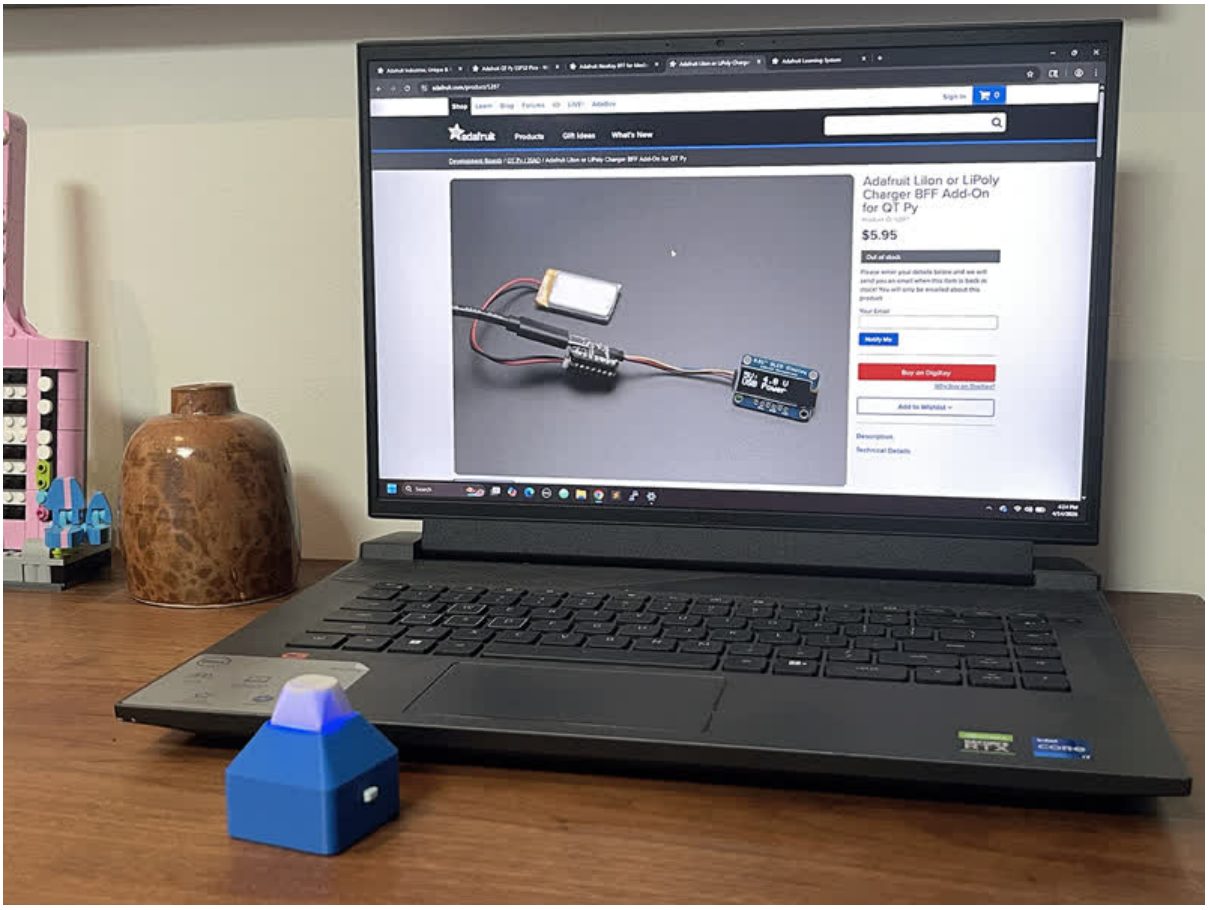
One Key: Single Button Bluetooth Keyboard

Created by John Park

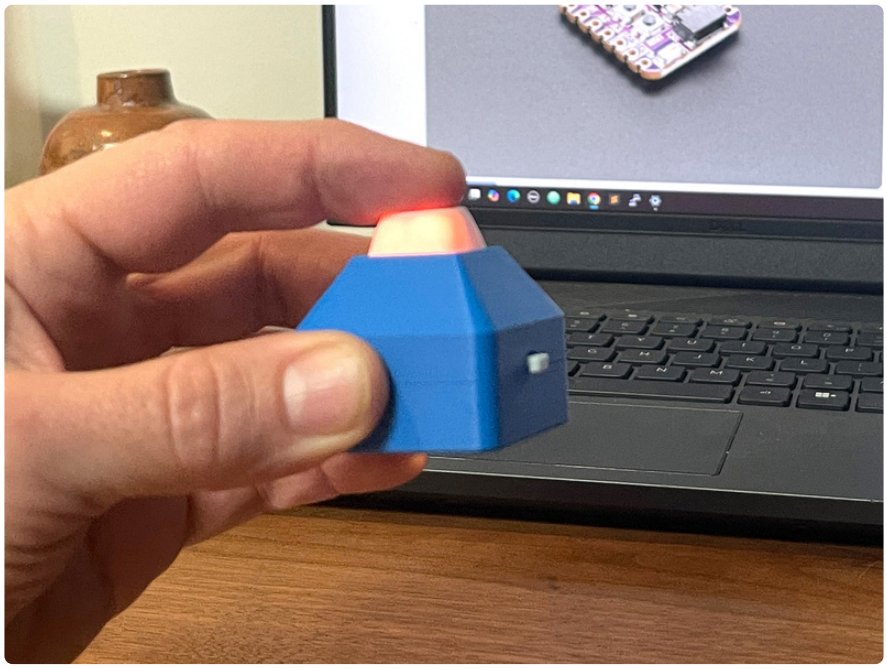


https://learn.adafruit.com/onekey

Last updated on 2026-04-16 04:09:02 PM UTC

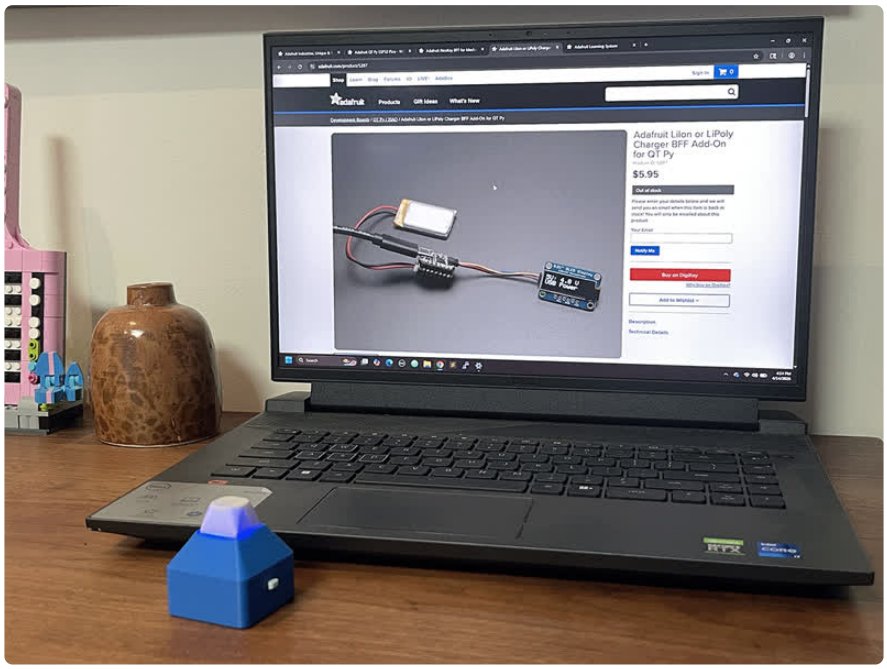
Table of Contents

Overview



Build a wireless, single button Bluetooth keyboard that you can use as a mute button, macro launcher, presentation clicker, and more. Plus, you can easily customize it from a WebSerial web page.

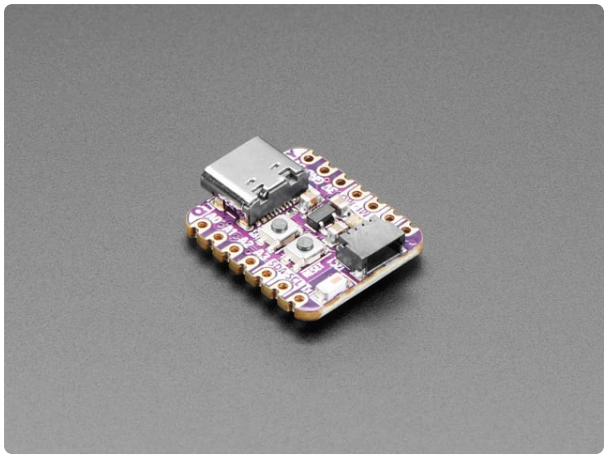
This project is inspired by the Monokey on Kickstarter (https://adafru.it/1aCC) (which was made in a limited run and is not currently available).



Features

- BLE HID keyboard with key combos, strings, media keys
- NVS persistence for all settings
- Configurable connected/pressed colors and brightness
- Auto sleep with user selected timeout
- Serial settings interface
- WebSerial interface web page

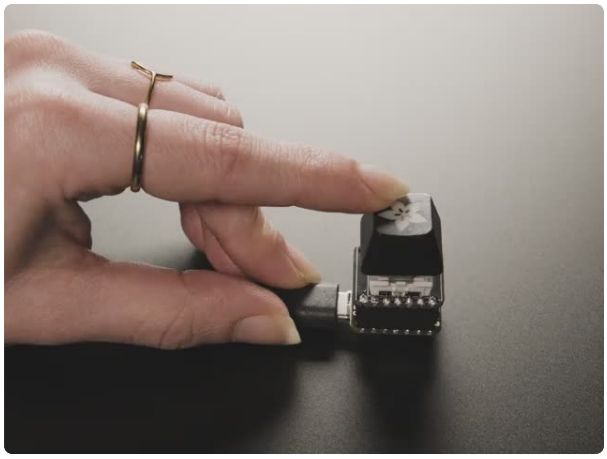
Parts



Adafruit QT Py ESP32 Pico - WiFi Dev Board with STEMMA QT

This dev board is like when you're watching a super-hero movie and the protagonist shows up in a totally amazing costume in the third act and you're like 'OMG! That's...

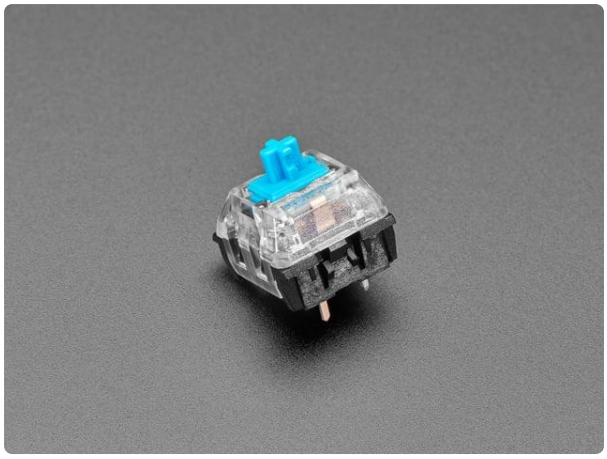
https://www.adafruit.com/product/5395



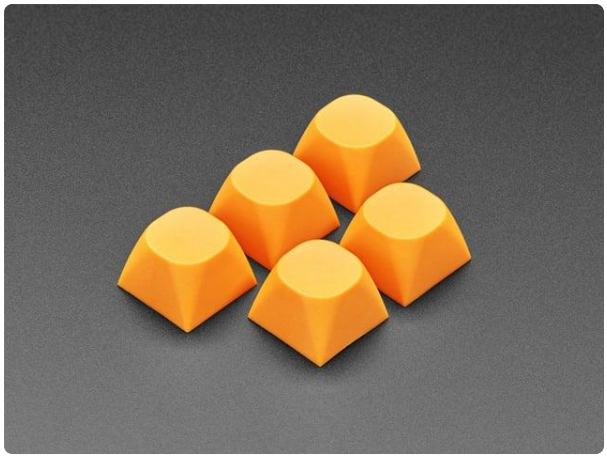
Adafruit NeoKey BFF for Mechanical Key Add-On for QT Py and Xiao

Our QT Py boards are a great way to make very small microcontroller projects that pack a ton of power - and now we have a way for you to quickly add a nice mechanical key that also can...

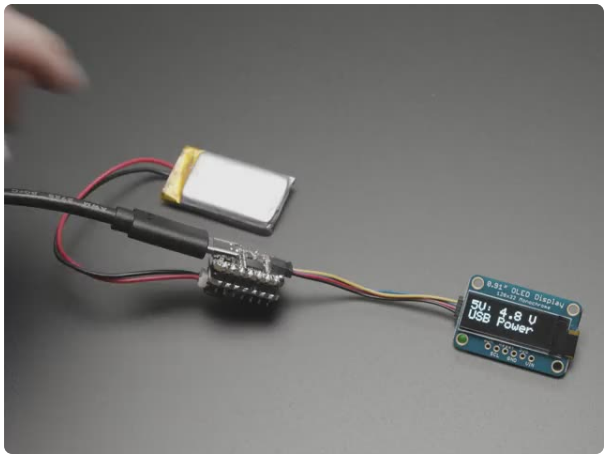
https://www.adafruit.com/product/5695



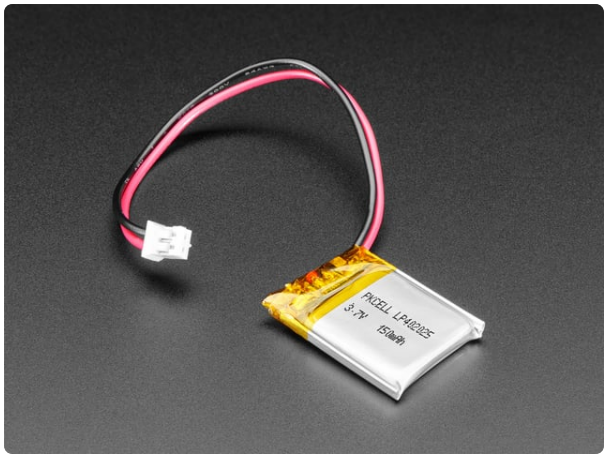
[Kailh Mechanical Key Switch - Clicky Blue - Single Piece]
For crafting your very own custom keyboard, a Kailh Blue Linear mechanical key switches is deeee-luxe! With smooth actuation and Cherry MX...
https://www.adafruit.com/product/5123



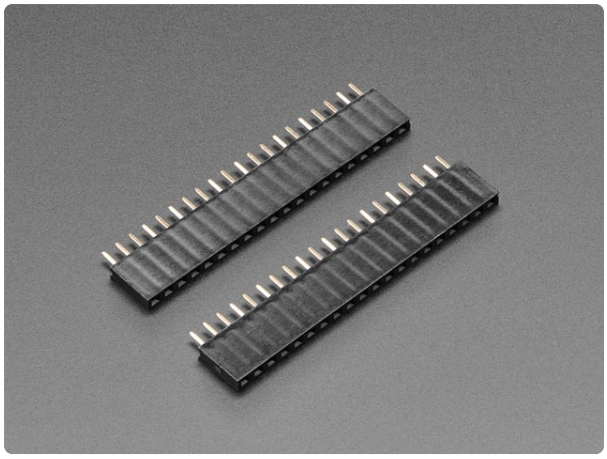
[Orange MA Keycaps for MX Compatible Switches - 5 pack]
Dress up your mechanical keys in your favorite colors with a wide selection of gumdrop-like, retro, curvy, and stylish MA profile keycaps. Here is a 5 pack of Orange MA...
https://www.adafruit.com/product/5175



[Adafruit LiIon or LiPoly Charger BFF Add-On for QT Py]
Is your QT Py all alone, lacking a friend to travel the wide world with? When you were a kid you may have learned...
https://www.adafruit.com/product/5397



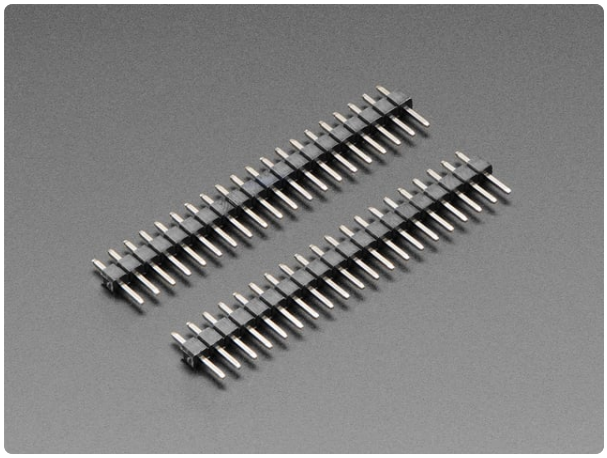
[Lithium Ion Polymer Battery - 3.7v 150mAh]
Lithium-ion polymer (also known as 'lipo' or 'lipoly') batteries are thin, light, and powerful. The output ranges from 4.2V when completely charged to 3.7V. This...
https://www.adafruit.com/product/1317



Socket Headers for Raspberry Pi Pico - 2 x 20 pin Female Headers

These Socket Headers alone are, well, lonely. But pair them with the Raspberry Pi Pico, and...

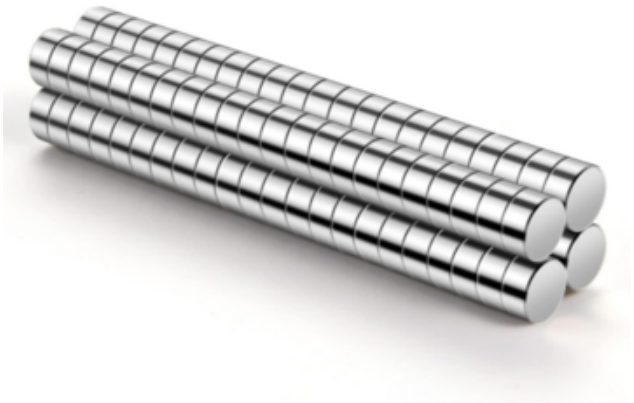
https://www.adafruit.com/product/5583



Short Plug Headers for Raspberry Pi Pico - 2 x 20 Male Headers

These two Short Plug / Male Headers alone are, well, lonely. But pair them with the

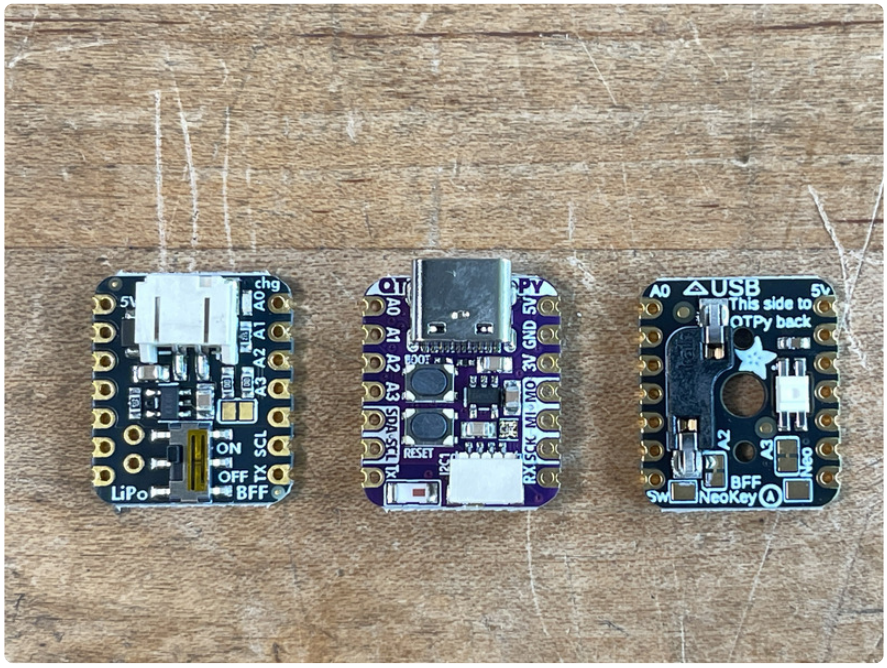
https://www.adafruit.com/product/5584

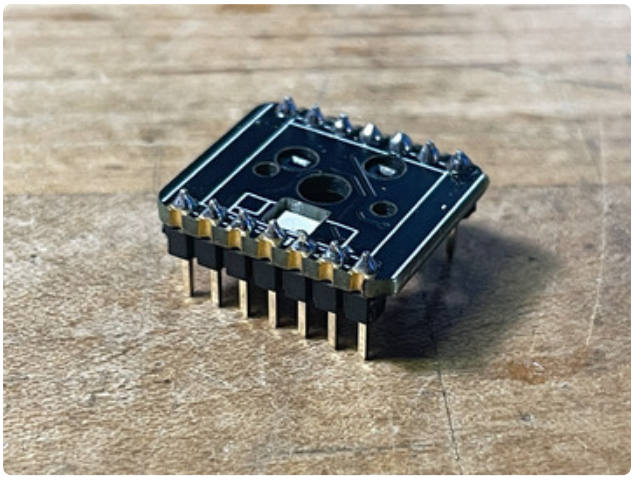


6mm x 3mm Neodymium Disc Magnets

Four magnets, such as these (https://adafru.it/1aCD), for securing the One Key case.

Connect the Boards

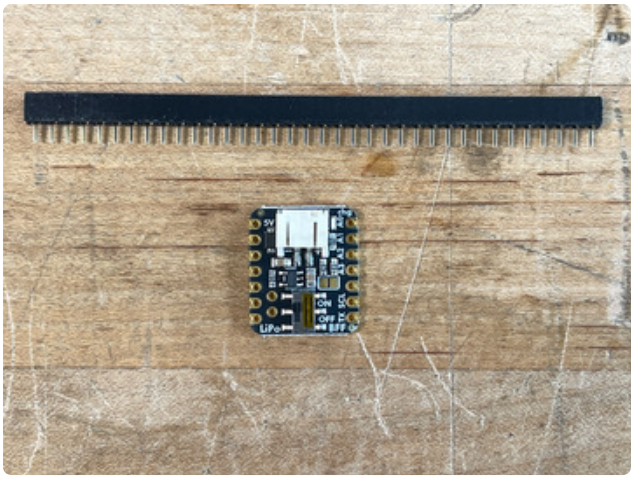


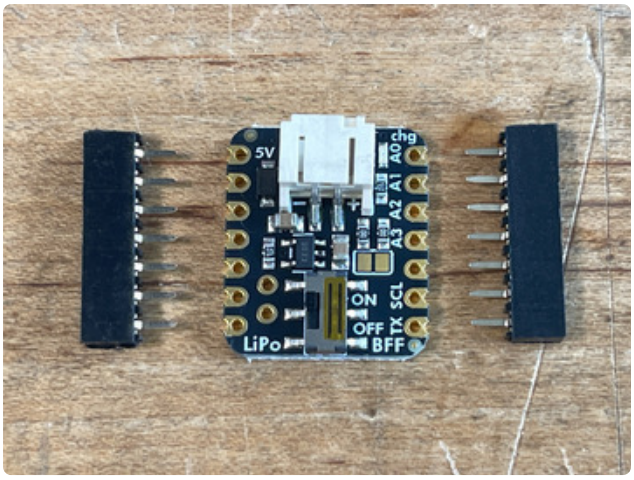


NeoKey BFF Header Pins

Cut two 7-pin sections of short male header pins.

Solder them with the legs facing the underside of the NeoKey BFF as shown here.

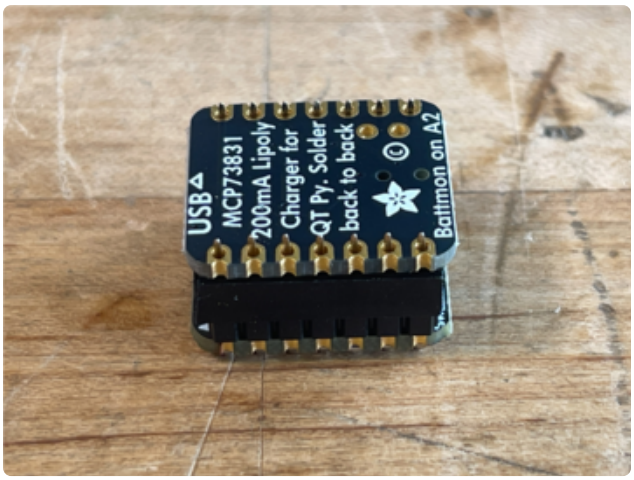


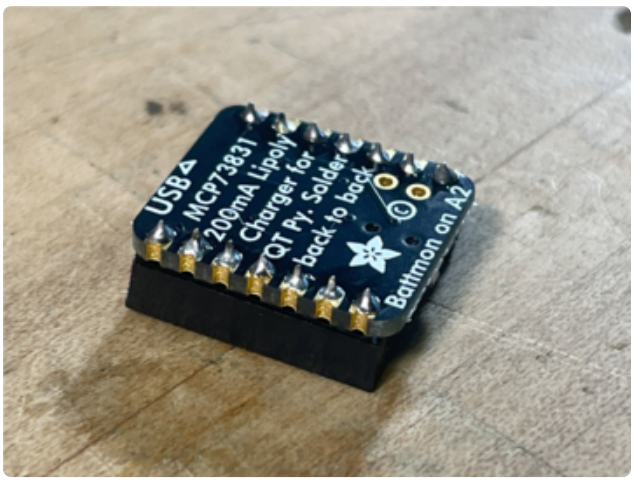


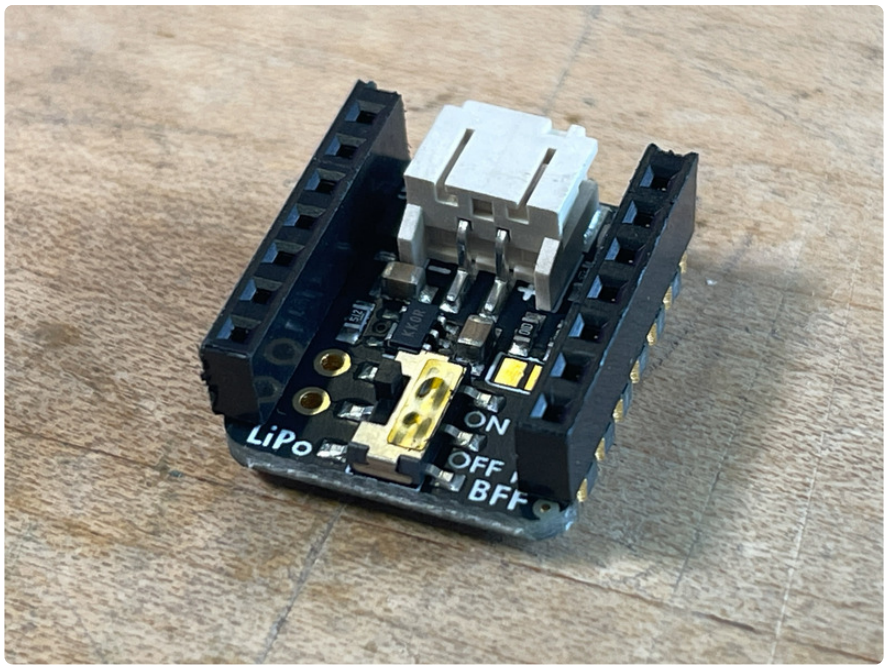
Lipoly BFF Headers

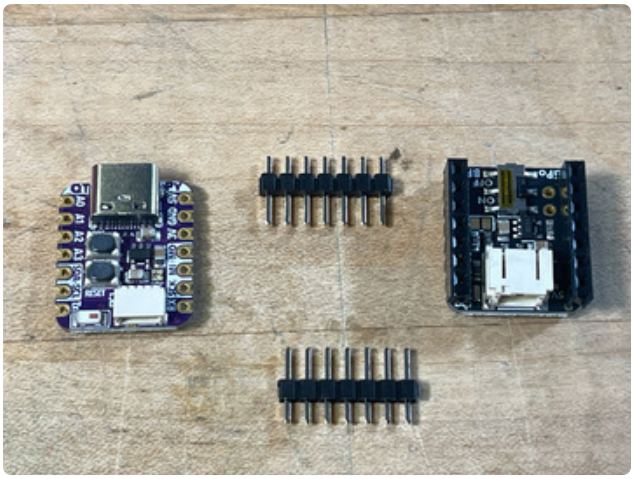
First cut two 7-pin sections of short female headers.

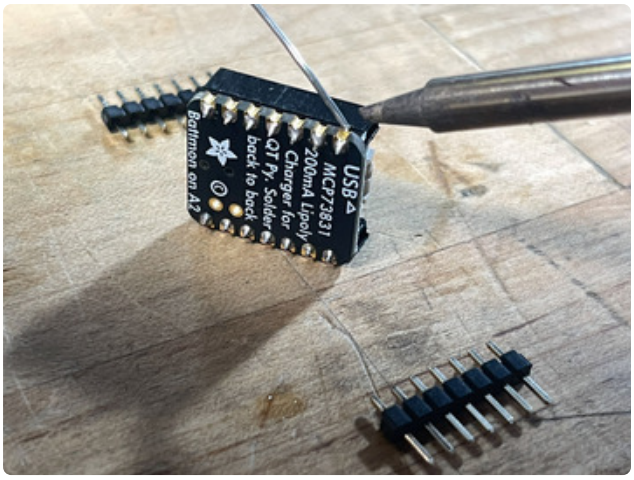
Solder them to the LiPoly BFF as shown.

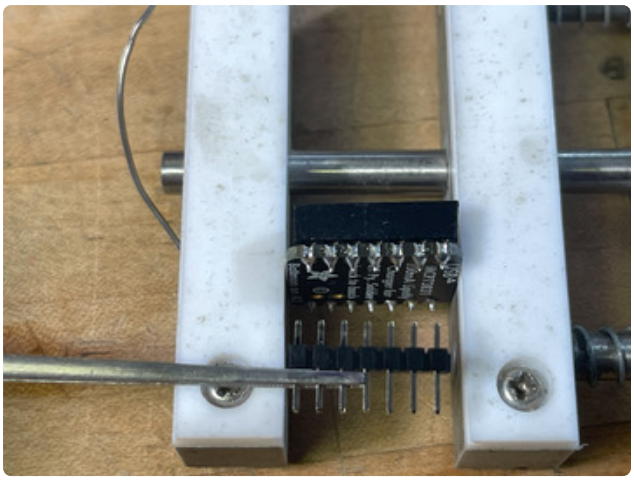


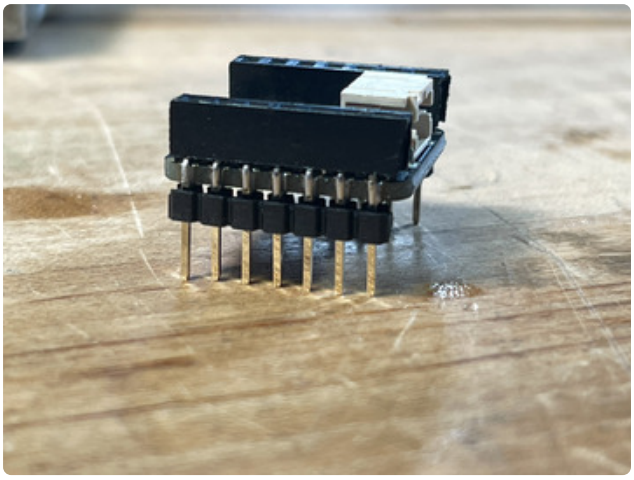



LiPo
ON
OFF









QT Py Connection

Here's a trick for making the shortest stack possible with our boards. The QT Py and BFF form factor includes castellated pads. We can tin those and solder short male header pins to join the QT Py and the LiPoly BFF!

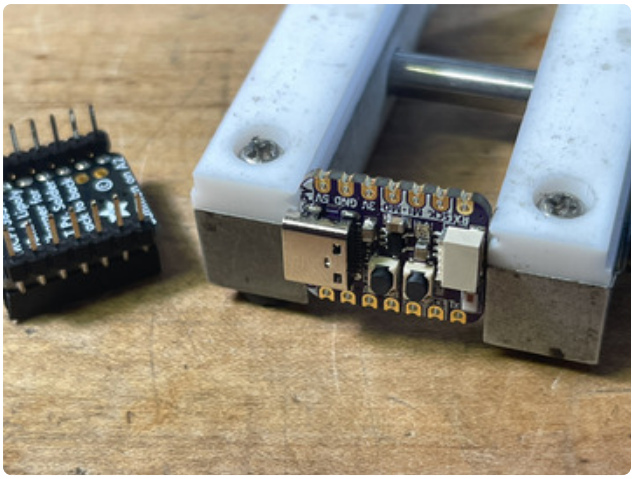
First, cut two 7-pin sections of short male header pins.

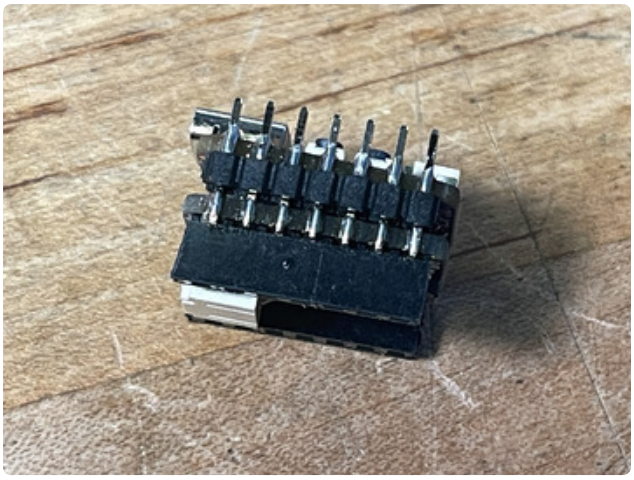
Then, tin the outer pads of the LiPoly BFF.

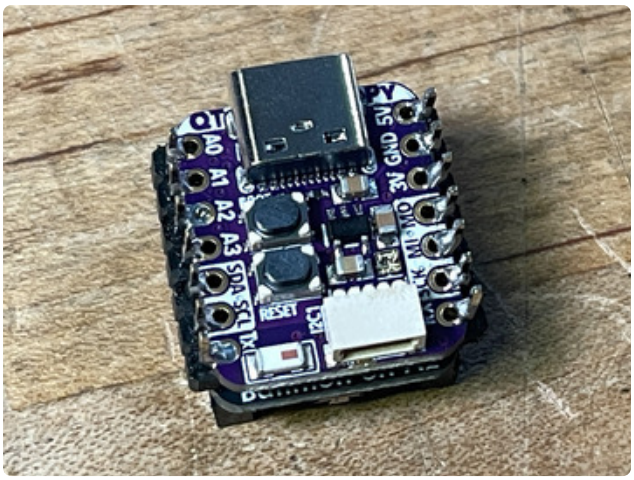
Hold the pins with pliers or tweezers to prevent cursing, then heat the pins to join them to the solder pads.

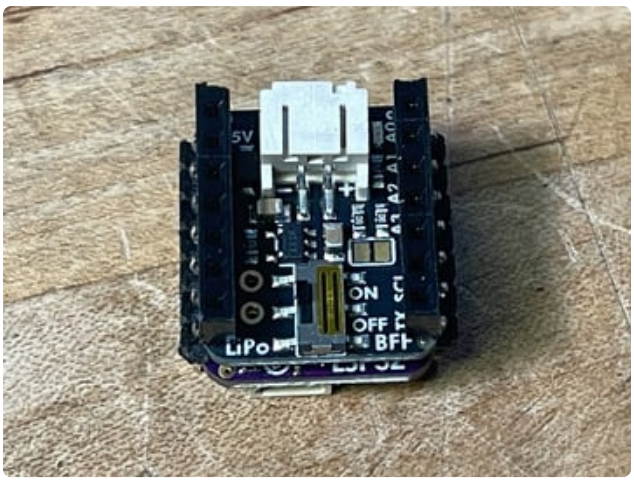
Push the plastic spacer up as far as it'll go.

Repeat for the other side of the board.



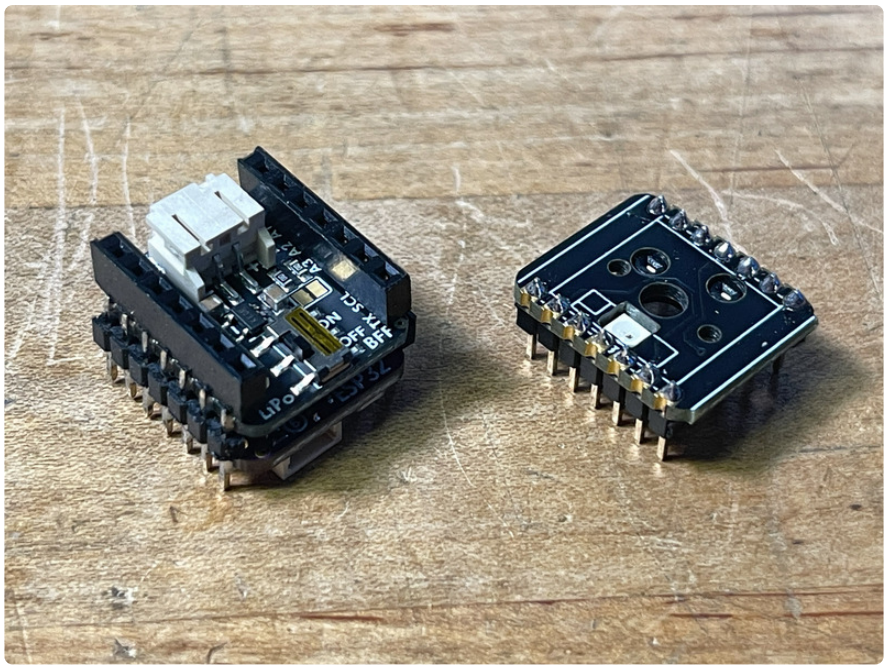


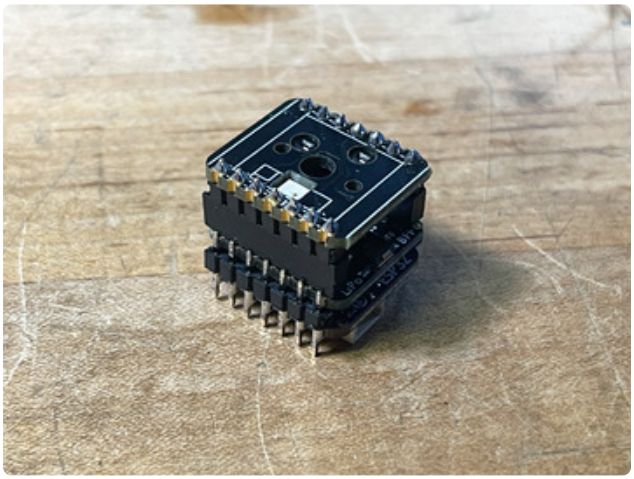




Repeat the tinning on the QT Py, then arrange the boards as shown, belly-to-belly with the LiPoly BFF battery JST connector on the same end as the QT Py's USB C connector.

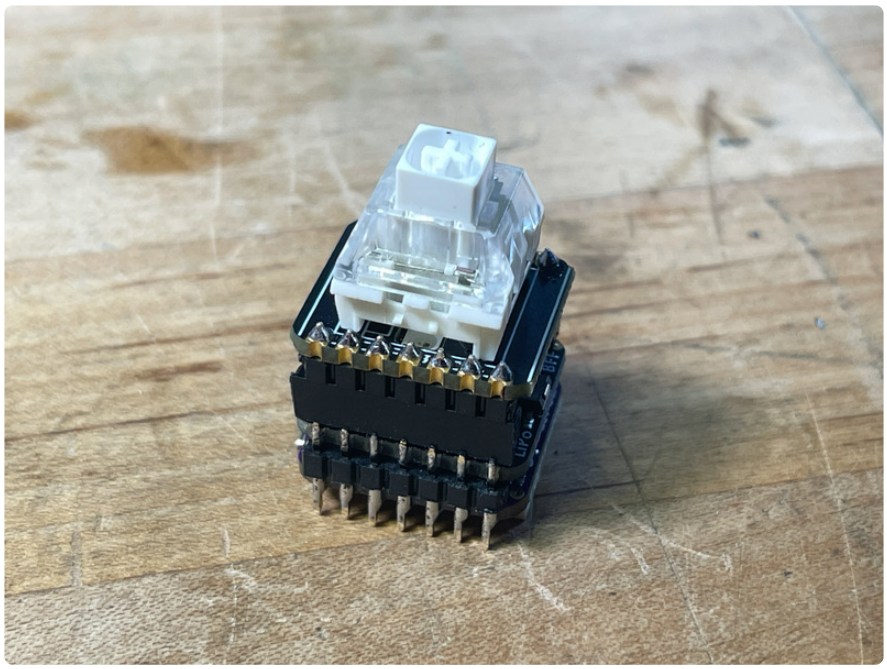
Heat the pins to solder join the two boards as shown.





Stack 'em Up

Now you can connect the NeoKey BFF to the LiPoly BFF. Heed the `^USB This side to QT Py back` silkscreen and press the header pins into the sockets as shown. (The NeoPixel is on the left side of the stack as shown in the photo above, with the LiPo power switch at the bottom.)



The next step is to code the One Key.

Arduino IDE Setup

You need to install the right USB-to-serial driver for your chip in addition to the Arduino IDE. If you are unsure which is the right one, install both!

Install Arduino IDE

The first thing you will need to do is to download the latest release of the Arduino IDE. You will need to be using **version 1.8** or higher for this guide

https://adafru.it/f1P

Install CP2104 / CP2102N USB Driver

The USB-to-Serial converter that talks to the ESP32 chip itself will need a driver on your computer's operating system. The driver is available for Mac and Windows. It is already built into Linux.

https://adafru.it/vrf

Install CH9102 / CH34X USB Driver

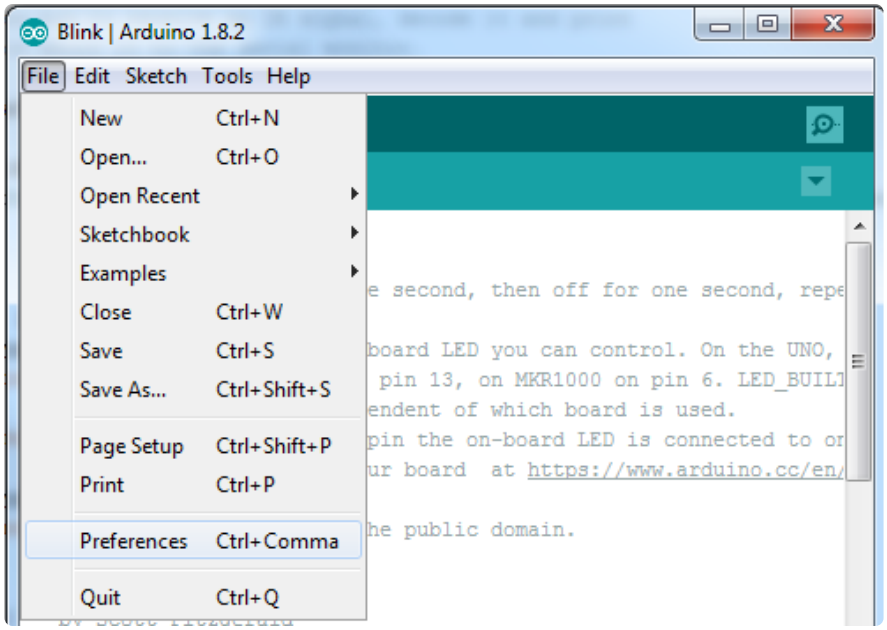
Newer ESP32 boards have a different USB-to-serial converter that talks to the chip itself, and will need a driver on your computer's operating system. The driver is available for Mac and Windows. It is already built into Linux.

If you would like more detail, check out the guide on installing these drivers (https://adafru.it/-f8).

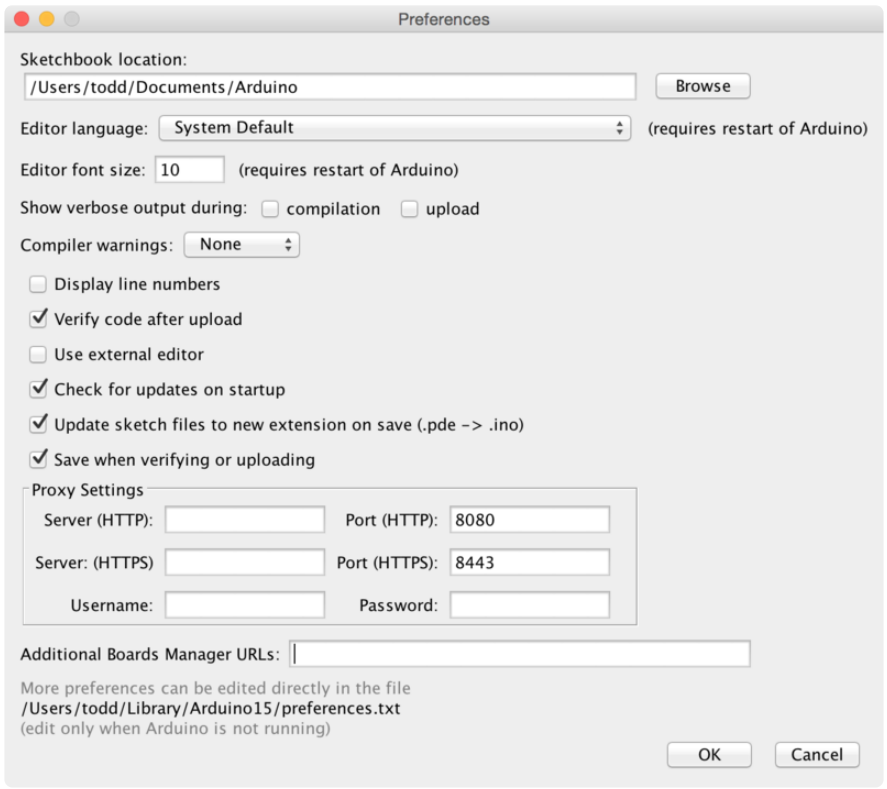
https://adafru.it/-f9

Install ESP32 Board Support Package

After you have downloaded and installed **the latest version of Arduino IDE**, you will need to start the IDE and navigate to the **Preferences** menu. You can access it from the **File** menu in Windows or Linux, or the **Arduino** menu on OS X.



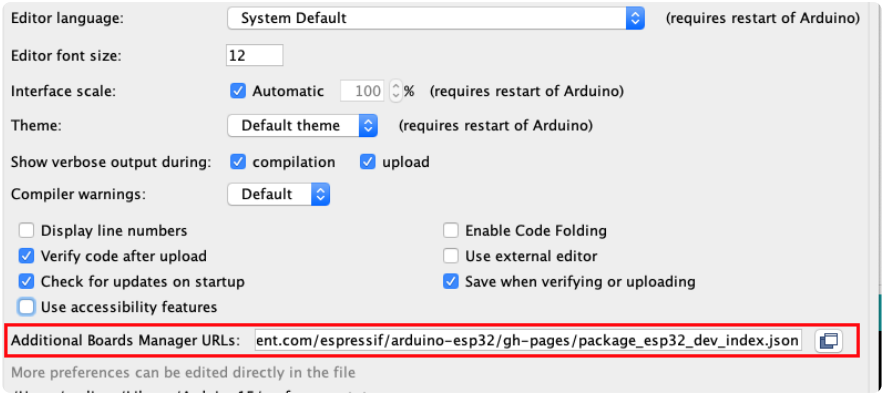
A dialog will pop up just like the one shown below.



We will be adding a URL to the new **Additional Boards Manager URLs** option. The list of URLs is comma separated, and you will only have to add each URL once. New Adafruit boards and updates to existing boards will automatically be picked up by the Board Manager each time it is opened. The URLs point to index files that the Board Manager uses to build the list of available & installed boards.

To find the most up to date list of URLs you can add, you can visit the list of third party board URLs on the Arduino IDE wiki (https://adafru.it/f7U). We will only need to add one URL to the IDE in this example, but **you can add multiple URLS by separating them with commas**. Copy and paste the link below into the **Additional Boards Manager URLs** option in the Arduino IDE preferences.

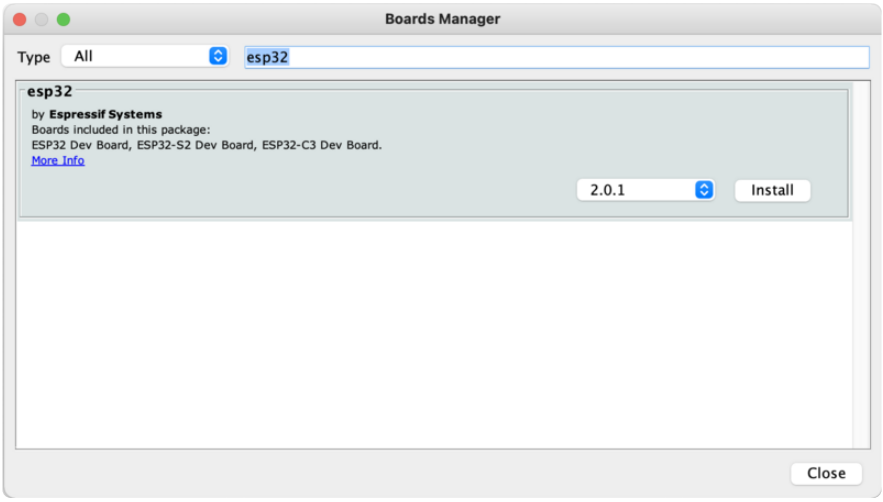
```
https://raw.githubusercontent.com/espressif/arduino-esp32/gh-pages/package_esp32_dev_index.json
```



If you have multiple boards you want to support, say ESP8266 and Adafruit, have both URLs in the text box separated by a comma (,)

Once done click **OK** to save the new preference settings.

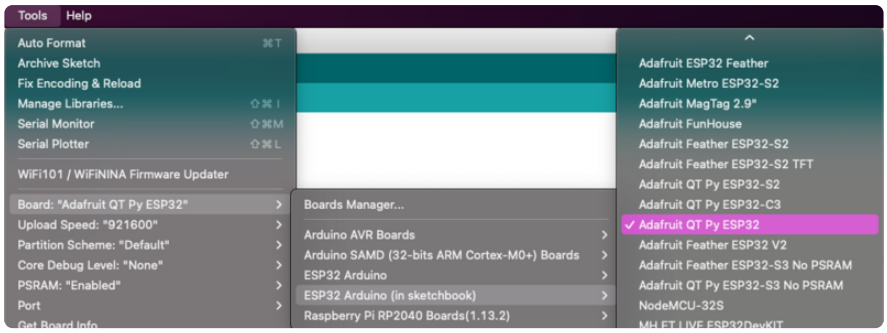
The next step is to actually install the Board Support Package (BSP). Go to the **Tools → Board → Board Manager** submenu. A dialog should come up with various BSPs. Search for **esp32**.



Click the **Install** button and wait for it to finish. Once it is finished, you can close the dialog.

In the **Tools → Board** submenu you should see **ESP32 Arduino** and in that dropdown it should contain the ESP32 boards along with all the latest ESP32 boards.

Look for the board called Adafruit QT Py ESP32.

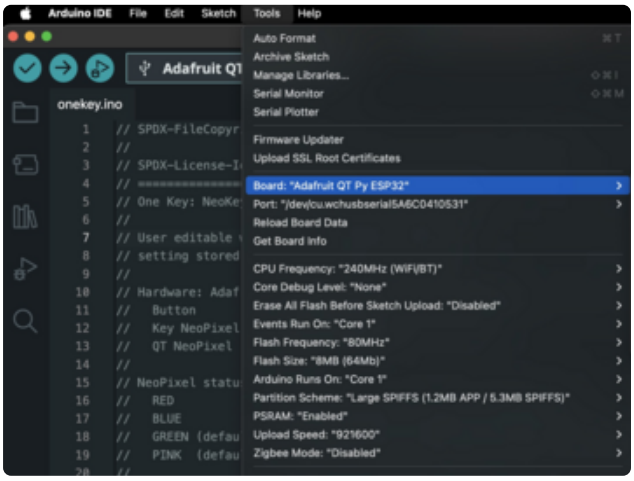


The upload speed can be changed: faster speed makes uploads take less time but sometimes can cause upload issues. **921600** should work fine, but if you're having issues, you can drop down lower.

Code the One Key

Now that you've got the Arduino IDE set up to use the QT Py ESP32, it's time to download the code, then you'll compile and upload it.

Download the Arduino file here (https://adafru.it/1aCt). Then open the **onekey.ino** sketch in Arduino.



Select Board & Port

On the QT Py, **hold** the **BOOT button**, **click-and-release** the **Reset button** then let go of the **BOOT button**. This puts the board into ROM Bootloader mode.

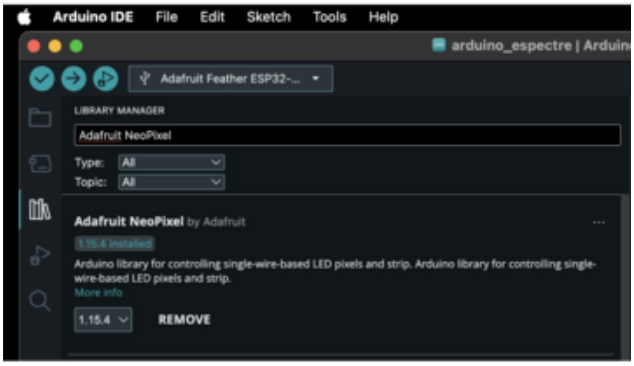
Then in the Arduino IDE, click: **Tools** > **Board** > **esp32** > **Adafruit QT Py ESP32**

Then, select the port by clicking: **Tools** > **Port** and then choose the port your Feather is on, in this case **/dev/cu.wchusbserial5AC0410531.** In Windows it will show up as a **COM** port.

Libraries

In Arduino, go to **Tools** > **Manage Libraries**, then install this library:

Adafruit NeoPixel



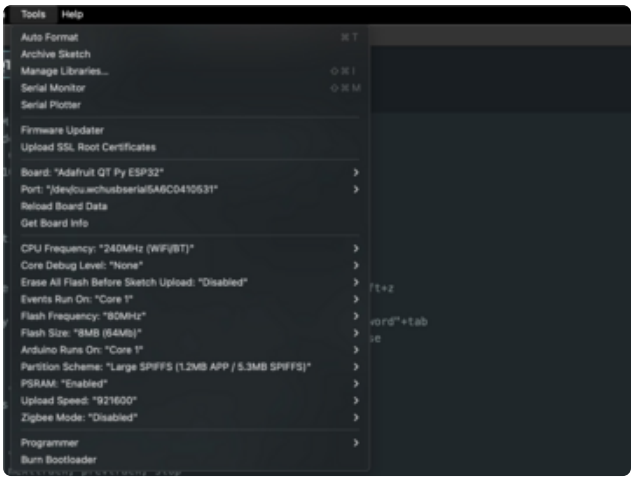
You'll also need to install the ESP32-BLE-Keyboard library from here (https://adafru.it/1aCu). To do so:

Go to the releases page at `github.com/sakul-the-one/sakuls-ESP32-BLE-Keyboard/releases` (https://adafru.it/1aCv) and download the ZIP from the latest release
Unzip it
Rename the folder to `ESP32_BLE_Keyboard`
Drop that folder directly into your Arduino Libraries folder (such as `~/Documents/Arduino/libraries/` on Mac)
Restart Arduino

Compile the Sketch

In Arduino, click **Sketch** > **Verify/Compile** to make sure everything can compile with the selected board and libraries.

When finished you should see a **Done compiling** message. Success!



Upload

You can now prep the upload settings. Click: **Tools** > and then set the following (the rest are defaults:

Partition Scheme: "Large SPIFFS (1.2MB APP / 5.3MB SPIFFS)
PSRAM: "Enabled"
Upload Speed: "921600"

The other settings should be the defaults, but you can double check against the screenshot here.

Then, click **Sketch** > **Upload** and it'll flash the board.

```
// SPDX-FileCopyrightText: 2026 John Park for Adafruit Industries
//
// SPDX-License-Identifier: MIT
// ============================================================
// One Key: NeoKey BFF — Single-Key BLE HID Keyboard
//
// User editable w serial (or WebSerial webpage)
// setting stored in NVS (non-volatile storage)
//
// Hardware: Adafruit QT Py ESP32 (Pico V3-02) + NeoKey BFF + Lipo Charger BFF
//   Button      : pin A2 (INPUT_PULLUP, LOW = pressed)
//   Key NeoPixel: pin A3 (NeoKey BFF)
//   QT NeoPixel : PIN_NEOPIXEL (onboard QT Py pixel)
//
// NeoPixel status colors:
//   RED             — waiting to connect (not user configurable)
//   BLUE            — going to sleep (not user configurable)
//   GREEN (default) — connected, user configurable via: color:connected:R,G,B
//   PINK  (default) — button pressed, user configurable via: color:pressed:R,G,B
//
// Sleep mode:
//   After sleepMinutes of inactivity the device turns off LEDs
//   and enters deep sleep to save battery. Flip the power switch
//   on the LiPo Charger BFF off and on to wake (full reboot).
//   Default is 10 minutes. To change via Serial Monitor:
//     sleep:10   — set to 10 minutes
//     sleep:0    — disable sleep entirely
//
// Key configuration (Serial Monitor, 115200 baud):
//   Single key       : q
//   Special key      : tab  or  f5  or  return
//   With modifier    : ctrl+z  or  alt+tab  or  shift+f5  or  ctrl+shift+z
//   String           : "hello world"
//   String + key     : "dir"+enter  or  "git status"+return  or  "password"+tab
//   Media key        : volumeup  or  volumedown  or  mute  or  playpause
//                      nexttrack  or  prevtrack  or  stop
//
//   Modifiers : ctrl, shift, alt, gui (gui = Win/Command key)
//   Special keys: tab, return, enter, esc, backspace, delete,
```

```
//                 up, down, left, right, f1-f24,
//                 space, home, end, pageup, pagedown, insert
//   Media keys: volumeup, volumedown, mute, playpause,
//               nexttrack, prevtrack, stop
//
// LED configuration (Serial Monitor, 115200 baud):
//   color:connected:0,255,0     — set connected color (default green)
//   color:pressed:255,20,100    — set pressed color (default pink)
//   bright:255                  — set brightness 0-255 (default 255)
//
// Libraries required:
//   - Adafruit NeoPixel         (Library Manager)
//   - sakuls-ESP32-BLE-Keyboard (github.com/sakul-the-one/sakuls-ESP32-BLE-
Keyboard)
//
// Board settings (Arduino IDE):
//   Board     : "Adafruit QT Py ESP32-Pico"
//   Partition : "Large SPIFFS"
// ============================================================

#include <BleKeyboard.h>
#include <Adafruit_NeoPixel.h>
#include <Preferences.h>
#include "esp_system.h"

#define NEOPIXEL_PIN          A3
#define BUTTON_PIN            A2
#define DEFAULT_BRIGHT        255
#define DEFAULT_SLEEP_MINUTES 10

BleKeyboard bleKeyboard("ESP32 BLE KB3", "Adafruit", 100);
Adafruit_NeoPixel key_pixel(1, NEOPIXEL_PIN, NEO_GRB + NEO_KHZ800);
Adafruit_NeoPixel qt_pixel(1, PIN_NEOPIXEL, NEO_GRB + NEO_KHZ800);
Preferences prefs;

uint32_t lastActivityMs = 0;
uint8_t  sleepMinutes   = DEFAULT_SLEEP_MINUTES;

// ---- LED color state ----------------------------------
uint8_t bright         = DEFAULT_BRIGHT;
uint8_t connR = 0,   connG = 255, connB = 0;    // connected: green
uint8_t pressR = 255, pressG = 20, pressB = 100; // pressed: pink

// ---- Key combo struct ---------------------------------
// mode: 0 = regular key, 1 = string, 2 = media
struct KeyCombo {
  uint8_t  modifier;   // 0 = none, else KEY_LEFT_CTRL etc.
  uint8_t  modifier2;  // optional second modifier
  uint8_t  key;        // main key (mode 0)
  char     str[64];    // text to type (mode 1)
  uint8_t  strTail;    // optional key after string (mode 1), or 0
  uint8_t  mode;       // 0=regular, 1=string, 2=media
  uint16_t mediaKey;   // packed MediaKeyReport (mode 2)
};

KeyCombo currentCombo = { 0, 0, 'q', "", 0, 0, 0 };

// ---- NeoPixel -----------------------------------------
void setPixels(uint8_t r, uint8_t g, uint8_t b) {
  key_pixel.setPixelColor(0, key_pixel.Color(r, g, b));
  key_pixel.show();
  qt_pixel.setPixelColor(0, qt_pixel.Color(r, g, b));
  qt_pixel.show();
}

void pixelWaiting()   { setPixels(255, 0, 0); }                  // red — not
configurable
void pixelConnected() { setPixels(connR, connG, connB); }         // user
configurable
```

```
void pixelPressed()   { setPixels(pressR, pressG, pressB); }      // user
configurable
void pixelYawn()      { setPixels(0, 0, 255); }                   // blue — not
configurable

// ---- Sleep ------------------------------------------------
void goToSleep() {
  Serial.println("[SLEEP] Inactivity timeout — going to deep sleep.");
  Serial.println("[SLEEP] Flip power switch to wake.");
  Serial.flush();
  pixelYawn();
  delay(1000);
  setPixels(0, 0, 0);
  digitalWrite(NEOPIXEL_POWER, LOW);
  esp_deep_sleep_start();
}

// ---- NVS --------------------------------------------------
void loadSettings() {
  prefs.begin("neokey", true);
  // Key combo
  currentCombo.modifier  = prefs.getUChar("mod1",      0);
  currentCombo.modifier2 = prefs.getUChar("mod2",      0);
  currentCombo.key       = prefs.getUChar("key",       'q');
  currentCombo.strTail   = prefs.getUChar("tail",      0);
  currentCombo.mode      = prefs.getUChar("mode",      0);
  currentCombo.mediaKey  = prefs.getUShort("mediakey", 0);
  prefs.getString("str", currentCombo.str, sizeof(currentCombo.str));
  // Sleep
  sleepMinutes = prefs.getUChar("sleepmin", DEFAULT_SLEEP_MINUTES);
  // LED
  bright  = prefs.getUChar("bright",  DEFAULT_BRIGHT);
  connR   = prefs.getUChar("conn_r",  0);
  connG   = prefs.getUChar("conn_g",  255);
  connB   = prefs.getUChar("conn_b",  0);
  pressR  = prefs.getUChar("press_r", 255);
  pressG  = prefs.getUChar("press_g", 20);
  pressB  = prefs.getUChar("press_b", 100);
  prefs.end();
}

void saveCombo(const KeyCombo& c) {
  prefs.begin("neokey", false);
  prefs.putUChar("mod1",      c.modifier);
  prefs.putUChar("mod2",      c.modifier2);
  prefs.putUChar("key",       c.key);
  prefs.putUChar("tail",      c.strTail);
  prefs.putUChar("mode",      c.mode);
  prefs.putUShort("mediakey", c.mediaKey);
  prefs.putString("str",      c.str);
  prefs.end();
  memcpy(&currentCombo, &c, sizeof(KeyCombo));
}

void saveSleepMinutes(uint8_t m) {
  prefs.begin("neokey", false);
  prefs.putUChar("sleepmin", m);
  prefs.end();
  sleepMinutes = m;
  if (m == 0)
    Serial.println("[CFG] Sleep disabled.");
  else
    Serial.printf("[CFG] Sleep timeout set to %d minute(s).\n", m);
}

void saveBrightness(uint8_t b) {
  prefs.begin("neokey", false);
  prefs.putUChar("bright", b);
  prefs.end();
```

```
  bright = b;
  key_pixel.setBrightness(bright);
  qt_pixel.setBrightness(bright);
  // Re-show current pixel state at new brightness
  key_pixel.show();
  qt_pixel.show();
  Serial.printf("[CFG] Brightness set to %d.\n", b);
}

void saveConnectedColor(uint8_t r, uint8_t g, uint8_t b) {
  prefs.begin("neokey", false);
  prefs.putUChar("conn_r", r);
  prefs.putUChar("conn_g", g);
  prefs.putUChar("conn_b", b);
  prefs.end();
  connR = r; connG = g; connB = b;
  Serial.printf("[CFG] Connected color set to %d,%d,%d.\n", r, g, b);
}

void savePressedColor(uint8_t r, uint8_t g, uint8_t b) {
  prefs.begin("neokey", false);
  prefs.putUChar("press_r", r);
  prefs.putUChar("press_g", g);
  prefs.putUChar("press_b", b);
  prefs.end();
  pressR = r; pressG = g; pressB = b;
  Serial.printf("[CFG] Pressed color set to %d,%d,%d.\n", r, g, b);
}

// ---- Media key helpers ------------------------------
uint16_t tokenToMediaKey(const String& t) {
  if (t == "volumeup")   return KEY_MEDIA_VOLUME_UP[0]      |
(KEY_MEDIA_VOLUME_UP[1] << 8);
  if (t == "volumedown") return KEY_MEDIA_VOLUME_DOWN[0]    |
(KEY_MEDIA_VOLUME_DOWN[1] << 8);
  if (t == "mute")       return KEY_MEDIA_MUTE[0]           | (KEY_MEDIA_MUTE[1] <<
8);
  if (t == "playpause")  return KEY_MEDIA_PLAY_PAUSE[0]     |
(KEY_MEDIA_PLAY_PAUSE[1] << 8);
  if (t == "nexttrack")  return KEY_MEDIA_NEXT_TRACK[0]     |
(KEY_MEDIA_NEXT_TRACK[1] << 8);
  if (t == "prevtrack")  return KEY_MEDIA_PREVIOUS_TRACK[0] |
(KEY_MEDIA_PREVIOUS_TRACK[1] << 8);
  if (t == "stop")       return KEY_MEDIA_STOP[0]           | (KEY_MEDIA_STOP[1] <<
8);
  return 0;
}

void uint16ToMediaKey(uint16_t v, MediaKeyReport& r) {
  r[0] = v & 0xFF;
  r[1] = (v >> 8) & 0xFF;
}

// ---- Token parser helpers -------------------------------
uint8_t tokenToModifier(const String& t) {
  if (t == "ctrl")  return KEY_LEFT_CTRL;
  if (t == "shift") return KEY_LEFT_SHIFT;
  if (t == "alt")   return KEY_LEFT_ALT;
  if (t == "gui")   return KEY_LEFT_GUI;
  if (t == "cmd")   return KEY_LEFT_GUI;
  if (t == "win")   return KEY_LEFT_GUI;
  return 0;
}

uint8_t tokenToKey(const String& t) {
  if (t == "tab")       return KEY_TAB;
  if (t == "return")    return KEY_RETURN;
  if (t == "enter")     return KEY_RETURN;
  if (t == "esc")       return KEY_ESC;
```

```
  if (t == "escape")    return KEY_ESC;
  if (t == "backspace") return KEY_BACKSPACE;
  if (t == "delete")    return KEY_DELETE;
  if (t == "up")        return KEY_UP_ARROW;
  if (t == "down")      return KEY_DOWN_ARROW;
  if (t == "left")      return KEY_LEFT_ARROW;
  if (t == "right")     return KEY_RIGHT_ARROW;
  if (t == "space")     return ' ';
  if (t == "home")      return KEY_HOME;
  if (t == "end")       return KEY_END;
  if (t == "pageup")    return KEY_PAGE_UP;
  if (t == "pagedown")  return KEY_PAGE_DOWN;
  if (t == "insert")    return KEY_INSERT;
  if (t.startsWith("f") && t.length() <= 3) {
    int n = t.substring(1).toInt();
    if (n >= 1 && n <= 24) return KEY_F1 + (n - 1);
  }
  if (t.length() == 1 && isprint(t.charAt(0))) return (uint8_t)t.charAt(0);
  return 0;
}

// ---- Print combo ----------------------------------------
void printCombo(const KeyCombo& c) {
  if (c.mode == 2) {  // media
    struct { const char* name; const MediaKeyReport* report; } mediaMap[] = {
      { "volumeup",   &KEY_MEDIA_VOLUME_UP },
      { "volumedown", &KEY_MEDIA_VOLUME_DOWN },
      { "mute",       &KEY_MEDIA_MUTE },
      { "playpause",  &KEY_MEDIA_PLAY_PAUSE },
      { "nexttrack",  &KEY_MEDIA_NEXT_TRACK },
      { "prevtrack",  &KEY_MEDIA_PREVIOUS_TRACK },
      { "stop",       &KEY_MEDIA_STOP },
    };
    String mediaDesc = "media:unknown";
    for (auto& m : mediaMap) {
      uint16_t v = (*m.report)[0] | ((*m.report)[1] << 8);
      if (v == c.mediaKey) { mediaDesc = String("media:") + m.name; break; }
    }
    Serial.printf("[CFG] Current combo: %s\n", mediaDesc.c_str());
    return;
  }
  if (c.mode == 1) {  // string
    String strDesc = "\"" + String(c.str) + "\"";
    if (c.strTail) {
      switch (c.strTail) {
        case KEY_RETURN:  strDesc += "+enter"; break;
        case KEY_TAB:     strDesc += "+tab"; break;
        case KEY_ESC:     strDesc += "+esc"; break;
        default:
          if (isprint(c.strTail)) strDesc += "+" + String((char)c.strTail);
          else strDesc += "+0x" + String(c.strTail, HEX);
          break;
      }
    }
    Serial.printf("[CFG] Current combo: %s\n", strDesc.c_str());
    return;
  }
  // mode 0 — regular key
  String keyDesc = "";
  if (c.modifier  == KEY_LEFT_CTRL)  keyDesc += "ctrl+";
  if (c.modifier  == KEY_LEFT_SHIFT) keyDesc += "shift+";
  if (c.modifier  == KEY_LEFT_ALT)   keyDesc += "alt+";
  if (c.modifier  == KEY_LEFT_GUI)   keyDesc += "gui+";
  if (c.modifier2 == KEY_LEFT_CTRL)  keyDesc += "ctrl+";
  if (c.modifier2 == KEY_LEFT_SHIFT) keyDesc += "shift+";
  if (c.modifier2 == KEY_LEFT_ALT)   keyDesc += "alt+";
  if (c.modifier2 == KEY_LEFT_GUI)   keyDesc += "gui+";
  switch (c.key) {
    case KEY_TAB:         keyDesc += "tab"; break;
```

```
    case KEY_RETURN:      keyDesc += "return"; break;
    case KEY_ESC:         keyDesc += "esc"; break;
    case KEY_BACKSPACE:   keyDesc += "backspace"; break;
    case KEY_DELETE:      keyDesc += "delete"; break;
    case KEY_UP_ARROW:    keyDesc += "up"; break;
    case KEY_DOWN_ARROW:  keyDesc += "down"; break;
    case KEY_LEFT_ARROW:  keyDesc += "left"; break;
    case KEY_RIGHT_ARROW: keyDesc += "right"; break;
    case KEY_HOME:        keyDesc += "home"; break;
    case KEY_END:         keyDesc += "end"; break;
    case KEY_PAGE_UP:     keyDesc += "pageup"; break;
    case KEY_PAGE_DOWN:   keyDesc += "pagedown"; break;
    case KEY_INSERT:      keyDesc += "insert"; break;
    default:
      if (c.key >= KEY_F1 && c.key <= KEY_F24)
        keyDesc += "f" + String(c.key - KEY_F1 + 1);
      else if (isprint(c.key))
        keyDesc += (char)c.key;
      else
        keyDesc += "0x" + String(c.key, HEX);
      break;
  }
  Serial.printf("[CFG] Current combo: %s\n", keyDesc.c_str());
}

// ---- Send combo ----------------------------------------
void sendCombo(const KeyCombo& c) {
  if (c.mode == 2) {  // media
    MediaKeyReport r;
    uint16ToMediaKey(c.mediaKey, r);
    bleKeyboard.press(r);
    delay(100);
    bleKeyboard.release(r);
    return;
  }
  if (c.mode == 1) {  // string
    bleKeyboard.print(c.str);
    if (c.strTail) {
      delay(50);
      bleKeyboard.press(c.strTail);
      delay(100);
      bleKeyboard.releaseAll();
    }
    return;
  }
  // mode 0 — regular key
  if (c.modifier)  bleKeyboard.press(c.modifier);
  if (c.modifier2) bleKeyboard.press(c.modifier2);
  bleKeyboard.press(c.key);
  delay(100);
  bleKeyboard.releaseAll();
}

// ---- Parse R,G,B from string ---------------------------
bool parseRGB(const String& s, uint8_t& r, uint8_t& g, uint8_t& b) {
  int c1 = s.indexOf(',');
  if (c1 < 0) return false;
  int c2 = s.indexOf(',', c1 + 1);
  if (c2 < 0) return false;
  r = (uint8_t)constrain(s.substring(0, c1).toInt(), 0, 255);
  g = (uint8_t)constrain(s.substring(c1 + 1, c2).toInt(), 0, 255);
  b = (uint8_t)constrain(s.substring(c2 + 1).toInt(), 0, 255);
  return true;
}

// ---- Serial input --------------------------------------
void checkSerial() {
  if (!Serial.available()) return;
  String input = Serial.readStringUntil('\n');
```

```
  input.trim();
  if (input.length() == 0) return;
  lastActivityMs = millis();

  // Sleep command: sleep:5  or  sleep:0
  if (input.startsWith("sleep:")) {
    int val = input.substring(6).toInt();
    if (val < 0 || val > 255) {
      Serial.println("[CFG] Invalid sleep value. Use 0-255 minutes.");
      return;
    }
    saveSleepMinutes((uint8_t)val);
    return;
  }

  // Brightness: bright:128
  if (input.startsWith("bright:")) {
    int val = input.substring(7).toInt();
    if (val < 0 || val > 255) {
      Serial.println("[CFG] Invalid brightness. Use 0-255.");
      return;
    }
    saveBrightness((uint8_t)val);
    return;
  }

  // Color: color:connected:R,G,B  or  color:pressed:R,G,B
  if (input.startsWith("color:")) {
    String rest = input.substring(6);
    uint8_t r, g, b;
    if (rest.startsWith("connected:")) {
      if (!parseRGB(rest.substring(10), r, g, b)) {
        Serial.println("[CFG] Format: color:connected:R,G,B");
        return;
      }
      saveConnectedColor(r, g, b);
      pixelConnected();  // preview new color immediately
    } else if (rest.startsWith("pressed:")) {
      if (!parseRGB(rest.substring(8), r, g, b)) {
        Serial.println("[CFG] Format: color:pressed:R,G,B");
        return;
      }
      savePressedColor(r, g, b);
    } else {
      Serial.println("[CFG] Unknown color target. Use: connected or pressed");
    }
    return;
  }

  // String mode: "hello"  or  "dir"+enter
  if (input.startsWith("\"")) {
    int closeQuote = input.indexOf('"', 1);
    if (closeQuote < 0) { Serial.println("[CFG] Missing closing quote."); return; }
    String s = input.substring(1, closeQuote);
    if (s.length() == 0) { Serial.println("[CFG] Empty string — no changes made.");
return; }
    if (s.length() >= sizeof(currentCombo.str)) {
      Serial.printf("[CFG] String too long (max %d chars).\n",
(int)sizeof(currentCombo.str) - 1);
      return;
    }
    KeyCombo parsed = { 0, 0, 0, "", 0, 1, 0 };
    s.toCharArray(parsed.str, sizeof(parsed.str));
    String tail = input.substring(closeQuote + 1);
    tail.trim();
    if (tail.startsWith("+")) {
      String tailToken = tail.substring(1);
      tailToken.trim();
      tailToken.toLowerCase();
```

```
      uint8_t tailKey = tokenToKey(tailToken);
      if (tailKey == 0) {
        Serial.printf("[CFG] Unrecognized tail key: '%s'\n", tailToken.c_str());
        return;
      }
      parsed.strTail = tailKey;
    }
    saveCombo(parsed);
    Serial.print("[CFG] Saved! ");
    printCombo(currentCombo);
    return;
  }

  // Combo / media / single key mode
  input.toLowerCase();
  KeyCombo parsed = { 0, 0, 0, "", 0, 0, 0 };
  uint8_t modCount = 0;
  String token = "";
  input += "+";  // sentinel

  for (int i = 0; i < (int)input.length(); i++) {
    char ch = input.charAt(i);
    if (ch == '+') {
      if (token.length() == 0) { token = ""; continue; }
      uint16_t mk = tokenToMediaKey(token);
      if (mk) {
        parsed.mode     = 2;
        parsed.mediaKey = mk;
        break;
      }
      uint8_t mod = tokenToModifier(token);
      if (mod) {
        if (modCount == 0) { parsed.modifier  = mod; modCount++; }
        else               { parsed.modifier2 = mod; modCount++; }
      } else {
        parsed.key = tokenToKey(token);
        if (parsed.key == 0) {
          Serial.printf("[CFG] Unrecognized key token: '%s'\n", token.c_str());
          Serial.println("[CFG] No changes made.");
          return;
        }
      }
      token = "";
    } else {
      token += ch;
    }
  }

  if (parsed.mode != 2 && parsed.key == 0) {
    Serial.println("[CFG] No valid key found. Format: q  or  ctrl+z  or  volumeup");
    return;
  }

  saveCombo(parsed);
  Serial.print("[CFG] Saved! ");
  printCombo(currentCombo);
}

// ---- Setup ------------------------------------------------
void setup() {
  Serial.begin(115200);
  delay(750);

  esp_reset_reason_t reason = esp_reset_reason();
  Serial.printf("[BOOT] Reset reason: %d", reason);
  switch (reason) {
    case ESP_RST_PANIC:    Serial.println(" (PANIC/crash)"); break;
    case ESP_RST_INT_WDT:  Serial.println(" (interrupt watchdog)"); break;
    case ESP_RST_TASK_WDT: Serial.println(" (task watchdog)"); break;
```

```
    case ESP_RST_WDT:      Serial.println(" (other watchdog)"); break;
    case ESP_RST_BROWNOUT: Serial.println(" (brownout)"); break;
    default:               Serial.println(" (normal/power-on)"); break;
  }

  pinMode(NEOPIXEL_POWER, OUTPUT);
  digitalWrite(NEOPIXEL_POWER, HIGH);
  key_pixel.begin();
  qt_pixel.begin();

  loadSettings();

  key_pixel.setBrightness(bright);
  qt_pixel.setBrightness(bright);
  pixelWaiting();

  pinMode(BUTTON_PIN, INPUT_PULLUP);

  Serial.print("[CFG] Loaded — ");
  printCombo(currentCombo);
  Serial.printf("[CFG] Connected color: %d,%d,%d\n", connR, connG, connB);
  Serial.printf("[CFG] Pressed color:   %d,%d,%d\n", pressR, pressG, pressB);
  Serial.printf("[CFG] Brightness:      %d\n", bright);
  if (sleepMinutes > 0)
    Serial.printf("[CFG] Sleep timeout: %d minute(s). To change: sleep:10\n",
sleepMinutes);
  else
    Serial.println("[CFG] Sleep disabled. To enable: sleep:5");
  Serial.println("[CFG] Examples: q    ctrl+z    f5    \"hello\"    volumeup
sleep:5");
  Serial.println("[CFG]           color:connected:0,255,0
color:pressed:255,20,100    bright:128");

  bleKeyboard.set_vendor_id(0x05AC);
  bleKeyboard.set_product_id(0x0220);
  bleKeyboard.set_version(0x0111);
  bleKeyboard.begin();
  Serial.println("[BLE] Advertising...");

  lastActivityMs = millis();
}

// ---- Loop ------------------------------------------------
void loop() {
  static bool lastBtn       = HIGH;
  static bool lastConnected = false;

  checkSerial();

  if (sleepMinutes > 0) {
    if (millis() - lastActivityMs > (uint32_t)sleepMinutes * 60 * 1000) {
      goToSleep();
    }
  }

  bool connected = bleKeyboard.isConnected();
  bool reading   = digitalRead(BUTTON_PIN);

  if (connected && !lastConnected) {
    Serial.println("[BLE] Connected");
    pixelConnected();
  } else if (!connected && lastConnected) {
    Serial.println("[BLE] Disconnected");
    pixelWaiting();
  }
  lastConnected = connected;

  if (reading == LOW && lastBtn == HIGH) {
    lastActivityMs = millis();
```

```
      pixelPressed();
      if (connected) {
        Serial.print("[KEY] Sending — ");
        printCombo(currentCombo);
        sendCombo(currentCombo);
      } else {
        Serial.println("[BTN] Not connected");
      }
    }
    if (reading == HIGH && lastBtn == LOW) {
      connected ? pixelConnected() : pixelWaiting();
    }

    lastBtn = reading;
    delay(10);
}
```

How the Code Works

The One Key sketch handles Bluetooth LE keyboard emulation, NeoPixel status feedback, serial configuration, and deep sleep power management. Here's a walkthrough of how all the pieces fit together.

Libraries

The sketch uses three libraries:

- **BleKeyboard** (`sakul-the-one` fork) handles all the Bluetooth LE HID protocol work, advertising the device as a keyboard and sending key reports to the connected host.
- **Adafruit NeoPixel** drives the two LEDs: one on the NeoKey BFF (pin A3) and one built into the QT Py board (PIN_NEOPIXEL).
- **Preferences** provides easy read/write access to the ESP32's NVS (non-volatile storage) flash partition, where all user settings are saved.

Key Combinations

Button press config is stored in a single `KeyCombo` struct. The `mode` field determines which kind of action is configured:

- **Mode 0** is a regular keypress, with optional modifier keys (Ctrl, Shift, Alt, GUI)
- **Mode 1** is a text string, with an optional trailing keypress (such as Return after a typed command)
- **Mode 2** is a media key (volume, playback, etc.)

This struct is what gets saved to and loaded from NVS on every boot.

NVS Settings

All user-configurable settings are stored in the ESP32's NVS flash under the namespace `"neokey"`. The `loadSettings()` function reads everything at startup: the key combo fields, sleep timeout, brightness, and the RGB values for the connected and pressed LED colors. Individual save functions handle each category of setting so only the relevant NVS keys are written when something changes.

Because NVS is retained across power cycles and deep sleep reboots, the One Key always comes back with the last configuration you've set.

NeoPixel Status Colors

Four pixel states give feedback about what the device is up to:

- `pixelWaiting()` shows red while the device is advertising and waiting for a BLE connection
- `pixelConnected()` shows the user selected connected color (green by default) once a host connects
- `pixelPressed()` shows the user selected pressed color (pink by default) while the button is held down
- `pixelYawn()` flashes blue for one second just before the device enters deep sleep

The connected and pressed colors are loaded from NVS at boot.

Serial Configuration

The `checkSerial()` function is called every loop iteration. It reads a line from the serial port and parses it into one of several command formats.

String mode is detected by a leading quote character. The parser finds the closing quote, extracts the text, and then checks for an optional `+key` suffix after the closing quote.

For regular key and combo input, the parser splits on `+` characters and classifies each token. It checks for media key names first, then modifier names (ctrl, shift, alt, gui), and finally looks up the key name or treats a single character as a literal keypress.

LED and sleep commands use a simple prefix format: `color:connected:`, `color:pressed:`, `bright:`, and `sleep:` are each detected by `startsWith()` and parsed accordingly.

Every successful command saves to NVS immediately and prints a confirmation to serial so you know the change took effect.

Sending Keystrokes

The `sendCombo()` function handles all three modes. For media keys, it presses and releases a `MediaKeyReport`. For strings, it calls `bleKeyboard.print()` and optionally follows up with a key press and `releaseAll()`. For regular keys, it presses any modifiers first, then the main key, then calls `releaseAll()`.

Button Handling

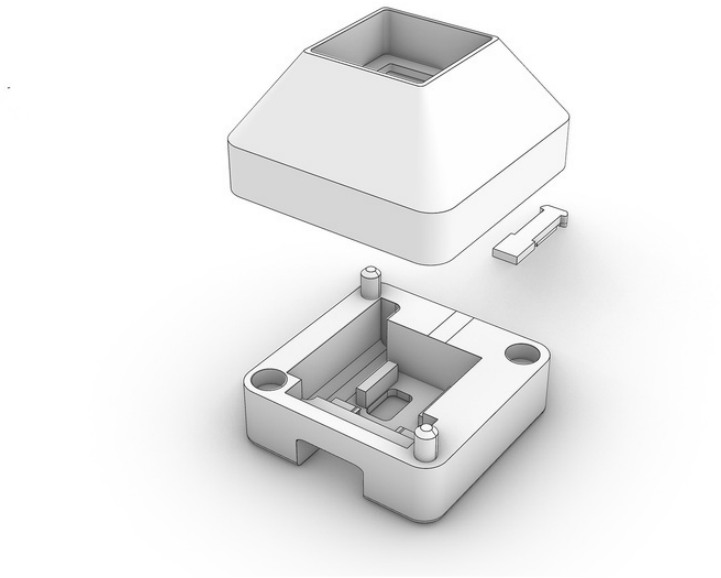
The main loop reads the button pin (A2) every 10ms. The NeoKey BFF uses a pullup resistor, so the pin reads LOW when pressed. A state comparison between the current and previous reading detects press and release transitions without a debounce timer.

On press, the activity timer resets, the pixel changes to the pressed color, and `sendCombo()` is called if BLE is connected. On release, the pixel returns to the appropriate status color.

Deep Sleep

If `sleepMinutes` is non-zero and no button press or serial input has occurred within the timeout period, `goToSleep()` is called. It prints a warning to serial, flashes the blue yawn color for one second, powers down the NeoPixel rail via the `NEOPIXEL_POWER` pin, and calls `esp_deep_sleep_start()` with no wakeup source configured. The device draws only a few hundred microamps in this state. To wake it, the user flips the power switch on the LiPo Charger BFF, which triggers a full reboot and runs `setup()` again, reloading all settings from NVS.

3D Print and Assemble the Case



To 3D print the case, grab the 3MF file below and load it into a compatible slicer such as Bambu Studio. Or you can grab the individual STL files.

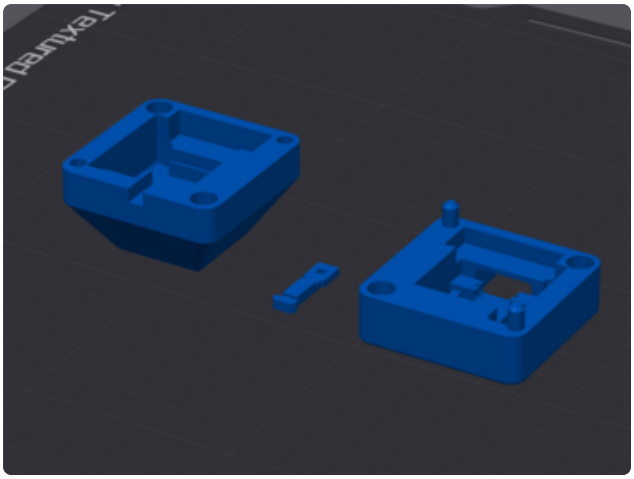
3MF and STL files for 3D printing are oriented to print "as-is" on FDM style machines. Some of the parts are designed to 3D print with tree supports material using PLA filament.

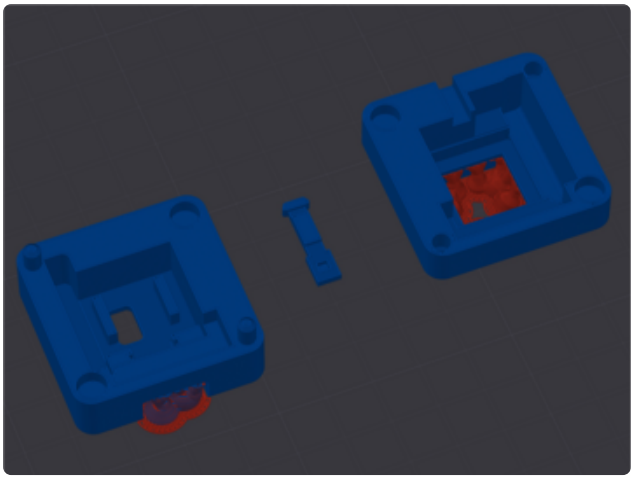
If you'd like to edit the model then grab the STEP file.

https://adafru.it/1aCq

https://adafru.it/1aCr

https://adafru.it/1aCs

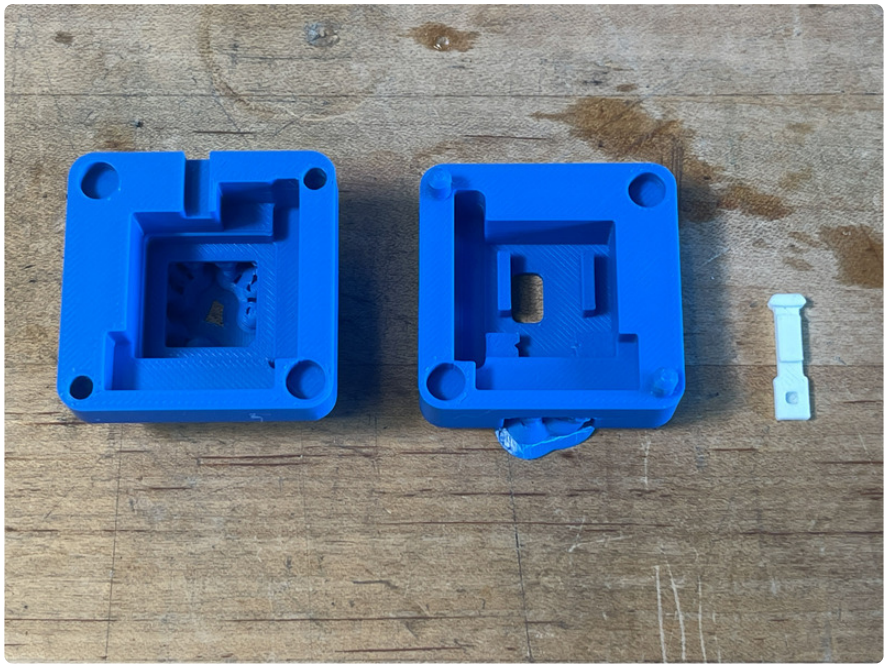




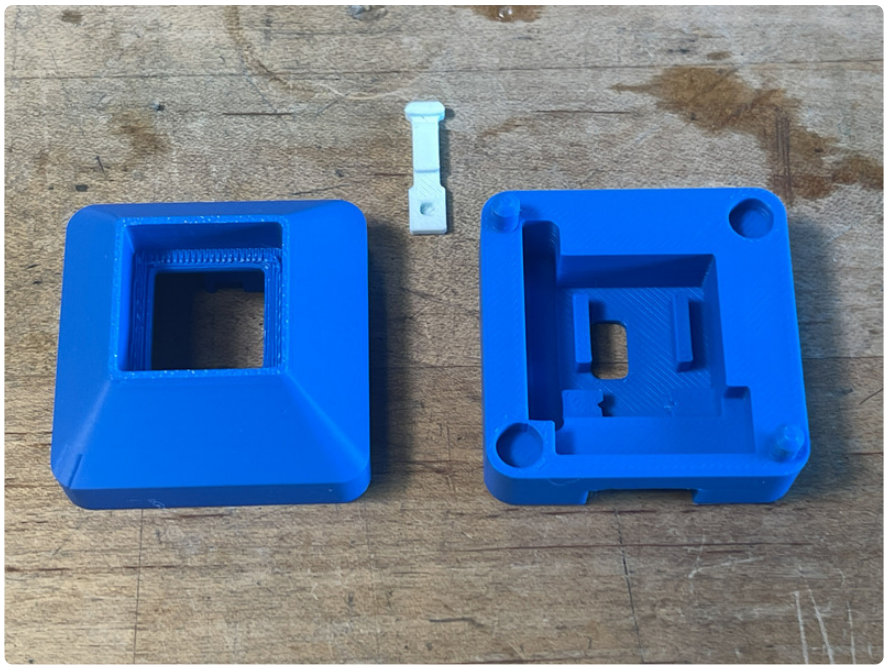
3D Print Settings

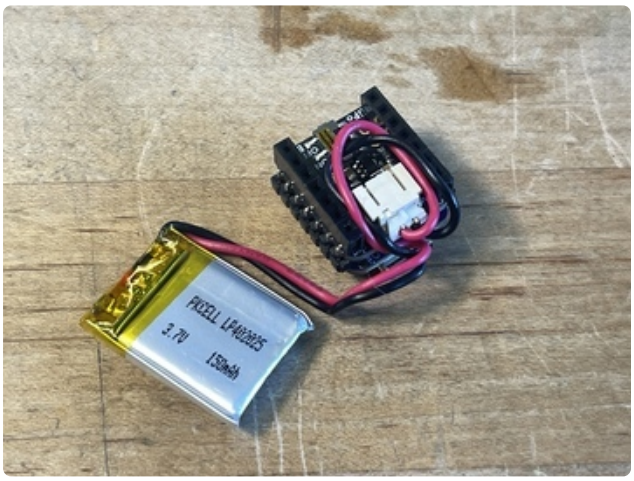
Print the models in the orientation shown here. I printed at 0.16mm layer height with supports turned on for the base and lid with a threshold angle of 25°. This allows the slicer to support just the internal parts without messing with the angled lid.

You can see the supports here, set to red just for illustration purposes.



Remove the supports with a pair of pliers or ask your pet sloth to gnaw them off for you.

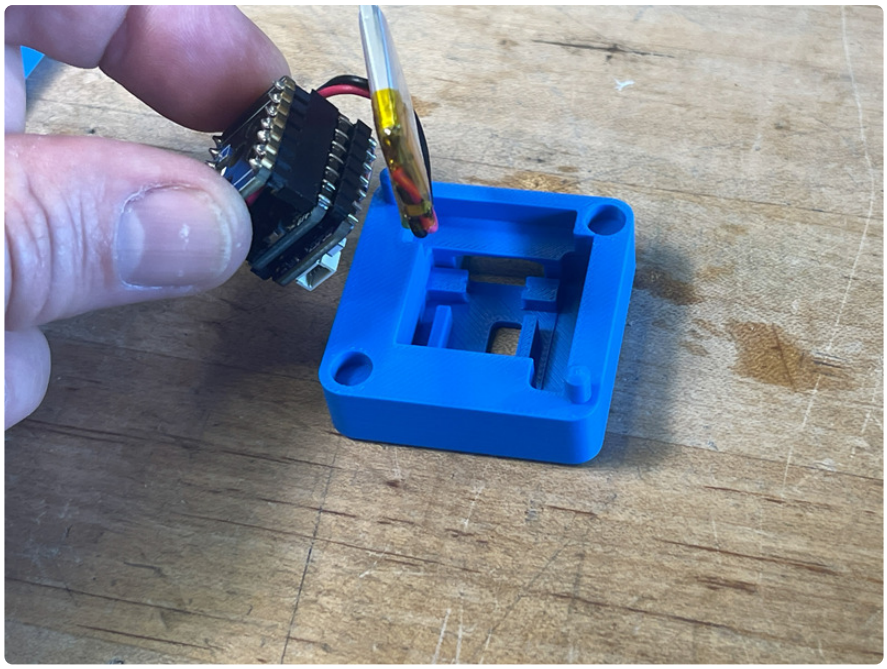


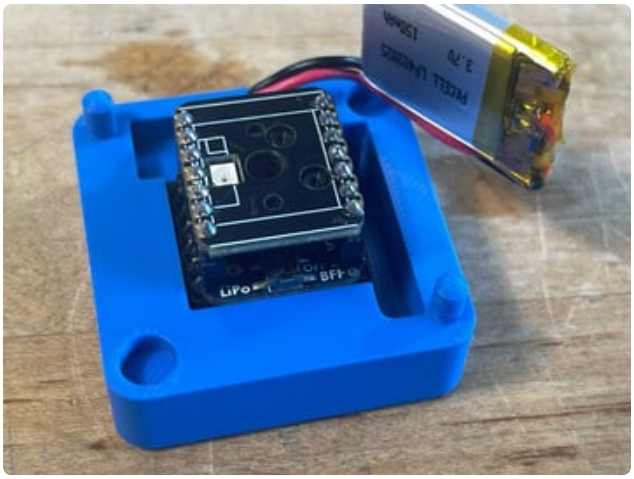


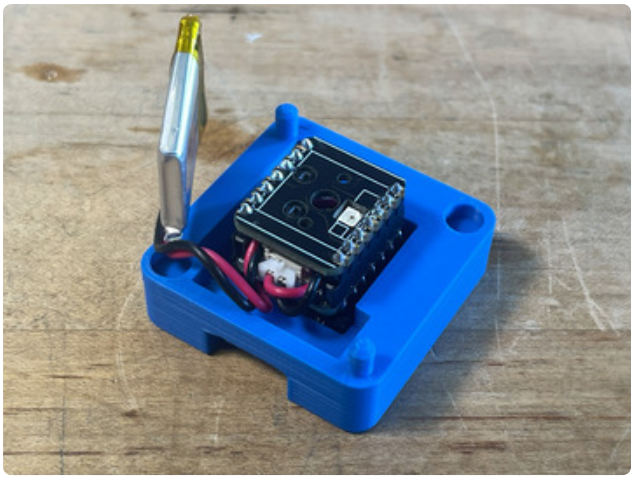
Battery Fit

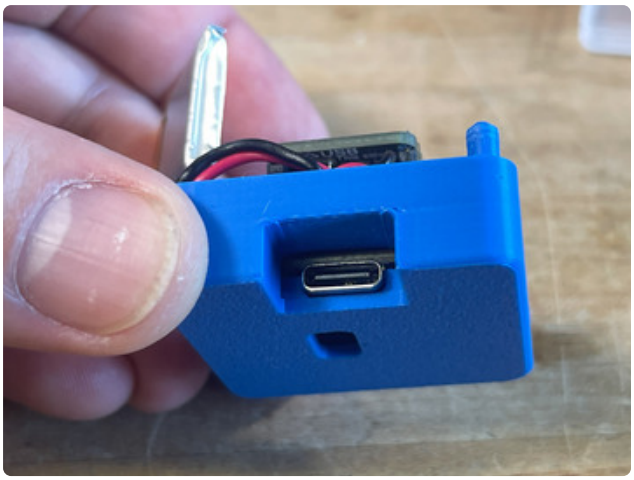
Temporarily lift off the NeoKey BFF, then plug in the battery as shown. You can wrap some of the excess wire around the JST connector as shown, just make sure the power switch can still be accessed and flipped between positions.

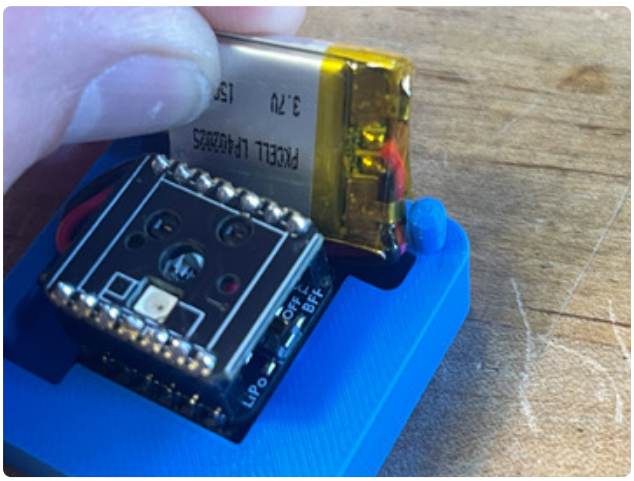
Then re-connect the NeoKey BFF as before.







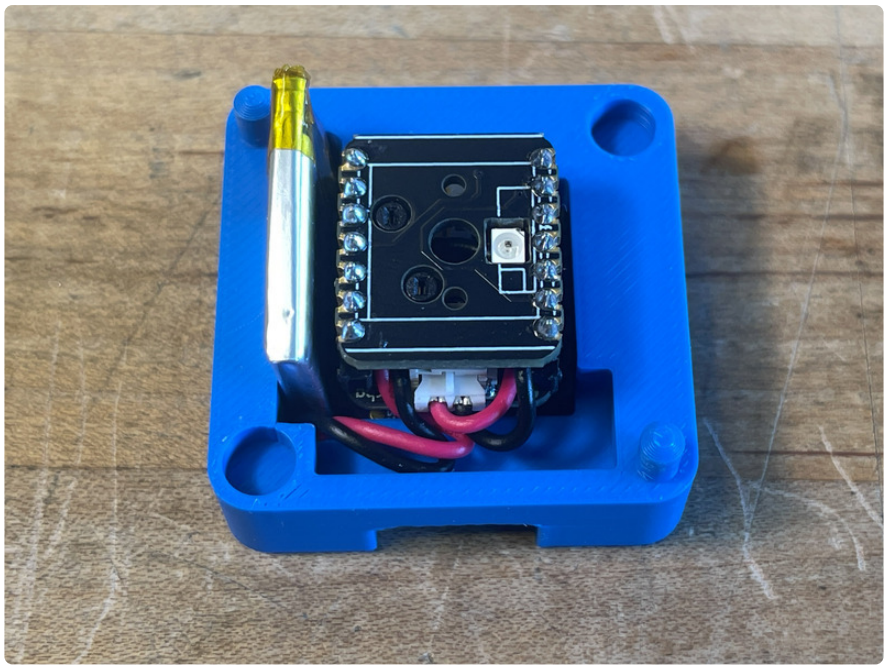




Case Fit

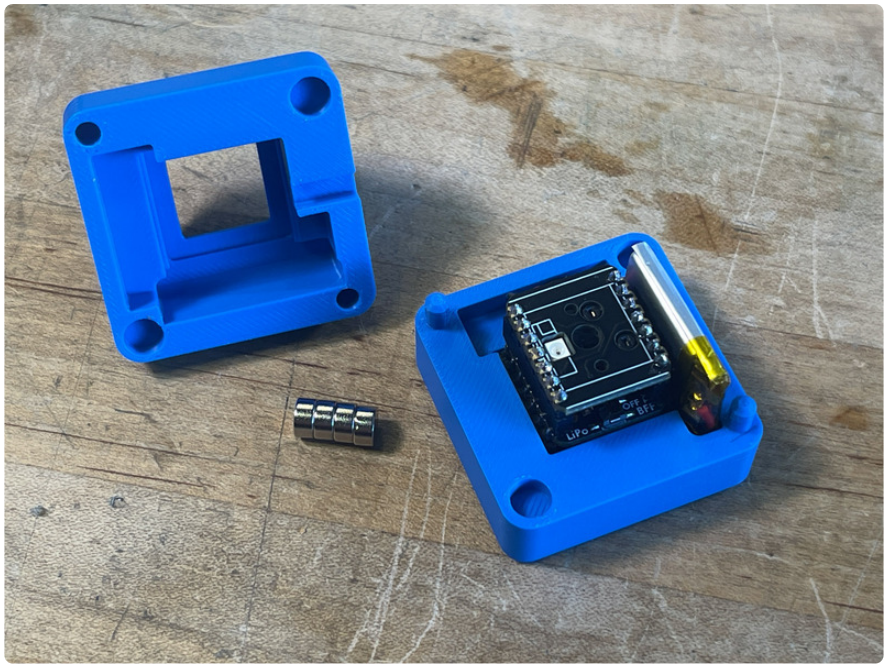
Press the stack into the base as shown with the USB C port aligned with the case cutout -- it should snap into place.

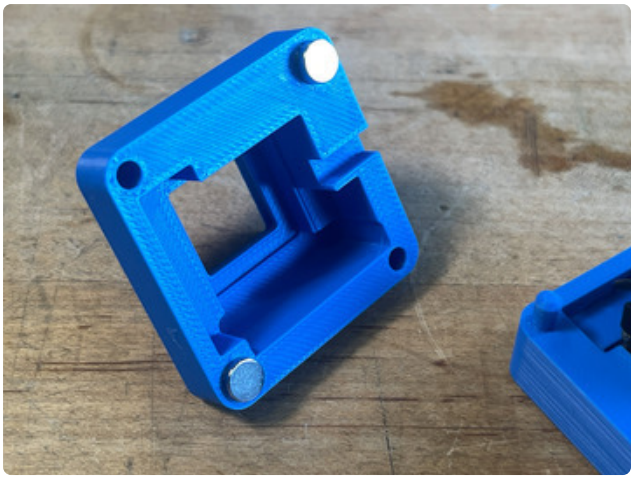
You can then tuck the wire and battery into their cozy little nook.

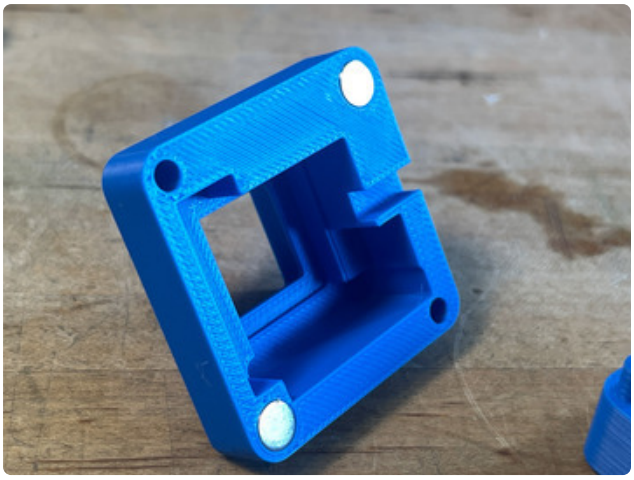


Magnets!

The case was designed to use two pairs of 6mm x 3mm magnets to keep it snapped closed. The posts on the opposing corners prevent it from shearing or rotating (these magnets are strongest on one axis), and in truth have tight enough tolerances that you may be able to skip the magnets. But magnets rule, so why would you want to!



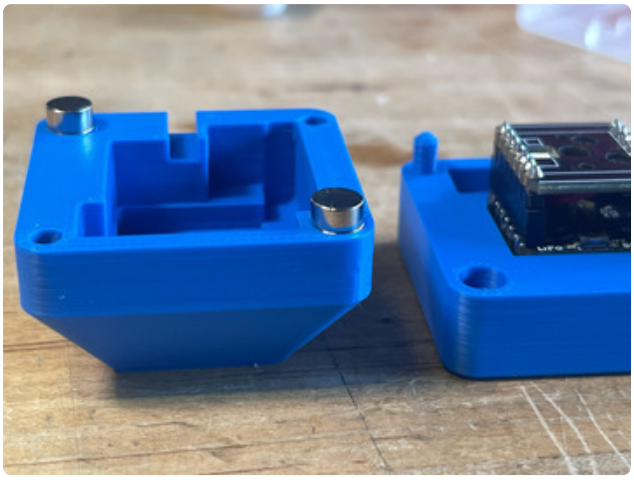




Press Fit

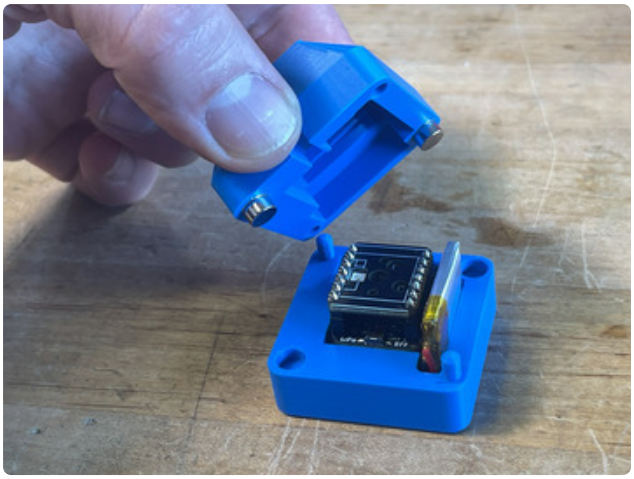
Place two magnets in the lid, either north or south up is fine for these two, they don't need to agree with each other, since we'll set the other two magnets on them to set them properly on the base. That said it's good practice to have them aligned the same so we don't accidentally put the base magnets in backwards later.

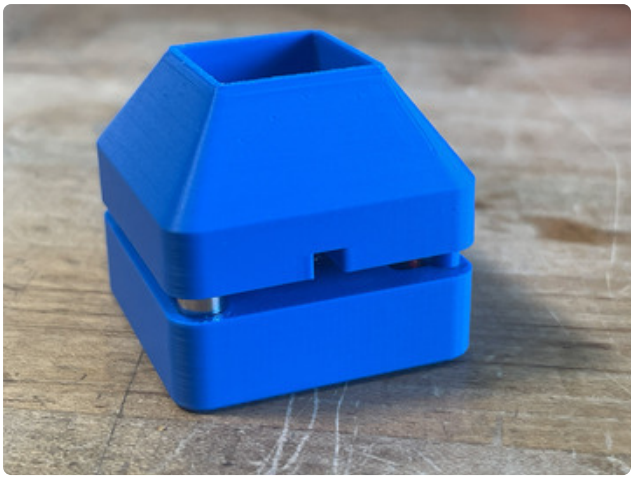
With the magnets partly seated, press them in by pushing the lid down on your work surface, with even pressure, until they are flush with the lid's flat bottom.

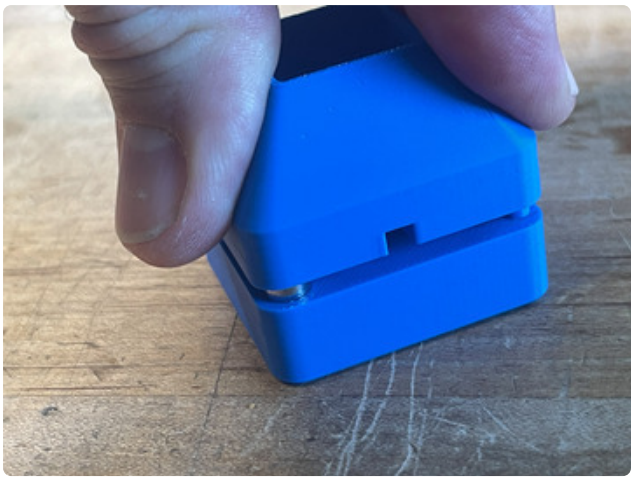


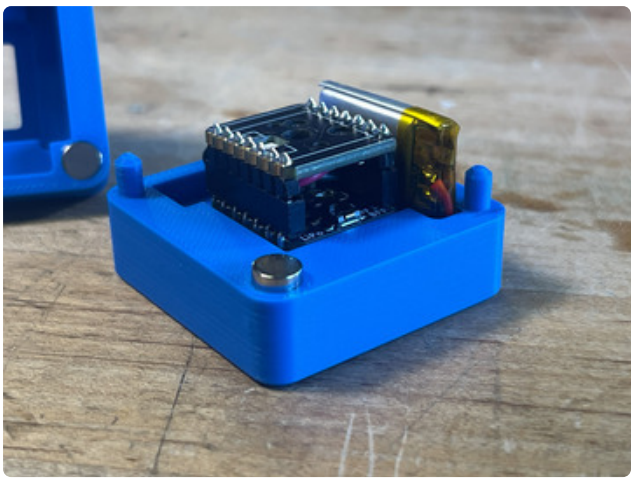
Prep Base Magnets

Set the two base magnets on the lid magnets. They will align automagically.





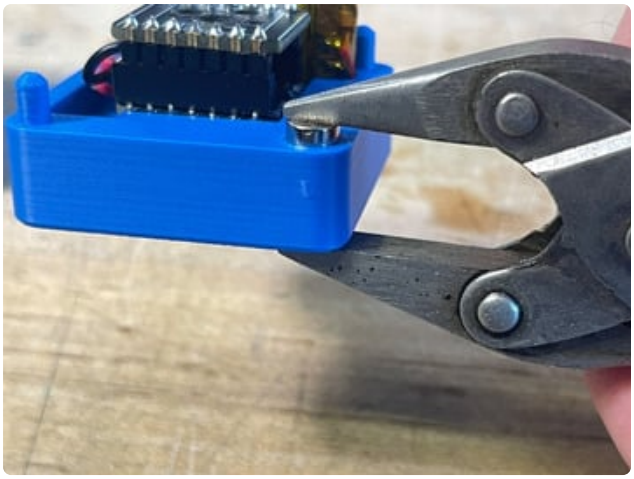


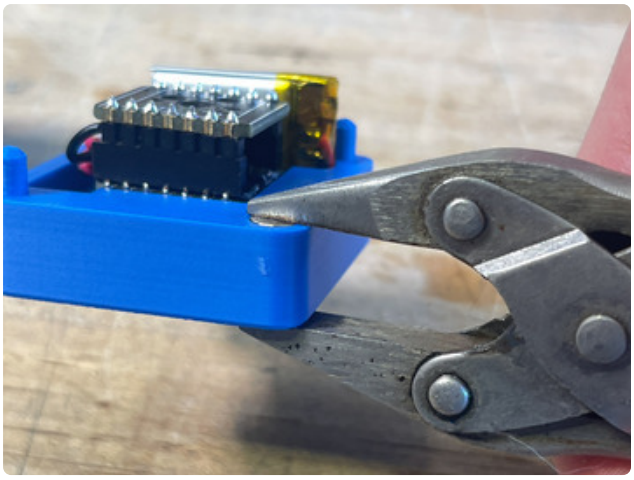


Push Into Base

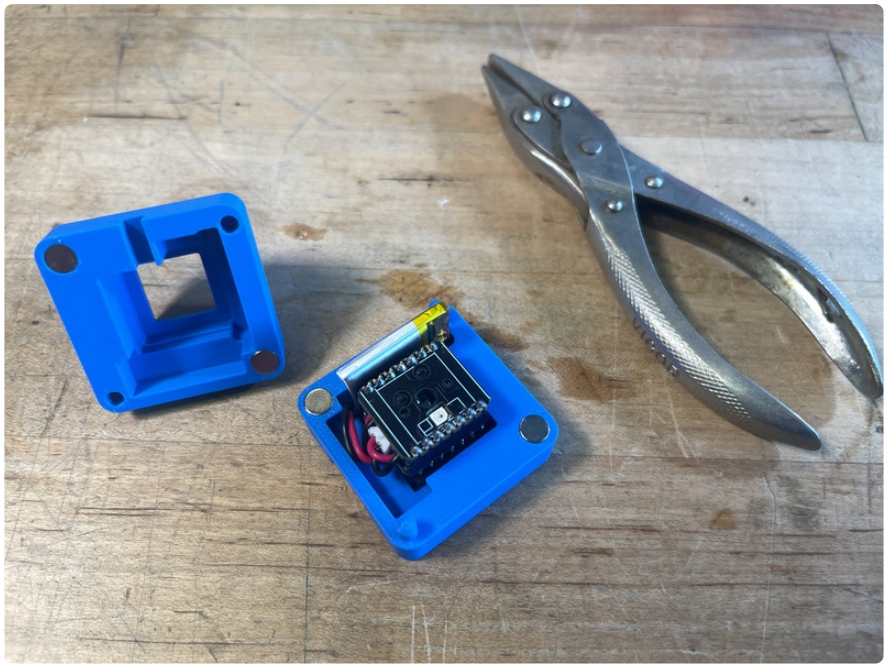
Now, set the lid over the base making sure the **power switch cutout of the lid is on the opposite side of the USB cutout of the base!**

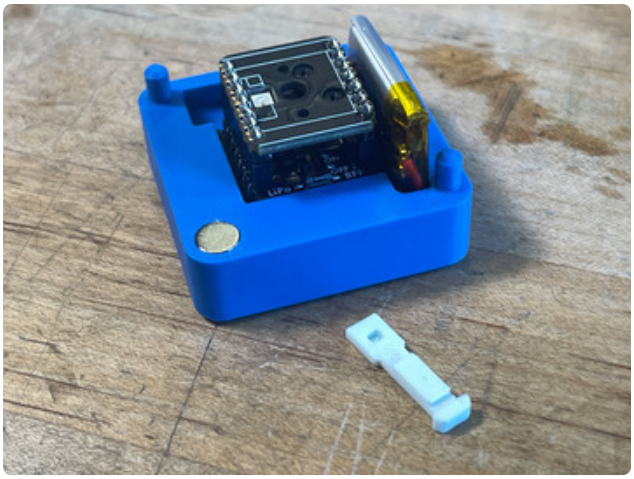
Push the lid down evenly and firmly to seat the base magnets. You can either press them down fully or use a set of parallel jaw pliers or pliers wrench as shown below.

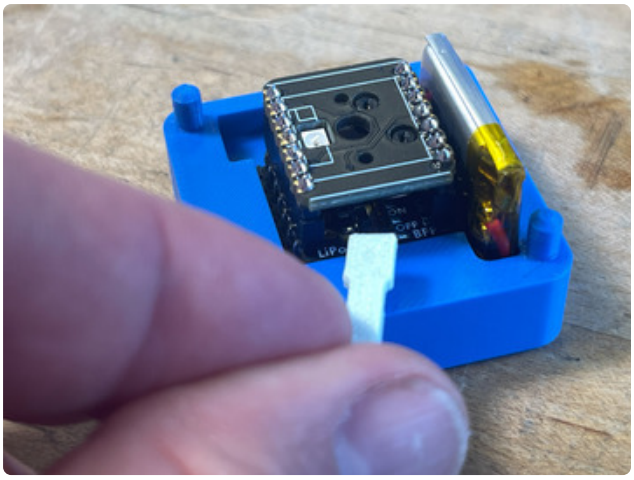




Optionally, squeeze the magnets in fully flush with parallel jaw pliers or a pliers wrench.



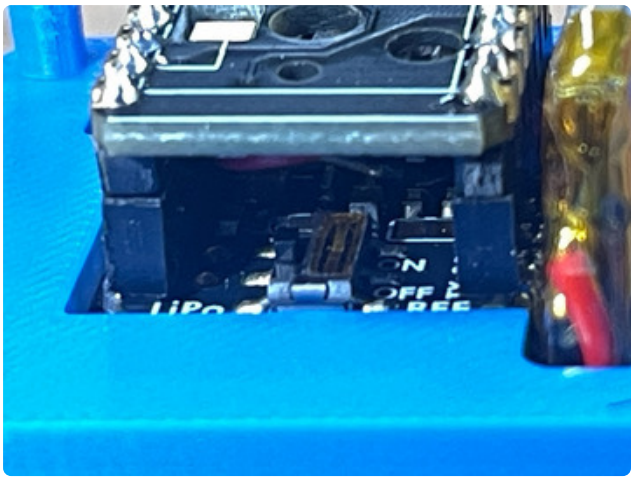


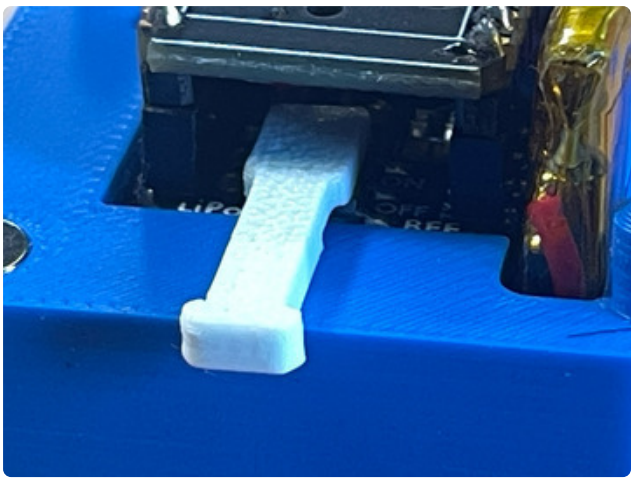


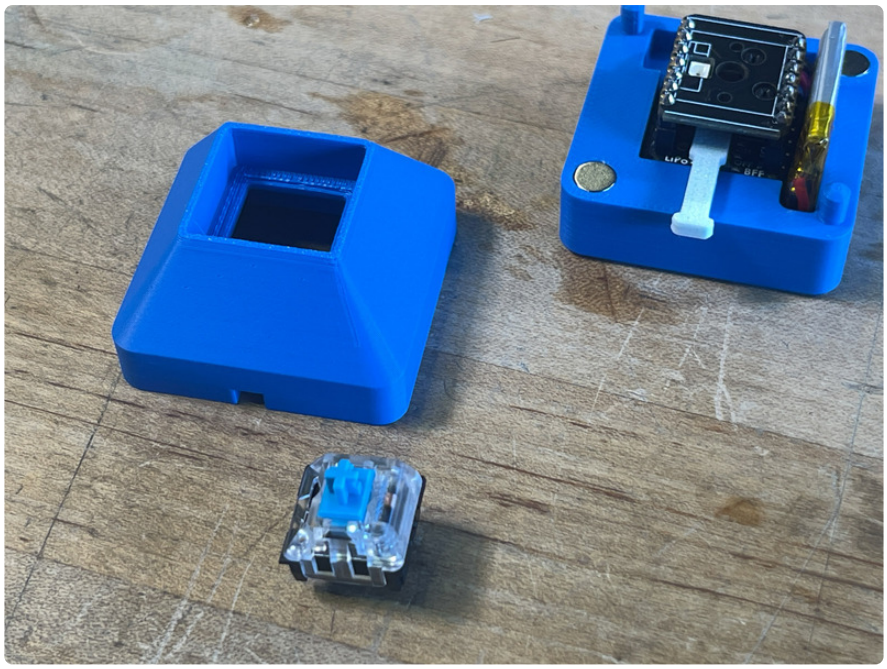
Power Switch

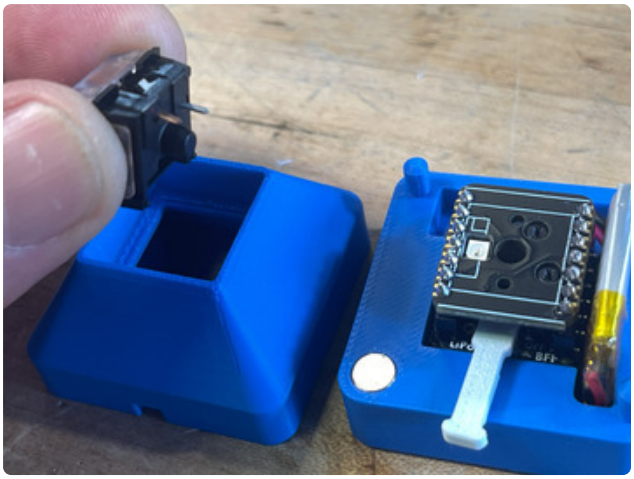
Extend the LiPoly BFF power switch using the 3D printed slide.

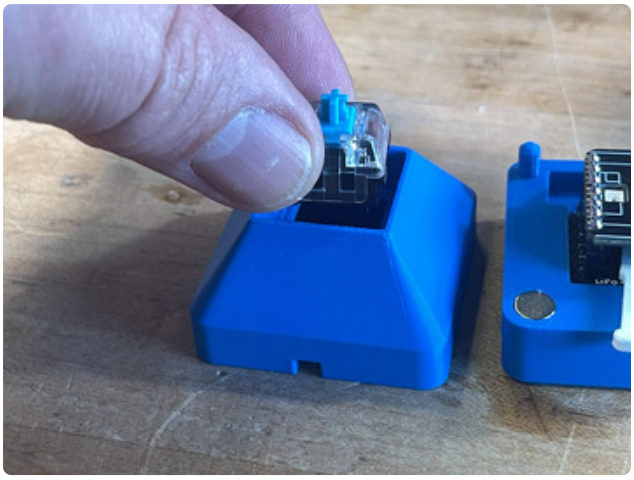
Align the cutout with the switch actuator as shown.





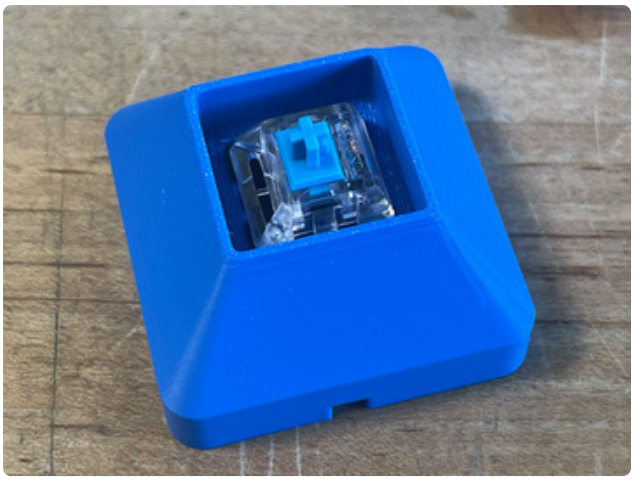


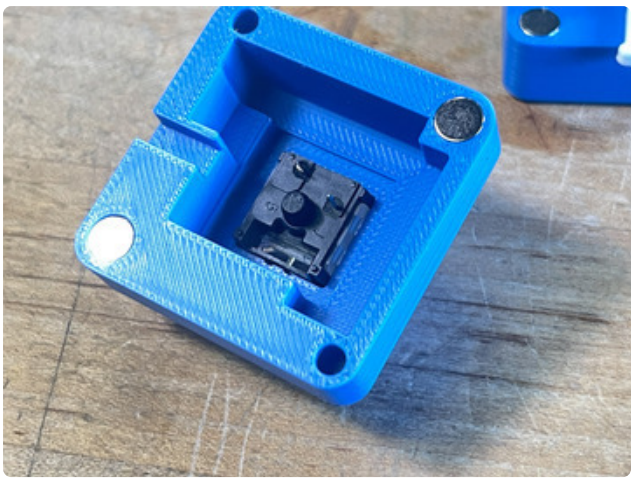


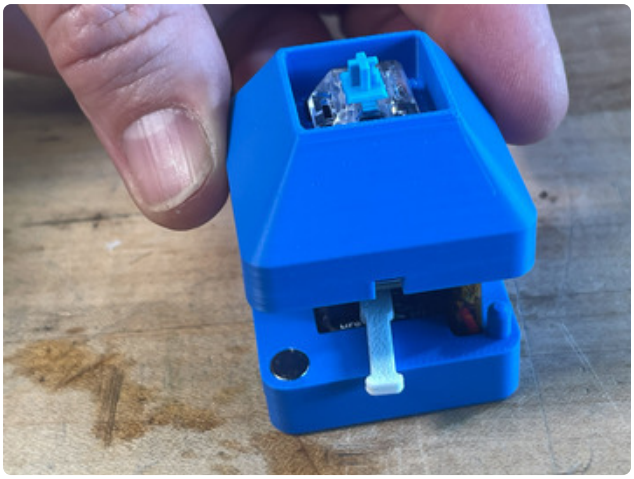


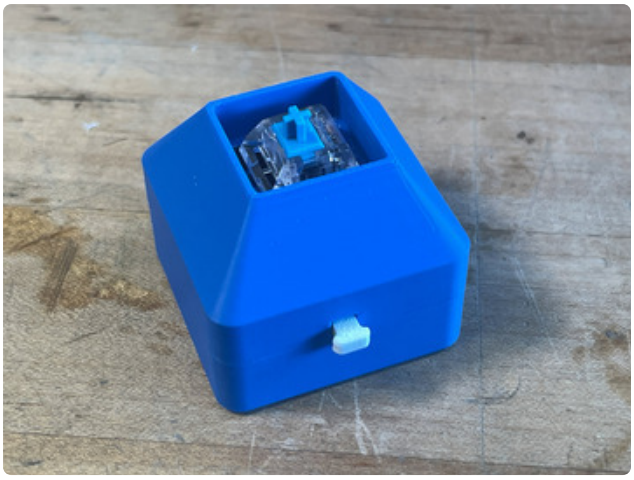
Keyswitch

With the top lid aligned with the base as shown, snap the keyswitch into the lid so the two legs will fit into the NeoKey BFF sockets.



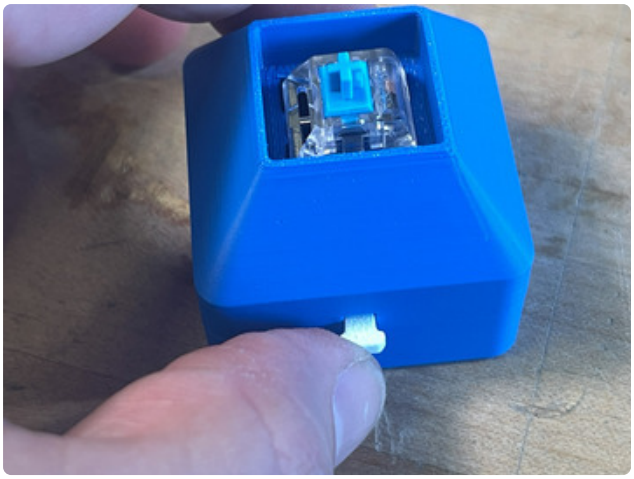


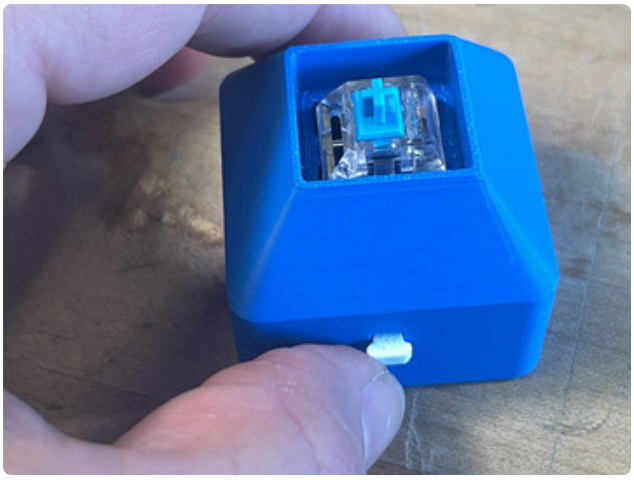




Close It Up

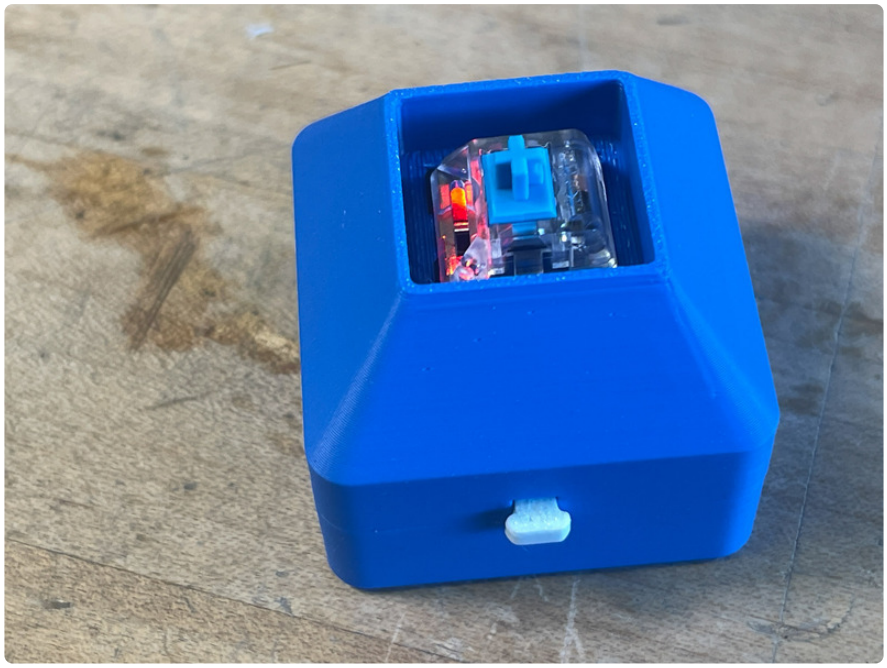
Set the lid onto the base, being sure the two keyswitch legs are aligned with the BFF. You'll also need to align the power switch actuator with the groove.

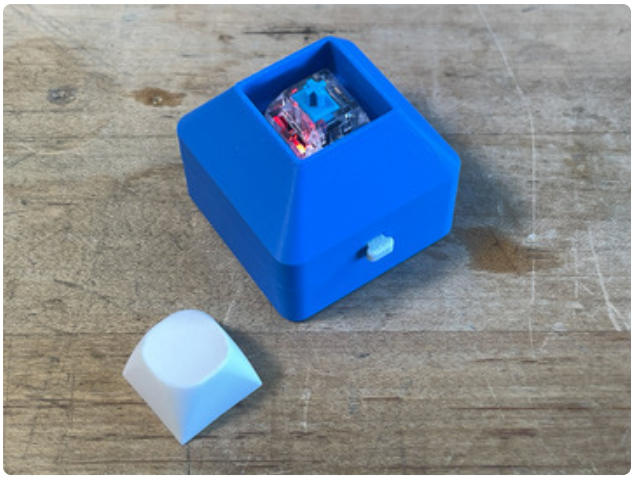


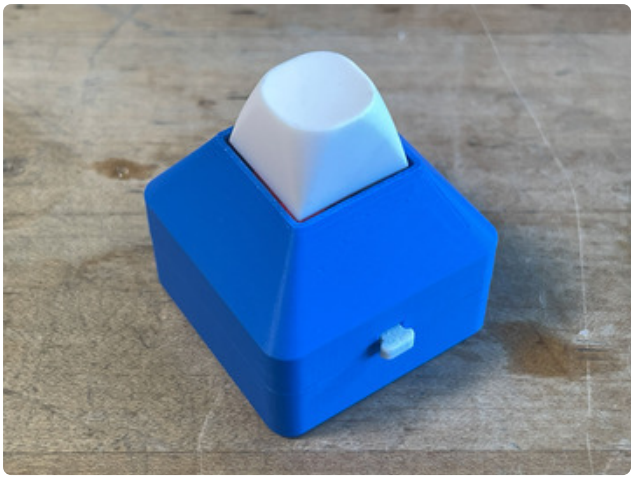


Test Power

Check that the power switch actuator can be pressed in to turn on the One Key, and pulled out to turn it off.



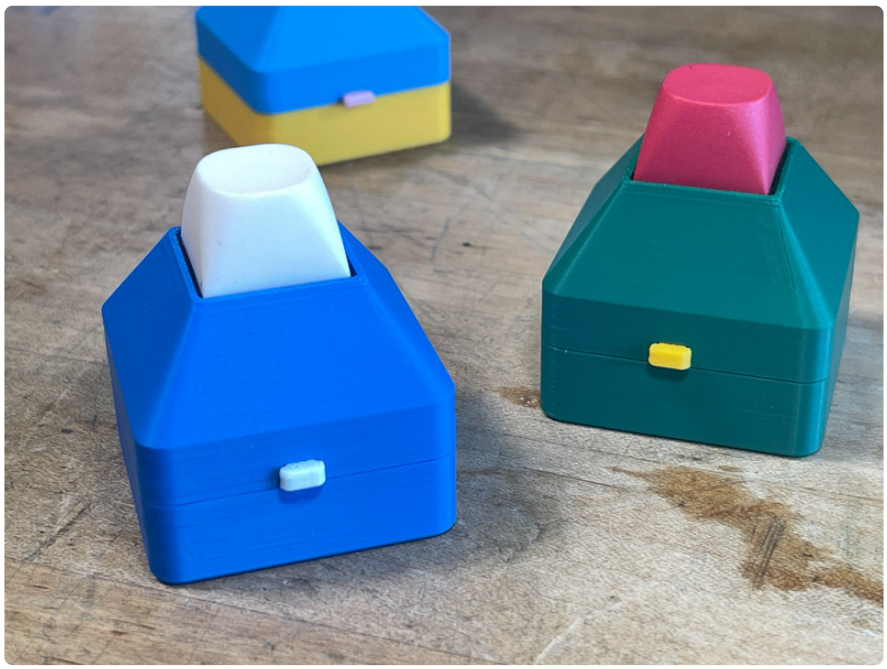




Keycap

Now, press your keycap into place!

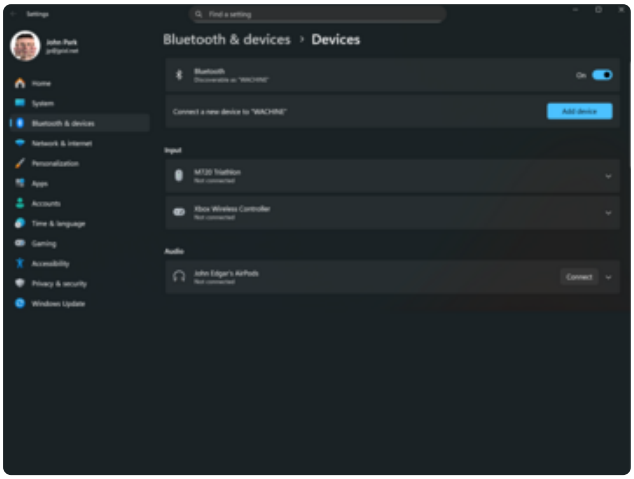
If you want, make a few in different colors for different functions!



Use the One Key

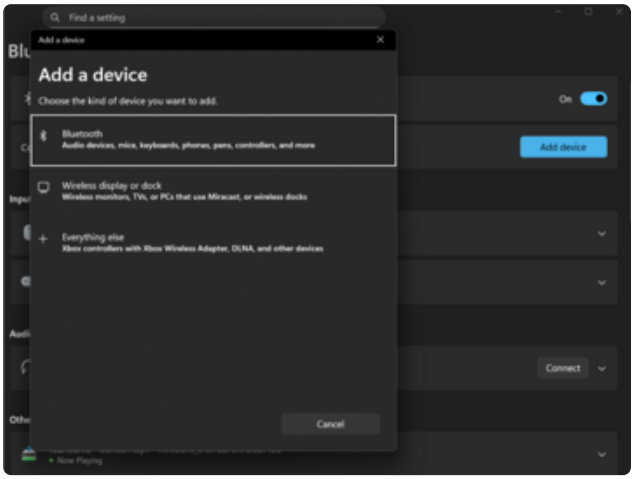
Bluetooth Pairing

You can pair your One Key with your computer or mobile device in the usual ways. Here's an example of pairing it on a Windows laptop.

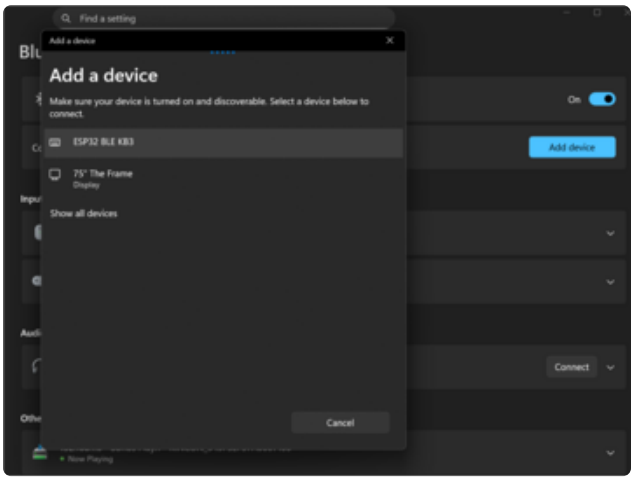


Go to **Windows** > **Settings** and pick **Bluetooth & Devices.**

Click **Add device.**

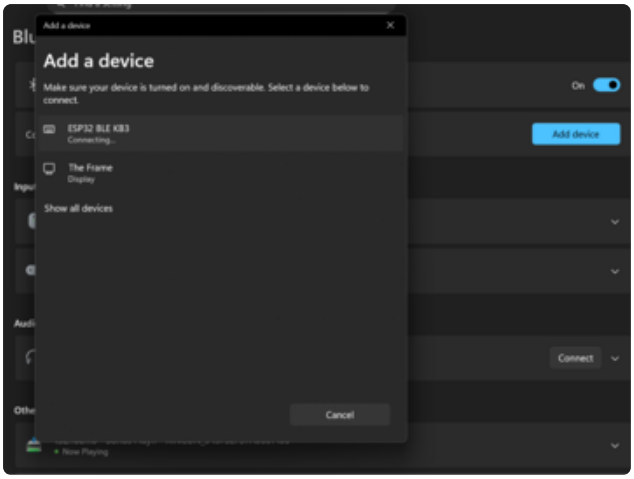


In the **Add a device** window, click on the **Bluetooth** option.

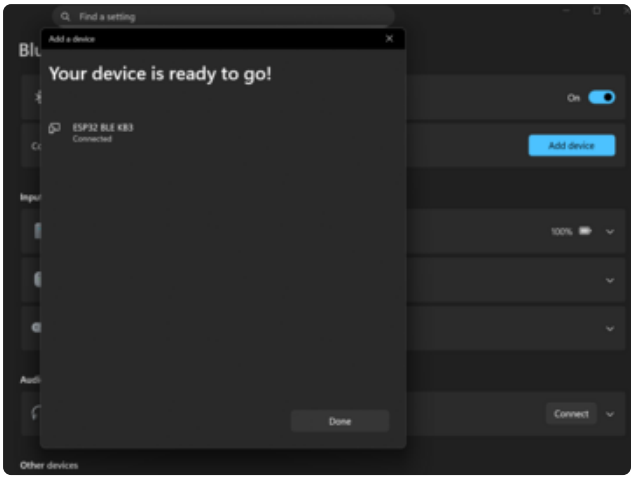


Turn on the One Key -- it will start up and the NeoPixels will display red, indicating it is unpaired.

The **ESP32 BLE KB3** device will show up in the list of available Bluetooth devices. Click it.



The devices will begin the connection secret handshake process.



The device is ready to go!

You'll see the One Key in the **Input** device list, and it'll auto-connect until you decide to remove it.

Serial Configuration

You will probably want to change features such as which key to press, NeoPixel color, and sleep timout at some point. Instead of editing the Arduino sketch, recompiling it, and uploading it to the One Key QT Py ESP32, you can use serial commands to do these things.

Configuration settings are stored in NVS (non-volatile storage in the ESP32's built-in flash) and can be changed via serial.

For example, with the One Key plugged in to your computer, open the Arduino Serial Monitor and type in `x` then press enter. The One Key will now send an 'x' key when pressed. This storage survives restarts and power cycling.

The One Key keyboard listens on the USB serial port at **115200 baud**. You can send commands from Arduino IDE's Serial Monitor, or any terminal app. Make sure line ending is set to **Newline** (not "No line ending" or "Both NL & CR").

Key commands Just type the key or combo and press Enter:

- `q` — single printable character
- `tab`, `return`, `esc`, `backspace`, `delete`, `space` : special keys
- `up`, `down`, `left`, `right`, `home`, `end`, `pageup`, `pagedown`, `insert` : navigation
- `f1` through `f24` : function keys
- `ctrl+z`, `alt+tab`, `shift+f5`, `ctrl+shift+z` : modifier combos (up to 2 modifiers)

String commands

- `"hello world"` — types a string
- `"git status"+return` — types a string then presses a key

Media key commands

- `volumeup`, `volumedown`, `mute`, `playpause`, `nexttrack`, `prevtrack`, `stop`

LED commands

- `color:connected:0,255,0` : set connected color as R,G,B (0–255 each)
- `color:pressed:255,20,100` : set pressed color as R,G,B
- `bright:128` : set brightness 0–255

Sleep command

- `sleep:5` : sleep after 5 minutes of inactivity
- `sleep:0` : disable sleep entirely

All settings are saved to NVS flash immediately and persist across power cycles. The device echoes a confirmation message after every successful command.

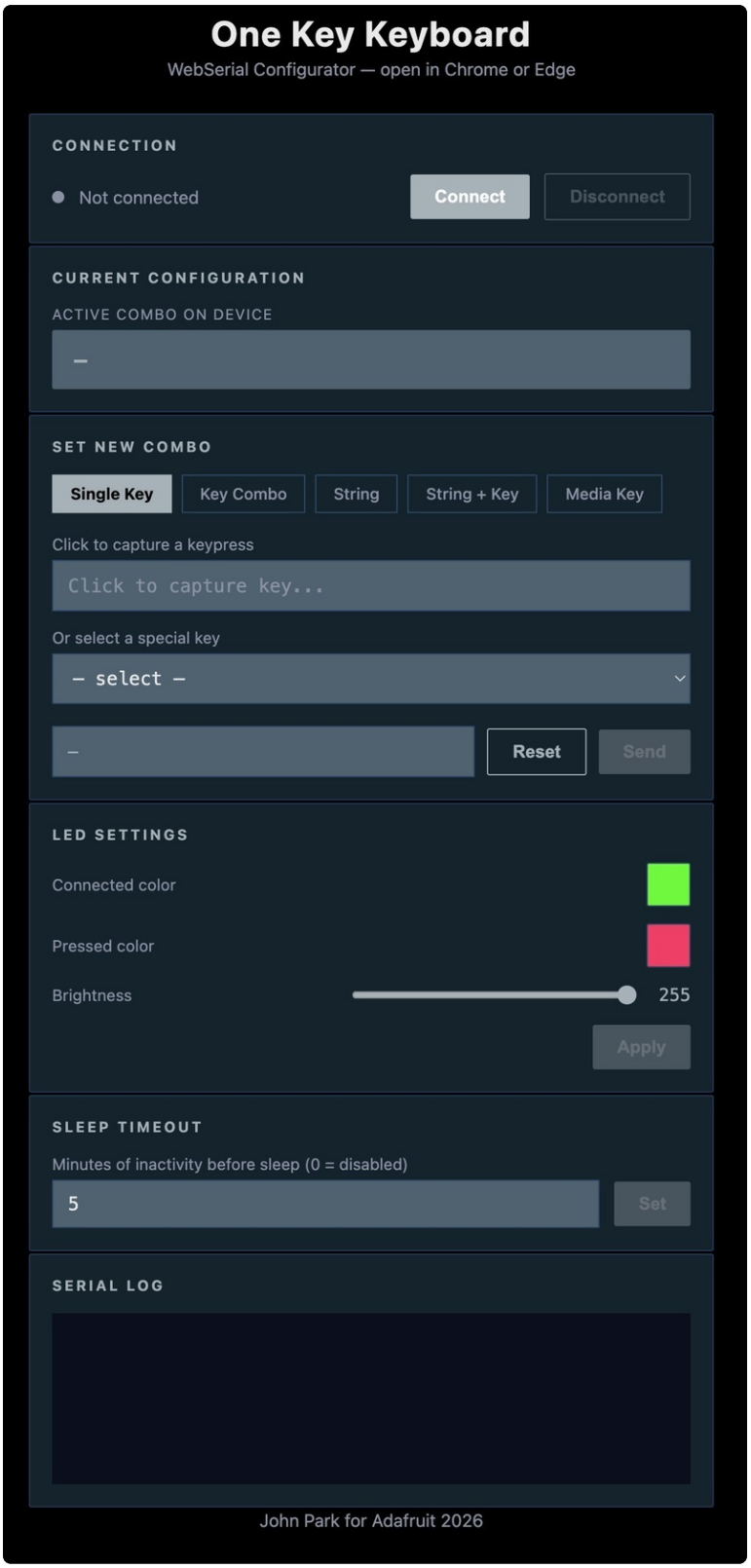
Battery life should be about 5 hours of continuous use (such as using it as a presentation clicker) if sleep mode isn't being entered. Intermittent use a times a day with sleep after 1 minute enabled should provide about two ks between charges. With no use but left in sleep mode you should get a few months of charge.

WebSerial

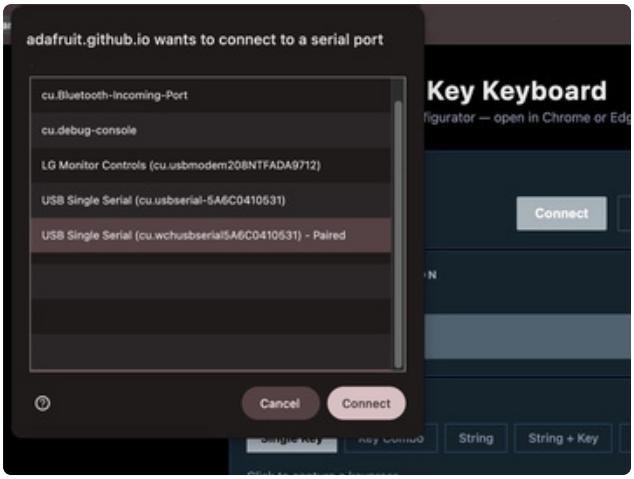
After adding all of these commands, I figured I'd need a cheat sheet to remember them all, and that bummed me out. What if, I thought, what if there were a GUI for configuring it? WebSerial to the rescue!

I created this WebSerial web page that you can access here (https://adafru.it/1aCK), or run locally with the included **index.html** file that acts as a front end for all of the above serial commands.

ure to run the page in a WebSerial capable browser such as Vivaldi or ome.



https://adafru.it/1aCK

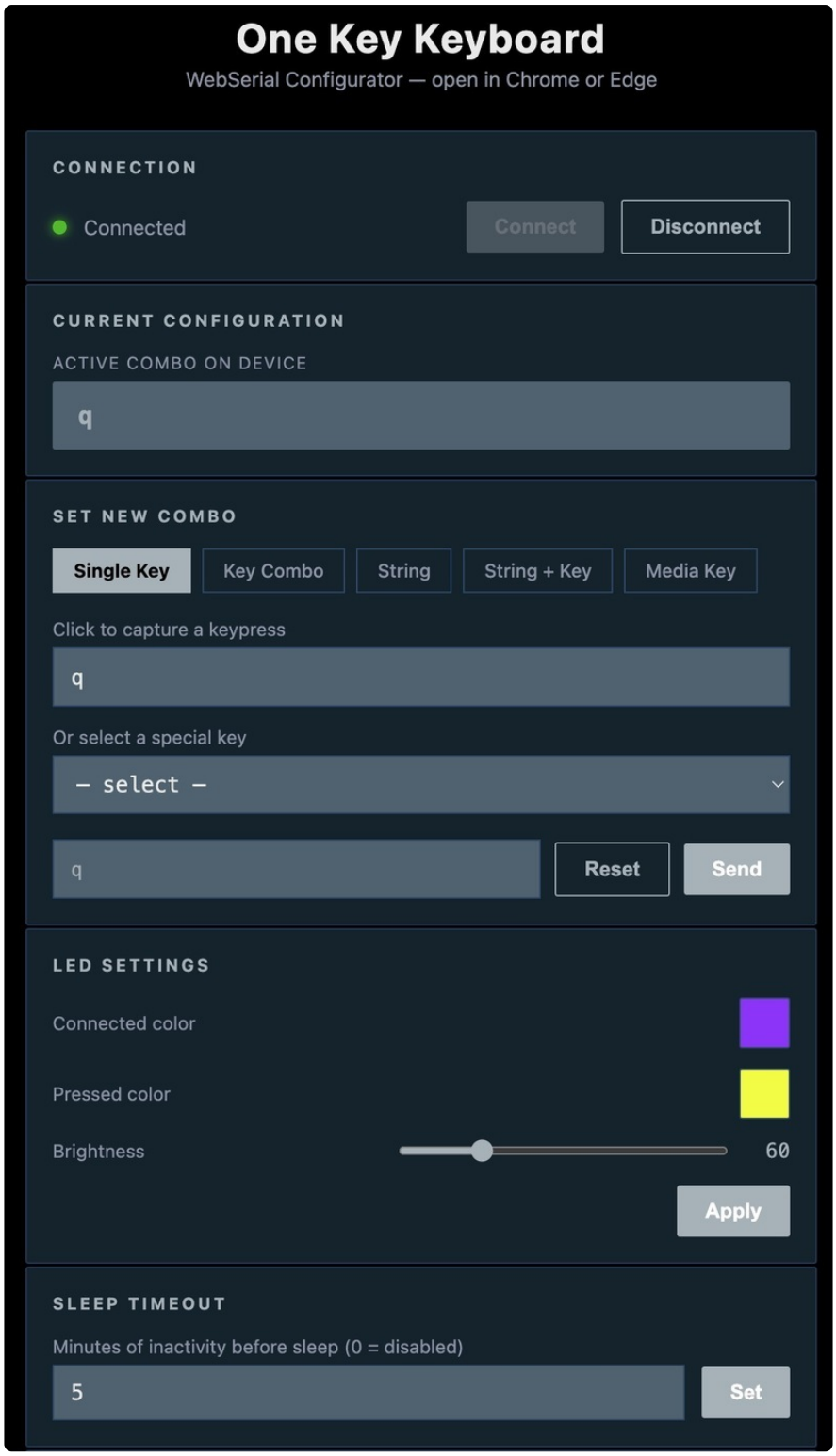


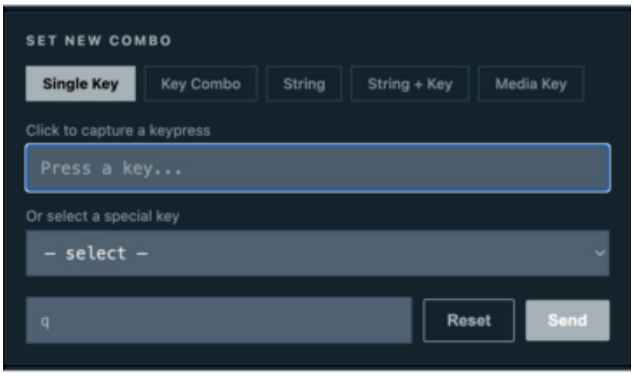
Connect

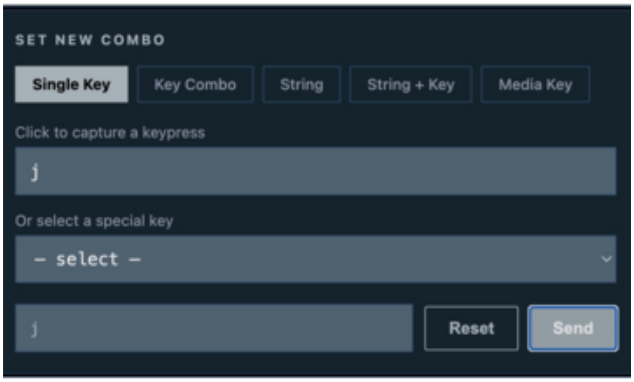
Plug in the One Key over USB, then click the **Connect** button.

In the pop-up window, select your serial port, then press the pop-up window's **Connect** button.

You'll see the "**Connected**" light turn green, and the current configuration should be filled in.


One Key Keyboard
WebSerial Configurator — open in Chrome or Edge
CONNECTION
Connected
Connect
Disconnect
CURRENT CONFIGURATION
ACTIVE COMBO ON DEVICE
q
SET NEW COMBO
Single Key
Key Combo
String
String + Key
Media Key
Click to capture a keypress
q
Or select a special key
— select —
q
Reset
Send
LED SETTINGS
Connected color
Pressed color
Brightness
60
Apply
SLEEP TIMEOUT
Minutes of inactivity before sleep (0 = disabled)
5
Set



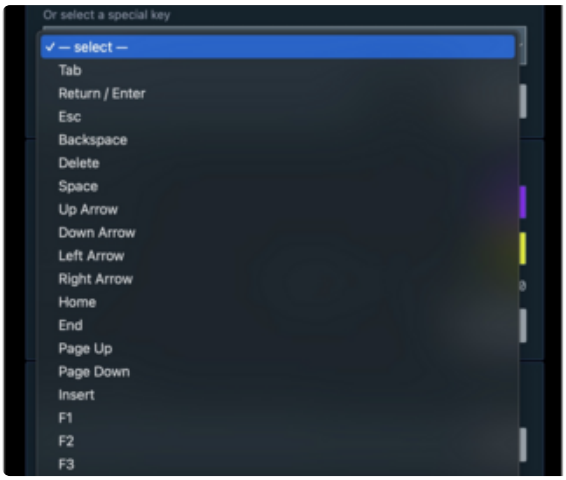


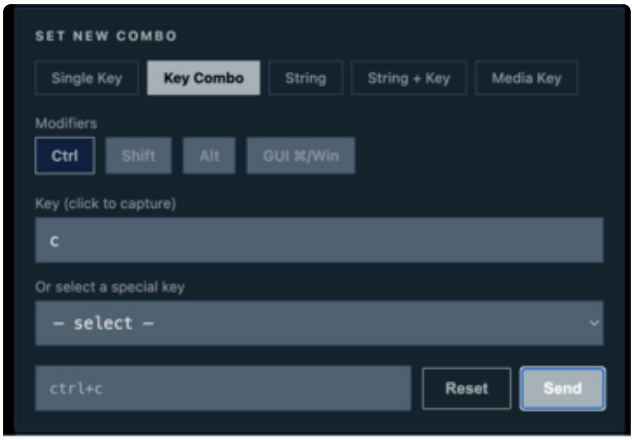
Single Key

To change a single key, click in the "Click to capture a keypress" box and type a key. Then click the **Send** button.

You can alternatively pick special keys it from the drop-down menu.

The One Key will now type your new single key.

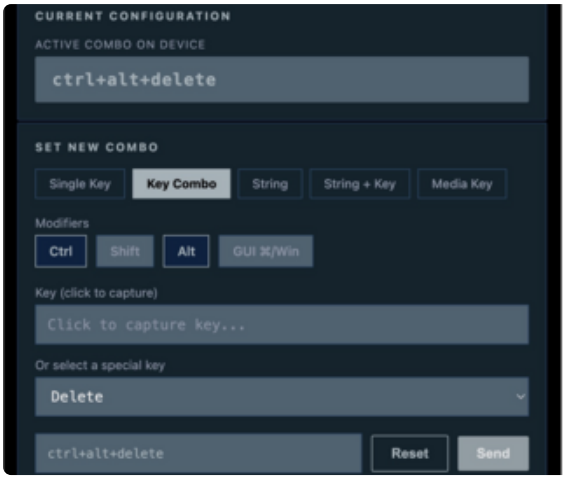


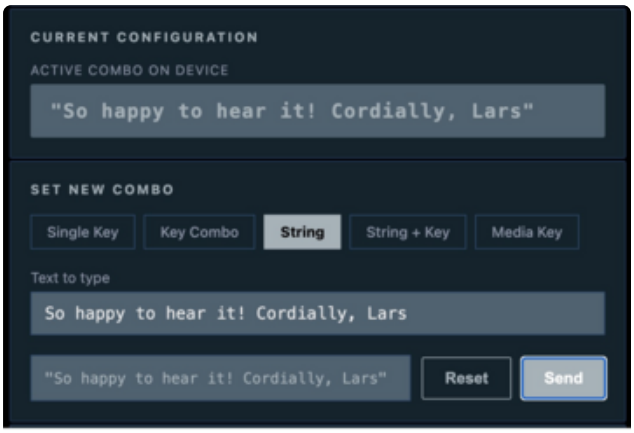


Key Combo

You can pick a key combo, such as `Ctrl-z / Command-z` for undo, or `Ctrl+Alt+Delete` by clicking the Key Combo button, and then picking modifiers (`Ctrl, Shift, Alt, GUI`) and a key or special key.

Then click **Send** to save it to the One Key.

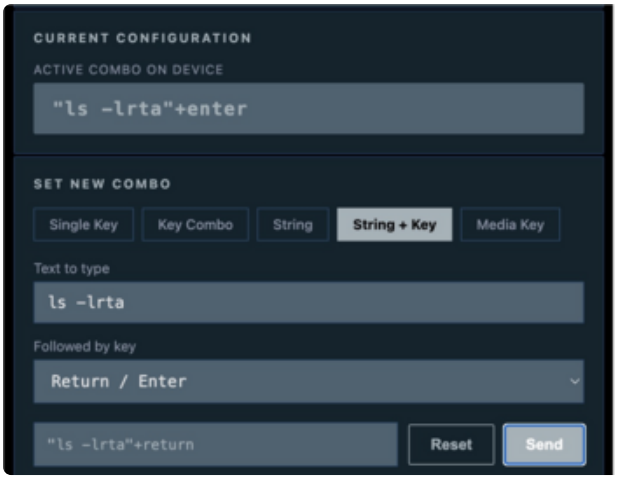




String

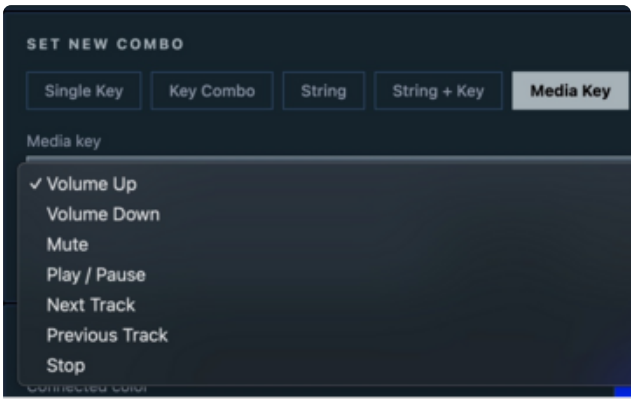
Have the One Key type out an entire message by using the **String** button and typing your characters (including alphabet, numerals, punctuation, symbols, and capitalization).

Click **Send** to save to the One Key.



String + Key

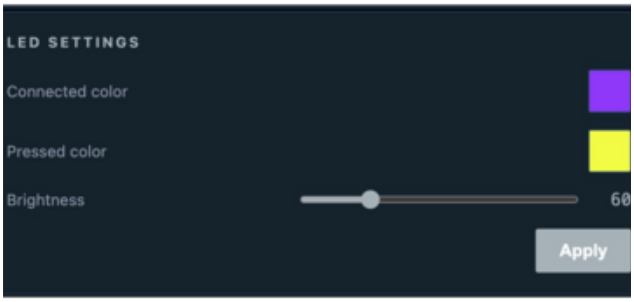
To have the One Key enter a full command, use **String** + **Key**. Then, type in your text and then a **Followed by key**, such as **Return / Enter**.

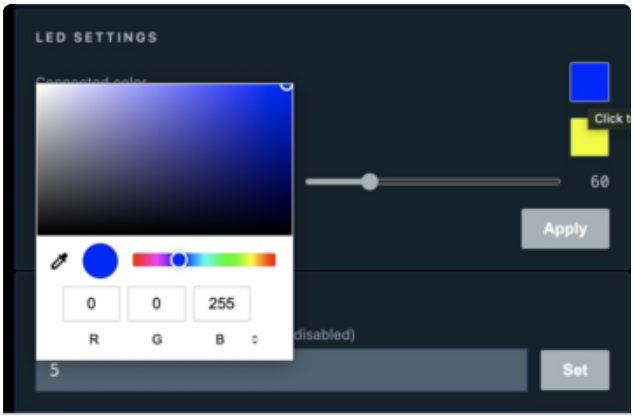


Media Keys

The Media Key section allows you to pick consumer controls:

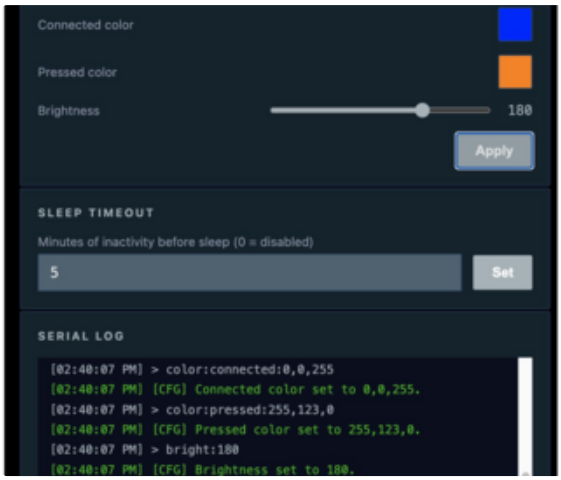
Volume Up
Volume Down
Mute
Play / Pause
Next Track
Previous Track
Stop

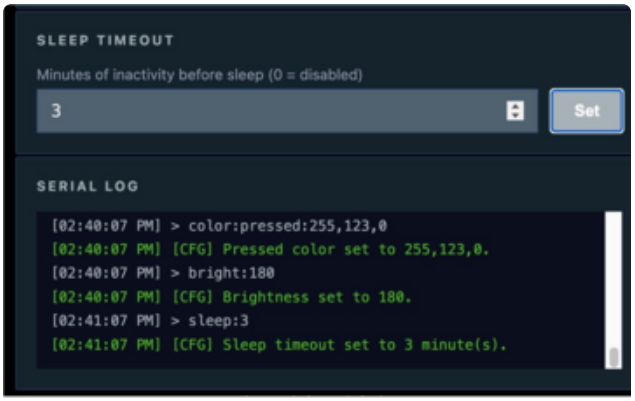




NeoPixel LED Settings

Adjust the **connected color** and **pressed color** as well as **brightness** for the NeoKey BFF NeoPixel and the QT PY ESP32 on-board NeoPixel in the **Led Settings** section. Press **Apply** when done.





Sleep Timeout

Change the number of minutes of inactivity before the One Key goes to sleep here. Using a setting of `0` will disable sleep entirely. Click **Set** to send to the One Key.

```
<!doctype html>
<!--
SPDX-FileCopyrightText: 2026 John Park for Adafruit Industries
SPDX-License-Identifier: MIT
-->
<html lang="en">
    <head>
        <meta charset="UTF-8" />
        <meta name="viewport" content="width=device-width, initial-scale=1.0" />
        <title>One Key Keyboard — Configurator</title>
        <style>
            :root {
                --bg: #000000;
                --surface: #12232e;
                --card: #4c6271;
                --accent: #a5b1b8;
                --accent2: #a5b1b8;
                --text: #eaeaea;
                --muted: #8892a4;
                --green: #00bf00;
                --red: #e94560;
                --radius: 2px;
                --font: "Segoe UI", system-ui, sans-serif;
            }
            * {
                box-sizing: border-box;
                margin: 0;
                padding: 0;
            }
            body {
                background: var(--bg);
                color: var(--text);
                font-family: var(--font);
                min-height: 100vh;
                display: flex;
                flex-direction: column;
                align-items: center;
                padding: 32px 16px;
            }
            h1 {
                font-size: 1.8rem;
                font-weight: 700;
                letter-spacing: 1px;
                margin-bottom: 4px;
            }
            h1 span {
                color: var(--accent);
            }
            .subtitle {
                color: var(--muted);
```

```
            font-size: 0.9rem;
            margin-bottom: 28px;
        }
        .card {
            background: var(--surface);
            border: 1px solid #1e3a5f;
            border-radius: var(--radius);
            padding: 18px;
            width: 100%;
            max-width: 560px;
            margin-bottom: 2px;
        }
        .card h2 {
            font-size: 0.75rem;
            text-transform: uppercase;
            letter-spacing: 2px;
            color: var(--accent2);
            margin-bottom: 16px;
        }
        .connect-row {
            display: flex;
            align-items: center;
            gap: 12px;
        }
        .dot {
            width: 10px;
            height: 10px;
            border-radius: 50%;
            background: var(--muted);
            flex-shrink: 0;
            transition: background 0.3s;
        }
        .dot.connected {
            background: var(--green);
            box-shadow: 0 0 8px var(--green);
        }
        .dot.error {
            background: var(--red);
        }
        #status-text {
            color: var(--muted);
            font-size: 0.9rem;
            flex: 1;
        }
        button {
            background: var(--accent);
            color: #fff;
            border: none;
            border-radius: 2px;
            padding: 10px 20px;
            font-size: 0.9rem;
            font-weight: 600;
            cursor: pointer;
            transition: opacity 0.2s;
            white-space: nowrap;
        }
        button:hover {
            opacity: 0.85;
        }
        button:disabled {
            opacity: 0.35;
            cursor: not-allowed;
        }
        button.secondary {
            background: transparent;
            border: 1px solid var(--accent2);
            color: var(--accent2);
        }
        .current-combo {
```

```
            background: var(--card);
            border-radius: 2px;
            padding: 14px 18px;
            font-size: 1.1rem;
            font-weight: 600;
            letter-spacing: 1px;
            color: var(--accent2);
            font-family: monospace;
            min-height: 46px;
        }
        .current-label {
            font-size: 0.75rem;
            color: var(--muted);
            margin-bottom: 6px;
            text-transform: uppercase;
            letter-spacing: 1px;
        }
        .tabs {
            display: flex;
            gap: 8px;
            margin-bottom: 18px;
            flex-wrap: wrap;
        }
        .tab {
            background: transparent;
            border: 1px solid #2a4a6e;
            color: var(--muted);
            padding: 7px 14px;
            font-size: 0.82rem;
            border-radius: 0;
            cursor: pointer;
            transition: all 0.2s;
            font-weight: 500;
        }
        .tab:hover {
            border-color: var(--accent2);
            color: var(--accent2);
        }
        .tab.active {
            background: var(--accent2);
            border-color: var(--accent2);
            color: #000;
            font-weight: 700;
        }
        .mode-panel {
            display: none;
            flex-direction: column;
            gap: 14px;
        }
        .mode-panel.active {
            display: flex;
        }
        label {
            font-size: 0.82rem;
            color: var(--muted);
            display: block;
            margin-bottom: 5px;
        }
        input[type="text"],
        select {
            width: 100%;
            background: var(--card);
            border: 1px solid #2a4a6e;
            border-radius: 0;
            color: var(--text);
            padding: 10px 12px;
            font-size: 0.95rem;
            font-family: monospace;
            outline: none;
```

```
            transition: border-color 0.2s;
        }
        input[type="text"]:focus,
        select:focus {
            border-color: var(--accent2);
        }
        select option {
            background: var(--card);
        }
        .modifiers {
            display: flex;
            gap: 10px;
            flex-wrap: wrap;
        }
        .mod-check {
            display: flex;
            align-items: center;
            gap: 6px;
            background: var(--card);
            border: 1px solid #2a4a6e;
            border-radius: 0;
            padding: 8px 14px;
            cursor: pointer;
            font-size: 0.85rem;
            font-weight: 600;
            transition: all 0.2s;
            user-select: none;
        }
        .mod-check input {
            display: none;
        }
        .mod-check.checked {
            border-color: var(--accent2);
            color: var(--accent2);
            background: #0a2040;
        }
        .key-capture {
            background: var(--card);
            border: 1px solid #2a4a6e;
            border-radius: 0;
            padding: 10px 12px;
            font-size: 0.95rem;
            font-family: monospace;
            cursor: pointer;
            min-height: 42px;
            display: flex;
            align-items: center;
            transition: border-color 0.2s;
            color: var(--muted);
        }
        .key-capture.listening {
            border-color: var(--accent);
            color: var(--accent);
            animation: pulse 1s infinite;
        }
        .key-capture.has-key {
            color: var(--text);
        }
        @keyframes pulse {
            0%,
            100% {
                opacity: 1;
            }
            50% {
                opacity: 0.5;
            }
        }
        .send-row {
            display: flex;
```

```
            gap: 10px;
            align-items: center;
            margin-top: 4px;
        }
        .preview {
            flex: 1;
            font-family: monospace;
            font-size: 0.85rem;
            color: var(--muted);
            background: var(--card);
            border-radius: 0;
            padding: 10px 12px;
            border: 1px solid #2a4a6e;
            min-height: 42px;
            display: flex;
            align-items: center;
            word-break: break-all;
        }
        .preview span {
            color: var(--accent2);
        }
        #log {
            background: #0a0f1e;
            border-radius: 0;
            padding: 12px;
            font-family: monospace;
            font-size: 0.8rem;
            height: 140px;
            overflow-y: auto;
            display: flex;
            flex-direction: column;
            gap: 2px;
        }
        .log-line {
            line-height: 1.5;
        }
        .log-line.tx {
            color: var(--accent2);
        }
        .log-line.rx {
            color: var(--green);
        }
        .log-line.info {
            color: var(--muted);
        }
        .log-line.err {
            color: var(--red);
        }
        .not-supported {
            background: #3a1010;
            border: 1px solid var(--red);
            border-radius: var(--radius);
            padding: 16px 20px;
            color: var(--red);
            font-size: 0.9rem;
            max-width: 560px;
            width: 100%;
            margin-bottom: 20px;
            display: none;
        }
        /* LED settings */
        .led-row {
            display: flex;
            align-items: center;
            gap: 12px;
            margin-bottom: 14px;
        }
        .led-row:last-child {
            margin-bottom: 0;
```

```
            }
            .led-row label {
                margin-bottom: 0;
                flex: 1;
            }
            .color-swatch {
                width: 36px;
                height: 36px;
                border-radius: 2px;
                border: 1px solid #2a4a6e;
                flex-shrink: 0;
                cursor: pointer;
                transition: border-color 0.2s;
            }
            .color-swatch:hover {
                border-color: var(--accent2);
            }
            input[type="color"] {
                width: 0;
                height: 0;
                opacity: 0;
                position: absolute;
                pointer-events: none;
            }
            input[type="range"] {
                flex: 1;
                accent-color: var(--accent2);
                height: 6px;
                cursor: pointer;
            }
            .bright-val {
                font-family: monospace;
                font-size: 0.9rem;
                color: var(--accent2);
                width: 32px;
                text-align: right;
                flex-shrink: 0;
            }
            .led-send-row {
                display: flex;
                justify-content: flex-end;
                margin-top: 16px;
            }
        </style>
    </head>
    <body>
        <h1>One Key Keyboard</h1>
        <p class="subtitle">WebSerial Configurator — open in Chrome or Edge</p>

        <div class="not-supported" id="not-supported">
            ⚠ WebSerial is not supported in this browser. Please open this file
            in Chrome or Edge.
        </div>

        <!-- Connection -->
        <div class="card">
            <h2>Connection</h2>
            <div class="connect-row">
                <div class="dot" id="dot"></div>
                <span id="status-text">Not connected</span>
                <button id="connect-btn">Connect</button>
                <button class="secondary" id="disconnect-btn" disabled>
                    Disconnect
                </button>
            </div>
        </div>

        <!-- Current config -->
        <div class="card">
```

```
            <h2>Current Configuration</h2>
            <div class="current-label">Active combo on device</div>
            <div class="current-combo" id="current-combo">—</div>
        </div>

        <!-- Config editor -->
        <div class="card">
            <h2>Set New Combo</h2>
            <div class="tabs">
                <div class="tab active" data-mode="single">Single Key</div>
                <div class="tab" data-mode="combo">Key Combo</div>
                <div class="tab" data-mode="string">String</div>
                <div class="tab" data-mode="string-key">String + Key</div>
                <div class="tab" data-mode="media">Media Key</div>
            </div>

            <!-- Single key -->
            <div class="mode-panel active" id="panel-single">
                <div>
                    <label>Click to capture a keypress</label>
                    <div class="key-capture" id="capture-single" tabindex="0">
                        Click to capture key...
                    </div>
                </div>
                <div>
                    <label>Or select a special key</label>
                    <select id="special-single">
                        <option value="">— select —</option>
                        <option value="tab">Tab</option>
                        <option value="return">Return / Enter</option>
                        <option value="esc">Esc</option>
                        <option value="backspace">Backspace</option>
                        <option value="delete">Delete</option>
                        <option value="space">Space</option>
                        <option value="up">Up Arrow</option>
                        <option value="down">Down Arrow</option>
                        <option value="left">Left Arrow</option>
                        <option value="right">Right Arrow</option>
                        <option value="home">Home</option>
                        <option value="end">End</option>
                        <option value="pageup">Page Up</option>
                        <option value="pagedown">Page Down</option>
                        <option value="insert">Insert</option>
                        <option value="f1">F1</option>
                        <option value="f2">F2</option>
                        <option value="f3">F3</option>
                        <option value="f4">F4</option>
                        <option value="f5">F5</option>
                        <option value="f6">F6</option>
                        <option value="f7">F7</option>
                        <option value="f8">F8</option>
                        <option value="f9">F9</option>
                        <option value="f10">F10</option>
                        <option value="f11">F11</option>
                        <option value="f12">F12</option>
                    </select>
                </div>
            </div>

            <!-- Combo -->
            <div class="mode-panel" id="panel-combo">
                <div>
                    <label>Modifiers</label>
                    <div class="modifiers">
                        <label class="mod-check" id="mod-ctrl"
                            ><input type="checkbox" value="ctrl" />Ctrl</label
                        >
                        <label class="mod-check" id="mod-shift"
                            ><input type="checkbox" value="shift" />Shift</label
```

```
                    >
                    <label class="mod-check" id="mod-alt"
                        ><input type="checkbox" value="alt" />Alt</label
                    >
                    <label class="mod-check" id="mod-gui"
                        ><input type="checkbox" value="gui" />GUI
                        ⌘/Win</label
                    >
                </div>
            </div>
            <div>
                <label>Key (click to capture)</label>
                <div class="key-capture" id="capture-combo" tabindex="0">
                    Click to capture key...
                </div>
            </div>
            <div>
                <label>Or select a special key</label>
                <select id="special-combo">
                    <option value="">— select —</option>
                    <option value="tab">Tab</option>
                    <option value="return">Return / Enter</option>
                    <option value="esc">Esc</option>
                    <option value="backspace">Backspace</option>
                    <option value="delete">Delete</option>
                    <option value="space">Space</option>
                    <option value="up">Up Arrow</option>
                    <option value="down">Down Arrow</option>
                    <option value="left">Left Arrow</option>
                    <option value="right">Right Arrow</option>
                    <option value="home">Home</option>
                    <option value="end">End</option>
                    <option value="pageup">Page Up</option>
                    <option value="pagedown">Page Down</option>
                    <option value="insert">Insert</option>
                    <option value="f1">F1</option>
                    <option value="f2">F2</option>
                    <option value="f3">F3</option>
                    <option value="f4">F4</option>
                    <option value="f5">F5</option>
                    <option value="f6">F6</option>
                    <option value="f7">F7</option>
                    <option value="f8">F8</option>
                    <option value="f9">F9</option>
                    <option value="f10">F10</option>
                    <option value="f11">F11</option>
                    <option value="f12">F12</option>
                </select>
            </div>
        </div>

        <!-- String -->
        <div class="mode-panel" id="panel-string">
            <div>
                <label>Text to type</label>
                <input
                    type="text"
                    id="string-input"
                    placeholder="e.g. hello world"
                />
            </div>
        </div>

        <!-- String + key -->
        <div class="mode-panel" id="panel-string-key">
            <div>
                <label>Text to type</label>
                <input
                    type="text"
```

```
                    id="string-key-input"
                    placeholder="e.g. git status"
                />
            </div>
            <div>
                <label>Followed by key</label>
                <select id="tail-key">
                    <option value="return">Return / Enter</option>
                    <option value="tab">Tab</option>
                    <option value="esc">Esc</option>
                    <option value="space">Space</option>
                    <option value="f1">F1</option>
                    <option value="f2">F2</option>
                    <option value="f3">F3</option>
                    <option value="f4">F4</option>
                    <option value="f5">F5</option>
                </select>
            </div>
        </div>

        <!-- Media key -->
        <div class="mode-panel" id="panel-media">
            <div>
                <label>Media key</label>
                <select id="media-key">
                    <option value="volumeup">Volume Up</option>
                    <option value="volumedown">Volume Down</option>
                    <option value="mute">Mute</option>
                    <option value="playpause">Play / Pause</option>
                    <option value="nexttrack">Next Track</option>
                    <option value="prevtrack">Previous Track</option>
                    <option value="stop">Stop</option>
                </select>
            </div>
        </div>

        <!-- Preview + Send -->
        <div class="send-row" style="margin-top: 18px">
            <div class="preview" id="preview"><span>—</span></div>
            <button class="secondary" id="reset-btn">Reset</button>
            <button id="send-btn" disabled>Send</button>
        </div>
    </div>

    <!-- LED Settings -->
    <div class="card">
        <h2>LED Settings</h2>
        <!-- Connected color -->
        <div class="led-row">
            <label>Connected color</label>
            <div
                class="color-swatch"
                id="swatch-connected"
                style="background: #00ff00"
                title="Click to change"
            ></div>
            <input type="color" id="picker-connected" value="#00ff00" />
        </div>
        <!-- Pressed color -->
        <div class="led-row">
            <label>Pressed color</label>
            <div
                class="color-swatch"
                id="swatch-pressed"
                style="background: #ff1464"
                title="Click to change"
            ></div>
            <input type="color" id="picker-pressed" value="#ff1464" />
        </div>
```

```
            <!-- Brightness -->
            <div class="led-row">
                <label>Brightness</label>
                <input
                    type="range"
                    id="bright-slider"
                    min="0"
                    max="255"
                    value="255"
                />
                <span class="bright-val" id="bright-val">255</span>
            </div>
            <div class="led-send-row">
                <button id="led-btn" disabled>Apply</button>
            </div>
        </div>

        <!-- Sleep timeout -->
        <div class="card">
            <h2>Sleep Timeout</h2>
            <div style="display: flex; align-items: center; gap: 12px">
                <div style="flex: 1">
                    <label
                        >Minutes of inactivity before sleep (0 =
                        disabled)</label
                    >
                    <input
                        type="number"
                        id="sleep-input"
                        min="0"
                        max="255"
                        value="5"
                        style="
                            width: 100%;
                            background: var(--card);
                            border: 1px solid #2a4a6e;
                            border-radius: 0;
                            color: var(--text);
                            padding: 10px 12px;
                            font-size: 0.95rem;
                            font-family: monospace;
                            outline: none;
                        "
                    />
                </div>
                <button id="sleep-btn" disabled style="margin-top: 20px">
                    Set
                </button>
            </div>
        </div>

        <!-- Log -->
        <div class="card">
            <h2>Serial Log</h2>
            <div id="log"></div>
        </div>
        <p class="subtitle">John Park for Adafruit 2026</p>

        <script>
            let port = null,
                reader = null,
                writer = null;
            let currentMode = "single";
            let capturedKey = { single: "", combo: "" };
            let readBuffer = "";

            if (!("serial" in navigator)) {
                document.getElementById("not-supported").style.display =
                    "block";
```

```
}

function log(msg, type = "info") {
    const el = document.getElementById("log");
    const line = document.createElement("div");
    line.className = `log-line ${type}`;
    const ts = new Date().toLocaleTimeString([], {
        hour: "2-digit",
        minute: "2-digit",
        second: "2-digit",
    });
    line.textContent = `[${ts}] ${msg}`;
    el.appendChild(line);
    el.scrollTop = el.scrollHeight;
}

document
    .getElementById("connect-btn")
    .addEventListener("click", async () => {
        try {
            port = await navigator.serial.requestPort();
            await port.open({ baudRate: 115200 });
            writer = port.writable.getWriter();
            setConnected(true);
            log("Connected to serial port", "info");
            startReading();
            setTimeout(() => sendRaw("\n"), 800);
        } catch (e) {
            log("Connection failed: " + e.message, "err");
        }
    });

document
    .getElementById("disconnect-btn")
    .addEventListener("click", async () => {
        await doDisconnect();
    });

async function doDisconnect() {
    try {
        if (reader) {
            await reader.cancel();
            reader = null;
        }
        if (writer) {
            writer.releaseLock();
            writer = null;
        }
        if (port) {
            await port.close();
            port = null;
        }
    } catch (e) {}
    setConnected(false);
    log("Disconnected", "info");
}

function setConnected(c) {
    document.getElementById("dot").className =
        "dot" + (c ? " connected" : "");
    document.getElementById("status-text").textContent = c
        ? "Connected"
        : "Not connected";
    document.getElementById("connect-btn").disabled = c;
    document.getElementById("disconnect-btn").disabled = !c;
    document.getElementById("send-btn").disabled = !c;
    document.getElementById("sleep-btn").disabled = !c;
    document.getElementById("led-btn").disabled = !c;
    if (!c)
```

```
                document.getElementById("current-combo").textContent = "—";
            }

            async function startReading() {
                const decoder = new TextDecoderStream();
                port.readable.pipeTo(decoder.writable);
                reader = decoder.readable.getReader();
                try {
                    while (true) {
                        const { value, done } = await reader.read();
                        if (done) break;
                        readBuffer += value;
                        let lines = readBuffer.split("\n");
                        readBuffer = lines.pop();
                        for (const line of lines) {
                            const t = line.trim();
                            if (!t) continue;
                            log(t, "rx");
                            parseIncoming(t);
                        }
                    }
                } catch (e) {
                    if (port) log("Read error: " + e.message, "err");
                }
            }

            function parseIncoming(line) {
                const m = line.match(
                    /\[CFG\].*(?:combo|Loaded)[^\:]*[:\—–]\s*(.+)/,
                );
                if (m)
                    document.getElementById("current-combo").textContent =
                        m[1].trim();

                const s = line.match(/\[CFG\] Sleep timeout:\s*(\d+)/);
                if (s) document.getElementById("sleep-input").value = s[1];

                const sd = line.match(/\[CFG\] Sleep disabled/);
                if (sd) document.getElementById("sleep-input").value = "0";

                // Sync brightness from boot: "[CFG] Brightness: 255"
                const br = line.match(/\[CFG\] Brightness:\s*(\d+)/);
                if (br) {
                    const v = parseInt(br[1]);
                    document.getElementById("bright-slider").value = v;
                    document.getElementById("bright-val").textContent = v;
                }

                // Sync connected color from boot: "[CFG] Connected color: 0,255,0"
                const cc = line.match(
                    /\[CFG\] Connected color:\s*(\d+),(\d+),(\d+)/,
                );
                if (cc) {
                    const hex = rgbToHex(
                        parseInt(cc[1]),
                        parseInt(cc[2]),
                        parseInt(cc[3]),
                    );
                    document.getElementById("picker-connected").value = hex;
                    document.getElementById(
                        "swatch-connected",
                    ).style.background = hex;
                }

                // Sync pressed color from boot: "[CFG] Pressed color: 255,20,100"
                const pc = line.match(
                    /\[CFG\] Pressed color:\s*(\d+),(\d+),(\d+)/,
                );
                if (pc) {
```

```
            const hex = rgbToHex(
                parseInt(pc[1]),
                parseInt(pc[2]),
                parseInt(pc[3]),
            );
            document.getElementById("picker-pressed").value = hex;
            document.getElementById("swatch-pressed").style.background =
                hex;
        }
    }

    // ---- LED helpers ----------------------------------
    function rgbToHex(r, g, b) {
        return (
            "#" +
            [r, g, b]
                .map((v) => v.toString(16).padStart(2, "0"))
                .join("")
        );
    }

    function hexToRgb(hex) {
        const r = parseInt(hex.slice(1, 3), 16);
        const g = parseInt(hex.slice(3, 5), 16);
        const b = parseInt(hex.slice(5, 7), 16);
        return { r, g, b };
    }

    // Color swatch clicks open the hidden color picker
    document
        .getElementById("swatch-connected")
        .addEventListener("click", () => {
            document.getElementById("picker-connected").click();
        });
    document
        .getElementById("swatch-pressed")
        .addEventListener("click", () => {
            document.getElementById("picker-pressed").click();
        });

    // Keep swatch in sync with picker
    document
        .getElementById("picker-connected")
        .addEventListener("input", (e) => {
            document.getElementById(
                "swatch-connected",
            ).style.background = e.target.value;
        });
    document
        .getElementById("picker-pressed")
        .addEventListener("input", (e) => {
            document.getElementById("swatch-pressed").style.background =
                e.target.value;
        });

    // Brightness slider live value display
    document
        .getElementById("bright-slider")
        .addEventListener("input", (e) => {
            document.getElementById("bright-val").textContent =
                e.target.value;
        });

    // Apply button — sends all three LED commands
    document.getElementById("led-btn").addEventListener("click", () => {
        const connHex =
            document.getElementById("picker-connected").value;
        const pressHex =
            document.getElementById("picker-pressed").value;
```

```
        const bright = document.getElementById("bright-slider").value;
        const conn = hexToRgb(connHex);
        const press = hexToRgb(pressHex);
        sendCommand(`color:connected:${conn.r},${conn.g},${conn.b}`);
        setTimeout(
            () =>
                sendCommand(
                    `color:pressed:${press.r},${press.g},${press.b}`,
                ),
            150,
        );
        setTimeout(() => sendCommand(`bright:${bright}`), 300);
    });

    // ---- Serial ---------------------------------------
    async function sendRaw(str) {
        if (!writer) return;
        await writer.write(new TextEncoder().encode(str));
    }

    async function sendCommand(cmd) {
        if (!writer) return;
        log("> " + cmd, "tx");
        await sendRaw(cmd + "\n");
    }

    document
        .getElementById("send-btn")
        .addEventListener("click", () => {
            const cmd = buildCommand();
            if (cmd) sendCommand(cmd);
        });

    function buildCommand() {
        switch (currentMode) {
            case "single": {
                const k =
                    capturedKey.single ||
                    document.getElementById("special-single").value;
                if (!k) {
                    log("No key selected", "err");
                    return null;
                }
                return k;
            }
            case "combo": {
                const mods = [
                    ...document.querySelectorAll(
                        ".mod-check input:checked",
                    ),
                ].map((i) => i.value);
                const k =
                    capturedKey.combo ||
                    document.getElementById("special-combo").value;
                if (!k) {
                    log("No key selected", "err");
                    return null;
                }
                return [...mods, k].join("+");
            }
            case "string": {
                const s = document.getElementById("string-input").value;
                if (!s) {
                    log("No string entered", "err");
                    return null;
                }
                return `"${s}"`;
            }
            case "string-key": {
```

```
                const s =
                    document.getElementById("string-key-input").value;
                const t = document.getElementById("tail-key").value;
                if (!s) {
                    log("No string entered", "err");
                    return null;
                }
                return `"${s}"+${t}`;
            }
            case "media": {
                return document.getElementById("media-key").value;
            }
        }
        return null;
    }

    function updatePreview() {
        const cmd = buildCommand();
        const el = document.getElementById("preview");
        el.innerHTML = cmd
            ? `<span>${cmd}</span>`
            : '<span style="color:var(--muted)">—</span>';
    }

    document
        .getElementById("sleep-btn")
        .addEventListener("click", () => {
            const val = parseInt(
                document.getElementById("sleep-input").value,
            );
            if (isNaN(val) || val < 0 || val > 255) {
                log("Invalid sleep value — enter 0–255", "err");
                return;
            }
            sendCommand(`sleep:${val}`);
        });

    document
        .getElementById("reset-btn")
        .addEventListener("click", () => {
            capturedKey = { single: "", combo: "" };
            document.getElementById("capture-single").textContent =
                "Click to capture key...";
            document.getElementById("capture-single").className =
                "key-capture";
            document.getElementById("capture-combo").textContent =
                "Click to capture key...";
            document.getElementById("capture-combo").className =
                "key-capture";
            document.getElementById("special-single").value = "";
            document.getElementById("special-combo").value = "";
            document.getElementById("string-input").value = "";
            document.getElementById("string-key-input").value = "";
            document.getElementById("tail-key").selectedIndex = 0;
            document.getElementById("media-key").selectedIndex = 0;
            document
                .querySelectorAll(".mod-check input")
                .forEach((cb) => {
                    cb.checked = false;
                    cb.closest(".mod-check").classList.remove(
                        "checked",
                    );
                });
            updatePreview();
        });

    document
        .getElementById("special-single")
        .addEventListener("change", () => {
```

```
            capturedKey.single = "";
            document.getElementById("capture-single").textContent =
                "Click to capture key...";
            document.getElementById("capture-single").className =
                "key-capture";
            updatePreview();
        });
    document
        .getElementById("special-combo")
        .addEventListener("change", () => {
            capturedKey.combo = "";
            document.getElementById("capture-combo").textContent =
                "Click to capture key...";
            document.getElementById("capture-combo").className =
                "key-capture";
            updatePreview();
        });
    document
        .getElementById("media-key")
        .addEventListener("change", updatePreview);
    document
        .getElementById("string-input")
        .addEventListener("input", updatePreview);
    document
        .getElementById("string-key-input")
        .addEventListener("input", updatePreview);
    document
        .getElementById("tail-key")
        .addEventListener("change", updatePreview);
    document.querySelectorAll(".mod-check input").forEach((cb) => {
        cb.addEventListener("change", () => {
            cb.closest(".mod-check").classList.toggle(
                "checked",
                cb.checked,
            );
            updatePreview();
        });
    });

    document.querySelectorAll(".tab").forEach((tab) => {
        tab.addEventListener("click", () => {
            currentMode = tab.dataset.mode;
            document
                .querySelectorAll(".tab")
                .forEach((t) => t.classList.remove("active"));
            document
                .querySelectorAll(".mode-panel")
                .forEach((p) => p.classList.remove("active"));
            tab.classList.add("active");
            document
                .getElementById("panel-" + currentMode)
                .classList.add("active");
            updatePreview();
        });
    });

    function setupCapture(id, stateKey) {
        const el = document.getElementById(id);
        let listening = false;
        el.addEventListener("click", () => {
            listening = true;
            el.textContent = "Press a key...";
            el.className = "key-capture listening";
            el.focus();
        });
        el.addEventListener("keydown", (e) => {
            if (!listening) return;
            e.preventDefault();
            if (["Control", "Shift", "Alt", "Meta"].includes(e.key))
```

```
                        return;
                    const k = keyEventToToken(e);
                    if (!k) return;
                    capturedKey[stateKey] = k;
                    el.textContent = k;
                    el.className = "key-capture has-key";
                    listening = false;
                    const sel =
                        id === "capture-single"
                            ? "special-single"
                            : "special-combo";
                    document.getElementById(sel).value = "";
                    updatePreview();
                });
                el.addEventListener("blur", () => {
                    if (listening) {
                        listening = false;
                        el.textContent =
                            capturedKey[stateKey] || "Click to capture key...";
                        el.className = capturedKey[stateKey]
                            ? "key-capture has-key"
                            : "key-capture";
                    }
                });
            }

            function keyEventToToken(e) {
                const map = {
                    Tab: "tab",
                    Enter: "return",
                    Escape: "esc",
                    Backspace: "backspace",
                    Delete: "delete",
                    " ": "space",
                    ArrowUp: "up",
                    ArrowDown: "down",
                    ArrowLeft: "left",
                    ArrowRight: "right",
                    Home: "home",
                    End: "end",
                    PageUp: "pageup",
                    PageDown: "pagedown",
                    Insert: "insert",
                };
                if (map[e.key]) return map[e.key];
                if (e.key.match(/^F\d{1,2}$/)) return e.key.toLowerCase();
                if (e.key.length === 1) return e.key.toLowerCase();
                return null;
            }

            setupCapture("capture-single", "single");
            setupCapture("capture-combo", "combo");
        </script>
    </body>
</html>
```