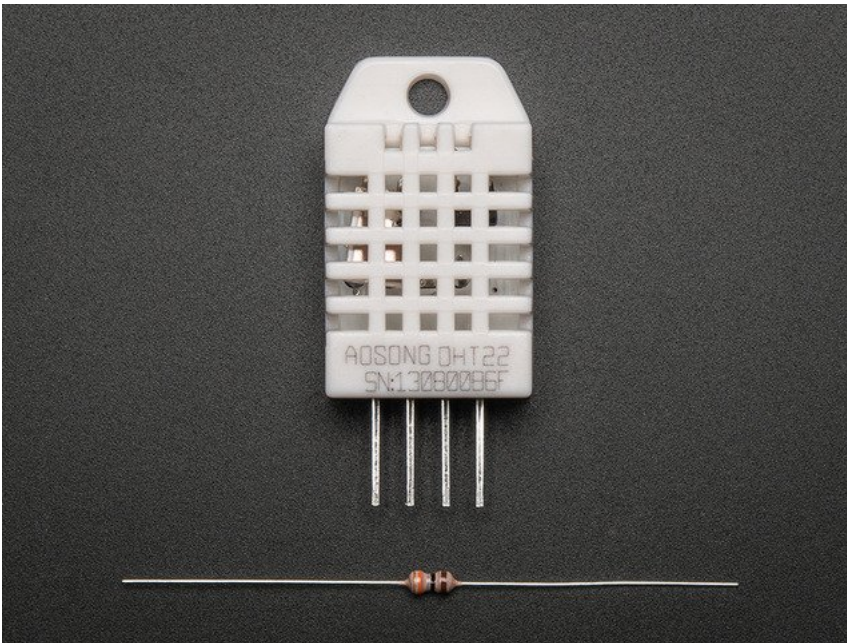
This is the Adafruit Feather M0 RFM95 LoRa Radio (900MHz). We call these RadioFruits, our take on an microcontroller with a...

\$34.95

In Stock

[Add to Cart](#)

We'll also want to *send data from our device to a* [gateway](#). This small sensor can read both the temperature and relative humidity. It's perfect for building small IoT data-loggers.



#### [DHT22 temperature-humidity sensor + extras](#)

The DHT22 is a basic, low-cost digital temperature and humidity sensor. It uses a capacitive humidity sensor and a thermistor to measure the surrounding air and spits out a digital...

\$9.95

In Stock

[Add to Cart](#)

#### Materials

1 x [Breadboard](#)

Half-size breadboard

[Add to Cart](#)

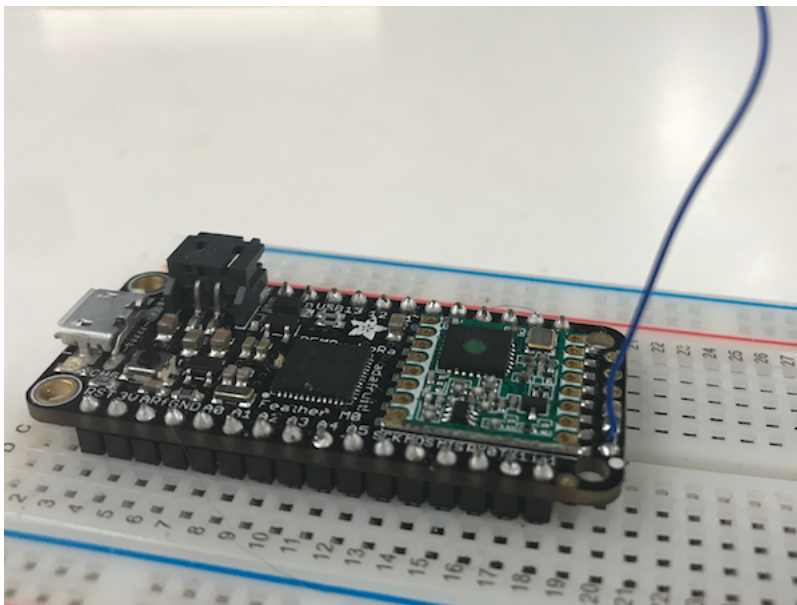
1 x [Breadboard Wires](#)

Breadboard Wiring Bundle

[Add to Cart](#)

### Arduino Wiring

#### Wiring the Antenna

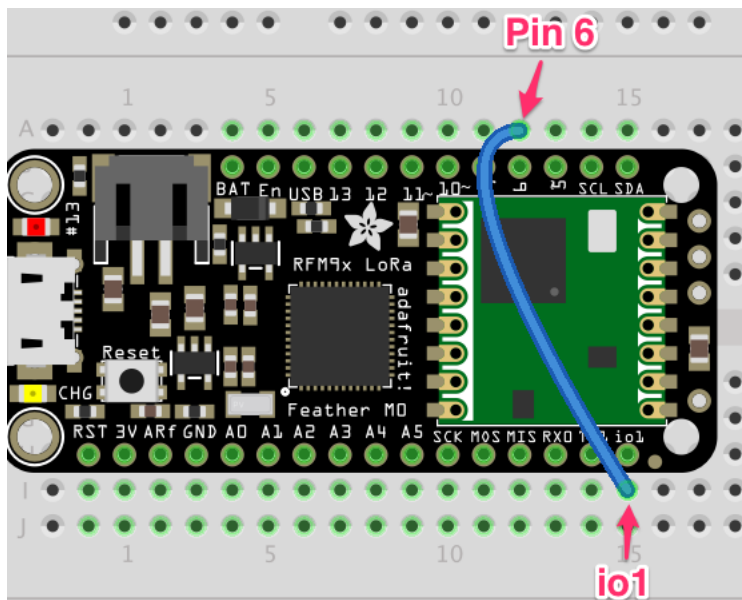


Your *Feather M0* does not come with an antenna, but there are two ways of wiring one up.

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](#).
- **Note:** Antenna length differs between regions, **make sure you cut the antenna to the correct length for your region.**

**Add a required jumper wire** - Pin 3 is internally connected to pin *io0*, but we'll need to **add a jumper wire between the Feather Digital Pin 6 and Feather Pin *io1*.**



## Wiring

Make the following connections between the Feather M0 and the DHT22:

- Feather 3V to DHT22 Pin 1
- Feather Pin 10 to DHT22 Pin 2
- Feather Ground to DHT22 Pin 3
- Add a 10k ohm resistor between the VCC (DHT Pin 1) and Data Pin (DHT Pin 2) on the DHT22.

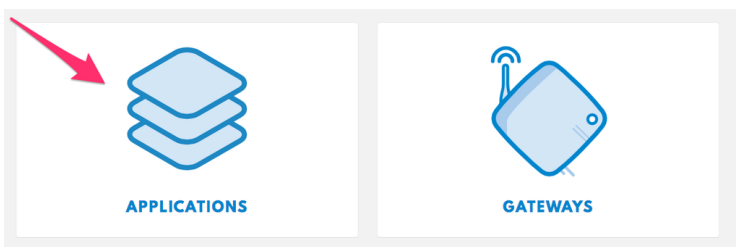
Make sure there is a wire connected between Pin 6 and the pin labeled 'io1'. Your sketch will not work properly if these are not connected.

## Registering a Feather with The Things Network

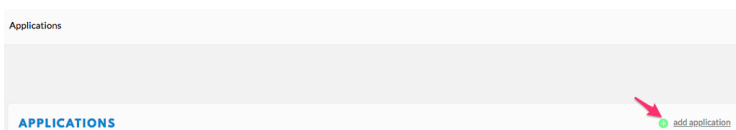
Before your Feather can communicate with The Things Network, you'll need to create an application.

First, we're going to register an account with TTN. [Navigate to their account registration page](#) to set up an account.

Once logged in, [navigate to the The Things Network Console](#). 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 by, 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.

**ADD APPLICATION**

**Application ID**  
The unique identifier of your application on the network.  
featherweather

**Description**  
A human readable description of your new app.  
feather weather station

**Application EUI**  
An application EUI will be issued for The Things Network block for convenience, you can add your own in the application settings page.  
EUI issued by The Things Network

**Handler registration**  
Select the handler you want to register this application to.  
ttn-handler-us-west

Cancel Add application

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

Applications > featherweather

Overview Devices Payload Formats Integrations Data Settings

**APPLICATION OVERVIEW**

Application ID featherweather  
Description feather weather station  
Created 58 seconds ago  
Handler ttn-handler-us-west

**APPLICATION EUIS**

70 83 05 7E D0 01 27 98

**DEVICES**

register device manage devices

0 registered devices

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

Applications > featherweather > Devices

Overview Devices Payload Formats Integrations Data Settings

**REGISTER DEVICE**

Device ID  
This is the unique identifier for the device in this app. The device ID will be immutable.  
featherm0

Device EUI  
The device EUI is the unique identifier for this device on the network. You can change the EUI later.  
98 76 B6 00 00 10 59 8C

App Key  
The App Key will be used to secure the communication between you device and the network.  
this field will be generated

App EUI  
70 83 05 7E D0 01 27 98

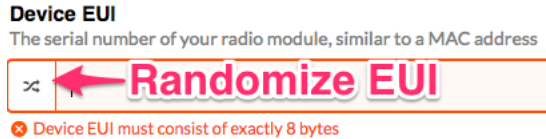
Cancel Register

Now that the application is set up, and the device is registered to the application, let's move on to wiring the Feather for use with TTN.

### The Sticker (EUI) which came with my feather is missing! *What now?*



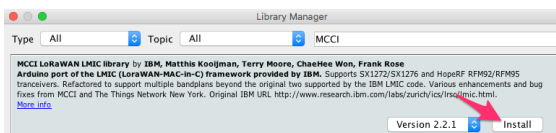
The Things Network's **Device EUI** can generate a device EUI. It also makes sure the EUI generated conforms to the IEEE spec.



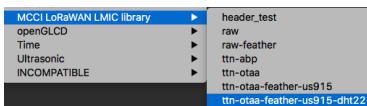
## Arduino Setup

Before proceeding with this guide, make sure your board is correctly set-up by following the [Using the Arduino IDE](#) page of the setup guide for the Feather M0 LoRa.

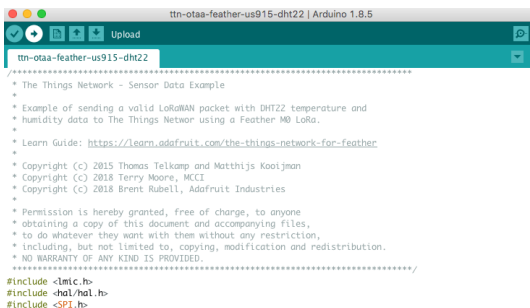
We're going to use MCCI's [arduino-lmic](#) library with this guide to communicate with The Things Network. **Open the Arduino Library Manager (Sketch->Include Library->Manage Libraries). In the search box, type MCCI. the MCCI LoRaWAN LMIC library should be the first result. Click Install.**



We've included the code for this guide inside the arduino-lmic library. To open it, from the Arduino IDE click **File->Examples->MCCI LoRaWAN LMIC library->ttn-otaa-feather-us915-dht22**



The code for the Feather M0 Weather Node should pop up.



Now that our code is set up, let's learn how to send data from our Feather to TTN and decode it!

## Region Configuration

If you're within the United States, you can skip this page. The code and setup is already targeted towards the US frequency ranges.

Dont fret if you're not in the US - you'll just have to make some small modifications to your code to ensure it runs on the correct region. The Arduino-Lmic library supports the following regions:

- eu\_868: EU
- us\_915: US
- au\_921: Australia
- as\_923: Asia
- in\_866: India

First, navigate to the [lmic\\_project\\_config.h](#) file within the LMIC Arduino library. On Windows and Macintosh machines, the default name of the **folder** is "**Arduino**" and is **located** in your Documents **folder**.



```
lmic_project_config.h | Arduino
lmic_project_config.h
1// project-specific definitions
2//define CFG_eu868 1
3#define CFG_us915 1
4//define CFG_us921 1
5//define CFG_us923 1
6// #define LMIC_COUNTRY_CODE LMIC_COUNTRY_CODE_JP /* for as923-JP */
7//define CFG_in866 1
8#define CFG_sx1276_radio 1
9//define LMIC_USE_INTERRUPTS
```

Simply uncomment the region your device is located in. We use a SX1276 transceiver on the Feather M0 LoRA, so this definition should not be changed.

## Arduino Code

### Remove regression test code

If you see this chunk of code, you can remove it from your code

```
//
// For normal use, we require that you edit the sketch to replace FILLMEIN
// with values assigned by the TTN console. However, for regression tests,
// we want to be able to compile these scripts. The regression tests define
// COMPILER_REGRESSION_TEST, and in that case we define FILLMEIN to a non-
// working but innocuous value.
//
#ifdef COMPILER_REGRESSION_TEST
#define FILLMEIN 0
#else
#warning "You must replace the values marked FILLMEIN with real values from the TTN control panel!"
#define FILLMEIN (#dont edit this, edit the lines that use FILLMEIN)
#endif
```

## Code Setup

To register your Feather with The Things Network, you need to set three unique identifiers in the code: APPEUI, DEVEUI, and APPKEY.

DEVICE OVERVIEW

Application ID

featherm0lora

Device ID

featherm0loratest

Activation Method

OTAA

Device EUI

<> ⇄ 98 76 B6 00 00 10 DD D6

Application EUI

<> ⇄ 70 B3 D5 7E D0 01 24 EF

App Key

<> ⇄ .....

Navigate to the **Device Overview** page for your Feather device. Make sure the **Activation Method** is set to **OTAA**.

Before adding the unique identifiers to our sketch, we'll need to first expand them by **clicking the <> icon**. Then, we'll need to switch the order of the Device EUI and Application EUI to little-endian format. You can swap the order by **clicking the button with two arrows**.

Device EUI

<> ⇄ msb { 0x98, 0x76, 0xB6, 0x00, 0x00, 0x10, 0xDD, 0xD6 }

Application EUI

⇄ 70 B3 D5 7E D0 01 24 EF

App Key

<> ⇄ BE DE CE 25 06 38 D1 4D 8E 73 E9 E5 61 9F CC C3

We're going to copy each unique identifier from the Device Overview to the variables within the sketch.

First, copy the **Application EUI** in **lsb** format from the TTN console to **APPEUI** variable in the sketch.

DEVICE OVERVIEW

Application ID

featherm0lora

Device ID

featherm0

Activation Method

OTAA

Device EUI

⇄ { 0xB6, 0x76, 0xB3, 0x00, 0x00, 0x10, 0xDD, 0xD6 }

Application EUI

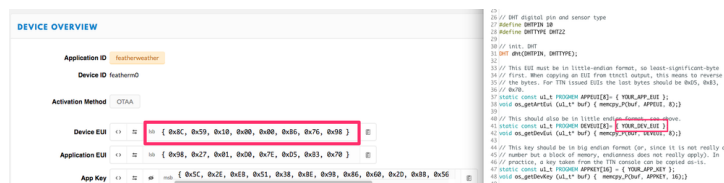
⇄ { 0xB3, 0xD5, 0x7E, 0xD0, 0x01, 0x24, 0xEF }

App Key

⇄ { 0xBE, 0xDE, 0xCE, 0x25, 0x06, 0x38, 0xD1, 0x4D, 0x8E, 0x73, 0xE9, 0xE5, 0x61, 0x9F, 0xCC, 0xC3 }

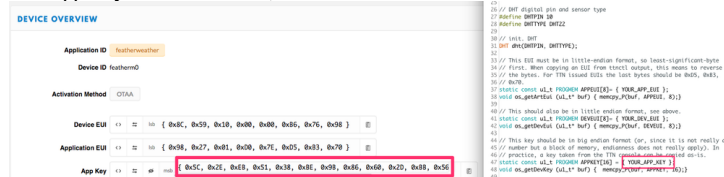
```
10 // DT digital pin and sensor type
11 #define DT_PIN 10
12 #define DT_TYPE DT_TEMP
13 // DT pin config
14 #define DT_PIN_CONFIG {DT_PIN, DT_TYPE}
15 // This EUI must be in little-endian format, so least-significant-byte
16 // first. When copying an EUI from trace output, this means to reverse
17 // the bytes. For TTN issued EUIs the last bytes should be 0x00, 0x00,
18 // 0x00, 0x00.
19 static const u1_t PROGMEM APPEUI = { 0xB3, 0xD5, 0x7E, 0xD0, 0x01, 0x24, 0xEF };
20 void memcpyEUI(u1_t* buf) { memcpy(buf, APPEUI, 8); }
21 // This should also be in little-endian format, see above.
22 static const u1_t PROGMEM DEVEUI = { 0xB6, 0x76, 0xB3, 0x00, 0x00, 0x10, 0xDD, 0xD6 };
23 void memcpyDEVEUI(u1_t* buf) { memcpy(buf, DEVEUI, 8); }
24 // This key should be in big-endian format (or, since it is not really a
25 // matter but a block of memory, endianness does not really apply). In
26 // practice, a key value from the TTN console can be copied as-is.
27 static const u1_t PROGMEM APPKEY = { 0xBE, 0xDE, 0xCE, 0x25, 0x06, 0x38, 0xD1, 0x4D, 0x8E, 0x73, 0xE9, 0xE5, 0x61, 0x9F, 0xCC, 0xC3 };
28 void memcpyAPPKEY(u1_t* buf) { memcpy(buf, APPKEY, 16); }
```

Next, copy the **Device EUI** in **lsb** format from the TTN console to the **DEVEUI** variable in the sketch.



Finally, copy the **App Key** from the TTN console to **APPKEY** variable in the sketch.

The App Key is in msb format, unlike the EUI's above!!!



That's all for now - we're ready to upload the code onto our Arduino.

Note that to see the output you'll need to open up the Serial console. Once you've debugged your example, you can comment out or remove the `while (! Serial);` line from `setup()`.

## Code Overview

Most of what happens in the code occurs in the `setup()`, `on_event()` and `do_send()` functions.

Within `setup()`, we initialize our DHT sensor and set up the LMIC (LoRaWAN-in-C, formerly LoRaMAC-in-C) framework for use with our radio (the RFM95) and region-specific radio settings.

Our main `loop()` calls `os_runloop_once()`, which calls the LMIC runloop processor. This loop causes radio events to occur based on events and time - such as callbacks for when the transmission is complete. While you *can* place additional code within this `loop()` routine, we don't advise it - the LoRaWAN timing is tight.

If there isn't a transmission job currently running, we're going to prepare the packet to send to The Things Network. This occurs in the `do_send()` function.

To read the temperature, we're going to first create a floating point variable, `temperature`, and assign it to the temperature value from the sensor.

```
float temperature = dht.readTemperature();
```

While we can't directly send floating point numbers (like 50.62 degrees) to The Things Network dashboard, the MCCI-LMIC library includes some data encoding utility functions to encode floating point data into integer data using a special bit layout - [sflt16](#).

Since the floating point is within the range -1 to 1, we'll need to divide the range (by 100 for temperature) to increase the exponent range. We're going to get the value back in the range by later multiplying it by the range (100, in this example) using the decoder on the TTN Console.

```
// adjust for the f2sflt16 range (-1 to 1)
temperature = temperature / 100; rHumidity = rHumidity / 100;
```

Next, we're going to convert the float to an integer. To do this, we'll use the library's float-to-signed-float-16 function (`LMIC_f2sflt16`):

```
// float -> int
uint16_t payloadTemp = LMIC_f2sflt16(temperature);
```

Almost there! Before loading our values into the payload, we'll need to convert them from an integer (`payloadTemp`) to bytes. To do this, we'll use the Arduino [lowByte](#) and [highByte](#) functions:

```
// int -> bytes
byte tempLow = lowByte(payloadTemp);
byte tempHigh = highByte(payloadTemp);
```

We defined a payload further up in the code (`static uint8_t payload[5]`). Now, we place the bits (low byte first) into the payload:

```
payload[0] = tempLow;
payload[1] = tempHigh;
```

and prepare the payload for sending to TTN at the next possible time.

```
LMIC_setTxData2(1, payload, sizeof(payload)-1, 0);
```

## Check Sketch Output

Compile (**cmd/ctrl + R**) and Upload (**ctrl/cmd+U**) the sketch onto the Feather. Then, pop open the serial monitor (**Tools->Serial Monitor**). You should see the Feather joining the Things Network. Once it joins, it'll dump its connection info. Then, it'll show *EV\_TXStart* (an event which begins the transmission) and *EV\_TXComplete* (the transmission has been received and acknowledged by the gateway). The output should be similar to the following:

[Download File](#)  
[Copy Code](#)

```

Starting
Temperature: 26.00 *C
%RH 48.10
105016549: EV_JOINING
105016641: EV_TXSTART
105357886: EV_JOINED
netid: 19
devaddr: 26022F78
artKey: 78-F6-78-6C-87-26-86-AE-E1-AC-6D-79-83-57-7E-11
nwkkKey: FE-14-C4-A7-BF-D3-B6-E6-95-D4-2F-93-DC-F9-D7-25
105358155: EV_TXSTART
105579674: EV_TXCOMPLETE (includes waiting for RX windows)

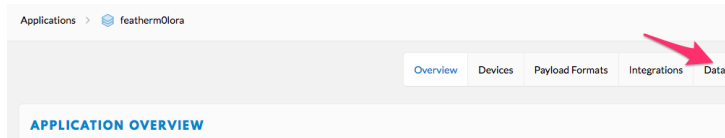
```

If you're stuck on *EV\_JOINING* or fail to join the network, make sure your device is within range of a The Things Network gateway.

If the code is looping *EV\_TXSTART*, make sure a jumper wire is connecting the Feather's Pin 6 and Feather Pin 'io1'.

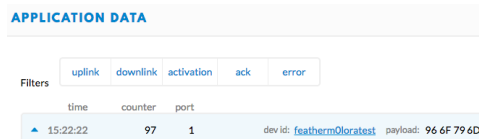
## Check Things Network Data

Navigate to the The Things Network Console and select your 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...

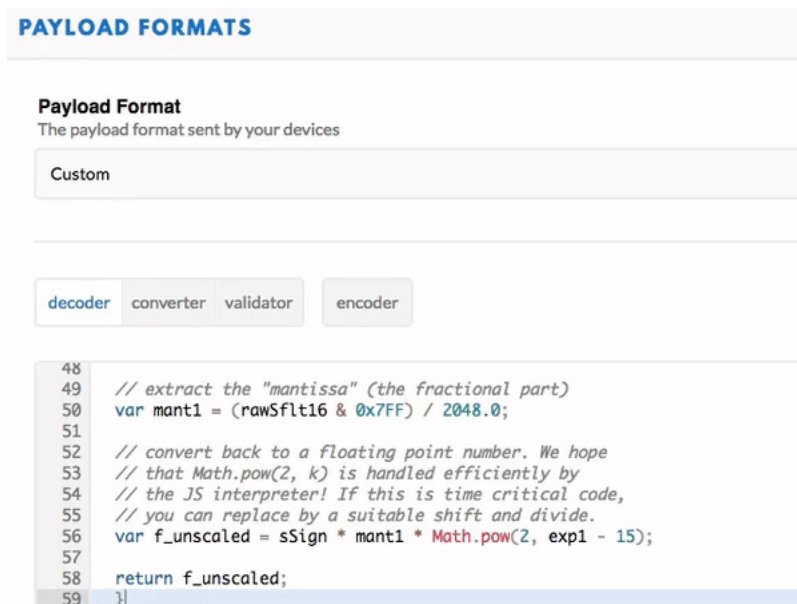


## Payload Decoding

### 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](#) 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.

time	counter	port	
02:35:51	0		
02:35:50	21	1	payload: D9 6E 02 75 degreesC: 21.39892578125 humidity: 31.298828125
02:35:31	0		
02:35:30	20	1	payload: E1 6E F6 74 degreesC: 21.49658283125 humidity: 31.005859375

## Decoder Code

[Download Project Bundle](#)

[Copy Code](#)

```
// TTN Decoder for TTN OTAA Feather US915 DHT22 Sketch
// Link: https://github.com/mcci-catena/arduino-lmic/blob/master/examples/ttn-otaa-feather-us915-dht22/ttn-otaa-feather-us915-dht22.ino
function Decoder(bytes, port) {
  // Decode an uplink message from a buffer
  // (array) of bytes to an object of fields.
  var decoded = {};

  // temperature

  rawTemp = bytes[0] + bytes[1] * 256;

  decoded.degreesC = sflt162f(rawTemp) * 100;

  // humidity
  rawHumid = bytes[2] + bytes[3] * 256;
  decoded.humidity = sflt162f(rawHumid) * 100;

  return decoded;
}

function sflt162f(rawSflt16)
{
  // rawSflt16 is the 2-byte number decoded from wherever;
  // it's in range 0..0xFFFF
  // bit 15 is the sign bit
  // bits 14..11 are the exponent
  // bits 10..0 are the mantissa. Unlike IEEE format,
  // the msb is transmitted; this means that numbers
  // might not be normalized, but makes coding for
  // underflow easier.
  // As with IEEE format, negative zero is possible, so
  // we special-case that in hopes that JavaScript will
  // also cooperate.
  //
  // The result is a number in the open interval (-1.0, 1.0);
  //

  // throw away high bits for repeatability.
  rawSflt16 &= 0xFFFF;

  // special case minus zero:
  if (rawSflt16 == 0x8000)
    return -0.0;

  // extract the sign.
  var sSign = ((rawSflt16 & 0x8000) != 0) ? -1 : 1;

  // extract the exponent
  var exp1 = (rawSflt16 >> 11) & 0xF;

  // extract the "mantissa" (the fractional part)
  var mant1 = (rawSflt16 & 0x7FF) / 2048.0;

  // convert back to a floating point number. We hope
  // that Math.pow(2, k) is handled efficiently by
  // the JS interpreter! If this is time critical code,
  // you can replace by a suitable shift and divide.
  var f_unscaled = sSign * mant1 * Math.pow(2, exp1 - 15);

  return f_unscaled;
}
```

[View on GitHub](#)

## Code

[Download Project Bundle](#)

[Copy Code](#)

```
/*
 * The Things Network - Sensor Data Example
 *
 * Example of sending a valid LoRaWAN packet with DHT22 temperature and
 * humidity data to The Things Network using a Feather M0 LoRa.
 *
 * Learn Guide: https://learn.adafruit.com/the-things-network-for-feather
 *
 * Copyright (c) 2015 Thomas Telkamp and Matthijs Kooijman
 * Copyright (c) 2018 Terry Moore, MCCI
 * Copyright (c) 2018 Brent Rubell, Adafruit Industries
 *
 * Permission is hereby granted, free of charge, to anyone
 * obtaining a copy of this document and accompanying files,
 * to do whatever they want with them without any restriction,
 * including, but not limited to, copying, modification and redistribution.
 * NO WARRANTY OF ANY KIND IS PROVIDED.
 */
#include <lmic.h>
#include <hal/hal.h>
#include <SPI.h>
```

```

// include the DHT22 Sensor Library
#include "DHT.h"

// DHT digital pin and sensor type
#define DHTPIN 10
#define DHTTYPE DHT22

// This EUI must be in little-endian format, so least-significant-byte
// first. When copying an EUI from ttnctl output, this means to reverse
// the bytes. For TTN issued EUIs the last bytes should be 0xD5, 0xB3,
// 0x70.
static const u1_t PROGMEM APPEUI[8] = { FILLMEIN };
void os_getArtEui (u1_t* buf) { memcpy_P(buf, APPEUI, 8);}

// This should also be in little endian format, see above.
static const u1_t PROGMEM DEVEUI[8] = { FILLMEIN };
void os_getDevEui (u1_t* buf) { memcpy_P(buf, DEVEUI, 8);}

// This key should be in big endian format (or, since it is not really a
// number but a block of memory, endianness does not really apply). In
// practice, a key taken from the TTN console can be copied as-is.
static const u1_t PROGMEM APPKEY[16] = { FILLMEIN };
void os_getDevKey (u1_t* buf) { memcpy_P(buf, APPKEY, 16);}

// payload to send to TTN gateway
static uint8_t payload[5];
static osjob_t sendjob;

// Schedule TX every this many seconds (might become longer due to duty
// cycle limitations).
const unsigned TX_INTERVAL = 30;

// Pin mapping for Adafruit Feather M0 LoRa
const lmhc_pinmap lmhc_pins = {
  .nss = 8,
  .rxtx = LMIC_UNUSED_PIN,
  .rst = 4,
  .dio = {3, 6, LMIC_UNUSED_PIN},
  .rxtx_rx_active = 0,
  .rssi_cal = 8,           // LBT cal for the Adafruit Feather M0 LoRa, in dB
  .spi_freq = 8000000,
};

// init. DHT
DHT dht(DHTPIN, DHTTYPE);

void onEvent (ev_t ev) {
  Serial.print(os_getTime());
  Serial.print(": ");
  switch(ev) {
    case EV_SCAN_TIMEOUT:
      Serial.println(F("EV_SCAN_TIMEOUT"));
      break;
    case EV_BEACON_FOUND:
      Serial.println(F("EV_BEACON_FOUND"));
      break;
    case EV_BEACON_MISSED:
      Serial.println(F("EV_BEACON_MISSED"));
      break;
    case EV_BEACON_TRACKED:
      Serial.println(F("EV_BEACON_TRACKED"));
      break;
    case EV_JOINING:
      Serial.println(F("EV_JOINING"));
      break;
    case EV_JOINED:
      Serial.println(F("EV_JOINED"));
      {
        u4_t netid = 0;
        devaddr_t devaddr = 0;
        u1_t nwkey[16];
        u1_t artkey[16];
        LMIC_getSessionKeys(&netid, &devaddr, nwkey, artkey);
        Serial.print("netid: ");
        Serial.println(netid, DEC);
        Serial.print("devaddr: ");
        Serial.println(devaddr, HEX);
        Serial.print("artKey: ");
        for (int i=0; i<sizeof(artkey); ++i) {
          if (i != 0)
            Serial.print("-");
          Serial.print(artkey[i], HEX);
        }
        Serial.println("");
        Serial.print("nwkey: ");
        for (int i=0; i<sizeof(nwkey); ++i) {
          if (i != 0)
            Serial.print("-");
          Serial.print(nwkey[i], HEX);
        }
        Serial.println("");
      }
      // Disable link check validation (automatically enabled
      // during join, but because slow data rates change max TX
      // size, we don't use it in this example.
      LMIC_setLinkCheckMode(0);
      break;
  }
}

```

```

/*
|| This event is defined but not used in the code. No
|| point in wasting codespace on it.
||
|| case EV_RFU1:
||     Serial.println(F("EV_RFU1"));
||     break;
||
*/
case EV_JOIN_FAILED:
    Serial.println(F("EV_JOIN_FAILED"));
    break;
case EV_REJOIN_FAILED:
    Serial.println(F("EV_REJOIN_FAILED"));
    break;
    break;
case EV_TXCOMPLETE:
    Serial.println(F("EV_TXCOMPLETE (includes waiting for RX windows)"));
    digitalWrite(LED_BUILTIN, LOW);
    if (LMIC.txrxFlags & TXRX_ACK)
        Serial.println(F("Received ack"));
    if (LMIC.dataLen) {
        Serial.println(F("Received "));
        Serial.println(LMIC.dataLen);
        Serial.println(F(" bytes of payload"));
    }
    // Schedule next transmission
    os_setTimedCallback(&sendjob, os_getTime()+sec2osticks(TX_INTERVAL), do_send);
    break;
case EV_LOST_TSYNC:
    Serial.println(F("EV_LOST_TSYNC"));
    break;
case EV_RESET:
    Serial.println(F("EV_RESET"));
    break;
case EV_RXCOMPLETE:
    // data received in ping slot
    Serial.println(F("EV_RXCOMPLETE"));
    break;
case EV_LINK_DEAD:
    Serial.println(F("EV_LINK_DEAD"));
    break;
case EV_LINK_ALIVE:
    Serial.println(F("EV_LINK_ALIVE"));
    break;

/*
|| This event is defined but not used in the code. No
|| point in wasting codespace on it.
||
|| case EV_SCAN_FOUND:
||     Serial.println(F("EV_SCAN_FOUND"));
||     break;
||
*/
case EV_TXSTART:
    Serial.println(F("EV_TXSTART"));
    digitalWrite(LED_BUILTIN, HIGH);
    break;
default:
    Serial.print(F("Unknown event: "));
    Serial.println((unsigned) ev);
    break;
}
}

void do_send(osjob_t* j){
    // Check if there is not a current TX/RX job running
    if (LMIC.opmode & OP_TXRXPEND) {
        Serial.println(F("OP_TXRXPEND, not sending"));
    } else {
        // read the temperature from the DHT22
        float temperature = dht.readTemperature();
        Serial.print("Temperature: "); Serial.print(temperature);
        Serial.println(" *C");
        // adjust for the f2sflt16 range (-1 to 1)
        temperature = temperature / 100;

        // read the humidity from the DHT22
        float rHumidity = dht.readHumidity();
        Serial.print("%RH ");
        Serial.println(rHumidity);
        // adjust for the f2sflt16 range (-1 to 1)
        rHumidity = rHumidity / 100;

        // float -> int
        // note: this uses the sflt16 datum (https://github.com/mcci-catena/arduino-lmic#sflt16)
        uint16_t payloadTemp = LMIC_f2sflt16(temperature);
        // int -> bytes
        byte tempLow = lowByte(payloadTemp);
        byte tempHigh = highByte(payloadTemp);
        // place the bytes into the payload
        payload[0] = tempLow;
        payload[1] = tempHigh;

        // float -> int
        uint16_t payloadHumid = LMIC_f2sflt16(rHumidity);
        // int -> bytes
        byte humidLow = lowByte(payloadHumid);
        byte humidHigh = highByte(payloadHumid);
        payload[2] = humidLow;

```



```

    payload[3] = humidHigh;

    // prepare upstream data transmission at the next possible time.
    // transmit on port 1 (the first parameter); you can use any value from 1 to 223 (others are reserved).
    // don't request an ack (the last parameter, if not zero, requests an ack from the network).
    // Remember, acks consume a lot of network resources; don't ask for an ack unless you really need it.
    LMIC_setTxData2(1, payload, sizeof(payload)-1, 0);
  }
  // Next TX is scheduled after TX_COMPLETE event.
}

void setup() {
  delay(5000);
  while (! Serial);
  Serial.begin(9600);
  Serial.println(F("Starting"));

  pinMode(LED_BUILTIN, OUTPUT);
  // dht init.
  dht.begin();

  // LMIC init.
  os_init();
  // Reset the MAC state. Session and pending data transfers will be discarded.
  LMIC_reset();
  // Disable link-check mode and ADR, because ADR tends to complicate testing.
  LMIC_setLinkCheckMode(0);
  // Set the data rate to Spreading Factor 7. This is the fastest supported rate for 125 kHz channels, and it
  // minimizes air time and battery power. Set the transmission power to 14 dBi (25 mW).
  LMIC_setDrTxpow(DR_SF7,14);
  // in the US, with TTN, it saves join time if we start on subband 1 (channels 8-15). This will
  // get overridden after the join by parameters from the network. If working with other
  // networks or in other regions, this will need to be changed.
  LMIC_selectSubBand(1);

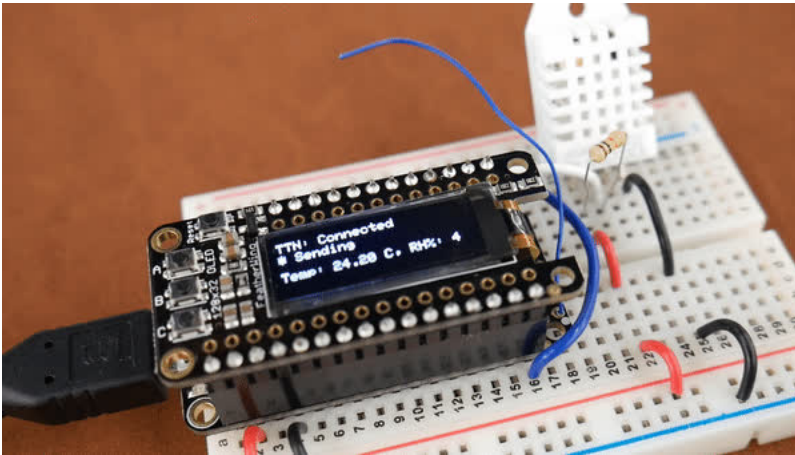
  // Start job (sending automatically starts OTAA too)
  do_send(&sendjob);
}

void loop() {
  // we call the LMIC's runloop processor. This will cause things to happen based on events and time. One
  // of the things that will happen is callbacks for transmission complete or received messages. We also
  // use this loop to queue periodic data transmissions. You can put other things here in the `loop()` routine,
  // but beware that LoRaWAN timing is pretty tight, so if you do more than a few milliseconds of work, you
  // will want to call `os_runloop_once()` every so often, to keep the radio running.
  os_runloop_once();
}

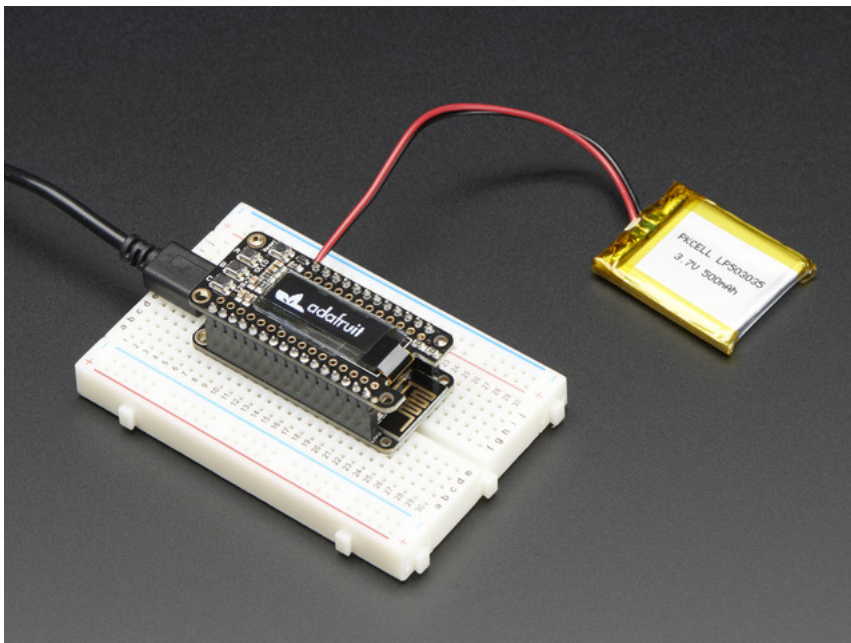
```

[View on GitHub](#)

## Add an OLED



Now that we've got our Feather M0 connecting to The Things Network, we'll add a FeatherWing OLED to view the active connection with The Things Network and the sensor measurements.



### [Adafruit FeatherWing OLED - 128x32 OLED Add-on For Feather](#)

A Feather board without ambition is a Feather board without FeatherWings! This is the FeatherWing OLED: it adds a 128x32 monochrome OLED plus 3 user buttons to...

\$14.95

In Stock

[Add to Cart](#)

Snap the FeatherWing onto your Feather, navigate to the FeatherWing OLED Arduino Setup guide, and [follow the instructions to set up and test the OLED](#). Once you verify that the FeatherWing OLED works, **copy and paste the code below into a new sketch**.

Make sure to re-enter your unique device identifiers before compiling and uploading the sketch.

## Code

[Download Project Bundle](#)

[Copy Code](#)

```

/*****
 * The Things Network - Sensor Data Example with OLED
 *
 * Example of sending a valid LoRaWAN packet with DHT22 temperature and
 * humidity data to The Things Network using a Feather M0 LoRa.
 *
 * Learn Guide: https://learn.adafruit.com/the-things-network-for-feather
 *
 * Copyright (c) 2015 Thomas Telkamp and Matthijs Kooijman
 * Copyright (c) 2018 Terry Moore, MCCI
 * Copyright (c) 2018 Brent Rubell, Adafruit Industries
 *
 * Permission is hereby granted, free of charge, to anyone
 * obtaining a copy of this document and accompanying files,
 * to do whatever they want with them without any restriction,
 * including, but not limited to, copying, modification and redistribution.
 * NO WARRANTY OF ANY KIND IS PROVIDED.
 *****/
#include <lmic.h>
#include <hal/hal.h>
#include <SPI.h>

// include the DHT22 Sensor Library
#include "DHT.h"

// include the FeatherWing OLED library
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>

// DHT digital pin and sensor type
#define DHTPIN 10
#define DHTTYPE DHT22

Adafruit_SSD1306 display = Adafruit_SSD1306();

#if (SSD1306_LCDHEIGHT != 32)
#error("Height incorrect, please fix Adafruit_SSD1306.h!");
#endif

//
// For normal use, we require that you edit the sketch to replace FILLMEIN
// with values assigned by the TTN console. However, for regression tests,
// we want to be able to compile these scripts. The regression tests define
// COMPILEREGRESSIONTEST, and in that case we define FILLMEIN to a non-
// working but innocuous value.

```

```

//
#ifdef COMPILER_REGRESSION_TEST
#define FILLMEIN 0
#else
#warning "You must replace the values marked FILLMEIN with real values from the TTN control panel!"
#define FILLMEIN (#dont edit this, edit the lines that use FILLMEIN)
#endif

// This EUI must be in little-endian format, so least-significant-byte
// first. When copying an EUI from ttnctl output, this means to reverse
// the bytes. For TTN issued EUIs the last bytes should be 0xD5, 0xB3,
// 0x70.
static const u1_t PROGMEM APPEUI[8] = { FILLMEIN };
void os_getArtEui (u1_t* buf) { memcpy_P(buf, APPEUI, 8);}

// This should also be in little endian format, see above.
static const u1_t PROGMEM DEVEUI[8] = { FILLMEIN };
void os_getDevEui (u1_t* buf) { memcpy_P(buf, DEVEUI, 8);}

// This key should be in big endian format (or, since it is not really a
// number but a block of memory, endianness does not really apply). In
// practice, a key taken from the TTN console can be copied as-is.
static const u1_t PROGMEM APPKEY[16] = { FILLMEIN };
void os_getDevKey (u1_t* buf) { memcpy_P(buf, APPKEY, 16);}

// payload to send to TTN gateway
static uint8_t payload[5];
static osjob_t sendjob;

// Schedule TX every this many seconds (might become longer due to duty
// cycle limitations).
const unsigned TX_INTERVAL = 10;

// dht
float temperature;
float rHumidity;

// Pin mapping for Adafruit Feather M0 LoRa
const lmic_pinmap lmic_pins = {
  .nss = 8,
  .rxtx = LMIC_UNUSED_PIN,
  .rst = 4,
  .dio = {3, 6, LMIC_UNUSED_PIN},
  .rxtx_rx_active = 0,
  .rssi_cal = 8, // LBT cal for the Adafruit Feather M0 LoRa, in dB
  .spi_freq = 800000,
};

// init. DHT
DHT dht(DHTPIN, DHTTYPE);

void onEvent (ev_t ev) {
  Serial.print(os_getTime());
  Serial.print(": ");
  switch(ev) {
    case EV_SCAN_TIMEOUT:
      Serial.println(F("EV_SCAN_TIMEOUT"));
      break;
    case EV_BEACON_FOUND:
      Serial.println(F("EV_BEACON_FOUND"));
      break;
    case EV_BEACON_MISSED:
      Serial.println(F("EV_BEACON_MISSED"));
      break;
    case EV_BEACON_TRACKED:
      Serial.println(F("EV_BEACON_TRACKED"));
      break;
    case EV_JOINING:
      display.print("TTN: Joining...");
      display.display();
      Serial.println(F("EV_JOINING"));
      break;
    case EV_JOINED:
      display.clearDisplay();
      display.display();
      display.setCursor(0, 0);
      display.println("TTN: Connected");
      display.display();
      Serial.println(F("EV_JOINED"));
      {
        u4_t netid = 0;
        devaddr_t devaddr = 0;
        u1_t nwkey[16];
        u1_t artkey[16];
        LMIC_getSessionKeys(&netid, &devaddr, nwkey, artkey);
        Serial.print("netid: ");
        Serial.println(netid, DEC);
        Serial.print("devaddr: ");
        Serial.println(devaddr, HEX);
        Serial.print("artKey: ");
        for (int i=0; i<sizeof(artkey); ++i) {
          if (i != 0)
            Serial.print("-");
          Serial.print(artkey[i], HEX);
        }
        Serial.println("");
        Serial.print("nwkey: ");
        for (int i=0; i<sizeof(nwkey); ++i) {

```

```

        if (i != 0)
            Serial.print("-");
        Serial.print(nwkKey[i], HEX);
    }
    Serial.println("");
}
// Disable link check validation (automatically enabled
// during join, but because slow data rates change max TX
// size, we don't use it in this example.
LMIC_setLinkCheckMode(0);
break;

/*
|| This event is defined but not used in the code. No
|| point in wasting codespace on it.
||
|| case EV_RFU1:
||     Serial.println(F("EV_RFU1"));
||     break;
||
*/
case EV_JOIN_FAILED:
    Serial.println(F("EV_JOIN_FAILED"));
    break;
case EV_REJOIN_FAILED:
    Serial.println(F("EV_REJOIN_FAILED"));
    break;
    break;
case EV_TXCOMPLETE:
    display.clearDisplay();
    display.display();
    display.setCursor(0, 0);
    display.println("TTN: Connected");
    display.display();
    display.setCursor(0, 20);
    display.println("* Sent!");
    display.display();
    Serial.println(F("EV_TXCOMPLETE (includes waiting for RX windows)"));
    if (LMIC.txrxFlags & TXRX_ACK)
        Serial.println(F("Received ack"));
    if (LMIC.dataLen) {
        Serial.println(F("Received "));
        Serial.println(LMIC.dataLen);
        Serial.println(F(" bytes of payload"));
    }
    // Schedule next transmission
    os_setTimedCallback(&sendjob, os_getTime()+sec2osticks(TX_INTERVAL), do_send);
    break;
case EV_LOST_TSYNC:
    Serial.println(F("EV_LOST_TSYNC"));
    break;
case EV_RESET:
    Serial.println(F("EV_RESET"));
    break;
case EV_RXCOMPLETE:
    // data received in ping slot
    Serial.println(F("EV_RXCOMPLETE"));
    break;
case EV_LINK_DEAD:
    Serial.println(F("EV_LINK_DEAD"));
    break;
case EV_LINK_ALIVE:
    Serial.println(F("EV_LINK_ALIVE"));
    break;

/*
|| This event is defined but not used in the code. No
|| point in wasting codespace on it.
||
|| case EV_SCAN_FOUND:
||     Serial.println(F("EV_SCAN_FOUND"));
||     break;
||
*/
case EV_TXSTART:
    display.clearDisplay();
    display.display();
    display.setCursor(0, 0);
    display.println("TTN: Connected");
    display.setCursor(0, 10);
    display.println("* Sending");
    display.setCursor(0, 25);
    display.print("Temp: ");display.print(temperature*100);display.print(" C, ");
    display.print("RH%: ");display.print(rHumidity*100);
    display.display();
    Serial.println(F("EV_TXSTART"));
    break;
default:
    Serial.print(F("Unknown event: "));
    Serial.println((unsigned) ev);
    break;
}
}

void do_send(osjob_t* j){
    // Check if there is not a current TX/RX job running
    if (LMIC.opmode & OP_TXRXPEND) {
        Serial.println(F("OP_TXRXPEND, not sending"));
    } else {
        // read the temperature from the DHT22
        temperature = dht.readTemperature();
        Serial.print("Temperature: "); Serial.print(temperature);
    }
}

```

```

Serial.println(" *C");
// adjust for the f2sflt16 range (-1 to 1)
temperature = temperature / 100;

// read the humidity from the DHT22
rHumidity = dht.readHumidity();
Serial.print("%RH ");
Serial.println(rHumidity);
// adjust for the f2sflt16 range (-1 to 1)
rHumidity = rHumidity / 100;

// float -> int
// note: this uses the sflt16 datum (https://github.com/mcci-catena/arduino-lmic#sflt16)
uint16_t payloadTemp = LMIC_f2sflt16(temperature);
// int -> bytes
byte tempLow = lowByte(payloadTemp);
byte tempHigh = highByte(payloadTemp);
// place the bytes into the payload
payload[0] = tempLow;
payload[1] = tempHigh;

// float -> int
uint16_t payloadHumid = LMIC_f2sflt16(rHumidity);
// int -> bytes
byte humidLow = lowByte(payloadHumid);
byte humidHigh = highByte(payloadHumid);
payload[2] = humidLow;
payload[3] = humidHigh;

// prepare upstream data transmission at the next possible time.
// transmit on port 1 (the first parameter); you can use any value from 1 to 223 (others are reserved).
// don't request an ack (the last parameter, if not zero, requests an ack from the network).
// Remember, acks consume a lot of network resources; don't ask for an ack unless you really need it.
LMIC_setTxData2(1, payload, sizeof(payload)-1, 0);
Serial.println(F("EV_TXSTART"));
}
// Next TX is scheduled after TX_COMPLETE event.
}

void setup() {
  delay(5000);
  while (! Serial);
  Serial.begin(9600);
  Serial.println(F("Starting"));

  dht.begin();
  // by default, we'll generate the high voltage from the 3.3v line internally! (neat!)
  display.begin(SSD1306_SWITCHCAPVCC, 0x3C); // initialize with the I2C addr 0x3C (for the 128x32)
  Serial.println("OLED and DHT init'd");

  // Show image buffer on the display hardware.
  // Since the buffer is initialized with an Adafruit splashscreen
  // internally, this will display the splashscreen.
  display.display();
  delay(1000);

  // Clear the buffer.
  display.clearDisplay();
  display.display();

  // set text display size/location
  display.setTextSize(1);
  display.setTextColor(WHITE);
  display.setCursor(0,0);

  // LMIC init
  os_init();
  // Reset the MAC state. Session and pending data transfers will be discarded.
  LMIC_reset();
  // Disable link-check mode and ADR, because ADR tends to complicate testing.
  LMIC_setLinkCheckMode(0);
  // Set the data rate to Spreading Factor 7. This is the fastest supported rate for 125 kHz channels, and it
  // minimizes air time and battery power. Set the transmission power to 14 dBi (25 mW).
  LMIC_setDrTxpow(DR_SF7,14);
  // in the US, with TTN, it saves join time if we start on subband 1 (channels 8-15). This will
  // get overridden after the join by parameters from the network. If working with other
  // networks or in other regions, this will need to be changed.
  LMIC_selectSubBand(1);

  // Start job (sending automatically starts OTAA too)
  do_send(&sendjob);
}

void loop() {
  // we call the LMIC's runloop processor. This will cause things to happen based on events and time. One
  // of the things that will happen is callbacks for transmission complete or received messages. We also
  // use this loop to queue periodic data transmissions. You can put other things here in the `loop()` routine,
  // but beware that LoRaWAN timing is pretty tight, so if you do more than a few milliseconds of work, you
  // will want to call `os_runloop_once()` every so often, to keep the radio running.
  os_runloop_once();
}

```

[View on GitHub](#)

## Using a Feather 32u4

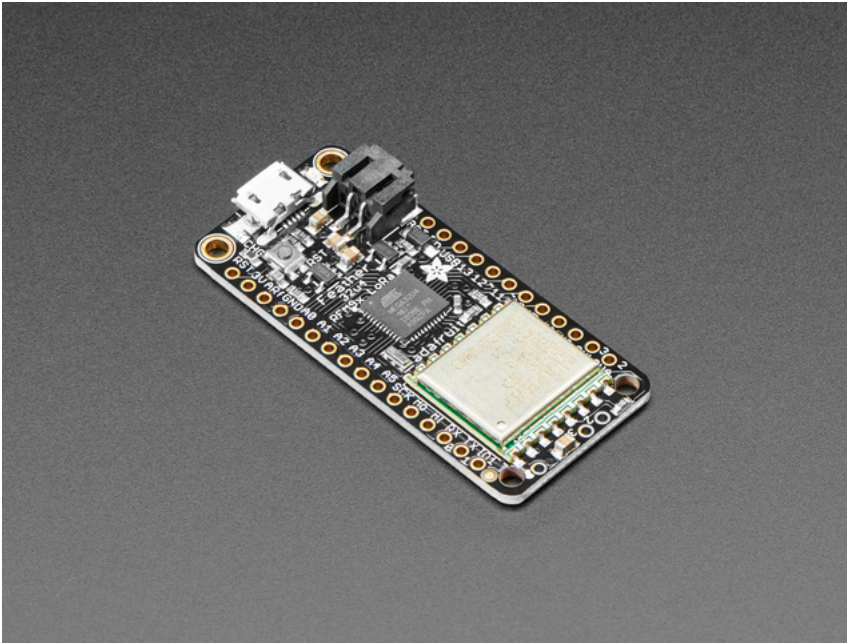
If you're using a feather with a small flash size (like the Feather 32u4), you'll need a smaller library to experiment with than the one in this guide.

Enter [TinyLoRa](#), an Arduino Library for Feathers and arduino-compatibles using the HopeRF RFM95/96W module. It's less than 8kB, making it perfect for smaller chips! It's also compatible with the Feather M0 LoRa, in case you want a smaller communications library for your speedy M0.

One of the trade-offs of using this tiny library (compared to a more featured library like the MCCI-LMIC Library) is that it uses a different method of device registration, Activation-by-Personalization (ABP). Instead of performing a secure join procedure, the security keys are hardcoded into the Sketch and "burned" into the Feather during compilation.

## Parts

First, let's gather the parts required for this project.



[Adafruit Feather 32u4 RFM95 LoRa Radio- 868 or 915 MHz](#)

This is the Adafruit Feather 32u4 LoRa Radio (RFM9x). We call these RadioFruits, our take on an microcontroller with a...

\$34.95

In Stock

[Add to Cart](#)

1 x [DHT22 Temperature + Humidity Sensor](#)

The DHT22 is a basic, low-cost digital temperature and humidity sensor.

[Add to Cart](#)

1 x [Breadboard](#)

Half-Sized Breadboard

[Add to Cart](#)

1 x [Breadboard Wires](#)

Breadboard Wiring Bundle

[Add to Cart](#)

## Wiring

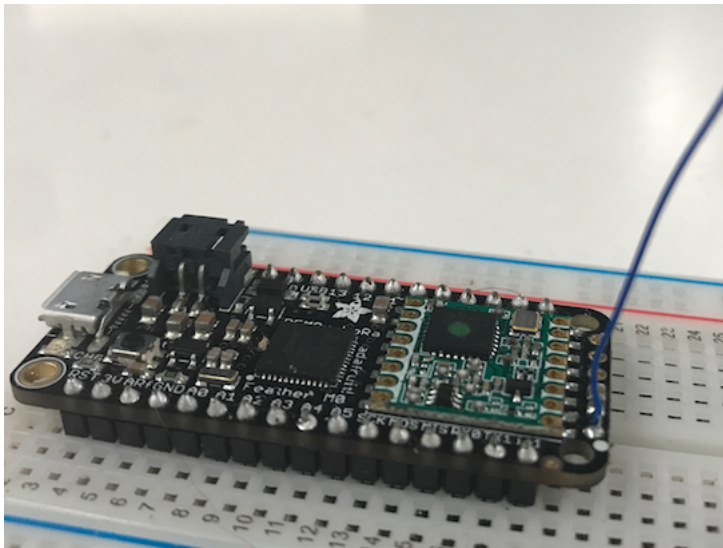
Next, let's wire up our circuit.

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

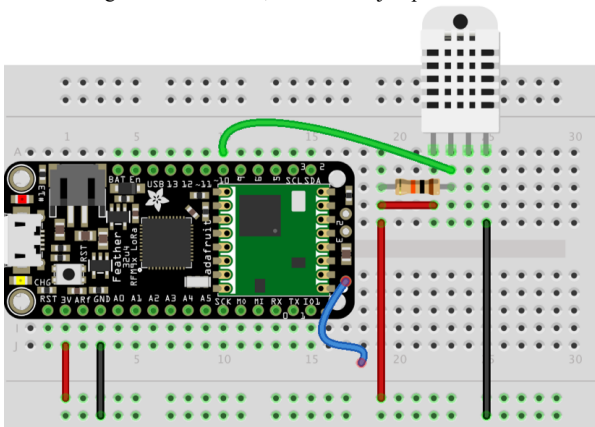
For this guide, and to keep the build cost-effective, we soldered a small, 82mm wire to the ANT pad.

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





When wiring the Feather 32u4, there is NO jumper between Pin 6 and IO1 as shown in the Feather M0 LoRa wiring diagram.



Make the following connections **between the Feather 32u4 and the DHT22**:

- Feather 3V to DHT22 Pin 1
- Feather Pin 10 to DHT22 Pin 2
- Feather Ground to DHT22 Pin 3
- Add a 10k ohm resistor between the DHT22's VCC (DHT Pin 1) and Data Pin (DHT Pin 2).

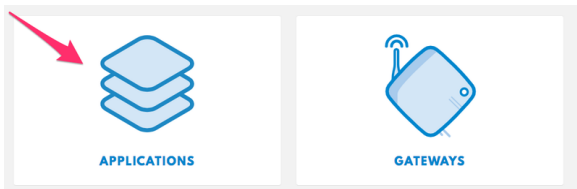
## TinyLoRa TTN Setup

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](#) to set up an account.

Once logged in, [navigate to the The Things Network Console](#). 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.

THE THINGS CONSOLE

Applications > Add Application

**ADD APPLICATION**

**Application ID**  
The unique identifier of your application on the network  
my-feather

**Description**  
A human readable description of your new app  
Eg. My sensor network application

**Application EUI**  
An application EUI will be issued for The Things Network block for convenience, you can add your own in the application settings page.  
EUI issued by The Things Network

**Handler registration**  
Select the handler you want to register this application to  
ttn-handler-us-west

Cancel Add application

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

Applications > my-feather

Overview Devices Payload Formats Integrations Data Settings

**APPLICATION OVERVIEW**

Application ID my-feather documentation

Description  
Created 13 seconds ago  
Handler ttn-handler-us-west

**APPLICATION EUIS** manage euis

70 83 05 7E D0 01 43 41

**DEVICES** register device manage devices

0 registered devices

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.

**REGISTER DEVICE** built-in root devices

**Device ID**  
This is the unique identifier for the device in this app. The device ID will be immutable.  
feather32j4

**Device EUI**  
The device EUI is the unique identifier for this device on the network. You can change the EUI later.  
this field will be generated

**App Key**  
The App Key will be used to secure the communication between you, device and the network.  
this field will be generated

**App EUI**  
70 83 05 7E D0 01 43 41

Cancel Register

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

Applications > my-feather > Devices > feather32j4

Overview Data Settings

**DEVICE OVERVIEW**

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

**SETTINGS**

**Description**  
A human-readable description of the device  
Feather 32u4 LoRa

**Device EUI**  
The serial number of your radio module, similar to a MAC address  
00 09 7A 42 9B 9C 1C CB

**Application EUI**  
70 B3D5 7ED001 43 41

**Activation Method**  
OTA ABP

**Frame Counter Width**  
16 bit 32 bit

**Frame Counter Checks**  
Disabling frame counter checks dra

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.

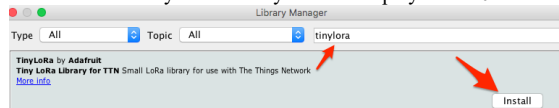
The Things Network's documentation page has [a full explanation about the role of the Frame Counter](#).

## TinyLoRa Arduino Setup

Before proceeding with this page, make sure your board is correctly set-up by following the [Using the Arduino IDE](#) page of the setup guide for the Feather 32u4 LoRa.

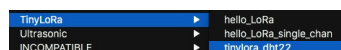
To install [the TinyLoRa library](#), **Open the Arduino Library Manager (Sketch->Include Library->Manage Libraries)**. In the search box, type **TinyLoRa**. The **TinyLoRa** library should be the first result. **Click Install**.

Make sure the TinyLoRa library version displayed is 1.0.1 or above.

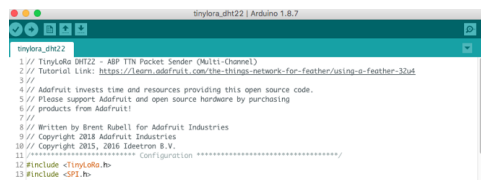


We've added an example for using the DHT22 with the TinyLoRa Library.

To open it from the Arduino IDE, click **File -> Examples -> TinyLoRa -> tinylora\_dht22**



The Arduino IDE should open a new window with the *tinylora\_dht22* sketch.



Now that our IDE is setup up, let's learn how to send data from our Feather to The Things Network and decode it!

## TinyLoRa Usage

### Code Setup

To register your Feather with The Things Network using ABP, you'll need to set three unique identifiers in the sketch: the Network Session Key, the Device Address, and the Application Session Key.

#### DEVICE OVERVIEW

Application ID my-feather


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

Device ID feather32u4

Description Feather 32u4 LoRa


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

Activation Method ABP

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


Device Address  msb { 0x26, 0x02, 0x14, 0x58 }


Network Session Key  9B 27 5D F6 65 88 1A D7 95 F1 97 A1 73 CB A4 FC


App Session Key  . . . . .

We're going to copy each unique identifier from the Device Overview to the variables within the sketch.

First, copy the **Network Session Key** from the TTN console to the `NwkKey` variable in the sketch.

Device Address  msb { 0x26, 0x02, 0x14, 0x58 }

Network Session Key  msb { 0x9B, 0x27, 0x5D, 0xF6, 0x65, 0x88, 0x1A, 0xD7, 0x95, 0xF1, 0x97, 0xA1, 0x73, 0xCB, 0xA4, 0xFC }


App Session Key  msb { 0x84, 0x9C, 0x26, 0x0E, 0xC3, 0x4A, 0x40, 0x98, 0x38, 0x7 }


```


1 // Network Session Key (NWK)
2 uint8_t NwkKey[16] = { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 };
3
4 // Application Session Key (APP)
5 uint8_t AppKey[16] = { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 };
6
7 // Device Address (DEV)
8 uint8_t DevAddr[4] = { 0x00, 0x00, 0x00, 0x00 };

```

Next, copy the **Application Session Key** from the TTN console to the `AppKey` variable in the sketch.

Device Address  msb { 0x26, 0x02, 0x14, 0x58 }

Network Session Key  msb { 0x9B, 0x27, 0x5D, 0xF6, 0x65, 0x88, 0x1A, 0xD7, 0x95, 0xF1, 0x97, 0xA1, 0x73, 0xCB, 0xA4, 0xFC }


App Session Key  msb { 0x84, 0x9C, 0x26, 0x0E, 0xC3, 0x4A, 0x40, 0x98, 0x38, 0x7 }


```


19 // Network Session Key (NWK)
20 uint8_t NwkKey[16] = { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 };
21
22 // Application Session Key (APP)
23 uint8_t AppKey[16] = { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 };
24
25 // Device Address (DEV)
26 uint8_t DevAddr[4] = { 0x00, 0x00, 0x00, 0x00 };

```

Finally, copy the **Device Address** from the TTN console to the `DevAddr` variable in the sketch.

Device Address  msb { 0x26, 0x02, 0x14, 0x58 }

Network Session Key  msb { 0x9B, 0x27, 0x5D, 0xF6, 0x65, 0x88, 0x1A, 0xD7, 0x95, 0xF1, 0x97, 0xA1, 0x73, 0xCB, 0xA4, 0xFC }

App Session Key  msb { 0x84, 0x9C, 0x26, 0x0E, 0xC3, 0x4A, 0x40, 0x98, 0x38, 0x7 }

```

19 // Network Session Key (NWK)
20 uint8_t NwkKey[16] = { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 };
21
22 // Application Session Key (APP)
23 uint8_t AppKey[16] = { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 };
24
25 // Device Address (DEV)
26 uint8_t DevAddr[4] = { 0x26, 0x02, 0x14, 0x58 };

```

That's all for now - we're ready to upload the code onto our Arduino.

Note that to see the output you'll need to open up the Serial console. Once you've debugged your example, you can comment out or remove the `while (! Serial);` line from `setup()`

### Running the Sketch

Compile (**cmd/ctrl+r**) and upload (**cmd/ctrl+u**) the sketch to the Feather.

Then, open the Serial Monitor. You should see the humidity and temperature values printed to the console. Then, the serial should print out the frame counter value.

Unlike the example using the OTAA activation method, the device does not send join requests to the network. Each time `lora.sendData()` is called, it'll randomize the frequency (from region-specific bands) and broadcast.

[Download File](#)

[Copy Code](#)

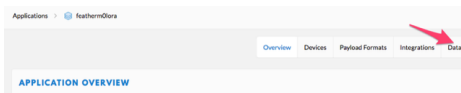
```

Starting LoRa...
Humidity: 33.20 %    Temperature: 23.30
Sending LoRa Data...
Frame Counter: 0
delaying...

```

### 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...

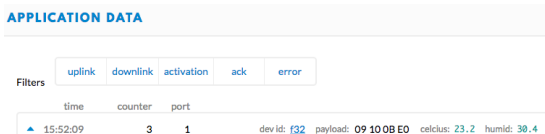
## Decoding the Payload

If you're sending packets in strange formats or encodings (like we are!), The Things Network Console has a [programmable data decoder](#) 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.



## Decoder Code

[Download Project Bundle](#)

[Copy Code](#)

```
// TinyLoRa - DHT22 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;
}
```

[View on GitHub](#)

This guide was first published on Sep 21, 2018. It was last updated on Sep 21, 2018.

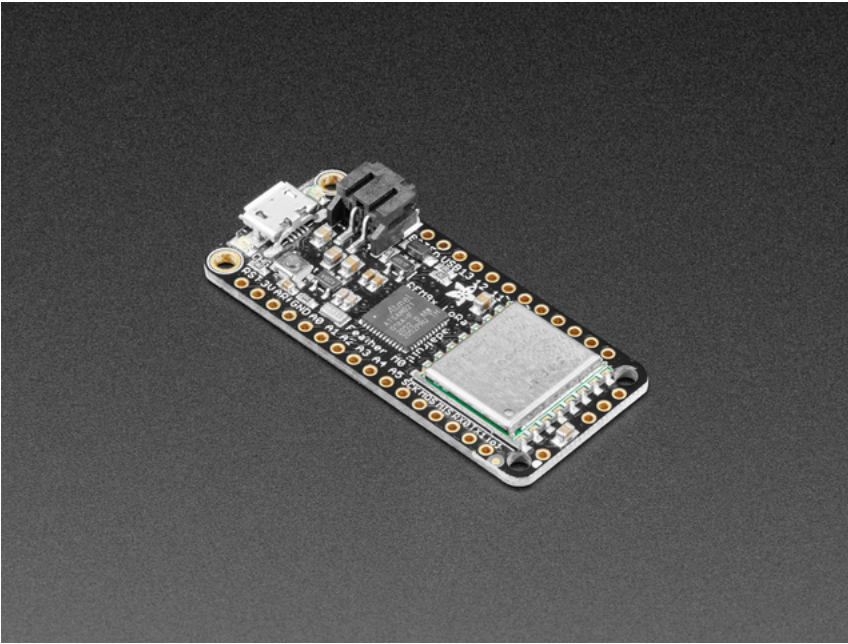
Difficulty: Intermediate

Guide Type: Project

Categories: [Feather](#)  
[Internet of Things - IOT](#)  
[Breakout Boards/Radio](#)

32 Saves

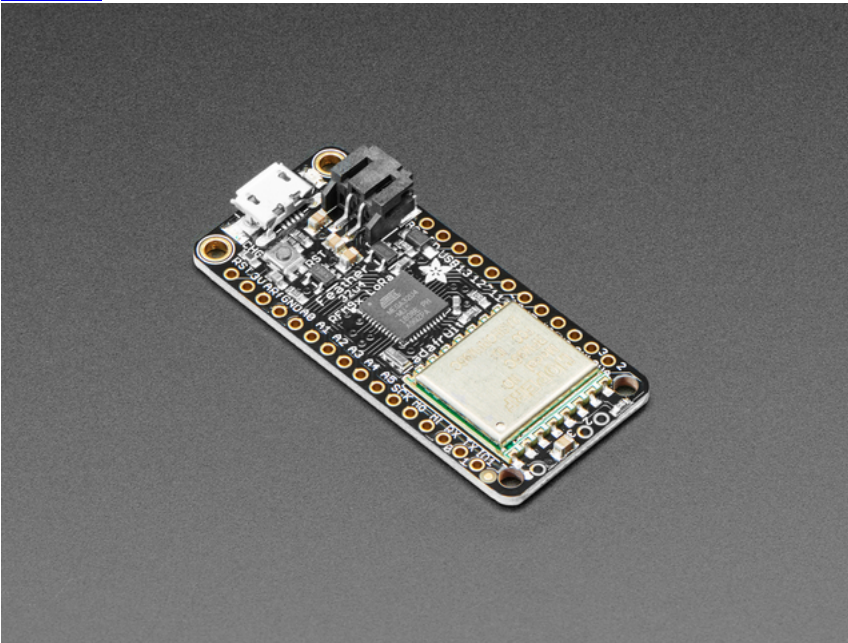
Featured Products



[Adafruit Feather M0 with RFM95 LoRa Radio - 900MHz](#)

\$34.95

[Add to Cart](#)

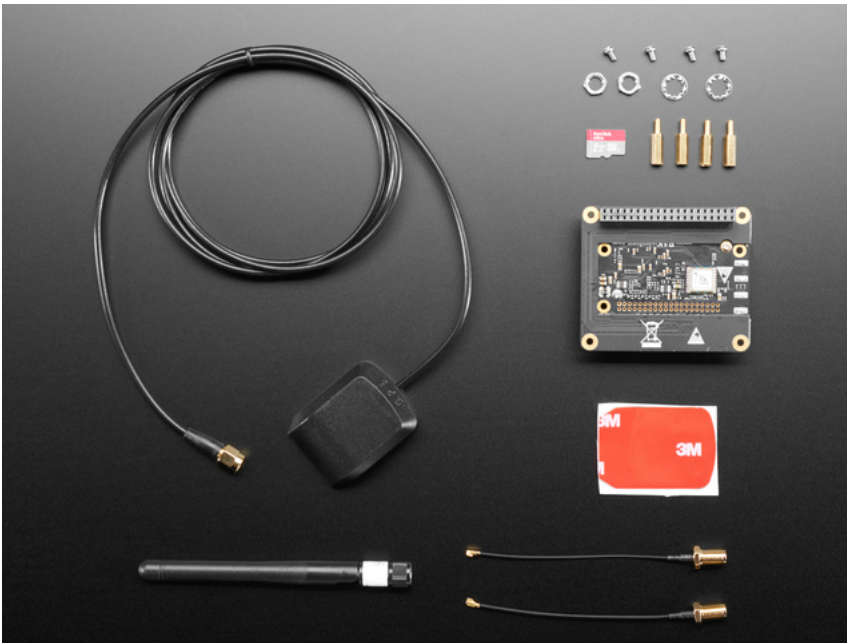


[Adafruit Feather 32u4 RFM95 LoRa Radio- 868 or 915 MHz](#)

\$34.95

[Add to Cart](#)

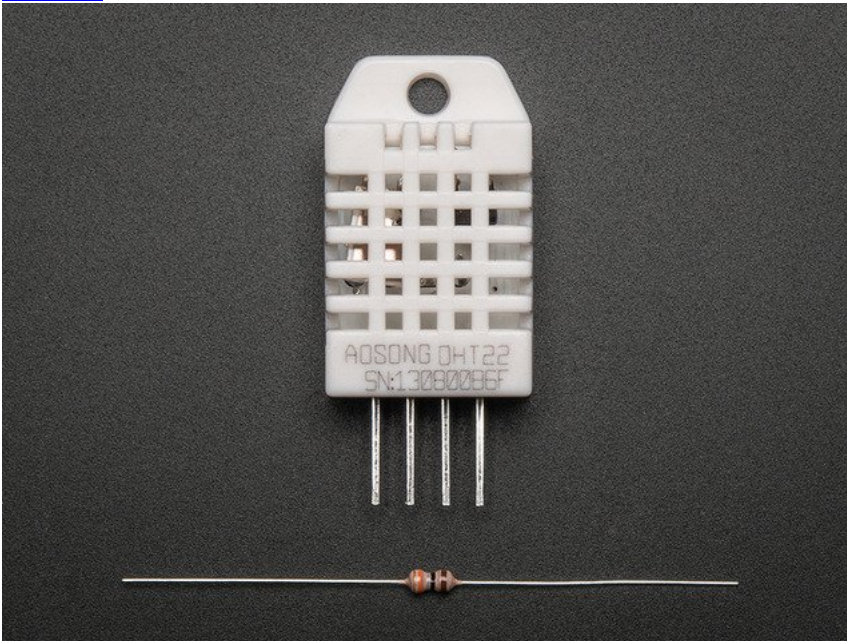




[8 Channel LoRa Gateway HAT with LoRa and GPS Antenna - SX1301](#)

\$159.95

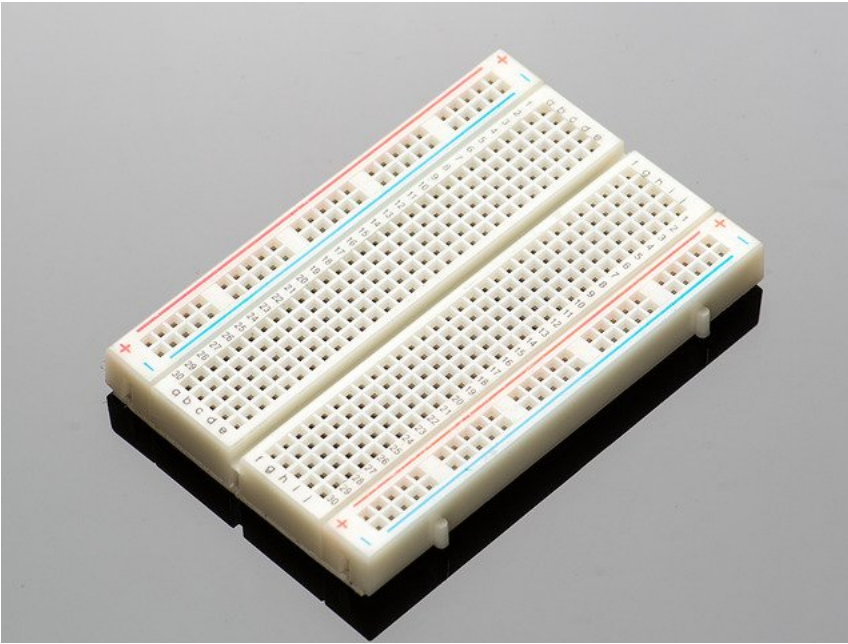
[Add to Cart](#)



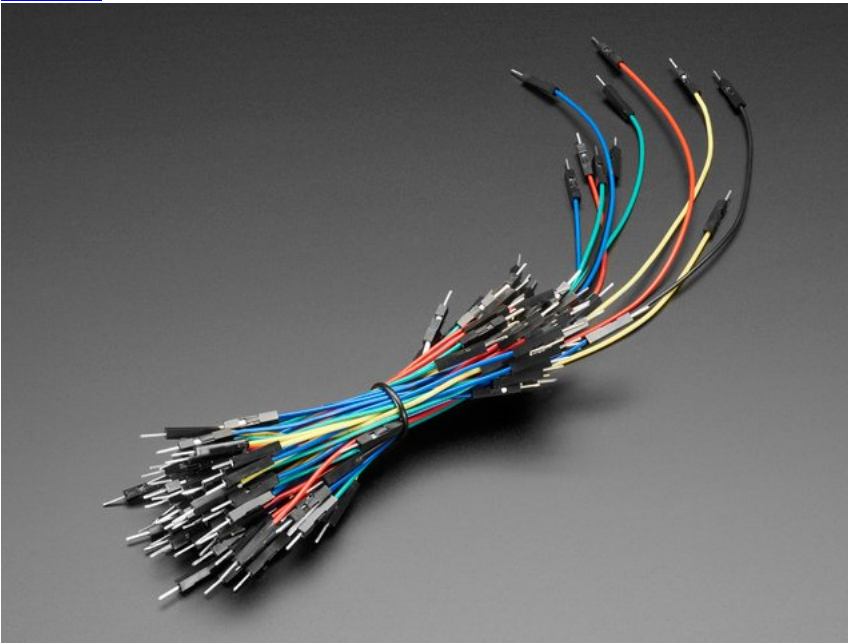
[DHT22 temperature-humidity sensor + extras](#)

\$9.95

[Add to Cart](#)

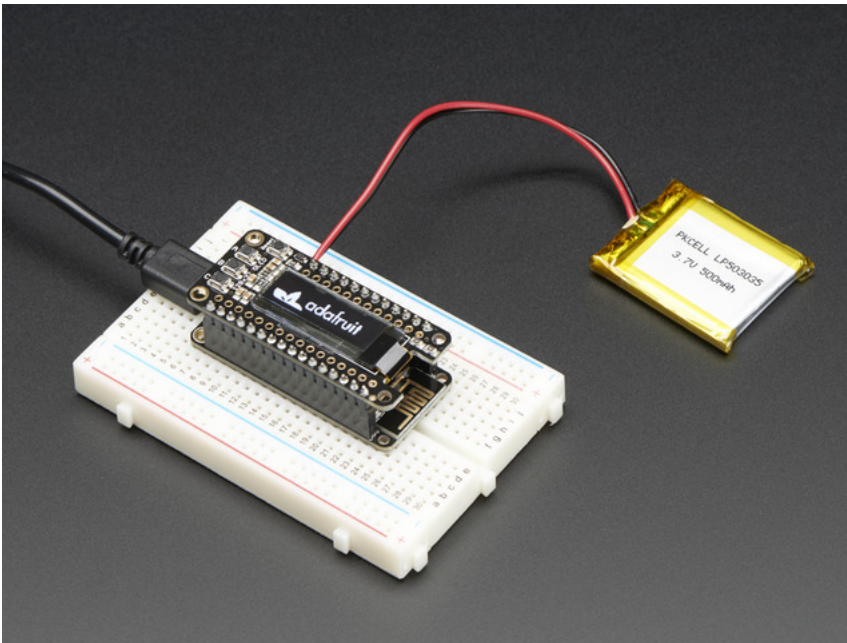
[Half-size breadboard](#)

\$5.00

[Add to Cart](#)[Breadboarding wire bundle](#)

\$4.95

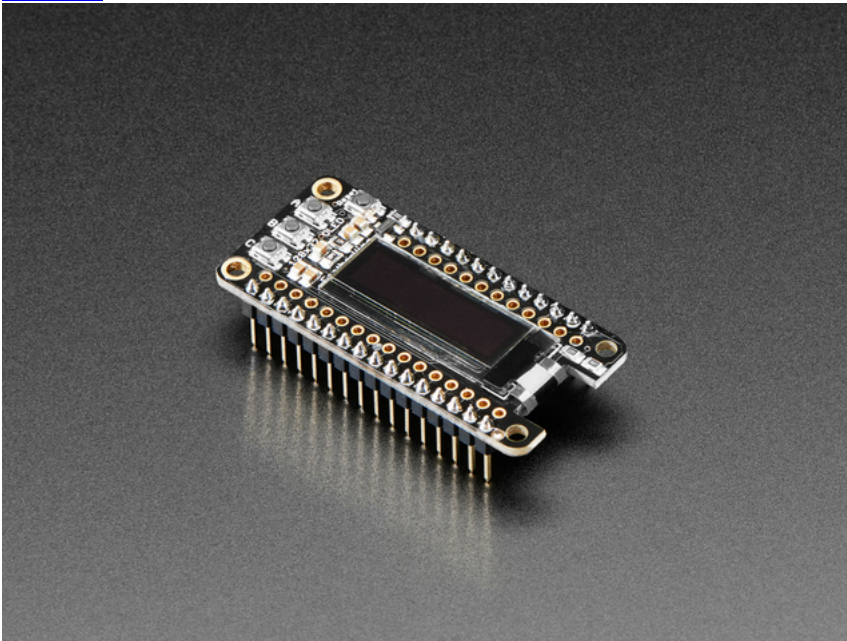
[Add to Cart](#)



[Adafruit FeatherWing OLED - 128x32 OLED Add-on For Feather](#)

\$14.95

[Add to Cart](#)



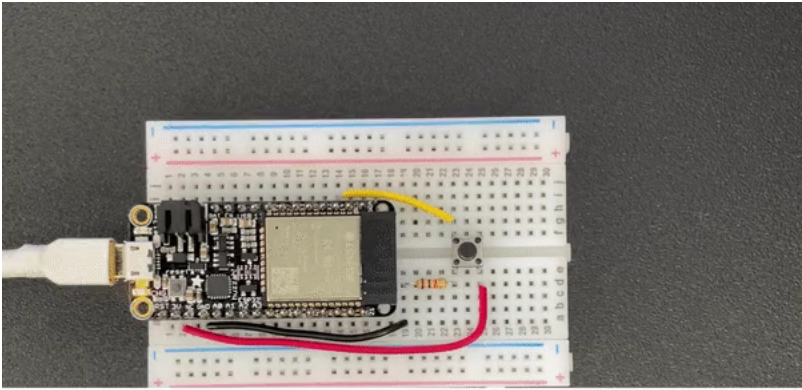
[Assembled Adafruit FeatherWing OLED](#)



\$15.95

[Add to Cart](#)

[Related Guides](#)

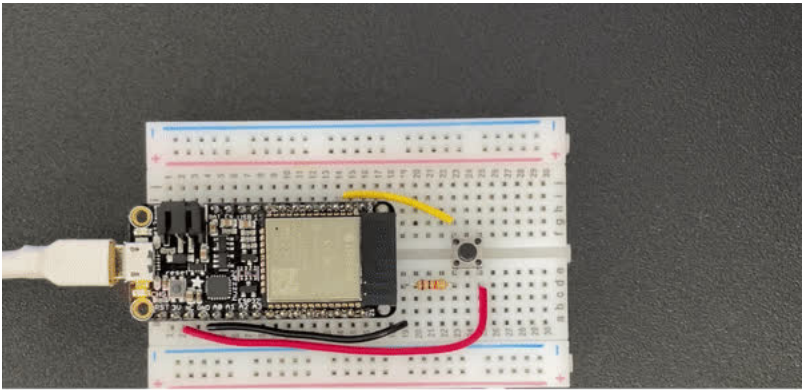






button D14 PUSH\_BUTTON  

Unpressed

Create Trigger | Add to Dashboard



button D14 PUSH\_BUTTON  

Unpressed

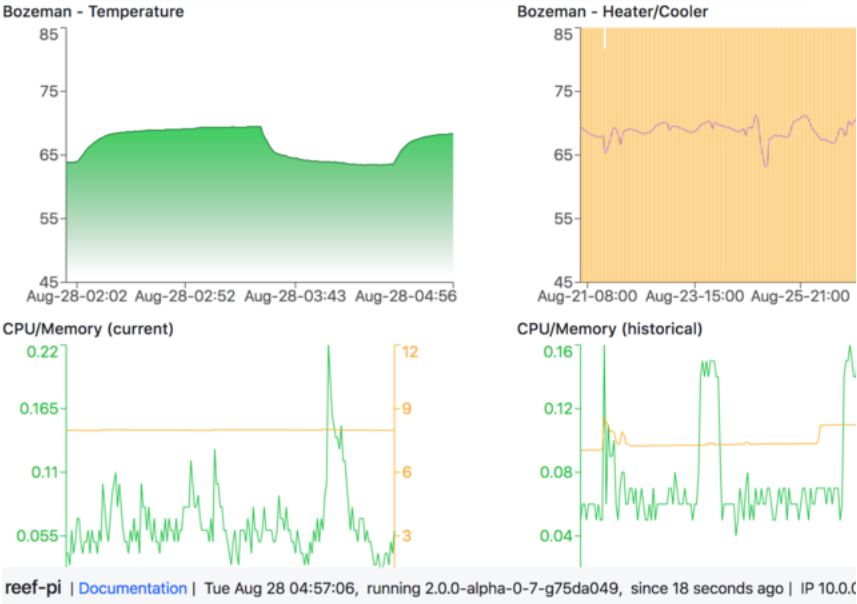
Create Trigger | Add to Dashboard

Digital Inputs with Adafruit IO WipperSnapper

By [Dylan Herrada](#)

1 Beginner

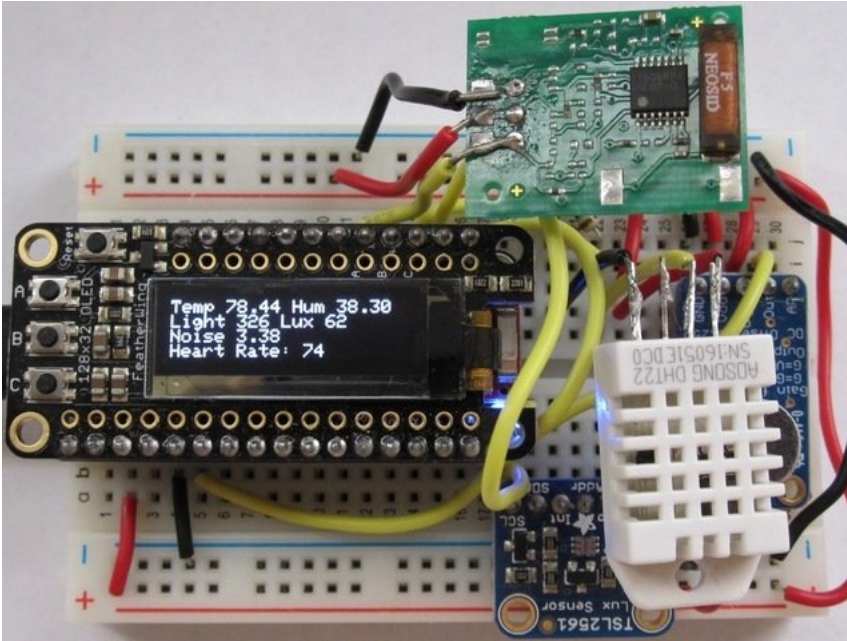
[ashboard](#) equipment timers lighting temperature ato doser macro configuration



[reef-pi Guide 1: Setup and Demonstration](#)By [Ranjib Dey](#)

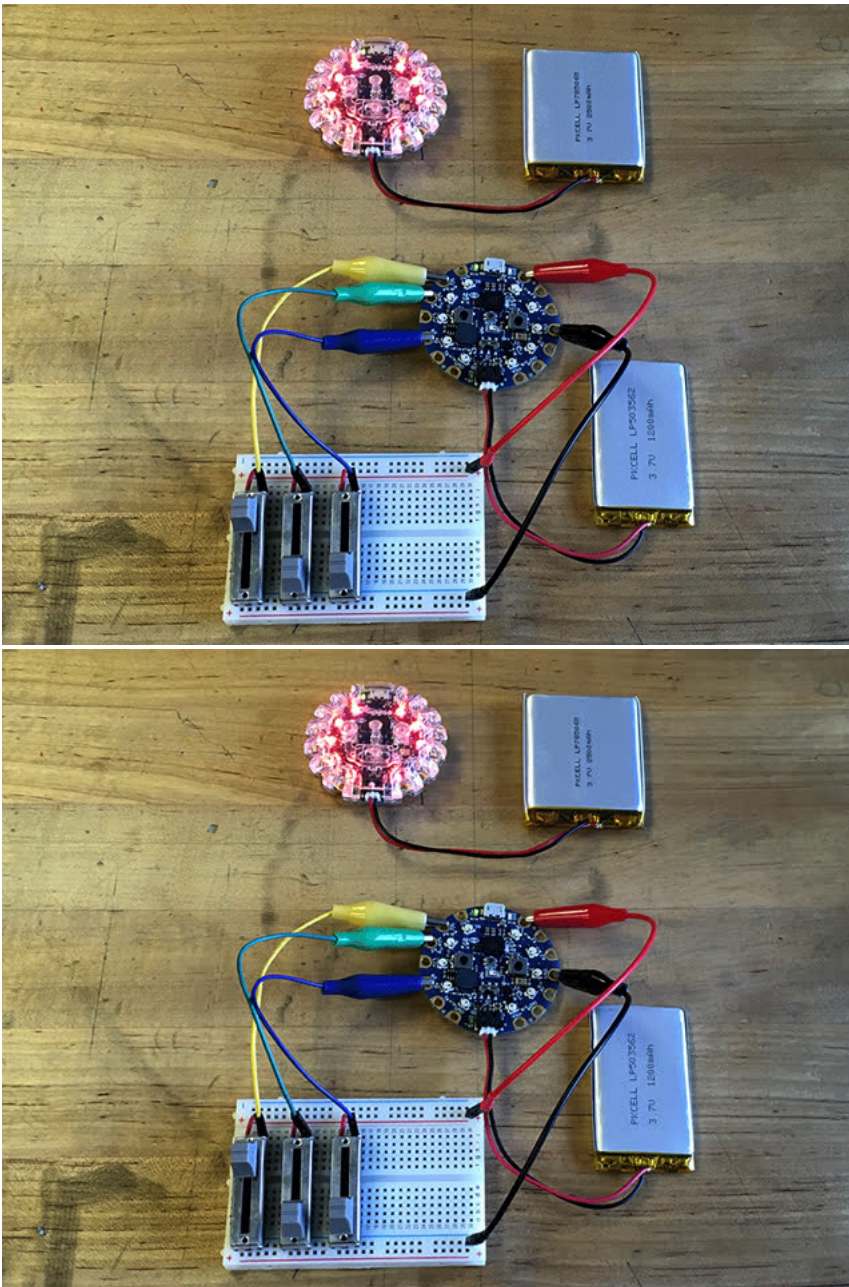
33

Intermediate

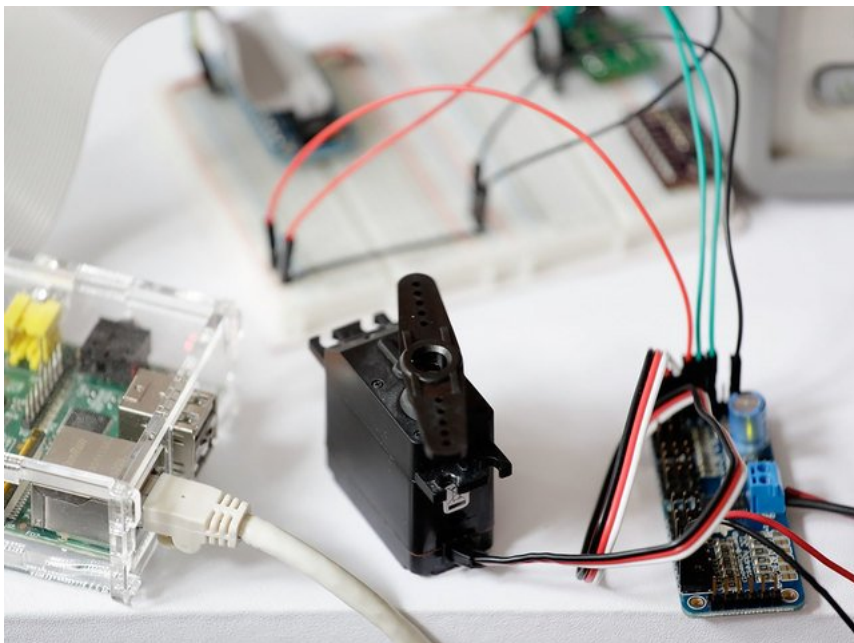
[Schluff - The Sleep Monitor](#)By [Michael Sklar](#)

26

Intermediate

By John Park11  
Beginner



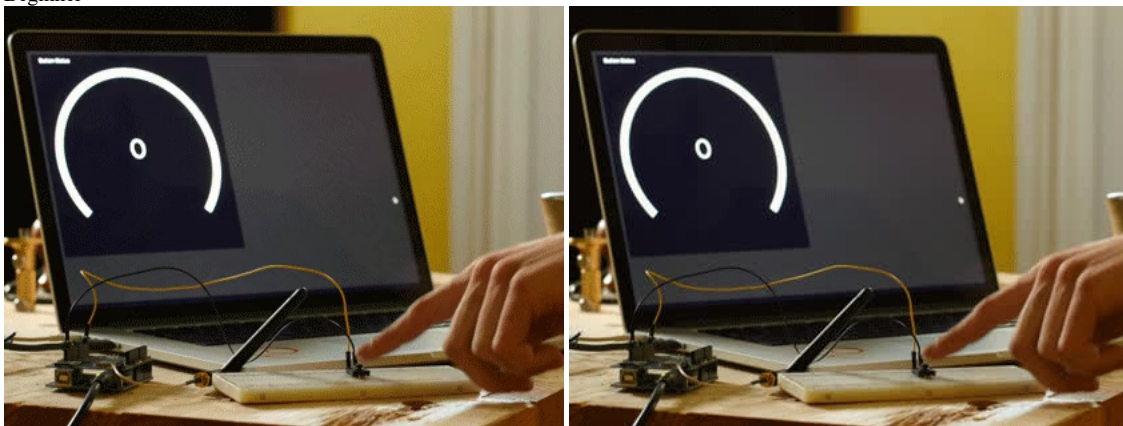


[Adafruit 16 Channel Servo Driver with Raspberry Pi](#)

By [Kevin Townsend](#)

47

Beginner



[Adafruit IO Basics: Digital Input](#)

By [Justin Cooper](#)

88

Beginner



[Compost Friend!](#)

By [Isaac Wellish](#)

17

Intermediate





[Arduino Lesson 11. LCD Displays - Part 1](#)

By [Simon Monk](#)

51

Beginner

[×](#)

#### OUT OF STOCK NOTIFICATION

YOUR NAME

YOUR EMAIL

[NOTIFY ME](#)

Search

## Search

#### Categories

No results for query

- «
- <
- [1](#)
- >
- »

- [Contact Us](#)
- [Tech Support Forums](#)
- [FAQs](#)
- [Shipping & Returns](#)
- [Terms of Service](#)
- [Privacy & Legal](#)

- [About Us](#)
- [Press](#)
- [Educators](#)
- [Distributors](#)
- [Jobs](#)
- [Gift Cards](#)

"The most dangerous phrase in the language is, We've always done it this way"

[Grace Hopper](#)





[A Minority and Woman-owned Business Enterprise \(M/WBE\)](#)

