Adafruit IO Basics: Schedule Actions

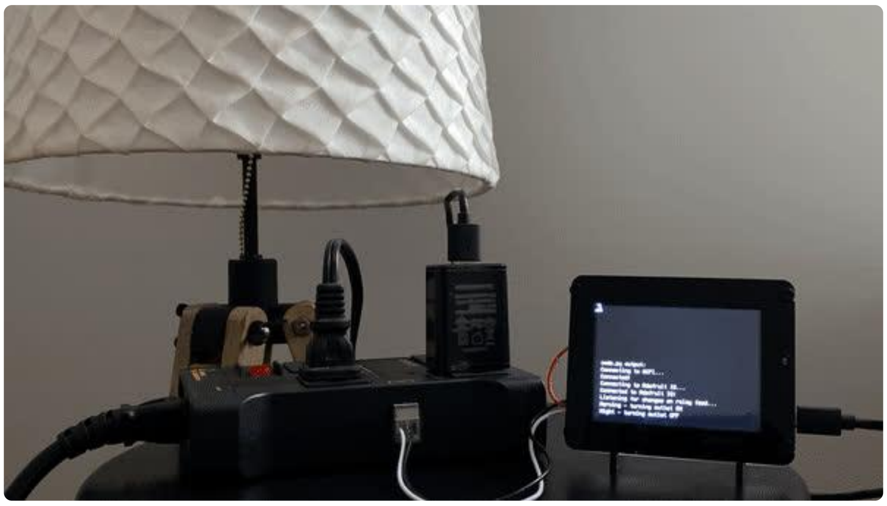
Created by Brent Rubell

https://learn.adafruit.com/adafruit-io-basics-scheduled-triggers

Last updated on 2022-12-01 03:57:03 PM EST

Table of Contents

Overview



Add simple scheduling to your projects with an Adafruit IO schedule action. Turn on or off lamps, fans, solenoids, and other small appliances without the headache of reading and parsing output from real-time-clock (RTC) or obtaining the network time.

This guide's project is a continuation of the IoT Power Outlet guide () where we built an internet-connected electrical outlet with an Adafruit PyPortal and connected it to Adafruit IO. This guide will take the IoT Power Outlet guide one step further by adding some scheduling logic to the outlet to turn a lamp on or off at a specific time or day.

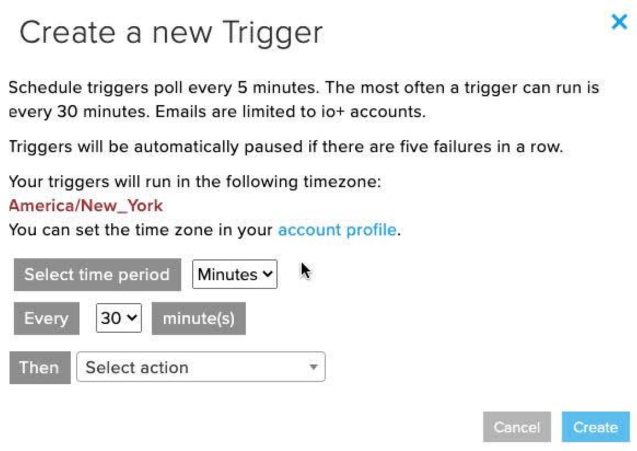
This project is not only for scheduling and automating lights - you may adapt it to control a fish feeder, turn off an interactive art exhibit at night, water your plants at specific times, or schedule anything powered by an A/C outlet.

What are Adafruit IO Actions?

Adafruit IO Actions add some lightweight logic to your IoT project without writing extra code. Actions are a way to do something when a certain situation occurs. This guide focuses on the simplest action type - scheduled actions.

What is a scheduled action?

You can configure a scheduled action to publish a value to a feed, send an email containing the value of a feed, or even send a webhook message to a URL at a specifi c time.



Create a new Trigger

Schedule triggers poll every 5 minutes. The most often a trigger can run is every 30 minutes. Emails are limited to io+ accounts.

Triggers will be automatically paused if there are five failures in a row.

Your triggers will run in the following timezone:
America/New_York
You can set the time zone in your account profile.

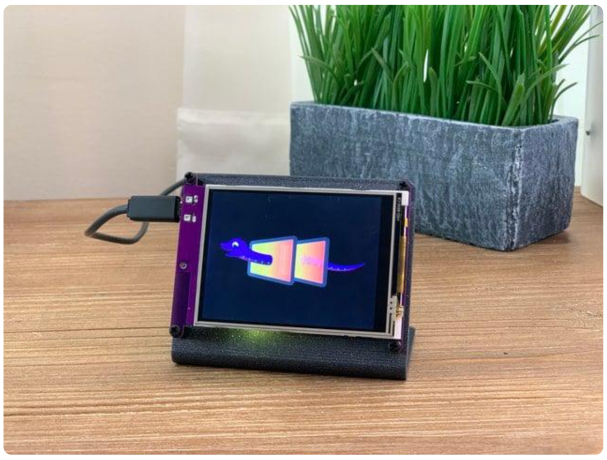
Select time period: Minutes
Every 30 minute(s)
Then: Select action

Scheduled triggers allow robust scheduling actions such as sending a value to a feed “At 2:00 PM, only on Friday”, “Every 4 hours”, or “At 1:00 PM, every 7 days”.

They also utilize your Adafruit IO account's local timezone - no complicated date-time math required!

Parts

You will need the following parts to complete this guide



Adafruit PyPortal - CircuitPython Powered Internet Display
PyPortal, our easy-to-use IoT device that allows you to create all the things for the “Internet of Things” in minutes. Make custom touch screen interface...
https://www.adafruit.com/product/4116

Controllable Four Outlet Power Relay Module version 2
Say goodbye to hazardous high voltage wiring and create the Internet of Things with safe, reliable power control....
https://www.adafruit.com/product/2935

STEMMA JST PH 2mm 3-Pin to Male Header Cable - 200mm
This cable will let you turn a JST PH 3-pin cable port into 3 individual wires with high-quality 0.1" male header plugs on the end. We're carrying these to match up with our...
https://www.adafruit.com/product/3893

5V 2.5A Switching Power Supply with 20AWG MicroUSB Cable
Our all-in-one 5V 2.5 Amp + MicroUSB cable power adapter is the perfect choice for powering single-board computers like Raspberry Pi, BeagleBone, or anything else that's...
https://www.adafruit.com/product/1995

1 x Micro-USB Cable
USB cable - USB A to Micro-B - 3 foot long
https://www.adafruit.com/product/592

Create a Schedule Action

Did you complete the IoT Power Outlet guide?

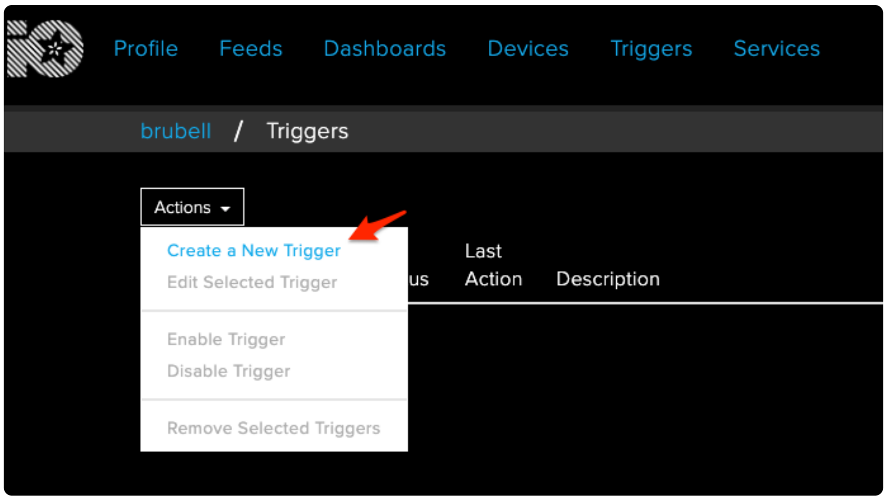
You will not be able to use this guide without completing the the IoT Power Outlet Guide ().

If you have not followed this guide, navigate to the IoT Power Outlet Guide's Adafruit IO setup page () and come back here when you are done.

Adding a Schedule Action

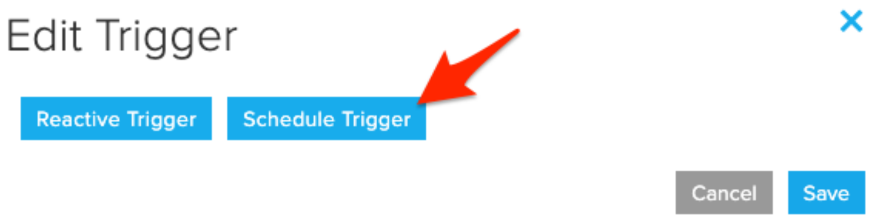
Log into Adafruit IO and navigate to the Adafruit IO Actions page ().

Click Actions -> Create a New Action.



A modal should appear, giving you the option of selecting a Reactive Action or a Schedule Action.

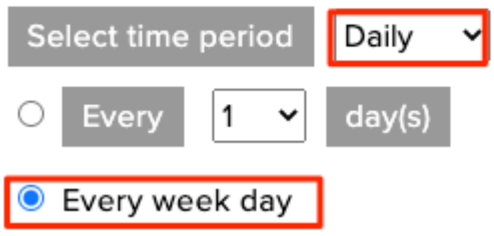
Click Schedule Action.



A schedule action can be activated based on specific times and dates such as “Tuesdays at 9pm” or “At 2:00 PM, only on Friday”, or “Every 4 hours”.

Let's turn the lights on every weekday morning at 8:00 AM to wake us up!

Set the time period to Daily and click Every week day.

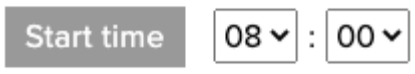


Select time period: Daily

○ Every 1 day(s)

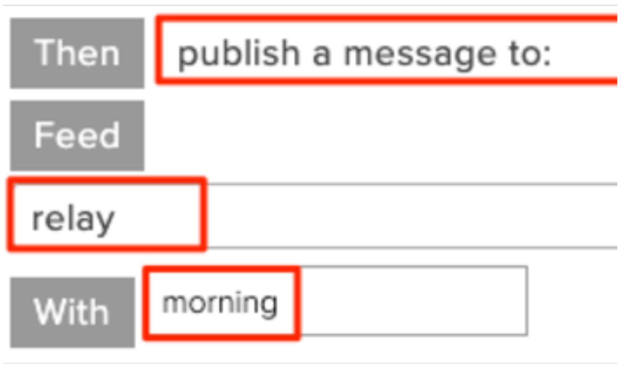
● Every week day

Then, set the start time to 08:00 or whenever you normally wake up.



Start time: 08 : 00

The CircuitPython or Arduino code will turn the outlet on depending on the feed's value at this time. Let's publish the value morning to the relay feed.



Then: publish a message to:
Feed: relay
With: morning

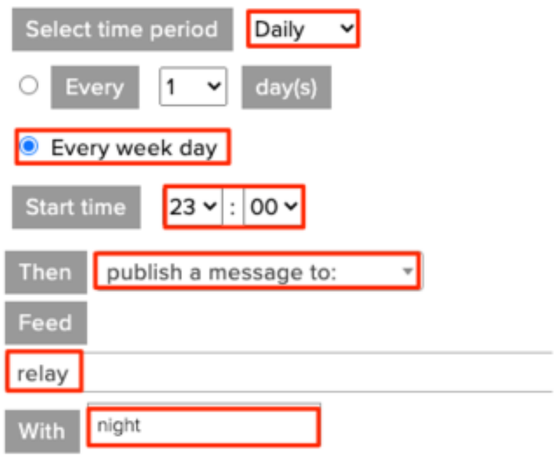
Set Then to publish a message to:

Select the relay feed

Enter morning as the value to be sent to the feed.

and Click Create

Let's turn off the lights after 11PM. To do this, you'll need to create another action.



Select time period: Daily
Every 1 day(s)
Every week day
Start time: 23 : 00
Then: publish a message to:
Feed: relay
With: night

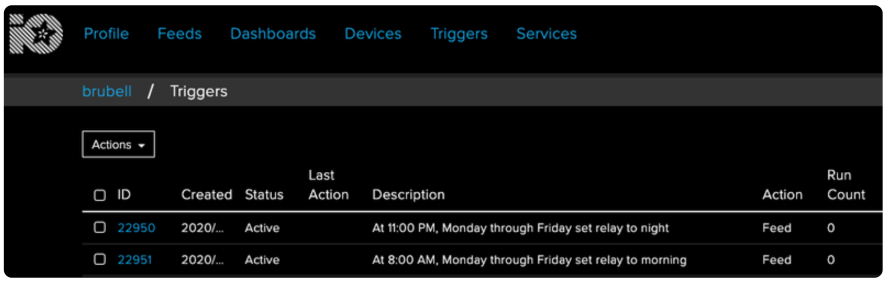
Select Daily as the time period.

Select Every week day

Set the Start time to 23:00

Then, configure the action to publish a message to the relay feed with the value night

Before moving on, make sure your actions page lists both the 8:00AM action and 11:00PM action.



ID	Created	Status	Last Action	Description	Action	Run Count
22950	2020/...	Active		At 11:00 PM, Monday through Friday set relay to night	Feed	0
22951	2020/...	Active		At 8:00 AM, Monday through Friday set relay to morning	Feed	0

Code with CircuitPython

Using CircuitPython is recommended for beginner to intermediate coders. It's more compact, easier to understand, and easy to change and debug.

Add CircuitPython Libraries

This page assumes your PyPortal is set up with the appropriate CircuitPython libraries and has been connected to the internet.

- If you have not done this, follow this page to install the correct libraries (). Then, follow this page to connect the PyPortal to the internet ().

Add CircuitPython Code

In the embedded code element below, click on the Download: Project Zip link, and save the .zip archive file to your computer.

Then, uncompress the .zip file, it will unpack to a folder named Adafruit_IO_Scheduled_Trigger.

Copy the contents of Adafruit_IO_Scheduled_Trigger directory to your PyPortal's CIR CUITPY drive.

```
# SPDX-FileCopyrightText: 2020 Brent Rubell for Adafruit Industries
#
# SPDX-License-Identifier: MIT

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

# Set up a pin for controlling the relay
power_pin = DigitalInOut(board.D3)
power_pin.switch_to_output()

# Define callback functions which will be called when certain events happen.
# pylint: disable=unused-argument
def connected(client):
    # Connected function will be called when the client is connected to Adafruit IO.
    # This is a good place to subscribe to feed changes.  The client parameter
    # passed to this function is the Adafruit IO MQTT client so you can make
    # calls against it easily.
    print("Connected to Adafruit IO!")


def subscribe(client, userdata, topic, granted_qos):
    # This method is called when the client subscribes to a new feed.
    print("Listening for changes on relay feed...")


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))


# pylint: disable=unused-argument
def disconnected(client):
    # Disconnected function will be called when the client disconnects.
    print("Disconnected from Adafruit IO!")


# pylint: disable=unused-argument
def on_message(client, feed_id, payload):
    # Message function will be called when a subscribed feed has a new value.
    # The feed_id parameter identifies the feed, and the payload parameter has
    # the new value.
    print("Feed {0} received new value: {1}".format(feed_id, payload))


def on_relay_msg(client, topic, message):
    # Method called whenever user/feeds/relay has a new value
    if message == "morning":
        print("Morning - turning outlet ON")
        power_pin.value = True
    elif message == "night":
        print("Night - turning outlet OFF")
        power_pin.value = False
    else:
        print("Unexpected value received on relay feed.")


# Connect to WiFi
print("Connecting to WiFi...")
wifi.connect()
print("Connected!")

# Initialize MQTT interface with the esp interface
MQTT.set_socket(socket, esp)

# Initialize a new MQTT Client object
mqtt_client = MQTT.MQTT(
    broker="io.adafruit.com",
    username=secrets["aio_username"],
    password=secrets["aio_key"],
)

# Initialize an Adafruit IO MQTT Client
io = IO_MQTT(mqtt_client)

# Connect the callback methods defined above to Adafruit IO
io.on_connect = connected
io.on_disconnect = disconnected
io.on_subscribe = subscribe
io.on_unsubscribe = unsubscribe
io.on_message = on_message

# Connect to Adafruit IO
print("Connecting to Adafruit IO...")
io.connect()

# Set up a message handler for the relay feed
io.add_feed_callback("relay", on_relay_msg)

# Subscribe to all messages on the relay feed
io.subscribe("relay")

# Get the most recent value on the relay feed
io.get("relay")

# Start a blocking loop to check for new messages
while True:
    try:
        io.loop()
    except (ValueError, RuntimeError, ConnectionError, OSError) as e:
        print("Failed to get data, retrying\n", e)
        wifi.reset()
        io.reconnect()
        continue
    time.sleep(0.5)
```

Secrets File Setup

Open the secrets.py file on your CircuitPython device using Mu or your favorite text editor. You're going to edit this file to enter your WiFi credentials along with your keys.

- Change `ssid` to the name of your WiFi network
- Change `password` to your WiFi network's password
- Change `aio_username` to your Adafruit IO Username
- Change `aio_key` to your Adafruit IO Key.

```
secrets = {
    'ssid' : 'home ssid',
    'password' : 'my password',
    'timezone' : "America/New_York", # http://worldtimeapi.org/timezones
    'aio_username' : 'MY_ADAFRUIT_IO_USERNAME',
    'aio_key' : 'MY_ADAFRUIT_IO_KEY',
    }
```

Code Usage

The PyPortal should boot up. The code connects the PyPortal to your WiFi network. Then, it connects to Adafruit IO and subscribes to the feed you set up earlier.

This code continuously checks the relay feed for new values.

Every weekday at 8AM, the scheduled action publishes the value "morning" to your relay feed.

When the feed updates with a new value, the `on_battery_msg()` function executes and checks if the value matches the text "morning".

If the feed's value matches the text "morning", the PyPortal's D3 pin is written high. This turns on the light connected to the outlet.

Every evening at 11PM, the scheduled trigger publishes the value "night" to your relay feed. The `on_battery_msg()` will read the new value, check if it matches the text "night", and turns off the outlet.

Troubleshooting

My appliance is not turning on or off.

First, check your wiring against the assembly and wiring page ().

Next, make sure your Adafruit IO feed is named relay and your triggers are set up to send the value morning and night every weekday.

Then, navigate to your Feeds page () and click the relay feed. If it's been a day since you set up the triggers, the values morning or night should be in the feed's value list. If they're not, check the action set up.

CircuitPython Code Walkthrough

The code sets up the PyPortal's D3 pin as a DigitalInOut object and set the direction to output.

```
# Set up a pin for controlling the relay
power_pin = DigitalInOut(board.D3)
power_pin.switch_to_output()
```

The following code block consists of MQTT callback methods. For an explanation of how these methods work, please see this section of the MQTT in CircuitPython guide ().

```
### 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!")

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

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

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 on_message(client, topic, message):
    # Method callled when a client's subscribed feed has a new value.
    print("New message on topic {0}: {1}".format(topic, message))
```

Whenever the relay feed receives new data, the `on_relay_msg` function executes. If the value on the feed is evaluated to be "morning", the light is turned on. Alternatively, if the value on the feed is evaluated to be "night", the light is turned off.

```
def on_relay_msg(client, topic, message):
    # Method called whenever user/feeds/relay has a new value
    if message == "morning":
        print("Morning - turning outlet ON")
        power_pin.value = True
    elif message == "night":
        print("Night - turning outlet OFF")
        power_pin.value = False
    else:
        print("Unexpected value received on relay feed.")
```

Next, the code connects to the WiFi network and set up a MQTT client to connect to Adafruit IO's MQTT broker with your credentials. An Adafruit IO MQTT client instance is initialized using the MiniMQTT client instance.

```
# Initialize MQTT interface with the esp interface
MQTT.set_socket(socket, esp)

# Initialize a new MQTT Client object
mqtt_client = MQTT.MQTT(
    broker="io.adafruit.com",
    username=secrets["aio_username"],
    password=secrets["aio_key"],
)

# Initialize an Adafruit IO MQTT Client
io = IO_MQTT(mqtt_client)
```

Sets up the callback methods above by connecting them to the client's default callback properties.

```
# Connect the callback methods defined above to Adafruit IO
io.on_connect = connected
io.on_disconnect = disconnected
io.on_subscribe = subscribe
io.on_unsubscribe = unsubscribe
io.on_message = on_message
```

Connect to the Adafruit IO MQTT broker and subscribe to the relay feed.

```
# Connect to Adafruit IO
print("Connecting to Adafruit IO...")
io.connect()
```

The `on_relay_msg` method is not a default Adafruit IO callback method, so it's added as a custom callback. This method will execute `on_relay_msg` whenever a new value is obtained by the relay feed.

```
# Add a callback to the relay feed
client.add_topic_callback(feed_relay, on_relay_msg)
```

Subscribes to all messages on the relay feed and obtains the most recent value of the relay feed.

```
# Subscribe to all messages on the relay feed
io.subscribe("relay")
```

```
# Get the most recent value on the relay feed
io.get("relay")
```

The code within the `while True` loop will check for new messages on the relay feed every 50 milliseconds. If there's an issue with the network connection, the WiFi connection will reset and the MQTT client will reconnect to Adafruit IO.

```
# Start a blocking loop to check for new messages
while True:
    try:
        io.loop()
    except (ValueError, RuntimeError) as e:
        print("Failed to get data, retrying\n", e)
        wifi.reset()
        io.reconnect()
        continue
    time.sleep(0.5)
```

Code with Arduino

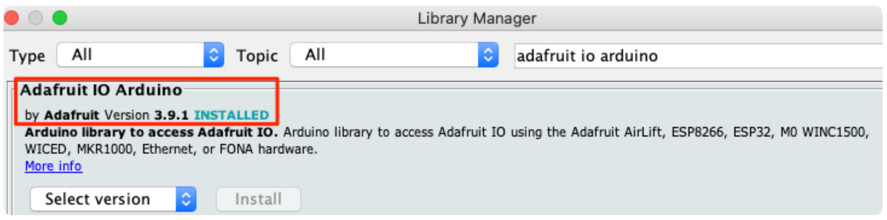
As an alternative to using CircuitPython, more advanced coders may wish to program the project using the Arduino IDE.

Setup Arduino

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.

- Adafruit PyPortal Arduino Setup Guide ()

You will need to make sure you have at least version 3.7.0 of the Adafruit IO Arduino library installed before continuing.



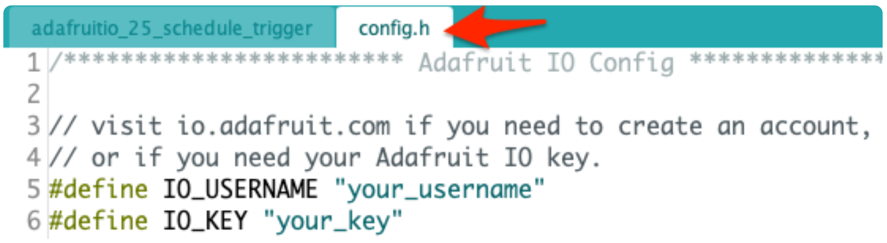
Adafruit IO Arduino
by Adafruit Version 3.9.1 INSTALLED
Arduino library to access Adafruit IO. Arduino library to access Adafruit IO using the Adafruit AirLift, ESP8266, ESP32, M0 WINC1500, WICED, MKR1000, Ethernet, or FONA hardware.

For this example you will need to open the adafruitio_25_scheduled_trigger example in the Adafruit IO Arduino library.

Configure Sketch

Before uploading the code, you'll need to configure the sketch to include your Adafruit IO account and network credentials.

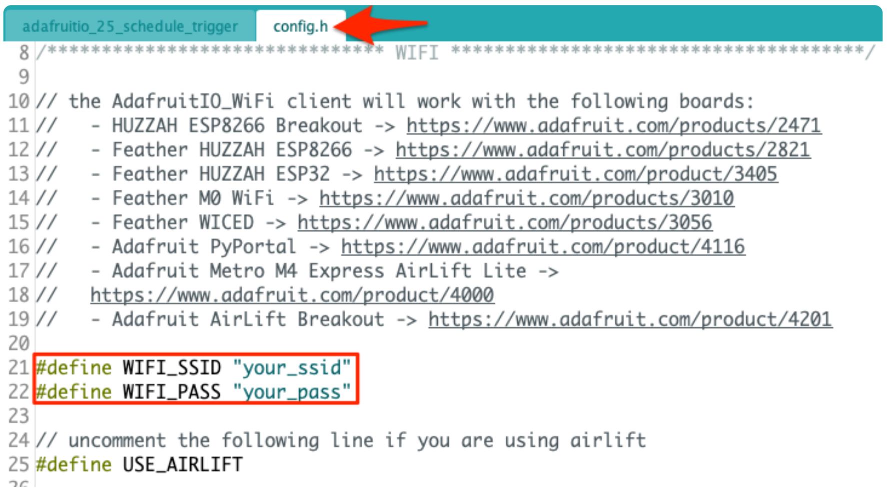
Click on the config.h tab in the sketch. You will need to set your Adafruit IO username in the `IO_USERNAME` define, and set your Adafruit IO key in the `IO_KEY` define.



```
/************************ Adafruit IO Config ************

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

Set your WiFi SSID after the `WIFI_SSID` define and your WiFi password after the `WIFI_PASS` define.



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

// the AdafruitIO_WiFi client will work with the following boards:
//   - HUZZAH ESP8266 Breakout -> https://www.adafruit.com/products/2471
//   - Feather HUZZAH ESP8266 -> https://www.adafruit.com/products/2821
//   - Feather HUZZAH ESP32 -> https://www.adafruit.com/product/3405
//   - Feather M0 WiFi -> https://www.adafruit.com/products/3010
//   - Feather WICED -> https://www.adafruit.com/products/3056
//   - Adafruit PyPortal -> https://www.adafruit.com/product/4116
//   - Adafruit Metro M4 Express AirLift Lite ->
//   https://www.adafruit.com/product/4000
//   - Adafruit AirLift Breakout -> https://www.adafruit.com/product/4201

#define WIFI_SSID "your_ssid"
#define WIFI_PASS "your_pass"

// uncomment the following line if you are using airlift
#define USE_AIRLIFT
```

You will also need to uncomment the line:

`#define USE_AIRLIFT`

Code Usage

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.
```

This sketch continuously checks the relay feed for new values.

Every weekday at 8AM, the scheduled action publishes the value "morning" to your relay feed.

When the feed updates with a new value, the sketch's `handleMessage()` function executes and checks if the value matches the text "morning".

If the feed's value matches the text "morning", the PyPortal's D3 pin is written HIGH. This turns on the outlet and your appliance.

Every evening at 11PM, the scheduled action publishes the value "night" to your relay feed. The `handleMessage()` will read the new value, check if it matches the text "night", and turns off the outlet.

Troubleshooting

My appliance is not turning on or off.

First, check your wiring against the assembly and wiring page ().

Next, make sure your Adafruit IO feed is named relay and your actions are set up to send the value morning and night every weekday.

Then, navigate to your Feeds page () and click the relay feed. If it's been a day since you set up the actions, the values morning or night should be in the feed's value list. If they're not, check the action set up.

Arduino Code Walkthrough

The code uses the PyPortal's D3 pin to control the relay's power signal.

```
// Relay is connected to PyPortal's D3 connector
#define RELAY_POWER_PIN 3
```

Next, the code sets up an instance of the relay feed you created in the previous guide.

```
// Set up the 'relay feed'
AdafruitIO_Feed *relay = io.feed("relay");
```

This chunk of code connects your device to Adafruit IO. It sets up a message handler for the relay feed. Whenever a new value is received on the relay feed, the `handleMessage` function will execute.

The code performs a `relay->get()` to obtain the last known value of the relay feed. This line is useful because if you lose connection for any reason, the code will set up the appliance in the correct state when it restarts.

```
void setup() {

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

  // wait for serial monitor to open
  while(! Serial);
  Serial.print("Connecting to Adafruit IO");

  // connect to io.adafruit.com
  io.connect();

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

  // Get the last known value from the feed
  relay-&gt;get();

}
```

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 the loop in every sketch. It helps keep a device connected to Adafruit IO, and processes any incoming data from the feeds its subscribed to.

```
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();

}
```

Whenever a new value is received on the relay feed, the `handleMessage` function will execute. This function prints out the new data it received

```
void handleMessage(AdafruitIO_Data *data) {

  Serial.print("feed received new data &lt;- ");
  Serial.println(data-&gt;toChar());
```

Checks if the data received by the feed matches the value published to the morning scheduled action. If there's a match, the signal pin is written `HIGH` which turns on the outlet.

```
// Check to see if the morning scheduled trigger has executed
  if (strcmp(data-&gt;toChar(), "morning") == 0) {
      Serial.println("Turning lights ON");
```

```
        digitalWrite(RELAY_POWER_PIN, HIGH);
    }
```

If the morning action value was not found, check if the evening action has fired. If the evening action value was found, turn off the outlet.

Any unexpected data received by the relay feed will print an error message.

```
// Check to see if the evening scheduled action has executed
  else if (strcmp(data-&gt;toChar(), "night") == 0) {
      Serial.println("Turning lights OFF");
      digitalWrite(RELAY_POWER_PIN, LOW);
  }
  else {
      Serial.println("Unexpected data received from Adafruit IO");
  }
}
```