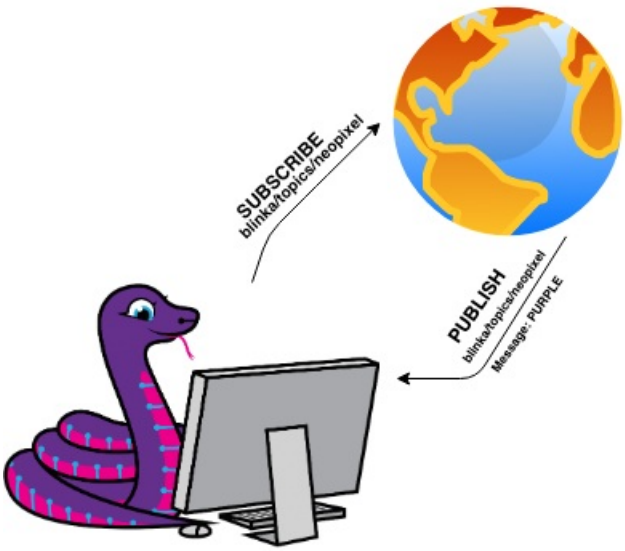
adafruit learning system

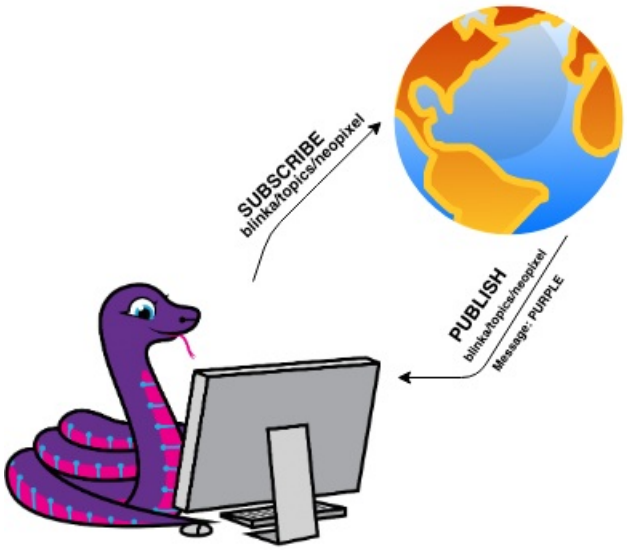
MQTT in CircuitPython

Created by Brent Rubell



Last updated on 2019-07-25 01:23:26 PM UTC

Overview



So, you have a CircuitPython project and want to connect it to the internet? You may want to consider adding MQTT to your project. MQTT is an extremely popular and lightweight protocol which can connect your project to the internet and quickly process network events.

MQTT (https://adafru.it/pYc) is a small, lightweight protocol which is incredibly easy to use in scenarios where bandwidth is at a premium, your project is sending a small amount of data every so often, or if you'd like to process network events incredibly quickly (clicking a button would cause your project to do something).

- **If you'd like to learn more about MQTT** - check out its section in the All The Internet of Things: Protocols guide (https://adafru.it/Fmp).

We've built a robust MQTT module for CircuitPython called CircuitPython MiniMQTT (https://adafru.it/Fmq) to quickly get you started connecting your projects to the internet and sending data around.

In this guide, you will set up your CircuitPython board with the necessary libraries, connect to the internet and connect your CircuitPython board to either a MQTT broker of your choice or the free Adafruit IO MQTT broker. We've included code-walkthrough and advanced usage sections to this guide so you can roll your own MiniMQTT project!

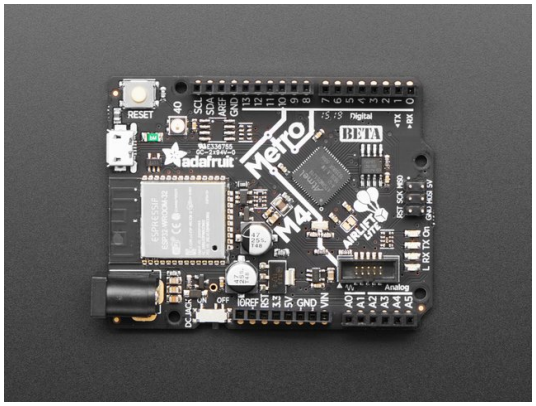
Note: MiniMQTT currently only supports CircuitPython devices connected using WiFi. It does not yet support other modes of network transports - such as Ethernet.

Parts

You'll need a CircuitPython board which either has network connectivity built-in or external network hardware .

All-in-One CircuitPython WiFi Boards

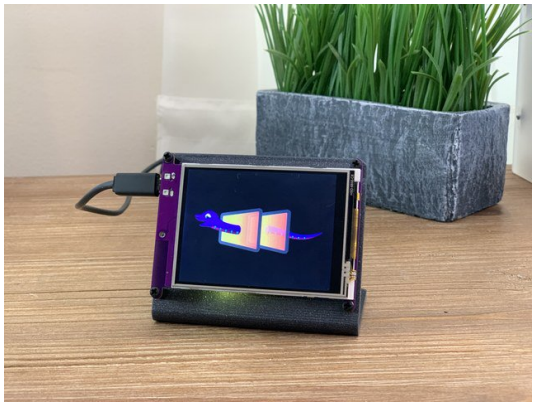
Don't want to add extra hardware to your project? Consider grabbing a board which has an ESP32 WiFi co-processor built-in!



Adafruit Metro M4 Express AirLift (WiFi) - Lite

OUT OF STOCK

OUT OF STOCK



Adafruit PyPortal - CircuitPython Powered Internet Display

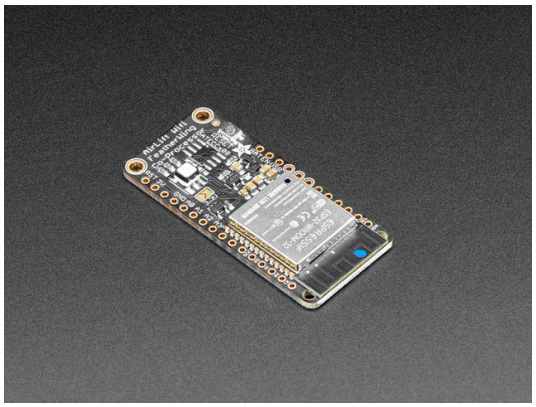
OUT OF STOCK

OUT OF STOCK

Externally Connected CircuitPython Network Hardware

If you already have a project which uses a CircuitPython board without internet connectivity, you can easily add WiFi connectivity with an externally connected AirLift module.

If you're using an Adafruit Feather, adding WiFi connectivity to your project is as easily as snapping the the AirLift FeatherWing on top.



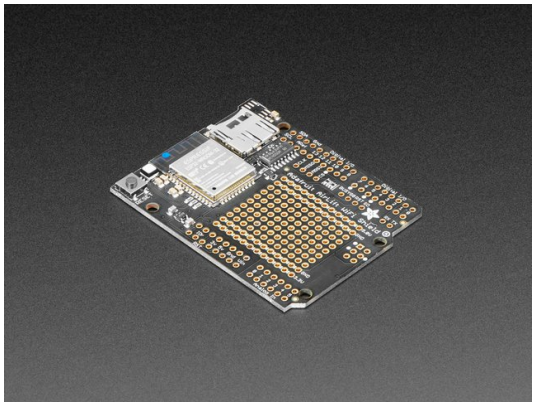
Adafruit AirLift FeatherWing – ESP32 WiFi Co-Processor

$12.95

IN STOCK

ADD TO CART

If you're using a CircuitPython board with an Arduino form-factor, you'll want to pick up an AirLift Shield.

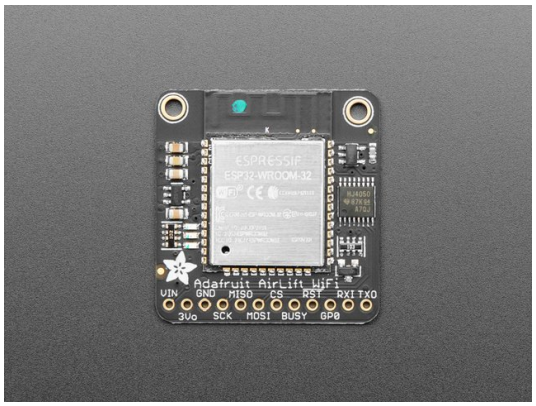


Adafruit AirLift Shield - ESP32 WiFi Co-Processor

OUT OF STOCK

OUT OF STOCK

If none of the form factors above work for your project - we also have a breakout board which can connect to any microcontroller over SPI.



Adafruit AirLift – ESP32 WiFi Co-Processor Breakout Board

$12.95
IN STOCK

ADD TO CART

CircuitPython Setup

MQTT devices, like your CircuitPython board, connect to a broker with a client library.

We've written an awesome CircuitPython MQTT client library called **Adafruit MiniMQTT (https://adafru.it/Fm2).**

This library is based off previous work by pfalcon on uMQTT (https://adafru.it/Fm3) (and the umqtt port to ESP32SPI by beachbc (https://adafru.it/Fm4)). MiniMQTT's primary difference from MicroPython's uMQTT library is its use of calling conventions and method names *similar* to The Eclipse Foundation's Paho.Mqtt.Python (https://adafru.it/Fm5).

MiniMQTT and Ethernet

We've written MiniMQTT as a protocol implementation - it doesn't have a hardware dependency for network connections. At this time, the MiniMQTT library is only available for CircuitPython devices connected over WiFi. Ethernet and cellular connectivity is not supported (yet! no ETA).

Install CircuitPython

Some CircuitPython compatible boards come with CircuitPython installed. Others are *CircuitPython-ready*, but need to have it installed. As well, you may want to update the version of CircuitPython already installed on your board. The steps are the same for installing and updating.

- To **install (or update) your CircuitPython board**, follow this page and come back here when you've successfully installed (or updated) CircuitPython. (https://adafru.it/Amd)

CircuitPython Library Installation

To interface your AirLift breakout/board with and the internet - you'll need to install a few CircuitPython libraries on your board.

First make sure you are running the latest version of Adafruit CircuitPython (https://adafru.it/tBa) for your board.

Next you'll need to install the necessary libraries to use the hardware--carefully follow the steps to find and install these libraries from Adafruit's CircuitPython library bundle (https://adafru.it/zdx) matching your version of CircuitPython.

CircuitPython hardware shows up on your computer operating system as a flash drive when connected via usb. The flash drive is called **CIRCUITPY** and contains a number of files. You will need to add additional files to enable the features of this project.

First, create a folder on the drive named lib if it is not already there.

Ensure your board's **lib** folder has the following files and folders copied over. The version of the files must be the same major version as your version of CircuitPython (i.e. 4.x for 4.x, 5.x for 5.x, etc.)

- **adafruit_minimqtt**
- **adafruit_logger**
- **adafruit_esp32spi**
- **adafruit_bus_device**
- **neopixel.mpy**

WiFi - Internet Connect!

Once you have CircuitPython setup and libraries installed, you can get your project connected to the Internet over WiFi.

To do this, you'll be editing CircuitPython code and will need an editor. We suggest using **Mu**, a lightweight text editor with support for CircuitPython built-in.

Click the button below to get instructions on how to install the Mu Editor.

https://adafru.it/ANO

https://adafru.it/ANO

If you have not yet connected your CircuitPython WiFi board to the Internet, follow one of the guides below and come back when you've successfully connected to the internet:

- **If you have an AirLift All-in-One board (like the Metro M4 WiFi or PyPortal),** follow this page to connect to the internet (https://adafru.it/Eao).
- **If you have an externally connected AirLift (like the AirLift FeatherWing or AirLift breakout)**, follow this page to connect to the internet (https://adafru.it/EF-).

Connecting to the Adafruit IO MQTT Broker

If you do not want to host your own MQTT broker, using Adafruit IO (https://adafru.it/eZ8)'s MQTT broker is a great way to get started connecting your CircuitPython project to the internet. Best of all - *it's free to use!*

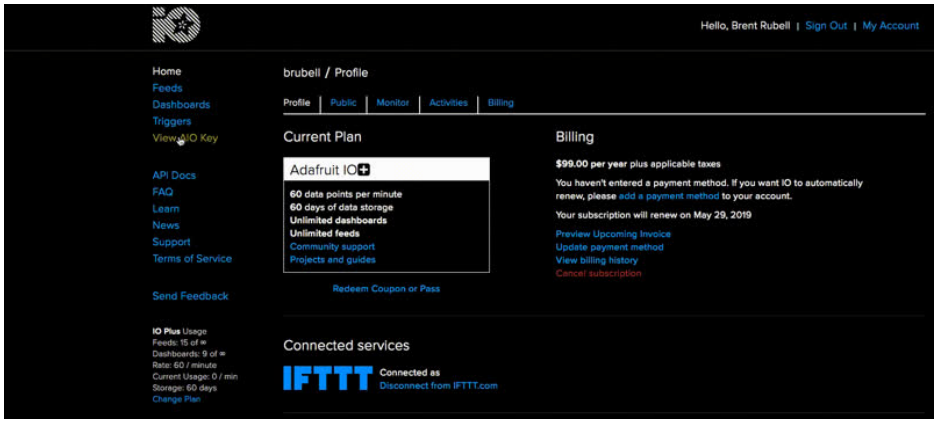
You're going to build an Adafruit IO Dashboard which can visualize incoming data from your CircuitPython board, and send data to it.

Obtain Adafruit IO Username and Key

If you have not already, sign up for an Adafruit IO account by clicking this link (https://adafru.it/Fm6).

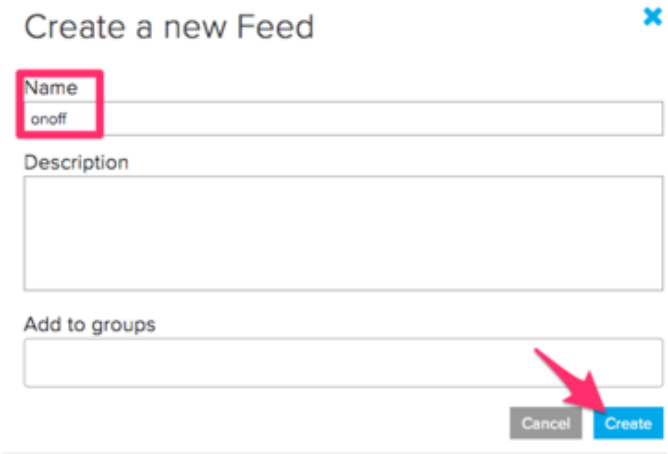
Next, you're going to need your Adafruit IO username and secret API key.

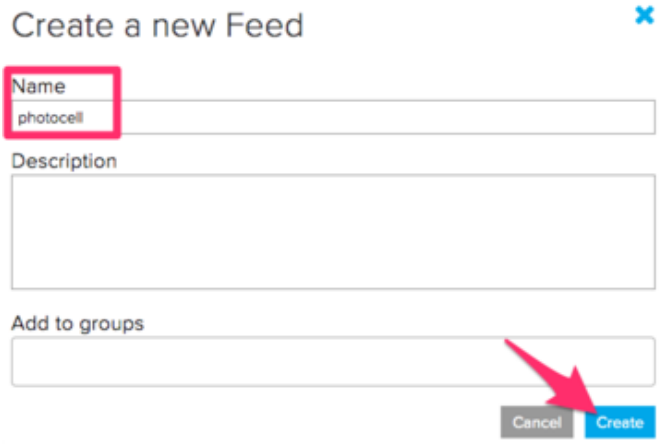
Navigate to your profile (https://adafru.it/fsU) and click the View AIO Key button to retrieve them. Write them down in a safe place, you'll need them later.



Create Adafruit IO Feeds

Adafruit IO uses a special type of MQTT Topic named a Feed to store data along with metadata (information about the data). You'll be publishing data to one feed, and subscribing to another.





Create two new Adafruit IO Feeds named *onoff* and *photocell*.

- ***photocell*** - This feed will store light data published *from* your device to Adafruit IO
- **onoff** - This feed will act as an on/off switch, publishing data *to* your device from Adafruit IO

If you have not created an Adafruit IO Feed before, follow this page and come back once you've the created two feeds above (https://adafru.it/f5k).

Create an Adafruit IO Dashboard

Adafruit IO Dashboards (https://adafru.it/f5m) are a way to interact with feeds. You can link *blocks* on dashboards to your feeds. The blocks can either display information about the feed (such as the current temperature) or allow you to interact with a feed by setting it to different values.

Start by creating a new dashboard. Name it whatever you'd like!

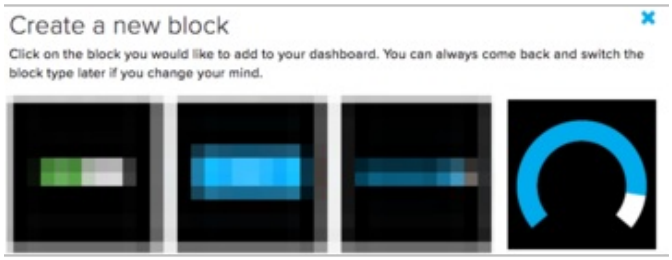
- I**f you do not know how to create a dashboard**, head over to this page and come back here when you've successfully created a dashboard. (https://adafru.it/Fm7)

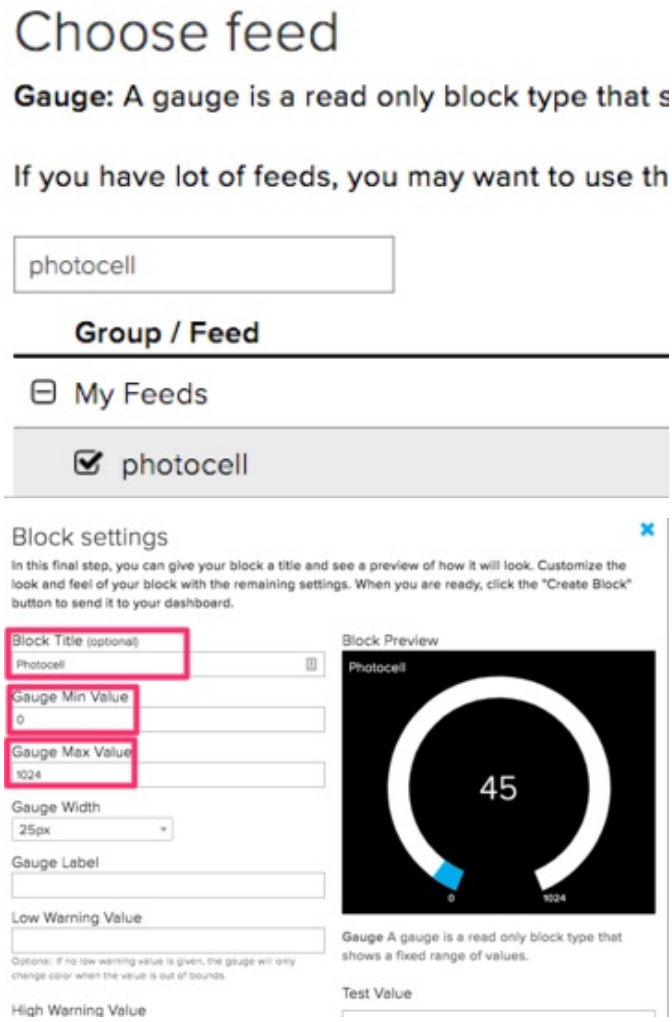
Create a Gauge Block

After creating a dashboard, **create a Gauge Block** to display the value of the Photocell feed.

- **Choose** the *photocell* feed
- **Change the block title to** *Photocell*
- **Set the Gauge Minimum Value to** *0*
- **Set the Gauge Maximum Value to** *1024*

If you do not know how to add blocks to a dashboard, head to over this page and come back when you've added a gauge block to your dashboard. (https://adafru.it/DZe)

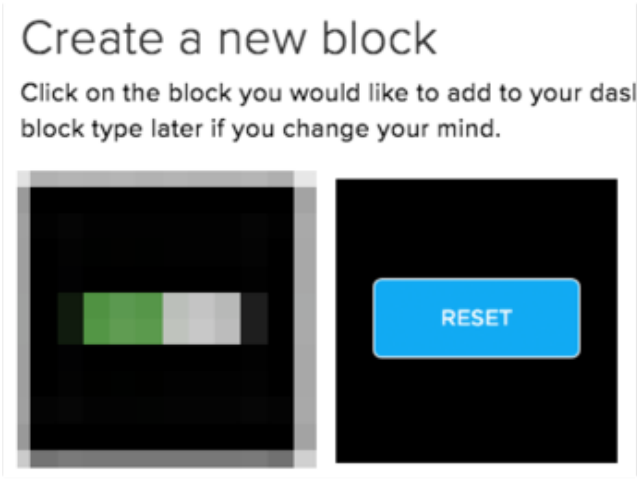




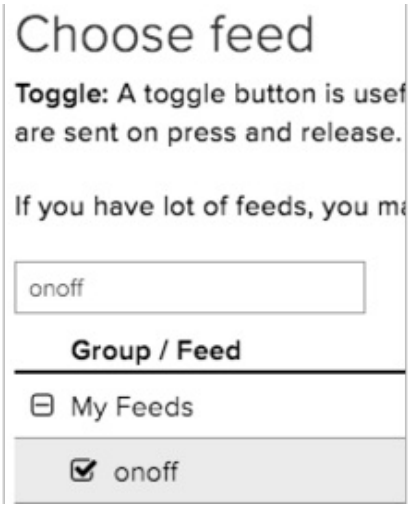
Create a Toggle Switch Block

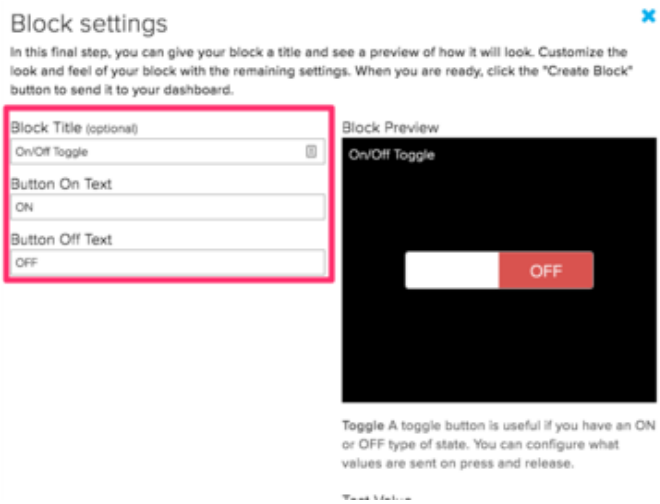
To send values to the *onoff* feed you created - create a toggle switch block.

- **Choose the *onoff* feed**
- **Set the block title to *On/Off***

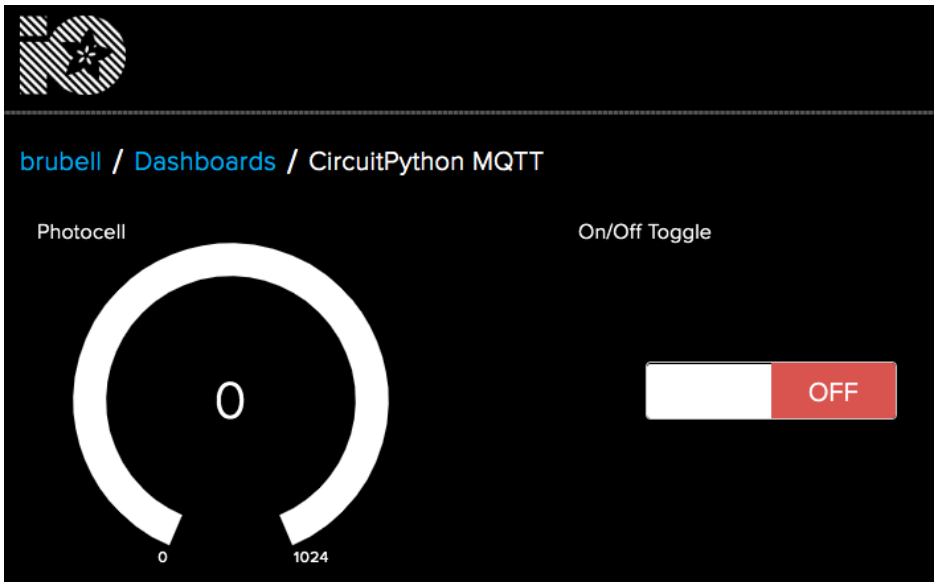


- **Set the Button On Text to *ON***
- **Set the Button Off Text to *OFF***





Your dashboard should look like the following:


brubell / Dashboards / CircuitPython MQTT
Photocell
0
0
1024
On/Off Toggle
OFF

CircuitPython Usage

Secrets File Setup for Adafruit IO

While you created a **secrets.py** file and connected to the internet in the previous step, you'll need to edit the **secrets.py** file to include your Adafruit IO **Username** and **Secret Key.**

Add the following code to your **secrets.py** file, **replacing** `_your_adafruit_io_username` **with your Adafruit IO username.**

Then, **replace** `_your_big_huge_super_long_aio_key_` **with your Adafruit IO Active Key**.

```
secrets = {
    'ssid' : '_your_wifi_ssid_',
    'password' : '_your_wifi_password_',
    'aio_username' : '_your_adafruit_io_username_',
    'aio_key' : '_your_big_huge_super_long_aio_key_'
    }
```

Make sure you **save this file** before proceeding as **secrets.py** in the root directory of your board **CIRCUITPY** drive.

Code Usage

Copy the following code to your **code.py** file on your microcontroller:

```
# Adafruit MiniMQTT Pub/Sub Example
# Written by Tony DiCola for Adafruit Industries
# Modified by Brent Rubell for Adafruit Industries
import time
import board
import busio
from digitalio import DigitalInOut
import neopixel
from adafruit_esp32spi import adafruit_esp32spi
from adafruit_esp32spi import adafruit_esp32spi_wifimanager
import adafruit_esp32spi.adafruit_esp32spi_socket as socket
from adafruit_minimqtt import MQTT

### WiFi ###

# Get wifi details and more from a secrets.py file
try:
    from secrets import secrets
except ImportError:
    print("WiFi secrets are kept in secrets.py, please add them there!")
    raise

# If you are using a board with pre-defined ESP32 Pins:
esp32_cs = DigitalInOut(board.ESP_CS)
esp32_ready = DigitalInOut(board.ESP_BUSY)
esp32_reset = DigitalInOut(board.ESP_RESET)

# If you have an externally connected ESP32:
# esp32_cs = DigitalInOut(board.D9)
# esp32_ready = DigitalInOut(board.D10)
```

```
# esp32_reset = DigitalInOut(board.D5)

spi = busio.SPI(board.SCK, board.MOSI, board.MISO)
esp = adafruit_esp32spi.ESP_SPIcontrol(spi, esp32_cs, esp32_ready, esp32_reset)
"""Use below for Most Boards"""
status_light = neopixel.NeoPixel(
    board.NEOPIXEL, 1, brightness=0.2)  # Uncomment for Most Boards
"""Uncomment below for ItsyBitsy M4"""
# status_light = dotstar.DotStar(board.APA102_SCK, board.APA102_MOSI, 1, brightness=0.2)
# Uncomment below for an externally defined RGB LED
# import adafruit_rgbled
# from adafruit_esp32spi import PWMOut
# RED_LED = PWMOut.PWMOut(esp, 26)
# GREEN_LED = PWMOut.PWMOut(esp, 27)
# BLUE_LED = PWMOut.PWMOut(esp, 25)
# status_light = adafruit_rgbled.RGBLED(RED_LED, BLUE_LED, GREEN_LED)
wifi = adafruit_esp32spi_wifimanager.ESPSPI_WiFiManager(
    esp, secrets, status_light)

### Feeds ###

# Setup a feed named 'photocell' for publishing to a feed
photocell_feed = secrets['aio_username'] + '/feeds/photocell'

# Setup a feed named 'onoff' for subscribing to changes
onoff_feed = secrets['aio_username'] + '/feeds/onoff'

### Code ###

# Define callback methods which are called when events occur
# pylint: disable=unused-argument, redefined-outer-name
def connected(client, userdata, flags, rc):
    # This function will be called when the client is connected
    # successfully to the broker.
    print('Connected to Adafruit IO! Listening for topic changes on %s' % onoff_feed)
    # Subscribe to all changes on the onoff_feed.
    client.subscribe(onoff_feed)


def disconnected(client, userdata, rc):
    # This method is called when the client is disconnected
    print('Disconnected from Adafruit IO!')


def message(client, topic, message):
    # This method is called when a topic the client is subscribed to
    # has a new message.
    print('New message on topic {0}: {1}'.format(topic, message))


# Connect to WiFi
wifi.connect()

# Set up a MiniMQTT Client
mqtt_client = MQTT(socket,
                   broker='io.adafruit.com',
                   username=secrets['aio_username'],
                   password=secrets['aio_key'],
                   network_manager=wifi)
```

```
# Setup the callback methods above
mqtt_client.on_connect = connected
mqtt_client.on_disconnect = disconnected
mqtt_client.on_message = message

# Connect the client to the MQTT broker.
print('Connecting to Adafruit IO...')
mqtt_client.connect()

photocell_val = 0
while True:
    # Poll the message queue
    mqtt_client.loop()

    # Send a new message
    print('Sending photocell value: %d...' % photocell_val)
    mqtt_client.publish(photocell_feed, photocell_val)
    print('Sent!')
    photocell_val += 1
    time.sleep(1)
```

WiFi Connection Pinouts

If you are using a board with a built-in ESP32 (like the PyPortal), you do not need to make changes to the code. It's already setup for usage with those boards. However, if you are using an externally connected ESP32 (like the AirLift Breakout), you'll need to define the ESP32's pinouts.

Make sure to **change the ESP32 pin definitions in the code to match your wiring**. You can do this by uncommenting and editing the following lines in your code to match your wiring.

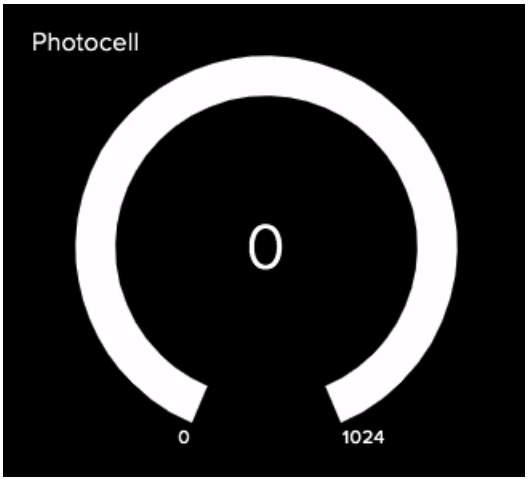
```
esp32_cs = DigitalInOut(board.D9)
esp32_ready = DigitalInOut(board.D10)
esp32_reset = DigitalInOut(board.D5)
```

Feed Publishing Example

Directly after saving the **code.py** file, open a serial monitor/REPL to see the output. It should look something like the following:

```
code.py output:
Connecting to Adafruit IO...
Connected to Adafruit IO! Listening for topic changes on brubell/feeds/onoff
Sending photocell value: 0...
Sent!
Sending photocell value: 1...
Sent!
Sending photocell value: 2...
Sent!
Sending photocell value: 3...
Sent!
```

Navigate to the dashboard you created earlier. **You should see the photocell gauge increasing its value** as your CircuitPython device publishes the increasing photocell value to Adafruit IO.



If you **navigate to the page for the photocell feed**, you'll see the values increasing there along with metadata about when the data was received by the Adafruit IO broker.

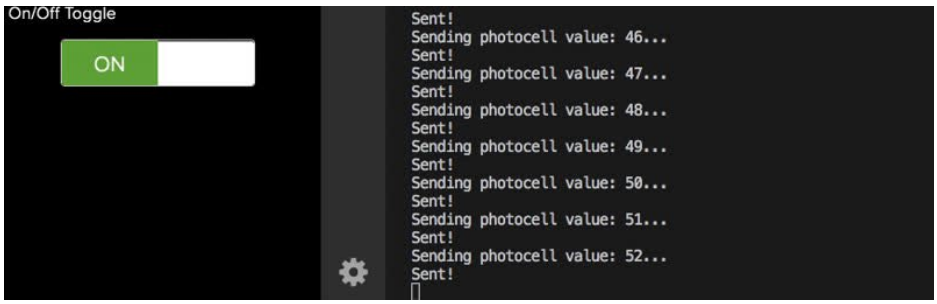
Created at	Value
2019/07/18 5:02:02pm	121
2019/07/18 5:02:01pm	120
2019/07/18 5:01:59pm	119
2019/07/18 5:01:58pm	118
2019/07/18 5:01:56pm	117
2019/07/18 5:01:55pm	116

Feed Subscription Example

While we're publishing the increasing *photocell* value *to* Adafruit IO - our code also *subscribes* to changes on the *onoff* feed.

The example *code.py* subscribes to the *onoff* feed when the client successfully connects to Adafruit IO. You don't need to make any changes to your code!

With the code still running on your CircuitPython device - **click the toggle switch** to send a value to the *onoff* feed. You should see the value appear on your serial monitor/REPL.



If you *really* want to see the speed of MQTT - remove the one second delay in the `while True` loop.

Change the following code (within `while True`) **from:**

```
print('Sent!')
photocell_val += 1
time.sleep(1)
```

to

```
print('Sent!')
photocell_val += 1
time.sleep(0.5)
```

Be warned - if you do this you will not be able to run the code for very long. This is because Adafruit IO's MQTT server imposes a rate limit to prevent excessive load on the service.

The current Adafruit IO Data Rate is at most 1 request per second (or 60 requests within 60 seconds), without an Adafruit IO+ Boost applied to your account.

Going Further - the Adafruit IO CircuitPython Module

While you *can* use this code to communicate with Adafruit IO, the recommend method of using CircuitPython with Adafruit IO is with the Adafruit IO CircuitPython module (https://adafru.it/EFZ).

This module has methods to simplify using the Adafruit IO MQTT API. It also includes helper features to make your experience using Adafruit IO better.

If you want to use Adafruit IO and CircuitPython, check out Adafruit IO CircuitPython (https://adafru.it/Fm8) and its code examples (https://adafru.it/Fm9) on GitHub!

Connecting to a MQTT Broker

You can connect your CircuitPython device to any MQTT broker of your choice. MQTT brokers generally fall in three categories: Commercial hosted (Paid), Free hosted (Adafruit IO's MQTT broker lives on servers owned by Adafruit - we have a free tier), or Free and self-hosted (bring your own server!).

Commercial Brokers

These large-scale commercial (paid) brokers come with a higher data-rate speed and usually include device management features.

These brokers can also connect to CircuitPython devices running MiniMQTT. The primary advantage of using one of these services over your own broker or Adafruit IO is integration with a larger *suite* of cloud-hosted tools. For example, if you are sending data to an Azure IoT Hub via MQTT, you could visualize sensor data using PowerBI or run machine learning models on it from Azure.

Self-Hosted Brokers

There are plenty of tools out there to host your own MQTT broker. We like Eclipse's Mosquitto (https://adafru.it/pey) - it's an open-source broker which implements the MQTT protocol. There are images for Mac, Linux, Windows and Raspberry Pi.

Secrets File Setup

While you created a secrets file and connected to the Internet in the previous step, you'll need to edit the**secrets.py** file to include information about your MQTT broker.

Add the following code to your **secrets.py** file, **replacing** `_your_broker_url_or_ip` **with the URL or IP Address of the MQTT broker you'd like to connect to.**

If your MQTT broker requires authentication, **replace** **`_your_mqtt_broker_username_`** **with the username you use to log into the broker**.

Then, **replace** **`_your_mqtt_broker_password_`** **with the password you use** to authenticate your username with the broker.

If your MQTT broker does not require authentication, you may remove these fields.

```
secrets = {
    'ssid' : '_your_wifi_ssid_',
    'password' : '_your_wifi_password_',
    'broker' : '_your_mqtt_broker_url_or_ip',
    'user' : '_your_mqtt_broker_username_',
    'pass' : '_your_mqtt_broker_password_'
    }
```

Code

Copy the following code to your **code.py** file on your microcontroller:

```
Temporarily unable to load content:
```

Before running your code, you'll need to make a few small changes for your configuration.

First, **replace `mqtt_topic` with the mqtt topic you'd like to subscribe to**. For example, if you are building a weather station and want to subscribe to its temperature.

Change the `mqtt_topic` **from**

`mqtt_topic = 'test/topic'`

to

`mqtt_topic= 'weatherstation/temperature'`

- **For more information about MQTT topic naming,** check out this page here (https://adafru.it/Fmb).

MiniMQTT Port Configuration

By default, MiniMQTT connects to port 8883 (Secure/SSL). If you'd like to connect to a different port on the broker, **use the following code to initialize the client.**

```
client =  MQTT(socket,
            broker = secrets['broker'],
             port = 1234,
            username = secrets['user'],
            password = secrets['pass'],
            network_manager = wifi)
```

You can **toggle between an insecure** (port 1883) **and secure** (port 8883) **connection by setting the `is_ssl` parameter to `True` or `False`**.

```
client =  MQTT(socket,
            broker = secrets['broker'],
            username = secrets['user'],
            password = secrets['pass'],
             is_ssl = False,
            network_manager = wifi)
```

Code Usage

After setting up your topic and broker configuration, it's time to connect. Save the **code.py** file and **open the serial monitor**.

The client attempts to connect to the MQTT broker you specified.

```
Attempting to connect to io.adafruit.com
Connected to MQTT Broker!
Flags: 0
RC: 0
```

The client **subscribes to the** `mqtt_topic` you specified.

```
Subscribed to user/topic with QOS level 1
```

And notifies you that it subscribed to `mqtt_topic` with a quality of service level of 1. You can set the quality of service level by modifying the call to `subscribe` (read the docs about this method here (https://adafru.it/Fmc)).

- For more about the MQTT quality of service level - check out this page (https://adafru.it/Fmd)

The client **publishes** the string *Hello Broker* **to the** `mqtt_topic`.

```
Published to user/topic with PID 11
```

The client **unsubscribes** from the `mqtt_topic`.

```
Unsubscribed from user/topic with PID 22
```

Finally, the client **disconnects** from the MQTT broker

```
Disconnected from MQTT Broker!
```

The code tests all the primary methods of the MiniMQTT client. We'll walk through the code to understand how this example works.

Code Walkthrough

The code first connects the ESP32 to the wireless network you specified in the **secrets.py** file on your **CIRCUITPY** drive.

```
# Connect to WiFi
wifi.connect()
```

Once a WiFi network connection has been established, the code sets up a new MiniMQTT client instance

```
client =  MQTT(socket,
               broker = secrets['broker'],
               username = secrets['user'],
               password = secrets['pass'],
               network_manager = wifi)
```

Then, the code attaches callback handler methods to the client.

```
# Connect callback handlers to client
client.on_connect = connect
client.on_disconnect = disconnect
client.on_subscribe = subscribe
client.on_unsubscribe = unsubscribe
client.on_publish = publish
```

MiniMQTT Callback Methods

We're going to stop here to explain the utility and operation of the callback methods which are an important part of building programs with MiniMQTT.

Further up in the code, there are methods named `connect()`, `publish()`, `subscribe()`, `unsubscribe()` and `disconnect()`.

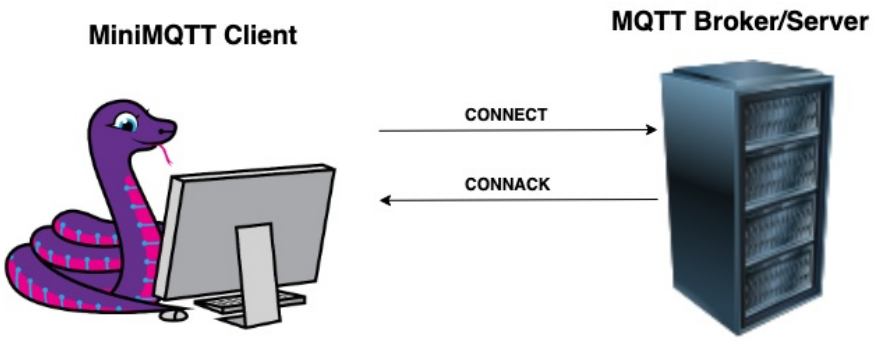
```
def connect(client, userdata, flags, rc):
    # This function will be called when the client is connected
    # successfully to the broker.
    print('Connected to MQTT Broker!')
    print('Flags: {0}\n RC: {1}'.format(flags, rc))

def disconnect(client, userdata, rc):
    # This method is called when the client disconnects
    # from the broker.
    print('Disconnected from MQTT Broker!')

def subscribe(client, userdata, topic, granted_qos):
    # This method is called when the client subscribes to a new feed.
    print('Subscribed to {0} with QOS level {1}'.format(topic, granted_qos))

def unsubscribe(client, userdata, topic, pid):
    # This method is called when the client unsubscribes from a feed.
    print('Unsubscribed from {0} with PID {1}'.format(topic, pid))

def publish(client, userdata, topic, pid):
    # This method is called when the client publishes data to a feed.
    print('Published to {0} with PID {1}'.format(topic, pid))
```

Each of these methods is executed when a **callback** is successfully read back from a MQTT control packet sent by the client.

- **If you do not know what a MQTT Client or a MQTT Broker is**, check out this page (https://adafru.it/Fme).

This means that whenever **MiniMQTT sends a CONNECT command to the MQTT broker**, the broker should respond that it received it and send another packet back to MiniMQTT (called a *CONNACK* packet).

If MiniMQTT receives this CONNACK packet, it will execute the client's on_connect method. It'll raise an error if it did not connect successfully.



As an example, we'll set up a connect callback method named `connect`. This method will print the string, *Connected to MQTT Broker*, if it successfully connects to your MQTT broker.

```
def connect(client, userdata, flags, rc):
    print('Connected to MQTT Broker!')
```

Then, we'll ***set*** the client's **`on_connect`** property to the **`connect`** method.

`client.on_connect = connect`

To connect the MiniMQTT client to the broker, we'll run the following line:

`client.connect()`

When the client's `connect()` method runs, it sends a CONNECT command to the MQTT broker with information including the broker's address and waits for the **CONNACK**.

Once the CONNACK is received, MiniMQTT calls *`on_connect`* from "behind the scenes". Since `connect` is attached to `on_connect`, **connect will execute and print** out the following message:

```
Connected to MQTT Broker!
```

Ok - we now understand how the *connect* method callback works. The rest of the example calls more functions which also execute similar callback methods.

First, the code attempts to connect to the MQTT broker you specified.

```
print('Attempting to connect to %s'%client.broker)
client.connect()
```

If the CircuitPython board connects successfully, it'll run the `connect` method.

The code will next attempt to subscribe to a `mqtt_topic` you defined earlier in the code.

```
print('Subscribing to %s'%mqtt_topic)
client.subscribe(mqtt_topic)
```

If the client successfully subscribes to the `mqtt_topic`, the `subscribe` method will execute.

Once the client successfully subscribes - it'll publish to the `mqtt_topic`.

```
print('Publishing to %s'%mqtt_topic)
client.publish(mqtt_topic, 'Hello Broker!')
```

After publishing, your broker should display that it received a value from the client. In addition, the `publish` method should run.

Next, the client unsubscribes from the `mqtt_topic` and disconnects from the broker.

```
print('Unsubscribing from %s'%mqtt_topic)
client.unsubscribe(mqtt_topic)

print('Disconnecting from %s'%client.broker)
client.disconnect()
```

These two methods should call the `unsubscribe` and `disconnect` methods.

That's it! The next page will go over some of the more advanced features of this library.

MiniMQTT Loop

You should **always use a loop when writing CircuitPython code which uses the MiniMQTT module**. The loop methods check incoming and process outgoing mqtt messages along with keeping the network connection between your board and the MQTT broker alive .

MiniMQTT supports two different types of loops: `loop` and `loop_forever`.

loop

Calling `loop` creates a **non-blocking** network loop. You can create new code below the call to `loop`, and it'll be executed. This type of loop should be run frequently to avoid disconnecting from the MQTT server. The `loop` method also **does not handle network hardware (WiFi) or MQTT broker disconnection**. You'll need to handle that yourself.

Here's an example of using a non-blocking loop.

```
import time
import board
import busio
from digitalio import DigitalInOut
import neopixel
from adafruit_esp32spi import adafruit_esp32spi
from adafruit_esp32spi import adafruit_esp32spi_wifimanager
import adafruit_esp32spi.adafruit_esp32spi_socket as socket
from adafruit_minimqtt import MQTT

### WiFi ###

# Get wifi details and more from a secrets.py file
try:
    from secrets import secrets
except ImportError:
    print("WiFi secrets are kept in secrets.py, please add them there!")
    raise

# If you are using a board with pre-defined ESP32 Pins:
esp32_cs = DigitalInOut(board.ESP_CS)
esp32_ready = DigitalInOut(board.ESP_BUSY)
esp32_reset = DigitalInOut(board.ESP_RESET)

# If you have an externally connected ESP32:
# esp32_cs = DigitalInOut(board.D9)
# esp32_ready = DigitalInOut(board.D10)
# esp32_reset = DigitalInOut(board.D5)

spi = busio.SPI(board.SCK, board.MOSI, board.MISO)
esp = adafruit_esp32spi.ESP_SPIcontrol(spi, esp32_cs, esp32_ready, esp32_reset)
"""Use below for Most Boards"""
status_light = neopixel.NeoPixel(board.NEOPIXEL, 1, brightness=0.2) # Uncomment for Most Boards
"""Uncomment below for ItsyBitsy M4"""
# status_light = dotstar.DotStar(board.APA102_SCK, board.APA102_MOSI, 1, brightness=0.2)
# Uncomment below for an externally defined RGB LED
# import adafruit_rgbled
# from adafruit_esp32spi import PWMOut
```

```
# RED_LED = PWMOut.PWMOut(esp, 26)
# GREEN_LED = PWMOut.PWMOut(esp, 27)
# BLUE_LED = PWMOut.PWMOut(esp, 25)
# status_light = adafruit_rgbled.RGBLED(RED_LED, BLUE_LED, GREEN_LED)
wifi = adafruit_esp32spi_wifimanager.ESPSPI_WiFiManager(esp, secrets, status_light)

### Adafruit IO Setup ###

# Setup a feed named `testfeed` for publishing.
default_topic = secrets['user']+'/feeds/testfeed'

### Code ###
# Define callback methods which are called when events occur
# pylint: disable=unused-argument, redefined-outer-name
def connected(client, userdata, flags, rc):
    # This function will be called when the client is connected
    # successfully to the broker.
    print('Connected to MQTT broker! Listening for topic changes on %s'%default_topic)
    # Subscribe to all changes on the default_topic feed.
    client.subscribe(default_topic)

def disconnected(client, userdata, rc):
    # This method is called when the client is disconnected
    print('Disconnected from MQTT Broker!')

def message(client, topic, message):
    """Method callled when a client's subscribed feed has a new
    value.
    :param str topic: The topic of the feed with a new value.
    :param str message: The new value
    """
    print('New message on topic {0}: {1}'.format(topic, message))

# Connect to WiFi
wifi.connect()

# Set up a MiniMQTT Client
mqtt_client = MQTT(socket,
                   broker = secrets['broker'],
                   username = secrets['user'],
                   password = secrets['pass'],
                   network_manager = wifi)

# Setup the callback methods above
mqtt_client.on_connect = connected
mqtt_client.on_disconnect = disconnected
mqtt_client.on_message = message

# Connect the client to the MQTT broker.
mqtt_client.connect()

photocell_val = 0
while True:
    # Poll the message queue
    mqtt_client.loop()

    # Send a new message
    print('Sending photocell value: %d'%photocell_val)
    mqtt_client.publish(default_topic, photocell_val)
    photocell_val += 1
```

```
    photocell_val += 1
    time.sleep(0.5)
```

loop_forever

The other type of loop you can use is `loop_forever`. This starts a blocking loop. **Network and broker error handling and reconnection is handled** within the `loop_forever` method.

Since this is a blocking loop, **any code below a call to `client.loop_forever()` will not execute**. With this method, you should handle all events within the callback methods.

Here's an example of using loop_forever:

```
# CircuitPython MiniMQTT Library
# Adafruit IO SSL/TLS Example for WiFi (ESP32SPI)
import board
import busio
from digitalio import DigitalInOut
import neopixel
from adafruit_esp32spi import adafruit_esp32spi
from adafruit_esp32spi import adafruit_esp32spi_wifimanager
import adafruit_esp32spi.adafruit_esp32spi_socket as socket
from adafruit_minimqtt import MQTT

### WiFi ###

# Get wifi details and more from a secrets.py file
try:
    from secrets import secrets
except ImportError:
    print("WiFi secrets are kept in secrets.py, please add them there!")
    raise

# If you are using a board with pre-defined ESP32 Pins:
esp32_cs = DigitalInOut(board.ESP_CS)
esp32_ready = DigitalInOut(board.ESP_BUSY)
esp32_reset = DigitalInOut(board.ESP_RESET)

# If you have an externally connected ESP32:
# esp32_cs = DigitalInOut(board.D9)
# esp32_ready = DigitalInOut(board.D10)
# esp32_reset = DigitalInOut(board.D5)

spi = busio.SPI(board.SCK, board.MOSI, board.MISO)
esp = adafruit_esp32spi.ESP_SPIcontrol(spi, esp32_cs, esp32_ready, esp32_reset)
"""Use below for Most Boards"""
status_light = neopixel.NeoPixel(board.NEOPIXEL, 1, brightness=0.2) # Uncomment for Most Boards
"""Uncomment below for ItsyBitsy M4"""
# status_light = dotstar.DotStar(board.APA102_SCK, board.APA102_MOSI, 1, brightness=0.2)
# Uncomment below for an externally defined RGB LED
# import adafruit_rgbled
# from adafruit_esp32spi import PWMOut
# RED_LED = PWMOut.PWMOut(esp, 26)
# GREEN_LED = PWMOut.PWMOut(esp, 27)
# BLUE_LED = PWMOut.PWMOut(esp, 25)
# status_light = adafruit_rgbled.RGBLED(RED_LED, BLUE_LED, GREEN_LED)
wifi = adafruit_esp32spi_wifimanager.ESPSPI_WiFiManager(esp, secrets, status_light)
```

```
### Adafruit IO Setup ###

# Setup a feed named `testfeed` for publishing.
default_topic = secrets['user']+'/feeds/testfeed'

### Code ###

# Define callback methods which are called when events occur
# pylint: disable=unused-argument, redefined-outer-name
def connected(client, userdata, flags, rc):
    # This function will be called when the client is connected
    # successfully to the broker.
    print('Connected to MQTT broker! Listening for topic changes on %s'%default_topic)
    # Subscribe to all changes on the default_topic feed.
    client.subscribe(default_topic)

def disconnected(client, userdata, rc):
    # This method is called when the client is disconnected
    print('Disconnected from MQTT Broker!')

def message(client, topic, message):
    """Method callled when a client's subscribed feed has a new
    value.
    :param str topic: The topic of the feed with a new value.
    :param str message: The new value
    """
    print('New message on topic {0}: {1}'.format(topic, message))

# Connect to WiFi
wifi.connect()

# Set up a MiniMQTT Client
mqtt_client = MQTT(socket,
                   broker = secrets['broker'],
                   username = secrets['user'],
                   password = secrets['pass'],
                   network_manager = wifi)

# Setup the callback methods above
mqtt_client.on_connect = connected
mqtt_client.on_disconnect = disconnected
mqtt_client.on_message = message

# Connect the client to the MQTT broker.
mqtt_client.connect()

# Start a blocking message loop
# If you only want to listen to incoming messages,
# you'll want to loop_forever as it handles network reconnections
# No code below this line will execute.
mqtt_client.loop_forever()
```

MiniMQTT Client Identifier

By default, MiniMQTT will generate a unique, randomly generated client identifier based off the CircuitPython device's microcontroller's UUID and a random number. The broker will see a client named something like *cpy-3123*

If you'd like to set a custom **client_id** (what the broker sees the CircuitPython device as), you can provide a string. Do make sure the client_id 's you create are unique, or your broker will disconnect them.

```
client =  MQTT(socket,
          broker = secrets['broker'],
          username = secrets['user'],
          password = secrets['pass'],
          network_manager = wifi,
          client_id = 'brentspyportal')
```

MiniMQTT Logging

MiniMQTT uses the CircuitPython logger module (https://adafru.it/Ehw) for printing different types of errors, depending on the priority the logger was set to.

To attach a logger to a MiniMQTT client, simply

```
client =  MQTT(socket,
          broker = secrets['broker'],
          username = secrets['user'],
          password = secrets['pass'],
          network_manager = wifi,
          log = True)
```

Then, you will need to add another line setting the logger's level. While the logger is initialized to the INFO level by default, you may want to see more information about your current MQTT session.

To set the logger to a higher priority logging level, like DEBUG, **add the following line** *after* the MQTT client has been initialized:

```
client.set_logger_level(DEBUG)
```

MiniMQTT Last Will and Testament

MiniMQTT supports setting publishing a message to a specific topic when your MQTT client disconnects.

- **For more information about MQTT's Last Will** - check this guide (https://adafru.it/Fmf).

To use the last will - specify the **topic** you'd like to publish to and provide it with a **message** to publish when the client disconnects.

```
client.last_will(topic='device/status/', message='Goodbye!')
```

This method must be called before the connect method. The **last will and testament also must be allowed by your MQTT Broker** - Adafruit IO does *not* support this feature.