adafruit learning system

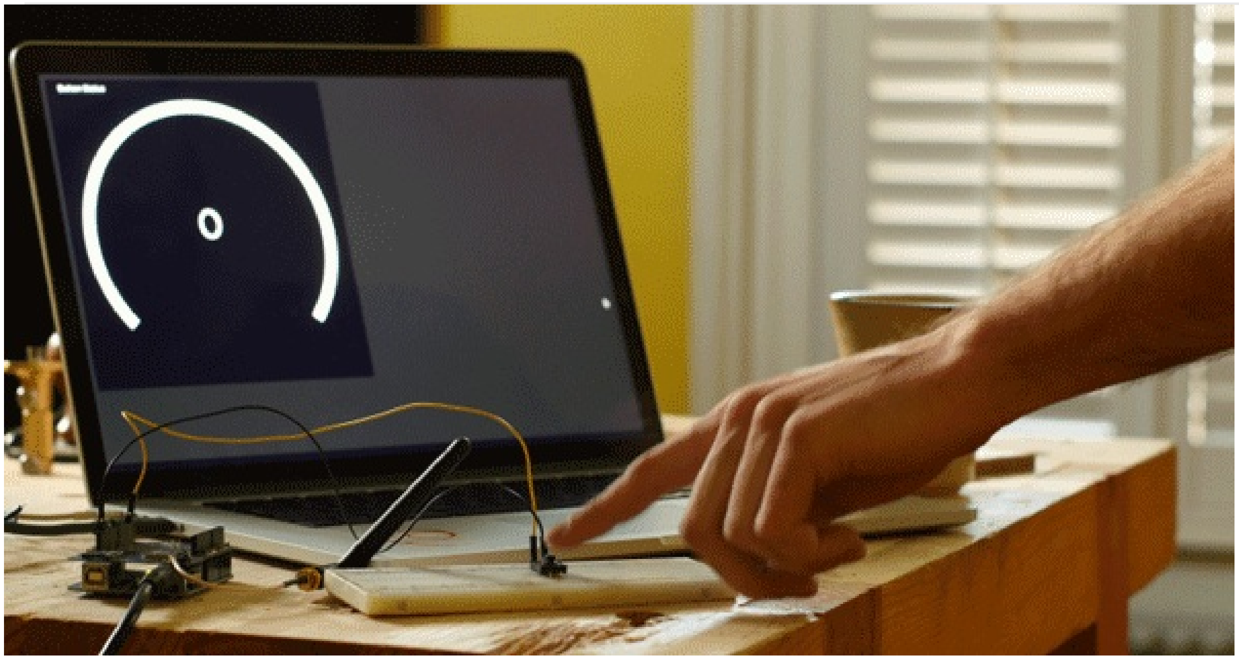
# Adafruit IO Basics: Digital Input

Created by Todd Treece

Last updated on 2015-07-23 12:30:28 PM EDT

# Guide Contents

## Overview



This guide is part of a series of guides that cover the basics of using Adafruit IO. It will show you how to send momentary button press data to Adafruit IO using four possible connection options:

- **Arduino Uno + CC300 WiFi shield**
- **Arduino Uno + Adafruit FONA cellular module**
- **HUZZAH ESP8266**
- **Raspberry Pi + USB WiFi dongle**

If you haven't worked your way through the Adafruit IO feed and dashboard basics guides, you should do that before continuing with this guide so you have a basic understanding of Adafruit IO.

- Adafruit IO Basics: Feeds (http://adafru.it/f09)
- Adafruit IO Basics: Dashboards (http://adafru.it/fCJ)

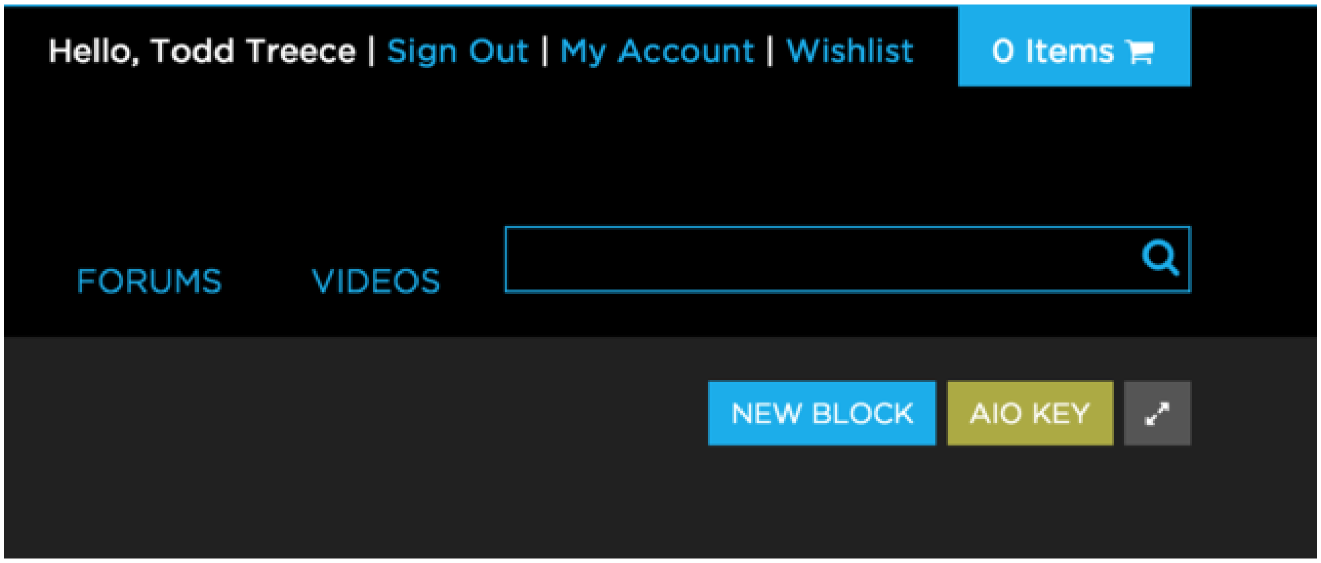
You should go through the setup guides associated with your selected set of hardware, and make sure you have internet connectivity with the device before continuing. The following links will take you to the guides for your selected platform.

- Adafruit CC300 WiFi Shield Setup Guide (http://adafru.it/fCK)
- Adafruit FONA Setup Guide (http://adafru.it/fCL)
- Adafruit HUZZAH ESP8266 Setup Guide (http://adafru.it/fCM)
- Raspberry Pi WiFi Setup Using the Adafruit Pi Finder (http://adafru.it/f72)

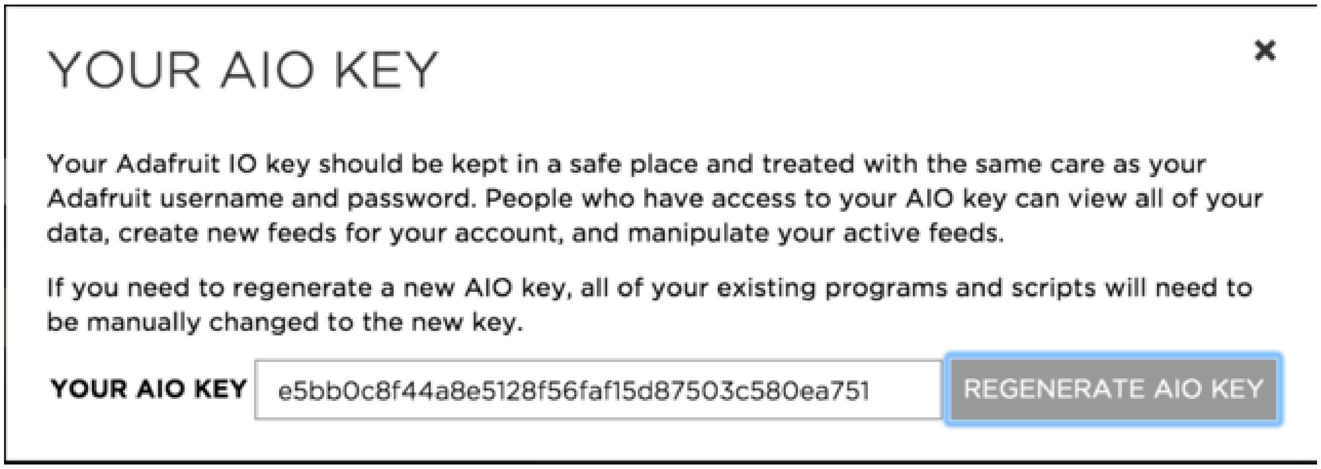
If you have went through all of the prerequisites for your selected hardware, you are now ready to move on to the Adafruit IO setup steps that are common between all three hardware choices for this project. Let's get started!

## Adafruit IO Setup

The first thing you will need to do is to login to your Adafruit IO account (http://adafru.it/eZ8) and get your Adafruit IO Key if you haven't already. Click the **AIO KEY** button on the right hand side of the window to retrieve your key.



A window will pop up with your Adafruit IO. Keep a copy of this in a safe place. We'll need it later.



YOUR AIO KEY

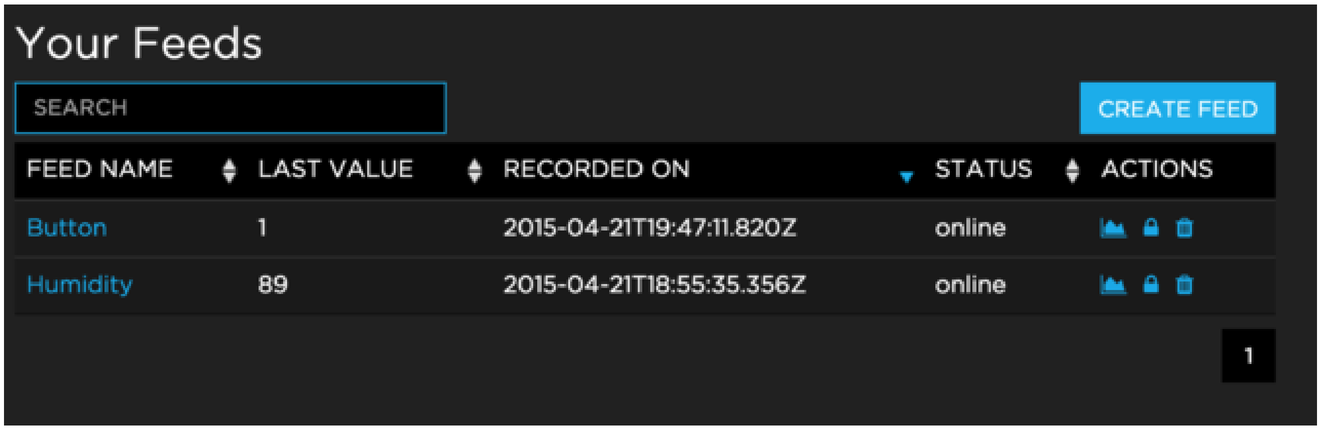
Your Adafruit IO key should be kept in a safe place and treated with the same care as your Adafruit username and password. People who have access to your AIO key can view all of your data, create new feeds for your account, and manipulate your active feeds.

If you need to regenerate a new AIO key, all of your existing programs and scripts will need to be manually changed to the new key.

YOUR AIO KEY e5bb0c8f44a8e5128f56faf15d87503c580ea751

### Creating the Button Feed

Next, you will need to create a feed called *"Button"*. If you need help getting started with creating feeds on Adafruit IO, check out the Adafruit IO Feed Basics guide (http://adafru.it/f20).

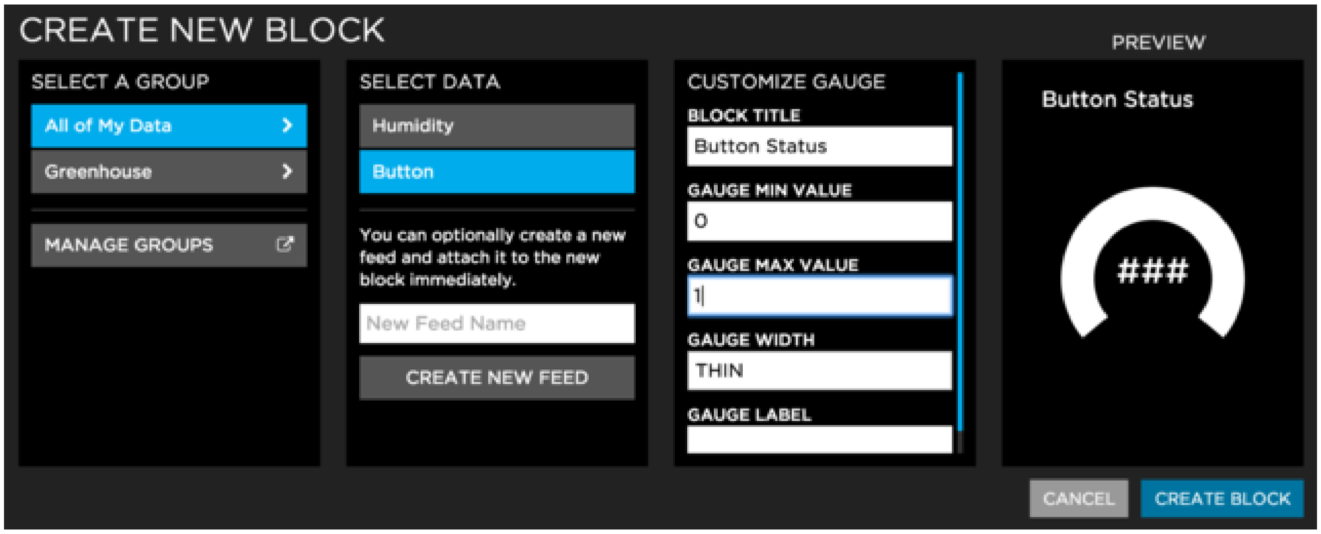

Your Feeds

| FEED NAME | LAST VALUE | RECORDED ON | STATUS | ACTIONS |
|---|---|---|---|---|
| Button | 1 | 2015-04-21T19:47:11.820Z | online | |
| Humidity | 89 | 2015-04-21T18:55:35.356Z | online | |

## Adding the Gauge Block

Next, add a new *Gauge* block to a new or existing dashboard. Name the block whatever you would like, and *give it a max value of 1*. Make sure you have selected the *Button* feed as the data source for the gauge.

If you need help getting started with Dashboards on Adafruit IO, check out the Adafruit IO Dashboard Basics guide (http://adafru.it/f21).


CREATE NEW BLOCK

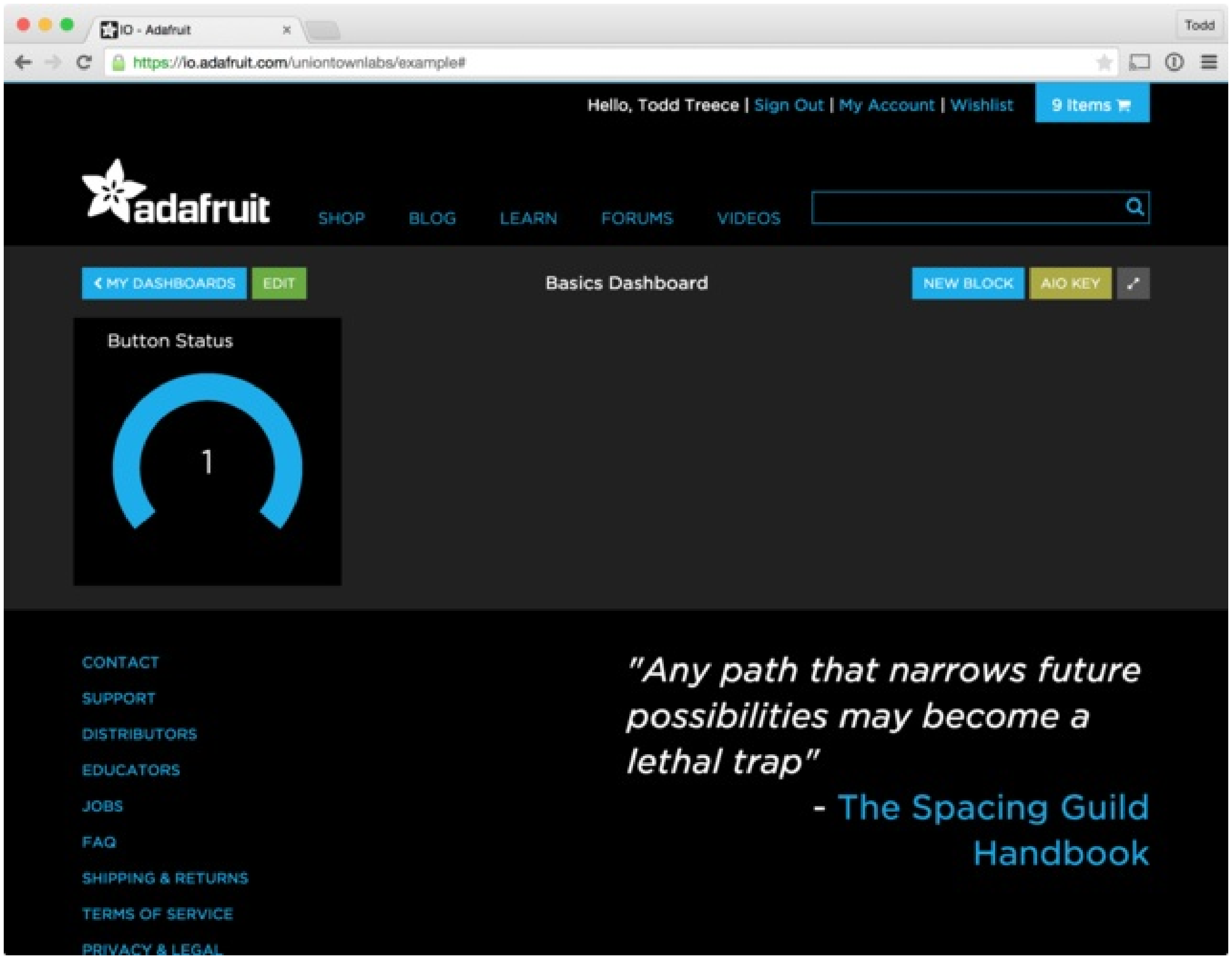
SELECT A GROUP: All of My Data

SELECT DATA: Button

You can optionally create a new feed and attach it to the new block immediately.

CUSTOMIZE GAUGE
BLOCK TITLE: Button Status
GAUGE MIN VALUE: 0
GAUGE MAX VALUE: 1
GAUGE WIDTH: THIN
GAUGE LABEL:

When you are finished editing the form, click *Create Block* to add the new block to the dashboard.



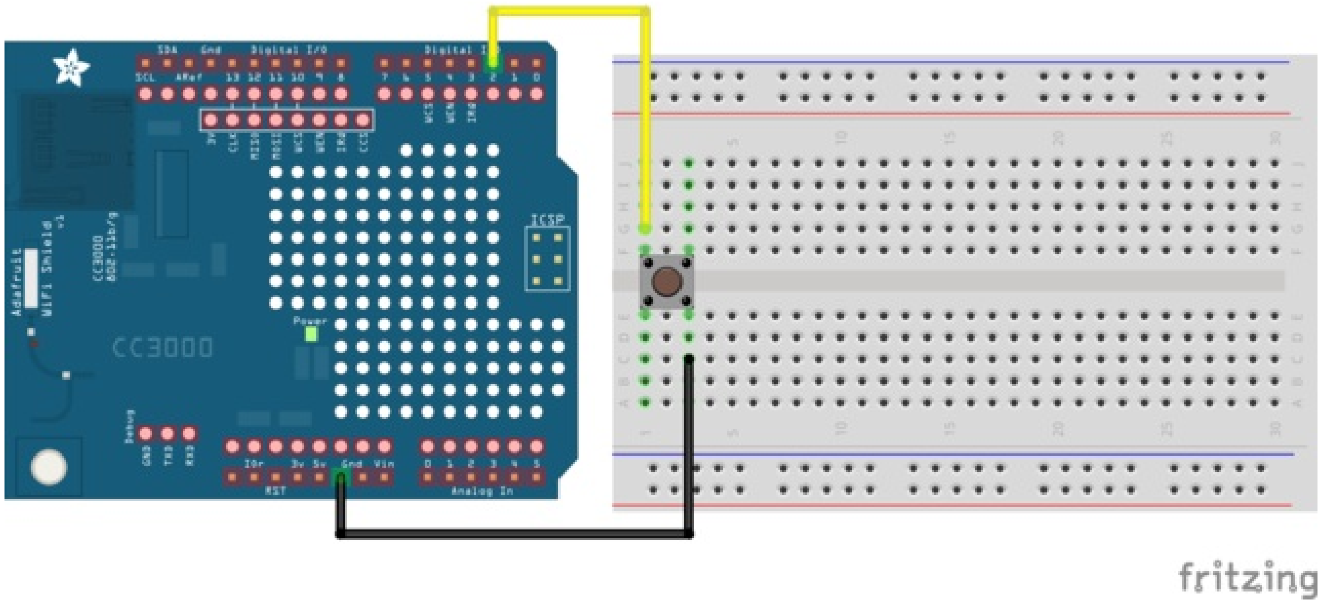
Next, we will look at how to connect an Arduino via WiFi to Adafruit IO.

## Arduino & CC3000

We will be using an Arduino Uno and a CC3000 WiFi shield for this section. If you have not went through the CC3000 setup guide (http://adafru.it/e7F) yet, it would be good to do that first before continuing with this section. A basic understanding of the CC3000 will make this tutorial much easier.

### Wiring

The wiring for this project only consists of wiring one side of a momentary button to **Gnd** on the Uno, and the other side to **digital pin 2** on the Uno.



### Arduino Library Dependencies

You will need the following Arduino libraries installed to compile the example sketch:

- Adafruit MQTT Library (http://adafru.it/fp6)
- Adafruit CC3000 Library (http://adafru.it/cHe)
- Adafruit SleepyDog Library (http://adafru.it/fp8)

The easiest way to install them is by using the Arduino IDE v.1.6.4+ Library Manager (http://adafru.it/fCN).

### Arduino Sketch

The Arduino sketch for this project is fairly straight forward. If you haven't downloaded or cloned the Adafruit IO Basics GitHub repo (http://adafru.it/f27), you should do that first. We will be using the digital in sketch located in the Arduino CC3000 folder.

Adafruit IO Basics Sketches

http://adafru.it/f28

The first thing you will need to do is to edit the WiFi connection info at the top of the CC3000 digital in sketch. You can use the same connection info you used when you tested your WiFi connection in the CC3000 setup guide (http://adafru.it/e7F).

```
#define WLAN_SSID       "...your SSID..."
#define WLAN_PASS       "...your password..."
#define WLAN_SECURITY   WLAN_SEC_WPA2
```

Next, you should replace the Adafruit IO username and key placeholders in the sketch with your username and the key that you retrieved in the Adafruit IO Setup section of this guide.

```
#define AIO_USERNAME    "...your AIO username..."
#define AIO_KEY         "...your AIO key..."
```

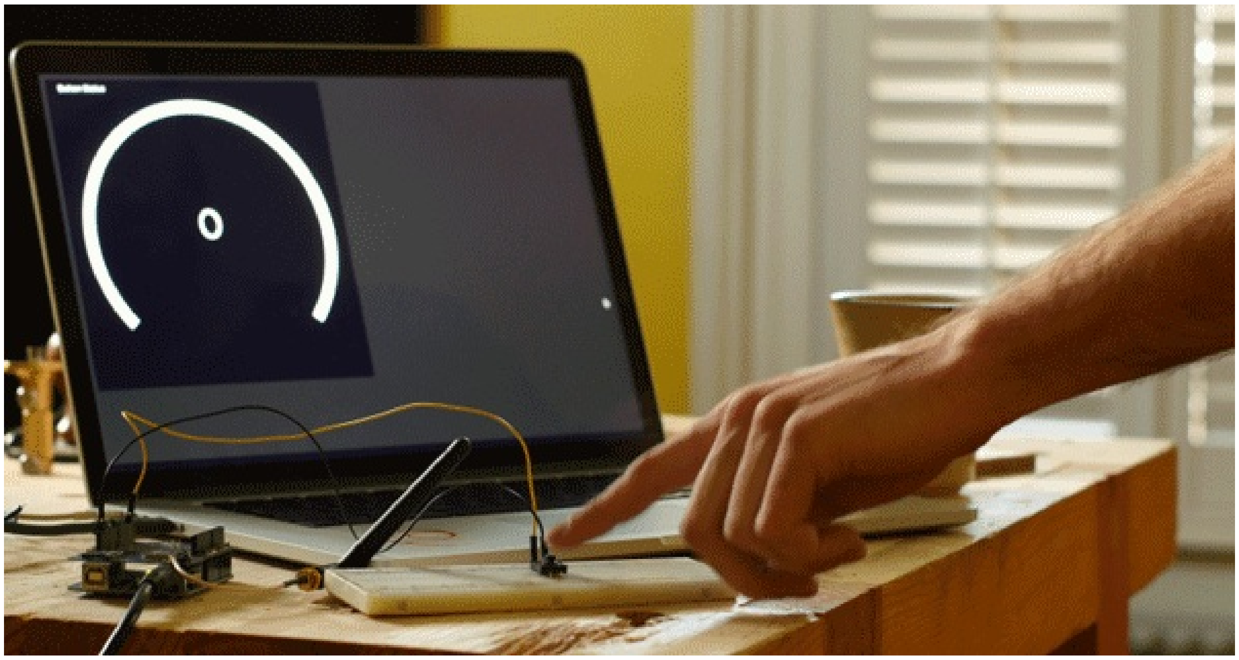
You will then need to check that the name of your feed matches the feed defined in the sketch.

```
// Setup a feed called 'button' for publishing changes.
// Notice MQTT paths for AIO follow the form: <username>/feeds/<feedname>
const char BUTTON_FEED[] PROGMEM = AIO_USERNAME "/feeds/button";
Adafruit_MQTT_Publish button = Adafruit_MQTT_Publish(&mqtt, BUTTON_FEED);
```

The bulk of this sketch is fairly similar to the button example sketch included with the Arduino IDE. The only main difference is that we send data using MQTT (http://adafru.it/f29) to Adafruit IO.

```
void loop() {

  // Make sure to reset watchdog every loop iteration!
  Watchdog.reset();

  // ping adafruit io a few times to make sure we remain connected
  if(! mqtt.ping(3)) {
    // reconnect to adafruit io
    if(! mqtt.connected())
      connect();
  }

  // grab the current state of the button
  current = digitalRead(BUTTON);

  // return if the value hasn't changed
  if(current == last)
    return;

  int32_t value = (current == LOW ? 1 : 0);

  // Now we can publish stuff!
  Serial.print(F("\nSending button value: "));
  Serial.print(value);
  Serial.print("... ");

  if (! button.publish(value))
    Serial.println(F("Failed."));
  else
    Serial.println(F("Success!"));

  // save the button state
  last = current;

}
```

When you are finished reviewing the sketch and have finished making the necessary config changes, upload the sketch to your Uno using the Arduino IDE. You should also open up your Adafruit IO dashboard so you can monitor your button gauge.



If everything goes as expected, you will see the gauge update on Adafruit IO. You should make sure to open the Arduino IDE's serial monitor if you are having issues with the sketch. It will provide valuable debugging info.

Next, we will look at accomplishing the same task using the Adafruit FONA cellular module instead of WiFi.

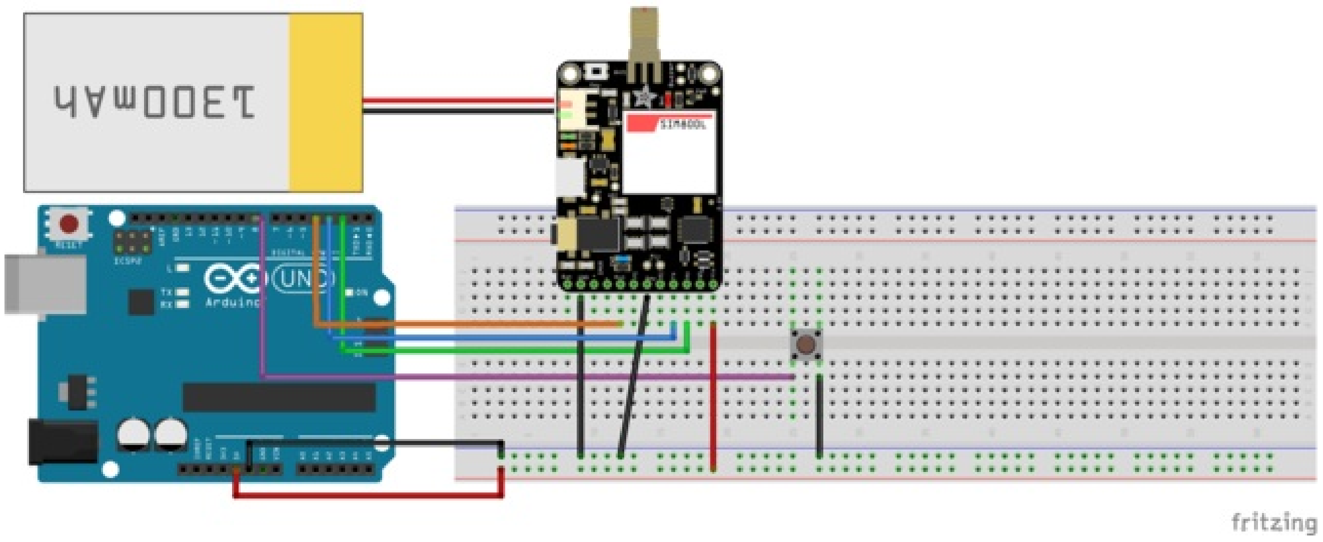
# Arduino & FONA Cellular

The cellular example sketch uses the Adafruit FONA and Arduino Uno to send values to Adafruit IO. If you need basic info about setting up your FONA, please check out our FONA guide (http://adafru.it/f22). First, let's take a look at wiring up the FONA and momentary button to the Uno.

## Wiring

First, connect the FONA's **Key** and **GND** pins to **Ground** on your breadboard. Connect the **FONA Rst** pin to **Uno digital pin 4.** Connect the **FONA TX** pin to **Uno digital pin 3**, and the **FONA RX** pin to **Uno digital pin 2**. Connect the **FONA Vio** pin to **5v** on your breadboard. Connect the **Uno 5v pin** to the **5v rail** on your breadboard, and one of the **Uno Gnd pins** to the **Ground rail** on your breadboard. Make sure you have a charged lithium ion battery plugged into the FONA

Next, connect one side of the momentary button to the Uno's **pin 8** and the other to breadboard **Ground**.



## Arduino Library Dependencies

You will need the following Arduino libraries installed to compile the example sketch:

- Adafruit FONA (http://adafru.it/dDD)
- Adafruit SleepyDog Library (http://adafru.it/fp8)
- Adafruit MQTT Library (http://adafru.it/fp6)

The easiest way to install them is by using the Arduino IDE v.1.6.4+ Library Manager (http://adafru.it/fCN).

## Arduino Sketch

The Arduino sketch for this project is fairly straight forward. If you haven't downloaded or cloned the Adafruit IO Basics GitHub repo (http://adafru.it/f27), you should do that first. We will be using the **digital-in** sketch located in the Arduino FONA folder.

Adafruit IO Basics Sketches

http://adafru.it/f28

Next, you should replace the Adafruit IO username and key placeholders in the sketch with your username and the key that you retrieved in the Adafruit IO Setup section of this guide.

```
#define AIO_USERNAME    "...your AIO username..."
#define AIO_KEY         "...your AIO key..."
```

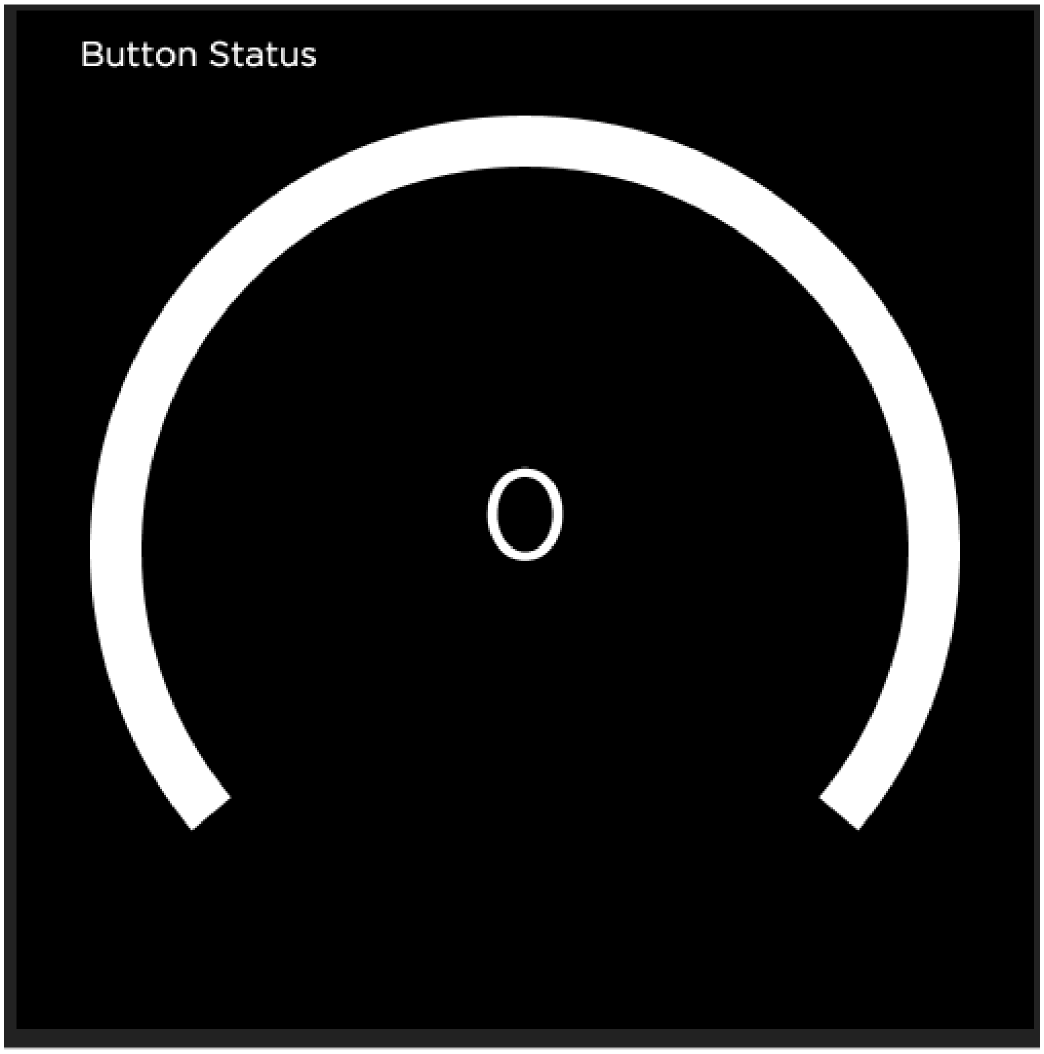
You will then need to check that the name of your feed matches the feed defined in the sketch.

```
// Setup a feed called 'button' for publishing changes.
// Notice MQTT paths for AIO follow the form: <username>/feeds/<feedname>
const char BUTTON_FEED[] PROGMEM = AIO_USERNAME "/feeds/button";
Adafruit_MQTT_Publish button = Adafruit_MQTT_Publish(&mqtt, BUTTON_FEED);
```

The bulk of this sketch is fairly similar to the button example sketch included with the Arduino IDE. The only main difference is that we will be sending data via MQTT (http://adafru.it/f29) to Adafruit IO using the FONA.

```
void loop() {

  // Make sure to reset watchdog every loop iteration!
  Watchdog.reset();

  // ping adafruit io a few times to make sure we remain connected
  if(! mqtt.ping(3)) {
    // reconnect to adafruit io
    if(! mqtt.connected())
      connect();
  }

  // grab the current state of the button
  current = digitalRead(BUTTON);

  // return if the value hasn't changed
  if(current == last)
    return;

  int32_t value = (current == LOW ? 1 : 0);

  // Now we can publish stuff!
  Serial.print(F("\nSending button value: "));
  Serial.print(value);
  Serial.print("... ");

  if (! button.publish(value))
    Serial.println(F("Failed."));
  else
    Serial.println(F("Success!"));

  // save the button state
  last = current;

}
```

When you are finished reviewing the sketch and have finished making the necessary config changes, upload the sketch to your Uno using the Arduino IDE. You should also open up your Adafruit IO dashboard so you can monitor the button gauge.



If everything goes as expected, you will see the gauge update on Adafruit IO. The gauge will not be as responsive as the WiFi example because of the slower cellular network speed. You should make sure to open the Arduino IDE's serial monitor if you are having issues with the sketch. It will provide valuable debugging info.

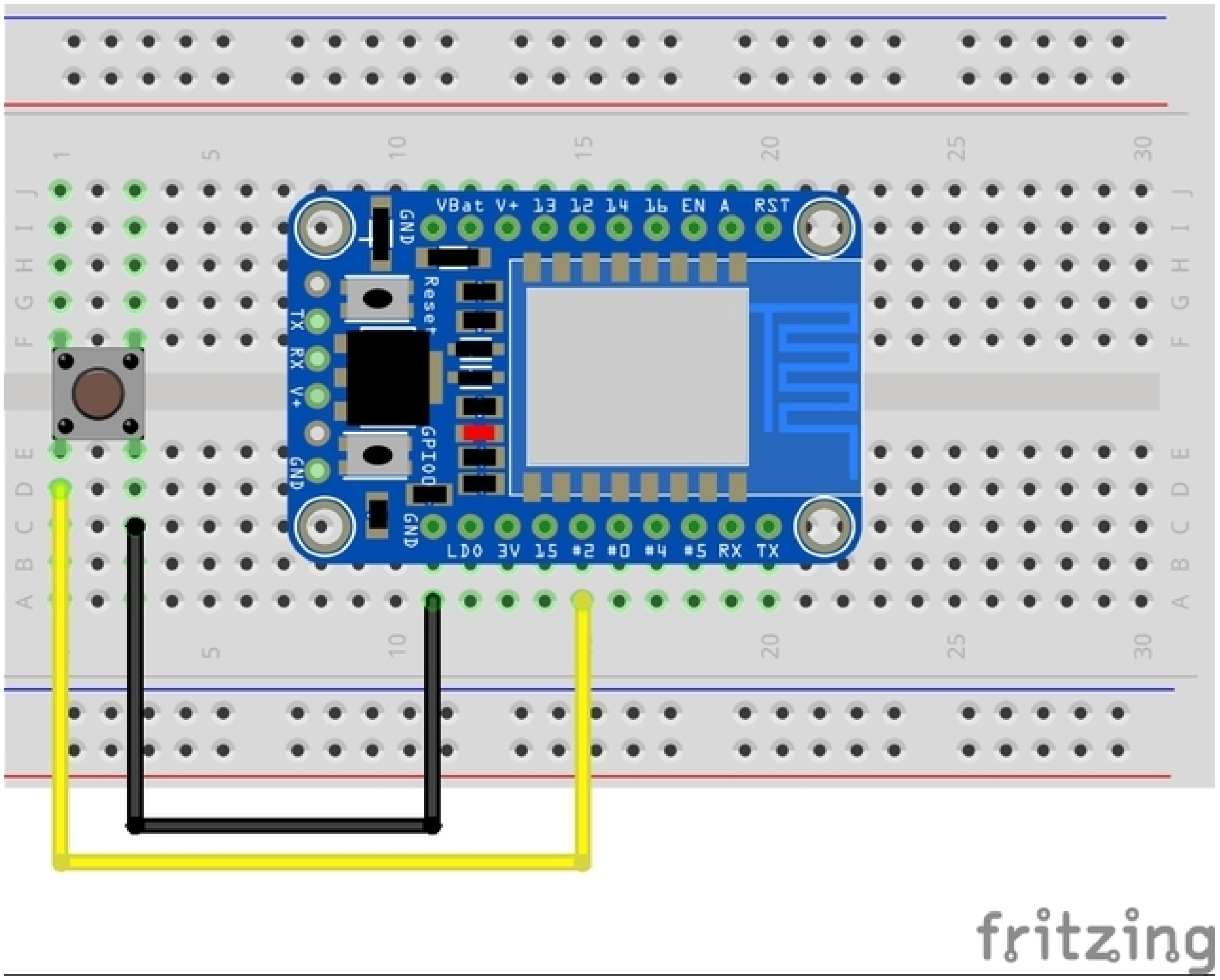
Next, we will look at accomplishing the same task using a HUZZAH ESP8266.

# ESP8266

This example uses the Adafruit HUZZAH ESP8266 to send values to Adafruit IO. If you need basic info about setting up your HUZZAH, please check out our HUZZAH ESP8266 setup guide (http://adafru.it/fCM). First, let's take a look at wiring up the HUZZAH to the momentary button.

## Wiring

The wiring for this project only consists of wiring one side of a momentary button to **Gnd** on the HUZZAH, and the other side of the button to **digital pin 2** on the HUZZAH.



## Arduino Dependencies

You will need the following Arduino library installed to compile the example sketch:

- Adafruit MQTT Library (http://adafru.it/fp6)

The easiest way to install it is by using the Arduino IDE v.1.6.4+ Library Manager (http://adafru.it/fCN). You will also need to have the ESP8266 board manager package installed. For more info about installing the ESP8266 package, visit the HUZZAH ESP8266 setup guide (http://adafru.it/fCM).

```
http://arduino.esp8266.com/stable/package_esp8266com_index.json
```

Use the package index URL above to get the latest version of the ESP8266 package.

## Arduino Sketch

The Arduino sketch for this project is fairly straight forward. If you haven't downloaded or cloned the Adafruit IO Basics GitHub repo (http://adafru.it/f27), you should do that first. We will be using the digital in sketch located in the ESP8266 folder.

Adafruit IO Basics Sketches

http://adafru.it/f28

The first thing you will need to do is to edit the WiFi connection info at the top of the ESP8266 *digital-in* sketch. You can use the same connection info you used when you tested your WiFi connection in the  (http://adafru.it/e7F)HUZZAH ESP8266 setup guide (http://adafru.it/fCM).

```
#define WLAN_SSID       "...your SSID..."
#define WLAN_PASS       "...your password..."
```

Next, you should replace the Adafruit IO username and key placeholders in the sketch with your username and the key that you retrieved in the Adafruit IO Setup section of this guide.

```
#define AIO_USERNAME    "...your AIO username..."
#define AIO_KEY         "...your AIO key..."
```

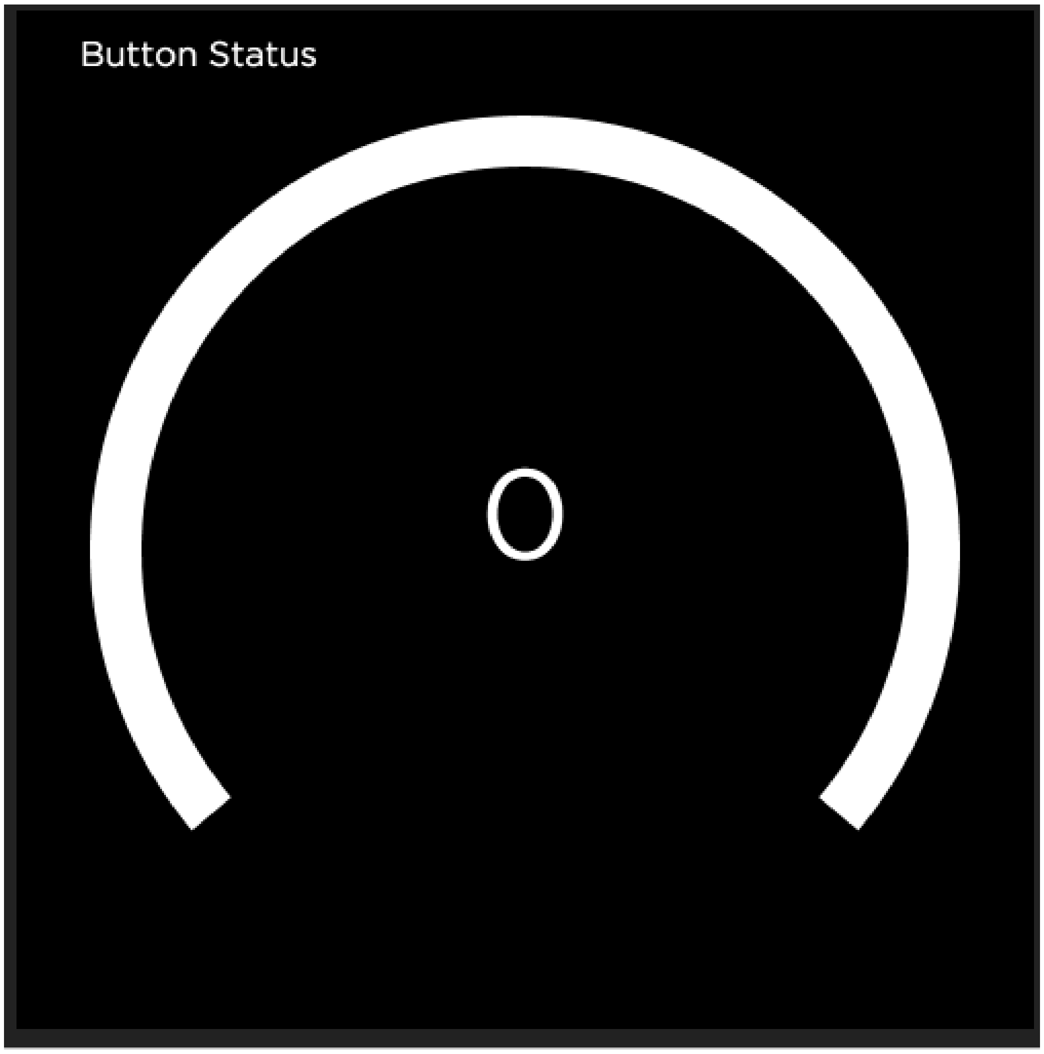
You will then need to check that the name of your feed matches the feed defined in the sketch.

```
// Setup a feed called 'button' for publishing changes.
// Notice MQTT paths for AIO follow the form: <username>/feeds/<feedname>
const char BUTTON_FEED[] PROGMEM = AIO_USERNAME "/feeds/button";
Adafruit_MQTT_Publish button = Adafruit_MQTT_Publish(&mqtt, BUTTON_FEED);
```

The bulk of this sketch is fairly similar to the button example sketch included with the Arduino IDE. The only main difference is that we will be sending data via MQTT (http://adafru.it/f29) to Adafruit IO using the ESP8266.

```
void loop() {

  // ping adafruit io a few times to make sure we remain connected
  if(! mqtt.ping(3)) {
    // reconnect to adafruit io
    if(! mqtt.connected())
      connect();
  }

  // grab the current state of the button
  current = digitalRead(BUTTON);

  // return if the value hasn't changed
  if(current == last)
    return;

  int32_t value = (current == LOW ? 1 : 0);

  // Now we can publish stuff!
  Serial.print(F("\nSending button value: "));
  Serial.print(value);
  Serial.print("... ");

  if (! button.publish(value))
    Serial.println(F("Failed."));
  else
    Serial.println(F("Success!"));

  // save the button state
  last = current;

}
```

When you are finished reviewing the sketch and have finished making the necessary config changes, upload the sketch to your HUZZAH using the Arduino IDE. You should also open up your Adafruit IO dashboard so you can monitor the button gauge.



If everything goes as expected, you will see the gauge update on Adafruit IO. You should make sure to open the Arduino IDE's serial monitor if you are having issues with the sketch. It will provide valuable debugging info.

Next, we will look at accomplishing the same task using a Raspberry Pi.

## Raspberry Pi

The first requirement of this project is that you have internet access from your Raspberry Pi. It can either be through an ethernet connection, or through WiFi using a USB dongle. If you need help getting WiFi set up, you can check out our Raspberry Pi Finder guide (http://adafru.it/f23), which includes a helpful utility for setting up WiFi on your Pi.

### Command Line Access

Next, you will need to make sure you have access to the command line on your Raspberry Pi. There are quite a few ways to accomplish this, and this handy guide (http://adafru.it/eUN) will help you get started if you need assistance.

### Installing Dependencies

The next step will be to install the latest version of Node.js. We have created a short guide that covers installing Node.js (http://adafru.it/eAZ) if you need help with the install process. You can check the version of node you have installed by running **node -v** from the command line on your Pi. You should have *v0.12.0* or higher installed.

```
pi@raspberrypi ~ $ node -v
v0.12.0
```

Next, you will need to get a copy of the Adafruit IO Basics example scripts. The easiest way to do that is to clone the GitHub repository using *git*.

```
git clone https://github.com/adafruit/adafruit-io-basics.git
```

Next, you will need to navigate to the *raspberry-pi* directory inside of the *adafruit-io-basics* folder.

```
cd adafruit-io-basics/raspberry-pi
```

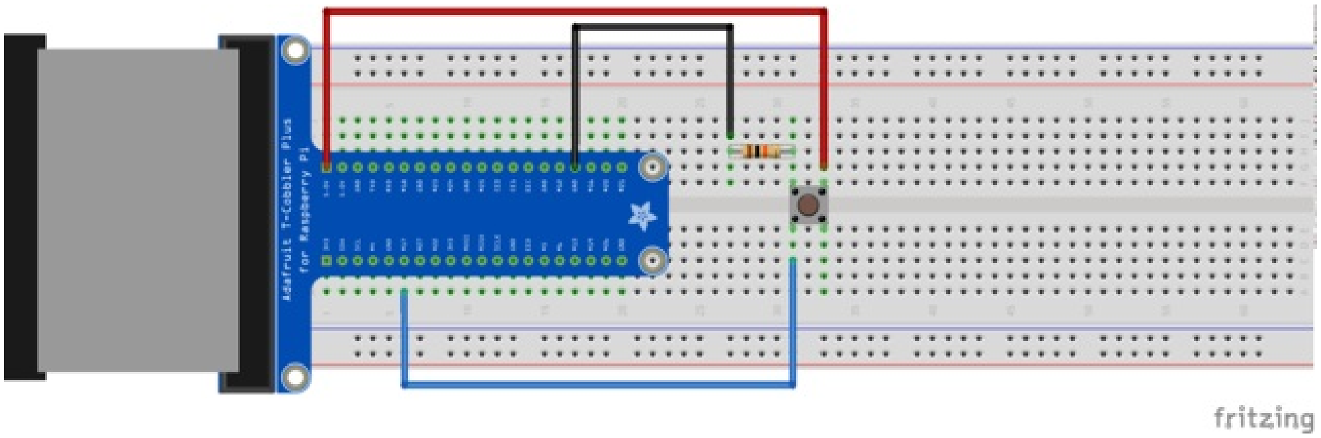
Next, you will need to install the Node.js library dependencies using *npm*.

```
npm install
```

### Wiring

We will be using the Adafruit T-Cobbler Plus to make it easier to wire the button to the Pi. First, attach the T-Cobbler to your Pi using the supplied ribbon cable.

Next, insert the T-Cobbler and momentary button into your breadboard using the diagram below as a reference. Wire **pin 17** from the T-Cobbler to one side of the button, and **5.0V** from the T-Cobbler to the other side of the button. Also attach a **10k ohm resistor** to the same side of the button as **pin 17**, and the other side of the resistor to **Gnd** on the T-Cobbler.



## Node.js Script

The *digital-in.js* script in the *raspberry-pi* folder of the repo is very simple. It uses the Node.js Stream API (http://adafru.it/f2d) to pipe button press data to Adafruit IO.

```
var GpioStream = require('gpio-stream'),
    button = GpioStream.readable(17),
    AIO = require('adafruit-io');

// replace xxxxxxxxxxx with your Adafruit IO key
var AIO_KEY = 'xxxxxxxxxxx',
    AIO_USERNAME = 'your_username';

// aio init
var aio = AIO(AIO_USERNAME, AIO_KEY);

// pipe button presses to the button feed
button.pipe(aio.feeds('Button'));

console.log('listening for button presses...');
```

The only modification to the example you will need to make is to add your Adafruit IO key to the script. You can do this using the **nano** text editor.

```
nano digital-in.js
```

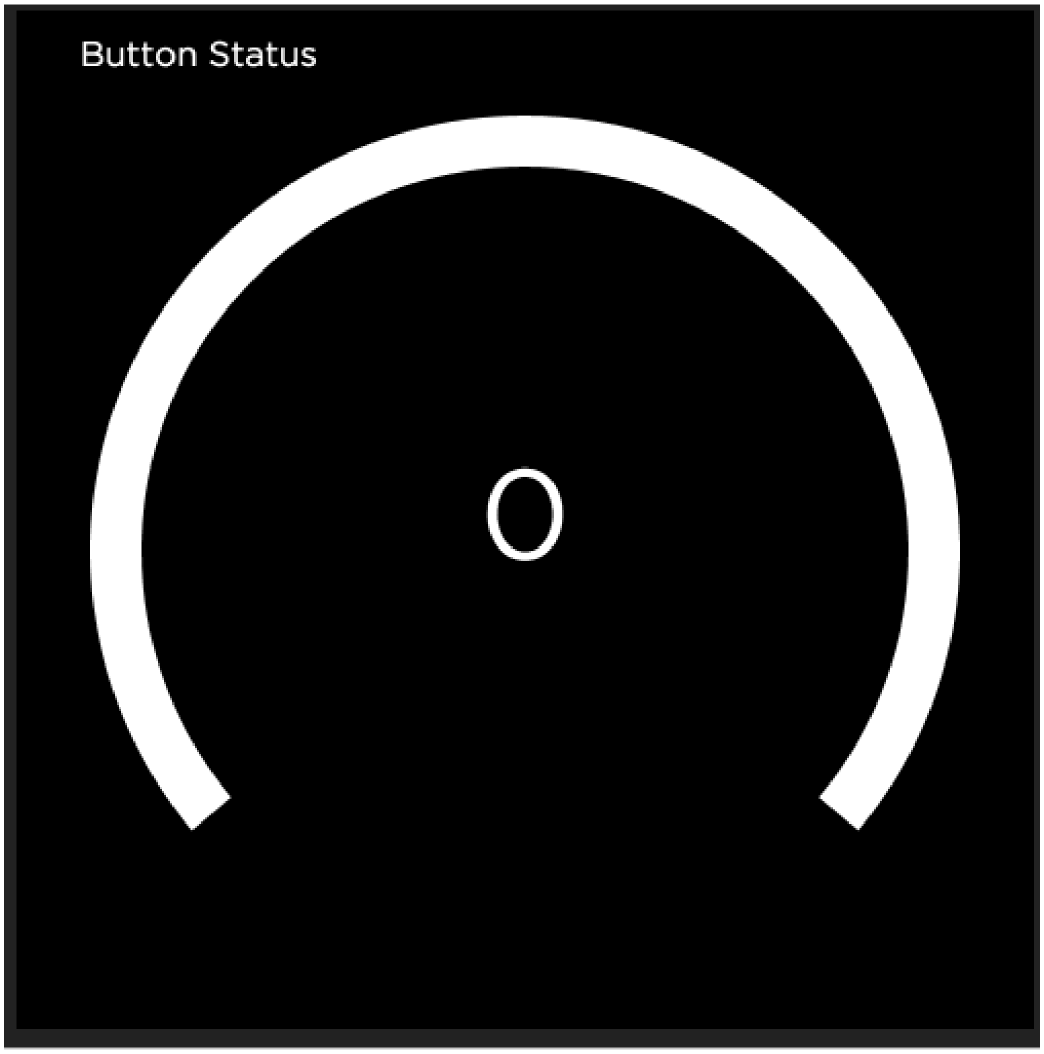
Edit the *AIO_KEY* variable and replace the placeholder with your Adafruit IO key, and the AIO_USERNAME variable with your Adafruit IO username.

```
var AIO_KEY = 'xxxxxxxxxxxx',
    AIO_USERNAME = 'your_username';
```

**Ctrl** + **X** followed by **Y** and then **Return** will save the file. You are now ready to run the script. You can do this by running **node digital-in.js**.

```
$ node digital-in.js
listening for button presses...
```

Make sure you have your Adafruit IO dashboard open, and you should be able to see the button presses on your gauge.



You can press **Ctrl + C** to quit the script.