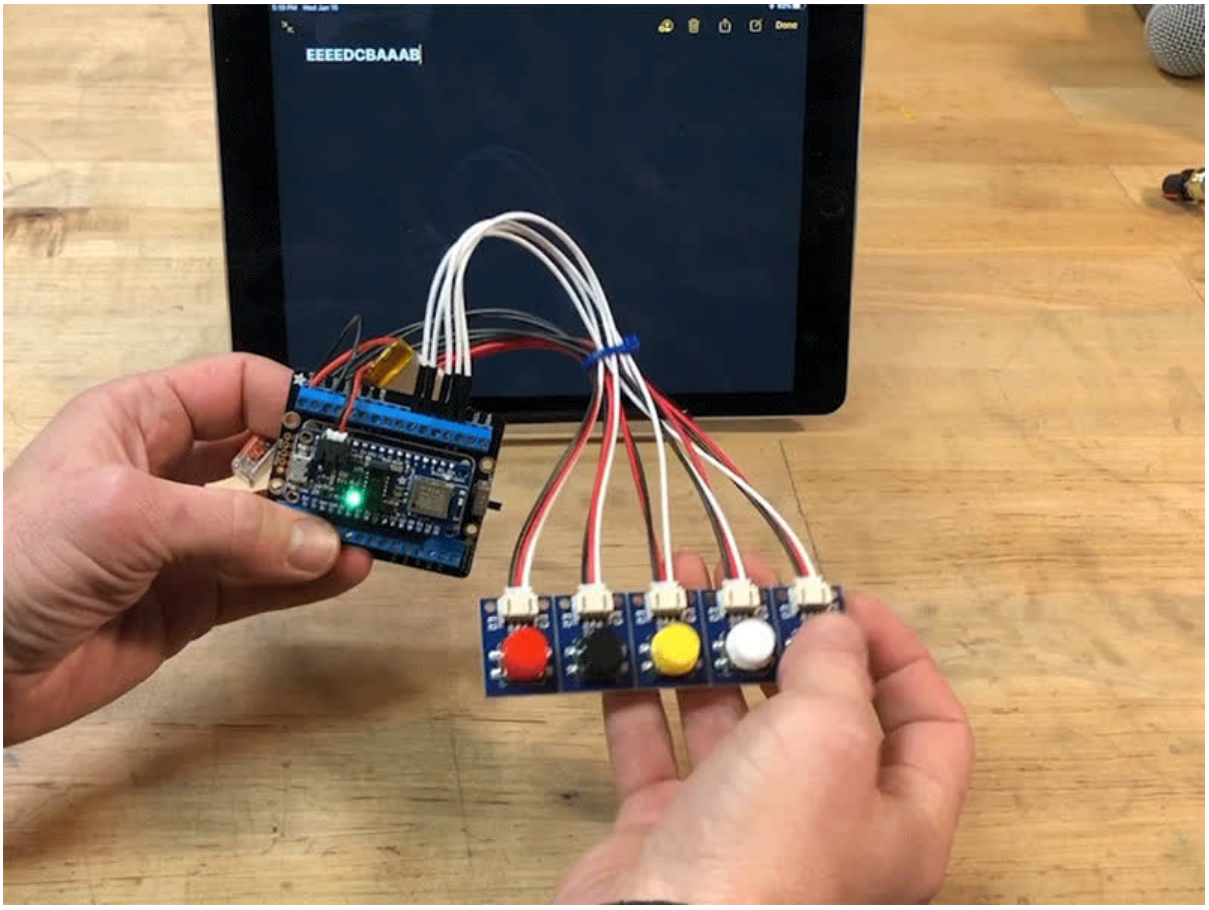
BLE HID Keyboard Buttons with CircuitPython

Created by John Park



https://learn.adafruit.com/ble-hid-keyboard-buttons-with-circuitpython

Last updated on 2025-04-18 10:44:27 AM EDT

Table of Contents

Overview

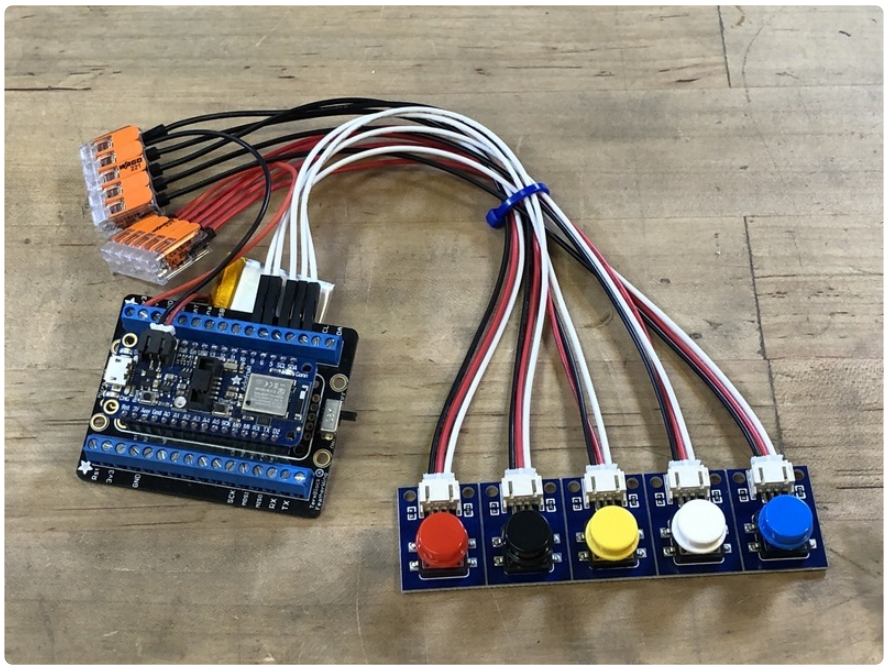
You can build your own devices that act like remote keyboards and HID devices for nearly any mobile device or computer with Bluetooth LE and the Adafruit HID library in CircuitPython!

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 shows just one of the many exciting projects you can build with these techniques.

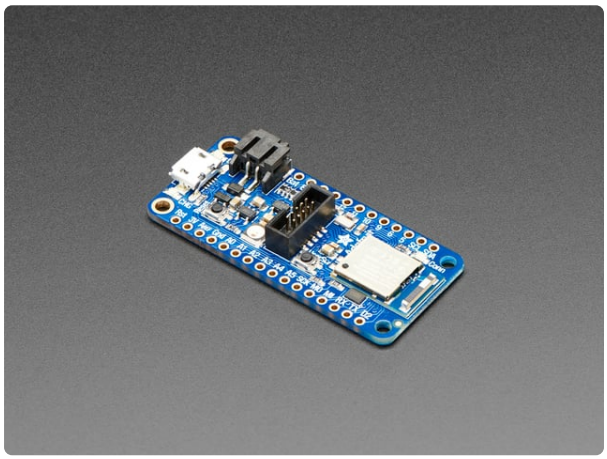
You'll create a remote five-key keyboard for sending keystrokes or even full words and sentences to your device, just as if you were using a highly specialized wireless keyboard.

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

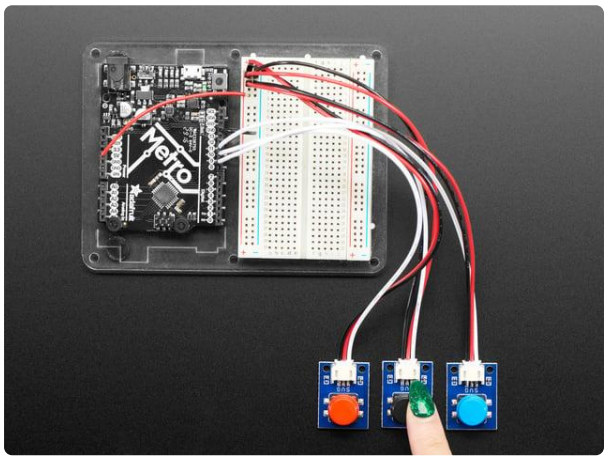


Note: you could also adjust the code here to send HID over USB.

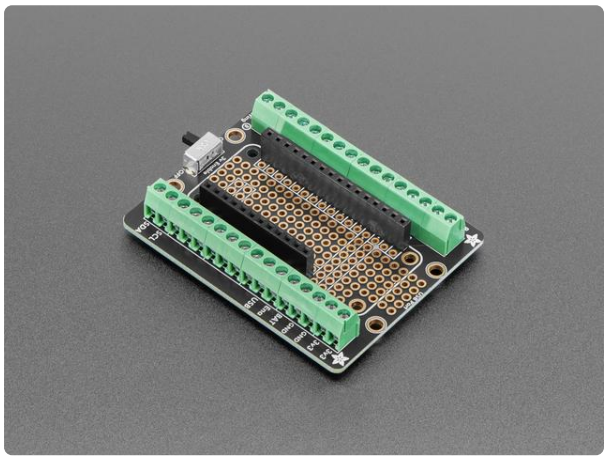
Parts



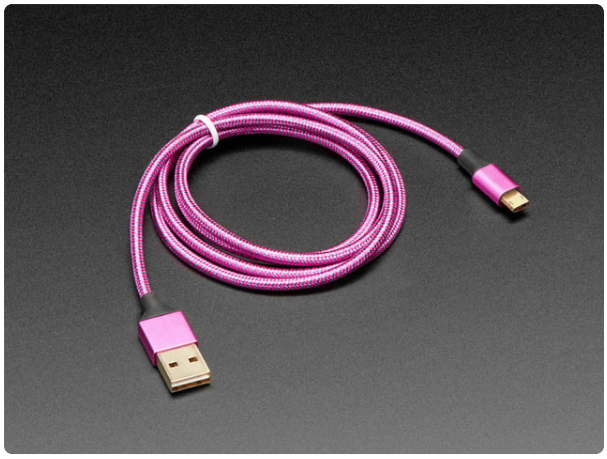
[Adafruit Feather nRF52840 Express](https://www.adafruit.com/product/4062)
The Adafruit Feather nRF52840 Express is the new Feather family member with Bluetooth Low Energy and native USB support featuring the nRF52840! It's...
https://www.adafruit.com/product/4062



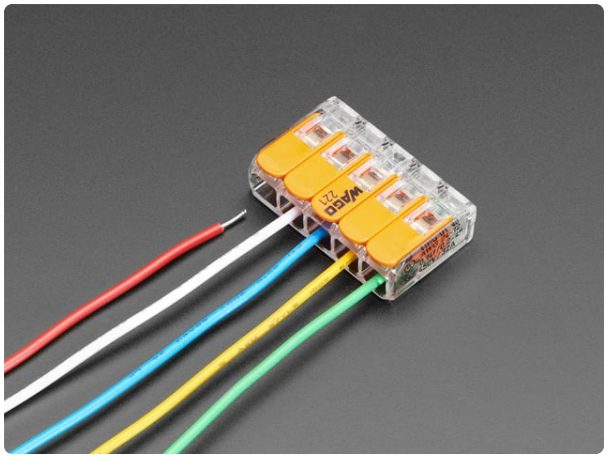
[STEMMA Wired Tactile Push-Button Pack - 5 Color Pack](https://www.adafruit.com/product/4431)
Little clicky switches are standard input "buttons" on electronic projects. These are just like our Colorful Round...
https://www.adafruit.com/product/4431



[Assembled Terminal Block Breakout FeatherWing for all Feathers](https://www.adafruit.com/product/2926)
The Terminal Block Breakout FeatherWing kit is like the Golden Eagle of prototyping FeatherWings (eg. majestic, powerful, good-looking). To start, you get a nice prototyping area...
https://www.adafruit.com/product/2926



[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...
https://www.adafruit.com/product/4111



[Snap-action 5-Wire Block Connector (12-24 AWG) - Pack of 3]
These are like the fancy electronics equivalent of the wire nuts electricians use to bind wires together. They are a lot easier to use as well! Each block connector has a metal block...
https://www.adafruit.com/product/874

[Hook-up Wire Spool Set - 22AWG Stranded-Core - 6 x 25ft]
This is a box of six 25ft spools of stranded-core wire. Stranded-core wire is best used for wiring jigs where there's...
https://www.adafruit.com/product/3111

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!

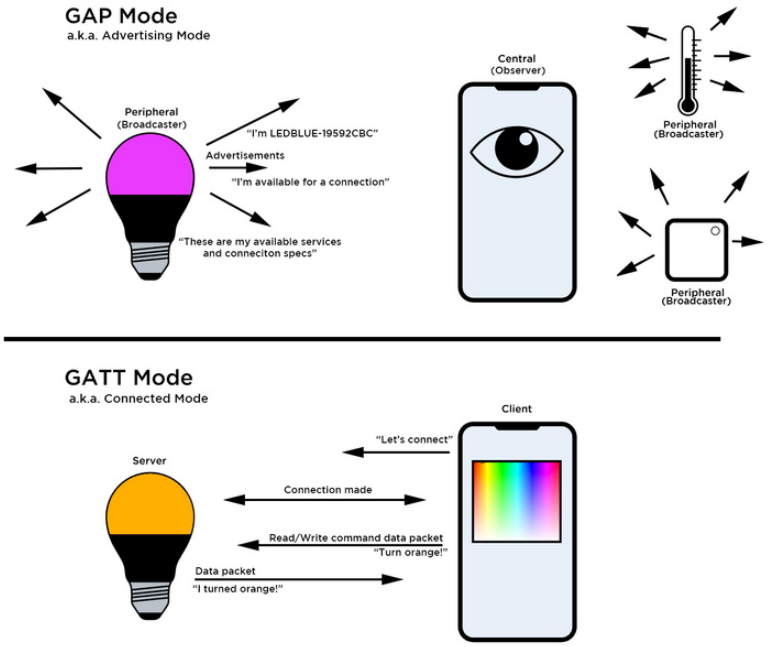
There are a few terms and concepts commonly used in BLE with which you may want to familiarize yourself. This will help you understand what your code is doing when you're using CircuitPython and BLE.

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 for Feather nRF52840

CircuitPython (https://adafru.it/tB7) is a derivative of MicroPython (https://adafru.it/BeZ) designed to simplify experimentation and education on low-cost microcontrollers. It makes it easier than ever to get prototyping by requiring no upfront desktop software downloads. Simply copy and edit files on the **CIRCUITPY** drive to iterate.

The following instructions will show you how to install CircuitPython. If you've already installed CircuitPython but are looking to update it or reinstall it, the same steps work for that as well!

Set up CircuitPython Quick Start!

Follow this quick step-by-step for super-fast Python power :)

Download the latest version of CircuitPython for this board via CircuitPython.org

https://adafru.it/FxJ

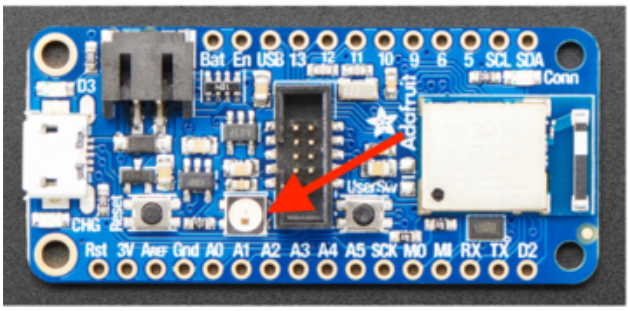
adafruit-circuitpython-feather_nrf52840_exp...en_US-4.0.0-beta.1.uf2

Click the link above to download the latest UF2 file.

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

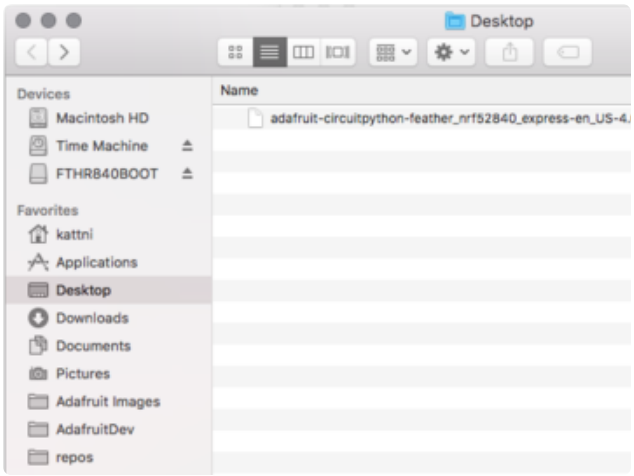
Plug your Feather nRF52840 into your computer using a known-good USB cable.

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.

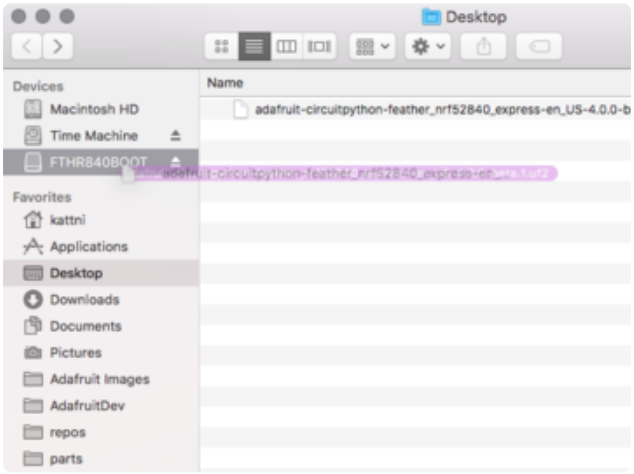


Double-click the **Reset** button next to the USB connector on your board, and you will see the NeoPixel RGB LED turn green (identified by the arrow in the image). If it turns red, check the USB cable, try another USB port, etc. **Note:** The little red LED next to the USB connector will pulse red. That's ok!

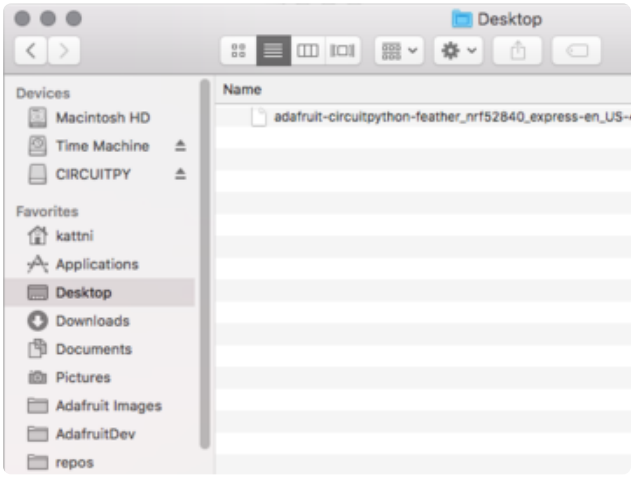
If double-clicking doesn't work the first time, try again. Sometimes it can take a few tries to get the rhythm right!



You will see a new disk drive appear called **FTHR840BOOT**.



Drag the **adafruit_circuitpython_etc.uf2** file to **FTHR840BOOT.**



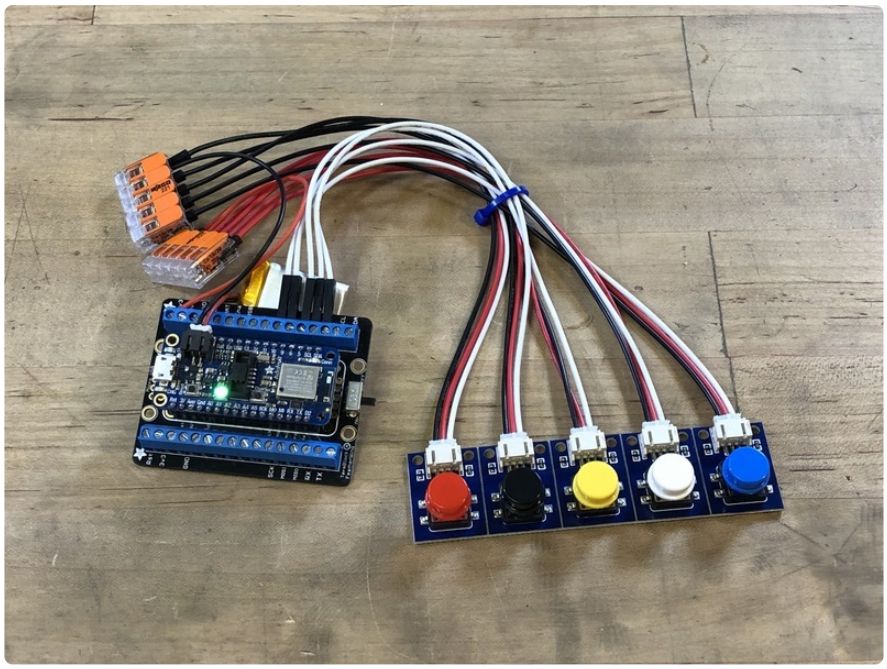
The LED will flash. Then, the **FTHR840BOOT** drive will disappear and a new disk drive called **CIRCUITPY** will appear.

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

BLE Keyboard Buttons

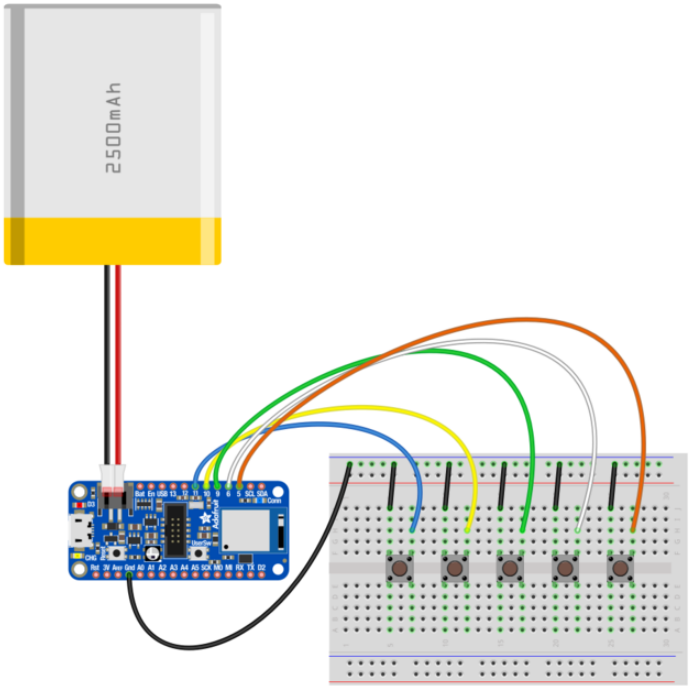
The Adafruit BLE HID library for CircuitPython makes it easy to send keyboard commands and keystrokes wirelessly over Bluetooth LE to mobile devices and BLE equipped computers!

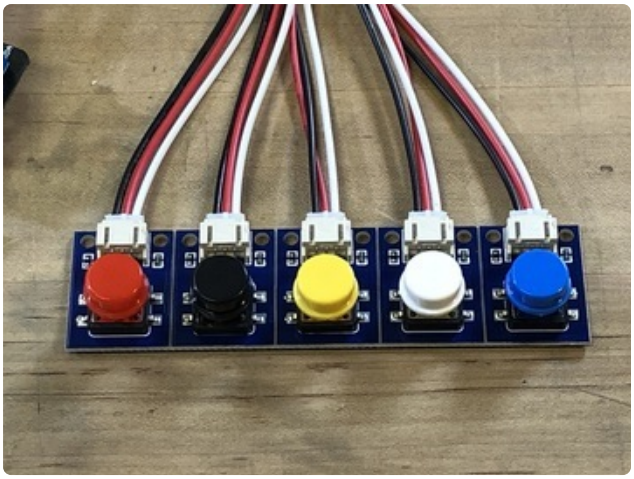
This means you can use buttons or switches to remotely send any keystroke, including key modifiers such as 'shift' and 'control', and even words or sentences.

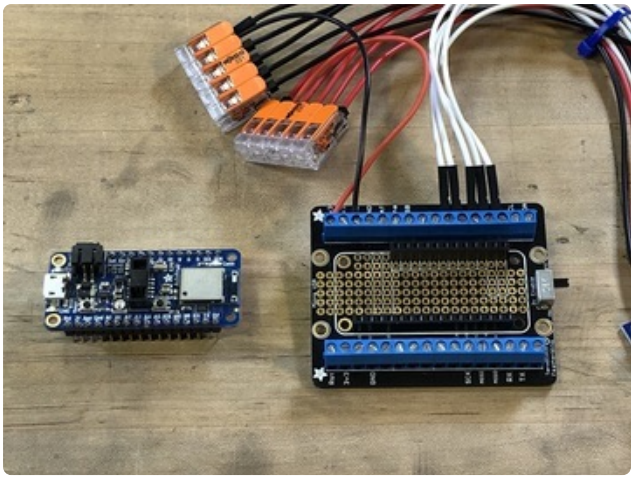


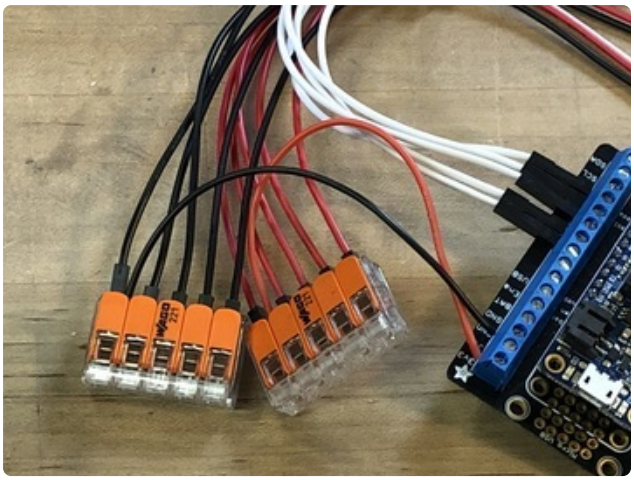
To build the BLE keyswitch, we'll use the Feather nRF52840, five wired push-button breakouts, and a screw terminal block breakout FeatherWing.

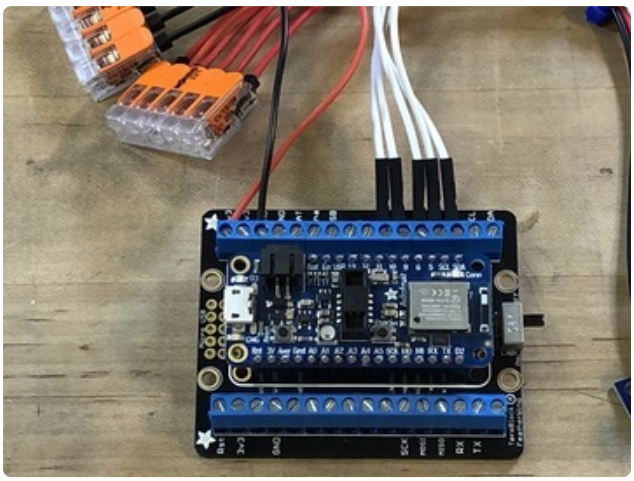
The push-button breakouts (http://adafru.it/4431) used in this project have built-in pull-up resistors. If you do not use these breakouts, you'll need to uncomment statements in the code to turn on internal pull-ups for the buttons.











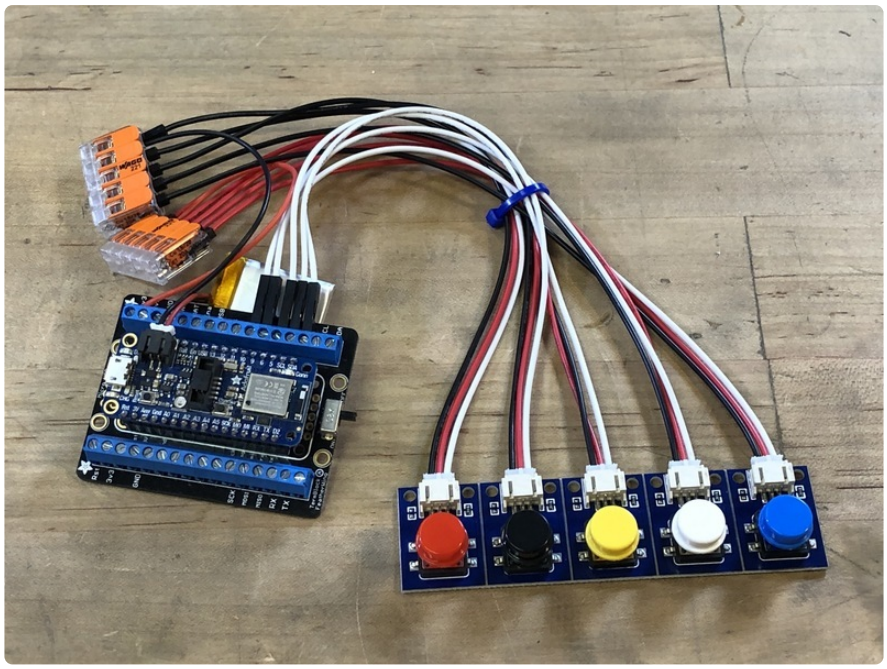
These button break-out boards are very convenient -- of course you can use any momentary switch you like. Also, these are designed to be snapped off into separate buttons, and even have mounting holes for attachment, but I decided to keep them connected for this project.

Plug the STEMMA wire connectors into each board, and then screw the white wires into these pins **5, 6, 9, 10 & 11** on the Feather terminal breakout board (or on a breadboard or directly to the Feather, if that's how you roll).

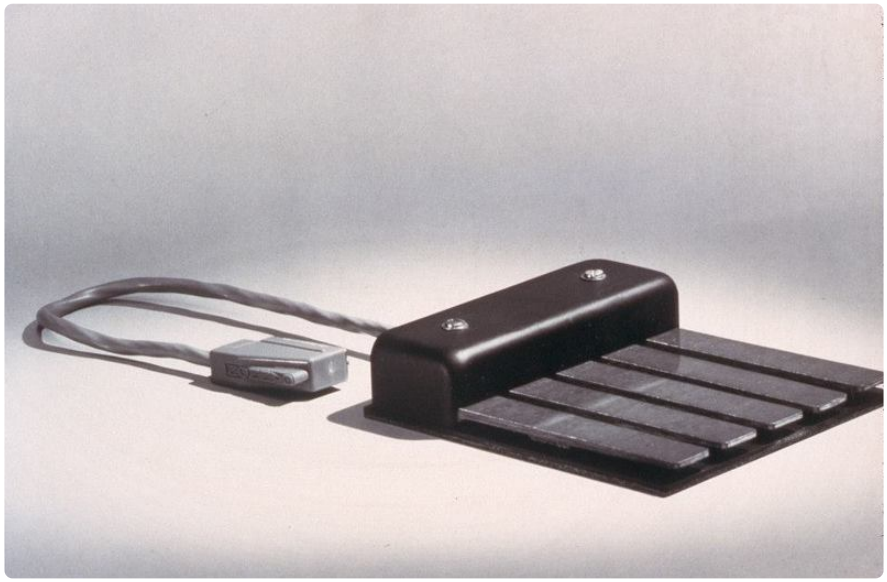
The button breakout boards each have a ground wire that needs to be tied to the Feather ground. I chose to do so with Wago connectors as shown here, with a sixth length of wire running back to the Feather.

I did the same with the red V+ wires, although this is totally optional. This allows you to use the built in pull-up resistors on the board.

Lastly, plug the Feather onto the terminal breakout FeatherWing.

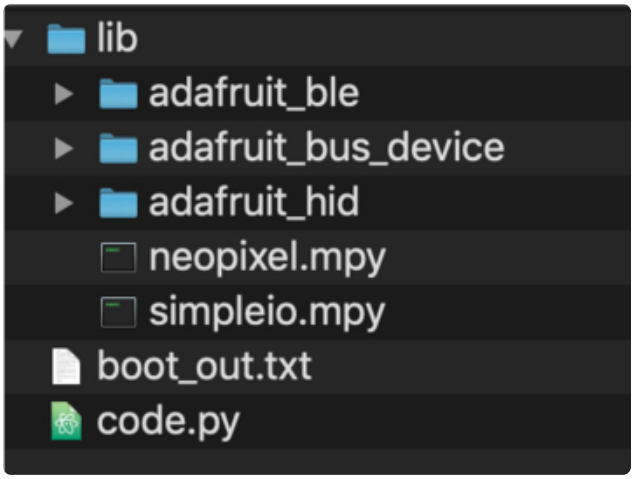


One possible use for such a keypad is to replicate this "keyset" idea [as explored in this article](https://adafru.it/lrc) (https://adafru.it/lrc).



Next, we'll put CircuitPython on the Feather nRF52840, as well as libraries and code.

BLE Keyboard Buttons Libraries and Code



Once your Feather nRF52840 is set up with CircuitPython you'll also need to add some libraries. Follow this page (https://adafru.it/ABU) for info on how to download and add libraries to your Feather.

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

adafruit_ble
adafruit_bus_device
adafruit_hid
neopixel.mpy
simpleio.mpy

Text Editor

Adafruit recommends using the Mu editor for using your CircuitPython code with the Feather 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 Feather as **code.py**.

For a full list of HID keycodes, check out these docs (https://adafru.it/1a14).

> If you are not using the specified push-button breakouts, which have built-in pullups, uncomment the pull-up code in the program below.

```
# SPDX-FileCopyrightText: 2020 John Park for Adafruit Industries
#
# SPDX-License-Identifier: MIT

"""
This example acts as a BLE HID keyboard to peer devices.
Attach five buttons with pullup resistors to Feather nRF52840
  each button will send a configurable keycode to mobile device or computer
"""
```

```
import time
import board
from digitalio import DigitalInOut, Direction
# Uncomment if setting .pull below.
# from digitalio import Pull

import adafruit_ble
from adafruit_ble.advertising import Advertisement
from adafruit_ble.advertising.standard import ProvideServicesAdvertisement
from adafruit_ble.services.standard.hid import HIDService
from adafruit_ble.services.standard.device_info import DeviceInfoService
from adafruit_hid.keyboard import Keyboard
from adafruit_hid.keyboard_layout_us import KeyboardLayoutUS
from adafruit_hid.keycode import Keycode

button_1 = DigitalInOut(board.D11)
button_2 = DigitalInOut(board.D10)
button_3 = DigitalInOut(board.D9)
button_4 = DigitalInOut(board.D6)
button_5 = DigitalInOut(board.D5)

button_1.direction = Direction.INPUT
button_2.direction = Direction.INPUT
button_3.direction = Direction.INPUT
button_4.direction = Direction.INPUT
button_5.direction = Direction.INPUT

# NOTE: If you are not using buttons with built-in pullups, uncomment the five lines
below,
# and the `from digitalio import Pull` above.
# button_1.pull = Pull.UP
# button_2.pull = Pull.UP
# button_3.pull = Pull.UP
# button_4.pull = Pull.UP
# button_5.pull = Pull.UP

hid = HIDService()

device_info = DeviceInfoService(software_revision=adafruit_ble.__version__,
                                manufacturer="Adafruit Industries")
advertisement = ProvideServicesAdvertisement(hid)
# Advertise as "Keyboard" (0x03C1) icon when pairing
# https://www.bluetooth.com/specifications/assigned-numbers/
advertisement.appearance = 961
scan_response = Advertisement()
scan_response.complete_name = "CircuitPython HID"

ble = adafruit_ble.BLERadio()
if not ble.connected:
    print("advertising")
    ble.start_advertising(advertisement, scan_response)
else:
    print("already connected")
    print(ble.connections)

k = Keyboard(hid.devices)
kl = KeyboardLayoutUS(k)
while True:
    while not ble.connected:
        pass
    print("Start typing:")

    while ble.connected:
        if not button_1.value:  # pull up logic means button low when pressed
            #print("back")  # for debug in REPL
            k.send(Keycode.BACKSPACE)
            time.sleep(0.1)

        if not button_2.value:
```

```
            kl.write("Bluefruit")  # use keyboard_layout for words
            time.sleep(0.4)

        if not button_3.value:
            k.send(Keycode.SHIFT, Keycode.L)  # add shift modifier
            time.sleep(0.4)

        if not button_4.value:
            kl.write("e")
            time.sleep(0.4)

        if not button_5.value:
            k.send(Keycode.ENTER)
            time.sleep(0.4)

    ble.start_advertising(advertisement)
```

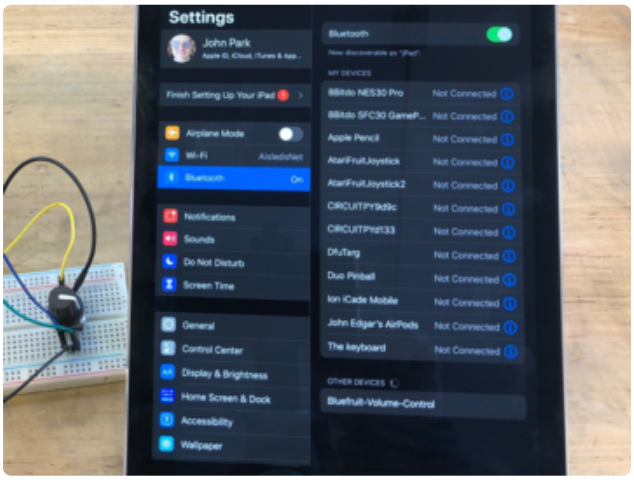
Pairing and Bonding

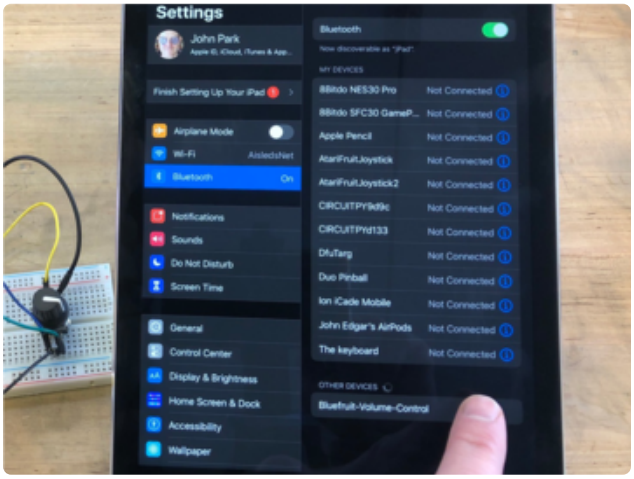
One of the more advance features used in this project is BLE bonding.

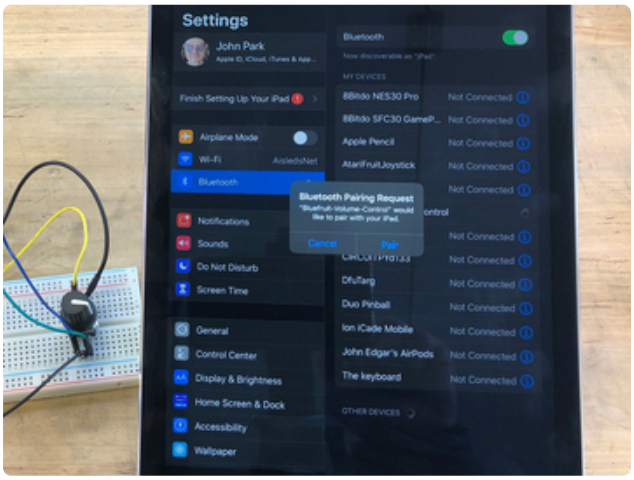
When the Central (your mobile device or computer) connects with the Peripheral (the CPB), you will be asked on the mobile device or computer if you want to **Pair** with the Feather. Once you agree to pair, a **bonding** process takes place.

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!

BLE Key Button Usage







Pair (and Bond)

Now you can power up the keyboard button device with a battery (or USB power) and it will begin to advertise.

On your mobile device or computer with BLE, pick the device (the first time you connect it may have a generic name such as **CIRCUITPYd133**, later it may have the name Bluefruit Keybutton Control when it reconnects), and agree to the Pairing prompt. (Note, these photos are with a different device, but the process is identical.)

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

Once connected, the keys will act just like any wireless keyboard to send input to your mobile device or computer. Try opening up a text editor and then press the five buttons.

In the sample code here's what each button will send:

- **button_1 = BACKSPACE**
- **button_2 = the word "Bluefruit"**
- **button_3 = the letter "L"**
- **button_4 = the letter "e"**
- **button_5 = ENTER**

Customization

You can customize the code so your buttons send other keycodes. [Here's a list of HID keycodes and modifiers (https://adafru.it/Dxd)](https://adafru.it/Dxd) to check out.

To customize the keycodes, look at this section of the code:

```
if not button_1.value:  # pull up logic means button low when pressed
            #print("back")  # for debug in REPL
            k.send(Keycode.BACKSPACE)
            time.sleep(0.1)

        if not button_2.value:
            kl.write("Bluefruit")  # use keyboard_layout for words
            time.sleep(0.4)

        if not button_3.value:
            k.send(Keycode.SHIFT, Keycode.L)  # add shift modifier
            time.sleep(0.4)

        if not button_4.value:
            kl.write("e")
            time.sleep(0.4)

        if not button_5.value:
            k.send(Keycode.ENTER)
            time.sleep(0.4)
```