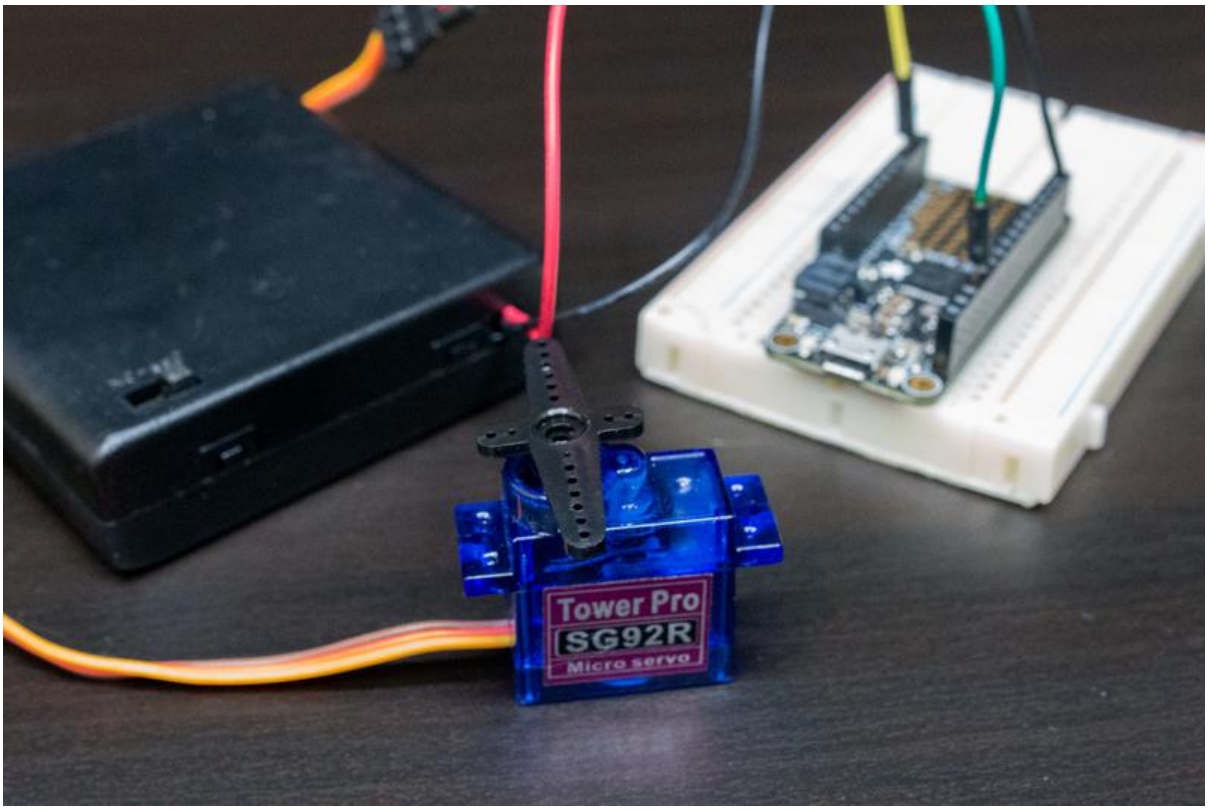
Using Servos With CircuitPython and Arduino

Created by Tony DiCola

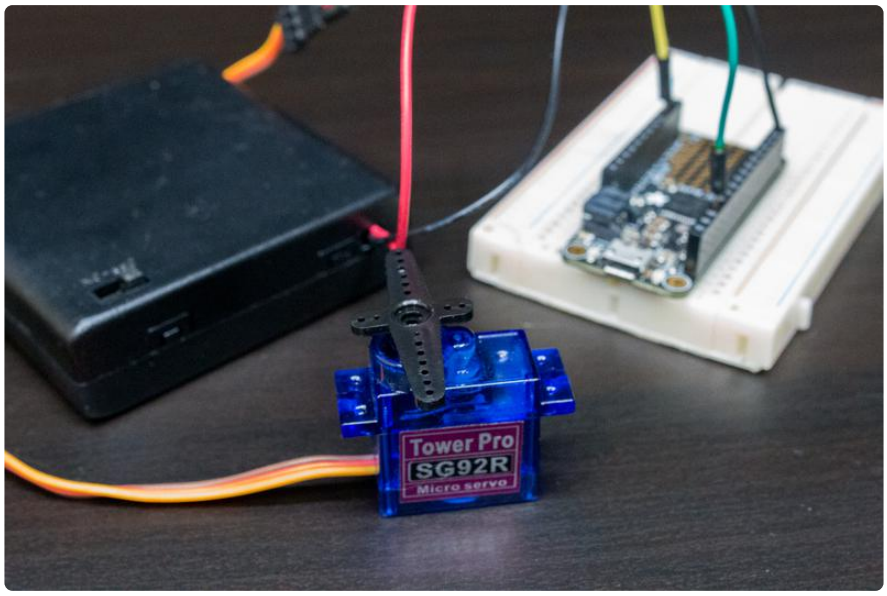


https://learn.adafruit.com/using-servos-with-circuitpython

Last updated on 2023-04-25 12:49:55 PM EDT

Table of Contents

Overview



Servos are tiny motors that you can control the position of by generating a special signal. You might use a servo to move something back and forth, like moving a dial to indicate a measurement or even moving a latch to open and close a door. There are even special 'continuous rotation' servos that can act like little motors with control over their speed and direction--perfect for building a simple robot! The possibilities for movement with servos are unlimited!

This guide will explore how to control servo motors from CircuitPython and Python code. You can use simple Python code to move a servo to different positions. You can even use the CircuitPython REPL to move a servo interactively!

In addition this guide will also show basic servo control with Arduino code too!

Familiarization

Before you get started it will help to familiarize yourself with servos by reading these guides:

- Adafruit Motor Selection Guide - Servo Motors ()
- Arduino Lesson 14: Servo Motors ()
- CircuitPython Essentials ()

Using Servos with the Adafruit CRICKIT Board

If you are using an Adafruit Circuit Playground Express, Adafruit Feather, BBC micro:bit or Raspberry Pi, you can control servos, motors, and more using the Adafruit CRICKIT add-on board.

CRICKIT has four ports for plugging in servo motors.

For more information on using CRICKIT and servos, see the following guide:

- Introducing Adafruit CRICKIT ()

Hardware

To follow this guide you'll need the following parts:

- A servo motor. Either a standard servo or 'continuous rotation' servo will work for this guide, but be sure it's a servo motor and not a DC motor (which requires a different method of driving it). Check out the Adafruit Motor Selection Guide () for more details on the different types of motors. If in doubt grab a small micro servo () to experiment with and follow this guide.
- A ~5 volt power supply for the servo. Servos and motors in general can pull a surprisingly large amount of power, particularly when they're stopped or meeting resistance. In some cases they can pull so much power that it damages or overwhelms your development board! To prevent issues like this it's highly recommended to use a separate power supply for the motors in your project. A 4x AA battery pack () is a perfect option for powering a few servos.
- A microcontroller running CircuitPython or a Raspberry Pi running Adafruit Blinka (). If you're controlling servos from CircuitPython you'll need a board like the Feather M0 basic () which can be loaded with CircuitPython firmware. Adafruit Blinka () enables you to use CircuitPython modules and libraries on Raspberry Pi and other single board computers. This guide will also show basic servo code for Arduino too.
- Breadboard () and jumper wires (). You'll need these parts to connect components to your development board.

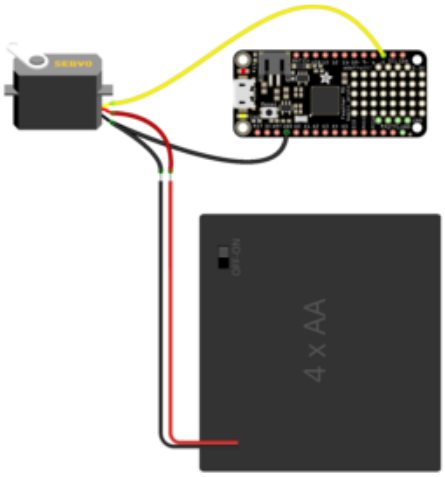
Wiring

Servo motors have three wires to connect to your board:

- Power - Typically a red wire which must be driven by 3-6 volts. However always check your servo datasheet or product information to make sure it's safe to power with the voltage you intend to provide. If the voltage is too high it might damage the servo!
- Ground - Typically a brown or black wire which must be connect to both your board and servo power supply grounds.
- Signal - Typically an orange or yellow wire. This wire is connected to a PWM or pulse-width modulation output on your development board. For boards like the Feather M0 look for pins with a squiggly line next to them to see those that support PWM output. For other boards check the product information or guide to see which outputs support PWM signals.

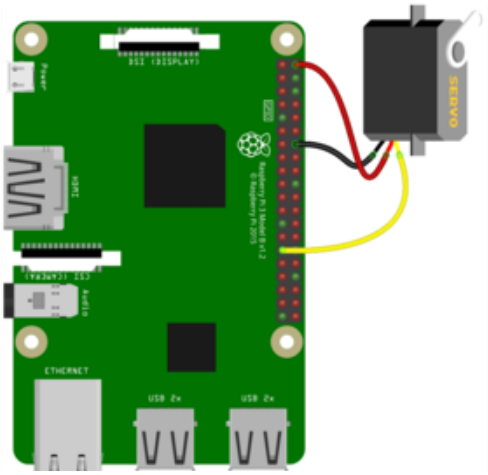
You might see a fourth wire coming from your servo motor too. Some advanced servos provide an output to help determine the position of the motor. You can ignore this fourth wire as it won't be used in this guide. However be absolutely sure you're using a servo motor and not a stepper or other type of motor if you see more or less than three wires coming from your device!

Here's an example of wiring a servo to a Feather M0 and a 4x AA power supply:



Servo power (red wire) to power supply positive voltage (red wire).
Servo ground (brown/black wire) to both power supply ground (black wire) and board GND.
Servo signal (orange/yellow wire) to board D5 (or any other output that supports PWM signals).

Here is an example of wiring a servo to a Raspberry Pi:



Servo power (red wire) to Raspberry Pi 5V
Servo ground (black/brown wire) to Raspberry Pi ground
Servo signal (yellow/white wire) to Raspberry Pi GPIO5

Wiring for Other Boards

See the guide CircuitPython Essentials - CircuitPython Servo () for wiring for the Adafruit Trinket, Gemma, Circuit Playground Express, Feather, and Metro.

CircuitPython

To control the servo from CircuitPython we'll use its built in PWM, or pulse-width modulation, signal generation capabilities. Be sure to read the CircuitPython analog I/O guide () for more details on PWM signals!

Before you get started it will help to read these guide pages for more background information on servo motors too:

- Adafruit Motor Selection Guide - Servo Motors ()
- Arduino Lesson 14: Servo Motors ()

Be sure your hardware is wired up as shown on the previous page, and your servo power supply is turned on (you might notice the servos jerk or move slightly when the power is turned on--that's normal!).

Next make sure you are running the latest version of Adafruit CircuitPython () for your board, then connect to the board's serial REPL ()so you are at the CircuitPython >>> prompt.

Pick Your Programming Method - High or Low Level

There's two ways to control servos, one is lower-level - where you will control the PWM pin directly. And one is higher-level using the Motor Library

For typical non-continuous servo motors, control is best done with the high level `adafruit_motor` library. The `servo` module provides an excellent interface and is easy to understand.

You can read more about using that library in the guide CircuitPython Essentials () on the CircuitPython Servo page ().

Low level control involves working with the Pulse Width Modulation (PWM) directly which is rarely required.

The following pages discuss high and low level control of servos in CircuitPython.

High Level Servo Control

Position Control with Motor Library

The best way to control servos is with a handy Adafruit CircuitPython Motor () module which simplifies setting the duty cycle to control servos (and even allows controlling servos from different PWM hardware like the PCA9685 board).

CircuitPython Library Install

To follow this approach you'll need to install the Adafruit CircuitPython Motor () library on your CircuitPython board.

First make sure you are running the latest version of Adafruit CircuitPython () 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 (). Our introduction guide has a great page on how to install the library bundle () for both express and non-express boards.

Remember for non-Express boards, you'll need to manually install the necessary libraries from the bundle:

- adafruit_motor

You can also download the adafruit_motor folder from its releases page on Github ().

Before continuing make sure your board's lib folder has the adafruit_motor folder copied over.

Now connect to the board's serial REPL ()so you are at the CircuitPython >>> prompt.

Python Library Install

On Linux, you'll need to install the `Adafruit_Blinka` library that provides the CircuitPython support in Python. Since each platform is a little different, and Linux changes often, please visit the CircuitPython on Linux guide to get your computer ready ()!

Once that's done, from your command line run the following command:

- `pip3 install adafruit-circuitpython-motor`

If your default Python is version 3 you may need to run 'pip' instead. Just make sure you aren't trying to use CircuitPython on Python 2.x, it isn't supported!

Controlling a Servo

Then import the `board` and `pwmio` modules as before. We'll create a PWM output just like as shown in the first section--the motor library will take this PWM output and add an extra level of simple control on top of it:

```
import board
import pwmio
pwm = pwmio.PWMOut(board.D5, frequency=50)
```

Now let's import the servo submodule from the `adafruit_motor` module and create an instance of the `Servo` class from it:

```
from adafruit_motor import servo
servo = servo.Servo(pwm, min_pulse=750, max_pulse=2250)
```

Notice the servo class needs to be told what PWM output the servo is connected to for your board. If you're following the wiring and this guide it will be a PWM output on board pin 5 (be sure to create this with a 50 hz frequency as shown!).

There are a few optional keyword parameters that you might specify in the initializer too. These aren't shown and are useful for using very specialized or custom servos with different ranges or pulse width values--for simple servos you don't typically need to set these values:

- actuation_range - The range in degrees of the servo movement. The default is 180 degrees.
- min_pulse - The minimum position pulse length in microseconds (default 1000 us).
- max_pulse - The maximum position pulse length in microseconds (default 2000 us).

Here we've change the minimum pulse from the default 1000 microseconds to 750, and the default maximum pulse from 2000 microseconds to 2250 to ensure we get the full sweep as some servos differ. Some experimentation may be required!

Controlling the servo is simple once you have the class created, just set the `angle` property to a value from 0 to 180 degrees!

```
servo.angle = 0
servo.angle = 90
servo.angle = 180
```

Notice each time you set angle the servo springs to life and moves!

If you don't see the servo moving be sure you have it wired and powered exactly as shown. Also make sure you've created the PWM output exactly as shown above too (note the 50 hz frequency!).

There's another type of servo class in the module that you might find useful too, the `ContinuousServo` class. This is for controlling continuous rotation servos that move completely around like a small motor instead of point at specific angles. For these servos you control their speed (or throttle). Create an instance just like with the `Servo` class above:

```
continuous = adafruit_motor.servo.ContinuousServo(pwm, min_pulse=750,
max_pulse=2250)
```

With these servos you set the `throttle` property to a value from -1 to 1 (or anything in between, including fractional values). Where -1 is fully speed 'backwards' and 1 is full speed 'forwards'. A value of 0 should stop the motor (but your servo might need to be trimmed a bit and have a small value set that stops its movement instead of zero, experiment with values yourself to see).

```
continuous.throttle = -1   # Full backwards
continuous.throttle = 0    # Stop
continuous.throttle = 1    # Full forwards
continuous.throttle = 0.5  # Half speed forwards
```

Examples

Note that the pin numbers are different in the examples below! You'll need to update them to match your wiring.

Standard Servo

Here's a complete standard servo sweep example written for the `adafruit_motor` library. Remember save this as code.py on your board to have it run:

```
# SPDX-FileCopyrightText: 2018 Kattni Rembor for Adafruit Industries
#
# SPDX-License-Identifier: MIT

"""CircuitPython Essentials Servo standard servo example"""
import time
import board
import pwmio
from adafruit_motor import servo

# create a PWMOut object on Pin A2.
pwm = pwmio.PWMOut(board.A2, duty_cycle=2 ** 15, frequency=50)

# Create a servo object, my_servo.
my_servo = servo.Servo(pwm)

while True:
    for angle in range(0, 180, 5):  # 0 - 180 degrees, 5 degrees at a time.
        my_servo.angle = angle
        time.sleep(0.05)
    for angle in range(180, 0, -5): # 180 - 0 degrees, 5 degrees at a time.
        my_servo.angle = angle
        time.sleep(0.05)
```

Continuous Servo

Here is an example for a continuous servo:

```
# SPDX-FileCopyrightText: 2019 Anne Barela for Adafruit Industries
#
# SPDX-License-Identifier: MIT

"""CircuitPython Essentials Servo continuous rotation servo example"""
import time
import board
import pwmio
from adafruit_motor import servo

# create a PWMOut object on Pin A2.
pwm = pwmio.PWMOut(board.A2, frequency=50)

# Create a servo object, my_servo.
my_servo = servo.ContinuousServo(pwm)

while True:
    print("forward")
    my_servo.throttle = 1.0
    time.sleep(2.0)
    print("stop")
    my_servo.throttle = 0.0
    time.sleep(2.0)
    print("reverse")
    my_servo.throttle = -1.0
    time.sleep(2.0)
    print("stop")
    my_servo.throttle = 0.0
    time.sleep(4.0)
```

Check out all of the examples in the Adafruit CircuitPython Motor module () too!

Low Level Servo Control

The low level control is rarely recommended in comparison to the `adafruit_motor` library shown previously. It is shown here as it may take less memory than the higher level code and may help you understand the underlying code.

`pulseio` is built into CircuitPython and therefore you do not need to install any libraries.

Python Library Install

You'll need to install the `Adafruit_Blinka` library that provides the CircuitPython support in Python. Since each platform is a little different, and Linux changes often, please visit the CircuitPython on Linux guide to get your computer ready ()!

Once that's done, from your command line run the following command to ensure you're running the latest version of Adafruit Blinka:

- `pip3 install --upgrade adafruit-blinka`

If your default Python is version 3 you may need to run 'pip' instead. Just make sure you aren't trying to use CircuitPython on Python 2.x, it isn't supported!

Low Level Servo Setup

First you need to import a few modules to access the PWM output capabilities of CircuitPython:

```
import board
import pulseio
```

Now you can create a PWM signal output that will drive the position of the servo:

```
servo = pulseio.PWMOut(board.D5, frequency=50)
```

There are a couple important things happening with the line above. This is an initializer which is creating an instance of the `PWMOut` class and part of that initialization process is specifying these two values:

- The pin that will be the PWM output. In this case it's pin D5 on the development board. If you're using a different output be sure to specify the right pin name here (and make sure the pin supports PWM output as mentioned on the previous page!).
- The frequency of the PWM signal. This is an optional value that we're specifying as a keyword argument to tell the PWM output that we want a signal with a frequency of 50 hertz. The frequency is how many times per second the servo signal changes and for most servos they expect a ~50 hz signal.

Position Control with PWM

Now we can control the position of the servo by changing the duty cycle of the PWM signal. If you aren't familiar with PWM signals and duty cycle be sure you've read the CircuitPython analog I/O guide PWM page () for more background.

With a servo motor it will change its position depending on the pulse length of the signal being sent to it. Specifically a 1 millisecond high pulse tells the servo to move all the way to one extreme (left or right) and a value of 2 milliseconds will move all the way to the opposite extreme. Any value in-between 1 to 2 milliseconds will move the servo to an appropriate in-between position. For example a value of 1.5 milliseconds will move the servo to its center or middle position.

Note for continuous rotation servos, they don't have a concept of moving to a specific position. Instead they will rotate freely in a circle and change their speed depending on the pulse length sent to them. A 1 millisecond pulse moves as quickly as possible in one direction, 1.5 millisecond pulse stops movement, and 2 millisecond pulse moves as quickly as possible in the opposite direction.

How do we control the amount of time a PWM signal is high vs. low? As mentioned in the CircuitPython analog I/O PWM page (), we do so by changing the duty cycle of the PWM signal. However, before you set the duty cycle, you'll need to convert from a desired millisecond value to a PWM duty cycle value. Remember with CircuitPython, duty cycle values are specified with a 16-bit unsigned value, i.e. something from 0 to 65535. We can actually create a little Python function to help with doing this mathematical conversion:

```
def servo_duty_cycle(pulse_ms, frequency=50):
    period_ms = 1.0 / frequency * 1000.0
    duty_cycle = int(pulse_ms / (period_ms / 65535.0))
    return duty_cycle
```

You can pass this function a pulse width in milliseconds and it will convert it into the appropriate duty cycle value to control the servo. The function assumes a frequency of 50 hz by default too, but you can change it by specifying the frequency keyword (however you don't typically need or want to change this unless you also changed the frequency of the PWM output!).

For example, see what output value you get with pulse widths of 1.0 and 2.0 milliseconds:

```
servo_duty_cycle(1.0)
servo_duty_cycle(2.0)
```

```
>>> def servo_duty_cycle(pulse_ms, frequency=50):
...     period_ms = 1.0 / frequency * 1000.0
...     duty_cycle = int(pulse_ms / (period_ms / 65535.0))
...     return duty_cycle
...
>>> servo_duty_cycle(1.0)
3276
>>> servo_duty_cycle(2.0)
6553
```

Now let's move the servo! Try changing the servo PWM output duty cycle to a value that corresponds to a 1.0 millisecond long pulse. You should see the servo move to one extreme:

```
servo.duty_cycle = servo_duty_cycle(1.0)
```

If you don't see your servo move, be sure the power supply is turned on and connected to the servo. Double check all of your wiring connections and that the output for the signal supports PWM signals too!

Now move to the opposite extreme with a 2.0 millisecond long pulse:

```
servo.duty_cycle = servo_duty_cycle(2.0)
```

You should see the servo move about 180 degrees to the opposite position.

Finally move to the center position with a 1.5 millisecond pulse:

```
servo.duty_cycle = servo_duty_cycle(1.5)
```

Experiment with setting any value in-between 1.0 and 2.0 to see how the servo reacts.

Sweep Example

Here's a complete example that will sweep the servo between both extremes (1.0 and 2.0 millisecond long pulses) repeatedly. ave this as a code.py file on your board (and be ready when you click save as the servo might instantly start moving!):

```
# SPDX-FileCopyrightText: 2020 Kattni Rembor for Adafruit Industries
#
# SPDX-License-Identifier: MIT

import time
import board
import pwmio

# Initialize PWM output for the servo (on pin D5):
servo = pwmio.PWMOut(board.D5, frequency=50)


# Create a function to simplify setting PWM duty cycle for the servo:
def servo_duty_cycle(pulse_ms, frequency=50):
    period_ms = 1.0 / frequency * 1000.0
    duty_cycle = int(pulse_ms / (period_ms / 65535.0))
    return duty_cycle


# Main loop will run forever moving between 1.0 and 2.0 mS long pulses:
```

```
while True:
    servo.duty_cycle = servo_duty_cycle(1.0)
    time.sleep(1.0)
    servo.duty_cycle = servo_duty_cycle(2.0)
    time.sleep(1.0)
```

That's all there is to controlling a servo directly with CircuitPython with a PWM output.

Arduino

You can also control a servo motor from Arduino in a similar way as CircuitPython with Arduino's Servo library (). There are actually quite a few resources and guides for using Arduino to control servos, so this page will just highlight how to use a servo in Arduino in the same way as with CircuitPython from the previous page. For more exploration of servos and Arduino be sure to check out the entire Arduino Lesson 14: Servo Motors guide ().

First make sure your servo is wired to your board exactly as shown on the hardware page of this guide. You'll also need the Arduino IDE installed and configured to upload to your board. Remember Arduino sketches have to be written entirely up front and uploaded to the board--you can't interactively control servos like you can with CircuitPython.

To control the duty cycle of the servo signal with Arduino we'll use the Servo library instead of directly controlling the PWM output signal. This is necessary because Arduino doesn't support as much control over PWM output as CircuitPython. In particular you can't easily change the frequency of a PWM output to the 50hz value required by servos. Luckily you can use a library to do this frequency control and PWM signal generation for you internally.

Upload the following sketch to your board and it will move the servo between its two extreme positions (a 1.0 and 2.0 millisecond pulse length):

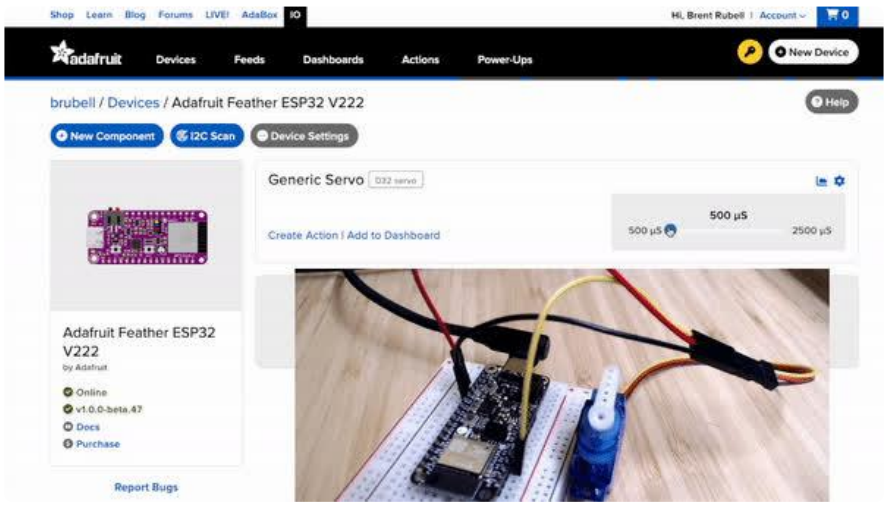
```
#include &lt;Servo.h&gt;

// Create a servo instance.
Servo servo;

void setup() {
  // Attach servo output to pin 5.
  servo.attach(5);
}

void loop() {
  // Move to position 0, or a 1.0 millisecond long pulse:
  // Remember the servo module in Arduino takes in a position in degrees
  // from 0 to 180 instead of a pulse length in milliseconds or other value.
  servo.write(0);
  // Delay for a second.
```

```
  delay(1000);
  // Move to position 180, or a 2.0 millisecond long pulse and pause again.
  servo.write(180);
  delay(1000);
}
```

Notice the servo library controls the servo using a slightly different method of specifying the position in degrees instead of the raw PWM pulse length in milliseconds. Internally the servo library is just converting that position value to a PWM pulse length like you saw with the CircuitPython page of this guide. You can specify a position of 0 for one extreme (1.0 millisecond long pulse), 90 for the center or middle position (1.5 millisecond long pulse), and 180 for the opposite extreme (2.0 millisecond long pulse).

That's all there is to controlling a servo with Arduino!

WipperSnapper



In this page, we'll go through the process of wiring and setting up a board with a Servo using Adafruit IO Wippersnapper.

Before you get started it will help to read these guide pages for more background information on servo motors too:

- Adafruit Motor Selection Guide - Servo Motors ()
- Arduino Lesson 14: Servo Motors ()

To control the servo from WipperSnapper we'll use its built in PWM, or pulse-width modulation, signal generation capabilities.

Do continuous rotation servos work in WipperSnapper?

Yes - they will work the same as a regular servo. However, we do not provide a calibration system so you will need to determine the "center" point for your servo.

Make sure to wire your board like on the Hardware page (). For this page, we're using the Feather ESP32 V2 as an example and wiring the servo to pin D14.

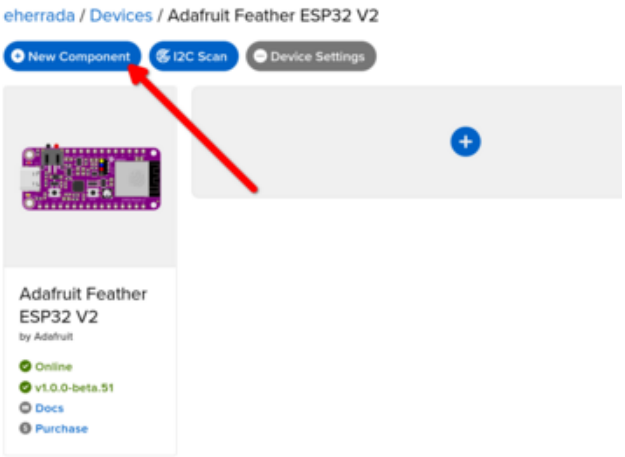
WipperSnapper Setup

If you haven't already added your board to wippersnapper (i.e. if it doesn't show up here ()) click the green button below to see how to do that.

Add your board to WipperSnapper by following this guide

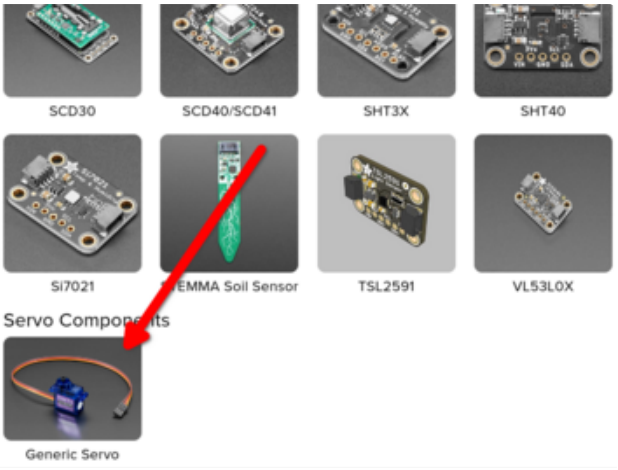
Now that you've already wired everything up, add your board to WipperSnapper, and verify that the board is connected, it's time to set up the servo on Adafruit IO.

Log into Adafruit IO

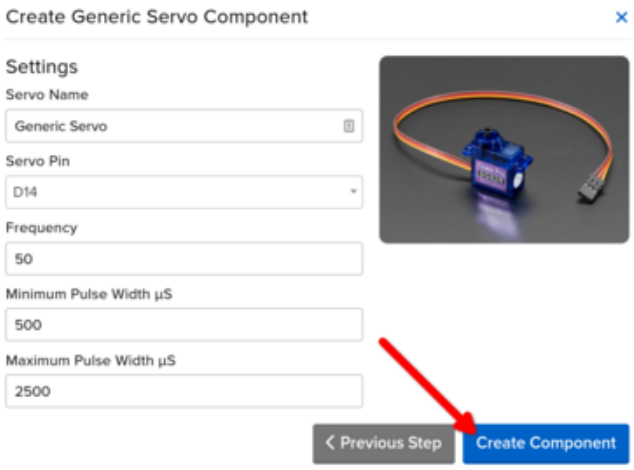


First, make sure your device says it is online next to the board's name. If the board is offline but was previously connected, try moving it closer to your router.

Next, from the device page, click + New Component.



Click Generic Servo



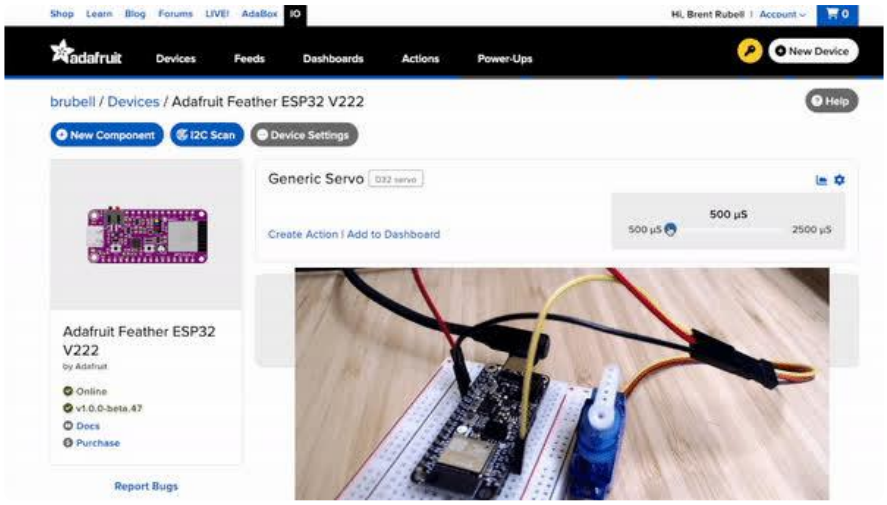
Set Servo Pin to D14, and keep the frequency at 50Hz.

The default minimum and maximum pulse widths for the "generic" servo are 500uS and 2500uS. Depending on the model of the servo, these defaults may not give your servo a full 180 degrees of motion. In that case, check your servo's datasheet to find the pulse widths it is designed to work with. Be aware that if you go too far you could break your servo!

Click Create Component.

Now, try moving the slider and you should see the servo move to the selected position as soon as you release the slider.

For this servo, the minimum position is at 500µs (0.5ms) which is all the way to the left. The middle position is 1500µs (1.5ms), and the maximum position, all the way to the right, is 2500µs (2.5ms).



Note that in this GIF the pin used is D22. In this guide we are using pin D14.