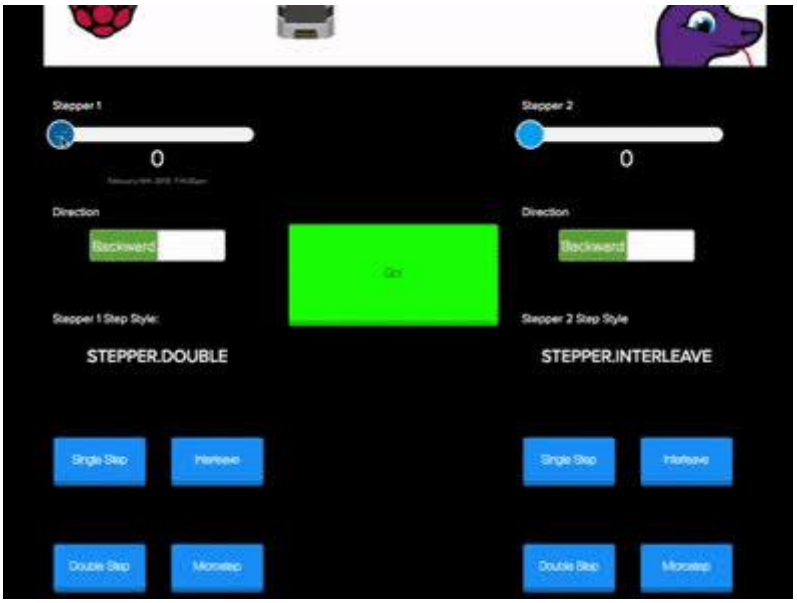
Wireless Dual Stepper Control with Adafruit IO, Raspberry Pi and Python

Created by Brent Rubell

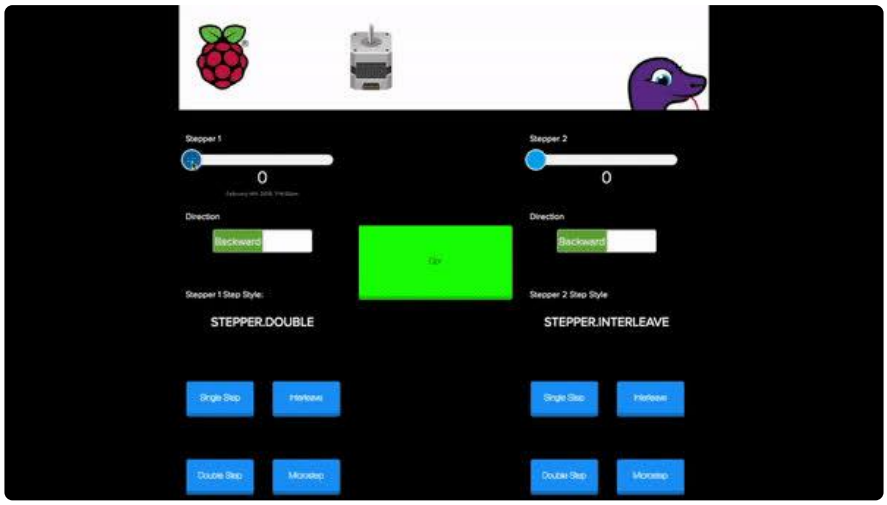


https://learn.adafruit.com/wireless-stepper-control-with-adafruit-io-circuitpython-raspberry-pi-python

Last updated on 2022-12-01 03:29:48 PM EST

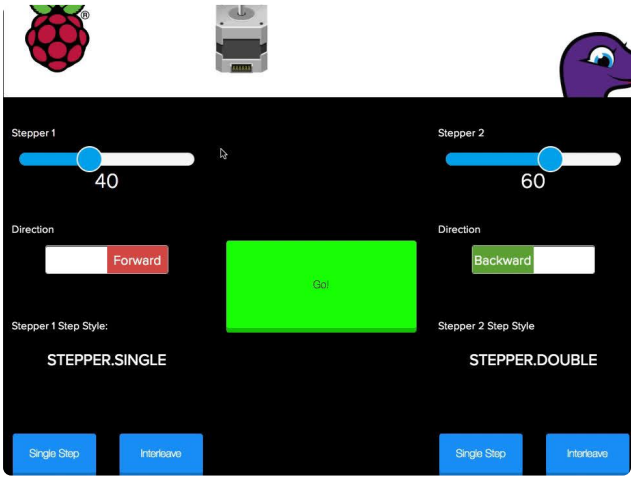
Table of Contents

Overview



Easily bring your Raspberry-Pi-controlled stepper motor projects online using the Motor Hat, CircuitPython libraries, and Adafruit IO. You can wirelessly control up to tw o steppers from a single Adafruit IO Dashboard!

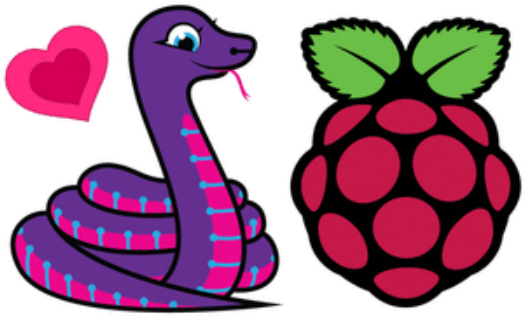
If you're building a home automation or robotics project which needs to be connected to the internet - look no further.



Adafruit IO

Adafruit IO is our Internet of Things Platform - for everyone! You'll create feeds to store the stepper's configuration: speed, step style, and direction.

To control it all, you'll build an interactive Adafruit IO Dashboard with buttons, switches, and toggles.



CircuitPython Code

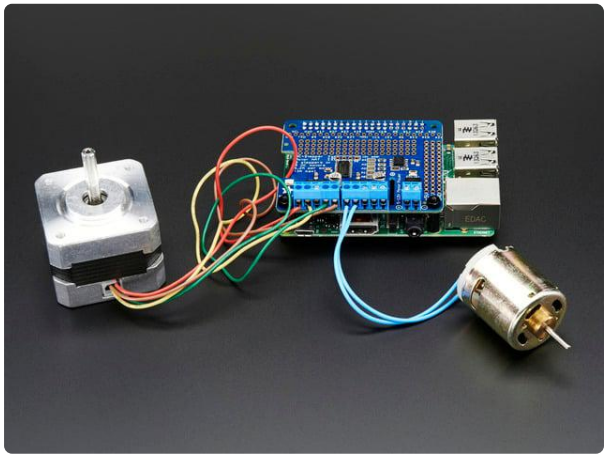
Adafruit's Circuit Python works on the Raspberry Pi! You will use the CircuitPython () MotorKit () library to control two stepper motors connected to the DC & Stepper Motor Hat. Connect them to the internet using the Adafruit IO Python () library. With Adafruit IO, you can upload new stepper motor configurations and control the steppers attached to the Raspberry Pi - wirelessly! You can rapidly update your code without having to compile and there's plenty of libraries, examples, and support.

Prerequisite Guides

If you're new to Adafruit IO or CircuitPython libraries, take a moment to walk through the following guides to get you started.

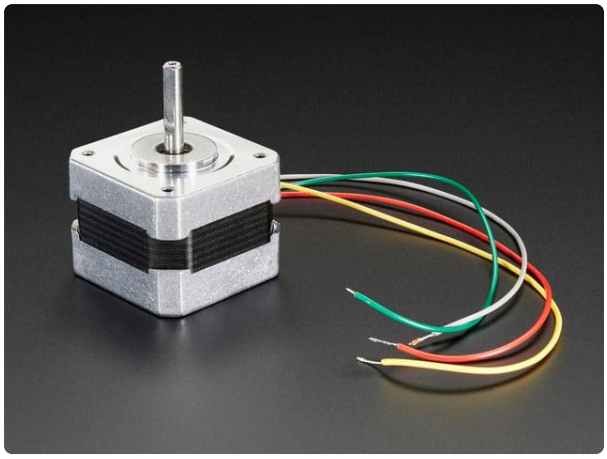
- Circuit Python libraries on Raspberry Pi ()
- Welcome to Adafruit IO ()

Parts



Adafruit DC & Stepper Motor HAT for Raspberry Pi - Mini Kit
Let your robotic dreams come true with the new DC+Stepper Motor HAT from Adafruit. This Raspberry Pi add-on is perfect for any motion project as it can drive up to 4 DC or 2 Stepper...
https://www.adafruit.com/product/2348

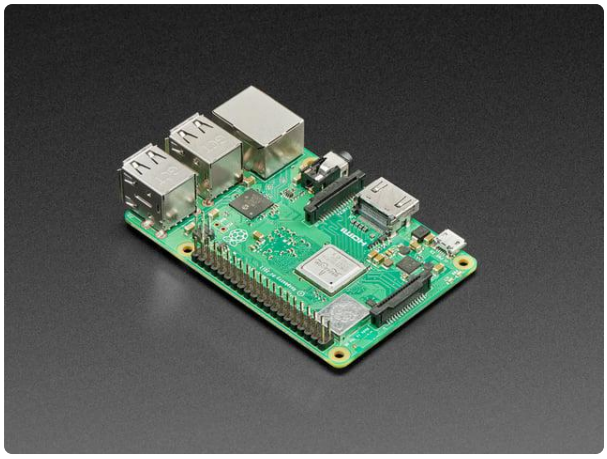
You'll want to pick up two of these NEMA-17 size stepper motors.



Stepper motor - NEMA-17 size - 200 steps/rev, 12V 350mA

A stepper motor to satisfy all your robotics needs! This 4-wire bipolar stepper has 1.8° per step for smooth motion and a nice holding torque. The motor was specified to have a max...

https://www.adafruit.com/product/324



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

NOTE: Due to stock limitations we may only be able to offer refunds or store credit for Pis that are defective, damaged or lost in transit.The Raspberry Pi...

https://www.adafruit.com/product/3775

Materials

You'll need some extra supplies to finish this project. If you do not have them already, pick some up from Adafruit:

1 x DC Power Adapter — https://www.adafruit.com/product/368

Female DC Power adapter - 2.1mm jack to screw terminal block

---

1 x MicroSD Card with OS — https://www.adafruit.com/product/1583

16GB Card with NOOBS 2.9

---

1 x Power Supply — https://www.adafruit.com/product/352

12V 5A switching power supply

---

1 x Power Supply with MicroUSB — https://www.adafruit.com/product/1995

5V 2.5A Switching Power Supply with 20AWG MicroUSB Cable

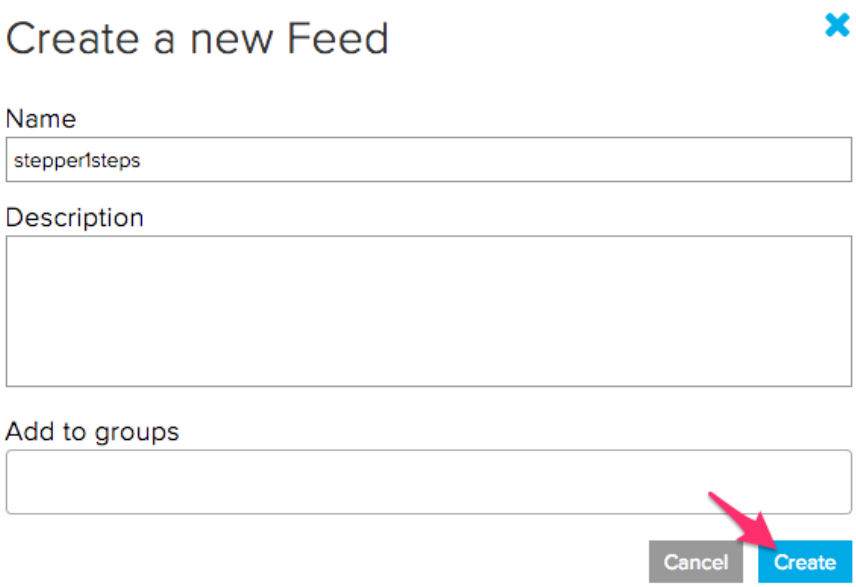
Adafruit IO Setup

If you do not already have an Adafruit IO account set up, head over to io.adafruit.com () to link your Adafruit.com account to Adafruit IO.

Feeds Setup

The first step is to create a new Adafruit IO feed to hold the stepper motor's number of steps. Navigate to the feeds page () on Adafruit IO. Then click Actions -> Create New Feed, and name this feed stepper1steps.

- If you do not already know how to create a feed, head over to Adafruit IO Basics: Feeds ().



You'll also want to create two more feeds to hold the first stepper's direction and the step size:

- stepper1direction
- stepper1stepsize

Then, create three more feeds (for the second stepper):

- stepper2steps
- stepper2direction

- stepper2stepsize

and one to control the stepper 'Go!' button

- stepperstart

Dashboard Setup

Next, step is to create an interactive dashboard to send data to the stepper motors from the Adafruit IO website.

- If you do not know how to create or use Dashboards in Adafruit IO, head over to the Adafruit IO Basics: Dashboards () guide.

Adding Sliders for Stepper Step Amount

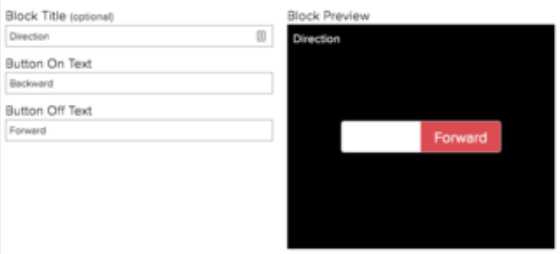
To control the amount of steps each stepper takes, add a slider block () to the dashboard.

Set Choose Feed to stepper1steps
Set Block Title to Stepper 1
Set Slider Min Value to 0
Set Slider Max Value to 100
Set Slider Step Size to 10 - the stepper will step by 10 steps each time the slider is toggled.

After setting up the first slider block, set up a second one attached to the stepper2steps feed to control stepper 2.

Adding Toggle Blocks for Stepper Direction

To control the direction of the stepper, add a toggle block () to the dashboard.



Set Choose Feed to stepper1direction
Set Block Title to Direction
Set Button On Text to Backward
Set Button Off Text to Forward

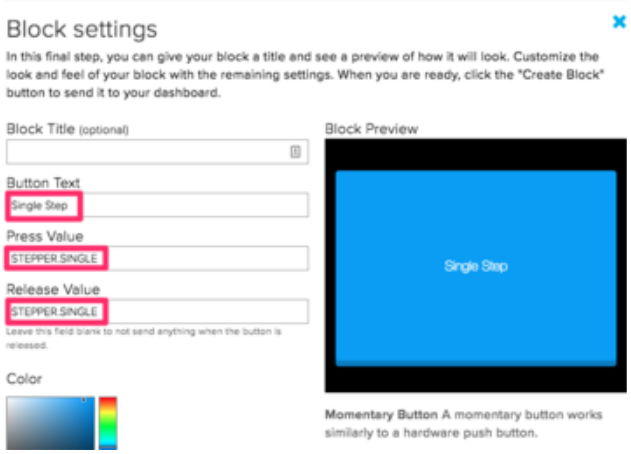
After setting up the first toggle block, set up a second one attached to the stepper2direction feed to control stepper 2.

Adding Buttons

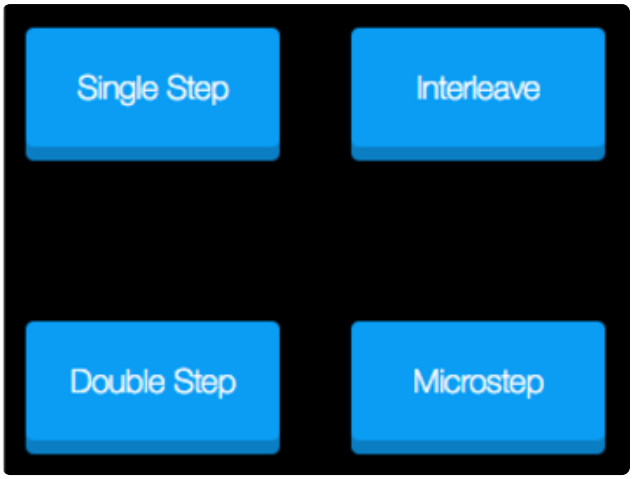
To send the stepper motor configurations to the stepper motors - create a momentary push-button block:

Set Choose Feed to stepperstart
Set Button Text to Go
Set Press Value to 1
Set Release Value to 0

Stepper motors can use four different step styles (). Use a momentary button () to configure the style of step.



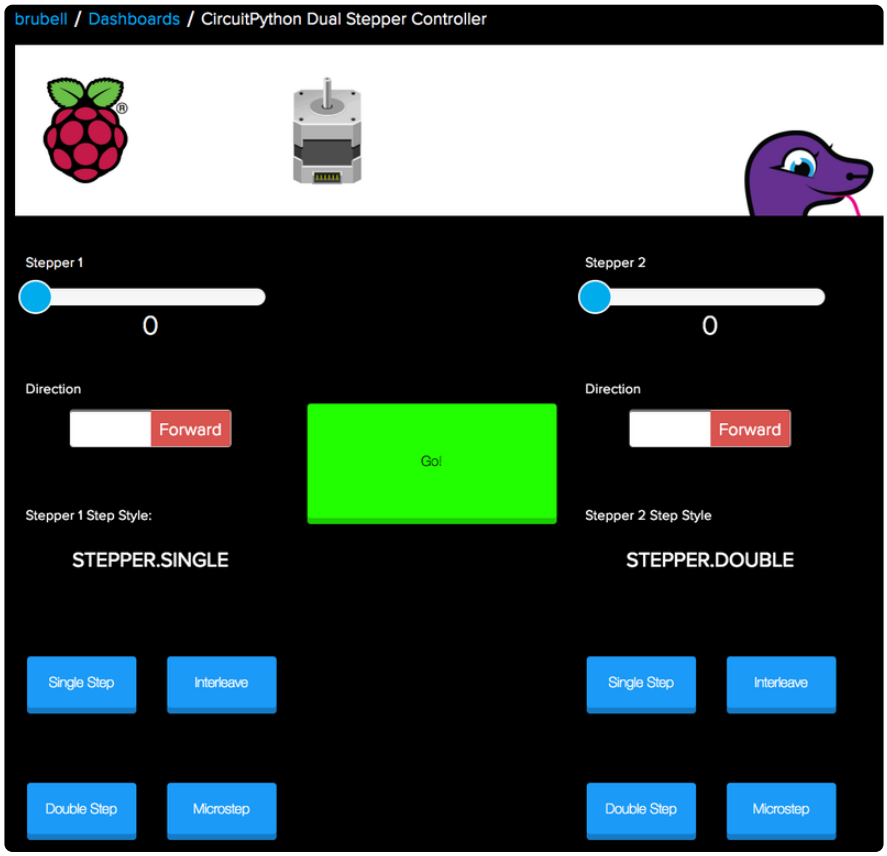
Set Choose Feed to stepper1stepsize
Set Button Text to Single Step
Set Press Value to STEPPER.SINGLE
Set Release Value to STEPPER.SINGLE



After setting up the first momentary button, you'll want three more. The press/release values should be set accordingly:

Double Step -> STEPPER.DOUBLE
Interleave -> STEPPER.INTERLEAVE
Microstep -> STEPPER.MICROSTEP
You'll also want to make four more buttons for the second stepper (stepper2stepsize feed)

Here's an example of a complete dashboard for this project:

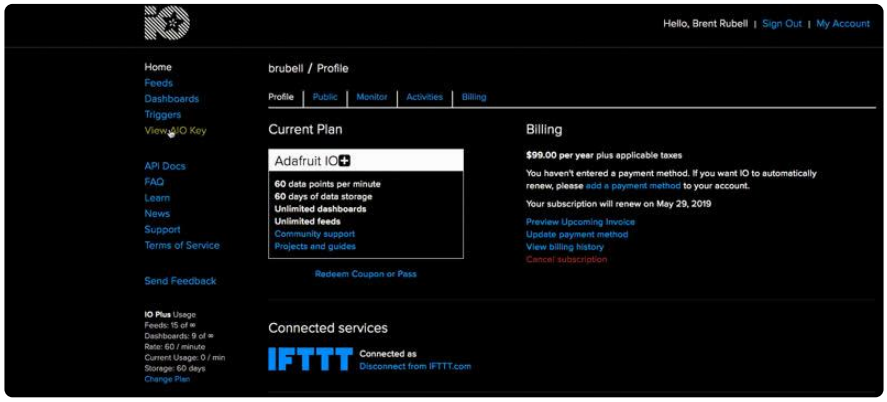


If you're an IO Plus user, you can set an image as an dashboard header. If you'd like to use the header created for this guide, right-click the image below to download it.

Saving your IO Keys

You are also going to need your Adafruit IO username and secret API key.

Navigate to your profile and click the View AIO Key button to retrieve them. Write them down in a safe place, you'll need them for later.

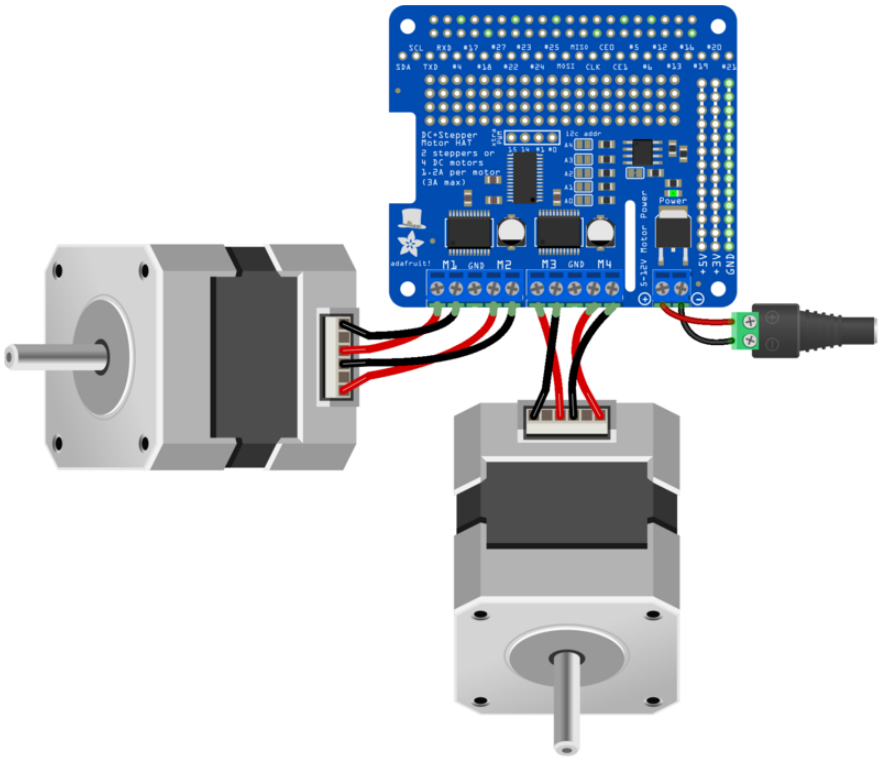


Wiring

Before you start wiring the steppers, there's a bit of soldering and assembly required.

- If you have not yet soldered the headers and terminal blocks onto the hat, click here for instructions ().

Before wiring the steppers, make sure the Motor Hat is attached to the Pi Header and that the Pi and the Motor Hat are powered off.



With the Motor Hat and the Raspberry Pi unplugged, make the following connections between the DC/Stepper Motor Hat and Stepper Motor 1:

- Stepper 1 Red to Hat M1+
- Stepper 1 Yellow to Hat M1-
- Stepper 1 Green to Hat M2+
- Stepper 1 Gray to Hat M2-

Make the following connections between the DC/Stepper Motor Hat and Stepper Motor 2:

- Stepper 2 Red to Hat M3+
- Stepper 2 Yellow to Hat M3-
- Stepper 2 Green to Hat M4+
- Stepper 2 Gray to Hat M4-

Then, connect the Female DC Power Adapter to the power terminal block on the hat.

- Unsure how to power the Motor Hat? Check out this page for more information ().

Python Code

This guide assumes that you have a Raspberry Pi connected to the Internet with CircuitPython installed.

- If you have not done this yet, follow this guide and come back to this page when you're set up. ()

Install Python Libraries

Your Raspberry Pi has Python already.

Visit the Motor HAT setup page to setup your HAT and make sure that works first! ()

Then, installing the library to control the Pi Motor Hat kit is fairly simple:

From a terminal, enter the following to install the CircuitPython_MotorKit () library

```
sudo pip3 install adafruit-circuitpython-motorkit
```

To communicate with Adafruit IO, you'll need to install the Adafruit IO Python () library:

```
sudo pip3 install adafruit-io
```

Code Configuration

Before you run the code, you'll need to configure it for your Adafruit IO account.

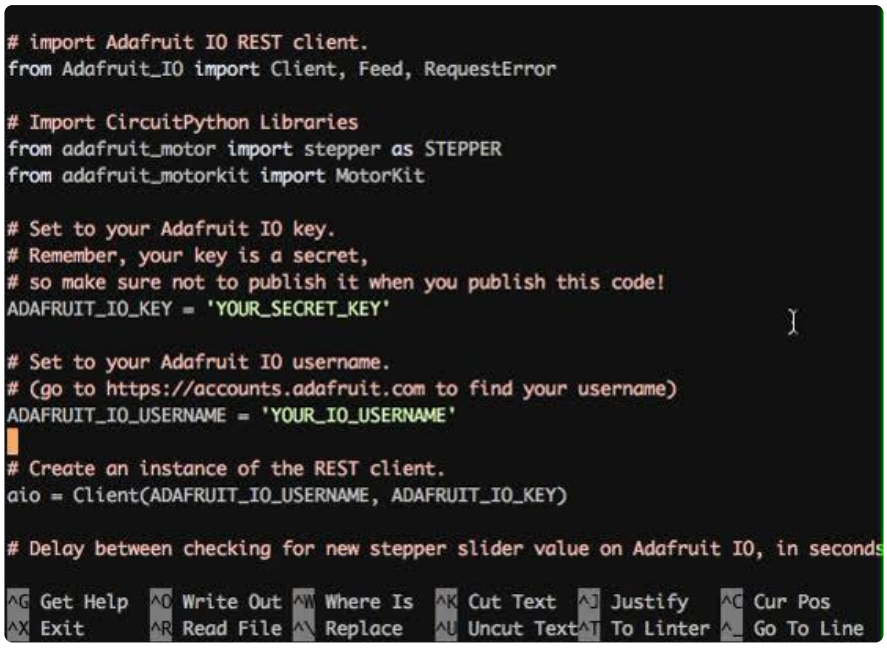
Open whichever text editor you'd like from the Raspberry Pi's command line (this example uses the `nano` editor) :

```
nano adafruit_io_steppers.py
```

To edit the file, set the `ADAFRUIT_IO_KEY` variable to the secret Adafruit IO key you saved earlier.

Set the `ADAFRUIT_IO_USERNAME` to your Adafruit IO username.

Then, save the file (CTRL+X, followed by Enter)




```
# SPDX-FileCopyrightText: 2019 Brent Rubell for Adafruit Industries
#
# SPDX-License-Identifier: MIT

"""
'adafruit_io_steppers.py'
==================================
Example of using CircuitPython and
Adafruit IO to control two stepper
motors over the internet.

Dependencies:
    - Adafruit_Blinka
        (https://github.com/adafruit/Adafruit_Blinka)
    - Adafruit_CircuitPython_MotorKit
        (https://github.com/adafruit/Adafruit_CircuitPython_MotorKit)
    - Adafruit_IO_Python
        (https://github.com/adafruit/Adafruit_IO_Python)
"""
# Import Python Libraries
import time
import atexit
import threading

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

# Import CircuitPython Libraries
from adafruit_motor import stepper as STEPPER
from adafruit_motorkit import MotorKit

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

# Delay between checking for `go` button press on Adafruit IO, in seconds
```

```
ADAFRUIT_IO_DELAY = 1

# Stepper 1 Adafruit IO Feeds
feed_step_1_steps = aio.feeds('stepper1steps')
feed_step_1_direction = aio.feeds('stepper1direction')
feed_step_1_step_size = aio.feeds('stepper1stepsize')
# Stepper 2 Adafruit Feeds
feed_step_2_steps = aio.feeds('stepper2steps')
feed_step_2_direction = aio.feeds('stepper2direction')
feed_step_2_step_size = aio.feeds('stepper2stepsize')
# Steppers start button
feed_steppers_status = aio.feeds('stepperstart')

# create a default object, no changes to I2C address or frequency
kit = MotorKit()

# create empty threads (these will hold the stepper 1 and 2 threads)
# pylint: disable=bad-thread-instantiation
st1 = threading.Thread()
st2 = threading.Thread()

# recommended for auto-disabling motors on shutdown!
def turnOffMotors():
    kit.stepper1.release()
    kit.stepper2.release()

atexit.register(turnOffMotors)
stepstyles = [STEPPER.SINGLE, STEPPER.DOUBLE, STEPPER.INTERLEAVE, STEPPER.MICROSTEP]

def stepper_worker(stepper, numsteps, direction, stepper_name, style,
show_steps=False):
    print("Steppin!")
    stepper_steps = numsteps
    print(stepper_steps)
    for _ in range(numsteps):
        stepper.onestep(direction=direction, style=style)
        if show_steps: # print out the steps and send to IO stepper slider
            stepper_steps -= 1
            print('Steps: ', stepper_steps)
            aio.send(feed_step_1_steps.key, stepper_steps)
            time.sleep(0.5)
    print("{0} Done Stepping".format(stepper_name))
    # Reset slider on dashboard
    if stepper_name == "Stepper 1":
        aio.send(feed_step_1_steps.key, 0)
    elif stepper_name == "Stepper 2":
        aio.send(feed_step_2_steps.key, 0)


while True:
    try: # attempt to poll the stepper status feed
        print('checking for GO button press...')
        stepper_start = aio.receive(feed_steppers_status.key)
    except RequestError.ThrottlingError:
        print('Exceeded the limit of Adafruit IO requests, delaying 30 seconds...')
        time.sleep(30)

    # Stepper 1
    if not st1.isAlive() and int(stepper_start.value):
        stepper_1_steps = aio.receive(feed_step_1_steps.key)
        stepper_1_steps = int(stepper_1_steps.value)
        if stepper_1_steps > 0: # stepper slider is set
            # Get stepper configuration from io feeds
            stepper_1_direction = aio.receive(feed_step_1_direction.key)
            stepper_1_step_size = aio.receive(feed_step_1_step_size.key)
            print('Stepper 1 Configuration')
            print('\t%d steps' % stepper_1_steps)
            print('\tStep Size: ', stepper_1_step_size.value)
            print('\tStepper Direction: ', stepper_1_direction.value)
```

```
            # Set Stepper Direction
            if stepper_1_direction.value == 'Forward':
                move_dir = STEPPER.FORWARD
            elif stepper_1_direction.value == 'Backward':
                move_dir = STEPPER.BACKWARD
            # Stepper 1 Thread
            st1 = threading.Thread(target=stepper_worker, args=(kit.stepper1,
                                                                stepper_1_steps,
                                                                move_dir,
                                                                "Stepper 1",

stepstyles[STEPPER.SINGLE],))
            st1.start()

    # Stepper 2
    if not st2.isAlive() and int(stepper_start.value):
        stepper_2_steps = aio.receive(feed_step_2_steps.key)
        stepper_2_steps = int(stepper_2_steps.value)
        if stepper_2_steps > 0: # stepper slider is set
            # Get stepper configuration from io feeds
            stepper_2_direction = aio.receive(feed_step_2_direction.key)
            stepper_2_step_size = aio.receive(feed_step_2_step_size.key)
            print('Stepper 2 Configuration')
            print('\t%d steps' % stepper_2_steps)
            print('\tStep Size: ', stepper_2_step_size.value)
            print('\tStepper Direction: ', stepper_2_direction.value)
            # Set Stepper Direction
            if stepper_2_direction.value == 'Forward':
                move_dir = STEPPER.FORWARD
            elif stepper_2_direction.value == 'Backward':
                move_dir = STEPPER.BACKWARD
            # Stepper 2 Thread
            st2 = threading.Thread(target=stepper_worker, args=(kit.stepper2,
                                                                stepper_2_steps,
                                                                move_dir,
                                                                "Stepper 2",

stepstyles[STEPPER.SINGLE],))
            st2.start()

    # delay polling the `go button` to avoid Adafruit IO timeouts
    print('Delaying for {0} seconds...'.format(ADAFRUIT_IO_DELAY))
    time.sleep(ADAFRUIT_IO_DELAY)
```

Code Usage

From the Raspberry Pi terminal, run the code by typing the following:

`python3 Adafruit_IO_Stepper_Control/code.py`

and press enter

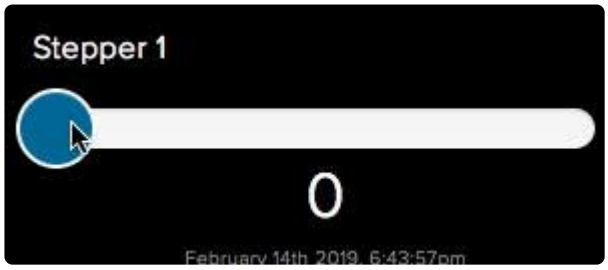
The terminal should output that it's checking the GO button, and delaying (to avoid sending too many fetch requests to Adafruit IO)

```
pi@stepper-pi:~ $ python3 dual_steppers.py
checking for GO button press...
Delaying for 1 seconds...
```

```
checking for GO button press...
Delaying for 1 seconds...
```

Next, navigate over to the Adafruit IO Dashboard to configure the first stepper.

Set the amount of steps by dragging the slider. The slider was configured to step in 10 step increments. You can re-configure to step more or less it by editing the block element.

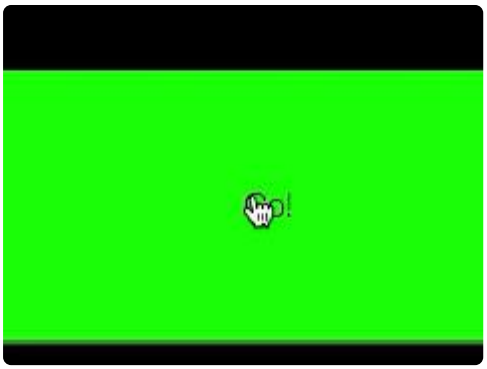


Set the stepper direction by toggling the switch. The stepper can either move forward (default) or backward.

Set the step style by pressing one of the four momentary push-buttons on the dashboard.

There are four essential types of steps you can use with your Motor HAT. All four kinds will work with any unipolar or bipolar stepper motor.

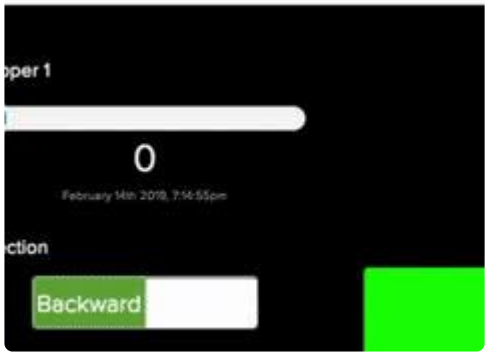
1. Single Steps - this is the simplest type of stepping, and uses the least power. It uses a single coil to 'hold' the motor in place, as seen in the animated GIF above
2. Double Steps - this is also fairly simple, except instead of a single coil, it has two coils on at once. For example, instead of just coil #1 on, you would have coil #1 and #2 on at once. This uses more power (approx 2x) but is stronger than single stepping (by maybe 25%)
3. Interleaved Steps - this is a mix of Single and Double stepping, where we use single steps interleaved with double. It has a little more strength than single stepping, and about 50% more power. What's nice about this style is that it makes your motor appear to have 2x as many steps, for a smoother transition between steps
4. Microstepping - this is where we use a mix of single stepping with PWM to slowly transition between steps. It's slower than single stepping but has much higher precision. We recommend 8 microstepping which multiplies the # of steps your stepper motor has by 8.



After the first stepper is configured, play with the controls for the second stepper.

When you're ready, press the green Go Button to send the presets to the Raspberry Pi and move the stepper motors.

The Raspberry Pi should configure and step the first stepper, then the second (if you configured it).

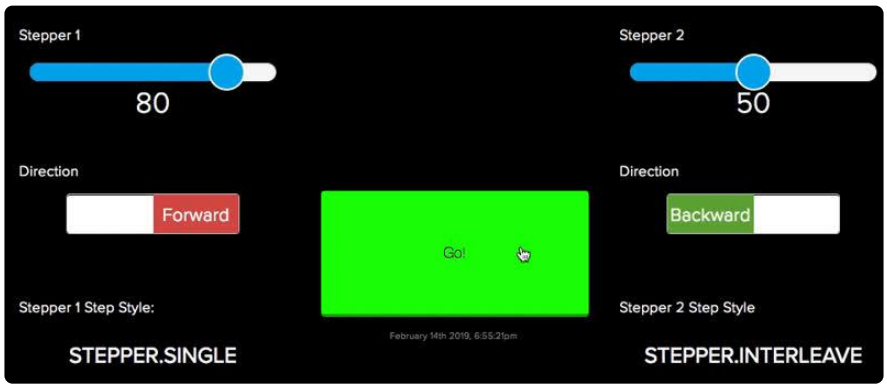


The terminal will also display the values it receives from Adafruit IO as it's stepping.

```
Stepper 1 Configuration
    70 steps
    Step Size:  STEPPER.DOUBLE
    Stepper Direction:  Forward
Steppin!
Stepper 1 Done Stepping

Stepper 2 Configuration
    60 steps
    Step Size:  STEPPER.DOUBLE
    Stepper Direction:  Backward
Steppin!
Stepper 2 Done Stepping
```

When it's done stepping, the stepper driver's code resets the step amount feed to zero. The slider, connected to the step amount feed, resets too.



Code Overview

Adafruit IO can receive up to 120 data points a minute, the code aims to minimize the amount of data polling which occurs in the `while True` loop.

First, it checks the stepper status feed, which is linked to the Go push-button on the dashboard. If the code exceeds the amount of requests it can send, it will wait thirty seconds for Adafruit IO to clear the timeout.

```
try: # attempt to poll the stepper status feed
    print('checking for GO button press...')
    stepper_start = aio.receive(feed_steppers_status.key)
except ThrottlingError:
    print('Exceeded the limit of Adafruit IO requests, delaying 30 seconds...')
    time.sleep(30)
```

The code for controlling both stepper motors is similar - but doesn't happen simultaneously (code for the first stepper occurs first).

First, the code checks if the stepper thread is alive (if the stepper is moving or not) and if the stepper status feed (`stepper_start`) was selected (the push-button was pressed). Then, it fetches the value from the Stepper Slider:

```
if not st1.isAlive() and int(stepper_start.value):
    stepper_1_steps = aio.receive(feed_step_1_steps.key)
    stepper_1_steps = int(stepper_1_steps.value)
```

If the value from the stepper slider is set, the code will fetch the stepper's direction and step size from Adafruit IO. Then, it'll call `stepper_worker` and pass in the arguments from the configuration.

```
        if stepper_1_steps &gt; 0: # stepper slider is set
            # Get stepper configuration from io feeds
            stepper_1_direction = aio.receive(feed_step_1_direction.key)
            stepper_1_step_size = aio.receive(feed_step_1_step_size.key)
            print('Stepper 1 Configuration')
            print('\t%d steps' % stepper_1_steps)
            print('\tStep Size: ', stepper_1_step_size.value)
            print('\tStepper Direction: ', stepper_1_direction.value)
            # Set Stepper Direction
            if stepper_1_direction.value == 'Forward':
                move_dir = STEPPER.FORWARD
            elif stepper_1_direction.value == 'Backward':
                move_dir = STEPPER.BACKWARD
            # Stepper 1 Thread
            st1 = threading.Thread(target=stepper_worker, args=(kit.stepper1,
                                                                stepper_1_steps,
                                                                move_dir,
                                                                "Stepper 1",

stepstyles[STEPPER.SINGLE],))
            st1.start()
```