Table of Contents

Overview

Adafruit IO Setup
• Creating the Analog Feed
• Adding the Slider Block

Arduino Wiring
• Arduino Wiring

Arduino Setup

Arduino Network Config
• WiFi Config
• FONA Config
• Ethernet Config

Arduino Code

Python Wiring

Python Setup
• Enable I2C
• Installing the CircuitPython-PCA9685 Library

Python Code
• Code

Adafruit IO FAQ
• Encountering an issue with your Adafruit IO Arduino Project?
Overview

This guide is part of a series of guides that cover the basics of using Adafruit IO. It will show you how to dim a LED from Adafruit IO using any modern web browser.

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
- Adafruit IO Basics: Dashboards

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 Feather HUZZAH ESP8266 Setup Guide

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 of the hardware choices for this project. Let's get started!

Adafruit IO Setup

The first thing you will need to do is to login to Adafruit IO and visit the Settings page.

Click the VIEW AIO KEY button 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.

Creating the Analog Feed

Next, you will need to create a feed called Analog. If you need help getting started with creating feeds on Adafruit IO, check out the Adafruit IO Feed Basics guide.
Adding the Slider Block

Next, add a new Slider Block to a new or existing dashboard. Name the block whatever you would like, and set min value to 0 and max value to 1024. Make sure you have selected the Analog feed as the data source for the slider.

If you need help getting started with Dashboards on Adafruit IO, check out the Adafruit IO Dashboard Basics guide.

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

Next, we will look at wiring the circuit.
You will need the following parts for this tutorial:

- 1x Adafruit IO compatible Feather
- 2x jumper wires
- 1x 560 ohm resistor
- 1x 10mm LED

You will need to connect the following pins to the LED and resistor:

- Feather GND to LED cathode (short leg)
- Feather Pin 5 to one leg of the 560 ohm resistor
- LED anode (long leg) to the second leg of the 560 ohm resistor

Note: Resistors are not polarized, so the 560 ohm resistor can be connected to the circuit in either direction.

Next, we're going to set up our Arduino.
Arduino Setup

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 Feather HUZZAH ESP8266 Setup Guide

You will need to make sure you have at least version 2.3.1 of the Adafruit IO Arduino library installed before continuing.

For this example you will need to open the adafruitio_09_analog_out example in the Adafruit IO Arduino library.

Next, we will look at the network configuration options in the sketch.

Arduino Network Config

To configure the network settings, click on the config.h tab in the sketch. You will need to set your Adafruit IO username in the IO_USERNAME define, and your Adafruit IO key in the IO_KEY define.
WiFi Config

WiFi is enabled by default in config.h so if you are using one of the supported WiFi boards, you will only need to modify the WiFi_SSID and WiFi_PASS options in the config.h tab.

```c
// visit io.adafruit.com to create an account, or if you need your Adafruit IO key.
#define IO_USERNAME "uniontownlabs"
#define IO_KEY ""

// the AdafruitIO_WiFi client will work with the following boards:
// - Feather HUZZAH ESP8266 -> https://www.adafruit.com
// - Feather M0 WiFi -> https://www.adafruit.com/product/1292
// - Feather WICED 3 -> https://www.adafruit.com/products/4056
#define WIFI_SSID "Test WiFi"
#define WIFI_PASS "my wifi password"

// comment out the following two lines if you are using fona or ethernet
#include "AdafruitIO_WiFi.h"
AdafruitIO_WiFi io(IO_USERNAME, IO_KEY, WIFI_SSID, WIFI_PASS);
```

FONA Config

If you wish to use the FONA 32u4 Feather to connect to Adafruit IO, you will need to first comment out the WiFi support in config.h

```c
// the AdafruitIO_WiFi client will work with the following boards:
// - Feather HUZZAH ESP8266 -> https://www.adafruit.com
// - Feather M0 WiFi -> https://www.adafruit.com/product/1292
// - Feather WICED 3 -> https://www.adafruit.com/products/4056
#define WIFI_SSID "Test WiFi"
#define WIFI_PASS "my wifi password"

// comment out the following two lines if you are using fona or ethernet
#include "AdafruitIO_WiFi.h"
AdafruitIO_WiFi io(IO_USERNAME, IO_KEY, WIFI_SSID, WIFI_PASS);
```

Next, remove the comments from both of the FONA config lines in the FONA section of config.h to enable FONA support.
Ethernet Config

If you wish to use the Ethernet Wing to connect to Adafruit IO, you will need to first comment out the WiFi support in config.h

```c
#include "AdafruitIO_FONA.h"
AdafruitIO_FONA io(IO_USERNAME, IO_KEY);
```

Next, remove the comments from both of the Ethernet config lines in the Ethernet section of config.h to enable Ethernet Wing support.

```c
#include "AdafruitIO_WiFi.h"
AdafruitIO_WiFi io(IO_USERNAME, IO_KEY, WIFI_SSID, WIFI_PASS);
```

Next, we will look at how the example sketch works.

Arduino Code

The adafruitio_09_analog_out example uses pin 5 by default, and that can be modified by changing the LED_PIN define at the top of the sketch. This pin should correspond to a pin on your feather with PWM capability.
// this should correspond to a pin with PWM capability
#define LED_PIN 5

The next chunk of code sets up an Adafruit IO Feed instance for a feed called analog.

// set up the 'analog' feed
AdafruitIO_Feed *analog = io.feed("analog");

In the setup function, we attach a function called handleMessage to the analog feed that will be called whenever your device receives messages for that feed.

The code will wait until you have a valid connection to Adafruit IO before continuing with the sketch. If you have any issues connecting, check config.h for any typos in your username or key.

void setup() {
    // start the serial connection
    Serial.begin(115200);

    // wait for serial monitor to open
    while(! Serial);

    // connect to io.adafruit.com
    Serial.print("Connecting to Adafruit IO");
    io.connect();

    // set up a message handler for the 'analog' feed.
    // the handleMessage function (defined below)
    // will be called whenever a message is received from adafruit io.
    analog-&gt;onMessage(handleMessage);

    // wait for a connection
    while(io.status() &lt; AIO_CONNECTED) {
        Serial.print(".");
        delay(500);
    }

    // we are connected
    Serial.println();
    Serial.println(io.statusText());
}

Next, we have the main loop() function. The first line of the loop function calls io.run(); this line will need to be present at the top of your loop in every sketch. It helps keep your device connected to Adafruit IO, and processes any incoming data.

void loop() {
    // io.run(); is required for all sketches.
    // it should always be present at the top of your loop
    // function. it keeps the client connected to
    // io.adafruit.com, and processes any incoming data.
    io.run();
}
The final chunk of code is the handleMessage function. This is the function that is called whenever the analog feed gets a message.

We use the `data->toInt()` function to convert the incoming data to an int, and set the state of the LED_PIN to that value using `analogWrite()`.

```c
// this function is called whenever an 'analog' message
// is received from Adafruit IO. it was attached to
// the analog feed in the setup() function above.
void handleMessage(AdafruitIO_Data *data) {

    // convert the data to integer
    int reading = data-&gt;toInt();
    Serial.print("received &lt; - ");
    Serial.println(reading);
    analogWrite(LED_PIN, reading);
}
```

Upload the sketch to your board, and open the Arduino Serial Monitor. Your board should now connect to Adafruit IO.

Connecting to Adafruit IO....
Adafruit IO connected.

Change the value of the slider on your Adafruit IO dashboard, and you should see something resembling the following in the Arduino Serial Monitor.

```
received &lt;- 940
received &lt;- 290
received &lt;- 230
received &lt;- 110
received &lt;- 0
received &lt;- 90
received &lt;- 320
received &lt;- 630
received &lt;- 840
received &lt;- 1020
```

You should also see your LED change brightness depending on the value you send.
Python Wiring

We're going to use a combination of the Adafruit IO Client Library and Adafruit's CircuitPython to control a Raspberry Pi over Adafruit IO.

1 x Raspberry Pi 3 - Model B+  
The Raspberry Pi is a small linux board compatible with Adafruit IO projects.  
https://www.adafruit.com/product/3775

If you're following along with a Raspberry Pi, we're going to use a T-Cobbler Plus for the IO Basics Projects. This add-on prototyping board lets you easily connect a Raspberry Pi (Raspberry Pi Model Zero, A+, B+, Pi 2, Pi 3) to a solderless breadboard:

1 x Assembled Pi T-Cobbler Plus  
GPIO Breakout - Pi A+, B+, Pi 2, Pi 3, Zero.  
https://www.adafruit.com/product/2028

Want to create an automatic fish-feeder, a door-lock system with vibration-feedback, or maybe you want to just chain a bunch of lights and motors together and control them with Adafruit IO?

You'll need a few PWM outputs.

The Raspberry Pi is limited to one PWM output. While we could use this PWM output for the servo, we're going to use the Adafruit 16-Channel 12-bit PWM/Servo driver. This board can be used to control up to 16 PWM outputs. This means you can have a bunch of servos, DC motors, LED lights, or even a combination of both.

1 x Adafruit 16-Channel 12-bit PWM/Servo Driver  
Adafruit PCA9685 Breakout.  
https://www.adafruit.com/product/815
You will need to make the following connections between the Pi and the PCA9685:

- Pi 3.3V to PCA9685 VCC
- Pi GND to PCA9685 GND
- Pi SDA to PCA9685 SDA
- Pi SCL to PCA9685 SCL

Make the remaining connections:

- PCA9685 PWM Pin (on channel 15) to one leg of the 560 ohm resistor
- Pi GND to LED Cathode (short LED leg)
- LED Anode (long LED leg) to the other leg of the 560 ohm resistor

  ◦ note: resistors are not polarized, so the 560 ohm resistor can be connected to the circuit in either direction.

Next, we're going to set up our Pi.

Python Setup

The latest Raspbian (currently this is `Stretch`) is required for the installation of Adafruit IO + Blinka.

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 (named after Blinka, the CircuitPython mascot) 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](#)

## Enable I2C

We use two pins on the Pi (SDA/SCL) to communicate over I2c with the PCA9685. You only have to do this step once per Raspberry Pi, the I2C interface is disabled by default.

- [Enabling I2C](#)

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

```
sudo i2cdetect -y 1
```

If your PCA9685 Breakout is wired up correctly, it'll show up at 0x40:

![i2cdetect output]

## Installing the CircuitPython-PCA9685 Library

You'll also need to install a library to communicate with the PWM breakout. Since we're using Adafruit Blinka (CircuitPython), we can install CircuitPython libraries straight to our Raspberry Pi. In this case, we're going to install the CircuitPython-PCA9685 library.

Run the following command to install the CircuitPython-PCA9685 library:
pip3 install adafruit-circuitpython-PCA9685

Python Code

Before you run the code, you'll need to set `ADAFRUIT_IO_KEY` and `ADAFRUIT_IO_USERNAME` to the key and username associated with your Adafruit IO account.

```python
# Set to your Adafruit IO username.
# (go to https://accounts.adafruit.com to find your username)
ADAFRUIT_IO_USERNAME = 'YOUR_USERNAME'

# Create an instance of the REST client.
aio = Client(ADAFRUIT_IO_USERNAME, ADAFRUIT_IO_KEY)

The analog_output example uses the PCA9685's channel 15 by default. If you wish to change it, you can modify the `PCA_CHANNEL` variable at the top of the code to correspond with a different PWM channel of the PCA9685.

```python
# Set the PCA channel to use.
PCA_CHANNEL = 15
```

Inside the `while True` loop, we grab the analog feed value and compare it to the previous analog feed value.

We want the LED's brightness to remain set to the previous feed value, unless changed.

To do this, we first check it's a new value (the values are not equal). Then, we print that we've received a new value, map the feed range to a write-able value range, and set the PCA's channel to this value. If we don't receive a new value, we simply store the previous value in the `prev_read` variable.

```python
while True:
    # grab the `analog` feed value
    analog_read = aio.receive(analog.key)
    if (analog_read.value != prev_read):
        print('received <- ', analog_read.value)
        # map the analog value from 0 - 1023 to 0 - 65534
        analog_value = map_range(int(analog_read.value), 0, 1024, 0, 65534)
        # set the LED to the mapped feed value
        pca.channels[PCA_CHANNEL].duty_cycle = int(analog_value)
        prev_read = analog_read.value
```

Run the code by entering the following in your terminal:

```
python3 analog_output.py
```
Change the value of the slider on your Adafruit IO dashboard, and you should see something resembling the following in the terminal:

```bash
$ python3 analog-out.py
received &lt;- 300
received &lt;- 500
received &lt;- 100
```

You should also see your LED change brightness depending on the value you send.

**Code**

```python
# Set to your Adafruit IO key.  # Remember, your key is a secret, # so make sure not to publish it when you publish this code!
ADAFRUIT_IO_KEY = 'YOUR_IO_KEY'

# Set to your Adafruit IO username.  # (go to https://accounts.adafruit.com to find your username)
ADAFRUIT_IO_USERNAME = 'YOUR_IO_USERNAME'

# Create the I2C bus interface.
i2c_bus = I2C(SCL, SDA)

# Create a simple PCA9685 class instance.
 pca = PCA9685(i2c_bus)
```
PCA_CHANNEL = 4

# Set the PWM frequency to 60hz.
pca.frequency = 60
prev_read = 0

# Create an instance of the REST client.
aio = Client(ADAFRUIT_IO_USERNAME, ADAFRUIT_IO_KEY)

try:  # if we have a 'analog' feed
    analog = aio.feeds('analog')
except RequestError:  # create an `analog` feed
    feed = Feed(name='analog')
    analog = aio.create_feed(feed)

def map_range(x, in_min, in_max, out_min, out_max):
    """re-maps a number from one range to another."""
    mapped = (x-in_min) * (out_max - out_min) / (in_max-in_min) + out_min
    if out_min <= out_max:
        return max(min(mapped, out_max), out_min)
    return min(max(mapped, out_max), out_min)

while True:
    # grab the `analog` feed value
    analog_read = aio.receive(analog.key)
    if analog_read.value != prev_read:
        print('received <- ', analog_read.value)
        # map the analog value from 0 - 1023 to 0 - 65534
        analog_value = map_range(int(analog_read.value), 0, 1024, 0, 65534)
        # set the LED to the mapped feed value
        pca.channels[PCA_CHANNEL].duty_cycle = int(analog_value)
        prev_read = analog_read.value
    # timeout so we don't flood IO with requests
    time.sleep(0.5)

---

**Adafruit IO FAQ**

**Encountering an issue with your Adafruit IO Arduino Project?**

If you're having an issue compiling, connecting, or troubleshooting your project, check this page first.

Don't see your issue? [Post up on the Adafruit IO Forum with your issue](https://www.adafruit.com/forum/).

I encounter the following error when compiling my sketch:

```
fatal error: Adafruit_MQTT.h: No such file or directory, #include "Adafruit_MQTT.h"
```

The Adafruit IO Arduino library is dependent on our Adafruit IO MQTT Library.
My Serial Monitor prints "..." endlessly after the "Connecting to Adafruit IO" message

Your board is not connecting to Adafruit IO, but why? Let's find out:

First, check in  config.h  that you have the correct IO_USERNAME, IO_KEY, WIFI_SSID, and WIFI_PASS are set correctly.

Next, we're going to modify the while loop which waits for an IO connection in your sketch. Change the line in the status check loop from Serial.println(.); to Serial.println(io.statusText());

```c
// wait for a connection
while(io.status() < AIO_CONNECTED) {
    Serial.println(io.statusText());
    delay(500);
}
```
Verify and re-upload the sketch. If you're receiving a Network disconnected error message, the board is not able to talk to the internet. Re-check your hardware, connections, and router settings.

If it's still not showing Adafruit IO connected, check the IO status on the Adafruit Status page (?) to make sure the service is online.

My data isn't displaying, is Adafruit IO's service/MQTT/API down?

 Possibly - you can check IO status on the Adafruit Status page (?) .

Is my data being sent properly? Am I sending too much data?

 There's a monitor page built-into Adafruit IO (?) which provides a live view of incoming data and error messages. Keep this page open while you send data to your Adafruit IO devices to monitor data and errors.