# Table of Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Overview</strong></td>
<td>3</td>
</tr>
<tr>
<td><strong>Adafruit IO Setup</strong></td>
<td>3</td>
</tr>
<tr>
<td>- Creating the Color Feed</td>
<td></td>
</tr>
<tr>
<td>- Adding the Color Block</td>
<td></td>
</tr>
<tr>
<td><strong>Wiring</strong></td>
<td>5</td>
</tr>
<tr>
<td>- Arduino Wiring</td>
<td></td>
</tr>
<tr>
<td><strong>Arduino Setup</strong></td>
<td>7</td>
</tr>
<tr>
<td><strong>Arduino Network Config</strong></td>
<td>8</td>
</tr>
<tr>
<td>- WiFi Config</td>
<td></td>
</tr>
<tr>
<td>- FONA Config</td>
<td></td>
</tr>
<tr>
<td>- Ethernet Config</td>
<td></td>
</tr>
<tr>
<td><strong>Arduino Code</strong></td>
<td>10</td>
</tr>
<tr>
<td><strong>Python Wiring</strong></td>
<td>13</td>
</tr>
<tr>
<td>- Parts</td>
<td></td>
</tr>
<tr>
<td>- Wiring</td>
<td></td>
</tr>
<tr>
<td><strong>Python Setup</strong></td>
<td>16</td>
</tr>
<tr>
<td>- Enable I2C</td>
<td></td>
</tr>
<tr>
<td>- Installing the CircuitPython-PCA9685 Library</td>
<td></td>
</tr>
<tr>
<td><strong>Python Code</strong></td>
<td>17</td>
</tr>
<tr>
<td>- Code</td>
<td></td>
</tr>
<tr>
<td><strong>Adafruit IO FAQ</strong></td>
<td>20</td>
</tr>
<tr>
<td>- Encountering an issue with your Adafruit IO Arduino Project?</td>
<td></td>
</tr>
</tbody>
</table>
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 color data from Adafruit IO to a RGB LED.

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 Color Feed

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

Next, add a new Color block to a new or existing dashboard. Name the block whatever you would like, but make sure you have selected the Color feed as the data source for the block.

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.

Wiring

You will need the following parts for this tutorial:

- 1x Adafruit IO compatible Feather
• 1x diffused RGB LED - common anode
• 3x 560 ohm resistors
• 4x jumper wires

We will be looking at the LED with the flat side of the LED on the left. If it's hard for you to see which side is the flat side, you can use the long pin as your guide. The long pin will be closer to the flat side of the LED.

Arduino Wiring

We will be using a Feather ESP8266 in this example, but you can adjust the pins in the sketch if you are using a different board. We will need to connect the following pins from the Feather to the resistors and RGB LED:

• Feather pin 4 to one leg of a 560 ohm resistor, and the other leg of the resistor to the first leg (R) of the LED
• Feather 3V to the second leg (3V) of the LED
• Feather pin 5 to one leg of a 560 ohm resistor, and the other leg of the resistor to the third leg (G) of the LED
• Feather pin 2 to one leg of a 560 ohm resistor, and the other leg of the resistor to the fourth leg (B) of the LED
Next, let's look at the example sketch we will be using.

**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 2.4.0 of the Adafruit IO Arduino library installed before continuing.

For this example, you will need to open the adafruitio_13_rgb 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.
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

```
/**/ WIFI /**/

// the AdafruitIO_Wifi client will work with the following:
// - Feather M0 WiFi -> https://www.adafruit.com/product/3010
#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.

```
/**/ FONA /**/

// the AdafruitIO_FONA client will work with the following:
// - Feather 32u4 FONA Breakout w/ Headers -> https://www.adafruit.com/product/3028
// - Feather 32u4 FONA Breakout w/ Headers & Headers -> https://www.adafruit.com/product/3032
// comment out the AdafruitIO_Wifi client in the WIFI section
#include "AdafruitIO_FONA.h"
AdafruitIO_FONA io(IO_USERNAME, IO_KEY);
```

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

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

Arduino Code

The adafruitio_13_rgb example uses pins 4, 5, & 2 for red, green, and blue by default. These pins can be modified by changing the RED_PIN, GREEN_PIN, and BLUE_PIN defines. You will need to choose pins on your board that support PWM output.

The next chunk of code sets up an instance of the color feed.

In the setup function, we connect to Adafruit IO, and attach a function called handleMessage to the color feed. This function will be called whenever your device receives messages for that feed.
We also use a special function for the ESP8266 platform that sets the analogWrite range to 0-255. This will be ignored on all other platforms.

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.

```cpp
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 'color' feed.
    // the handleMessage function (defined below)
    // will be called whenever a message is
    // received from adafruit io.
    color-&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());

    // set analogWrite range for ESP8266
    #ifdef ESP8266
        analogWriteRange(255);
    #endif
}
```

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.

```cpp
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 color feed gets a message.
We use the `data->toRed()`, `data->toGreen()`, and `data->toBlue()` functions to convert the incoming hex color values to integers that will be compatible with `analogWrite`.

Because we are using a common anode RGB LED, we will need to flip the incoming RGB values. We do this by subtracting the values from 255 before sending the values to `analogWrite`.

If you are using a common cathode RGB LED, you can send the values directly to `analogWrite` from `data->toRed()`, `data->toGreen()`, and `data->toBlue()`.

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

    // print RGB values and hex value
    Serial.println("Received:");
    Serial.print(" - R: ");
    Serial.println(data->toRed());
    Serial.print(" - G: ");
    Serial.println(data->toGreen());
    Serial.print(" - B: ");
    Serial.println(data->toBlue());
    Serial.print(" - HEX: ");
    Serial.println(data->value());

    // invert RGB values for common anode LEDs
    analogWrite(RED_PIN, 255 - data->toRed());
    analogWrite(GREEN_PIN, 255 - data->toGreen());
    analogWrite(BLUE_PIN, 255 - data->toBlue());
}
```

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.

You can now use the color block on Adafruit IO to set a color, and you should see something resembling the following in the Arduino Serial Monitor.

```
Received:
 - R: 0
 - G: 0
 - B: 0
 - HEX: #000000
Received:
 - R: 210
 - G: 31
 - B: 31
 - HEX: #d21f1f
```
You should also see the RGB LED update with the color you picked in the color block.

---

**Python Wiring**

**Parts**

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.

**Raspberry Pi 3 - Model B+ - 1.4GHz Cortex-A53 with 1GB RAM**

The Raspberry Pi 3 Model B is the most popular Raspberry Pi computer made, and the Pi Foundation knows you can always make a good thing better! And what could make the Pi 3...

[https://www.adafruit.com/product/3775](https://www.adafruit.com/product/3775)

If you're following along with a [Raspberry Pi](https://www.adafruit.com/product/3775), 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:
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. This guide uses three.

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.

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
<th>URL</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 x Adafruit 16-Channel 12-bit PWM/Servo Driver</td>
<td>Adafruit PCA9685 Breakout.</td>
<td><a href="https://www.adafruit.com/product/815">https://www.adafruit.com/product/815</a></td>
</tr>
<tr>
<td>3 x Resistors</td>
<td>560 Ohm</td>
<td><a href="https://www.adafruit.com/product/2784">https://www.adafruit.com/product/2784</a></td>
</tr>
<tr>
<td>1 x RGB LED</td>
<td>Diffused RGB (tri-color) LED - Common Anode</td>
<td><a href="https://www.adafruit.com/product/159">https://www.adafruit.com/product/159</a></td>
</tr>
</tbody>
</table>
Wiring

Make the following connections between the Raspberry Pi and the PCA9685:

- Pi 3.3V to Power Rail
- Pi GND to Ground Rail
- 3.3V to PCA9685 VCC
- Pi SDA to PCA9685 SDA
- Pi SCL to PCA9685 SCL

Now we're going to connect the RGB LED to the PCA9685. We'll be using the LED with the flat side of the LED on the left. If you're having trouble seeing which side is the flat side, use the longest pin (3V) as your guide. The longest pin will be closer to the flat side of the RGB LED.
Make the following connections between the RGB LED, resistors and the PCA9685:

- PCA9685 Channel 15 to one leg of a 560 ohm resistor, and the other leg of the resistor to the first leg (R) of the RGB LED.
- 3.3V to the second leg (3V) of the RGB led.
- PCA9685 Channel 14 to one leg of a 560 ohm resistor, and the other leg of the resistor to the third leg (G) of the RGB LED.
- PCA9685 Channel 13 to one leg of a 560 ohm resistor, and the other leg of the resistor to the fourth leg (B) of the RGB LED.

Next, let's install the required CircuitPython libraries and 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 + 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 I2C devices with the command:
sudo i2cdetect -y 1

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

```
pi@io-pi:~ $ sudo i2cdetect -y 1
```

### 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 on 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

The rgb_led.py example uses the PCA9685's channels 6, 5 and 4 by default. These channels can be modified by changing the RED_PIN, GREEN_PIN, and BLUE_PIN variables. You can use any three channels on the PCA (0 to 15).

```python
# PWM Pins
RED_PIN = 6
GREEN_PIN = 5
BLUE_PIN = 4
```

After importing our system libraries, Adafruit Blinka, the PCA9685, and the Adafruit IO REST client, we need to set the `ADAFRUIT_IO_KEY` and `ADAFRUIT_IO_USERNAME` variables to our account credentials.

```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_AIO_KEY'

# Set to your Adafruit IO username.
```
In the setup of this code, we create an instance of the Adafruit IO Client, create a color feed, and create a PCA965 class instance which is used to control the PWM outputs.

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

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

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

# Create a simple PCA965 class instance.
pca = PCA965(i2c_bus)
pca.frequency = 60
```

Whenever the color feed receives a new message, we'll first convert the feed (in hexadecimal) to separate red, green, and blue values compatible with the PCA using the `toRed()`, `toGreen()`, and `toBlue()` functions.

Then, we use the `map_range()` method to map the values to 16-bit values for the PCA965:

```python
red = map_range(int(red), 0, 255, 0, 65535)
green = map_range(int(green), 0, 255, 0, 65535)
blue = map_range(int(blue), 0, 255, 0, 65535)
```

Then, we're going to send set each pin's duty cycle. Because the RGB LED we are using is common anode, we'll need to flip the incoming RGB values by subtracting them from 65535.

```python
pca.channels[RED_PIN].duty_cycle = 65535 - int(red)
pca.channels[GREEN_PIN].duty_cycle = 65535 - int(green)
pca.channels[BLUE_PIN].duty_cycle = 65535 - int(blue)
```

Save the code to your Raspberry Pi. Run the code by entering the following into your terminal:

```
python3 rgb_led.py
```

You can now use the color block on Adafruit IO to set the color feed. You should see something resembling the following in the terminal when you change this feed:
Received Color:
- R:  255
- G:  0
- B:  0
- HEX:  #ff0000

Received Color:
- R:  0
- G:  0
- B:  0
- HEX:  #000000

You should also see the RGB LED update with the color you picked in the color block.

Code

```python
# import system libraries
import time

# import Adafruit Blinka
from board import SCL, SDA
from busio import I2C

# import the PCA9685 module.
from adafruit_pca9685 import PCA9685

# import Adafruit IO REST client
from Adafruit_IO import Client, Feed, RequestError

# PWM Pins
RED_PIN = 6
GREEN_PIN = 5
BLUE_PIN = 4

# 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_AIO_KEY'

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

```
# Control a RGB LED using Adafruit IO and Python

Tutorial Link: https://learn.adafruit.com/adafruit-io-basics-color

Adafruit invests time and resources providing this open source code. Please support Adafruit and open source hardware by purchasing products from Adafruit!

Author(s): Brent Rubell for Adafruit Industries
Copyright (c) 2018 Adafruit Industries
Licensed under the MIT license.
All text above must be included in any redistribution.

Dependencies:
- Adafruit_Blinka
  (https://github.com/adafruit/Adafruit_Blinka)
- Adafruit_CircuitPython_PCA9685
  (https://github.com/adafruit/Adafruit_CircuitPython_PCA9685)
```
# Create an instance of the REST client.
aio = Client(ADATAFRUIT_IO_USERNAME, ADAFRUIT_IO_KEY)

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

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

# Create a simple PCA9685 class instance.
pca = PCA9685(i2c_bus)
pca.frequency = 60
prev_color = '#000000'

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 `color` feed
    color_val = aio.receive(color.key)
    if color_val != prev_color:
        # print rgb values and hex value
        print('Received Color: ')
        red = aio.to_red(color_val.value)
        print('\t - R: ', red)
        green = aio.to_green(color_val.value)
        print('\t - G: ', green)
        blue = aio.to_blue(color_val.value)
        print('\t - B: ', blue)
        print('\t - HEX: ', color_val.value)
        # map color values (0-255) to 16-bit values for the pca
        red = map_range(int(red), 0, 255, 0, 65535)
        green = map_range(int(green), 0, 255, 0, 65535)
        blue = map_range(int(blue), 0, 255, 0, 65535)
        # invert RGB values for common anode LEDs.
        pca.channels[RED_PIN].duty_cycle = 65535 - int(red)
        pca.channels[GREEN_PIN].duty_cycle = 65535 - int(green)
        pca.channels[BLUE_PIN].duty_cycle = 65535 - int(blue)
        prev_color = color_val
        # let's wait a bit so we don't flood adafruit io's servers...
time.sleep(1)

---

**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://forums.adafruit.com)!

©Adafruit Industries
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.

To resolve this error, from the Arduino IDE, navigate to the Manage Libraries... option in the Sketch -> Include Library menu.

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());`
// 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.