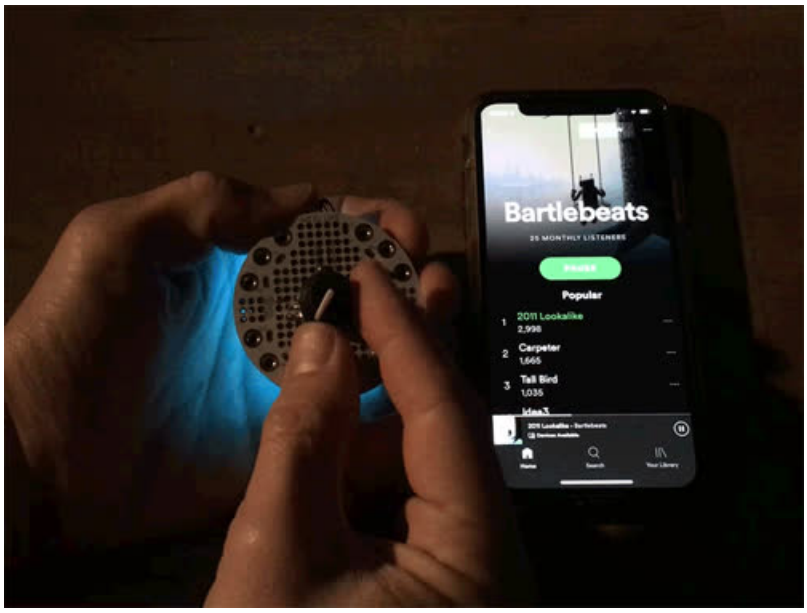
adafruit learning system

BLE Volume Knob with CircuitPython

Created by John Park



Last updated on 2021-02-08 06:46:23 PM EST

Guide Contents

Overview

You can build your own devices that act like remote keyboards and HID devices for nearly any mobile device or computer with Bluetooth LE and the Adafruit HID library in CircuitPython!

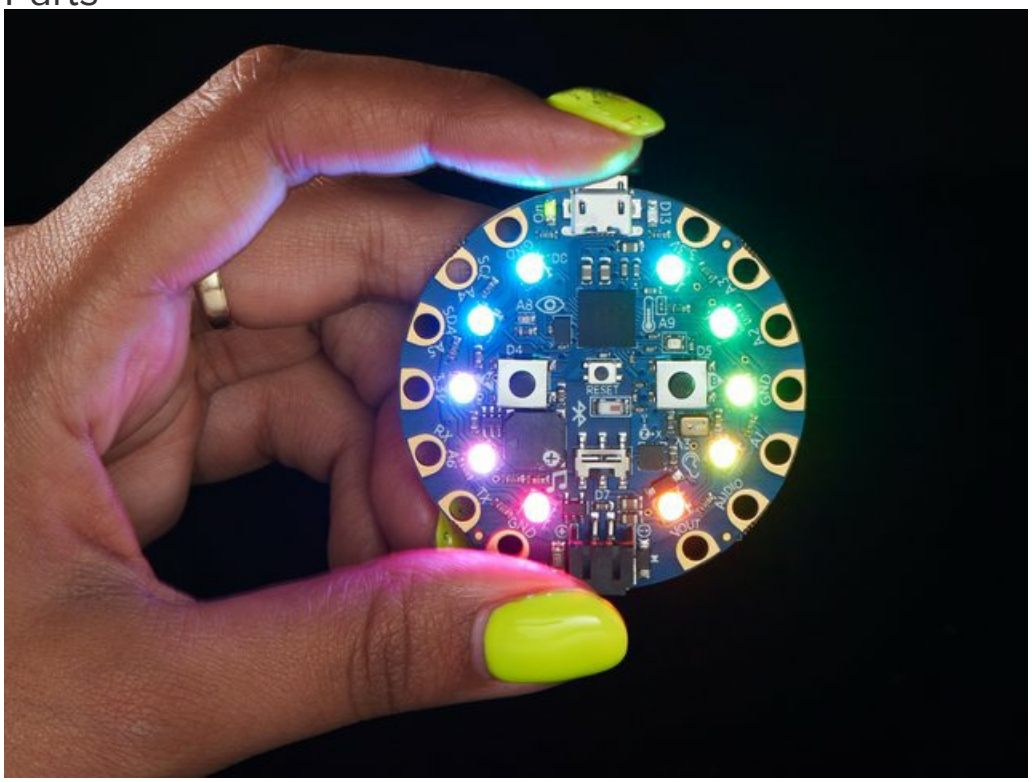
Traditionally, the USB HID library has been used to send keyboard and mouse commands over a USB cable to a computer or mobile device. Now, you can cut that wire and do all the same things using BLE wirelessly!

This tutorial will show one of the many exciting projects you can build with these techniques.

You'll build a wireless volume knob that can turn your sound up and down, mute, and press the play/pause button from across the room to control your mobile device or computer sound!

Plus, we've added Bluetooth LE bonding support, so once you pair your devices, they'll automatically reconnect whenever you turn on the devices.

Parts



Circuit Playground Bluefruit - Bluetooth Low Energy

Circuit Playground Bluefruit is our third board in the Circuit Playground series, another step towards a perfect introduction to electronics and programming. We've...

$24.95

In Stock

Add to Cart

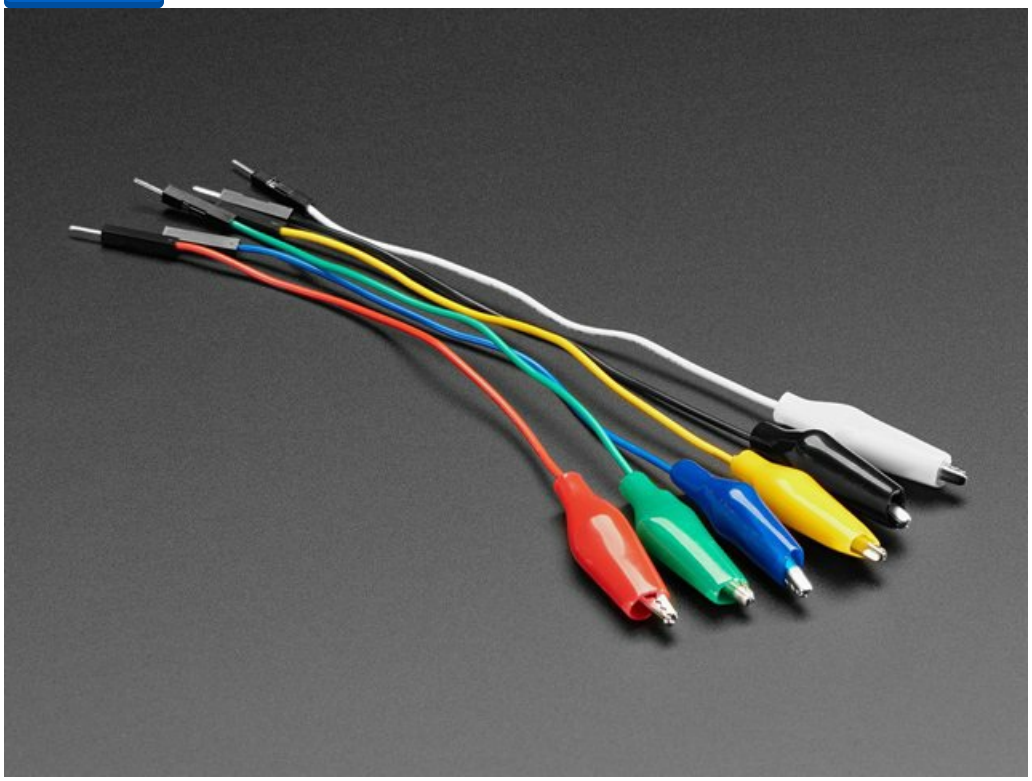
Rotary Encoder + Extras

This rotary encoder is the best of the best, its a high quality 24-pulse encoder, with detents and a nice feel. It is panel mountable for placement in a box, or you can plug it into a...

$4.50

In Stock

Add to Cart



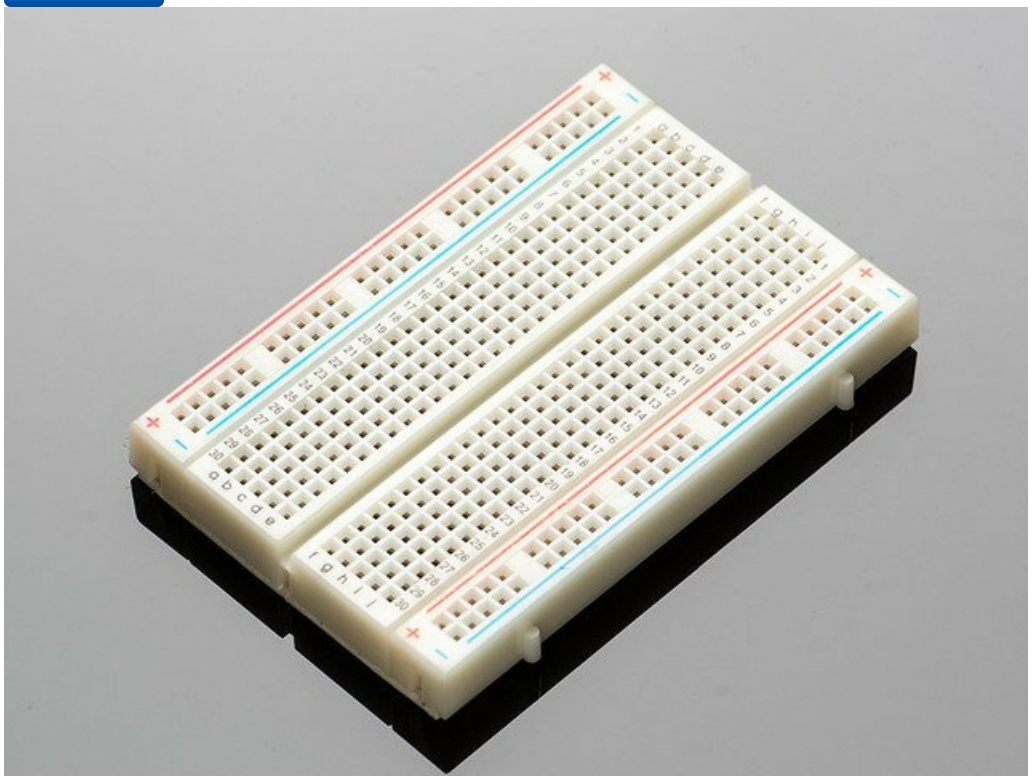
Small Alligator Clip to Male Jumper Wire Bundle - 6 Pieces

When working with unusual non-header-friendly surfaces, these handy cables will be your best friends! No longer will you have long, cumbersome strands of alligator clips. These...

$3.95

In Stock

Add to Cart

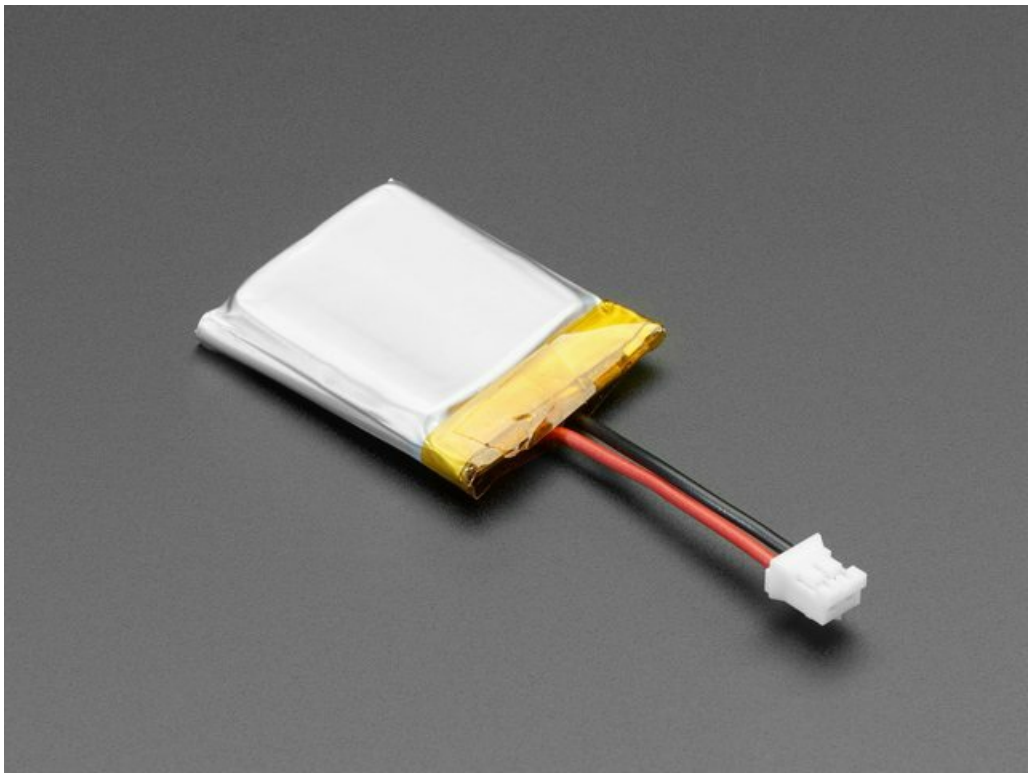


Half-size breadboard

This is a cute half size breadboard, good for small projects. It's 2.2" x 3.4" (5.5 cm x 8.5 cm) with a standard double-strip in the middle and two power rails on both...

Out of Stock

Out of Stock



Lithium Ion Polymer Battery with Short Cable - 3.7V 350mAh

Lithium ion polymer (also known as 'lipo' or 'lipoly') batteries are thin, light and powerful. The output ranges from 4.2V when completely charged to 3.7V. This battery...

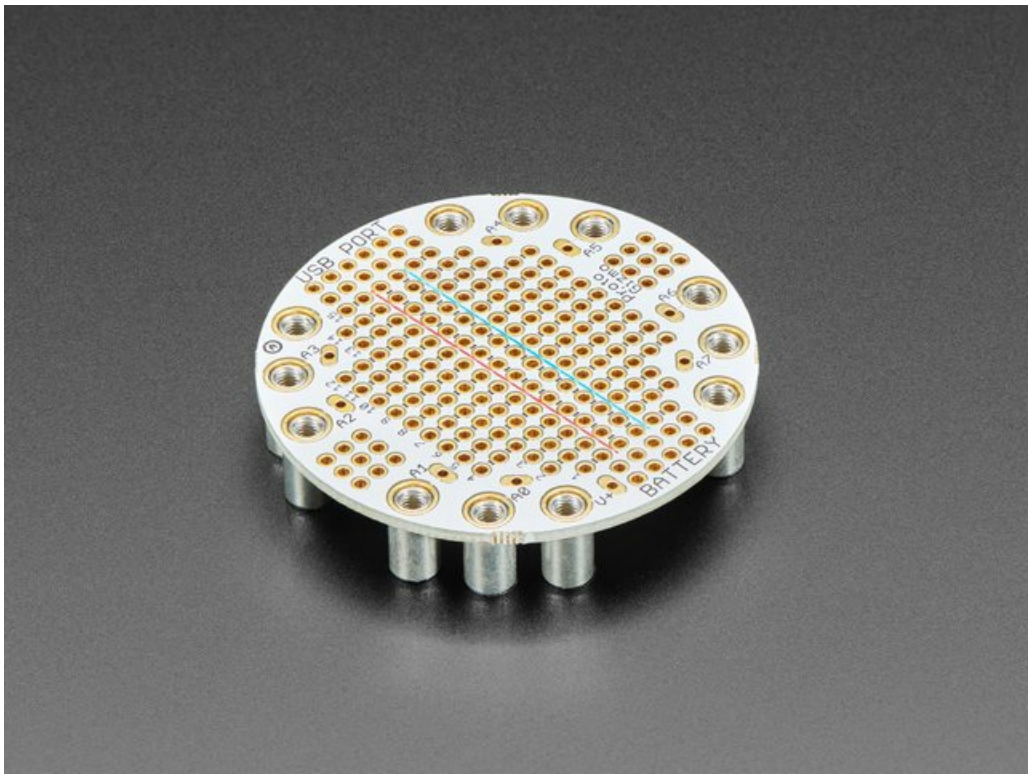
$5.95

In Stock

Add to Cart

Optional Parts

If you want to build a more permanent volume knob, you can use a Proto Gizmo and solder down the encoder.



Circuit Playground Proto Gizmo - Bolt-on Perma-Proto

Extend and expand your Circuit Playground projects with a bolt on Perma-Proto that lets you connect solderable circuits in a sturdy and reliable fashion. This PCB looks just like a...

$7.95

In Stock

Add to Cart

Hook-up Wire Spool Set - 22AWG Solid Core - 6 x 25 ft

Perfect for bread-boarding, free wiring, etc. This box contains 6 spools of solid-core wire. The wire is easy to solder to and when bent it keeps its shape pretty well. We like to have...

$15.95

In Stock

Add to Cart

Fully Reversible Pink/Purple USB A to micro B Cable - 1m long

This cable is not only super-fashionable, with a woven pink and purple Blinka-like pattern, it's also fully reversible! That's right, you will save seconds a day by...

$3.95

In Stock

Add to Cart

Understanding BLE

BLE Basics

To understand how we communicate between the MagicLight Bulb and the Circuit Playground Bluefruit (CPB), it's first important to get an overview of how Bluetooth Low Energy (BLE) works in general.

The nRF52840 chip on the CPB uses Bluetooth Low Energy, or BLE. BLE is a wireless communication protocol used by many devices, including mobile devices. You can communicate between your CPB and peripherals such as the Magic Light, mobile devices, and even other CPB boards!

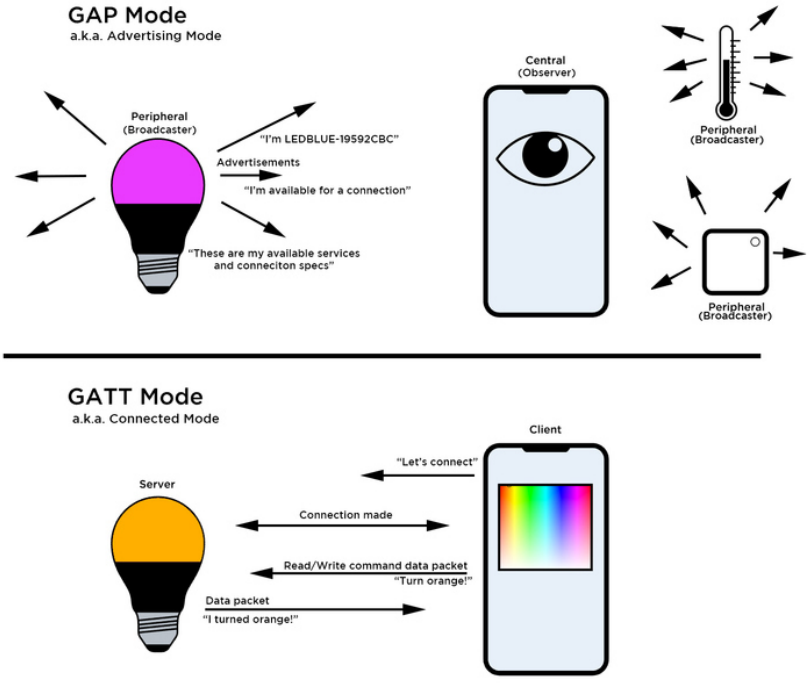
There are a few terms and concepts commonly used in BLE with which you may want to familiarize yourself. This will help you understand what your code is doing when you're using CircuitPython and BLE.

Two major concepts to know about are the two modes of BLE devices:

- Broadcasting mode (also called **GAP** for **G**eneric **A**ccess **P**rofile)
- Connected device mode (also called **GATT** for **G**eneric **ATT**ribute Profile).

GAP mode deals with broadcasting peripheral advertisements, such as "*I'm a device named LEDBlue-19592CBC*", as well as advertising information necessary to establish a dedicated device connection if desired. The peripheral may also be advertising available services.

GATT mode deals with communications and attribute transfer between two devices once they are connected, such as between a heart monitor and a phone, or between your CPB and the Magic Light.



Bluetooth LE Terms

GAP Mode

Device Roles:

- **Peripheral** - The low-power device that broadcasts advertisements. Examples of peripherals include: heart rate monitor, smart watch, fitness tracker, iBeacon, and the Magic Light. The CPB can also work as a peripheral.
- **Central** - The host "computer" that observes advertisements being broadcast by the Peripherals. This is often a mobile device such as a phone, tablet, desktop or laptop, but the CPB can also act as a central (which it will in this project).

Terms:

- **Advertising** - Information sent by the peripheral before a dedicated connection has been established. **All** nearby Centrals can observe these advertisements. When a peripheral device advertises, it may be transmitting the name of the device, describing its capabilities, and/or some other piece of data. Central can look for advertising peripherals to connect to, and use that information to determine each peripheral's capabilities (or Services offered, more on that below).

GATT Mode

Device Roles:

- **Server -** In connected mode, a device may take on a new role as a **Server**, providing a Service available to clients. It can now send and receive data packets as requested by the Client device to which it now has a connection.
- **Client -** In connected mode, a device may also take on a new role as **Client** that can send requests to one or more of a Server's available Services to send and receive data packets.

NOTE: A device in GATT mode can take on the role of both Server and Client while connected to another device.

Terms:

- **Profile -** A pre-defined collection of **Services** that a BLE device can provide. For example, the Heart Rate Profile, or the Cycling Sensor (bike computer) Profile. These Profiles are defined by the Bluetooth Special Interest Group (SIG). For devices that don't fit into one of the pre-defined Profiles, the manufacturer creates their own Profile. For example, there is not a "Smart Bulb" profile, so the Magic Light manufacturer has created their own unique one.
- **Service** - A function the Server provides. For example, a heart rate monitor armband may have separate Services for **Device Information**, **Battery Service**, and **Heart Rate** itself. Each Service is comprised of collections of information called **Characteristics**. In the case of the Heart Rate Service, the two Characteristics are **Heart Rate Measurement** and **Body Sensor Location.** The peripheral advertises its services.
- **Characteristic -** A Characteristic is a container for the value, or attribute, of a piece of data along with any associated metadata, such as a human-readable name. A characteristic may be readable, writable, or both. For example, the Heart Rate Measurement Characteristic can be served up to the

Client device and will report the heart rate measurement as a number, as well as the unit string "bpm" for beats-per-minute. The Magic Light Server has a Characteristic for the RGB value of the bulb which can be written to by the Central to change the color. Characteristics each have a Universal Unique Identifier (UUID) which is a 16-bit or 128-bit ID.

- **Packet** - Data transmitted by a device. BLE devices and host computers transmit and receive data in small bursts called packets.

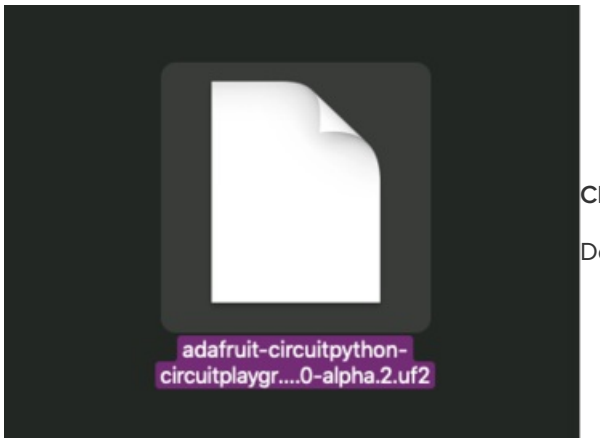
This guide (https://adafru.it/iCS) is another good introduction to the concepts of BLE, including GAP, GATT, Profiles, Services, and Characteristics.

CircuitPython on Circuit Playground Bluefruit
Install or Update CircuitPython

Follow this quick step-by-step to install or update CircuitPython on your Circuit Playground Bluefruit.

https://adafru.it/FNK

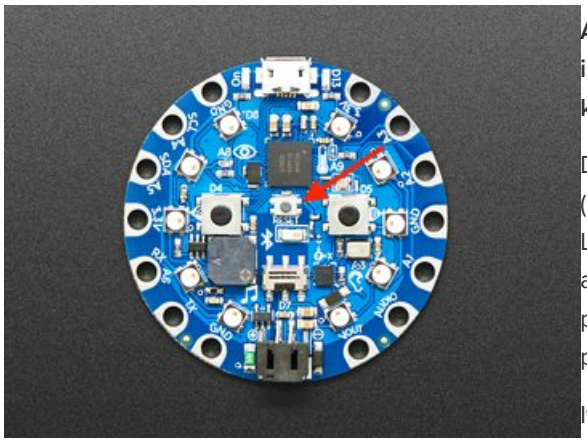
https://adafru.it/FNK

- 

Click the link above and download the latest UF2 file

Download and save it to your Desktop (or wherever is handy)

Plug your Circuit Playground Bluefruit into your computer using a known-good data-capable USB cable.

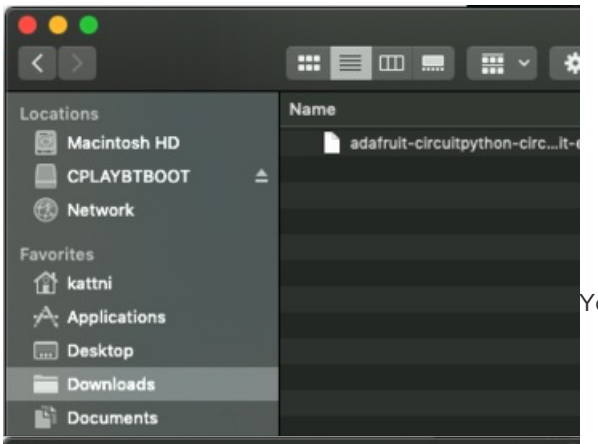
- 

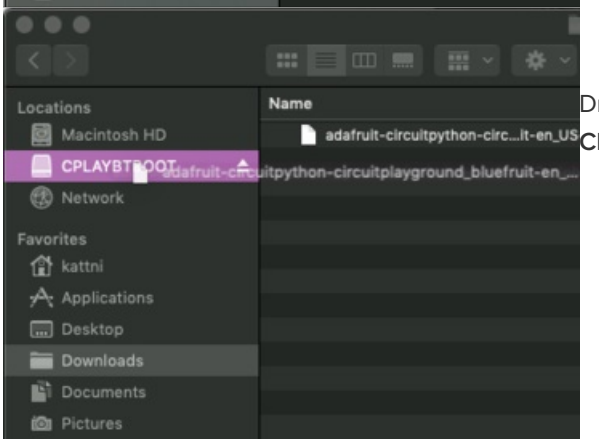
A lot of people end up using charge-only USB cables and it is very frustrating! So make sure you have a USB cable you know is good for data sync.

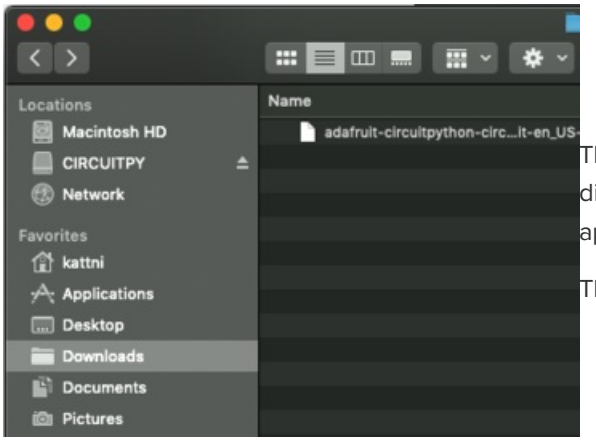
Double-click the small **Reset** button in the middle of the CPB (indicated by the red arrow in the image). The ten NeoPixel LEDs will all turn red, and then will all turn green. If they turn all red and stay red, check the USB cable, try another USB port, etc. The little red LED next to the USB connector will pulse red - this is ok!

If double-clicking doesn't work the first time, try again. Sometimes it can take a few tries to get the rhythm right!

(If double-clicking doesn't do it, try a single-click!)

- 

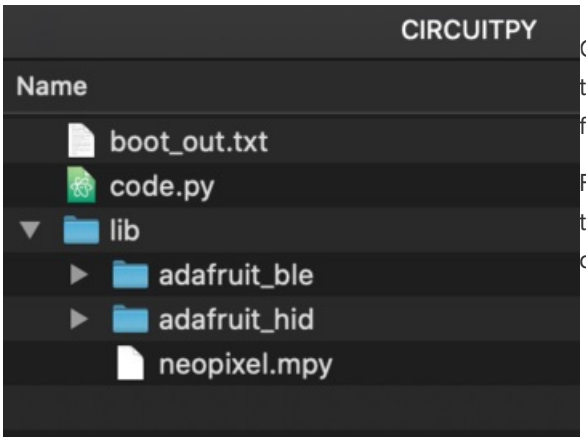
You will see a new disk drive appear called **CPLAYBTBOOT**.
- 

Drag the **adafruit_circuitpython_etc.uf2** file to **CPLAYBTBOOT.**
- 

The LEDs will turn red. Then, the **CPLAYBTBOOT** drive will disappear and a new disk drive called **CIRCUITPY** will appear.

That's it, you're done! :)

Code the BLE Volume Knob

- 

Once your CPB is set up with CircuitPython, you'll also need to add some libraries. Follow this page (https://adafru.it/GdM) for info on how to download and add libraries to your CPB.

From the library bundle you downloaded in that guide page, transfer the following libraries onto the CPB board's **/lib** directory:

- **adafruit_ble**
- **adafruit_hid**
- **neopixel**

Text Editor

Adafruit recommends using the Mu editor for using your CircuitPython code with the Circuit Playground Bluefruit boards. You can get more info in this guide (https://adafru.it/ANO).

Alternatively, you can use any text editor that saves files.

Code.py

Copy the code below and paste it into Mu. Then, save it to your CPB as **code.py**.

```
"""
A CircuitPython 'multimedia' dial demo
Uses a Circuit Playground Bluefruit + Rotary Encoder -> BLE out
Knob controls volume, push encoder for mute, CPB button A for Play/Pause
Once paired, bonding will auto re-connect devices
"""

import time
import digitalio
import board
import rotaryio
import neopixel
from adafruit_hid.consumer_control import ConsumerControl
from adafruit_hid.consumer_control_code import ConsumerControlCode

import adafruit_ble
from adafruit_ble.advertising.standard import ProvideServicesAdvertisement
from adafruit_ble.services.standard.hid import HIDService
from adafruit_ble.services.standard.device_info import DeviceInfoService


ble = adafruit_ble.BLERadio()
ble.name = "Bluefruit-Volume-Control"
# Using default HID Descriptor.
hid = HIDService()
device_info = DeviceInfoService(software_revision=adafruit_ble.__version__,
                                manufacturer="Adafruit Industries")
advertisement = ProvideServicesAdvertisement(hid)
cc = ConsumerControl(hid.devices)

FILL_COLOR = (0, 32, 32)
UNMUTED_COLOR = (0, 128, 128)
MUTED_COLOR = (128, 0, 0)
```

```
DISCONNECTED_COLOR = (40, 40, 0)

# NeoPixel LED ring
# Ring code will auto-adjust if not 16 so change to any value!
ring = neopixel.NeoPixel(board.NEOPIXEL, 10, brightness=0.05, auto_write = False)
ring.fill(DISCONNECTED_COLOR)
ring.show()
dot_location = 0  # what dot is currently lit

# CPB button for Play/Pause
button_A = digitalio.DigitalInOut(board.BUTTON_A)
button_A.switch_to_input(pull=digitalio.Pull.DOWN)

button_a_pressed = False  # for debounce state

# Encoder button is a digital input with pullup on A1
# so button.value == False means pressed.
button = digitalio.DigitalInOut(board.A1)
button.pull = digitalio.Pull.UP

encoder = rotaryio.IncrementalEncoder(board.A2, board.A3)

last_pos = encoder.position
muted = False
command = None
# Disconnect if already connected, so that we pair properly.
if ble.connected:
    for connection in ble.connections:
        connection.disconnect()


def draw():
    if not muted:
        ring.fill(FILL_COLOR)
        ring[dot_location] = UNMUTED_COLOR
    else:
        ring.fill(MUTED_COLOR)
    ring.show()


advertising = False
connection_made = False
print("let's go!")
while True:
    if not ble.connected:
        ring.fill(DISCONNECTED_COLOR)
        ring.show()
        connection_made = False
        if not advertising:
            ble.start_advertising(advertisement)
            advertising = True
        continue
    else:
        if connection_made:
            pass
        else:
            ring.fill(FILL_COLOR)
            ring.show()
            connection_made = True

    advertising = False

    pos = encoder.position
```

```
    delta = pos - last_pos
    last_pos = pos
    direction = 0

    if delta > 0:
        command = ConsumerControlCode.VOLUME_INCREMENT
        direction = -1
    elif delta < 0:
        command = ConsumerControlCode.VOLUME_DECREMENT
        direction = 1

    if direction:
        muted = False
        for _ in range(abs(delta)):
            cc.send(command)
            # spin neopixel LED around in the correct direction!
            dot_location = (dot_location + direction) % len(ring)
            draw()

    if not button.value:
        if not muted:
            print("Muting")
            cc.send(ConsumerControlCode.MUTE)
            muted = True
        else:
            print("Unmuting")
            cc.send(ConsumerControlCode.MUTE)
            muted = False
        draw()
        while not button.value:  # debounce
            time.sleep(0.1)

    if button_A.value and not button_a_pressed:  # button is pushed
        cc.send(ConsumerControlCode.PLAY_PAUSE)
        print("Play/Pause")
        button_a_pressed = True  # state for debouncing
        time.sleep(0.05)

    if not button_A.value and button_a_pressed:
        button_a_pressed = False
        time.sleep(0.05)
```

Code Explainer

Here are the main things our code does:

- Loading libraries for time, digitalio, rotaryio, neopixel, hid consumer control (media buttons), and adafruit BLE
- Advertising that it is a BLE device that can be connected to
- setting up the button A on the CPB and rotary encoder for reading
- lighting up the NeoPixels yellow until there's a connection made
- lighting the NeoPixels cyan, with one brighter NeoPixel to represent relative volume position
- lighting the NeoPixels red when the push encoder is pressed to mute the volume

Pairing and Bonding

One of the more advanced features used in this project is BLE bonding.

When the Central (your mobile device or computer) connects with the Peripheral (the CPB), you will be asked on the mobile device or computer if you want to **Pair** with the CPB. Once you agree to pair, a

bonding process takes place.

During bonding, encrypted keys are exchanged between the two devices and saved away for use the next time the devices attempt to connect. Since they are bonded, the two will connect automatically without asking for a pairing confirmation. This is really convenient, because it means you can walk one of the devices out of range, thus dropping the connection, and when you return, the two devices will re-connect as if nothing ever happened!

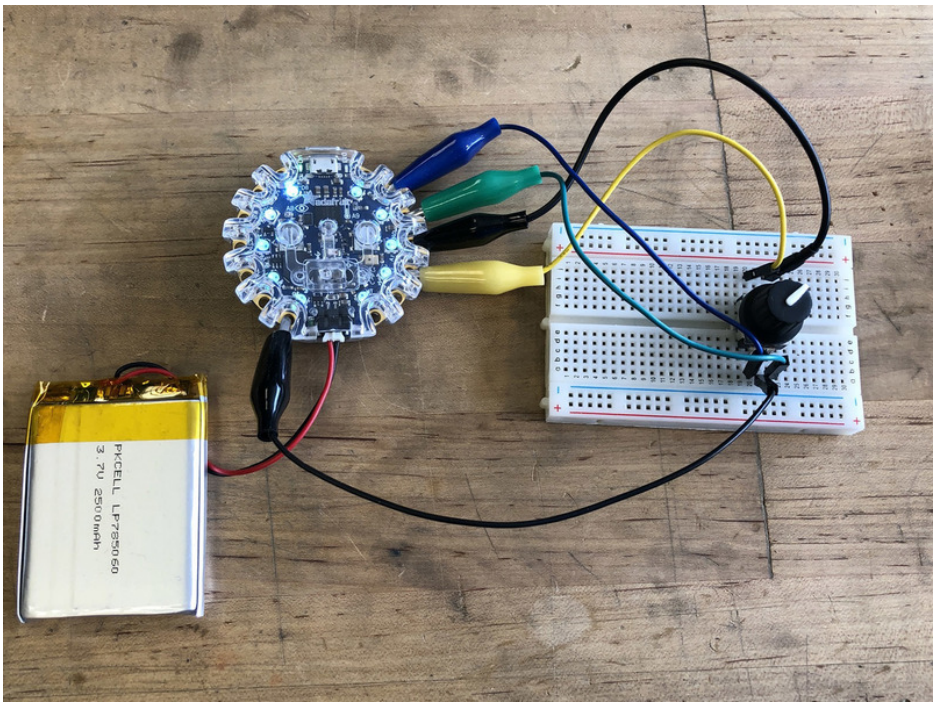
Now, let's build the volume knob!

Beyond the Code

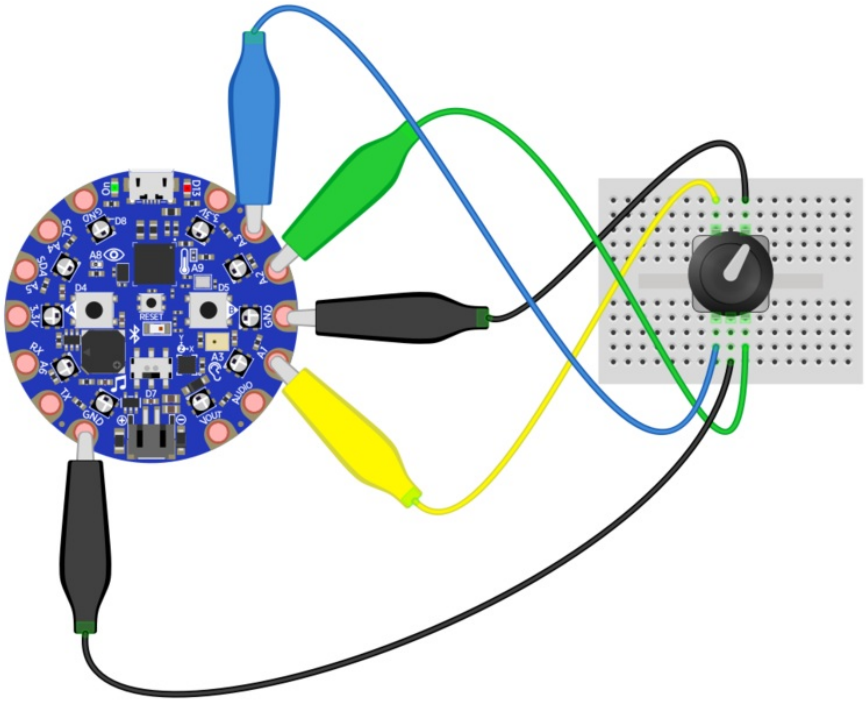
Want to try some variations on the code? A user pointed out:

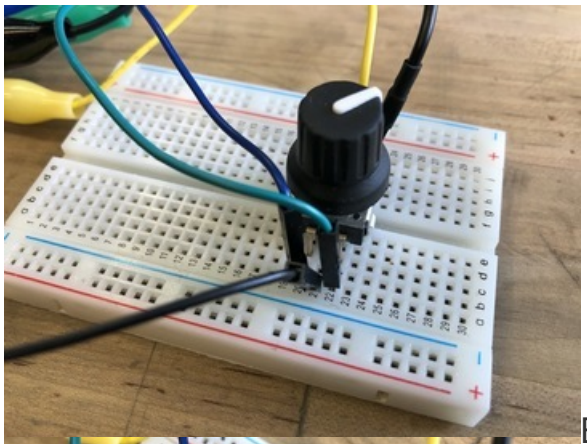
> Maybe this should use the Circuit Playground to simplify accessing the button: https://learn.adafruit.com/circuitpython-made-easy-on-circuit-playground-express/circuit-playground-express-library
>
> Maybe this could use the Debouncing library: https://learn.adafruit.com/debouncer-library-python-circuitpython-buttons-sensors/overview

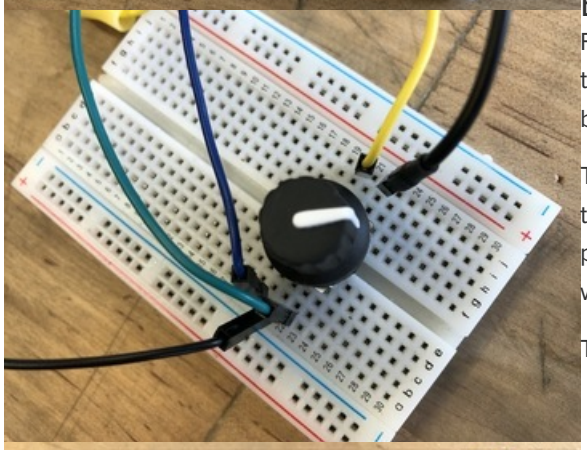
Build and Use the BLE Volume Knob

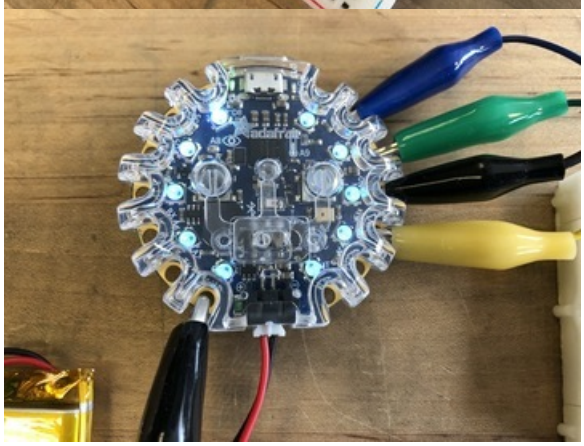


Here we'll build a simple-yet-powerful volume knob using a rotary encoder and Circuit Playground Bluefruit.









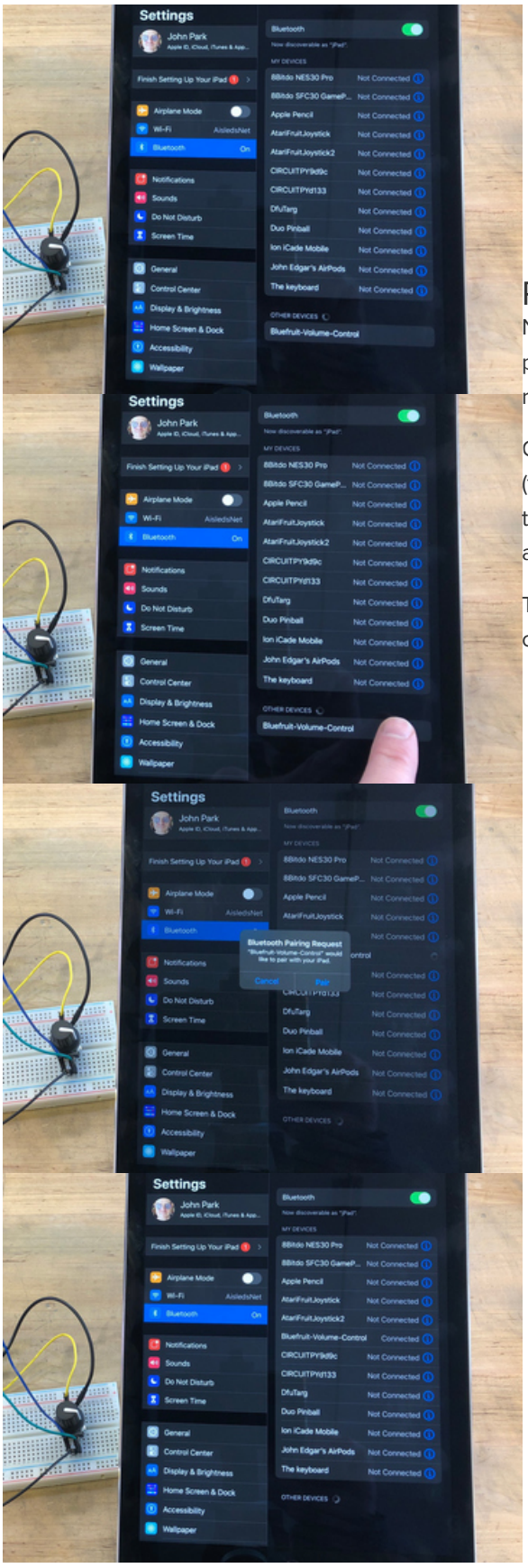
Build on Breadboard

Following the schematic above, plug the rotary encoder into the breadboard, and then connect the alligator clip wires between the encoder pins and the CPB.

The encoder is essentially two components in one -- the three lower pins are for measuring the knob rotational position, while the two upper pins are for the push switch when the knob is pressed down.

This is the wiring connection list:

- Encoder **A** pin to CPB **A3**
- Encoder **GND** to CPB **GND**
- Encoder **B** pin to CPB **A2**
- Switch **ground** to CPB **GND**
- Switch **button** to CPB **A1**

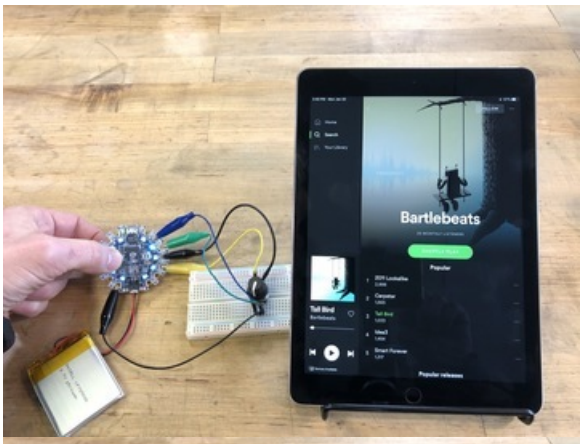


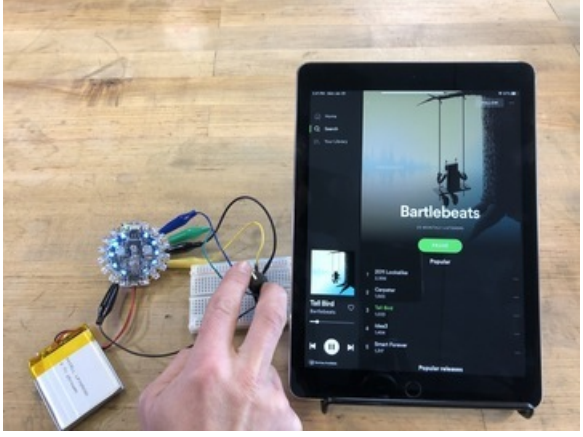
Pair (and Bond)

Now you can power up the device with a battery (or USB power) and it will begin to advertise and turn the NeoPixel ring yellow.

On your mobile device or computer with BLE, pick the device (the first time you connect it may have a nondescript name, or the name Bluefruit Volume Control as seen here on an iPad), and agree to the Pairing prompt.

The devices will be paired, and automatically bonded so they can auto reconnect later.



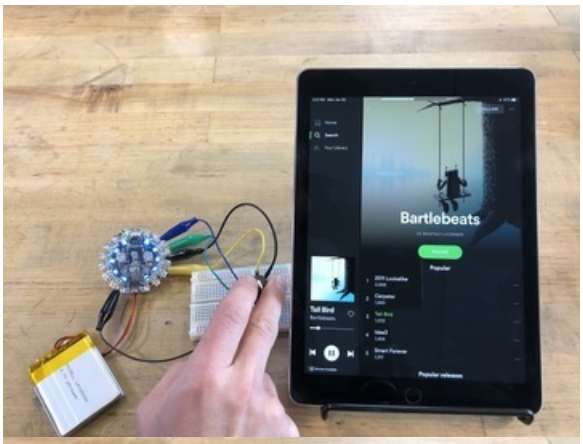


Remote Control

Once connected, the NeoPixel ring will turn cyan. Rotate the knob to raise and lower the volume. Neat huh?! You'll see the bright pixel rotating around along with the knob.

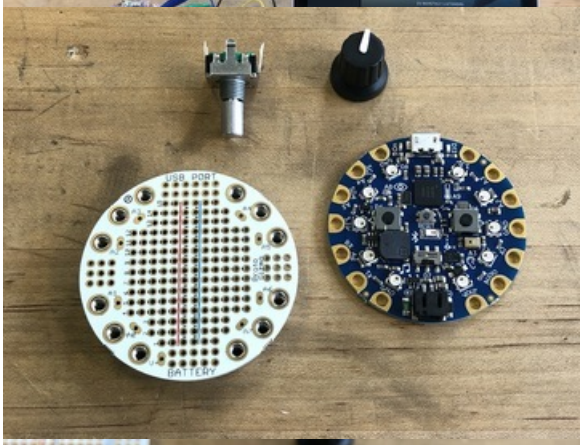
Now, press the push encoder knob down to mute, and the NeoPixels turn red. You can unmute by pressing the button again, or by turning the volume knob.

You can also use the button A on the CPB to toggle Play/Pause.

- 

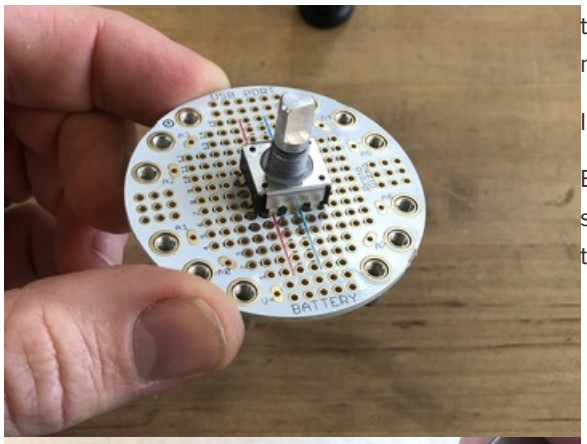
Fancy Version

If you like, you can build a more permanent version using a Perma-Proto Gizmo.

- 

- 

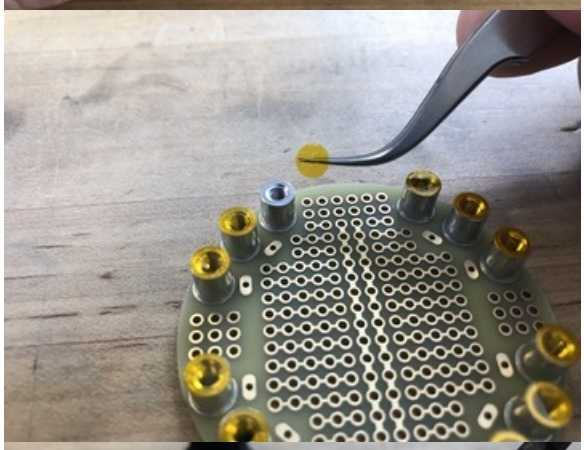
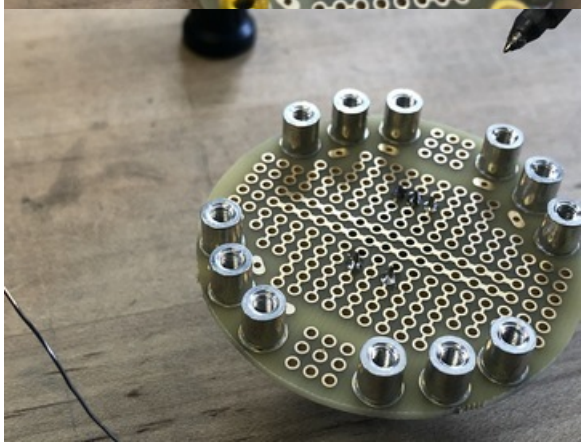
First, remove the side mounting legs from the encoder using some diagonal cutters. We don't have mounting holes for

- 

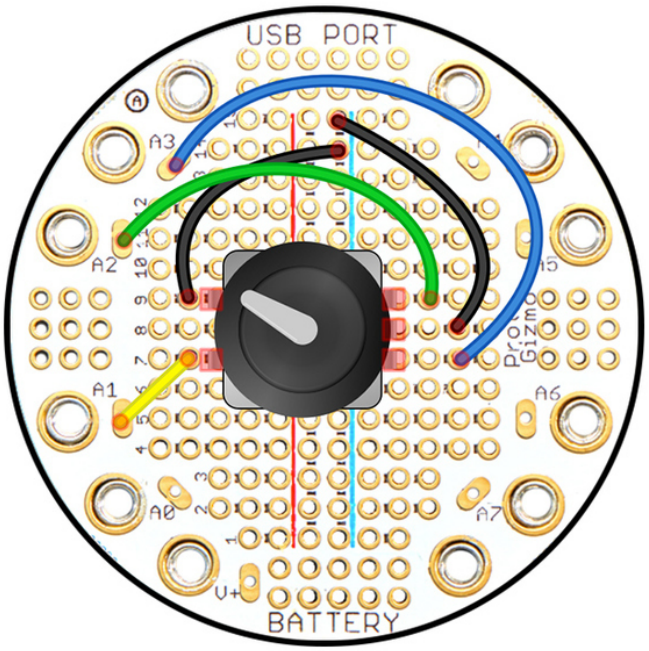
them, and they aren't part of the circuit, they are just for mechanical stability.

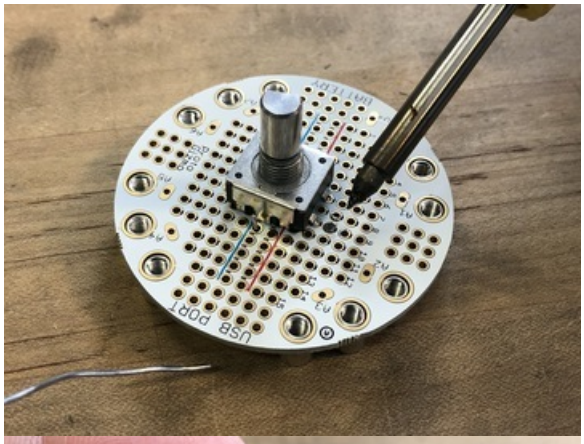
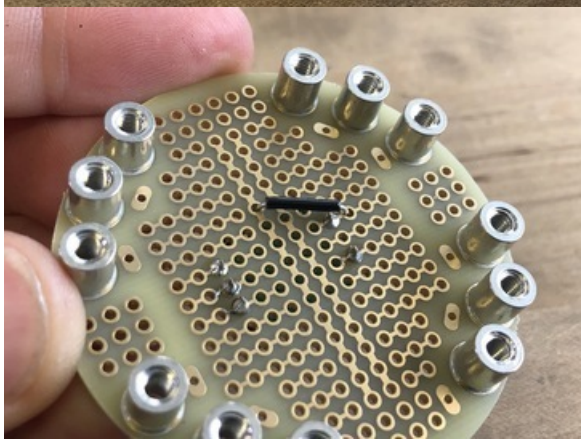
Insert the encoder in the board as shown.

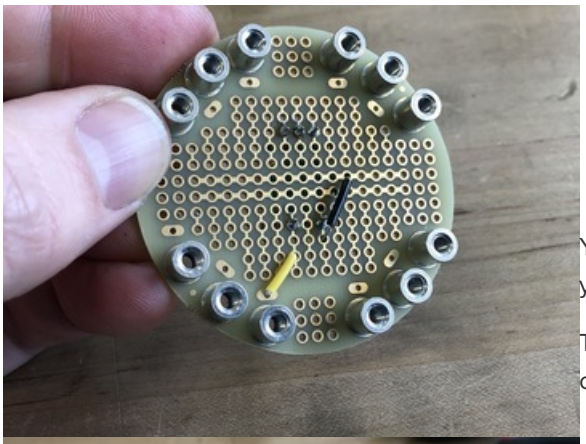
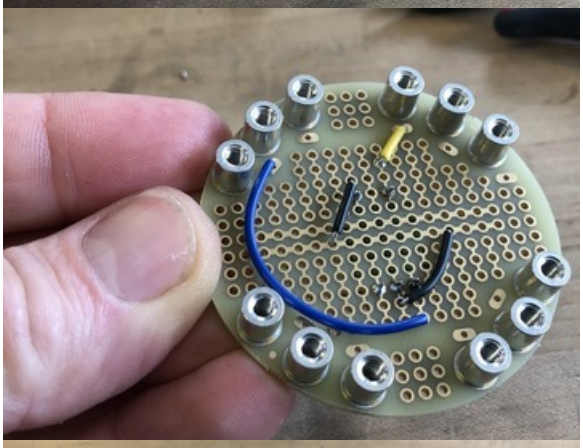
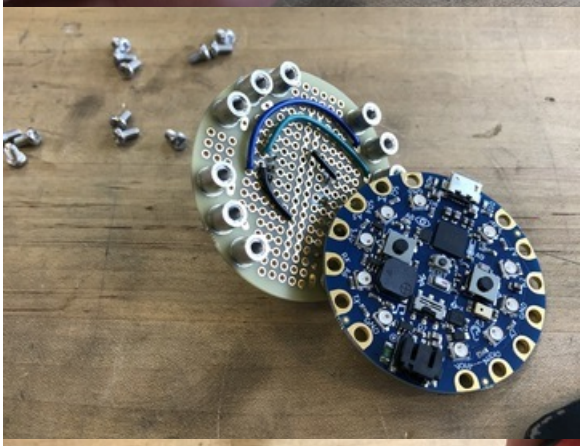
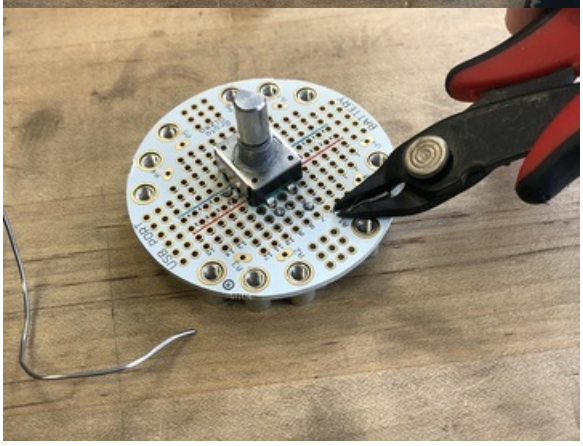
Be sure to remove the twelve amber Kapton stickers from the standoffs, otherwise there won't be an electrical connection to the CPB when we screw them together.

- 
- 

Using short lengths of hookup wire, solder the connections as shown in the schematic drawing below.


USB PORT
A3
A2
A1
A0
A6
A7
V+
BATTERY

- 
- 

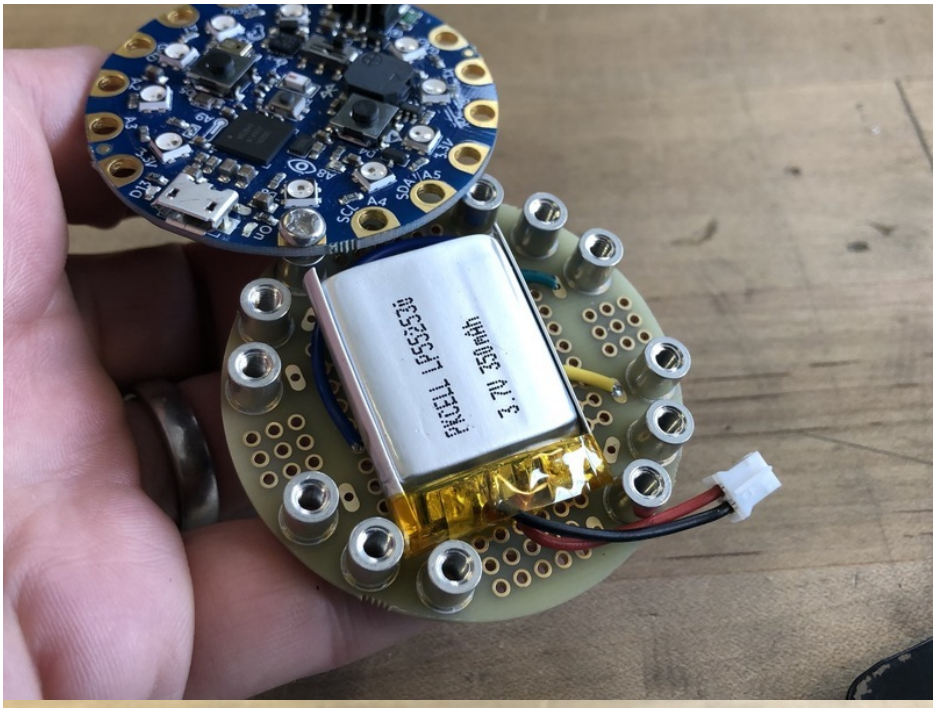
- 
- 
- 
- 

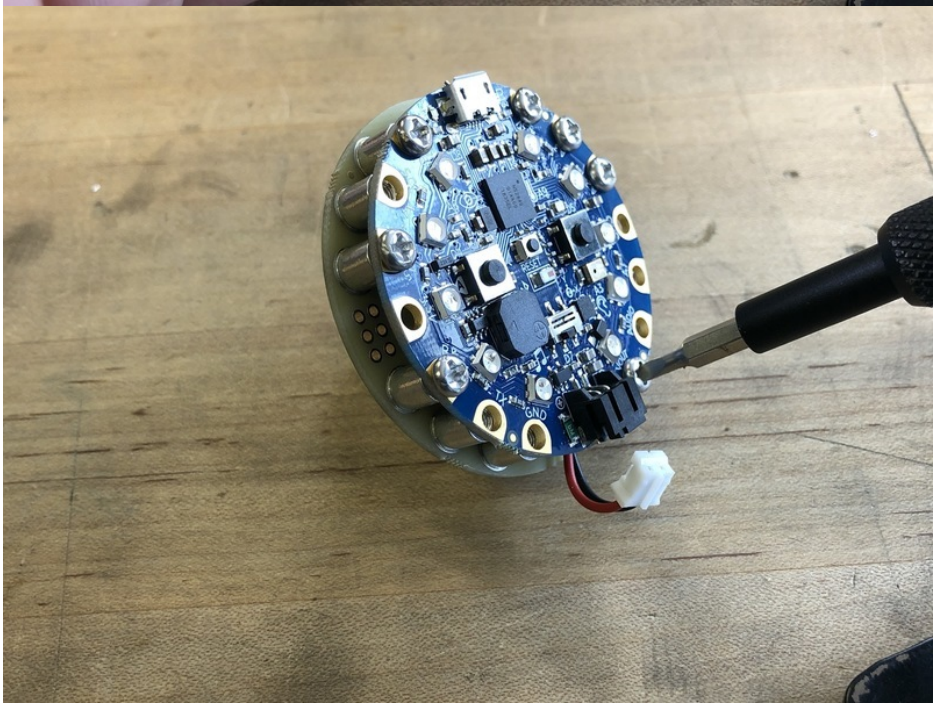
You can run all of the wiring on the underside of the Gizmo if you like,for a neater appearance.

Trim all of the excess pin and wire lengths with diagonal side cutters.

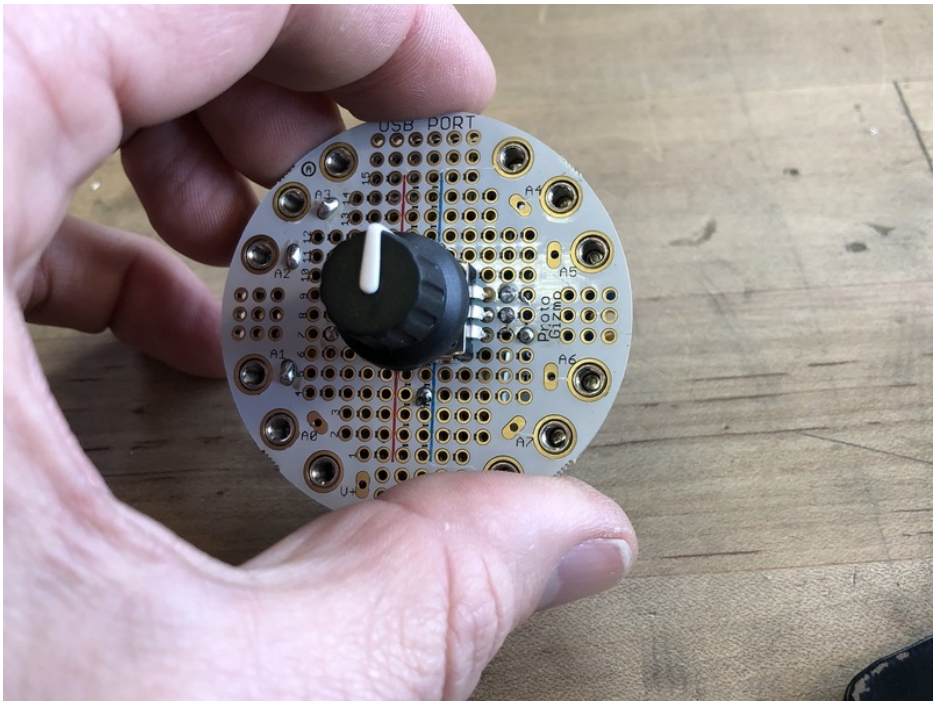
Battery

Place the battery between the two boards, and then screw them together with the provided M3 mounting screws.





Plug in the power, pair the CPB with your mobile device or computer with BLE, and you can use your volume knob in style!


USB PORT
A3
A4
A2
A5
A1
A6
A0
A7
Proto Gizmo