



# Adafruit NeoTrellis M4 Express

Created by lady ada



<https://learn.adafruit.com/adafruit-neotrellis-m4>

Last updated on 2024-04-09 10:47:56 AM EDT

# Table of Contents

Overview	7
Update the UF2 Bootloader	12
• Updating Your Bootloader	
Board Tour	15
• NeoPixel & Button Pads	
• Main Chipset & SPI FLASH	
• Audio TRRS Headset Jack	
• Stereo Audio Out	
• Microphone Input	
• JST Hacking Port and Pads	
• Triple Axis Accelerometer	
• Everything Else!	
Enclosure Assembly	22
• Prep	
• Elastomer Pads	
• Place the Trellis M4	
• Frame Layer	
• Penultimate Layer	
• Back Layer	
• Fasteners	
• Customization	
Arduino IDE Setup	33
Using with Arduino IDE	36
• Install SAMD Support	
• Install Adafruit SAMD	
• Windows 7 and 8.1	
• Blink	
• Successful Upload	
• Compilation Issues	
• Manually bootloading	
• Ubuntu & Linux Issue Fix	
Arduino Libraries	42
• Install Libraries	
• Adafruit NeoPixel	
• Adafruit DMA NeoPixel	
• Adafruit Unified Sensor	
• Adafruit SPIFlash	
• Adafruit Keypad	
• MIDI USB	
• ADXL343	
• NeoTrellis M4	
• SdFat - Adafruit Fork	
• Audio - Adafruit Fork	

Adapting Sketches to M0 & M4	45
<ul style="list-style-type: none"><li>• Analog References</li><li>• Pin Outputs &amp; Pullups</li><li>• Serial vs SerialUSB</li><li>• AnalogWrite / PWM on Feather/Metro M0</li><li>• analogWrite() PWM range</li><li>• analogWrite() DAC on A0</li><li>• serialEvent() and serialEvent1()</li><li>• Missing header files</li><li>• Bootloader Launching</li><li>• Aligned Memory Access</li><li>• Floating Point Conversion</li><li>• How Much RAM Available?</li><li>• Storing data in FLASH</li><li>• Pretty-Printing out registers</li><li>• M4 Performance Options</li><li>• CPU Speed (overclocking)</li><li>• Optimize</li><li>• Cache</li><li>• Max SPI and Max QSPI</li><li>• Enabling the Buck Converter on some M4 Boards</li></ul>	
Arduino Examples	53
<ul style="list-style-type: none"><li>• NeoPixel Test</li><li>• Keypad Test</li><li>• MIDI USB Test</li><li>• Audio Library Test</li><li>• Microphone Feed-thru Test</li><li>• Microphone FFT Test</li></ul>	
What is CircuitPython?	55
<ul style="list-style-type: none"><li>• CircuitPython is based on Python</li><li>• Why would I use CircuitPython?</li></ul>	
CircuitPython	56
Creating and Editing Code	59
<ul style="list-style-type: none"><li>• CircuitPython Libraries</li><li>• Choosing an Editor</li><li>• Creating Code</li><li>• Editing Code</li><li>• Exploring Your First CircuitPython Program</li><li>• Imports &amp; Libraries</li><li>• Setting Up The Trellis M4</li><li>• Loop-de-loops</li><li>• More Changes</li></ul>	
Connecting to the Serial Console	64
<ul style="list-style-type: none"><li>• Serial Console on Mac and Linux</li><li>• Serial Console on Windows</li></ul>	
Interacting with the Serial Console	65
The REPL	68
<ul style="list-style-type: none"><li>• Entering the REPL</li></ul>	

- [Interacting with the REPL](#)
- [Returning to the Serial Console](#)

## CircuitPython Libraries

72

- [The Adafruit Learn Guide Project Bundle](#)
- [The Adafruit CircuitPython Library Bundle](#)
- [Downloading the Adafruit CircuitPython Library Bundle](#)
- [The CircuitPython Community Library Bundle](#)
- [Downloading the CircuitPython Community Library Bundle](#)
- [Understanding the Bundle](#)
- [Example Files](#)
- [Copying Libraries to Your Board](#)
- [Understanding Which Libraries to Install](#)
- [Example: ImportError Due to Missing Library](#)
- [Library Install on Non-Express Boards](#)
- [Updating CircuitPython Libraries and Examples](#)
- [CircUp CLI Tool](#)

## CircuitPython Pins and Modules

83

- [CircuitPython Pins](#)
- [import board](#)
- [I2C, SPI, and UART](#)
- [What Are All the Available Names?](#)
- [Microcontroller Pin Names](#)
- [CircuitPython Built-In Modules](#)

## Frequently Asked Questions

89

- [Using Older Versions](#)
- [Python Arithmetic](#)
- [Wireless Connectivity](#)
- [Asyncio and Interrupts](#)
- [Status RGB LED](#)
- [Memory Issues](#)
- [Unsupported Hardware](#)

## Advanced Serial Console on Mac

95

- [What's the Port?](#)
- [Connect with screen](#)

## Advanced Serial Console on Windows

97

- [Windows 7 and 8.1](#)
- [What's the COM?](#)
- [Install Putty](#)

## Troubleshooting

101

- [Always Run the Latest Version of CircuitPython and Libraries](#)
- [I have to continue using CircuitPython 7.x or earlier. Where can I find compatible libraries?](#)
- [macOS Sonoma 14.x: Disk Errors Writing to CIRCUITPY](#)
- [Bootloader \(boardnameBOOT\) Drive Not Present](#)
- [Windows Explorer Locks Up When Accessing boardnameBOOT Drive](#)
- [Copying UF2 to boardnameBOOT Drive Hangs at 0% Copied](#)
- [CIRCUITPY Drive Does Not Appear or Disappears Quickly](#)
- [Device Errors or Problems on Windows](#)
- [Serial Console in Mu Not Displaying Anything](#)
- [code.py Restarts Constantly](#)
- [CircuitPython RGB Status Light](#)



- [CircuitPython 7.0.0 and Later](#)
- [CircuitPython 6.3.0 and earlier](#)
- [Serial console showing ValueError: Incompatible .mpy file](#)
- [CIRCUITPY Drive Issues](#)
- [Safe Mode](#)
- [To erase CIRCUITPY: storage.erase\\_filesystem\(\)](#)
- [Erase CIRCUITPY Without Access to the REPL](#)
- [For the specific boards listed below:](#)
- [For SAMD21 non-Express boards that have a UF2 bootloader:](#)
- [For SAMD21 non-Express boards that do not have a UF2 bootloader:](#)
- [Running Out of File Space on SAMD21 Non-Express Boards](#)
- [Delete something!](#)
- [Use tabs](#)
- [On MacOS?](#)
- [Prevent & Remove MacOS Hidden Files](#)
- [Copy Files on MacOS Without Creating Hidden Files](#)
- [Other MacOS Space-Saving Tips](#)
- [Device Locked Up or Boot Looping](#)

## Adafruit CircuitPython TrellisM4 Library

120

- [Installing the CircuitPython TrellisM4 Library](#)
- [TrellisM4 Library Features](#)
- [Now Let's Rotate the Board!](#)
- [Trellis M4 Coordinate Layout](#)
- [Width and Height Can Make Rainbows!](#)
- [More To Come!](#)

## Trellis M4 CircuitPython Demo

130

- [Imports and Setup](#)
- [wheel for the Rainbow](#)
- [A Set of Button Presses](#)
- [A Multidimensional List of NeoPixel States](#)
- [Extra Credit: List Comprehensions](#)
- [NeoPixels On: True or False?](#)
- [NeoTrellis M4 NeoPixel Toggle Code](#)

## Downloads

138

- [Files](#)
- [Schematics and Fabrication Prints](#)

## UF2 Bootloader Details

139

- [Entering Bootloader Mode](#)
- [Using the Mass Storage Bootloader](#)
- [Using the BOSSA Bootloader](#)
- [Running bossac on the command line](#)
- [Updating the bootloader](#)
- [Getting Rid of Windows Pop-ups](#)
- [Making your own UF2](#)
- [Installing the bootloader on a fresh/bricked board](#)

## FAQ/Troubleshooting

150



---

# Overview



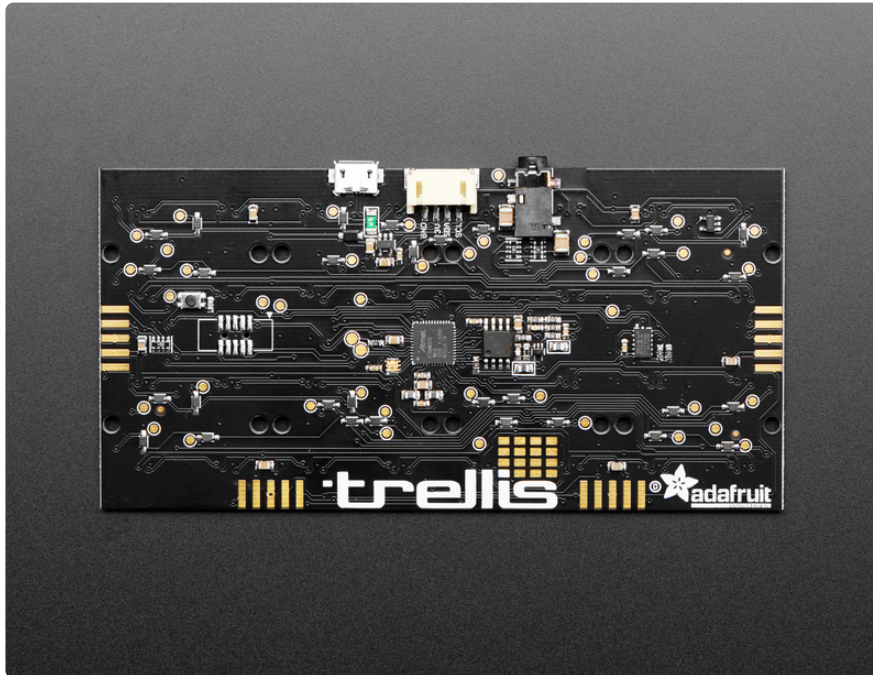
So you've got a cool/witty name for your band, a Soundcloud account, [a 3D-printed Daft Punk helmet \(https://adafru.it/CVS\)](https://adafru.it/CVS)... so what could be missing from your road to stardom? The **NeoTrellis M4**, of course!



The **NeoTrellis M4** is an all-in-one **USB + NeoPixel + Elastomer + Audio** board. It's powered by our new favoritest-chip-in-the-world, the SAMD51, a Cortex M4 core running at 120 MHz. This chip has a speedy core with CircuitPython and Arduino support, hardware DSP/floating point, dual DACs (more on that later!) and all the goodies you expect from normal chips like I2C, ADC, DMA, etc. It has a roomy 512KB

of flash and 192KB of SRAM so it's great for CircuitPython, we added a full 8MB flash chip so tons of space for files and audio clips. Or you can load Arduino in for bonkers-fast audio processing/generation [with our fork of the PJRC Audio library \(https://adafru.it/CVT\)](https://adafru.it/CVT).

You can also use [MakeCode \(https://adafru.it/C9N\)](https://adafru.it/C9N)'s block-based GUI coding environment on this board.

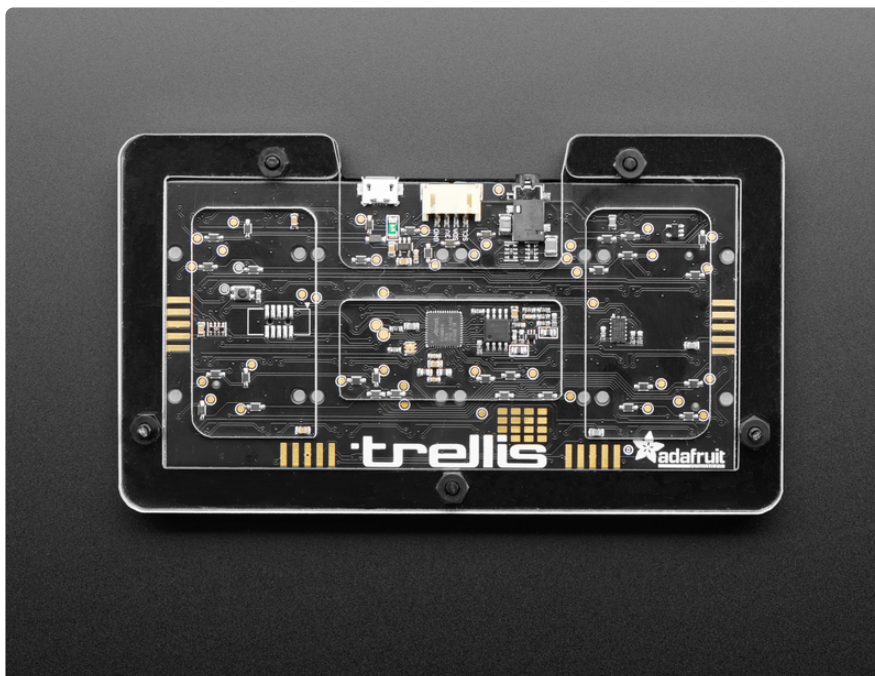


The native USB port can turn it into a MIDI USB device if you like - currently that's only supported in Arduino. Tether it to a computer or tablet, if you like. Or use it in standalone mode, as long as its powered from a USB power plug, it'll run whatever firmware is burned into it.

OK so you've got this big brain, but now you need inputs and outputs! There's a 4x8 grid of elastomer button pads with a NeoPixel nestled in the center of each one. You can read any/all button presses simultaneously thanks to the fully diode'd matrix, and also set each button color to any of 24-bit colors. (We've even got some DMA Arduino code for the NeoPixels so that they won't take up any processor time.) The elastomer buttons are translucent so they glow beautifully when lit.



Time to make some noise! We picked the SAMD51 mostly because its got that dual DAC - that's two 12-bit, 500KSPS 'true analog' outputs and we connected them to left and right on a standard headphone jack. Since the DAC pins are also ADC pins you could also use the left/right for audio line level input if you so choose. You're not going to get audiophile-quality outputs from two 12-bit DACs but you can certainly play audio clips and make beeps and bloops.



And if you want to have some audio inputs, we have you covered. The 4th pin on the headphone jack is for microphone or line in. If you have a classic 'mobile phone headset', the electret mic will go through an amplifier into an ADC pin. Again, it's not audiophile quality (we're talking about an electret mic here) but you can do audio

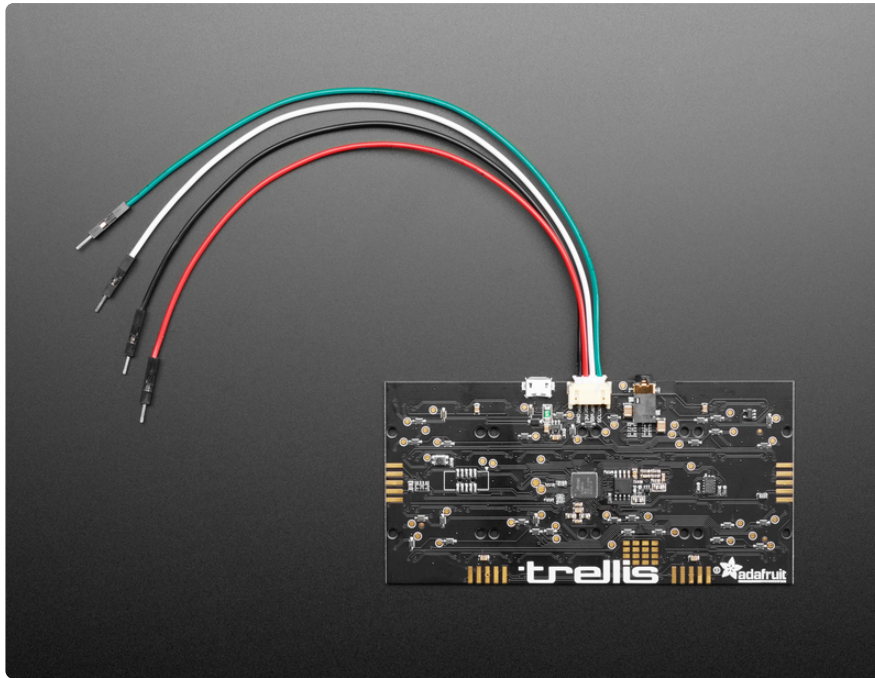
recording and filtering on the mono input. The raw un-amplified mic input is also connected to a DAC so you can read button presses on headsets that have a resistor-selector for their buttons, or some other 3.3V-max analog signal.



To add more interactivity, a precision triple-axis accelerometer from Analog Devices, the ADXL343, is included as well, and provides sensor information on tilt, motion, or tapping. Great for adding another dimension of data input in addition to the button pads.

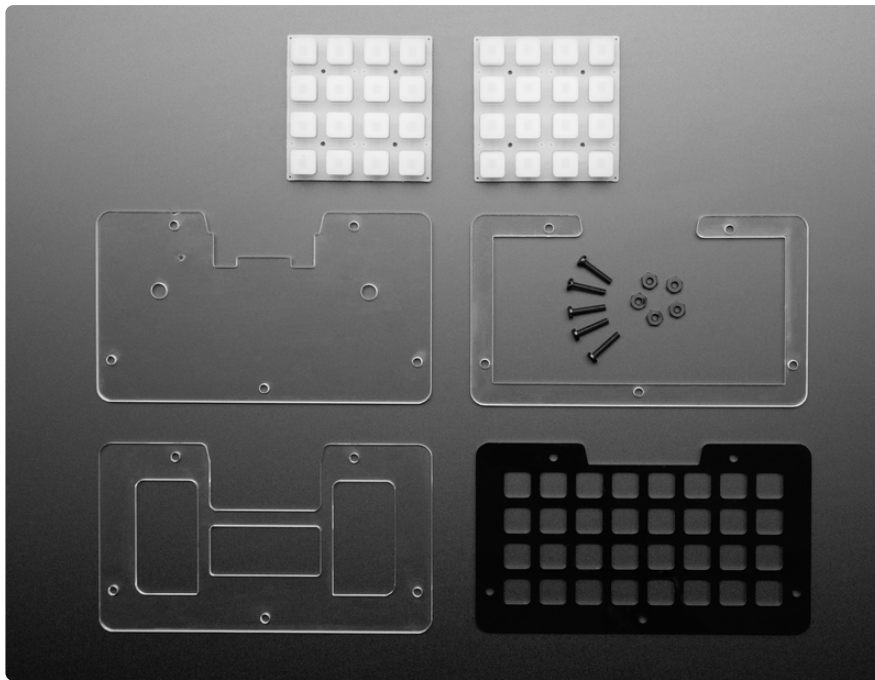
Finally, a 4 pin JST hacking port is available for extra add-ons. It's STEMMA and Grove compatible, and provides GND, 3.3V power, and two pins that can be used for I2C, ADC, or a UART. So connect some other sensor, or read stereo audio in, or maybe hack together a MIDI port. Whatever you like!





Here are some of the features you can look forward to when using NeoTrellis M4

- **ATSAMD51 32-bit Cortex M4** core running at **120 MHz** (32-bit, 3.3V logic and power)
- Hardware DSP and floating point support
- **512 KB** flash, **192 KB** RAM
- Native USB that can act as a true USB MIDI device if you like.
- **8 MB SPI FLASH** chip for storing files and CircuitPython code storage.
- 4x8 elastomer pads with fully diode'd matrix - no ghosting!
- 4x8 NeoPixels for each pad, glows through the elastomer buttons
- TRRS Headphone jack with stereo DAC outputs on Left/Right, can also be stereo ADC inputs. Fourth pin on headphone for electret/ADC input
- Built in [MAX4466 electret mic amplifier](http://adafru.it/1063) (<http://adafru.it/1063>) for mobile phone headsets. 'Raw' DC level reading also available on a separate ADC
- 4-JST hacking port with 3.3V power, ground, and two GPIO that can be I2C/ADC/UART
- [Analog Devices ADXL343 triple-axis accelerometer](https://adafru.it/CVU) (<https://adafru.it/CVU>)
- Really fun to press buttons and have sounds come out!



---

## Update the UF2 Bootloader

Update the Bootloader on your SAMD51 M4 board to prevent a somewhat rare problem of parts of internal flash being overwritten on power-up.

Your SAMD51 M4 board bootloader may need to be updated to fix an intermittent bug that can erase parts of internal flash.

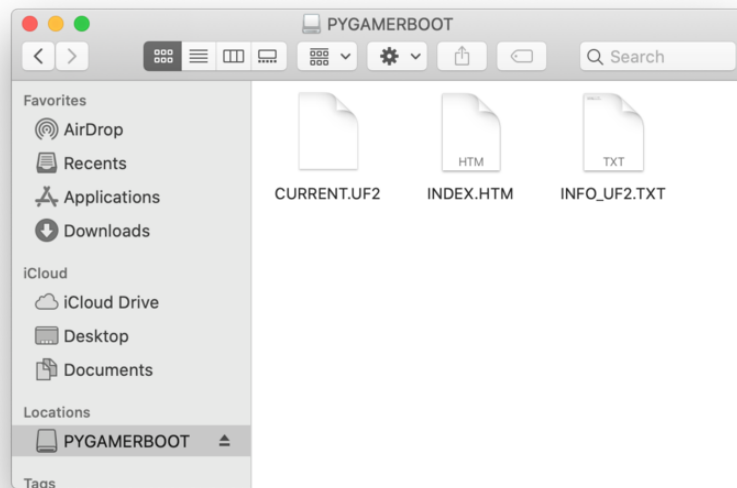
### Updating Your Bootloader

To see if you need to update your bootloader, get the UF2 boot drive to appear as a mounted drive on your computer, in a file browser window. If you're running MakeCode, click the reset button once. If you're running CircuitPython or an Arduino program, double-click the reset button.

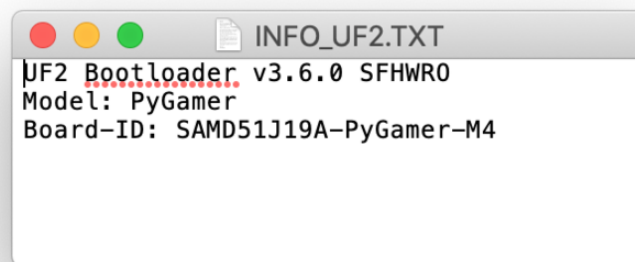
When you see the **...BOOT** drive (**FEATHERBOOT**, **METROM4BOOT**, **ITSYM4BOOT**, **PORTALBOOT**, etc.) , click the drive in the file browser window and then double-click the **INFO\_UF2.TXT** file to see what's inside.

The example screenshots below are for a PyGamer. What you see for your board will be largely the same except for the board name and the **BOOT** drive name.





The bootloader version is listed in **INFO\_UF2.TXT**. In this example, the version is **v3.6.0**.



**If the bootloader version you see is older than v3.9.0, you need to update.** For instance, the bootloader above needs to be upgraded.

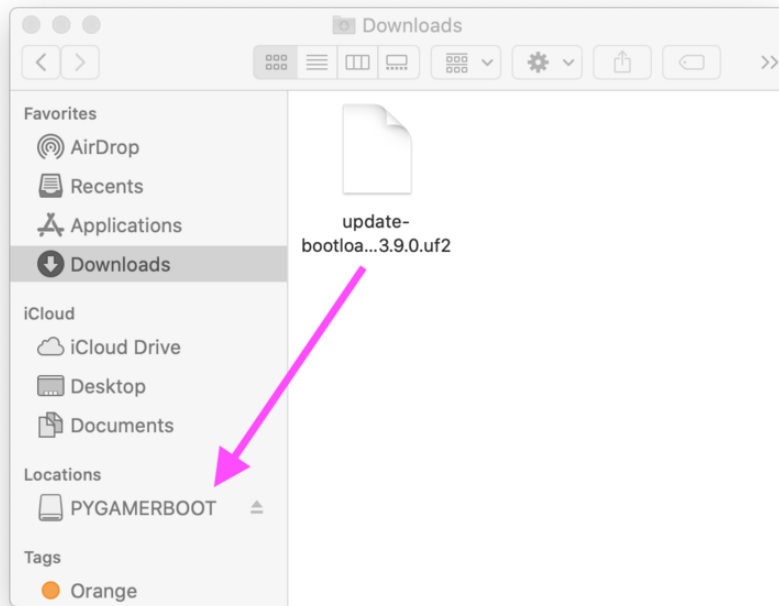
Download the latest version of the bootloader updater from the [circuitpython.org Downloads](https://circuitpython.org/downloads) page for your board.

The screenshot shows the Adafruit PyGamer product page. The main content area includes a large image of the PyGamer board, a detailed description of its features, and a sidebar with links to downloads, libraries, and a 'Get Started' button. A blue arrow points to the 'UF2 Bootloader' section in the sidebar, which includes a 'Download Updater UF2' button.

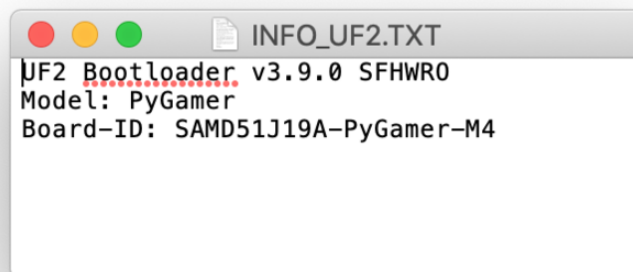
## Latest board downloads

<https://adafru.it/Em8>

- The bootloader updater will be named **update-bootloader-name\_of\_your\_board-v3.9.0.uf2** or some later version. Drag that file from your **Downloads** folder onto the **BOOT** drive:



After you drag the updater onto the boot drive, the red LED on the board will flicker and then blink slowly about five times. A few seconds later, the **BOOT** will appear in the Finder. After that, you can click on the **BOOT** drive and double-click **INFO\_UF2.TXT** again to confirm you've updated the bootloader.

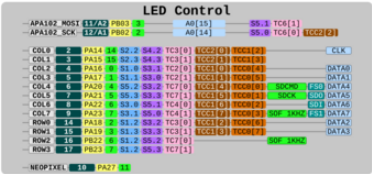


---

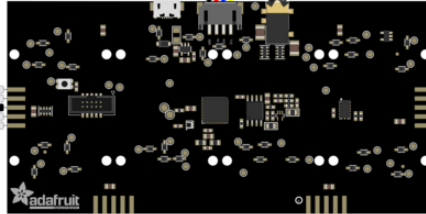
## Board Tour

OK there's a lot on the NeoTrellis M4 - so hold tight while we take a tour

<https://www.adafruit.com/product/3938>



The Microchip (nee Atmel) SAMD51 is an ARM Cortex-M4F running at 120 MHz with 192 or 256KB on-chip SRAM, up to 1MB Flash memory and built in USB. All GPIO is 3.3V in/out max unless otherwise stated. SERCOMs can be used as UART (TX on SERCOM pad 0, RX on any pad), I2C (SDA on pad 0, SCL on pad 1), or SPI (SCK on pad 1, MOSI on pad 0 or 3, MISO on any pad remaining)



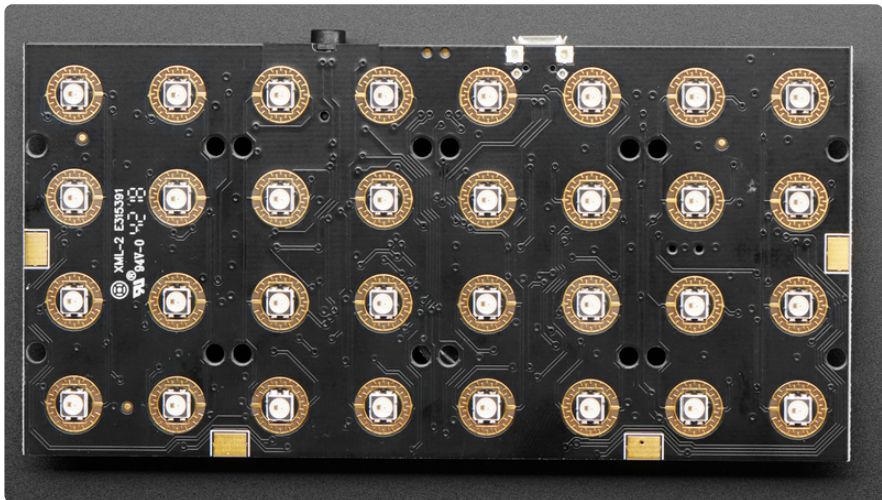
TC4[1]	S4,0	A0[3]/A1[1]	0	PB09	34	5
TC4[0]	S4,0	A0[2]/A1[0]	0	PB08	33	5
TC0[0]	S0,0	A0[4]	0	PB04	18	2

SCL	34	P809	9	A0[3]/A1[1]	S4.1	TC4[1]
SDA	33	P808	8	A0[2]/A1[0]	S4.0	TC4[0]
GND						
VDD						
INT	18	PA04	4	VDD#B	A0[4]	S0.0
						TC0[0]

[Click here to view a PDF version of the pinout diagram \(https://adafru.it/-At\)](https://adafru.it/-At)

# NeoPixel & Button Pads

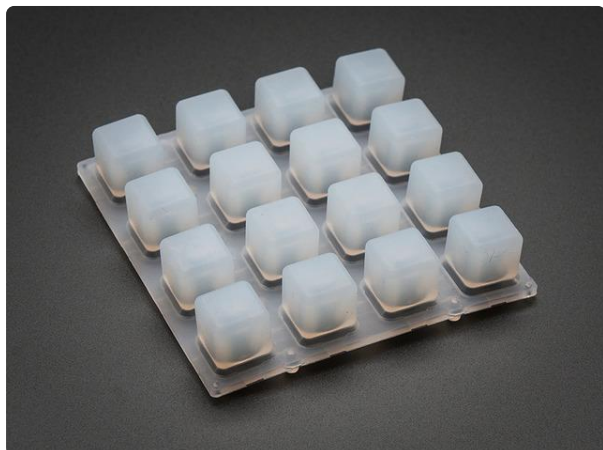
On the front side of the NeoTrellis is a 4x8 grid of NeoPixels and round gold inter-digit pads for elastomer buttons.



The NeoPixels are chained together, and connect in a zig-zag. The first pixel is in the top left, the last pixel is on the bottom right. In Arduino, the NeoPixels are on pin **10**, in CircuitPython they are on `board.NEOPIXEL`

You can use any NeoPixel library you like. Note that FastLED does not at the time of this writing, support the NeoTrellis M4. We do have a DMA NeoPixel library in Arduino that will take care of writes for you without processor time, and we recommend that!

The elastomer pads require buttons to sap on top. There's holes in the PCB that nubs from our Trellis pads fit into. You'll need **two** 4x4 pads:



#### Silicone Elastomer 4x4 Button Keypad - for 3mm LEDs

So squishy! These silicone elastomer keypads are just waiting for your fingers to press them. Go ahead, squish all you like! (They're durable and easy to clean, just wipe with mild...

<https://www.adafruit.com/product/1611>

The buttons are set up in a 4 x 8 matrix, there are 4 rows and 8 columns. Each button has a back-stop diode so you can press any/all keys at the same time without any 'ghosting'.

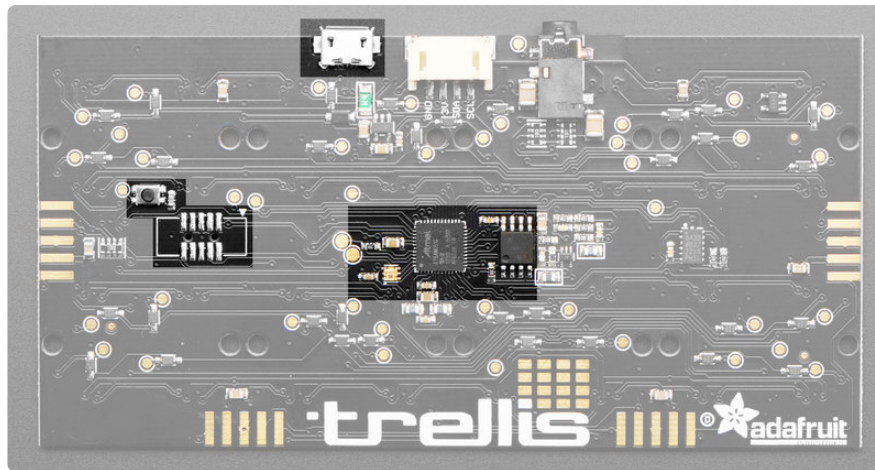
In CircuitPython, the rows/columns are on `board.ROW0` , `board.ROW1` , `board.ROW2` , `board.ROW3` and `board.COL0` , `board.COL1` , `board.COL2` , `board.COL3` , `board.COL4` , `board.COL5` , `board.COL6` , `board.COL7`

In Arduino the 4 rows are on pin **14-17** and the 8 columns are on pins **2-9**

(We recommend just using our libraries to read the matrix instead of DIYing)

## Main Chipset & SPI FLASH

This is what you came for!



Each NeoTrellis M4 comes with the main processor

- **ATSAMD51 32-bit Cortex M4** core running at **120 MHz** (32-bit, 3.3V logic and power)
- Hardware DSP and floating point support
- **512 KB** flash, **192 KB** RAM

At the top of the board there's a micro USB port, used for debugging/uploading code and is a native USB devices so it can act as a true USB MIDI device if you like. At the time of this writing only Arduino support has USB MIDI but we hope to add it to CircuitPython as well.

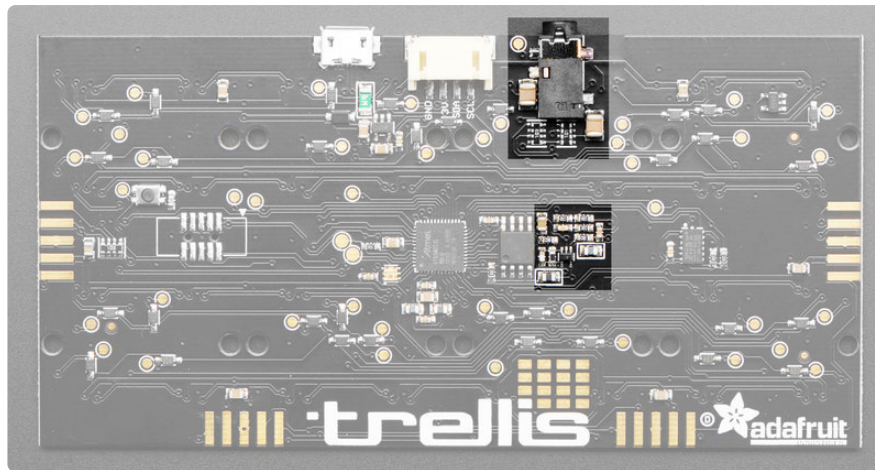
There is a 500mA fuse on the USB port which will throw when more than 1A is drawn, to protect any device you plug the NeoTrellis into.

For on-board storage of audio files, code, or whatever else you like, there's also a **8 MB QSPI FLASH** chip. In general we tend to use this only in CircuitPython because it appears as a USB disk drive. However, you can use it in Arduino, you just will need to load CircuitPython temporarily if you want to drag-n-drop files on and off

## Audio TRRS Headset Jack

Unlike most button boards, the NeoTrellis M4 has built in audio generation/output and input





## Stereo Audio Out

The SAMD51 has a dual DAC - that's two 12-bit, 500KSPS 'true analog' outputs and we connected them to left and right on a standard headphone jack. You're not going to get audiophile-quality outputs from two 12-bit DACs but you can certainly play audio clips and make beeps and bloops. They are AC coupled with a 100uF capacitor, and have a small divider so that the output is about line level.

Since the DAC pins are also ADC pins you could also use the left/right for audio line level input if you so choose. In that case, the same DAC channels can be used in ADC mode, the inputs are AC coupled and then biased to a 1.65V center.

The left channel is on Arduino **A1** or CircuitPython `board.A1`. The right channel is on Arduino **A2** or CircuitPython `board.A0`.

## Microphone Input

If you want to have some audio inputs, the 4th pin on the headphone jack is for microphone or line in. If you have a classic 'mobile phone headset', the electret mic will go through a MAX4466 electret amplifier with 100x gain into an ADC pin. Again, it's not audiophile quality (we're talking about an electret mic here) but you can do audio recording and filtering on the mono input.

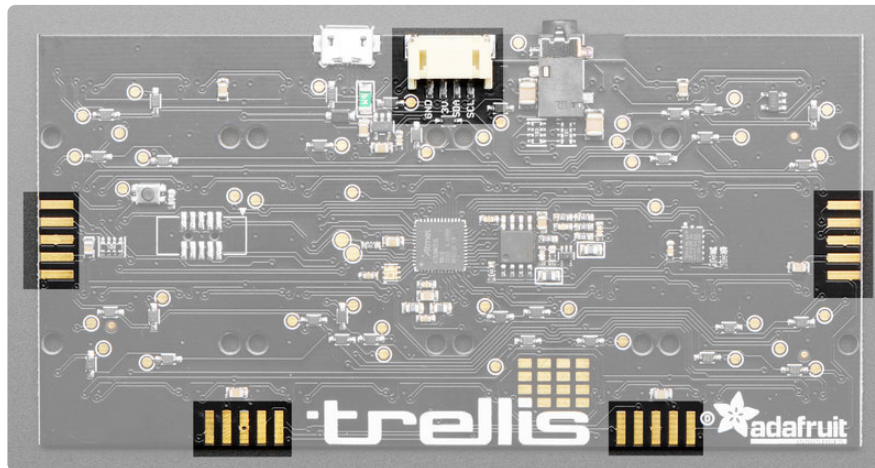
The raw un-amplified mic input is also connected to an ADC so you can read button presses on headsets that have a resistor-selector for their buttons, or some other 3.3V-max analog signal. In general, the DC bias will be about 2.5VDC when a microphone is plugged in. When pressed, the 'center' pause button on headsets will set the DC bias on this pin to ground.

If you have an Android (not Apple) headset with up/down buttons as well, the bias shift to 1.7V and 0.9V (approximately) so you can detect those button presses. Apple headsets send digital chirps on the up/down buttons and we don't have code or hardware to decode those!

On Arduino, the amplified microphone signal is on **PIN\_MIC** and the raw DC signal is available on **PIN\_MIC\_RAW**. With CircuitPython use `board.MICOUT` and `board.MICOUT_RAW`

## JST Hacking Port and Pads

If you want to plug in some custom hardware, we make it easy with a 4 pin JST hacking port, available for extra add-ons. It's STEMMA and Grove compatible, and provides GND, 3.3V power, and two pins that can be used for I2C, ADC, or a UART.



If you're using Arduino, the two GPIO are available on **PIN\_WIRE\_SDA** and **PIN\_WIRE\_SCL** for I2C usage, or **A4** and **A5** for analog reading.

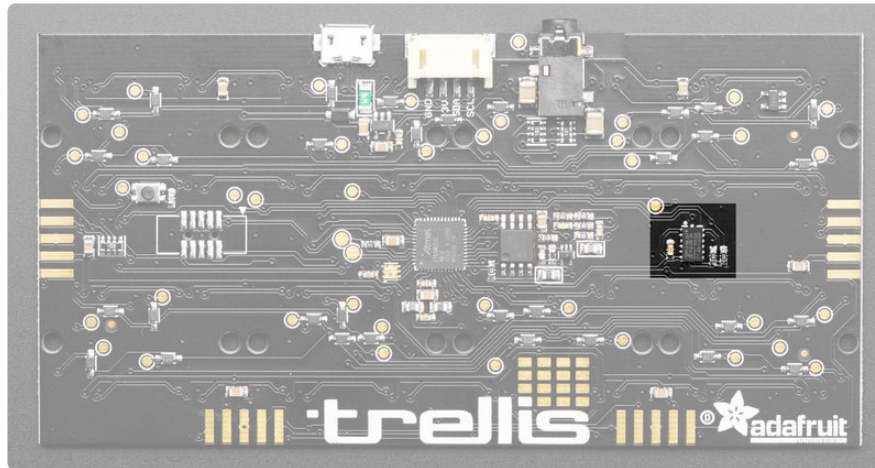
In CircuitPython, you can use the STEMMA connector with `board.SCL` and `board.SDA`, or `board.STEMMA_I2C()`.

There are also pads around the PCB which provide VIN, GND, SDA, SCL as well as a fifth pin, INT, in case you want to extend the NeoTrellis M4 with extra NeoTrellis boards. (This is advanced and we don't have any documentation/tutorials on how to do it)



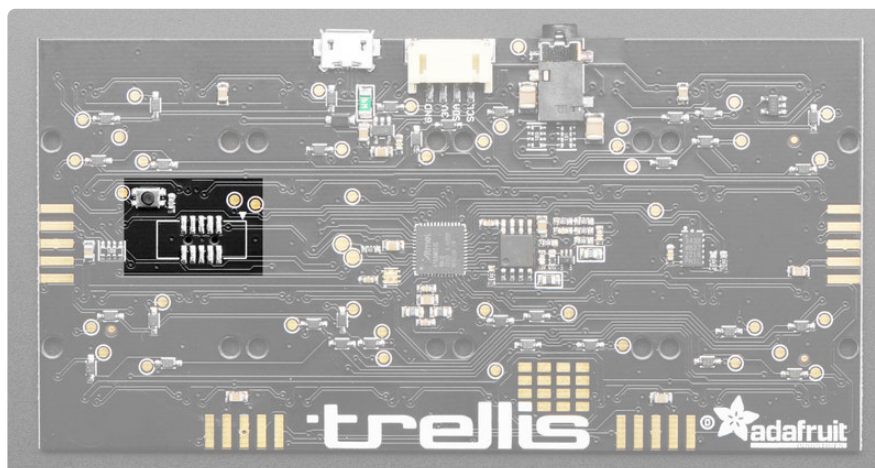
# Triple Axis Accelerometer

To add more interactivity, a precision triple-axis accelerometer from Analog Devices, the ADXL343, is included as well, and provides sensor information on tilt, motion, or tapping. Great for adding another dimension of data input in addition to the button pads.



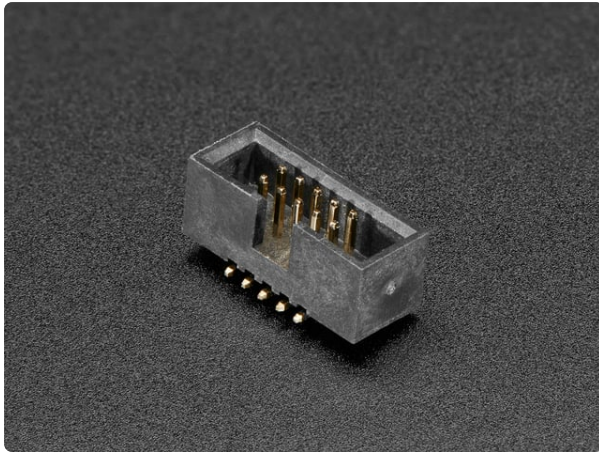
This sensor is on a separate I2C port, which is available in Arduino under **Wire1** or in CircuitPython under `board.ACCELEROMETER_SCL` and `board.ACCELEROMETER_SDA`

## Everything Else!



There's a small reset button on the back of the PCB, you can double-click it to enter the bootloader.

We also have a spot for an SWD port. We didn't solder this in because most people don't need it and it would keep the case from being nice n slim. You can solder a 2x5 0.05" connector if you like to use SWD debugging



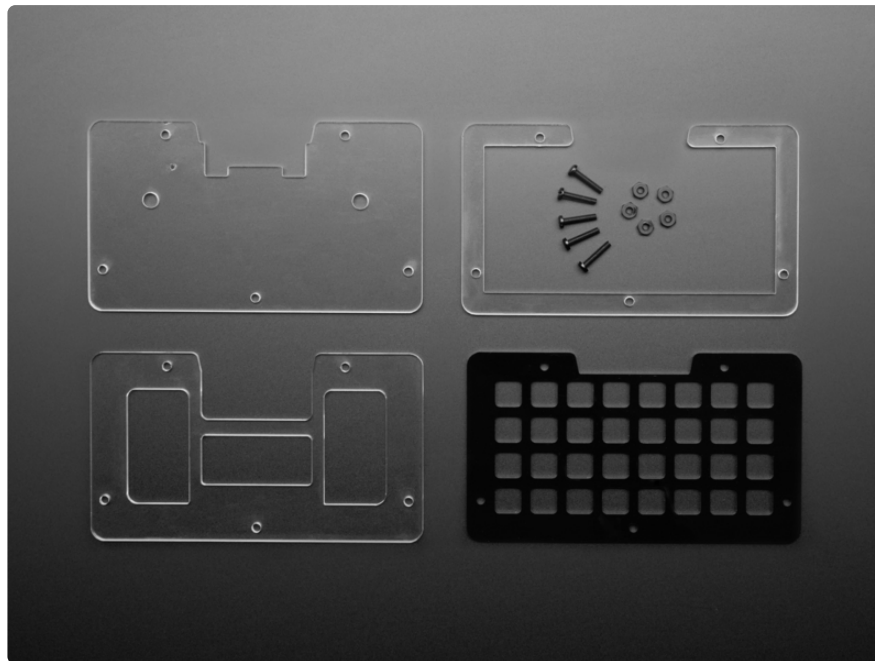
### SWD 0.05" Pitch Connector - 10 Pin SMT Box Header

This 1.27mm pitch, 2x5 male SMT Box Header is the same one used on our SWD Cable Breakout Board. The header...

<https://www.adafruit.com/product/752>

---

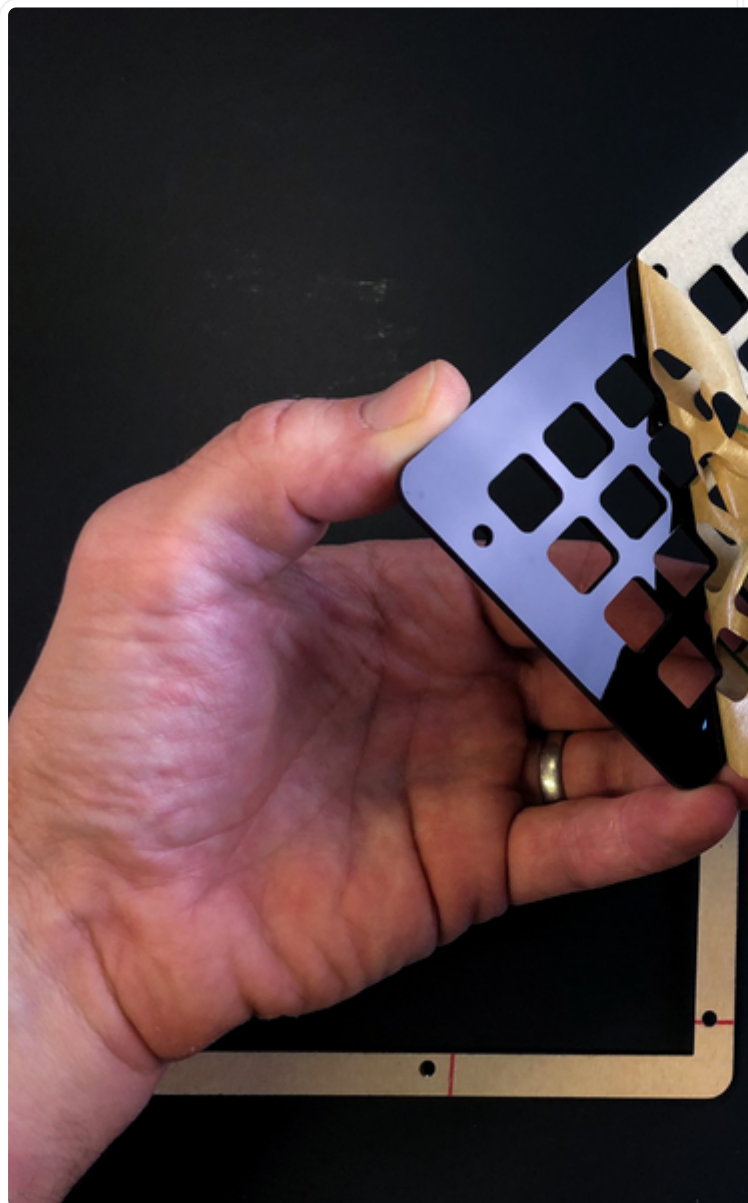
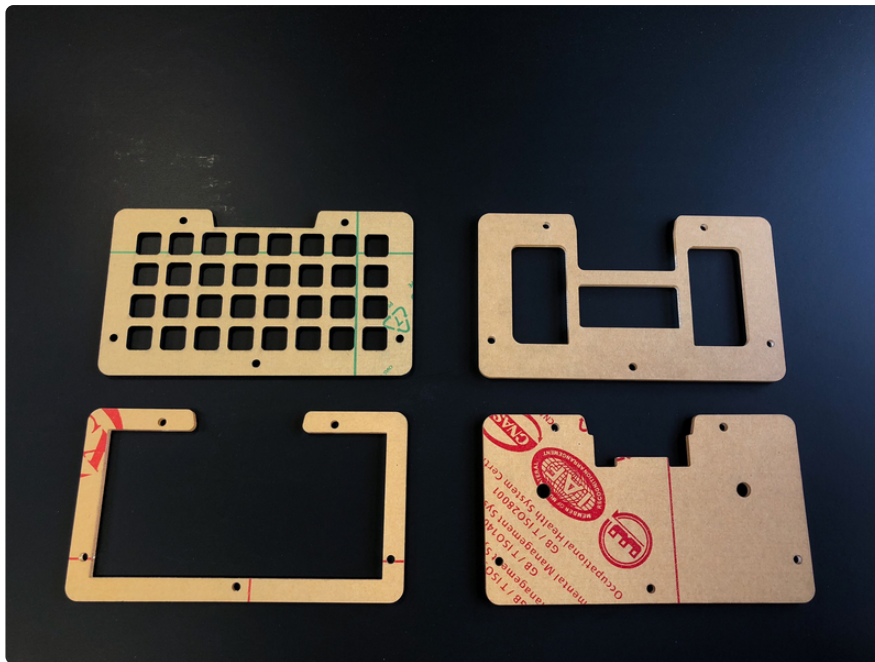
## Enclosure Assembly



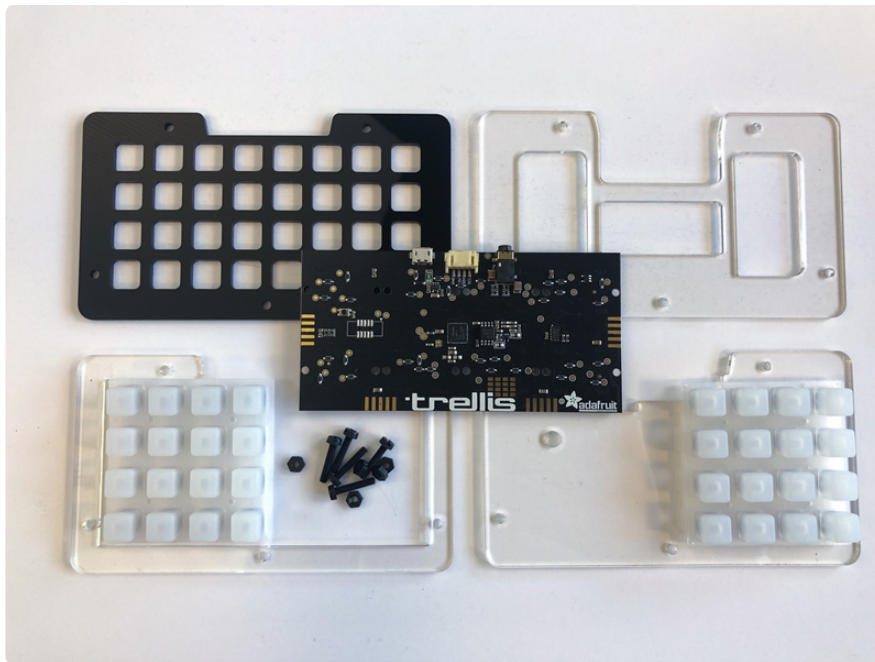
The acrylic enclosure is quick and easy to assemble. Each part is horizontally symmetrical, so there is very little worry about placing them in a "wrong" direction.

### Prep

First, you have the satisfying task of peeling off the protective paper from the acrylic pieces. If you peel off all of the paper without tearing them you will have seven weeks of good luck. Do not squander them.

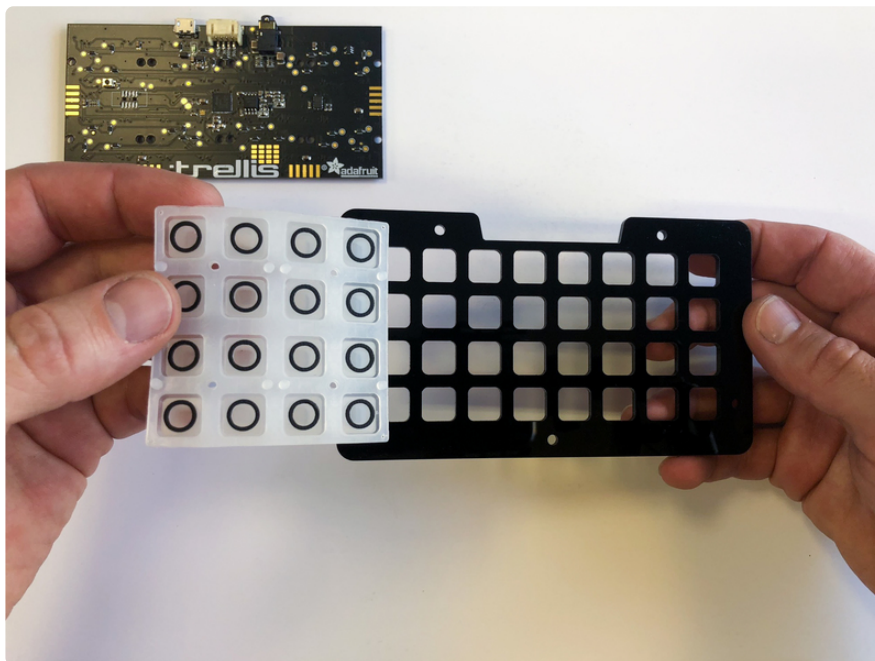


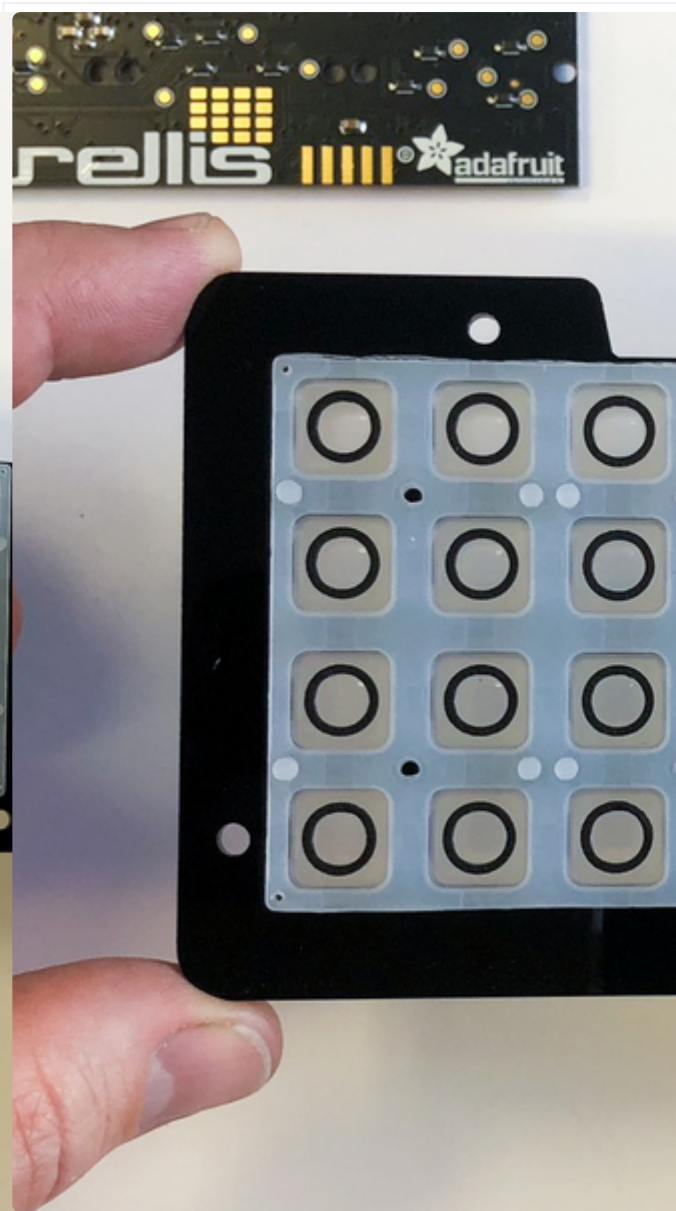
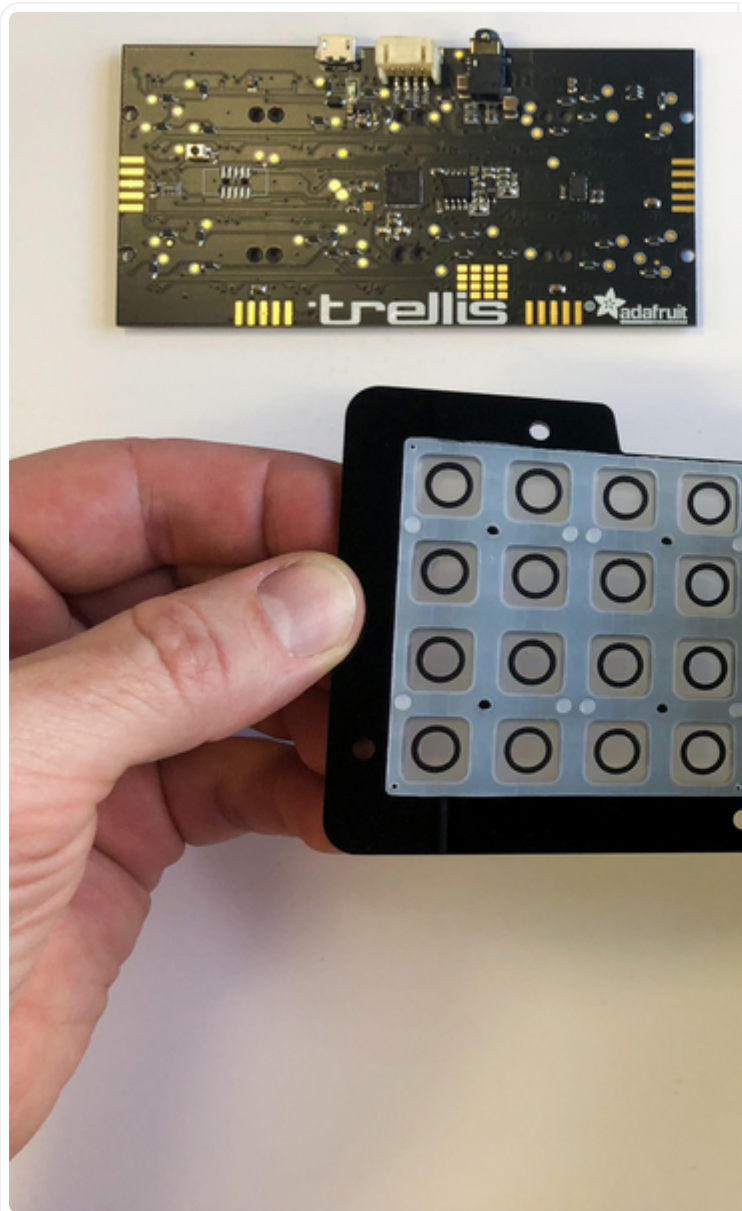




## Elastomer Pads

Place both of the button pads into the top piece as shown. Note, there are two pairs of rubber registration nubs which must align with their associated holes on the Trellis M4 board, so match the orientation as seen in the photos.

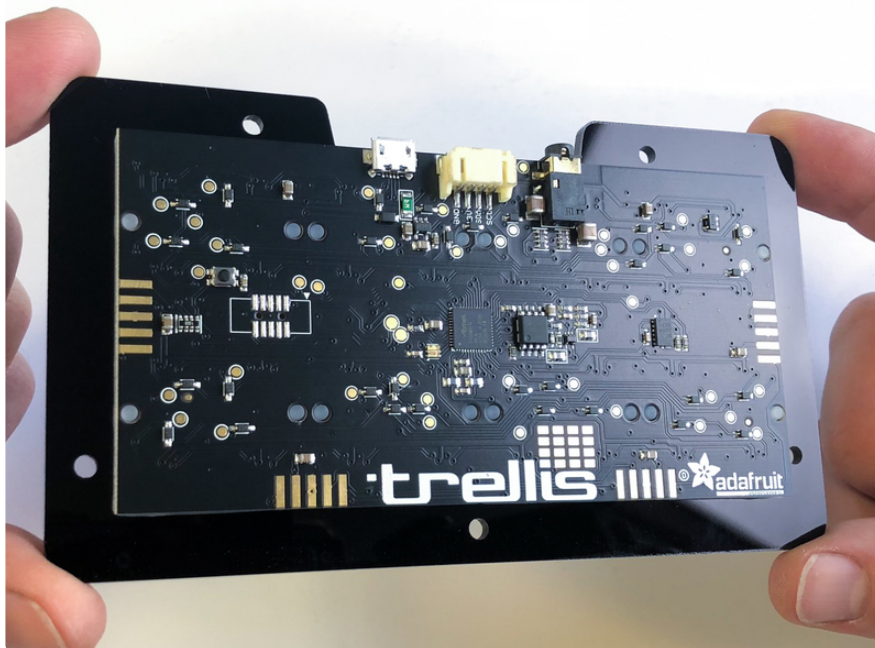
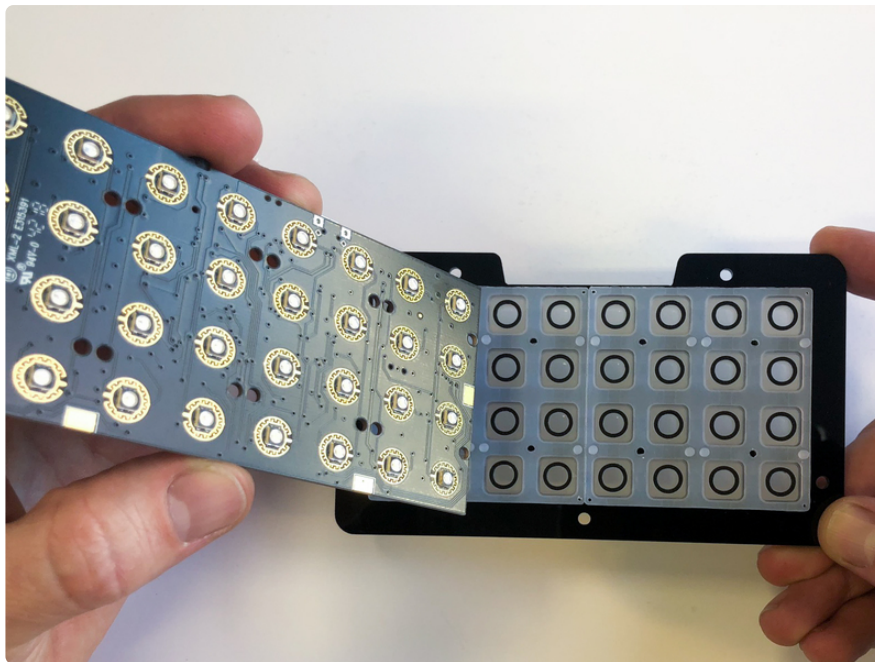




## Place the Trellis M4

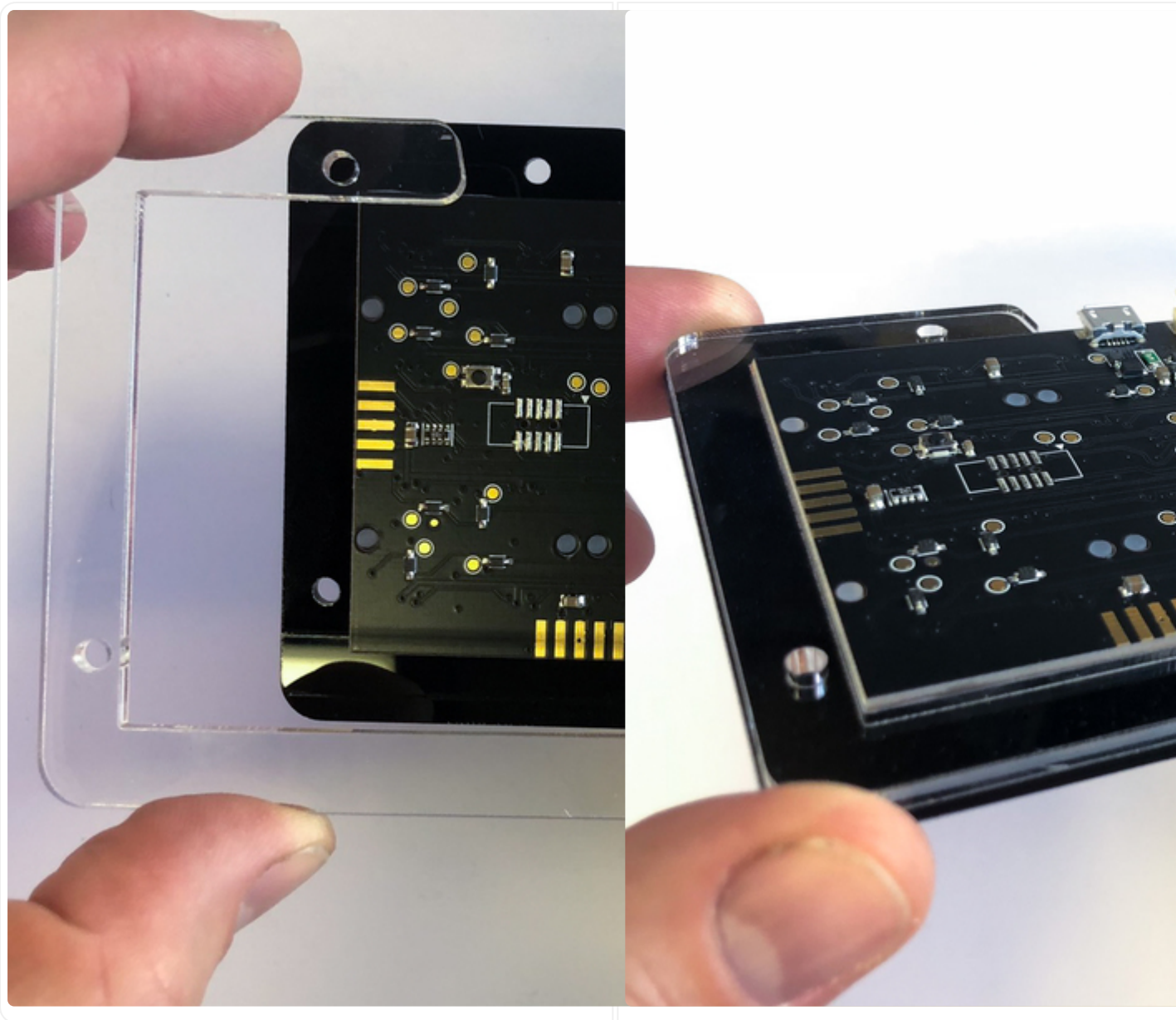
Place the Trellis M4 board down onto the elastomer pads with the NeoPixels and button contacts aligned with the buttons as shown. Make sure all of the sixteen registration nubs fit into their associated holes on the PCB.





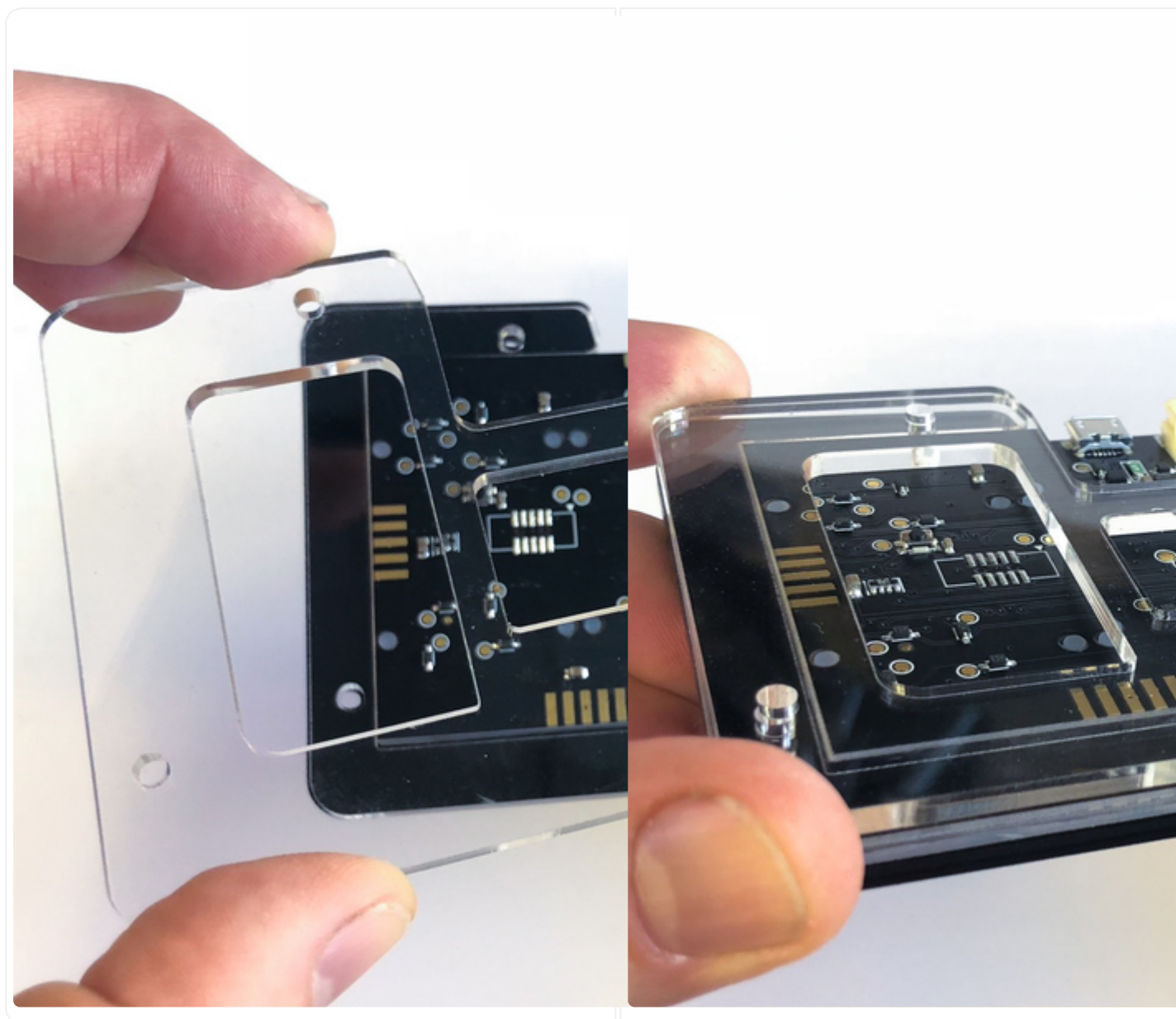
## Frame Layer

Now, place the frame layer which helps hold the board in place.



## Penultimate Layer

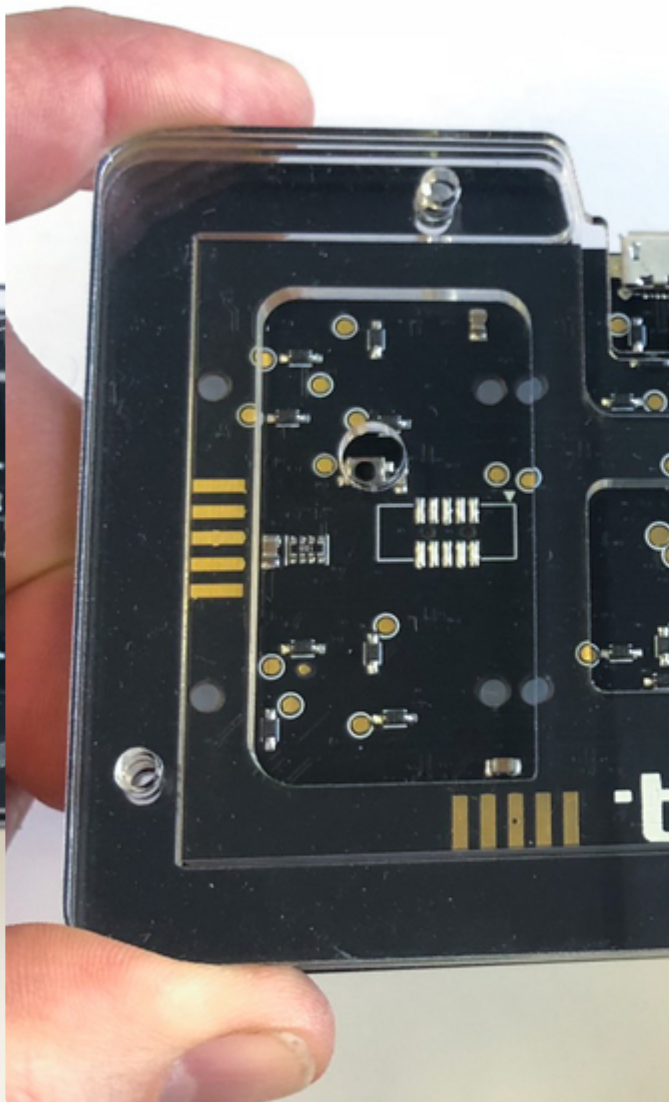
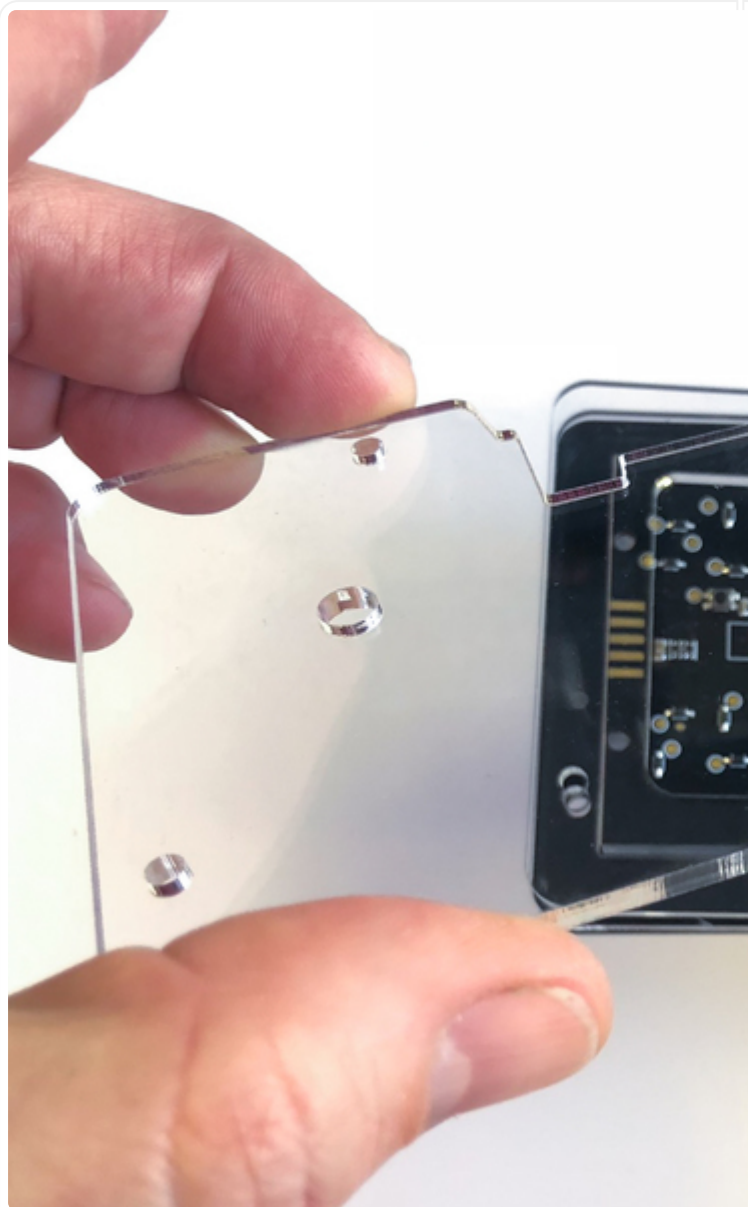
This next layer has some large cutouts in it to allow for parts clearance.



## Back Layer

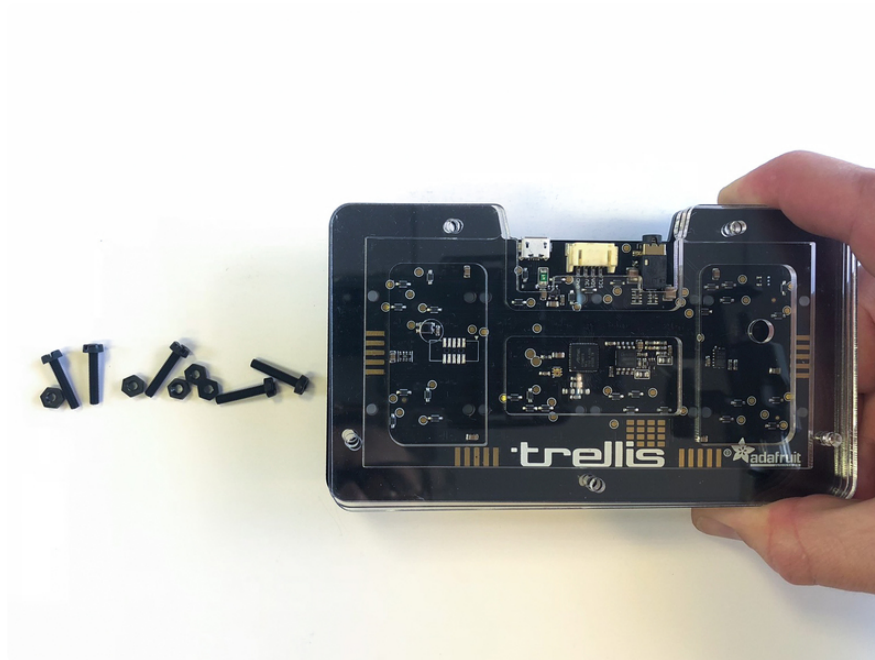
It's the final layer! Lay it down and get ready for some fastening action.



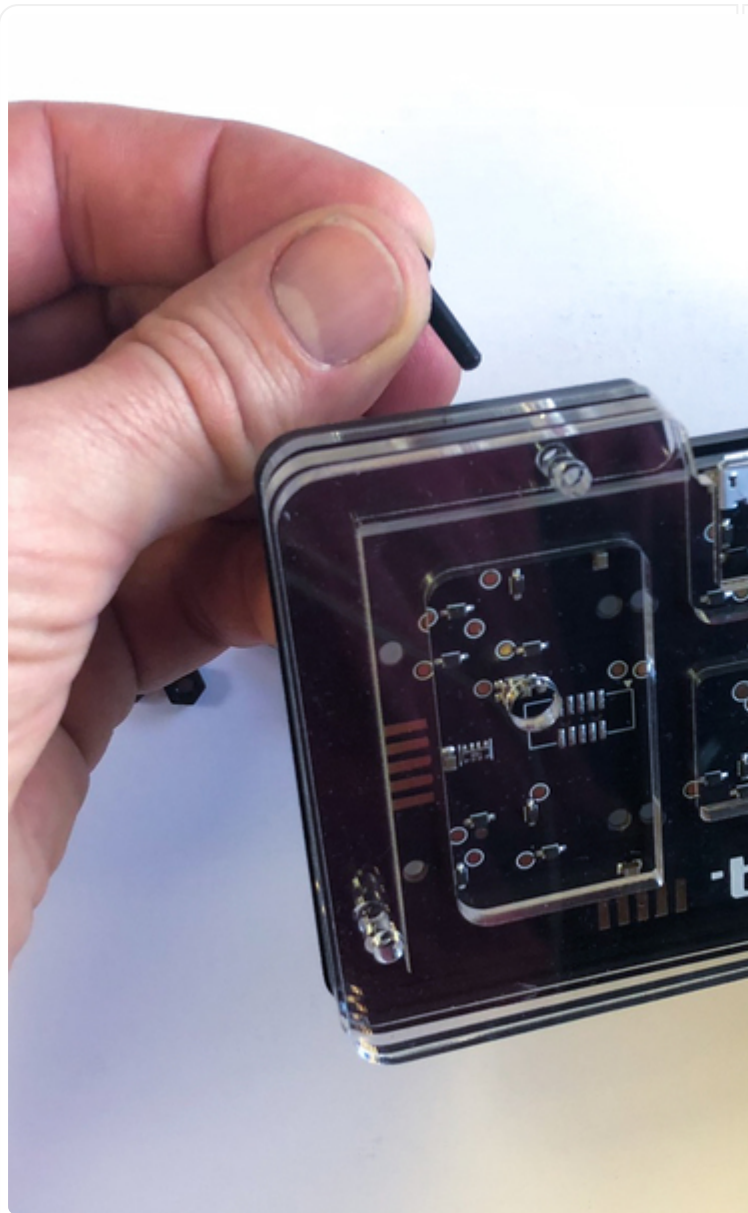


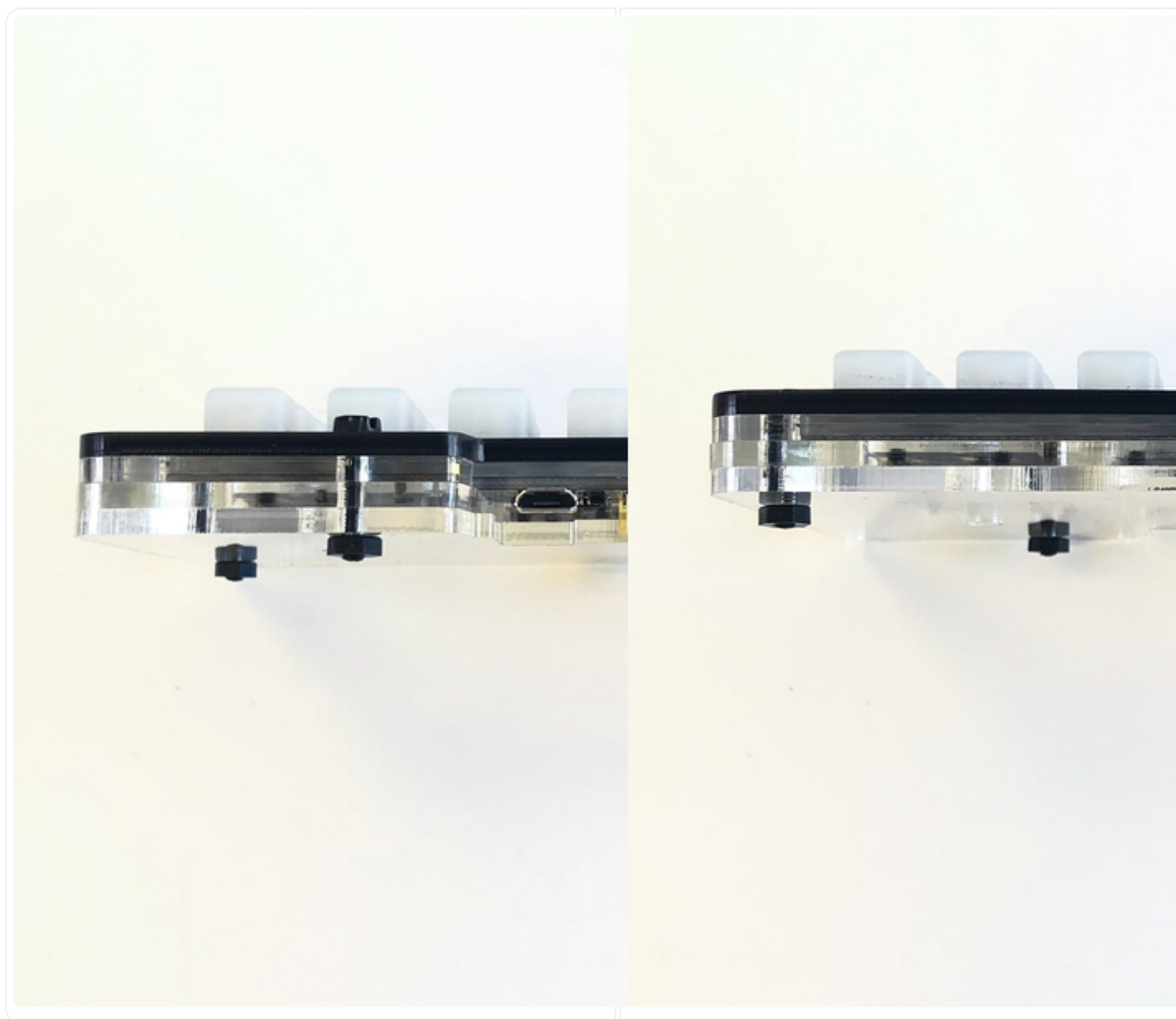
## Fasteners

Place the five nylon M3 screws through their holes from the front of the case toward the back, then thread the nuts onto them and finger tighten.









Your Trellis M4 is ready for use!



## Customization

Are you the sort of tinkerer who feels the urge to customize things? We knew it! In case you want to try your hand at building a custom enclosure for your Trellis M4, and make all your other DJ/Producer friends jealous, [here are the CAD files to get you started \(https://adafru.it/D0i\)](https://adafru.it/D0i). Use them on a laser cutter to create a unique wooden faceplate. Or, print the top piece out with ill graphics on a color printer, trim the button holes and screw holes, and make a cool skin. You might even want to use the templates as a reference for 3D modelling a case!

It's easy to press the Trellis M4's reset button through the case using a small object such as a headphone plug, chopstick, or 5mm LED. But, if you're doing a lot of development in Arduino and resetting a lot, you may want to make a button extender. Here's a file you can 3D print for that!

[resetbuttonTrellisM4.stl.zip](https://adafru.it/DIS)

<https://adafru.it/DIS>

---

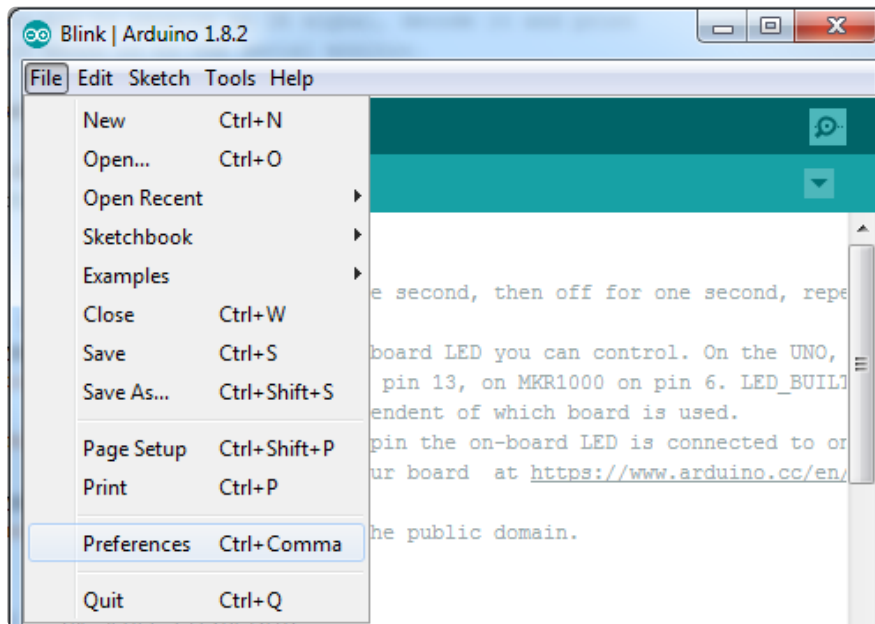
## Arduino IDE Setup

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

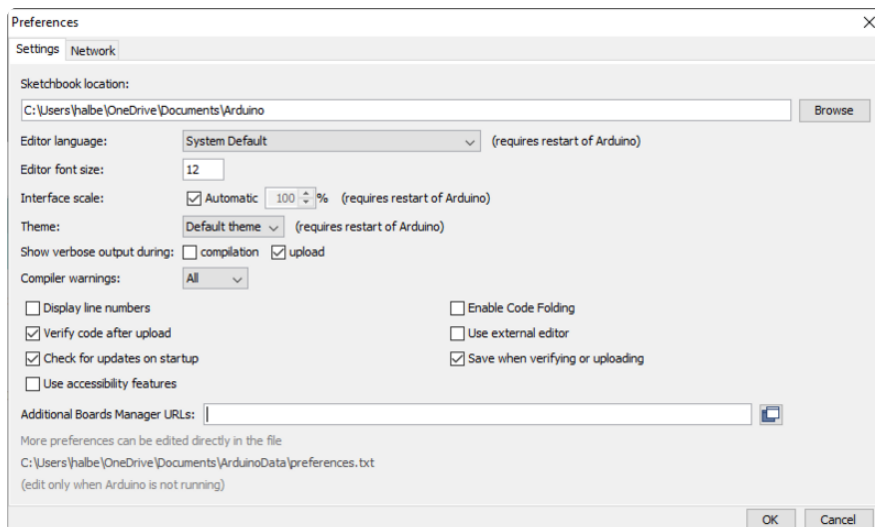
[Arduino IDE Download](https://adafru.it/f1P)

<https://adafru.it/f1P>

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.



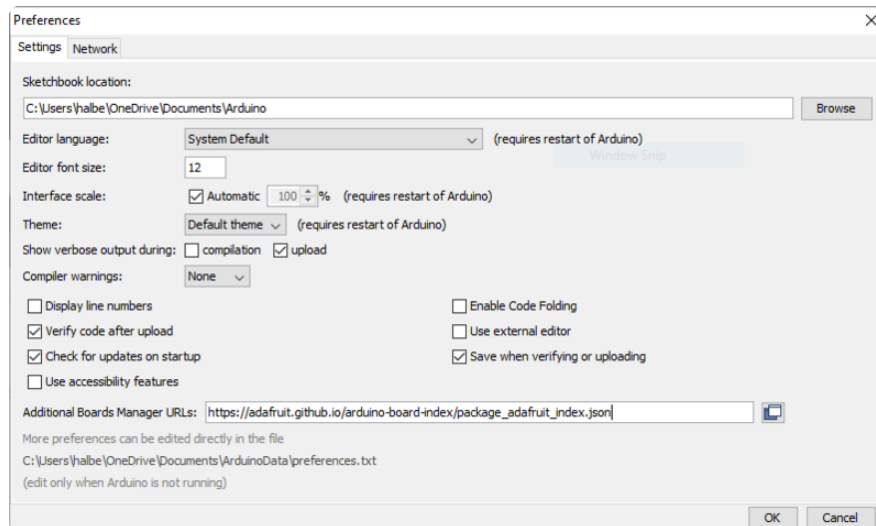
We will be adding a URL to the new **Additional Boards Manager URLs** option. The list of URLs is comma separated, and you will only have to add each URL once. New Adafruit boards and updates to existing boards will automatically be picked up by the Board Manager each time it is opened. The URLs point 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 \(https://adafru.it/f7U\)](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://adafruit.github.io/arduino-board-index/package\\_adafruit\\_index.json](https://adafruit.github.io/arduino-board-index/package_adafruit_index.json)



Here's a short description of each of the Adafruit supplied packages that will be available in the Board Manager when you add the URL:

- **Adafruit AVR Boards** - Includes support for Flora, Gemma, Feather 32u4, ItsyBitsy 32u4, Trinket, & Trinket Pro.
- **Adafruit SAMD Boards** - Includes support for Feather M0 and M4, Metro M0 and M4, ItsyBitsy M0 and M4, Circuit Playground Express, Gemma M0 and Trinket M0
- **Arduino Leonardo & Micro MIDI-USB** - This adds MIDI over USB support for the Flora, Feather 32u4, Micro and Leonardo using the [arcore project \(https://adafruit.it/eSI\)](https://adafruit.it/eSI).

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. Next we will look at installing boards with the Board Manager.

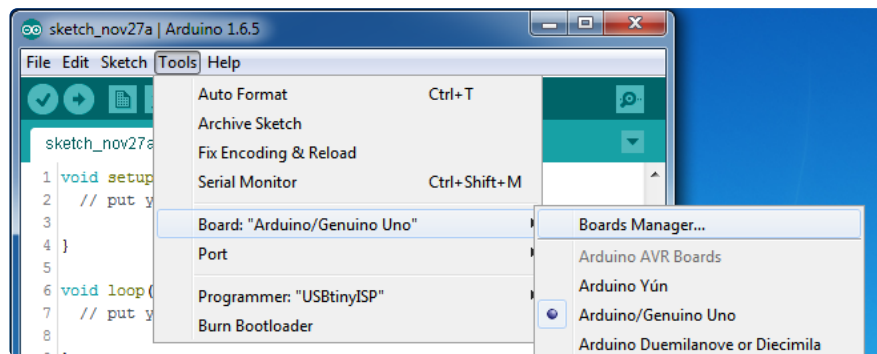
Now continue to the next step to actually install the board support package!

---

# Using with Arduino IDE

Adafruit boards that use ATSAM21 ("M0") or ATSAM51 ("M4") chips are easy to get working with the Arduino IDE. Most libraries (including the popular ones like NeoPixels and display) will work with those boards, especially devices & sensors that use I2C or SPI.

Now that you have added the appropriate URLs to the Arduino IDE preferences in the previous page, you can open the **Boards Manager** by navigating to the **Tools->Board** menu.



Once the Board Manager opens, click on the category drop down menu on the top left hand side of the window and select **All**. You will then be able to select and install the boards supplied by the URLs added to the preferences.

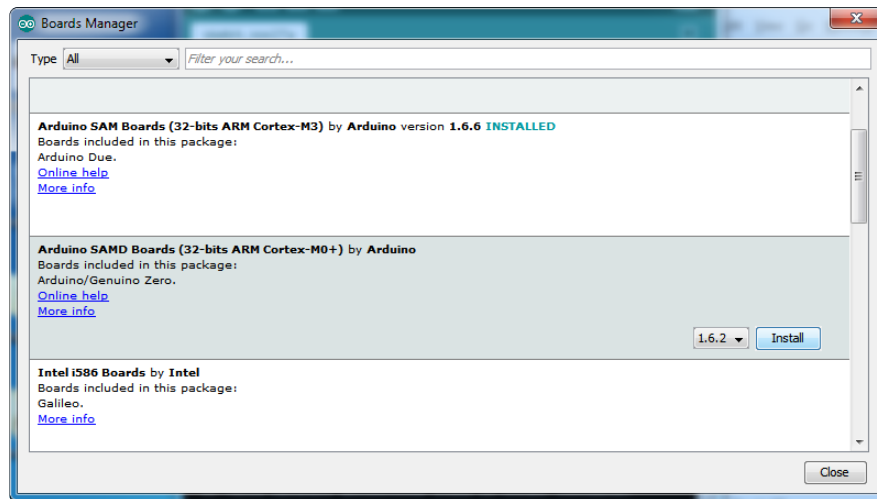
Remember you need **SETUP** the Arduino IDE to support our board packages - see the previous page on how to add adafruit's URL to the preferences

## Install SAMD Support

First up, install the latest **Arduino SAMD Boards** (version **1.6.11** or later)

You can type **Arduino SAMD** in the top search bar, then when you see the entry, click **Install**



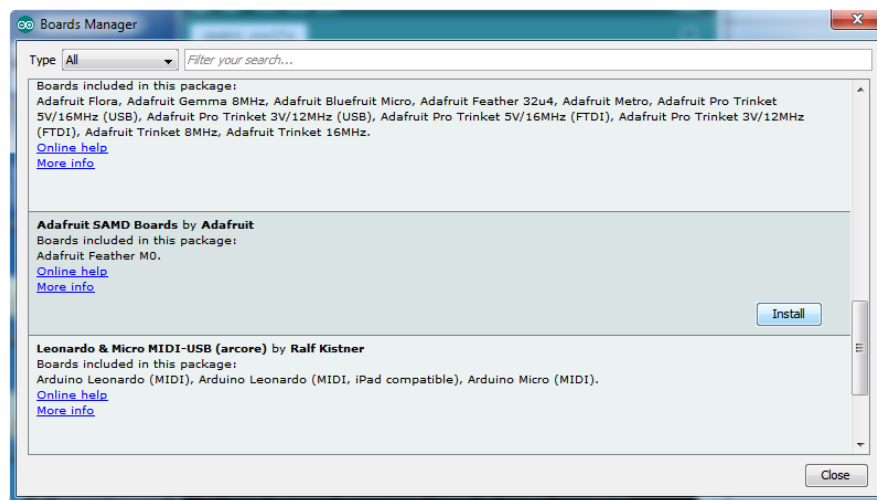


## Install Adafruit SAMD

Next you can install the Adafruit SAMD package to add the board file definitions

Make sure you have **Type All** selected to the left of the Filter your search... box

You can type **Adafruit SAMD** in the top search bar, then when you see the entry, click **Install**

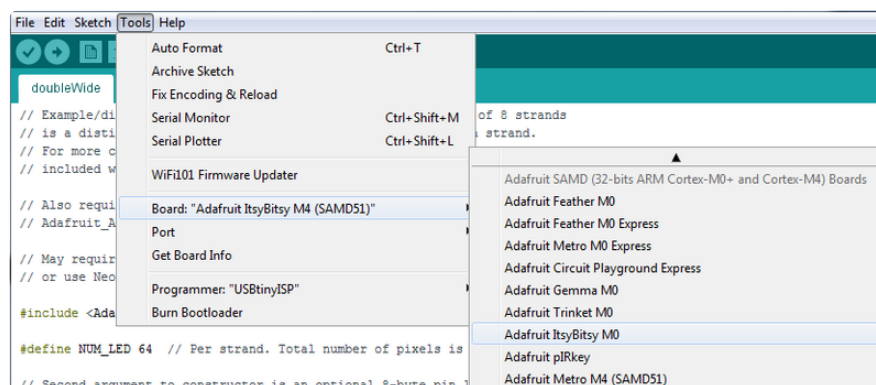


Quit and reopen the **Arduino IDE** to ensure that all of the boards are properly installed. You should now be able to select and upload to the new boards listed in the **Tools->Board** menu.

Select the matching board, the current options are:

- **Feather M0** (for use with any Feather M0 other than the Express)
- **Feather M0 Express**

- Metro M0 Express
- Circuit Playground Express
- Gemma M0
- Trinket M0
- QT Py M0
- ItsyBitsy M0
- Hallowing M0
- Crickit M0 (this is for direct programming of the Crickit, which is probably not what you want! For advanced hacking only)
- Metro M4 Express
- Grand Central M4 Express
- ItsyBitsy M4 Express
- Feather M4 Express
- Trellis M4 Express
- PyPortal M4
- PyPortal M4 Titano
- PyBadge M4 Express
- Metro M4 Airlift Lite
- PyGamer M4 Express
- MONSTER M4SK
- Hallowing M4
- MatrixPortal M4
- BLM Badge



## Windows 7 and 8.1

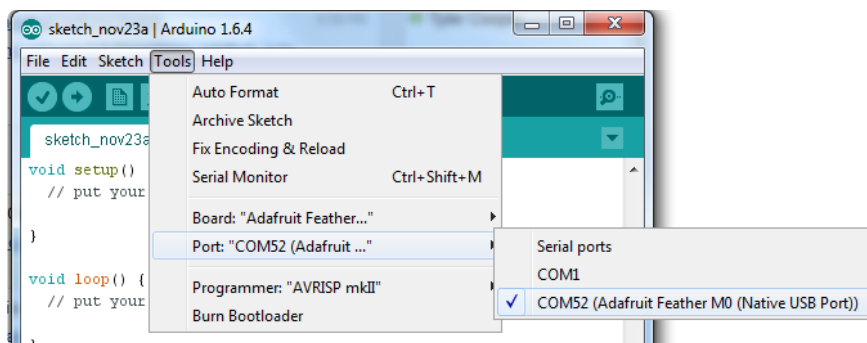
**Windows 7 and Windows 8.1** have reached end-of-life and are no longer supported. They required driver installation. A limited set of drivers is available for older boards, but drivers for most newer boards are not available.

# Blink

Now you can upload your first blink sketch!

Plug in the SAMD21 M0 or SAMD51 M4 board, and wait for it to be recognized by the OS (just takes a few seconds). It will create a serial/COM port, you can now select it from the drop-down, it'll even be 'indicated' as Trinket/Gemma/Metro/Feather/ItsyBitsy/QT Py/Trellis or whatever the board is named!

A few boards, such as the QT Py SAMD21, Trellis M4 Express, and certain Trinkey boards, do not have an onboard pin 13 LED. You can follow this section to practice uploading but you won't see an LED blink!



Now load up the Blink example

```
// the setup function runs once when you press reset or power the board
void setup() {
  // initialize digital pin 13 as an output.
  pinMode(13, OUTPUT);
}

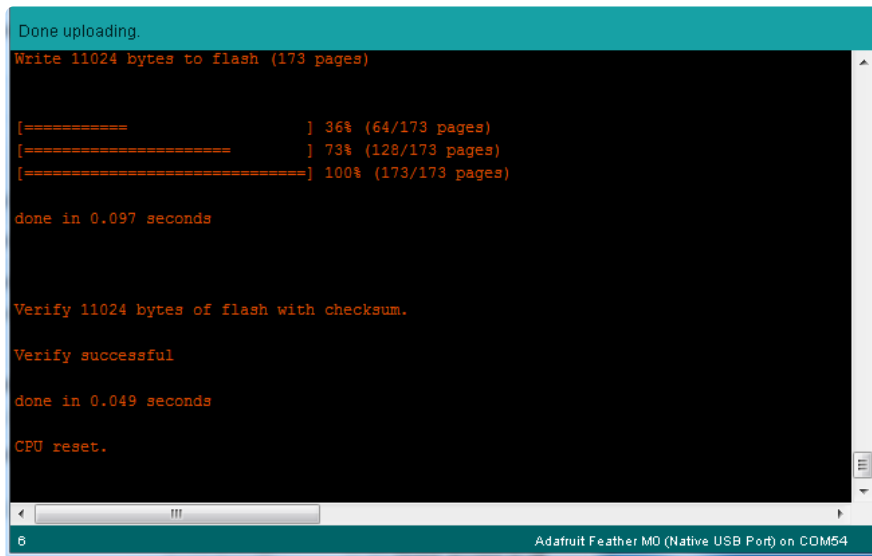
// the loop function runs over and over again forever
void loop() {
  digitalWrite(13, HIGH);   // turn the LED on (HIGH is the voltage level)
  delay(1000);              // wait for a second
  digitalWrite(13, LOW);    // turn the LED off by making the voltage LOW
  delay(1000);              // wait for a second
}
```

And click upload! That's it, you will be able to see the LED blink rate change as you adapt the **delay()** calls.

If you are having issues, make sure you selected the matching Board in the menu that matches the hardware you have in your hand.

## Successful Upload

If you have a successful upload, you'll get a bunch of red text that tells you that the device was found and it was programmed, verified & reset



```
Done uploading.
Write 11024 bytes to flash (173 pages)

[=====] 36% (64/173 pages)
[=====] 73% (128/173 pages)
[=====] 100% (173/173 pages)

done in 0.097 seconds

Verify 11024 bytes of flash with checksum.
Verify successful
done in 0.049 seconds

CPU reset.
```

The screenshot shows the Arduino IDE terminal window. The title bar indicates the board is 'Adafruit Feather M0 (Native USB Port) on COM54'. The terminal output shows the upload progress with red text on a black background. It reports writing 11024 bytes to flash in 0.097 seconds, followed by a successful verification in 0.049 seconds, and finally a CPU reset.

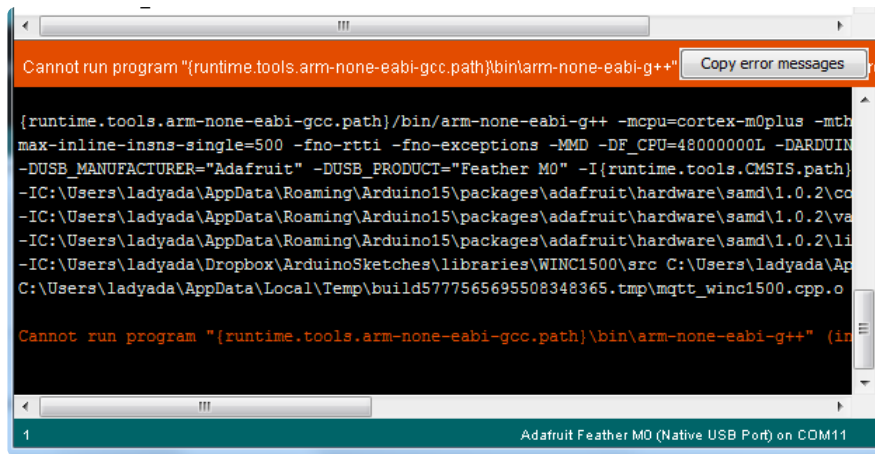
After uploading, you may see a message saying "Disk Not Ejected Properly" about the ...BOOT drive. You can ignore that message: it's an artifact of how the bootloader and uploading work.

## Compilation Issues

If you get an alert that looks like

Cannot run program "{runtime.tools.arm-none-eabi-gcc.path}\\bin\\arm-non-eabi-g++"

Make sure you have installed the **Arduino SAMD** boards package, you need both Arduino & Adafruit SAMD board packages

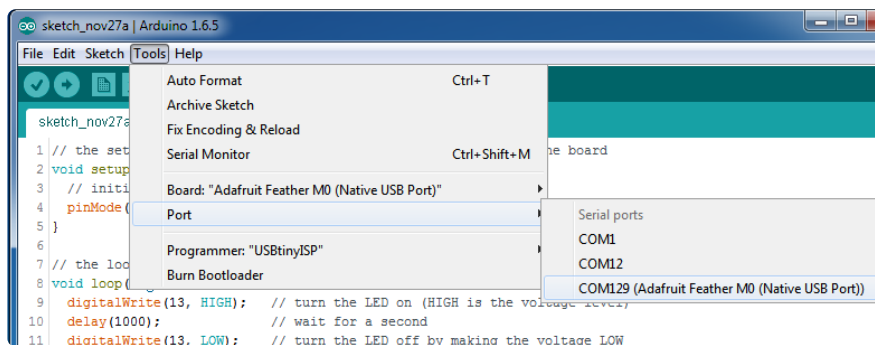


## Manually bootloading

If you ever get in a 'weird' spot with the bootloader, or you have uploaded code that crashes and doesn't auto-reboot into the bootloader, click the **RST** button **twice** (like a double-click) to get back into the bootloader.

The red LED will pulse and/or RGB LED will be green, so you know that its in bootloader mode.

Once it is in bootloader mode, you can select the newly created COM/Serial port and re-try uploading.



You may need to go back and reselect the 'normal' USB serial port next time you want to use the normal upload.

## Ubuntu & Linux Issue Fix

[Follow the steps for installing Adafruit's udev rules on this page. \(https://adafru.it/iOE\)](https://adafru.it/iOE)

---

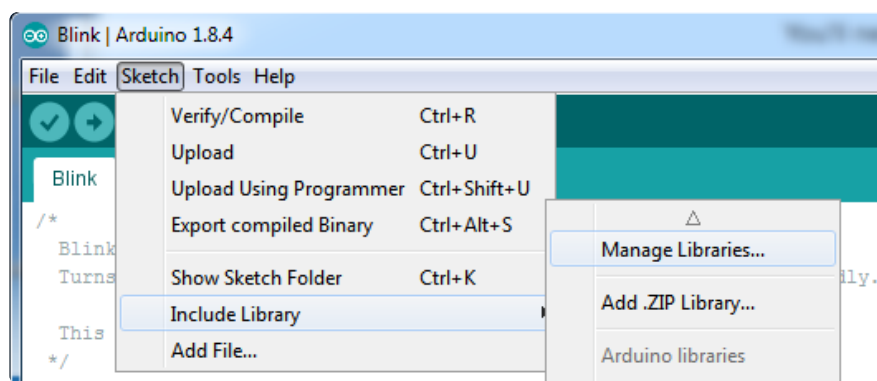
# Arduino Libraries

OK now that you have Arduino IDE set up, drivers installed if necessary and you've practiced uploading code, you can start installing all the Libraries we'll be using to program it.

There's a lot of libraries!

## Install Libraries

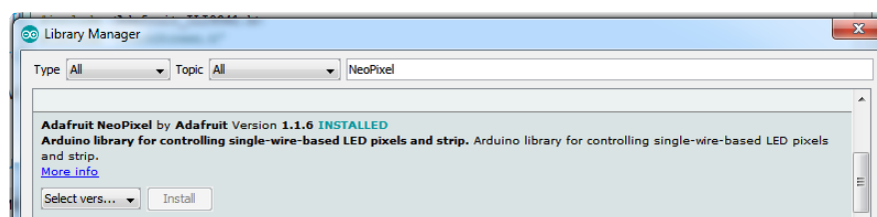
Open up the library manager...



And install the following libraries:

## Adafruit NeoPixel

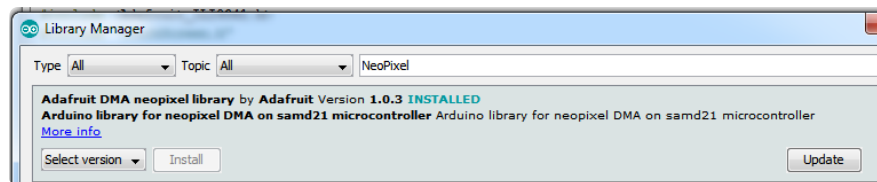
This will let you light up the LEDs on the front



## Adafruit DMA NeoPixel

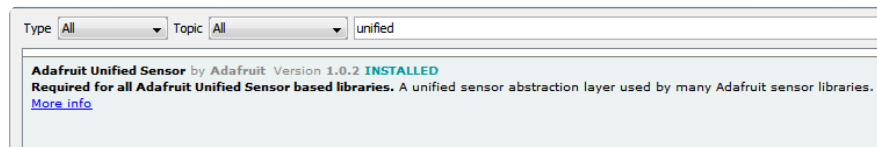
This adds a special NeoPixel library that uses DMA so the NeoPixel stuff happens without processor time taken.





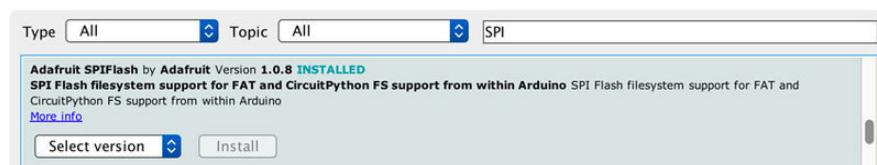
## Adafruit Unified Sensor

The underlying sensor library for ADXL343 support



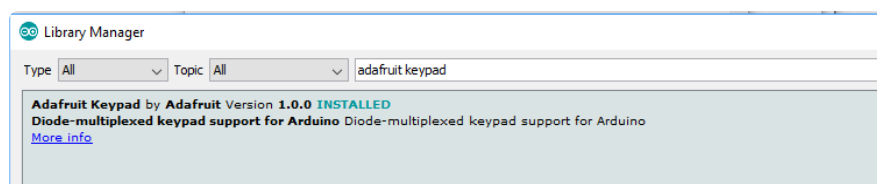
## Adafruit SPIFlash

This will let you read/write to the onboard FLASH memory with super-fast QSPI support



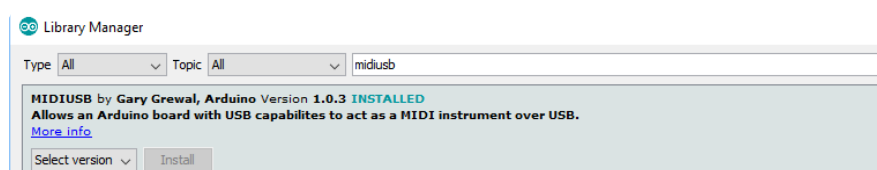
## Adafruit Keypad

Our Keypad support library (for reading the button matrix)



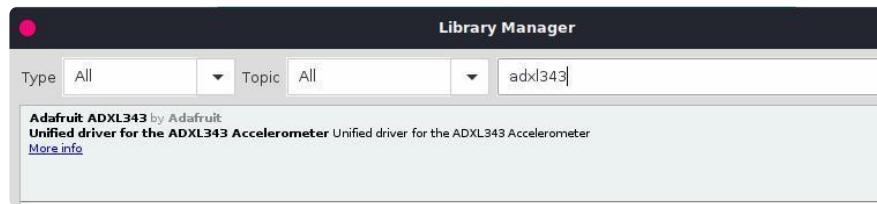
## MIDI USB

So you can have the Trellis M4 act like a MIDI device over USB



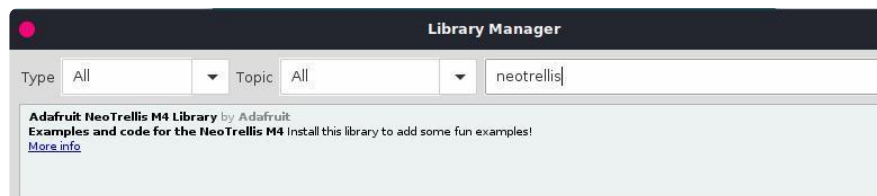
# ADXL343

The ADXL343 Library which provides accelerometer support



# NeoTrellis M4

The NeoTrellis\_M4 Library that handles MIDI, LEDs & button presses



# SdFat - Adafruit Fork

Our fork of the SdFat library provides read/write access to FAT16/FAT32 file systems on SD/SDHC flash cards.



# Audio - Adafruit Fork

Our fork of the Audio library provides a toolkit for building streaming audio projects.



---

# Adapting Sketches to M0 & M4

The ATSAM21 and 51 are very nice little chips, but fairly new as Arduino-compatible cores go. **Most** sketches & libraries will work but here's a collection of things we noticed.

The notes below cover a range of Adafruit M0 and M4 boards, but not every rule will apply to every board (e.g. Trinket and Gemma M0 do not have ARef, so you can skip the Analog References note!).

## Analog References

If you'd like to use the **ARef** pin for a non-3.3V analog reference, the code to use is `analogReference(AR_EXTERNAL)` (it's AR\_EXTERNAL not EXTERNAL)

## Pin Outputs & Pullups

The old-style way of turning on a pin as an input with a pullup is to use

```
pinMode(pin, INPUT)
digitalWrite(pin, HIGH)
```

This is because the pullup-selection register on 8-bit AVR chips is the same as the output-selection register.

For M0 & M4 boards, you can't do this anymore! Instead, use:

```
pinMode(pin, INPUT_PULLUP)
```

Code written this way still has the benefit of being backwards compatible with AVR. You don't need separate versions for the different board types.

## Serial vs SerialUSB

99.9% of your existing Arduino sketches use **Serial.print** to debug and give output. For the Official Arduino SAMD/M0 core, this goes to the Serial5 port, which isn't exposed on the Feather. The USB port for the Official Arduino M0 core is called **SerialUSB** instead.

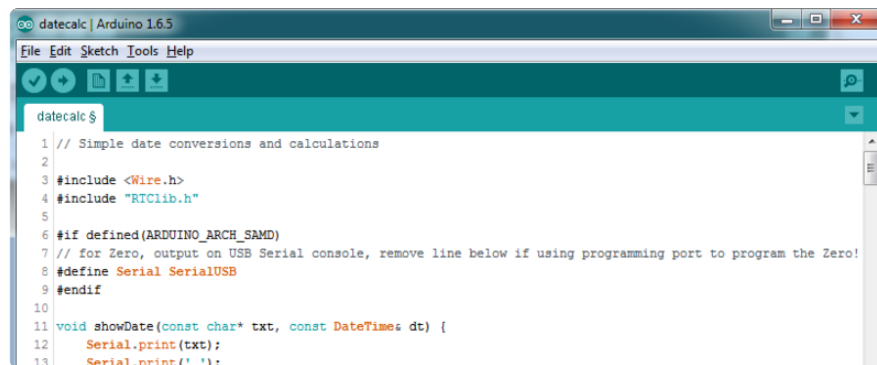
In the Adafruit M0/M4 Core, we fixed it so that **Serial goes to USB** so it will automatically work just fine.

However, on the off chance you are using the official Arduino SAMD core and not the Adafruit version (which really, we recommend you use our version because it's been tuned to our boards), and you want your Serial prints and reads to use the USB port, use **SerialUSB** instead of **Serial** in your sketch.

If you have existing sketches and code and you want them to work with the M0 without a huge find-replace, put

```
#if defined(ARDUINO_SAMD_ZERO) && defined(SERIAL_PORT_USBVIRTUAL)
// Required for Serial on Zero based boards
#define Serial SERIAL_PORT_USBVIRTUAL
#endif
```

right above the first function definition in your code. For example:



## AnalogWrite / PWM on Feather/Metro M0

After looking through the SAMD21 datasheet, we've found that some of the options listed in the multiplexer table don't exist on the specific chip used in the Feather M0.

For all SAMD21 chips, there are two peripherals that can generate PWM signals: The Timer/Counter (TC) and Timer/Counter for Control Applications (TCC). Each SAMD21 has multiple copies of each, called 'instances'.

Each TC instance has one count register, one control register, and two output channels. Either channel can be enabled and disabled, and either channel can be inverted. The pins connected to a TC instance can output identical versions of the same PWM waveform, or complementary waveforms.

Each TCC instance has a single count register, but multiple compare registers and output channels. There are options for different kinds of waveform, interleaved switching, programmable dead time, and so on.

The biggest members of the SAMD21 family have five TC instances with two 'waveform output' (WO) channels, and three TCC instances with eight WO channels:

- TC[0-4],WO[0-1]
- TCC[0-2],WO[0-7]

And those are the ones shown in the datasheet's multiplexer tables.

The SAMD21G used in the Feather M0 only has three TC instances with two output channels, and three TCC instances with eight output channels:

- TC[3-5],WO[0-1]
- TCC[0-2],WO[0-7]

Tracing the signals to the pins broken out on the Feather M0, the following pins can't do PWM at all:

- **Analog pin A5**

The following pins can be configured for PWM without any signal conflicts as long as the SPI, I2C, and UART pins keep their protocol functions:

- **Digital pins 5, 6, 9, 10, 11, 12, and 13**
- **Analog pins A3 and A4**

If only the SPI pins keep their protocol functions, you can also do PWM on the following pins:

- **TX and SDA (Digital pins 1 and 20)**

## analogWrite() PWM range

On AVR, if you set a pin's PWM with `analogWrite(pin, 255)` it will turn the pin fully HIGH. On the ARM cortex, it will set it to be 255/256 so there will be very slim but

still-existing pulses-to-0V. If you need the pin to be fully on, add test code that checks if you are trying to `analogWrite(pin, 255)` and, instead, does a `digitalWrite(pin, HIGH)`

## analogWrite() DAC on A0

If you are trying to use `analogWrite()` to control the DAC output on **A0**, make sure you do **not** have a line that sets the pin to output. **Remove:** `pinMode(A0, OUTPUT)`.

## serialEvent() and serialEvent1()

`serialEvent()` and `serialEvent1()` do not work (<https://adafru.it/19NA>) on any non-AVR Arduino boards. Use `Serial.available()` instead.

## Missing header files

There might be code that uses libraries that are not supported by the M0 core. For example if you have a line with

```
#include <util/delay.h>
```

you'll get an error that says

```
fatal error: util/delay.h: No such file or directory
#include <util/delay.h>
^
compilation terminated.
Error compiling.
```

In which case you can simply locate where the line is (the error will give you the file name and line number) and 'wrap it' with `#ifdef`'s so it looks like:

```
#if !defined(ARDUINO_ARCH_SAM) && !defined(ARDUINO_ARCH_SAMD) && !
defined(ESP8266) && !defined(ARDUINO_ARCH_STM32F2)
#include <util/delay.h>
#endif
```

The above will also make sure that header file isn't included for other architectures

If the `#include` is in the arduino sketch itself, you can try just removing the line.



# Bootloader Launching

For most other AVR's, clicking **reset** while plugged into USB will launch the bootloader manually, the bootloader will time out after a few seconds. For the M0/M4, you'll need to **double click** the button. You will see a pulsing red LED to let you know you're in bootloader mode. Once in that mode, it won't time out! Click reset again if you want to go back to launching code.

## Aligned Memory Access

This is a little less likely to happen to you but it happened to me! If you're used to 8-bit platforms, you can do this nice thing where you can typecast variables around. e.g.

```
uint8_t mybuffer[4];  
float f = (float)mybuffer;
```

You can't be guaranteed that this will work on a 32-bit platform because **mybuffer** might not be aligned to a 2 or 4-byte boundary. The ARM Cortex-M0 can only directly access data on 16-bit boundaries (every 2 or 4 bytes). Trying to access an odd-boundary byte (on a 1 or 3 byte location) will cause a Hard Fault and stop the MCU. Thankfully, there's an easy work around ... just use `memcpy`!

```
uint8_t mybuffer[4];  
float f;  
memcpy(&f, mybuffer, 4)
```

## Floating Point Conversion

Like the AVR Arduinos, the M0 library does not have full support for converting floating point numbers to ASCII strings. Functions like `sprintf` will not convert floating point. Fortunately, the standard AVR-LIBC library includes the `dtostrf` function which can handle the conversion for you.

Unfortunately, the M0 run-time library does not have `dtostrf`. You may see some references to using `#include <avr/dtostrf.h>` to get `dtostrf` in your code. And while it will compile, it does **not** work.

Instead, check out this thread to find a working `dtostrf` function you can include in your code:

<http://forum.arduino.cc/index.php?topic=368720.0> (<https://adafru.it/IFS>)

## How Much RAM Available?

The ATSAM21G18 has 32K of RAM, but you still might need to track it for some reason. You can do so with this handy function:

```
extern "C" char *sbrk(int i);

int FreeRam () {
  char stack_dummy = 0;
  return &stack_dummy - sbrk(0);
}
```

Thx to <http://forum.arduino.cc/index.php?topic=365830.msg2542879#msg2542879> (<https://adafru.it/m6D>) for the tip!

## Storing data in FLASH

If you're used to AVR, you've probably used **PROGMEM** to let the compiler know you'd like to put a variable or string in flash memory to save on RAM. On the ARM, it's a little easier, simply add **const** before the variable name:

```
const char str[] = "My very long string";
```

That string is now in FLASH. You can manipulate the string just like RAM data, the compiler will automatically read from FLASH so you don't need special progmem-knowledgeable functions.

You can verify where data is stored by printing out the address:

```
Serial.print("Address of str $"); Serial.println((int)&str, HEX);
```

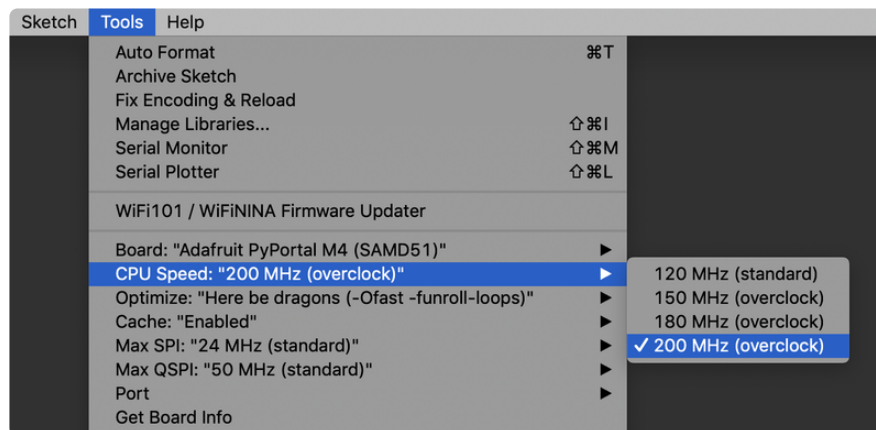
If the address is \$2000000 or larger, it's in SRAM. If the address is between \$0000 and \$3FFFF Then it is in FLASH

## Pretty-Printing out registers

There's a lot of registers on the SAMD21, and you often are going through ASF or another framework to get to them. So having a way to see exactly what's going on is handy. This library from drewfish will help a ton!

## M4 Performance Options

As of version 1.4.0 of the Adafruit SAMD Boards package in the Arduino Boards Manager, some options are available to wring extra performance out of M4-based devices. These are in the Tools menu.



**All of these performance tweaks involve a degree of uncertainty.** There's no guarantee of improved performance in any given project, and some may even be detrimental, failing to work in part or in whole. If you encounter trouble, **select the default performance settings** and re-upload.

Here's what you get and some issues you might encounter...

### CPU Speed (overclocking)

This option lets you adjust the microcontroller core clock...the speed at which it processes instructions...beyond the official datasheet specifications.

Manufacturers often rate speeds conservatively because such devices are marketed for harsh industrial environments...if a system crashes, someone could lose a limb or worse. But most creative tasks are less critical and operate in more comfortable settings, and we can push things a bit if we want more speed.

There is a small but nonzero chance of code **locking up** or **failing to run** entirely. If this happens, try **dialing back the speed by one notch and re-upload**, see if it's more stable.

Much more likely, **some code or libraries may not play well** with the nonstandard CPU speed. For example, currently the NeoPixel library assumes a 120 MHz CPU speed and won't issue the correct data at other settings (this will be worked on). Other libraries may exhibit similar problems, usually anything that strictly depends on CPU timing...you might encounter problems with audio- or servo-related code depending how it's written. **If you encounter such code or libraries, set the CPU speed to the default 120 MHz and re-upload.**

## Optimize

There's usually more than one way to solve a problem, some more resource-intensive than others. Since Arduino got its start on resource-limited AVR microcontrollers, the C++ compiler has always aimed for the **smallest compiled program size**. The "Optimize" menu gives some choices for the compiler to take different and often faster approaches, at the expense of slightly larger program size...with the huge flash memory capacity of M4 devices, that's rarely a problem now.

The "**Small**" setting will compile your code like it always has in the past, aiming for the smallest compiled program size.

The "**Fast**" setting invokes various speed optimizations. The resulting program should produce the same results, is slightly larger, and usually (but not always) noticeably faster. It's worth a shot!

"**Here be dragons**" invokes some more intensive optimizations...code will be larger still, faster still, but there's a possibility these optimizations could cause unexpected behaviors. Some code may not work the same as before. Hence the name. Maybe you'll discover treasure here, or maybe you'll sail right off the edge of the world.

Most code and libraries will continue to function regardless of the optimizer settings. If you do encounter problems, **dial it back one notch and re-upload.**

## Cache

This option allows a small collection of instructions and data to be accessed more quickly than from flash memory, boosting performance. It's enabled by default and should work fine with all code and libraries. But if you encounter some esoteric situation, the cache can be disabled, then recompile and upload.

## Max SPI and Max QSPI

These should probably be left at their defaults. They're present mostly for our own experiments and can cause **serious headaches**.

Max SPI determines the clock source for the M4's SPI peripherals. Under normal circumstances this allows transfers up to 24 MHz, and should usually be left at that setting. But...if you're using write-only SPI devices (such as TFT or OLED displays), this option lets you drive them faster (we've successfully used 60 MHz with some TFT screens). The caveat is, if using any read/write devices (such as an SD card), this will not work at all...SPI reads absolutely max out at the default 24 MHz setting, and anything else will fail. **Write = OK. Read = FAIL**. This is true even if your code is using a lower bitrate setting...just having the different clock source prevents SPI reads.

Max QSPI does similarly for the extra flash storage on M4 "Express" boards. Very few Arduino sketches access this storage at all, let alone in a bandwidth-constrained context, so this will benefit next to nobody. Additionally, due to the way clock dividers are selected, this will only provide some benefit when certain "CPU Speed" settings are active. Our [PyPortal Animated GIF Display \(https://adafru.it/EkO\)](https://adafru.it/EkO) runs marginally better with it, if using the QSPI flash.

## Enabling the Buck Converter on some M4 Boards

If you want to reduce power draw, some of our boards have an inductor so you can use the 1.8V buck converter instead of the built in linear regulator. If the board does have an inductor (see the schematic) you can add the line `SUPC->VREG.bit.SEL = 1;` to your code to switch to it. Note it will make ADC/DAC reads a bit noisier so we don't use it by default. [You'll save ~4mA \(https://adafru.it/FOH\)](https://adafru.it/FOH).

---

## Arduino Examples

Once you have the Arduino libraries installed you can try these examples!

All of our example code lives in the github repo at [https://github.com/adafruit/Adafruit\\_NeoTrellisM4](https://github.com/adafruit/Adafruit_NeoTrellisM4) (<https://adafru.it/CW5>)

[You can download the code by clicking here \(https://adafru.it/CW6\)](https://adafru.it/CW6)



# NeoPixel Test

[Load this example \(https://adafru.it/CW7\)](https://adafru.it/CW7) to display a variety of colors and effects on the NeoPixels, good to test that they're all working and shining as expected. You need to have the Adafruit NeoPixel library installed first

# Keypad Test

[Load this example \(https://adafru.it/CW8\)](https://adafru.it/CW8) to turn on NeoPixels whenever you press a button. Press again to turn it off. Good for checking button presses, elastomers and NeoPixels.

# MIDI USB Test

Your NeoTrellis M4 can act as a 32-button MIDI board. [This example \(https://adafru.it/CW9\)](https://adafru.it/CW9) will send Note On and Note Off reports for every button when pressed and released.

# Audio Library Test

The Audio library (originally by PJRC) allows the creation of waveforms and filters dynamically by the chip! [You can try this out with the simple synth example \(https://adafru.it/CWa\)](https://adafru.it/CWa), which will give you an octave each of four different voices. The playback is polyphonic, try pressing multiple buttons!

# Microphone Feed-thru Test

[This example will take microphone input and then pipe it out the headphones \(https://adafru.it/CWb\)](https://adafru.it/CWb). Simple but good for checking that your headset is wired correctly.

# Microphone FFT Test

You can take audio input, then display it on the LEDs - [here's an FFT example \(https://adafru.it/CWc\)](https://adafru.it/CWc). You don't get a lot of bins but it can make for a neat audio effect!

---

# What is CircuitPython?

CircuitPython is a programming language designed to simplify experimenting and learning to program on low-cost microcontroller boards. It makes getting started easier than ever with no upfront desktop downloads needed. Once you get your board set up, open any text editor, and get started editing code. It's that simple.



## CircuitPython is based on Python

Python is the fastest growing programming language. It's taught in schools and universities. It's a high-level programming language which means it's designed to be easier to read, write and maintain. It supports modules and packages which means it's easy to reuse your code for other projects. It has a built in interpreter which means there are no extra steps, like compiling, to get your code to work. And of course, Python is Open Source Software which means it's free for anyone to use, modify or improve upon.

CircuitPython adds hardware support to all of these amazing features. If you already have Python knowledge, you can easily apply that to using CircuitPython. If you have no previous experience, it's really simple to get started!



# Why would I use CircuitPython?

CircuitPython is designed to run on microcontroller boards. A microcontroller board is a board with a microcontroller chip that's essentially an itty-bitty all-in-one computer. The board you're holding is a microcontroller board! CircuitPython is easy to use because all you need is that little board, a USB cable, and a computer with a USB connection. But that's only the beginning.

Other reasons to use CircuitPython include:

- **You want to get up and running quickly.** Create a file, edit your code, save the file, and it runs immediately. There is no compiling, no downloading and no uploading needed.
- **You're new to programming.** CircuitPython is designed with education in mind. It's easy to start learning how to program and you get immediate feedback from the board.
- **Easily update your code.** Since your code lives on the disk drive, you can edit it whenever you like, you can also keep multiple files around for easy experimentation.
- **The serial console and REPL.** These allow for live feedback from your code and interactive programming.
- **File storage.** The internal storage for CircuitPython makes it great for data-logging, playing audio clips, and otherwise interacting with files.
- **Strong hardware support.** CircuitPython has builtin support for microcontroller hardware features like digital I/O pins, hardware buses (UART, I2C, SPI), audio I/O, and other capabilities. There are also many libraries and drivers for sensors, breakout boards and other external components.
- **It's Python!** Python is the fastest-growing programming language. It's taught in schools and universities. CircuitPython is almost-completely compatible with Python. It simply adds hardware support.

This is just the beginning. CircuitPython continues to evolve, and is constantly being updated. Adafruit welcomes and encourages feedback from the community, and incorporate it into the development of CircuitPython. That's the core of the open source concept. This makes CircuitPython better for you and everyone who uses it!

---

## CircuitPython

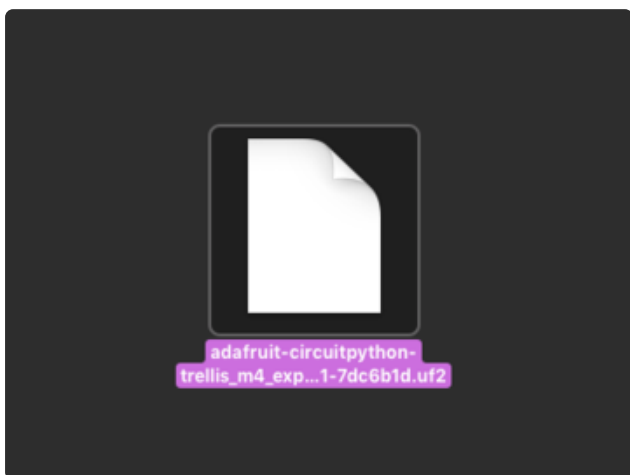
[CircuitPython \(https://adafru.it/tB7\)](https://adafru.it/tB7) is designed to simplify experimentation and education on low-cost microcontrollers. It makes it easier than ever to get prototyping

by requiring no upfront desktop software downloads. Simply copy and edit files on the flash drive named **CIRCUITPY** which appears when NeoTrellis is plugged into a computer to iterate.

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

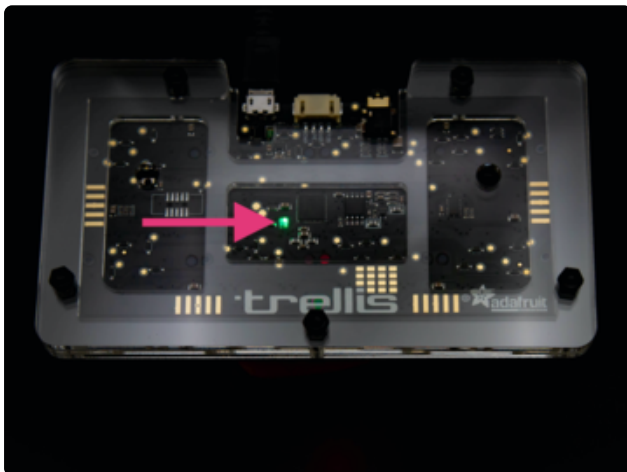
Download the latest version of  
CircuitPython for this board via  
CircuitPython.org

<https://adafru.it/Em6>



Click the link above and download the latest UF2 file.

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

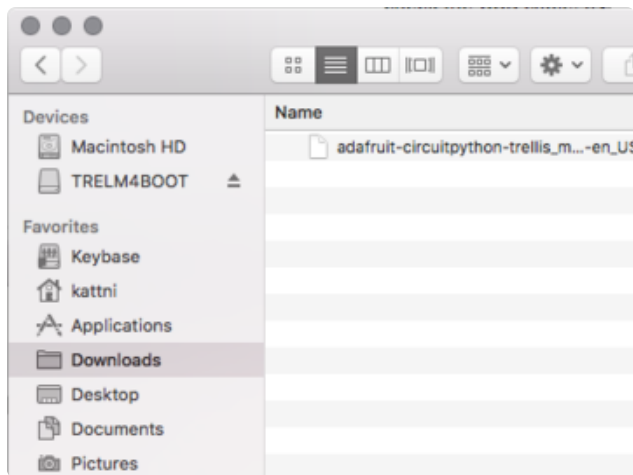


Plug your NeoTrellis M4 Express into your computer using a known-good USB cable.

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

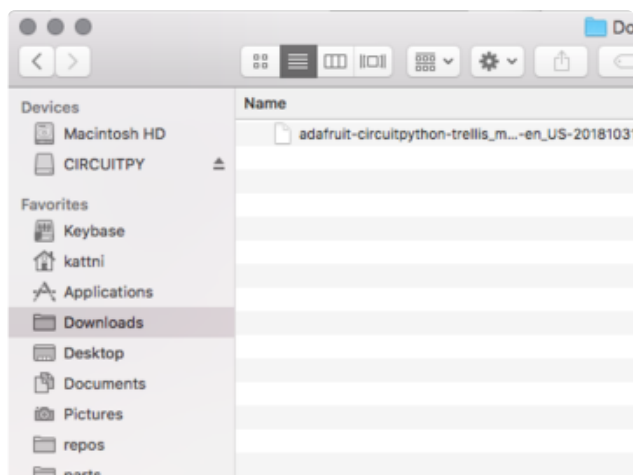
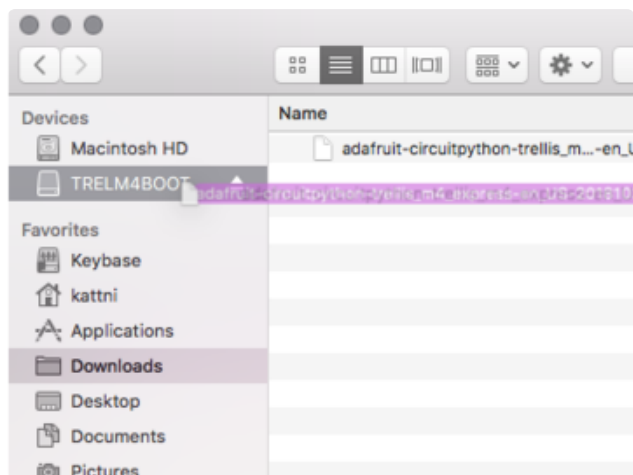
Double-click the **Reset** button next to the USB connector on your board, and you will see the status DotStar RGB LED turn green. If it turns red, check the USB cable, try another USB port, etc.

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



You will see a new flash disk drive appear called **TRELM4BOOT**.

Drag the `adafruit_circuitpython_etc.uf2` file to **TRELM4BOOT**.



The LED will flash. Then, the **TRELM4BOOT** drive will disappear and a new disk drive called **CIRCUITPY** will appear.

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

You can then unzip this download and drag the files onto **CIRCUITPY** drive to get back to the default drum machine

**neotrellis\_m4\_default\_files.zip**

<https://adafru.it/Djj>



---

# Creating and Editing Code

One of the best things about CircuitPython is how simple it is to get code up and running. In this section, we're going to cover how to load libraries, and create and edit your first CircuitPython program.

## CircuitPython Libraries

The first thing you'll need to do is make sure you have the [CircuitPython library bundle \(https://adafru.it/y8E\)](https://adafru.it/y8E) installed. Each CircuitPython program you run needs to have a lot of information to work. The reason CircuitPython is so simple to use is that most of that information is stored in other files and works in the background. These files are called **libraries**. Some of them are built into CircuitPython. Others are stored on your **CIRCUITPY** drive in a folder called **lib**. Part of what makes CircuitPython so awesome is its ability to store code separately from the firmware itself. Storing code separately from the firmware makes it easier to update both the code you write and the libraries you depend.

Your board may ship with a **lib** folder already present. We're always updating and improving libraries, so **it's best to download the latest version** and replace the version that it shipped with!

Click on the link below to go to the CircuitPython Library Bundle Releases page. **Download the 4.x bundle.** You always want to download the bundle that matches the version of CircuitPython you're using.

**CircuitPython Library Bundle Latest Release**

<https://adafru.it/y8E>

Unzip the file you downloaded, open the folder, and then copy the **lib** folder to your **CIRCUITPY** drive. For a more detailed explanation, please see [the CircuitPython Libraries page in this guide \(https://adafru.it/CYo\)](https://adafru.it/CYo).

That's all there is to installing the CircuitPython library bundle!

## Choosing an Editor

To create and edit code, all you'll need is an editor. There are many options. There are basic text editors built into every operating system such as Notepad on Windows,

TextEdit on Mac, and gedit on Linux. However, many of these editors don't write back changes immediately to files that you edit. That can cause problems when using CircuitPython. If you choose to use one of these editors, make sure you do "Eject" or "Safe Remove" on Windows or "sync" on Linux after writing a file. (This is not a problem on MacOS.) However, here are some editors that write the file completely on save:

- [emacs](https://adafru.it/xNA) (<https://adafru.it/xNA>) is also an editor that will [fully write files on save](https://adafru.it/Be7) (<https://adafru.it/Be7>)
- [vim](https://adafru.it/ek9) (<https://adafru.it/ek9>) / [vi](#) safely writes all changes
- [Sublime Text](https://adafru.it/xNB) (<https://adafru.it/xNB>) safely writes all changes
- The [PyCharm IDE](https://adafru.it/xNC) (<https://adafru.it/xNC>) is safe if "Safe Write" is turned on in Settings->System Settings->Synchronization (on by default).
- If you are using [Atom](https://adafru.it/fMG) (<https://adafru.it/fMG>), install the [fsync-on-save package](https://adafru.it/E9m) (<https://adafru.it/E9m>) so that it will always write out all changes to files on **CIRCUITPY**.
- [Visual Studio Code](https://adafru.it/Be9) (<https://adafru.it/Be9>) appears to safely write all changes
- [gedit](#) on Linux appears to safely write all changes

## Creating Code

To create your first program, create a file called **code.py** on your **CIRCUITPY** drive using your editor. Copy and paste the following code into your **code.py** file and save. The first NeoPixel will start blinking red!

```
import time
import adafruit_trellism4

trellis = adafruit_trellism4.TrellisM4Express()

while True:
    trellis.pixels[0, 0] = (100, 0, 0)
    time.sleep(0.5)
    trellis.pixels[0, 0] = (0, 0, 0)
    time.sleep(0.5)
```

There's one warning we have to give you before we continue...

**Don't Click Reset or Unplug!**

The CircuitPython code on your board detects when the files are changed or written and will automatically re-start your code. This makes coding very fast because you save, and it re-runs.

However, you must wait until the file is done being saved before unplugging or resetting your board! On Windows using some editors this can sometimes take up to 90 seconds, on Linux it can take 30 seconds to complete because the text editor does not save the file completely. Mac OS does not seem to have this delay, which is nice!

This is really important to be aware of. If you unplug or reset the board before your computer finishes writing the file to your board, you can corrupt the drive. If this happens, you may lose the code you've written, so it's important to backup your code to your computer regularly.

You can avoid this in two ways:

- Use an editor that writes fully on save, like the editors suggested in the list above.
- Always eject or sync the after writing. On Windows, you can **Eject** or **Safe Remove** the **CIRCUITPY** drive. It won't actually eject, but it will force the operating system to save your file to disk. On Linux, use the **sync** command in a terminal to force the write to disk.

## Editing Code

Now that you've created and run your CircuitPython program, let's take a look at editing it. We'll make a simple change. Change the first **0.5** to **0.1**. The code should look like this:

```
import time
import adafruit_trellism4

trellis = adafruit_trellism4.TrellisM4Express()

while True:
    trellis.pixels[0, 0] = (100, 0, 0)
    time.sleep(0.1)
    trellis.pixels[0, 0] = (0, 0, 0)
    time.sleep(0.5)
```

Leave the rest of the code as-is. Save your file. See what happens to the NeoPixel on your board? Something changed! Do you know why? Let's find out!

## Exploring Your First CircuitPython Program

First, we'll take a look at the code we're editing.

Here is the original code again:

```
import time
import adafruit_trellism4

trellis = adafruit_trellism4.TrellisM4Express()

while True:
    trellis.pixels[0, 0] = (100, 0, 0)
    time.sleep(0.5)
    trellis.pixels[0, 0] = (0, 0, 0)
    time.sleep(0.5)
```

## Imports & Libraries

Each CircuitPython program you run needs to have a lot of information to work. The reason CircuitPython is so simple to use is that most of that information is stored in other files and works in the background. These files are called **libraries**. Some of them are built into CircuitPython. Others are stored on your CIRCUITPY drive in a folder called **lib**.

```
import time
import adafruit_trellism4
```

The `import` statements tells the board that you're going to use a particular library in your code. In this example, we imported two libraries: `time` and `adafruit_trellism4`. `time` lets you pass time by 'sleeping', and `adafruit_trellism4` lets you interact with the buttons and NeoPixels on the front of your board.

## Setting Up The Trellis M4

The next line sets up Trellis M4 library.

```
trellis = adafruit_trellism4.TrellisM4Express()
```

We assign `trellis` to allow us to use the features of the Trellis M4 library in our code.

## Loop-de-loops

The third section starts with a `while` statement. `while True:` essentially means, "forever do the following:". `while True:` creates a loop. Code will loop "while" the condition is "true" (vs. false), and as `True` is never False, the code will loop forever. All code that is indented under `while True:` is "inside" the loop.

Inside our loop, we have four items:

```
while True:
    trellis.pixels[0, 0] = (100, 0, 0)
    time.sleep(0.5)
    trellis.pixels[0, 0] = (0, 0, 0)
    time.sleep(0.5)
```

First, we have `trellis.pixels[0, 0] = (100, 0, 0)`. This line tells the first NeoPixel to turn on red. On the next line, we have `time.sleep(0.5)`. This line is telling CircuitPython to pause running code for 0.5 seconds. Since this is between turning the NeoPixel red and off, the led will be on for 0.5 seconds.

The next two lines are similar. `trellis.pixels[0, 0] = (100, 0, 0)` tells the NeoPixel to turn off, and `time.sleep(0.5)` tells CircuitPython to pause for another 0.5 seconds. This occurs between turning the NeoPixel off and back on so the NeoPixel will be off for 0.5 seconds too.

Then the loop will begin again, and continue to do so as long as the code is running!

So, when you changed the first `0.5` to `0.1`, you decreased the amount of time that the code leaves the NeoPixel on. So it blinks on really quickly before turning off!

Great job! You've edited code in a CircuitPython program!

## More Changes

We don't have to stop there! Let's keep going. Change the second `0.5` to `0.1` so it looks like this:

```
while True:
    trellis.pixels[0, 0] = (100, 0, 0)
    time.sleep(0.1)
    trellis.pixels[0, 0] = (0, 0, 0)
    time.sleep(0.1)
```

Now it blinks really fast! You decreased the both time that the code leaves the NeoPixel on and off!

Now try increasing both of the `0.1` to `1`. Your NeoPixel will blink much more slowly because you've increased the amount of time that the NeoPixel is turned on and off.

Well done! You're doing great! You're ready to start into new examples and edit them to see what happens! These were simple changes, but major changes are done using



the same process. Make your desired change, save it, and get the results. That's really all there is to it!

---

## Connecting to the Serial Console

One of the staples of CircuitPython (and programming in general!) is something called a "print statement". This is a line you include in your code that causes your code to output text. A print statement in CircuitPython looks like this:

```
print("Hello, world!")
```

This line would result in:

```
Hello, world!
```

However, these print statements need somewhere to display. That's where the serial console comes in!

The serial console receives output from your CircuitPython board sent over USB and displays it so you can see it. This is necessary when you've included a print statement in your code and you'd like to see what you printed. It is also helpful for troubleshooting errors, because your board will send errors and the serial console will print those too.

The serial console requires a terminal program. A terminal is a program that gives you a text-based interface to perform various tasks.

## Serial Console on Mac and Linux

Connecting to the serial console on Mac and Linux uses essentially the same process. Neither operating system needs drivers installed. On MacOSX, **Terminal** comes installed. On Linux, there are a variety such as gnome-terminal (called **Terminal**) or Konsole on KDE.

To connect to the serial output, you'll use the `screen` command. For a detailed explanation of how to connect to the serial console using screen, please see [the Advanced Serial Console on Mac and Linux page in this guide \(https://adafru.it/CYp\)](https://adafru.it/CYp).

# Serial Console on Windows

If you're using Windows, you'll need to download a terminal program. We suggest PuTTY. First, download the [latest version of PuTTY](https://adafru.it/Bf1) (<https://adafru.it/Bf1>). You'll want to download the Windows installer file. It is most likely that you'll need the 64-bit version. Download the file and install the program on your machine. If you run into issues, you can try downloading the 32-bit version instead. However, the 64-bit version will work on most PCs.

You'll use PuTTY to connect to the serial output. For a detailed explanation of how to use PuTTY to connect to the serial console, please see [the Advanced Serial Console on Windows page in this guide](https://adafru.it/CYq) (<https://adafru.it/CYq>).

---

## Interacting with the Serial Console

Once you've successfully connected to the serial console, it's time to start using it.

The code you wrote earlier has no output to the serial console. So, we're going to edit it to create some output.

Open your code.py file into your editor, and include a `print` statement. You can print anything you like! Just include your phrase between the quotation marks inside the parentheses. For example:

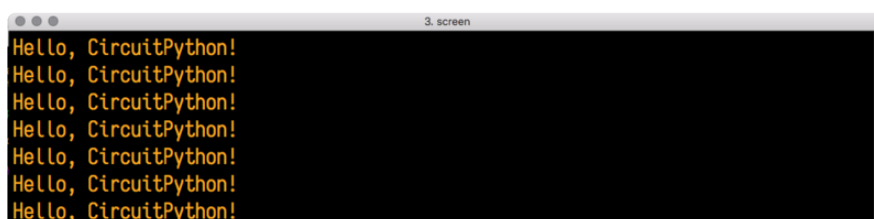
```
import time
import adafruit_trellism4

trellis = adafruit_trellism4.TrellisM4Express()

while True:
    print("Hello, CircuitPython!")
    trellis.pixels[0, 0] = (100, 0, 0)
    time.sleep(0.5)
    trellis.pixels[0, 0] = (0, 0, 0)
    time.sleep(0.5)
```

Save your file.

Now, let's go take a look at the window with our connection to the serial console.



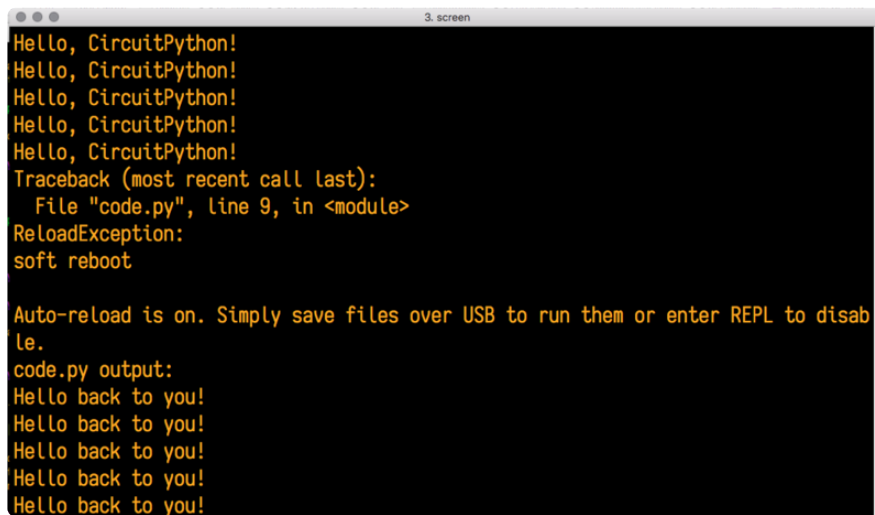
Excellent! Our print statement is showing up in our console! Try changing the printed text to something else.

```
import time
import adafruit_trellism4

trellis = adafruit_trellism4.TrellisM4Express()

while True:
    print("Hello back to you!")
    trellis.pixels[0, 0] = (100, 0, 0)
    time.sleep(0.5)
    trellis.pixels[0, 0] = (0, 0, 0)
    time.sleep(0.5)
```

Keep your serial console window where you can see it. Save your file. You'll see what the serial console displays when the board reboots. Then you'll see your new change!

A screenshot of a serial console window titled "3. screen". The window has a black background with yellow text. It shows a "Traceback (most recent call last):" error followed by "File 'code.py', line 9, in <module>", "ReloadException:", and "soft reboot". Below this, it says "Auto-reload is on. Simply save files over USB to run them or enter REPL to disable." and "code.py output:". The output shows five "Hello back to you!" messages.

```
3. screen
Hello, CircuitPython!
Hello, CircuitPython!
Hello, CircuitPython!
Hello, CircuitPython!
Hello, CircuitPython!
Traceback (most recent call last):
  File "code.py", line 9, in <module>
ReloadException:
soft reboot

Auto-reload is on. Simply save files over USB to run them or enter REPL to disable.
code.py output:
Hello back to you!
Hello back to you!
Hello back to you!
Hello back to you!
Hello back to you!
```

The **Traceback (most recent call last):** is telling you the last thing your board was doing before you saved your file. This is normal behavior and will happen every time the board resets. This is really handy for troubleshooting. Let's introduce an error so we can see how it is used.

Delete the **)** at the end of **(100, 0, 0)** from the line **trellis.pixels[0, 0] = (100, 0, 0)** so that it says **trellis.pixels[0, 0] = (100, 0, 0**.

```
import time
import adafruit_trellism4

trellis = adafruit_trellism4.TrellisM4Express()

while True:
    print("Hello back to you!")
    trellis.pixels[0, 0] = (100, 0, 0
    time.sleep(0.5)
    trellis.pixels[0, 0] = (0, 0, 0)
    time.sleep(0.5)
```

Save your file. You will notice that your red LED will stop blinking, and you may have a colored status LED blinking at you. This is because the code is no longer correct and can no longer run properly. We need to fix it!

Usually when you run into errors, it's not because you introduced them on purpose. You may have 200 lines of code, and have no idea where your error could be hiding. This is where the serial console can help. Let's take a look!

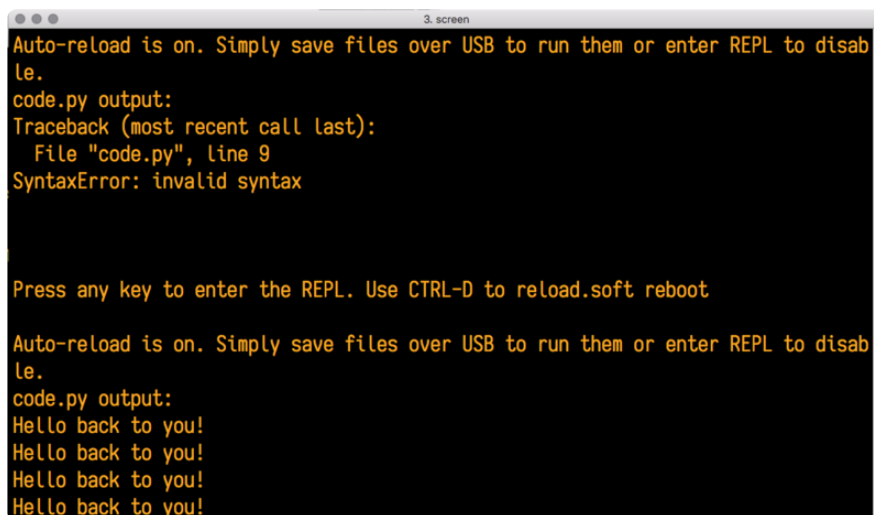


```
Auto-reload is on. Simply save files over USB to run them or enter REPL to disable.
code.py output:
Traceback (most recent call last):
  File "code.py", line 9
SyntaxError: invalid syntax

Press any key to enter the REPL. Use CTRL-D to reload.
```

The **Traceback (most recent call last):** is telling you that the last thing it was able to run was line 9 in your code. The next line is your error: **SyntaxError: invalid syntax**. This error might not mean a lot to you, but combined with knowing the issue is on line 9, it gives you a great place to start!

Go back to your code, and take a look at line 9. Obviously, you know what the problem is already. But if you didn't, you'd want to look at line 9 and see if you could figure it out. If you're still unsure, try googling the error to get some help. In this case, you know what to look for. You removed a parenthesis. Fix the typo and save your file.



```
Auto-reload is on. Simply save files over USB to run them or enter REPL to disable.
code.py output:
Traceback (most recent call last):
  File "code.py", line 9
SyntaxError: invalid syntax

Press any key to enter the REPL. Use CTRL-D to reload.soft reboot

Auto-reload is on. Simply save files over USB to run them or enter REPL to disable.
code.py output:
Hello back to you!
Hello back to you!
Hello back to you!
Hello back to you!
```

Nice job fixing the error! Your serial console is streaming and your NeoPixel is blinking red again.

The serial console will display any output generated by your code. Some sensors, such as a humidity sensor or a thermistor, receive data and you can use print statements to display that information. You can also use print statements for

troubleshooting. If your code isn't working, and you want to know where it's failing, you can put print statements in various places to see where it stops printing.

The serial console has many uses, and is an amazing tool overall for learning and programming!

---

## The REPL

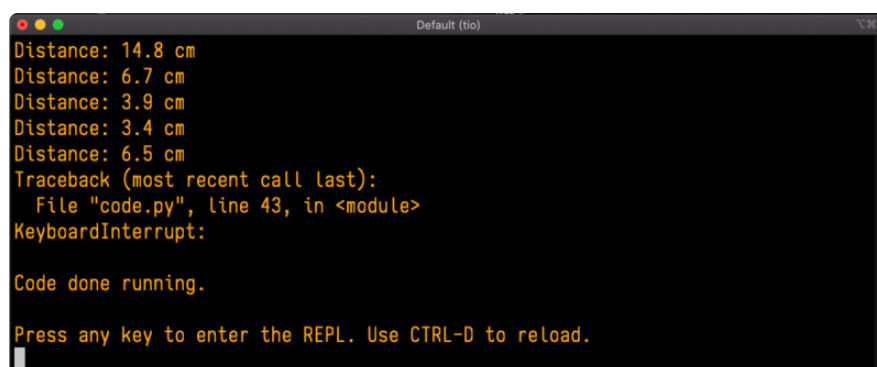
The other feature of the serial connection is the **Read-Evaluate-Print-Loop**, or REPL. The REPL allows you to enter individual lines of code and have them run immediately. It's really handy if you're running into trouble with a particular program and can't figure out why. It's interactive so it's great for testing new ideas.

## Entering the REPL

To use the REPL, you first need to be connected to the serial console. Once that connection has been established, you'll want to press **CTRL+C**.

If there is code running, in this case code measuring distance, it will stop and you'll see **Press any key to enter the REPL. Use CTRL-D to reload.** Follow those instructions, and press any key on your keyboard.

The **Traceback (most recent call last):** is telling you the last thing your board was doing before you pressed Ctrl + C and interrupted it. The **KeyboardInterrupt** is you pressing CTRL+C. This information can be handy when troubleshooting, but for now, don't worry about it. Just note that it is expected behavior.



```
Distance: 14.8 cm
Distance: 6.7 cm
Distance: 3.9 cm
Distance: 3.4 cm
Distance: 6.5 cm
Traceback (most recent call last):
  File "code.py", line 43, in <module>
KeyboardInterrupt:

Code done running.

Press any key to enter the REPL. Use CTRL-D to reload.
```

If your **code.py** file is empty or does not contain a loop, it will show an empty output and **Code done running.** There is no information about what your board was doing before you interrupted it because there is no code running.



```
Default (tio)
Auto-reload is on. Simply save files over USB to run them or enter REPL to disable.
code.py output:

Code done running.

Press any key to enter the REPL. Use CTRL-D to reload.
```

If you have no `code.py` on your **CIRCUITPY** drive, you will enter the REPL immediately after pressing CTRL+C. Again, there is no information about what your board was doing before you interrupted it because there is no code running.

```
Default (tio)
Auto-reload is on. Simply save files over USB to run them or enter REPL to disable.

Code done running.

Press any key to enter the REPL. Use CTRL-D to reload.
```

Regardless, once you press a key you'll see a `>>>` prompt welcoming you to the REPL!

```
Default (tio)
Adafruit CircuitPython 7.0.0 on 2021-10-26; Adafruit Feather RP2040 with rp2040
>>> |
```

If you have trouble getting to the `>>>` prompt, try pressing Ctrl + C a few more times.

The first thing you get from the REPL is information about your board.

```
Adafruit CircuitPython 7.0.0 on 2021-10-26; Adafruit Feather RP2040 with rp2040
```

This line tells you the version of CircuitPython you're using and when it was released. Next, it gives you the type of board you're using and the type of microcontroller the board uses. Each part of this may be different for your board depending on the versions you're working with.

This is followed by the CircuitPython prompt.

```
>>> |
```

## Interacting with the REPL

From this prompt you can run all sorts of commands and code. The first thing you'll do is run `help()`. This will tell you where to start exploring the REPL. To run code in the REPL, type it in next to the REPL prompt.

Type `help()` next to the prompt in the REPL.

```

Adafruit CircuitPython 7.0.0 on 2021-10-26; Adafruit Feather RP2040 with rp2040
>>> help()

```

Then press enter. You should then see a message.

```

Adafruit CircuitPython 7.0.0 on 2021-10-26; Adafruit Feather RP2040 with rp2040
>>> help()
Welcome to Adafruit CircuitPython 7.0.0!

Visit circuitpython.org for more information.

To list built-in modules type `help("modules")`.
>>>

```

First part of the message is another reference to the version of CircuitPython you're using. Second, a URL for the CircuitPython related project guides. Then... wait. What's this? `To list built-in modules type `help("modules")`.` Remember the modules you learned about while going through creating code? That's exactly what this is talking about! This is a perfect place to start. Let's take a look!

Type `help("modules")` into the REPL next to the prompt, and press enter.

```

>>> help("modules")
__main__      board          micropython    storage
_bleio        builtins       msgpack         struct
adafruit_bus_device  collections    busio           neopixel_write  supervisor
adafruit_pixelbuf  countio       onewireio      synthio
aesio         digitalio     paralleldisplay terminalio
analogio      displayio    pulseio        time
array         errno        pwmio          touchio
atexit        fontio       qrio           traceback
audiobusio    framebufferio  rainbowio      ulab
audiocore     gc           random         usb_cdc
audiomixer    getpass      re            usb_hid
audiomp3      imagecapture  rgbmatrix     usb_midi
audiopwmio    io           rotaryio      vectorio
binascii      json         rp2pio        watchdog
bitbangio     keypad       rtc
bitmaptools   math         sdcardio
bitops        microcontroller  sharpdisplay
Plus any modules on the filesystem
>>>

```

This is a list of all the core modules built into CircuitPython, including `board`. Remember, `board` contains all of the pins on the board that you can use in your code. From the REPL, you are able to see that list!

Type `import board` into the REPL and press enter. It'll go to a new prompt. It might look like nothing happened, but that's not the case! If you recall, the `import` statement simply tells the code to expect to do something with that module. In this case, it's telling the REPL that you plan to do something with that module.

```

>>> import board
>>>

```

Next, type `dir(board)` into the REPL and press enter.

```
>>> dir(board)
['_class_', '_name_', 'A0', 'A1', 'A2', 'A3', 'D0', 'D1', 'D10', 'D11', 'D12', 'D13',
'D24', 'D25', 'D4', 'D5', 'D6', 'D9', 'I2C', 'LED', 'MISO', 'MOSI', 'NEOPIXEL', 'RX', 'SCK',
', 'SCL', 'SDA', 'SPI', 'TX', 'UART', 'board_id']
>>>
```

This is a list of all of the pins on your board that are available for you to use in your code. Each board's list will differ slightly depending on the number of pins available. Do you see **LED** ? That's the pin you used to blink the red LED!

The REPL can also be used to run code. Be aware that **any code you enter into the REPL isn't saved** anywhere. If you're testing something new that you'd like to keep, make sure you have it saved somewhere on your computer as well!

Every programmer in every programming language starts with a piece of code that says, "Hello, World." You're going to say hello to something else. Type into the REPL:

```
print("Hello, CircuitPython!")
```

Then press enter.

```
>>> print("Hello, CircuitPython")
Hello, CircuitPython
>>>
```

That's all there is to running code in the REPL! Nice job!

You can write single lines of code that run stand-alone. You can also write entire programs into the REPL to test them. Remember that nothing typed into the REPL is saved.

There's a lot the REPL can do for you. It's great for testing new ideas if you want to see if a few new lines of code will work. It's fantastic for troubleshooting code by entering it one line at a time and finding out where it fails. It lets you see what modules are available and explore those modules.

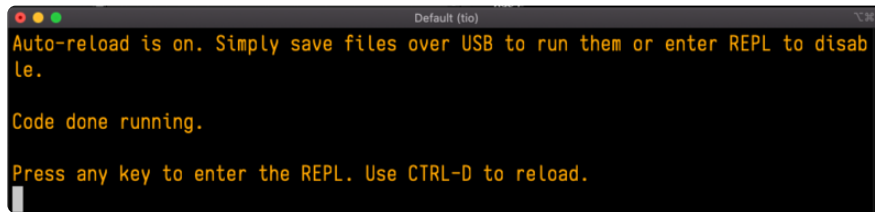
Try typing more into the REPL to see what happens!

Everything typed into the REPL is ephemeral. Once you reload the REPL or return to the serial console, nothing you typed will be retained in any memory space. So be sure to save any desired code you wrote somewhere else, or you'll lose it when you leave the current REPL instance!

# Returning to the Serial Console

When you're ready to leave the REPL and return to the serial console, simply press **CTRL+D**. This will reload your board and reenter the serial console. You will restart the program you had running before entering the REPL. In the console window, you'll see any output from the program you had running. And if your program was affecting anything visual on the board, you'll see that start up again as well.

You can return to the REPL at any time!

A screenshot of a serial console window titled "Default (tio)". The window has a black background with yellow text. The text reads: "Auto-reload is on. Simply save files over USB to run them or enter REPL to disable.", "Code done running.", and "Press any key to enter the REPL. Use CTRL-D to reload." There is a cursor at the end of the last line.

```
Auto-reload is on. Simply save files over USB to run them or enter REPL to disable.
Code done running.
Press any key to enter the REPL. Use CTRL-D to reload.
```

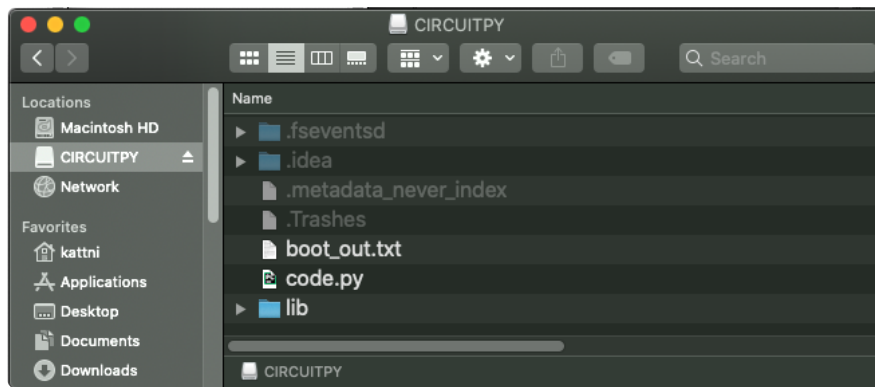
---

## CircuitPython Libraries

As CircuitPython development continues and there are new releases, Adafruit will stop supporting older releases. Visit <https://circuitpython.org/downloads> to download the latest version of CircuitPython for your board. You must download the CircuitPython Library Bundle that matches your version of CircuitPython. Please update CircuitPython and then visit <https://circuitpython.org/libraries> to download the latest Library Bundle.

Each CircuitPython program you run needs to have a lot of information to work. The reason CircuitPython is so simple to use is that most of that information is stored in other files and works in the background. These files are called libraries. Some of them are built into CircuitPython. Others are stored on your **CIRCUITPY** drive in a folder called **lib**. Part of what makes CircuitPython so great is its ability to store code separately from the firmware itself. Storing code separately from the firmware makes it easier to update both the code you write and the libraries you depend.

Your board may ship with a **lib** folder already, it's in the base directory of the drive. If not, simply create the folder yourself. When you first install CircuitPython, an empty **lib** directory will be created for you.



CircuitPython libraries work in the same way as regular Python modules so the [Python docs \(https://adafru.it/rar\)](https://adafru.it/rar) are an excellent reference for how it all should work. In Python terms, you can place our library files in the **lib** directory because it's part of the Python path by default.

One downside of this approach of separate libraries is that they are not built in. To use them, one needs to copy them to the **CIRCUITPY** drive before they can be used. Fortunately, there is a library bundle.

The bundle and the library releases on GitHub also feature optimized versions of the libraries with the **.mpy** file extension. These files take less space on the drive and have a smaller memory footprint as they are loaded.

Due to the regular updates and space constraints, Adafruit does not ship boards with the entire bundle. Therefore, you will need to load the libraries you need when you begin working with your board. You can find example code in the guides for your board that depends on external libraries.

Either way, as you start to explore CircuitPython, you'll want to know how to get libraries on board.

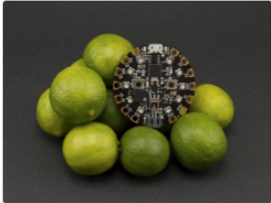
## The Adafruit Learn Guide Project Bundle

The quickest and easiest way to get going with a project from the Adafruit Learn System is by utilising the Project Bundle. Most guides now have a **Download Project Bundle** button available at the top of the full code example embed. This button downloads all the necessary files, including images, etc., to get the guide project up and running. Simply click, open the resulting zip, copy over the right files, and you're good to go!

The first step is to find the Download Project Bundle button in the guide you're working on.

The Download Project Bundle button is only available on full demo code embedded from GitHub in a Learn guide. Code snippets will NOT have the button available.

🏠 > Circuit Playground Express: Piano in the Key of Lime > Piano in the Key of Lime



## Piano in the Key of Lime

EN Save Subscribe

Now we'll take everything we learned and put it together!

Be sure to save your current code.py if you've changed anything you'd like to keep. Download the following file. Rename it to code.py and save it to your Circuit Playground Express.

**Download Project Bundle** Copy Code

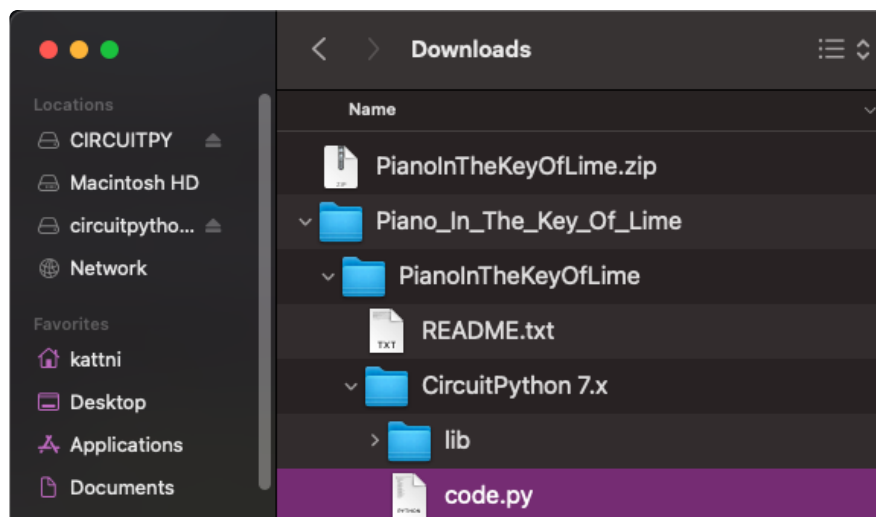
```
# SPDX-FileCopyrightText: 2017 Kattni Rembor for Adafruit Industries
#
# SPDX-License-Identifier: MIT

from adafruit_circuitplayground import cp

while True:
    if cp.switch:
        print("Slide switch off!")
        cp.pixels.fill((0, 0, 0))
```

When you copy the contents of the Project Bundle to your CIRCUITPY drive, it will replace all the existing content! If you don't want to lose anything, ensure you copy your current code to your computer before you copy over the new Project Bundle content!

The Download Project Bundle button downloads a zip file. This zip contains a series of directories, nested within which is the **code.py**, any applicable assets like images or audio, and the **lib/** folder containing all the necessary libraries. The following zip was downloaded from the Piano in the Key of Lime guide.



The Piano in the Key of Lime guide was chosen as an example. That guide is specific to Circuit Playground Express, and cannot be used on all boards. Do not



expect to download that exact bundle and have it work on your non-CPX microcontroller.

When you open the zip, you'll find some nested directories. Navigate through them until you find what you need. You'll eventually find a directory for your CircuitPython version (in this case, 7.x). In the version directory, you'll find the file and directory you need: **code.py** and **lib/**. Once you find the content you need, you can copy it all over to your **CIRCUITPY** drive, replacing any files already on the drive with the files from the freshly downloaded zip.

In some cases, there will be other files such as audio or images in the same directory as **code.py** and **lib/**. Make sure you include all the files when you copy things over!

Once you copy over all the relevant files, the project should begin running! If you find that the project is not running as expected, make sure you've copied ALL of the project files onto your microcontroller board.

That's all there is to using the Project Bundle!

## The Adafruit CircuitPython Library Bundle

Adafruit provides CircuitPython libraries for much of the hardware they provide, including sensors, breakouts and more. To eliminate the need for searching for each library individually, the libraries are available together in the Adafruit CircuitPython Library Bundle. The bundle contains all the files needed to use each library.

### Downloading the Adafruit CircuitPython Library Bundle

You can download the latest Adafruit CircuitPython Library Bundle release by clicking the button below. The libraries are being constantly updated and improved, so you'll always want to download the latest bundle.

**Match up the bundle version with the version of CircuitPython you are running.** For example, you would download the 6.x library bundle if you're running any version of CircuitPython 6, or the 7.x library bundle if you're running any version of CircuitPython 7, etc. If you mix libraries with major CircuitPython versions, you will get incompatible mpy errors due to changes in library interfaces possible during major version changes.

Click to visit [circuitpython.org](https://circuitpython.org) for  
the latest Adafruit CircuitPython  
Library Bundle

<https://adafru.it/ENC>

Download the bundle version that matches your CircuitPython firmware version. If you don't know the version, check the version info in `boot_out.txt` file on the **CIRCUITPY** drive, or the initial prompt in the CircuitPython REPL. For example, if you're running v7.0.0, download the 7.x library bundle.

There's also a **py** bundle which contains the uncompressed python files, you probably don't want that unless you are doing advanced work on libraries.

## The CircuitPython Community Library Bundle

The CircuitPython Community Library Bundle is made up of libraries written and provided by members of the CircuitPython community. These libraries are often written when community members encountered hardware not supported in the Adafruit Bundle, or to support a personal project. The authors all chose to submit these libraries to the Community Bundle make them available to the community.

**These libraries are maintained by their authors and are not supported by Adafruit.**

As you would with any library, if you run into problems, feel free to file an issue on the GitHub repo for the library. Bear in mind, though, that most of these libraries are supported by a single person and you should be patient about receiving a response. Remember, these folks are not paid by Adafruit, and are volunteering their personal time when possible to provide support.

## Downloading the CircuitPython Community Library Bundle

You can download the latest CircuitPython Community Library Bundle release by clicking the button below. The libraries are being constantly updated and improved, so you'll always want to download the latest bundle.

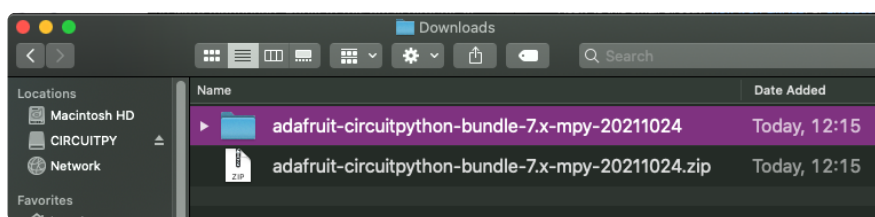
Click for the latest CircuitPython  
Community Library Bundle release

<https://adafru.it/VCn>

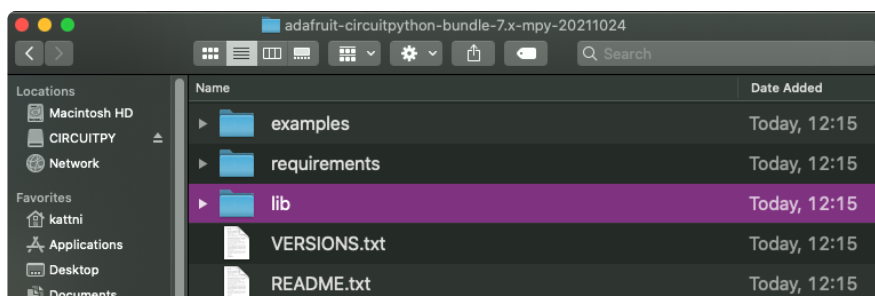
The link takes you to the latest release of the CircuitPython Community Library Bundle on GitHub. There are multiple versions of the bundle available. **Download the bundle version that matches your CircuitPython firmware version.** If you don't know the version, check the version info in `boot_out.txt` file on the **CIRCUITPY** drive, or the initial prompt in the CircuitPython REPL. For example, if you're running v7.0.0, download the 7.x library bundle.

## Understanding the Bundle

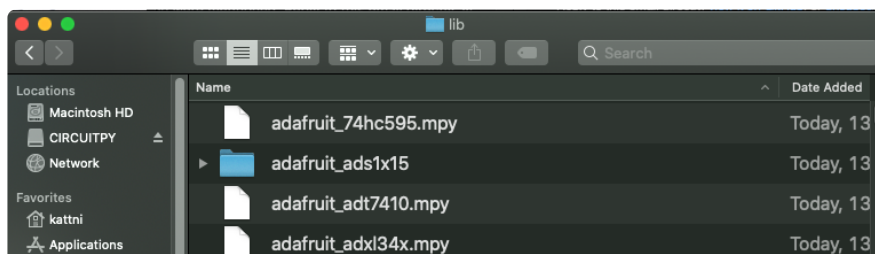
After downloading the zip, extract its contents. This is usually done by double clicking on the zip. On Mac OSX, it places the file in the same directory as the zip.



Open the bundle folder. Inside you'll find two information files, and two folders. One folder is the lib bundle, and the other folder is the examples bundle.



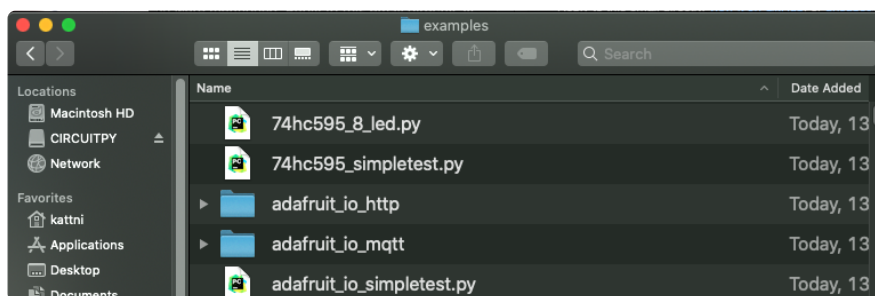
Now open the lib folder. When you open the folder, you'll see a large number of `.mpy` files, and folders.



## Example Files

All example files from each library are now included in the bundles in an **examples** directory (as seen above), as well as an examples-only bundle. These are included for two main reasons:

- Allow for quick testing of devices.
- Provide an example base of code, that is easily built upon for individualized purposes.



## Copying Libraries to Your Board

First open the **lib** folder on your **CIRCUITPY** drive. Then, open the **lib** folder you extracted from the downloaded zip. Inside you'll find a number of folders and **.mpy** files. Find the library you'd like to use, and copy it to the **lib** folder on **CIRCUITPY**.

If the library is a directory with multiple **.mpy** files in it, be sure to **copy the entire folder to CIRCUITPY/lib**.

This also applies to example files. Open the **examples** folder you extracted from the downloaded zip, and copy the applicable file to your **CIRCUITPY** drive. Then, rename it to **code.py** to run it.

If a library has multiple **.mpy** files contained in a folder, be sure to copy the entire folder to **CIRCUITPY/lib**.

## Understanding Which Libraries to Install

You now know how to load libraries on to your CircuitPython-compatible microcontroller board. You may now be wondering, how do you know which libraries you need to install? Unfortunately, it's not always straightforward. Fortunately, there is

an obvious place to start, and a relatively simple way to figure out the rest. First up: the best place to start.

When you look at most CircuitPython examples, you'll see they begin with one or more `import` statements. These typically look like the following:

- `import library_or_module`

However, `import` statements can also sometimes look like the following:

- `from library_or_module import name`
- `from library_or_module.subpackage import name`
- `from library_or_module import name as local_name`

They can also have more complicated formats, such as including a `try` / `except` block, etc.

The important thing to know is that an `import` statement will always include the name of the module or library that you're importing.

Therefore, the best place to start is by reading through the `import` statements.

Here is an example import list for you to work with in this section. There is no setup or other code shown here, as the purpose of this section involves only the import list.

```
import time
import board
import neopixel
import adafruit_lis3dh
import usb_hid
from adafruit_hid.consumer_control import ConsumerControl
from adafruit_hid.consumer_control_code import ConsumerControlCode
```

Keep in mind, not all imported items are libraries. Some of them are almost always built-in CircuitPython modules. How do you know the difference? Time to visit the REPL.

In the [Interacting with the REPL section \(https://adafru.it/Awz\)](https://adafru.it/Awz) on [The REPL page \(https://adafru.it/Awz\)](https://adafru.it/Awz) in this guide, the `help("modules")` command is discussed. This command provides a list of all of the built-in modules available in CircuitPython for your board. So, if you connect to the serial console on your board, and enter the REPL, you can run `help("modules")` to see what modules are available for your board. Then, as you read through the `import` statements, you can,

for the purposes of figuring out which libraries to load, ignore the statement that import modules.

The following is the list of modules built into CircuitPython for the Feather RP2040. Your list may look similar or be anything down to a significant subset of this list for smaller boards.

```
>>> help("modules")
__main__      board      micropython   storage
__bleio       builtins      msgpack        struct
adafruit_bus_device busio         neopixel_write supervisor
adafruit_pixelbuf collections  onewireio      synthio
aesio         countio       os              sys
alarm         digitalio     paralleldisplay terminalio
analogio      displayio     pulseio         time
array         errno         pwmio           touchio
atexit        fontio        qrio            traceback
audiobusio    framebufferio rainbowio        ulab
audiocore     gc            random          usb_cdc
audiomixer    getpass       re              usb_hid
audiomp3      imagecapture  rgbmatrix       usb_midi
audiopwmio    io            rotaryio        vectorio
binascii      json          rp2pio          watchdog
bitbangio     keypad        rtc
bitmaptools   math          sdcardio
bitops        microcontroller sharpdisplay
```

Now that you know what you're looking for, it's time to read through the import statements. The first two, `time` and `board`, are on the modules list above, so they're built-in.

The next one, `neopixel`, is not on the module list. That means it's your first library! So, you would head over to the bundle zip you downloaded, and search for `neopixel`. There is a `neopixel.mpy` file in the bundle zip. Copy it over to the `lib` folder on your **CIRCUITPY** drive. The following one, `adafruit_lis3dh`, is also not on the module list. Follow the same process for `adafruit_lis3dh`, where you'll find `adafruit_lis3dh.mpy`, and copy that over.

The fifth one is `usb_hid`, and it is in the modules list, so it is built in. Often all of the built-in modules come first in the import list, but sometimes they don't! Don't assume that everything after the first library is also a library, and verify each import with the modules list to be sure. Otherwise, you'll search the bundle and come up empty!

The final two imports are not as clear. Remember, when `import` statements are formatted like this, the first thing after the `from` is the library name. In this case, the library name is `adafruit_hid`. A search of the bundle will find an `adafruit_hid` folder. When a library is a folder, you must copy the **entire folder and its contents as it is in the bundle** to the `lib` folder on your **CIRCUITPY** drive. In this case, you would copy the entire `adafruit_hid` folder to your **CIRCUITPY/lib** folder.



Notice that there are two imports that begin with `adafruit_hid`. Sometimes you will need to import more than one thing from the same library. Regardless of how many times you import the same library, you only need to load the library by copying over the `adafruit_hid` folder once.

That is how you can use your example code to figure out what libraries to load on your CircuitPython-compatible board!

There are cases, however, where libraries require other libraries internally. The internally required library is called a dependency. In the event of library dependencies, the easiest way to figure out what other libraries are required is to connect to the serial console and follow along with the `ImportError` printed there. The following is a very simple example of an `ImportError`, but the concept is the same for any missing library.

## Example: `ImportError` Due to Missing Library

If you choose to load libraries as you need them, or you're starting fresh with an existing example, you may end up with code that tries to use a library you haven't yet loaded. This section will demonstrate what happens when you try to utilise a library that you don't have loaded on your board, and cover the steps required to resolve the issue.

This demonstration will only return an error if you do not have the required library loaded into the `lib` folder on your **CIRCUITPY** drive.

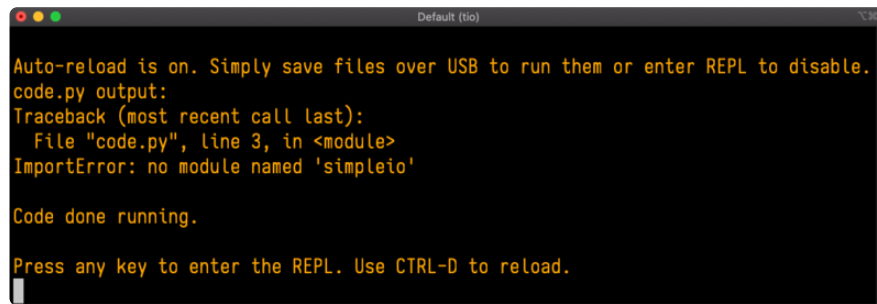
Let's use a modified version of the Blink example.

```
import board
import time
import simpleio

led = simpleio.DigitalOut(board.LED)

while True:
    led.value = True
    time.sleep(0.5)
    led.value = False
    time.sleep(0.5)
```

Save this file. Nothing happens to your board. Let's check the serial console to see what's going on.

A screenshot of a terminal window titled "Default (tio)" with a dark background and yellow text. The text displays the output of a Python script, showing an ImportError for the 'simpleio' module. The message indicates that auto-reload is on and provides instructions on how to enter the REPL or reload the code.

```
Auto-reload is on. Simply save files over USB to run them or enter REPL to disable.  
code.py output:  
Traceback (most recent call last):  
  File "code.py", line 3, in <module>  
    ImportError: no module named 'simpleio'  
  
Code done running.  
  
Press any key to enter the REPL. Use CTRL-D to reload.
```

You have an `ImportError`. It says there is `no module named 'simpleio'`. That's the one you just included in your code!

Click the link above to download the correct bundle. Extract the lib folder from the downloaded bundle file. Scroll down to find `simpleio.mpy`. This is the library file you're looking for! Follow the steps above to load an individual library file.

The LED starts blinking again! Let's check the serial console.

A screenshot of a terminal window titled "Default (tio)" with a dark background and yellow text. The text shows the terminal after a soft reboot, indicating that the code is running successfully without any errors.

```
Press any key to enter the REPL. Use CTRL-D to reload.  
soft reboot  
  
Auto-reload is on. Simply save files over USB to run them or enter REPL to disable.  
code.py output:
```

No errors! Excellent. You've successfully resolved an `ImportError`!

If you run into this error in the future, follow along with the steps above and choose the library that matches the one you're missing.

## Library Install on Non-Express Boards

If you have an M0 non-Express board such as Trinket M0, Gemma M0, QT Py M0, or one of the M0 Trinkeys, you'll want to follow the same steps in the example above to install libraries as you need them. Remember, you don't need to wait for an `ImportError` if you know what library you added to your code. Open the library bundle you downloaded, find the library you need, and drag it to the `lib` folder on your **CIRCUITPY** drive.

You can still end up running out of space on your M0 non-Express board even if you only load libraries as you need them. There are a number of steps you can use to try to resolve this issue. You'll find suggestions on the [Troubleshooting page \(https://adafruit.it/Den\)](https://adafruit.it/Den).

# Updating CircuitPython Libraries and Examples

Libraries and examples are updated from time to time, and it's important to update the files you have on your **CIRCUITPY** drive.

To update a single library or example, follow the same steps above. When you drag the library file to your lib folder, it will ask if you want to replace it. Say yes. That's it!

A new library bundle is released every time there's an update to a library. Updates include things like bug fixes and new features. It's important to check in every so often to see if the libraries you're using have been updated.

## CircUp CLI Tool

There is a command line interface (CLI) utility called [CircUp \(https://adafru.it/Tfi\)](https://adafru.it/Tfi) that can be used to easily install and update libraries on your device. Follow the directions on the [install page within the CircUp learn guide \(https://adafru.it/-Ad\)](https://adafru.it/-Ad). Once you've got it installed you run the command `circup update` in a terminal to interactively update all libraries on the connected CircuitPython device. See the [usage page in the CircUp guide \(https://adafru.it/-Ah\)](https://adafru.it/-Ah) for a full list of functionality

---

## CircuitPython Pins and Modules

CircuitPython is designed to run on microcontrollers and allows you to interface with all kinds of sensors, inputs and other hardware peripherals. There are tons of guides showing how to wire up a circuit, and use CircuitPython to, for example, read data from a sensor, or detect a button press. Most CircuitPython code includes hardware setup which requires various modules, such as `board` or `digitalio`. You import these modules and then use them in your code. How does CircuitPython know to look for hardware in the specific place you connected it, and where do these modules come from?

This page explains both. You'll learn how CircuitPython finds the pins on your microcontroller board, including how to find the available pins for your board and what each pin is named. You'll also learn about the modules built into CircuitPython, including how to find all the modules available for your board.

# CircuitPython Pins

When using hardware peripherals with a CircuitPython compatible microcontroller, you'll almost certainly be utilising pins. This section will cover how to access your board's pins using CircuitPython, how to discover what pins and board-specific objects are available in CircuitPython for your board, how to use the board-specific objects, and how to determine all available pin names for a given pin on your board.

## `import board`

When you're using any kind of hardware peripherals wired up to your microcontroller board, the import list in your code will include `import board`. The `board` module is built into CircuitPython, and is used to provide access to a series of board-specific objects, including pins. Take a look at your microcontroller board. You'll notice that next to the pins are pin labels. You can always access a pin by its pin label. However, there are almost always multiple names for a given pin.

To see all the available board-specific objects and pins for your board, enter the REPL (`>>>`) and run the following commands:

```
import board
dir(board)
```

Here is the output for the QT Py SAMD21. **You may have a different board, and this list will vary, based on the board.**

```
>>> import board
>>> dir(board)
['__class__', 'A0', 'A1', 'A10', 'A2', 'A3', 'A6', 'A7', 'A8', 'A9', 'D0', 'D1',
'D10', 'D2', 'D3', 'D4', 'D5', 'D6', 'D7', 'D8', 'D9', 'I2C', 'MISO', 'MOSI',
'NEOPIXEL', 'NEOPIXEL_POWER', 'RX', 'SCK', 'SCL', 'SDA', 'SPI', 'TX', 'UART']
```

The following pins have labels on the physical QT Py SAMD21 board: A0, A1, A2, A3, SDA, SCL, TX, RX, SCK, MISO, and MOSI. You see that there are many more entries available in `board` than the labels on the QT Py.

You can use the pin names on the physical board, regardless of whether they seem to be specific to a certain protocol.

For example, you do not have to use the SDA pin for I2C - you can use it for a button or LED.

On the flip side, there may be multiple names for one pin. For example, on the QT Py SAMD21, pin **A0** is labeled on the physical board silkscreen, but it is available in CircuitPython as both **A0** and **D0**. For more information on finding all the names for a given pin, see the [What Are All the Available Pin Names?](https://adafru.it/QkA) (<https://adafru.it/QkA>) section below.

The results of `dir(board)` for CircuitPython compatible boards will look similar to the results for the QT Py SAMD21 in terms of the pin names, e.g. A0, D0, etc. However, some boards, for example, the Metro ESP32-S2, have different styled pin names. Here is the output for the Metro ESP32-S2.

```
>>> import board
>>> dir(board)
['__class__', 'A0', 'A1', 'A2', 'A3', 'A4', 'A5', 'DEBUG_RX', 'DEBUG_TX', 'I2C',
'I01', 'I010', 'I011', 'I012', 'I013', 'I014', 'I015', 'I016', 'I017', 'I018',
'I02', 'I021', 'I03', 'I033', 'I034', 'I035', 'I036', 'I037', 'I04', 'I042', 'I045',
'I05', 'I06', 'I07', 'I08', 'I09', 'LED', 'MISO', 'MOSI', 'NEOPIXEL', 'RX',
'SCK', 'SCL', 'SDA', 'SPI', 'TX', 'UART']
```

Note that most of the pins are named in an IO# style, such as **IO1** and **IO2**. Those pins on the physical board are labeled only with a number, so an easy way to know how to access them in CircuitPython, is to run those commands in the REPL and find the pin naming scheme.

If your code is failing to run because it can't find a pin name you provided, verify that you have the proper pin name by running these commands in the REPL.

## I2C, SPI, and UART

You'll also see there are often (**but not always!**) three special board-specific objects included: **I2C**, **SPI**, and **UART** - each one is for the default pin-set used for each of the three common protocol busses they are named for. These are called singletons.

What's a singleton? When you create an object in CircuitPython, you are instantiating ('creating') it. Instantiating an object means you are creating an instance of the object with the unique values that are provided, or "passed", to it.

For example, When you instantiate an I2C object using the **busio** module, it expects two pins: clock and data, typically SCL and SDA. It often looks like this:

```
i2c = busio.I2C(board.SCL, board.SDA)
```

Then, you pass the I2C object to a driver for the hardware you're using. For example, if you were using the TSL2591 light sensor and its CircuitPython library, the next line of code would be:

```
tsl2591 = adafruit_tsl2591.TSL2591(i2c)
```

However, CircuitPython makes this simpler by including the `I2C` singleton in the `board` module. Instead of the two lines of code above, you simply provide the singleton as the I2C object. So if you were using the TSL2591 and its CircuitPython library, the two above lines of code would be replaced with:

```
tsl2591 = adafruit_tsl2591.TSL2591(board.I2C())
```

The `board.I2C()`, `board.SPI()`, and `board.UART()` singletons do not exist on all boards. They exist if there are board markings for the default pins for those devices.

This eliminates the need for the `busio` module, and simplifies the code. Behind the scenes, the `board.I2C()` object is instantiated when you call it, but not before, and on subsequent calls, it returns the same object. Basically, it does not create an object until you need it, and provides the same object every time you need it. You can call `board.I2C()` as many times as you like, and it will always return the same object.

The UART/SPI/I2C singletons will use the 'default' bus pins for each board - often labeled as RX/TX (UART), MOSI/MISO/SCK (SPI), or SDA/SCL (I2C). Check your board documentation/pinout for the default busses.

## What Are All the Available Names?

Many pins on CircuitPython compatible microcontroller boards have multiple names, however, typically, there's only one name labeled on the physical board. So how do you find out what the other available pin names are? Simple, with the following script! Each line printed out to the serial console contains the set of names for a particular pin.

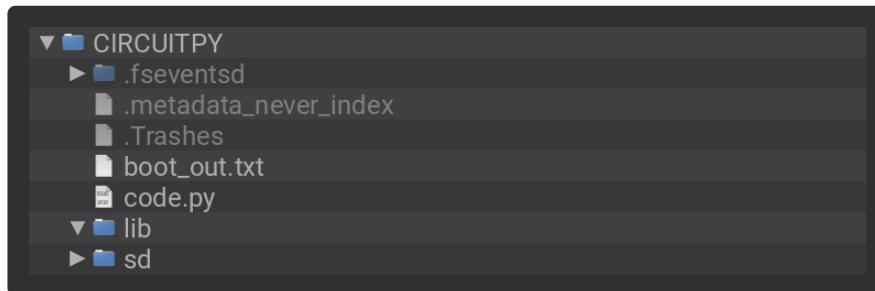
On a microcontroller board running CircuitPython, first, connect to the serial console.

In the example below, click the **Download Project Bundle** button below to download the necessary libraries and the `code.py` file in a zip file. Extract the contents of the zip file, open the directory `CircuitPython_Essentials/Pin_Map_Script/` and then click on



the directory that matches the version of CircuitPython you're using and copy the contents of that directory to your **CIRCUITPY** drive.

Your **CIRCUITPY** drive should now look similar to the following image:



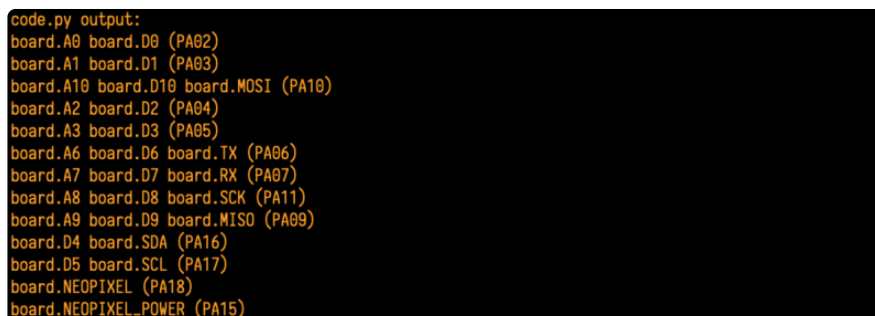
```
# SPDX-FileCopyrightText: 2020 anecdata for Adafruit Industries
# SPDX-FileCopyrightText: 2021 Neradoc for Adafruit Industries
# SPDX-FileCopyrightText: 2021-2023 Kattni Rembor for Adafruit Industries
# SPDX-FileCopyrightText: 2023 Dan Halbert for Adafruit Industries
#
# SPDX-License-Identifier: MIT

"""CircuitPython Essentials Pin Map Script"""
import microcontroller
import board
try:
    import cyw43 # raspberrypi
except ImportError:
    cyw43 = None

board_pins = []
for pin in dir(microcontroller.pin):
    if (isinstance(getattr(microcontroller.pin, pin), microcontroller.Pin) or
        (cyw43 and isinstance(getattr(microcontroller.pin, pin), cyw43.CywPin))):
        pins = []
        for alias in dir(board):
            if getattr(board, alias) is getattr(microcontroller.pin, pin):
                pins.append(f"board.{alias}")
        # Add the original GPIO name, in parentheses.
        if pins:
            # Only include pins that are in board.
            pins.append(f"({str(pin)})")
            board_pins.append(" ".join(pins))

for pins in sorted(board_pins):
    print(pins)
```

Here is the result when this script is run on QT Py SAMD21:



Each line represents a single pin. Find the line containing the pin name that's labeled on the physical board, and you'll find the other names available for that pin. For example, the first pin on the board is labeled **A0**. The first line in the output is `board.A0 board.D0 (PA02)`. This means that you can access pin **A0** in CircuitPython using both `board.A0` and `board.D0`.

The pins in parentheses are the microcontroller pin names. See the next section for more info on those.

You'll notice there are two "pins" that aren't labeled on the board but appear in the list: `board.NEOPIXEL` and `board.NEOPIXEL_POWER`. Many boards have several of these special pins that give you access to built-in board hardware, such as an LED or an on-board sensor. The QT Py SAMD21 only has one on-board extra piece of hardware, a NeoPixel LED, so there's only the one available in the list. But you can also control whether or not power is applied to the NeoPixel, so there's a separate pin for that.

That's all there is to figuring out the available names for a pin on a compatible microcontroller board in CircuitPython!

## Microcontroller Pin Names

The pin names available to you in the CircuitPython `board` module are not the same as the names of the pins on the microcontroller itself. The board pin names are aliases to the microcontroller pin names. If you look at the datasheet for your microcontroller, you'll likely find a pinout with a series of pin names, such as "PA18" or "GPIO5". If you want to get to the actual microcontroller pin name in CircuitPython, you'll need the `microcontroller.pin` module. As with `board`, you can run `dir(microcontroller.pin)` in the REPL to receive a list of the microcontroller pin names.

```
>>> import microcontroller
>>> dir(microcontroller.pin)
['__class__', 'PA02', 'PA03', 'PA04', 'PA05', 'PA06', 'PA07', 'PA08', 'PA09',
'PA10', 'PA11', 'PA15', 'PA16', 'PA17', 'PA18', 'PA19', 'PA22', 'PA23']
```

## CircuitPython Built-In Modules

There is a set of modules used in most CircuitPython programs. One or more of these modules is always used in projects involving hardware. Often hardware requires installing a separate library from the Adafruit CircuitPython Bundle. But, if you try to find `board` or `digitalio` in the same bundle, you'll come up lacking. So, where do

these modules come from? They're built into CircuitPython! You can find an comprehensive list of built-in CircuitPython modules and the technical details of their functionality from CircuitPython [here](https://adafru.it/QkB) (<https://adafru.it/QkB>) and the Python-like modules included [here](https://adafru.it/QkC) (<https://adafru.it/QkC>). However, **not every module is available for every board** due to size constraints or hardware limitations. How do you find out what modules are available for your board?

There are two options for this. You can check the [support matrix](https://adafru.it/N2a) (<https://adafru.it/N2a>), and search for your board by name. Or, you can use the REPL.

Plug in your board, connect to the serial console and enter the REPL. Type the following command.

```
help("modules")
```

```
>>> help("modules")
__main__      collections    neopixel_write  supervisor
_pixelbuf     digitalio     os               sys
adafruit_bus_device  displayio      pulseio         terminalio
analogio      errno         pwmio            time
array          fontio        random           touchio
audiocore      gamepad       re               usb_hid
audioio        gc            rotaryio         usb_midi
board          math          rtc              vectorio
builtins       microcontroller  storage
busio          micropython     struct
Plus any modules on the filesystem
```

That's it! You now know two ways to find all of the modules built into CircuitPython for your compatible microcontroller board.

---

## Frequently Asked Questions

These are some of the common questions regarding CircuitPython and CircuitPython microcontrollers.

---

### What are some common acronyms to know?

CP or CPy = [CircuitPython](https://adafru.it/KJD) (<https://adafru.it/KJD>)

CPC = [Circuit Playground Classic](http://adafru.it/3000) (<http://adafru.it/3000>) (does not run CircuitPython)

CPX = [Circuit Playground Express](http://adafru.it/3333) (<http://adafru.it/3333>)

CPB = [Circuit Playground Bluefruit](http://adafru.it/4333) (<http://adafru.it/4333>)

---

## Using Older Versions

As CircuitPython development continues and there are new releases, Adafruit will stop supporting older releases. Visit <https://circuitpython.org/downloads> to download the latest version of CircuitPython for your board. You must download the CircuitPython Library Bundle that matches your version of CircuitPython. Please update CircuitPython and then visit <https://circuitpython.org/libraries> to download the latest Library Bundle.

---

### I have to continue using CircuitPython 7.x or earlier. Where can I find compatible libraries?

We are no longer building or supporting the CircuitPython 7.x or earlier library bundles. We highly encourage you to [update CircuitPython to the latest version \(https://adafru.it/Em8\)](https://adafru.it/Em8) and use [the current version of the libraries \(https://adafru.it/ENC\)](https://adafru.it/ENC). However, if for some reason you cannot update, here are the last available library bundles for older versions:

- [2.x bundle \(https://adafru.it/FJA\)](https://adafru.it/FJA)
- [3.x bundle \(https://adafru.it/FJB\)](https://adafru.it/FJB)
- [4.x bundle \(https://adafru.it/QDL\)](https://adafru.it/QDL)
- [5.x bundle \(https://adafru.it/QDJ\)](https://adafru.it/QDJ)
- [6.x bundle \(https://adafru.it/Xmf\)](https://adafru.it/Xmf)
- [7.x bundle \(https://adafru.it/18e9\)](https://adafru.it/18e9)

---

## Python Arithmetic

---

### Does CircuitPython support floating-point numbers?

All CircuitPython boards support floating point arithmetic, even if the microcontroller chip does not support floating point in hardware. Floating point numbers are stored in 30 bits, with an 8-bit exponent and a 22-bit mantissa. Note that this is two bits less than standard 32-bit single-precision floats. You will get about 5-1/2 digits of decimal precision.

(The **broadcom** port may provide 64-bit floats in some cases.)

## Does CircuitPython support long integers, like regular Python?

Python long integers (integers of arbitrary size) are available on most builds, except those on boards with the smallest available firmware size. On these boards, integers are stored in 31 bits.

Boards without long integer support are mostly SAMD21 ("M0") boards without an external flash chip, such as the Adafruit Gemma M0, Trinket M0, QT Py M0, and the Trinket series. There are also a number of third-party boards in this category. There are also a few small STM third-party boards without long integer support.

`time.localtime()`, `time.mktime()`, `time.time()`, and `time.monotonic_ns()` are available only on builds with long integers.

---

## Wireless Connectivity

---

### How do I connect to the Internet with CircuitPython?

If you'd like to include WiFi in your project, your best bet is to use a board that is running natively on ESP32 chipsets - those have WiFi built in!

If your development board has an SPI port and at least 4 additional pins, you can check out [this guide \(https://adafru.it/F5X\)](https://adafru.it/F5X) on using AirLift with CircuitPython - extra wiring is required and some boards like the MacroPad or NeoTrellis do not have enough available pins to add the hardware support.

For further project examples, and guides about using AirLift with specific hardware, check out [the Adafruit Learn System \(https://adafru.it/VBr\)](https://adafru.it/VBr).

---

### How do I do BLE (Bluetooth Low Energy) with CircuitPython?

The nRF52840 and nRF52833 boards have the most complete BLE implementation. Your program can act as both a BLE central and peripheral. As a central, you can scan for advertisements, and connect to an advertising board. As a peripheral, you can advertise, and you can create services available to a central. Pairing and bonding are supported.

ESP32-C3 and ESP32-S3 boards currently provide an [incomplete \(https://adafru.it/11Au\)](https://adafru.it/11Au) BLE implementation. Your program can act as a central, and connect to a

peripheral. You can advertise, but you cannot create services. You cannot advertise anonymously. Pairing and bonding are not supported.

The ESP32 could provide a similar implementation, but it is not yet available. Note that the ESP32-S2 does not have Bluetooth capability.

On most other boards with adequate firmware space, [BLE is available for use with AirLift \(https://adafru.it/11Av\)](https://adafru.it/11Av) or other NINA-FW-based co-processors. Some boards have this coprocessor on board, such as the [PyPortal \(https://adafru.it/11Aw\)](https://adafru.it/11Aw).

Currently, this implementation only supports acting as a BLE peripheral. Scanning and connecting as a central are not yet implemented. Bonding and pairing are not supported.

---

## Are there other ways to communicate by radio with CircuitPython?

Check out [Adafruit's RFM boards \(https://adafru.it/11Ay\)](https://adafru.it/11Ay) for simple radio communication supported by CircuitPython, which can be used over distances of 100m to over a km, depending on the version. The RFM SAMD21 M0 boards can be used, but they were not designed for CircuitPython, and have limited RAM and flash space; using the RFM breakouts or FeatherWings with more capable boards will be easier.

---

## Asyncio and Interrupts

---

### Is there asyncio support in CircuitPython?

There is support for asyncio starting with CircuitPython 7.1.0, on all boards except the smallest SAMD21 builds. Read about using it in the [Cooperative Multitasking in CircuitPython \(https://adafru.it/XnA\)](https://adafru.it/XnA) Guide.

---

### Does CircuitPython support interrupts?

No. CircuitPython does not currently support interrupts - please use asyncio for multitasking / 'threaded' control of your code

---

## Status RGB LED

---



## My RGB NeoPixel/DotStar LED is blinking funny colors - what does it mean?

The status LED can tell you what's going on with your CircuitPython board. [Read more here for what the colors mean! \(https://adafru.it/Den\)](https://adafru.it/Den)

---

## Memory Issues

---

### What is a MemoryError?

Memory allocation errors happen when you're trying to store too much on the board. The CircuitPython microcontroller boards have a limited amount of memory available. You can have about 250 lines of code on the M0 Express boards. If you try to `import` too many libraries, a combination of large libraries, or run a program with too many lines of code, your code will fail to run and you will receive a `MemoryError` in the serial console.

---

### What do I do when I encounter a MemoryError?

Try resetting your board. Each time you reset the board, it reallocates the memory. While this is unlikely to resolve your issue, it's a simple step and is worth trying.

Make sure you are using `.mpy` versions of libraries. All of the CircuitPython libraries are available in the bundle in a `.mpy` format which takes up less memory than `.py` format. Be sure that you're using [the latest library bundle \(https://adafru.it/uap\)](https://adafru.it/uap) for your version of CircuitPython.

If that does not resolve your issue, try shortening your code. Shorten comments, remove extraneous or unneeded code, or any other clean up you can do to shorten your code. If you're using a lot of functions, you could try moving those into a separate library, creating a `.mpy` of that library, and importing it into your code.

You can turn your entire file into a `.mpy` and `import` that into `code.py`. This means you will be unable to edit your code live on the board, but it can save you space.

---

### Can the order of my `import` statements affect memory?

It can because the memory gets fragmented differently depending on allocation order and the size of objects. Loading `.mpy` files uses less memory so its recommended to do that for files you aren't editing.

---

## How can I create my own .mpy files?

You can make your own .mpy versions of files with `mpy-cross`.

You can download `mpy-cross` for your operating system from [here \(https://adafru.it/QDK\)](https://adafru.it/QDK). Builds are available for Windows, macOS, x64 Linux, and Raspberry Pi Linux. Choose the latest `mpy-cross` whose version matches the version of CircuitPython you are using.

To make a .mpy file, run `./mpy-cross path/to/yourfile.py` to create a `yourfile.mpy` in the same directory as the original file.

---

## How do I check how much memory I have free?

Run the following to see the number of bytes available for use:

```
import gc
gc.mem_free()
```

---

## Unsupported Hardware

---

### Is ESP8266 or ESP32 supported in CircuitPython? Why not?

We dropped ESP8266 support as of 4.x - For more information please read about it [here \(https://adafru.it/CiG\)](https://adafru.it/CiG)!

As of CircuitPython 8.x we have started to support ESP32 and ESP32-C3 and have added a WiFi workflow for wireless coding! (<https://adafru.it/10JF>)

We also support ESP32-S2 & ESP32-S3, which have native USB.

---

### Does Feather M0 support WINC1500?

No, WINC1500 will not fit into the M0 flash space.

---

### Can AVR's such as ATmega328 or ATmega2560 run CircuitPython?

No.

---

# Advanced Serial Console on Mac

Connecting to the serial console on Mac does not require installing any drivers or extra software. You'll use a terminal program to find your board, and `screen` to connect to it. Terminal and `screen` both come installed by default.

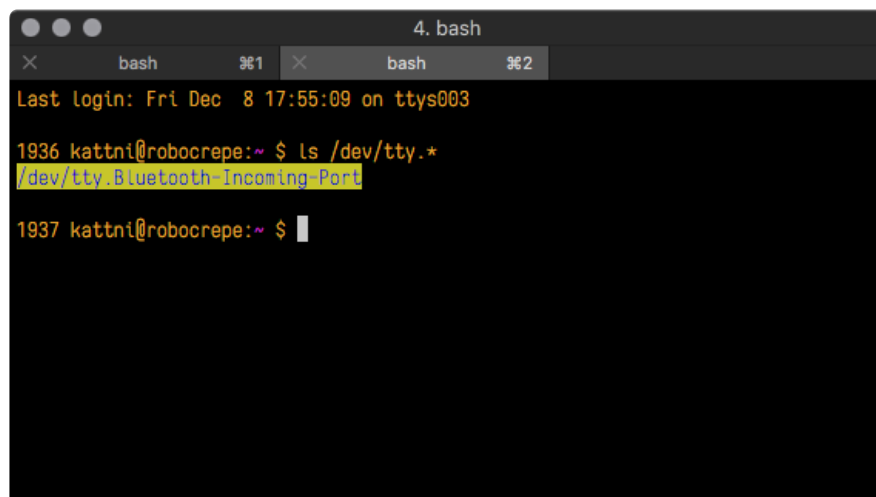
## What's the Port?

First you'll want to find out which serial port your board is using. When you plug your board in to USB on your computer, it connects to a serial port. The port is like a door through which your board can communicate with your computer using USB.

The easiest way to determine which port the board is using is to first check **without** the board plugged in. Open Terminal and type the following:

```
ls /dev/tty.*
```

Each serial connection shows up in the `/dev/` directory. It has a name that starts with `tty.`. The command `ls` shows you a list of items in a directory. You can use `*` as a wildcard, to search for files that start with the same letters but end in something different. In this case, you're asking to see all of the listings in `/dev/` that start with `tty.` and end in anything. This will show us the current serial connections.

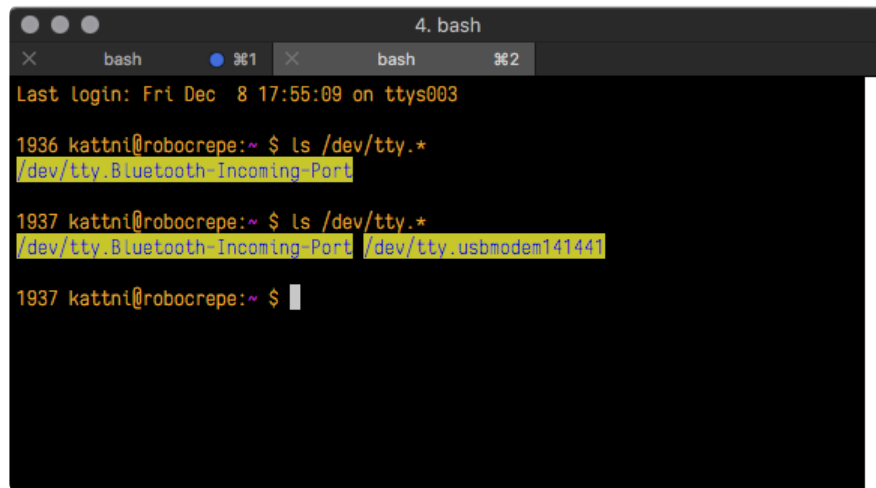


```
4. bash
bash %1 bash %2
Last login: Fri Dec 8 17:55:09 on ttys003
1936 kattni@robocrepe:~ $ ls /dev/tty.*
/dev/tty.Bluetooth-Incoming-Port
1937 kattni@robocrepe:~ $
```

Now, plug your board. In Terminal, type:

```
ls /dev/tty.*
```

This will show you the current serial connections, which will now include your board.



```
4. bash
bash %1 bash %2
Last login: Fri Dec 8 17:55:09 on ttys003
1936 kattni@robocrepe:~ $ ls /dev/tty.*
/dev/tty.Bluetooth-Incoming-Port
1937 kattni@robocrepe:~ $ ls /dev/tty.*
/dev/tty.Bluetooth-Incoming-Port /dev/tty.usbmodem141441
1937 kattni@robocrepe:~ $
```

A new listing has appeared called `/dev/tty.usbmodem141441`. The `tty.usbmodem141441` part of this listing is the name the example board is using. Yours will be called something similar.

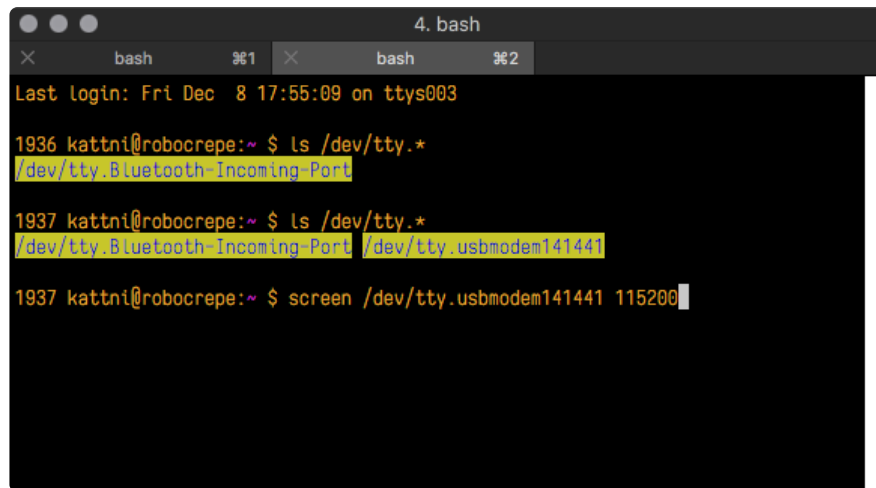
Using Linux, a new listing has appeared called `/dev/ttyACM0`. The `ttyACM0` part of this listing is the name the example board is using. Yours will be called something similar.

## Connect with screen

Now that you know the name your board is using, you're ready connect to the serial console. You're going to use a command called `screen`. The `screen` command is included with MacOS. To connect to the serial console, use Terminal. Type the following command, replacing `board_name` with the name you found your board is using:

```
screen /dev/tty.board_name 115200
```

The first part of this establishes using the `screen` command. The second part tells screen the name of the board you're trying to use. The third part tells screen what baud rate to use for the serial connection. The baud rate is the speed in bits per second that data is sent over the serial connection. In this case, the speed required by the board is 115200 bits per second.

A terminal window titled '4. bash' with two tabs labeled 'bash %1' and 'bash %2'. The terminal shows the following commands and output:

```
Last login: Fri Dec 8 17:55:09 on ttys003
1936 kattni@robocrepe:~ $ ls /dev/tty.*
/dev/tty.Bluetooth-Incoming-Port
1937 kattni@robocrepe:~ $ ls /dev/tty.*
/dev/tty.Bluetooth-Incoming-Port /dev/tty.usbmodem141441
1937 kattni@robocrepe:~ $ screen /dev/tty.usbmodem141441 115200
```

Press enter to run the command. It will open in the same window. If no code is running, the window will be blank. Otherwise, you'll see the output of your code.

Great job! You've connected to the serial console!

---

## Advanced Serial Console on Windows

### Windows 7 and 8.1

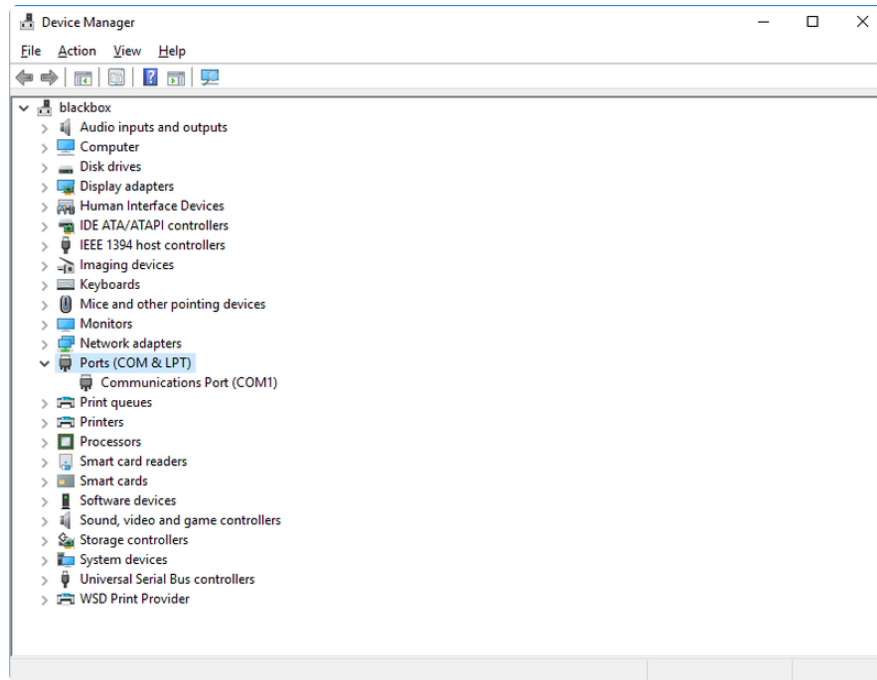
If you're using Windows 7 (or 8 or 8.1), you'll need to install drivers. See the [Windows 7 and 8.1 Drivers page \(https://adafru.it/VuB\)](https://adafru.it/VuB) for details. You will not need to install drivers on Mac, Linux or Windows 10.

You are strongly encouraged to upgrade to Windows 10 if you are still using Windows 7 or Windows 8 or 8.1. Windows 7 has reached end-of-life and no longer receives security updates. A free upgrade to Windows 10 is [still available \(https://adafru.it/RWc\)](https://adafru.it/RWc).

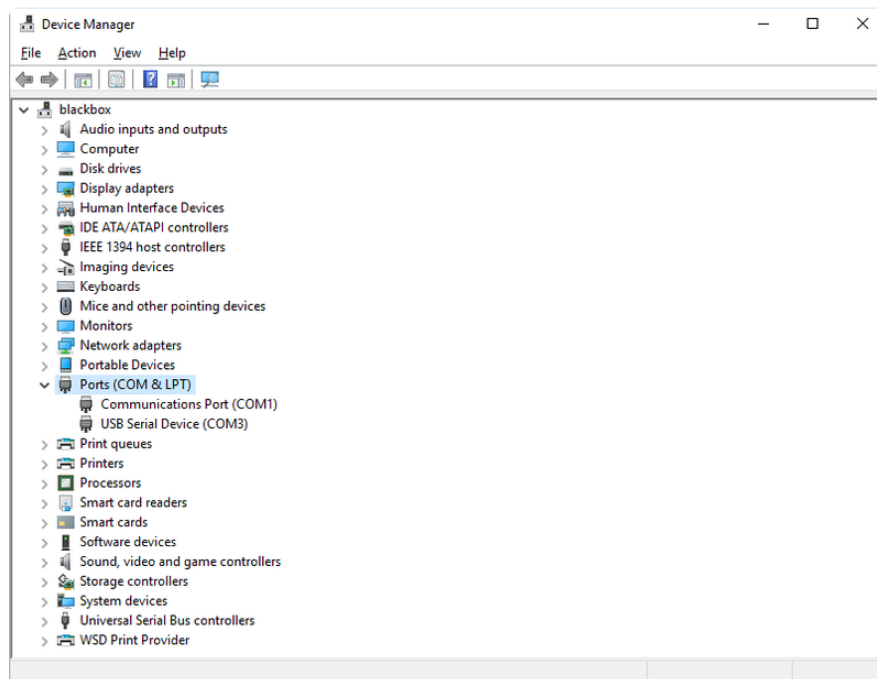
### What's the COM?

First, you'll want to find out which serial port your board is using. When you plug your board in to USB on your computer, it connects to a serial port. The port is like a door through which your board can communicate with your computer using USB.

You'll use Windows Device Manager to determine which port the board is using. The easiest way to determine which port the board is using is to first check **without** the board plugged in. Open Device Manager. Click on Ports (COM & LPT). You should find something already in that list with (COM#) after it where # is a number.



Now plug in your board. The Device Manager list will refresh and a new item will appear under Ports (COM & LPT). You'll find a different (COM#) after this item in the list.



Sometimes the item will refer to the name of the board. Other times it may be called something like USB Serial Device, as seen in the image above. Either way, there is a new (COM#) following the name. This is the port your board is using.



# Install Putty

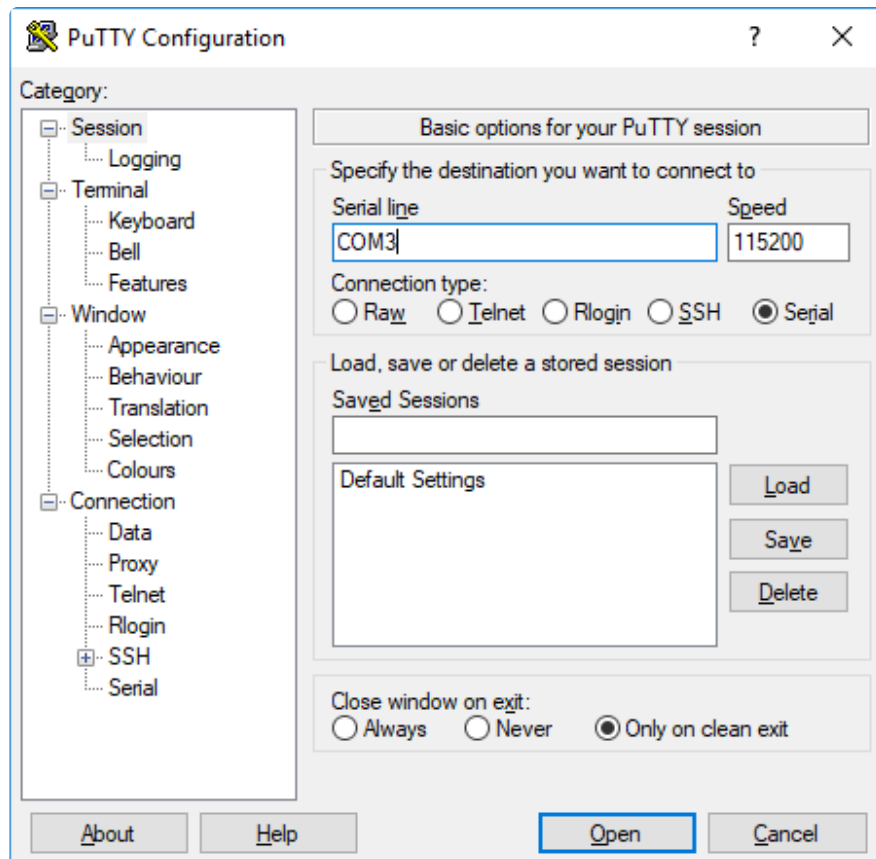
If you're using Windows, you'll need to download a terminal program. You're going to use PuTTY.

The first thing to do is download the [latest version of PuTTY \(https://adafru.it/Bf1\)](https://adafru.it/Bf1). You'll want to download the Windows installer file. It is most likely that you'll need the 64-bit version. Download the file and install the program on your machine. If you run into issues, you can try downloading the 32-bit version instead. However, the 64-bit version will work on most PCs.

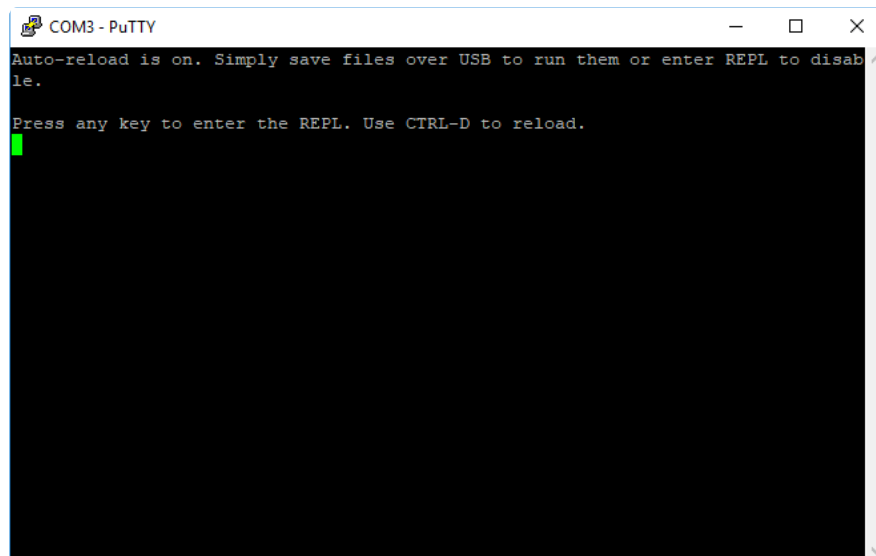
Now you need to open PuTTY.

- Under **Connection type**: choose the button next to **Serial**.
- In the box under **Serial line**, enter the serial port you found that your board is using.
- In the box under **Speed**, enter 115200. This called the baud rate, which is the speed in bits per second that data is sent over the serial connection. For boards with built in USB it doesn't matter so much but for ESP8266 and other board with a separate chip, the speed required by the board is 115200 bits per second. So you might as well just use 115200!

If you want to save those settings for later, use the options under **Load, save or delete a stored session**. Enter a name in the box under **Saved Sessions**, and click the **Save** button on the right.



Once your settings are entered, you're ready to connect to the serial console. Click "Open" at the bottom of the window. A new window will open.



If no code is running, the window will either be blank or will look like the window above. Now you're ready to see the results of your code.

Great job! You've connected to the serial console!

---

# Troubleshooting

From time to time, you will run into issues when working with CircuitPython. Here are a few things you may encounter and how to resolve them.

As CircuitPython development continues and there are new releases, Adafruit will stop supporting older releases. Visit <https://circuitpython.org/downloads> to download the latest version of CircuitPython for your board. You must download the CircuitPython Library Bundle that matches your version of CircuitPython. Please update CircuitPython and then visit <https://circuitpython.org/libraries> to download the latest Library Bundle.

## Always Run the Latest Version of CircuitPython and Libraries

As CircuitPython development continues and there are new releases, Adafruit will stop supporting older releases. **You need to [update to the latest CircuitPython](https://adafru.it/Em8)**. (<https://adafru.it/Em8>).

You need to download the CircuitPython Library Bundle that matches your version of CircuitPython. **Please update CircuitPython and then [download the latest bundle](https://adafru.it/ENC)** (<https://adafru.it/ENC>).

As new versions of CircuitPython are released, Adafruit will stop providing the previous bundles as automatically created downloads on the Adafruit CircuitPython Library Bundle repo. If you must continue to use an earlier version, you can still download the appropriate version of `mpy-cross` from the particular release of CircuitPython on the CircuitPython repo and create your own compatible .mpy library files. **However, it is best to update to the latest for both CircuitPython and the library bundle.**

### I have to continue using CircuitPython 7.x or earlier. Where can I find compatible libraries?

Adafruit is no longer building or supporting the CircuitPython 7.x or earlier library bundles. You are highly encouraged to [update CircuitPython to the latest version](https://adafru.it/Em8) (<https://adafru.it/Em8>) and use [the current version of the libraries](https://adafru.it/ENC) (<https://adafru.it/ENC>). However, if for some reason you cannot update, links to the previous bundles are available in the [FAQ](https://adafru.it/FwY) (<https://adafru.it/FwY>).

# macOS Sonoma 14.x: Disk Errors Writing to CIRCUITPY

macOS Sonoma before 14.4 beta 2 takes many seconds to complete writes to small FAT drives, 8MB or smaller. This causes errors when writing to CIRCUITPY. The best solution is to remount the CIRCUITPY drive after it is automatically mounted. Or consider downgrading back to Ventura if that works for you. This problem is being tracked in [CircuitPython GitHub issue 8449 \(https://adafru.it/18ea\)](https://adafru.it/18ea).

Here is a shell script to do this remount conveniently (courtesy [@czei in GitHub \(https://adafru.it/18ea\)](https://adafru.it/18ea)). Copy the code here into a file named, say, **remount-CIRCUITPY.sh**. Place the file in a directory on your PATH, or in some other convenient place.

macOS Sonoma 14.4 beta and after does not have the problem above, but does take an inordinately long time to write to FAT drives of size 1GB or less (40 times longer than 2GB drives). This problem is being tracked in [CircuitPython GitHub issue 8918 \(https://adafru.it/19iD\)](https://adafru.it/19iD).

```
#!/bin/sh
#
# This works around bug where, by default, macOS 14.x writes part of a file
# immediately, and then doesn't update the directory for 20-60 seconds, causing
# the file system to be corrupted.
#
disky=`df | grep CIRCUITPY | cut -d" " -f1`
sudo umount /Volumes/CIRCUITPY
sudo mkdir /Volumes/CIRCUITPY
sleep 2
sudo mount -v -o noasync -t msdos $disky /Volumes/CIRCUITPY
```

Then in a Terminal window, do this to make this script executable:

```
chmod +x remount-CIRCUITPY.sh
```

Place the file in a directory on your **PATH**, or in some other convenient place.

Now, each time you plug in or reset your CIRCUITPY board, run the file **remount-CIRCUITPY.sh**. You can run it in a Terminal window or you may be able to place it on the desktop or in your dock to run it just by double-clicking.

This will be something of a nuisance but it is the safest solution.

This problem is being tracked in [this CircuitPython issue \(https://adafru.it/18ea\)](https://adafru.it/18ea).

# Bootloader (boardnameBOOT) Drive Not Present

You may have a different board.

Only Adafruit Express boards and the SAMD21 non-Express boards ship with the [UF2 bootloader](https://adafruit.it/zbX) (<https://adafruit.it/zbX>) installed. The Feather M0 Basic, Feather M0 Adalogger, and similar boards use a regular Arduino-compatible bootloader, which does not show a **boardnameBOOT** drive.

## MakeCode

If you are running a [MakeCode](https://adafruit.it/zbY) (<https://adafruit.it/zbY>) program on Circuit Playground Express, press the reset button just once to get the **CPLAYBOOT** drive to show up. Pressing it twice will not work.

## macOS

**DriveDx** and its accompanying **SAT SMART Driver** can interfere with seeing the BOOT drive. [See this forum post](https://adafruit.it/sTc) (<https://adafruit.it/sTc>) for how to fix the problem.

## Windows 10

Did you install the Adafruit Windows Drivers package by mistake, or did you upgrade to Windows 10 with the driver package installed? You don't need to install this package on Windows 10 for most Adafruit boards. The old version (v1.5) can interfere with recognizing your device. Go to **Settings** -> **Apps** and uninstall all the "Adafruit" driver programs.

## Windows 7 or 8.1

To use a CircuitPython-compatible board with Windows 7 or 8.1, you must install a driver. Installation instructions are available [here](https://adafruit.it/VuB) (<https://adafruit.it/VuB>).

Windows 7 and 8.1 have reached end of life. It is [recommended](https://adafruit.it/Amd) (<https://adafruit.it/Amd>) that you upgrade to Windows 10 if possible; an upgrade is probably still free for you. Check [here](https://adafruit.it/Amd) (<https://adafruit.it/Amd>).

The Windows Drivers installer was last updated in November 2020 (v2.5.0.0) . Windows 7 drivers for CircuitPython boards released since then, including RP2040 boards, are not available. There are no plans to release drivers for new boards. The boards work fine on Windows 10.

You should now be done! Test by unplugging and replugging the board. You should see the **CIRCUITPY** drive, and when you double-click the reset button (single click on Circuit Playground Express running MakeCode), you should see the appropriate **boardnameBOOT** drive.

Let us know in the [Adafruit support forums \(https://adafru.it/jlf\)](https://adafru.it/jlf) or on the [Adafruit Discord \(\)](#) if this does not work for you!

## Windows Explorer Locks Up When Accessing **boardnameBOOT** Drive

On Windows, several third-party programs that can cause issues. The symptom is that you try to access the **boardnameBOOT** drive, and Windows or Windows Explorer seems to lock up. These programs are known to cause trouble:

- **AIDA64**: to fix, stop the program. This problem has been reported to AIDA64. They acquired hardware to test, and released a beta version that fixes the problem. This may have been incorporated into the latest release. Please let us know in the forums if you test this.
- **Hard Disk Sentinel**
- **Kaspersky anti-virus**: To fix, you may need to disable Kaspersky completely. Disabling some aspects of Kaspersky does not always solve the problem. This problem has been reported to Kaspersky.
- **ESET NOD32 anti-virus**: There have been problems with at least version 9.0.386.0, solved by uninstallation.

## Copying UF2 to **boardnameBOOT** Drive Hangs at 0% Copied

On Windows, a **Western Digital (WD) utility** that comes with their external USB drives can interfere with copying UF2 files to the **boardnameBOOT** drive. Uninstall that utility to fix the problem.

# CIRCUITPY Drive Does Not Appear or Disappears Quickly

**Kaspersky anti-virus** can block the appearance of the **CIRCUITPY** drive. There has not yet been settings change discovered that prevents this. Complete uninstallation of Kaspersky fixes the problem.

**Norton anti-virus** can interfere with **CIRCUITPY**. A user has reported this problem on Windows 7. The user turned off both Smart Firewall and Auto Protect, and **CIRCUITPY** then appeared.

**Sophos Endpoint** security software [can cause CIRCUITPY to disappear \(https://adafru.it/ELr\)](https://adafru.it/ELr) and the BOOT drive to reappear. It is not clear what causes this behavior.

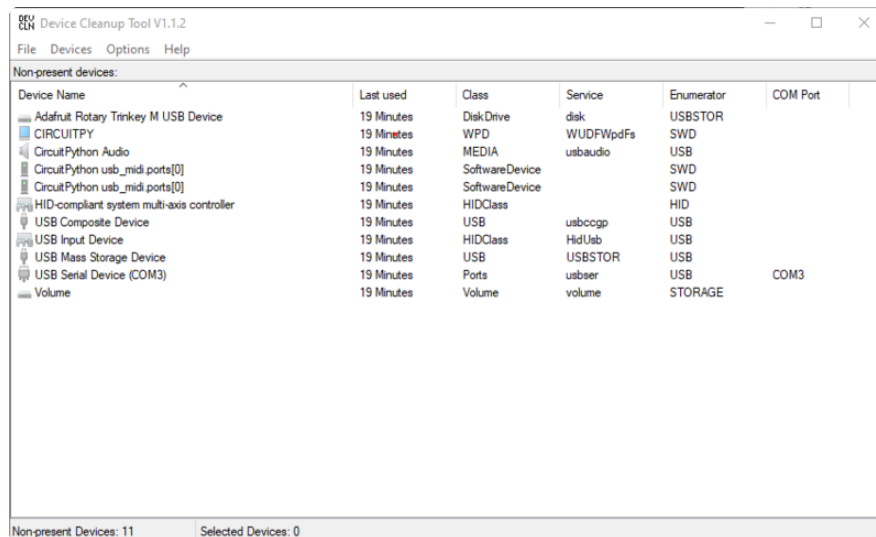
**Samsung Magician** can cause CIRCUITPY to disappear (reported [here \(https://adafru.it/18eb\)](https://adafru.it/18eb) and [here \(https://adafru.it/18ec\)](https://adafru.it/18ec)).

## Device Errors or Problems on Windows

Windows can become confused about USB device installations. This is particularly true of Windows 7 and 8.1. It is [recommended \(https://adafru.it/Amd\)](https://adafru.it/Amd) that you upgrade to Windows 10 if possible; an upgrade is probably still free for you: see this [link \(https://adafru.it/V2a\)](https://adafru.it/V2a).

If not, try cleaning up your USB devices. Use [Uwe Sieber's Device Cleanup Tool \(https://adafru.it/RWd\)](https://adafru.it/RWd) (on that page, scroll down to "Device Cleanup Tool"). Download and unzip the tool. Unplug all the boards and other USB devices you want to clean up. Run the tool as Administrator. You will see a listing like this, probably with many more devices. It is listing all the USB devices that are not currently attached.





Select all the devices you want to remove, and then press Delete. It is usually safe just to select everything. Any device that is removed will get a fresh install when you plug it in. Using the Device Cleanup Tool also discards all the COM port assignments for the unplugged boards. If you have used many Arduino and CircuitPython boards, you have probably seen higher and higher COM port numbers used, seemingly without end. This will fix that problem.

## Serial Console in Mu Not Displaying Anything

There are times when the serial console will accurately not display anything, such as, when no code is currently running, or when code with no serial output is already running before you open the console. However, if you find yourself in a situation where you feel it should be displaying something like an error, consider the following.

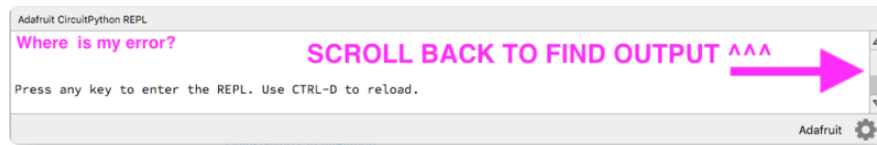
Depending on the size of your screen or Mu window, when you open the serial console, the serial console panel may be very small. This can be a problem. A basic CircuitPython error takes 10 lines to display!

```
Auto-reload is on. Simply save files over USB to run them or enter REPL to disable.
code.py output:
Traceback (most recent call last):
  File "code.py", line 7
SyntaxError: invalid syntax
```

```
Press any key to enter the REPL. Use CTRL-D to reload.
```

More complex errors take even more lines!

Therefore, if your serial console panel is five lines tall or less, you may only see blank lines or blank lines followed by **Press any key to enter the REPL. Use CTRL-D to reload.** . If this is the case, you need to either mouse over the top of the panel to utilise the option to resize the serial panel, or use the scrollbar on the right side to scroll up and find your message.



This applies to any kind of serial output whether it be error messages or print statements. So before you start trying to debug your problem on the hardware side, be sure to check that you haven't simply missed the serial messages due to serial output panel height.

## code.py Restarts Constantly

CircuitPython will restart **code.py** if you or your computer writes to something on the CIRCUITPY drive. This feature is called auto-reload, and lets you test a change to your program immediately.

Some utility programs, such as backup, anti-virus, or disk-checking apps, will write to the CIRCUITPY as part of their operation. Sometimes they do this very frequently, causing constant restarts.

**Acronis True Image** and related Acronis programs on Windows are known to cause this problem. It is possible to prevent this by [disabling the " \(https://adafru.it/XDZ\)Acronis Managed Machine Service Mini" \(https://adafru.it/XDZ\)](https://adafru.it/XDZ).

If you cannot stop whatever is causing the writes, you can disable auto-reload by putting this code in **boot.py** or **code.py**:

```
import supervisor
supervisor.runtime.autoreload = False
```

## CircuitPython RGB Status Light

Nearly all CircuitPython-capable boards have a single NeoPixel or DotStar RGB LED on the board that indicates the status of CircuitPython. A few boards designed before CircuitPython existed, such as the Feather M0 Basic, do not.

Circuit Playground Express and Circuit Playground Bluefruit have multiple RGB LEDs, but do NOT have a status LED. The LEDs are all green when in the bootloader. In versions before 7.0.0, they do NOT indicate any status while running CircuitPython.

## CircuitPython 7.0.0 and Later

The status LED blinks were changed in CircuitPython 7.0.0 in order to save battery power and simplify the blinks. These blink patterns will occur on single color LEDs when the board does not have any RGB LEDs. Speed and blink count also vary for this reason.

On start up, the LED will blink **YELLOW** multiple times for 1 second. Pressing the RESET button (or on Espressif, the BOOT button) during this time will restart the board and then enter safe mode. On Bluetooth capable boards, after the yellow blinks, there will be a set of faster blue blinks. Pressing reset during the **BLUE** blinks will clear Bluetooth information and start the device in discoverable mode, so it can be used with a BLE code editor.

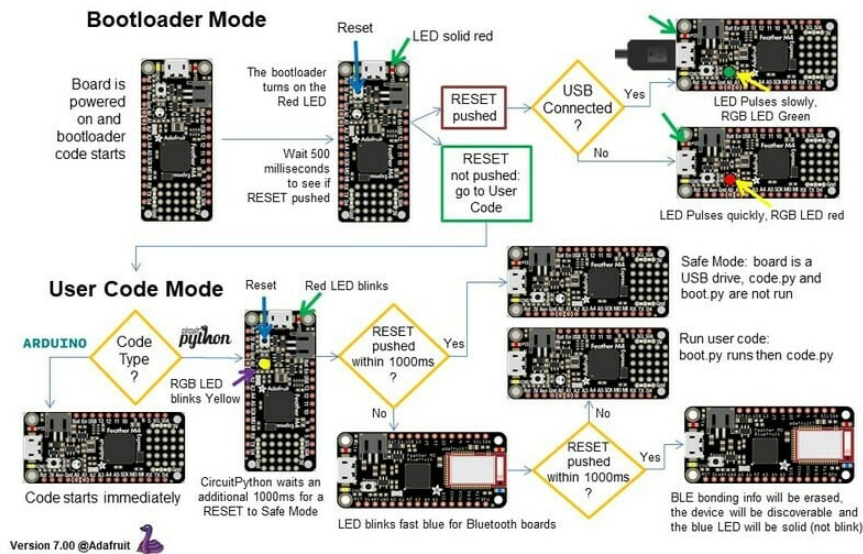
Once started, CircuitPython will blink a pattern every 5 seconds when no user code is running to indicate why the code stopped:

- 1 **GREEN** blink: Code finished without error.
- 2 **RED** blinks: Code ended due to an exception. Check the serial console for details.
- 3 **YELLOW** blinks: CircuitPython is in safe mode. No user code was run. Check the serial console for safe mode reason.

When in the REPL, CircuitPython will set the status LED to **WHITE**. You can change the LED color from the REPL. The status indicator will not persist on non-NeoPixel or DotStar LEDs.

## The CircuitPython Boot Sequence

Version 7.0 and later



## CircuitPython 6.3.0 and earlier

Here's what the colors and blinking mean:

- steady **GREEN**: `code.py` (or `code.txt`, `main.py`, or `main.txt`) is running
- pulsing **GREEN**: `code.py` (etc.) has finished or does not exist
- steady **YELLOW** at start up: (4.0.0-alpha.5 and newer) CircuitPython is waiting for a reset to indicate that it should start in safe mode
- pulsing **YELLOW**: Circuit Python is in safe mode: it crashed and restarted
- steady **WHITE**: REPL is running
- steady **BLUE**: `boot.py` is running

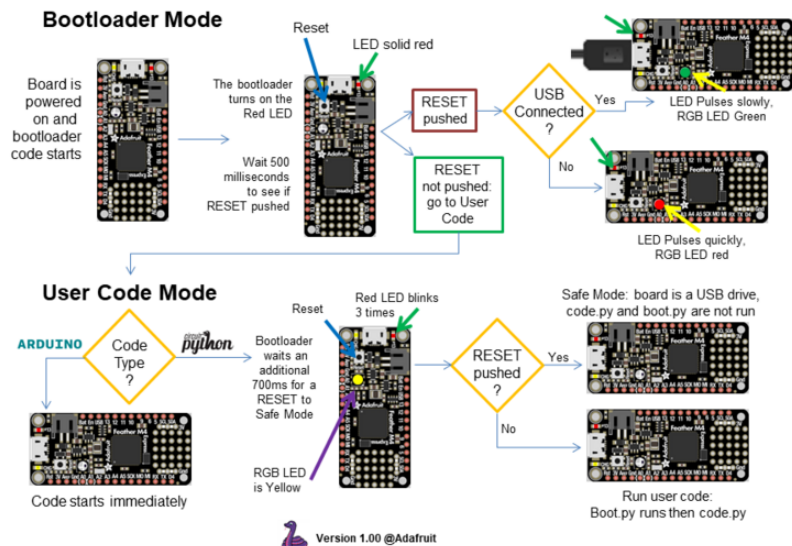
Colors with multiple flashes following indicate a Python exception and then indicate the line number of the error. The color of the first flash indicates the type of error:

- **GREEN**: IndentationError
- **CYAN**: SyntaxError
- **WHITE**: NameError
- **ORANGE**: OSError
- **PURPLE**: ValueError
- **YELLOW**: other error

These are followed by flashes indicating the line number, including place value. **WHITE** flashes are thousands' place, **BLUE** are hundreds' place, **YELLOW** are tens' place, and **CYAN** are one's place. So for example, an error on line 32 would flash

**YELLOW** three times and then **CYAN** two times. Zeroes are indicated by an extra-long dark gap.

### The CircuitPython Boot Sequence



## Serial console showing **ValueError: Incompatible .mpy file**

This error occurs when importing a module that is stored as a **.mpy** binary file that was generated by a different version of CircuitPython than the one its being loaded into. In particular, the mpy binary format changed between CircuitPython versions 6.x and 7.x, 2.x and 3.x, and 1.x and 2.x.

So, for instance, if you upgraded to CircuitPython 7.x from 6.x you'll need to download a newer version of the library that triggered the error on **import**. All libraries are available in the [Adafruit bundle \(https://adafru.it/y8E\)](https://adafru.it/y8E).

## CIRCUITPY Drive Issues

You may find that you can no longer save files to your **CIRCUITPY** drive. You may find that your **CIRCUITPY** stops showing up in your file explorer, or shows up as **NO\_NAME**. These are indicators that your filesystem has issues. When the **CIRCUITPY** disk is not safely ejected before being reset by the button or being disconnected from USB, it may corrupt the flash drive. It can happen on Windows, Mac or Linux, though it is more common on Windows.

Be aware, if you have used Arduino to program your board, CircuitPython is no longer able to provide the USB services. You will need to reload CircuitPython to resolve this situation.

The easiest first step is to reload CircuitPython. Double-tap reset on the board so you get a **boardnameBOOT** drive rather than a **CIRCUITPY** drive, and copy the latest version of CircuitPython (.uf2) back to the board. This may restore **CIRCUITPY** functionality.

If reloading CircuitPython does not resolve your issue, the next step is to try putting the board into safe mode.

## Safe Mode

Whether you've run into a situation where you can no longer edit your **code.py** on your **CIRCUITPY** drive, your board has gotten into a state where **CIRCUITPY** is read-only, or you have turned off the **CIRCUITPY** drive altogether, safe mode can help.

**Safe mode** in CircuitPython does not run any user code on startup, and disables auto-reload. This means a few things. First, safe mode bypasses any code in **boot.py** (where you can set **CIRCUITPY** read-only or turn it off completely). Second, it does not run the code in **code.py**. And finally, it does not automatically soft-reload when data is written to the **CIRCUITPY** drive.

Therefore, whatever you may have done to put your board in a non-interactive state, safe mode gives you the opportunity to correct it without losing all of the data on the **CIRCUITPY** drive.

### Entering Safe Mode in CircuitPython 7.x and Later

You can enter safe by pressing reset during the right time when the board boots. Immediately after the board starts up or resets, it waits one second. On some boards, the onboard status LED will blink yellow during that time. If you press reset during that one second period, the board will start up in safe mode. It can be difficult to react to the yellow LED, so you may want to think of it simply as a "slow" double click of the reset button. (Remember, a fast double click of reset enters the bootloader.)

### Entering Safe Mode in CircuitPython 6.x

You can enter safe by pressing reset during the right time when the board boots.. Immediately after the board starts up or resets, it waits 700ms. On some boards, the

onboard status LED (highlighted in green above) will turn solid yellow during this time. If you press reset during that 700ms, the board will start up in safe mode. It can be difficult to react to the yellow LED, so you may want to think of it simply as a slow double click of the reset button. (Remember, a fast double click of reset enters the bootloader.)

## In Safe Mode

Once you've entered safe mode successfully in CircuitPython 6.x, the LED will pulse yellow.

If you successfully enter safe mode on CircuitPython 7.x, the LED will intermittently blink yellow three times.

If you connect to the serial console, you'll find the following message.

```
Auto-reload is off.  
Running in safe mode! Not running saved code.  
  
CircuitPython is in safe mode because you pressed the reset button during boot.  
Press again to exit safe mode.  
  
Press any key to enter the REPL. Use CTRL-D to reload.
```

You can now edit the contents of the **CIRCUITPY** drive. Remember, your code will not run until you press the reset button, or unplug and plug in your board, to get out of safe mode.

At this point, you'll want to remove any user code in **code.py** and, if present, the **boot.py** file from **CIRCUITPY**. Once removed, tap the reset button, or unplug and plug in your board, to restart CircuitPython. This will restart the board and may resolve your drive issues. If resolved, you can begin coding again as usual.

If safe mode does not resolve your issue, the board must be completely erased and CircuitPython must be reloaded onto the board.

You WILL lose everything on the board when you complete the following steps. If possible, make a copy of your code before continuing.



## To erase CIRCUITPY: `storage.erase_filesystem()`

CircuitPython includes a built-in function to erase and reformat the filesystem. If you have a version of CircuitPython older than 2.3.0 on your board, you can [update to the newest version \(https://adafru.it/Amd\)](https://adafru.it/Amd) to do this.

1. [Connect to the CircuitPython REPL \(https://adafru.it/Bec\)](https://adafru.it/Bec) using Mu or a terminal program.
2. Type the following into the REPL:

```
>>> import storage
>>> storage.erase_filesystem()
```

CIRCUITPY will be erased and reformatted, and your board will restart. That's it!

## Erase CIRCUITPY Without Access to the REPL

If you can't access the REPL, or you're running a version of CircuitPython previous to 2.3.0 and you don't want to upgrade, there are options available for some specific boards.

The options listed below are considered to be the "old way" of erasing your board. The method shown above using the REPL is highly recommended as the best method for erasing your board.

If at all possible, it is recommended to use the REPL to erase your CIRCUITPY drive. The REPL method is explained above.

## For the specific boards listed below:

If the board you are trying to erase is listed below, follow the steps to use the file to erase your board.

1. Download the correct erase file:

**Circuit Playground Express**

<https://adafru.it/AdI>

**Feather M0 Express**

<https://adafru.it/AdJ>

**Feather M4 Express**

<https://adafru.it/EVK>

**Metro M0 Express**

<https://adafru.it/AdK>

**Metro M4 Express QSPI Eraser**

<https://adafru.it/EoM>

**Trellis M4 Express (QSPI)**

<https://adafru.it/DjD>

**Grand Central M4 Express (QSPI)**

<https://adafru.it/DBA>

**PyPortal M4 Express (QSPI)**

<https://adafru.it/Eca>

**Circuit Playground Bluefruit (QSPI)**

<https://adafru.it/Gnc>

**Monster M4SK (QSPI)**

<https://adafru.it/GAN>

**PyBadge/PyGamer QSPI Eraser.UF2**

<https://adafru.it/GAO>

**CLUE\_Flash\_Erase.UF2**

<https://adafru.it/Jat>

**Matrix\_Portal\_M4\_(QSPI).UF2**

<https://adafru.it/Q5B>

**RP2040 boards (flash\_nuke.uf2)**

<https://adafru.it/18ed>

2. Double-click the reset button on the board to bring up the **boardnameBOOT** drive.
3. Drag the erase **.uf2** file to the **boardnameBOOT** drive.
4. The status LED will turn yellow or blue, indicating the erase has started.
5. After approximately 15 seconds, the status LED will light up green. On the NeoTrellis M4 this is the first NeoPixel on the grid
6. Double-click the reset button on the board to bring up the **boardnameBOOT** drive.
7. [Drag the appropriate latest release of CircuitPython \(https://adafru.it/Em8\)](https://adafru.it/Em8) **.uf2** file to the **boardnameBOOT** drive.

It should reboot automatically and you should see **CIRCUITPY** in your file explorer again.

If the LED flashes red during step 5, it means the erase has failed. Repeat the steps starting with 2.

[If you haven't already downloaded the latest release of CircuitPython for your board, check out the installation page \(https://adafru.it/Amd\)](https://adafru.it/Amd). You'll also need to load your code and reinstall your libraries!

## For SAMD21 non-Express boards that have a UF2 bootloader:

Any SAMD21-based microcontroller that does not have external flash available is considered a SAMD21 non-Express board. Non-Express boards that have a UF2 bootloader include Trinket M0, GEMMA M0, QT Py M0, and the SAMD21-based Trinky boards.

If you are trying to erase a SAMD21 non-Express board, follow these steps to erase your board.

1. Download the erase file:

**SAMD21 non-Express Boards**

<https://adafru.it/VB->

2. Double-click the reset button on the board to bring up the **boardnameBOOT** drive.
3. Drag the erase **.uf2** file to the **boardnameBOOT** drive.
4. The boot LED will start flashing again, and the **boardnameBOOT** drive will

reappear.

5. [Drag the appropriate latest release CircuitPython \(https://adafru.it/Em8\)](https://adafru.it/Em8) .uf2 file to the boardnameBOOT drive.

It should reboot automatically and you should see **CIRCUITPY** in your file explorer again.

[If you haven't already downloaded the latest release of CircuitPython for your board, check out the installation page \(https://adafru.it/Amd\)](https://adafru.it/Amd) YYou'll also need to load your code and reinstall your libraries!

## For SAMD21 non-Express boards that do not have a UF2 bootloader:

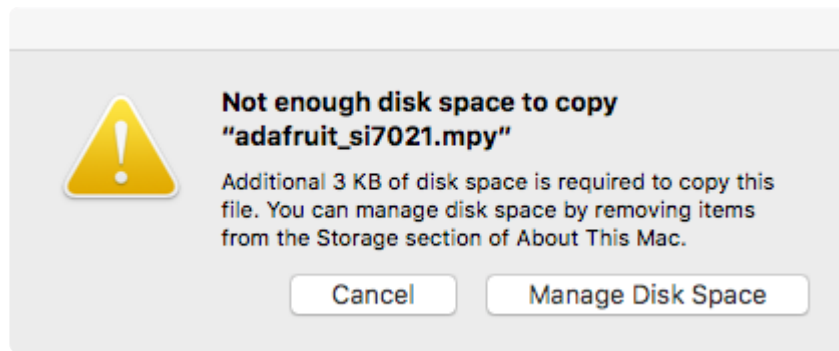
Any SAMD21-based microcontroller that does not have external flash available is considered a SAMD21 non-Express board. Non-Express boards that do **not** have a UF2 bootloader include the Feather M0 Basic Proto, Feather Adalogger, or the Arduino Zero.

If you are trying to erase a non-Express board that does not have a UF2 bootloader, [follow these directions to reload CircuitPython using \*\*bossac\*\* \(https://adafru.it/Bed\)](https://adafru.it/Bed), which will erase and re-create **CIRCUITPY**.

## Running Out of File Space on SAMD21 Non-Express Boards

Any SAMD21-based microcontroller that does not have external flash available is considered a SAMD21 non-Express board. This includes boards like the Trinket M0, GEMMA M0, QT Py M0, and the SAMD21-based Trinkey boards.

The file system on the board is very tiny. (Smaller than an ancient floppy disk.) So, its likely you'll run out of space but don't panic! There are a number of ways to free up space.



## Delete something!

The simplest way of freeing up space is to delete files from the drive. Perhaps there are libraries in the **lib** folder that you aren't using anymore or test code that isn't in use. Don't delete the **lib** folder completely, though, just remove what you don't need.

The board ships with the Windows 7 serial driver too! Feel free to delete that if you don't need it or have already installed it. It's ~12KiB or so.

## Use tabs

One unique feature of Python is that the indentation of code matters. Usually the recommendation is to indent code with four spaces for every indent. In general, that is recommended too. **However**, one trick to storing more human-readable code is to use a single tab character for indentation. This approach uses 1/4 of the space for indentation and can be significant when you're counting bytes.

## On MacOS?

MacOS loves to generate hidden files. Luckily you can disable some of the extra hidden files that macOS adds by running a few commands to disable search indexing and create zero byte placeholders. Follow the steps below to maximize the amount of space available on macOS.

## Prevent & Remove MacOS Hidden Files

First find the volume name for your board. With the board plugged in run this command in a terminal to list all the volumes:

```
ls -l /Volumes
```

Look for a volume with a name like **CIRCUITPY** (the default for CircuitPython). The full path to the volume is the **/Volumes/CIRCUITPY** path.

Now follow the [steps from this question \(https://adafru.it/u1c\)](https://adafru.it/u1c) to run these terminal commands that stop hidden files from being created on the board:

```
mdutil -i off /Volumes/CIRCUITPY
cd /Volumes/CIRCUITPY
rm -rf .{,._}{fseventsd,Spotlight-V*,Trashes}
mkdir .fseventsd
touch .fseventsd/no_log .metadata_never_index .Trashes
cd -
```

Replace **/Volumes/CIRCUITPY** in the commands above with the full path to your board's volume if it's different. At this point all the hidden files should be cleared from the board and some hidden files will be prevented from being created.

Alternatively, with CircuitPython 4.x and above, the special files and folders mentioned above will be created automatically if you erase and reformat the filesystem. **WARNING: Save your files first!** Do this in the REPL:

```
>>> import storage
>>> storage.erase_filesystem()
```

However there are still some cases where hidden files will be created by MacOS. In particular if you copy a file that was downloaded from the internet it will have special metadata that MacOS stores as a hidden file. Luckily you can run a copy command from the terminal to copy files **without** this hidden metadata file. See the steps below.

## Copy Files on MacOS Without Creating Hidden Files

Once you've disabled and removed hidden files with the above commands on macOS you need to be careful to copy files to the board with a special command that prevents future hidden files from being created. Unfortunately you **cannot** use drag and drop copy in Finder because it will still create these hidden extended attribute files in some cases (for files downloaded from the internet, like Adafruit's modules).

To copy a file or folder use the **-X** option for the **cp** command in a terminal. For example to copy a **file\_name.mpy** file to the board use a command like:

```
cp -X file_name.mpy /Volumes/CIRCUITPY
```

(Replace **file\_name.mpy** with the name of the file you want to copy.)

Or to copy a folder and all of the files and folders contained within, use a command like:

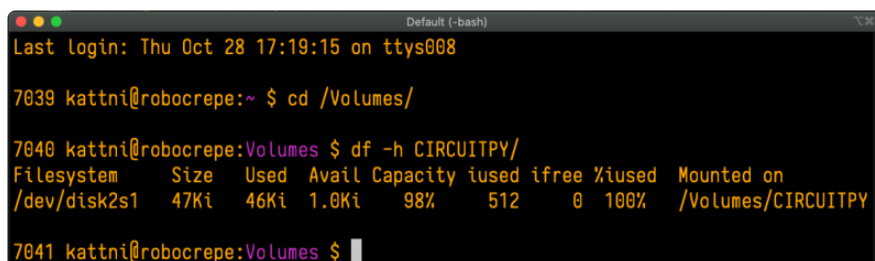
```
cp -rX folder_to_copy /Volumes/CIRCUITPY
```

If you are copying to the **lib** folder, or another folder, make sure it exists before copying.

```
# if lib does not exist, you'll create a file named lib !
cp -X file_name.mpy /Volumes/CIRCUITPY/lib
# This is safer, and will complain if a lib folder does not exist.
cp -X file_name.mpy /Volumes/CIRCUITPY/lib/
```

## Other MacOS Space-Saving Tips

If you'd like to see the amount of space used on the drive and manually delete hidden files here's how to do so. First, move into the **Volumes/** directory with `cd /Volumes/`, and then list the amount of space used on the **CIRCUITPY** drive with the `df` command.



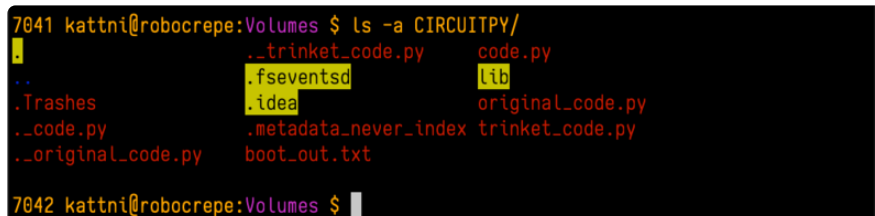
```
Default (-bash)
Last login: Thu Oct 28 17:19:15 on ttys008

7039 kattni@robocrepe:~ $ cd /Volumes/

7040 kattni@robocrepe:Volumes $ df -h CIRCUITPY/
Filesystem      Size  Used Avail Capacity iused ifree %used  Mounted on
/dev/disk2s1    47Ki  46Ki  1.0Ki   98%    512     0 100%  /Volumes/CIRCUITPY

7041 kattni@robocrepe:Volumes $
```

That's not very much space left! The next step is to show a list of the files currently on the **CIRCUITPY** drive, including the hidden files, using the `ls` command. You cannot use Finder to do this, you must do it via command line!



```
7041 kattni@robocrepe:Volumes $ ls -a CIRCUITPY/
.          .trinket_code.py  code.py
..         .fseventsd        lib
.Trashes   .idea             original_code.py
._code.py  .metadata_never_index trinket_code.py
._original_code.py boot_out.txt

7042 kattni@robocrepe:Volumes $
```

There are a few of the hidden files that MacOS loves to generate, all of which begin with a `._` before the file name. Remove the `._` files using the `rm` command. You can remove them all once by running `rm CIRCUITPY/._*`. The `*` acts as a wildcard to apply the command to everything that begins with `._` at the same time.



```
7042 kattni@robocrepe:Volumes $ rm CIRCUITPY/._*

7043 kattni@robocrepe:Volumes $
```



Finally, you can run `df` again to see the current space used.

```
7043 kattni@robocrepe:Volumes $ df -h CIRCUITPY/
Filesystem      Size  Used Avail Capacity iused ifree %used  Mounted on
/dev/disk2s1    47Ki  34Ki  13Ki   73%     512     0 100%  /Volumes/CIRCUITPY
7044 kattni@robocrepe:Volumes $
```

Nice! You have 12Ki more than before! This space can now be used for libraries and code!

## Device Locked Up or Boot Looping

In rare cases, it may happen that something in your `code.py` or `boot.py` files causes the device to get locked up, or even go into a boot loop. A boot loop occurs when the board reboots repeatedly and never fully loads. These are not caused by your everyday Python exceptions, typically it's the result of a deeper problem within CircuitPython. In this situation, it can be difficult to recover your device if **CIRCUITPY** is not allowing you to modify the `code.py` or `boot.py` files. Safe mode is one recovery option. When the device boots up in safe mode it will not run the `code.py` or `boot.py` scripts, but will still connect the **CIRCUITPY** drive so that you can remove or modify those files as needed.

The method used to manually enter safe mode can be different for different devices. It is also very similar to the method used for getting into bootloader mode, which is a different thing. So it can take a few tries to get the timing right. If you end up in bootloader mode, no problem, you can try again without needing to do anything else.

### For most devices:

Press the reset button, and then when the RGB status LED blinks yellow, press the reset button again. Since your reaction time may not be that fast, try a "slow" double click, to catch the yellow LED on the second click.

### For ESP32-S2 based devices:

Press and release the reset button, then press and release the boot button about 3/4 of a second later.

Refer to the diagrams above for boot sequence details.

---

## Adafruit CircuitPython TrellisM4 Library

We've written a library that makes it super easy to use the buttons and NeoPixel LEDs on your NeoTrellis M4 Express board. It allows you to address both the buttons and

the NeoPixels using coordinates (i.e. `(0, 0)`) so it's easy to know which button and NeoPixel you're working with. It also allows you to tell the code that you've rotated your board and then alters the coordinates to match the orientation. Let's take a look!

## Installing the CircuitPython TrellisM4 Library

First you need to make sure you've loaded the [Adafruit CircuitPython TrellisM4](https://adafru.it/CYr) (<https://adafru.it/CYr>) library onto your board. Download the latest [CircuitPython Library Bundle](https://adafru.it/ENC) (<https://adafru.it/ENC>) that corresponds to the version of Circuit Python you're using (as of this writing that would be **4.x**), unzip the downloaded file, open the folder, and copy the **lib** folder found inside to your **CIRCUITPY** drive.

If you're loading libraries individually, ensure you have the following in your **lib** folder:

- **adafruit\_trellism4.mpy**
- **adafruit\_matrixkeypad.mpy**
- **neopixel.mpy**

Check out [the CircuitPython Libraries page of this guide](https://adafru.it/CYo) (<https://adafru.it/CYo>) for a detailed explanation of how to load the library bundle on your board.

## TrellisM4 Library Features

With this library, you can optionally specify the orientation of the board at the beginning of your code when you initialise the library by setting:

- **rotation** - Specify the orientation of your board in 90 degree increments. **Default is 0**. When **rotation** is not specified or **rotation=0**, the code assumes the board is oriented with the USB port facing away from you or pointing upward. Acceptable rotations are: **0**, **90**, **180**, and **270**.

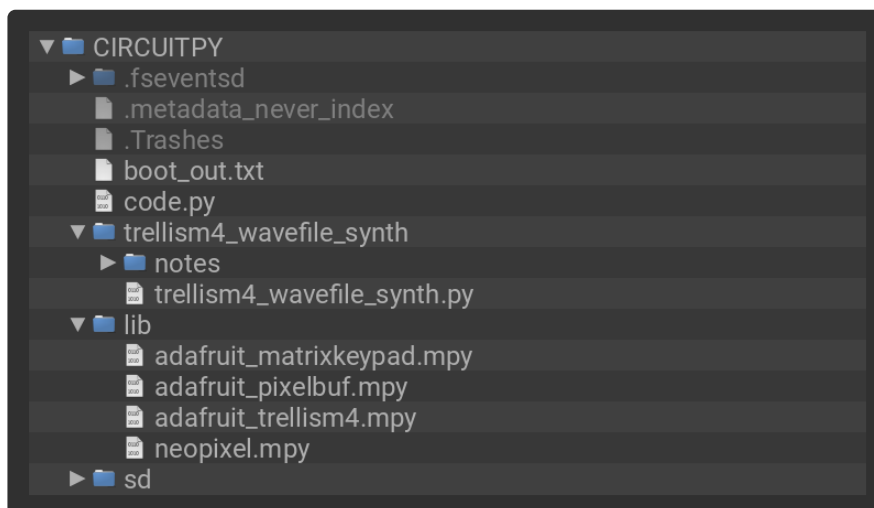
You can use the following properties to interact with your NeoTrellis M4 Express:

- **pressed\_keys** - A list of tuples of currently pressed button coordinates.
- **pixels** - Provides a two-dimensional grid representation of the NeoPixels on the NeoTrellis M4.
- **pixels.fill** - Colors all the NeoPixels a given color.

- **pixels.brightness** - The overall brightness of the pixels. Must be a number between **0** and **1**, which represents a percentage, i.e. **0.3** is 30%.
- **pixels.width** - The width of the NeoPixel grid. When rotation is **0** or **180**, the width is 8. When rotation is **90** or **270**, the width is 4.
- **pixels.height** - The height of the NeoPixel grid. When rotation is **0** or **180**, the height is 4. When rotation is **90** or **270**, the height is 8.

Let's take a look at a simple example. In the example below, click the **Download Project Bundle** button below to download the necessary libraries and the **code.py** file in a zip file. Extract the contents of the zip file, open the directory **examples/** and then click on the directory that matches the version of CircuitPython you're using and copy the contents of that directory to your **CIRCUITPY** drive.

Your **CIRCUITPY** drive should now look similar to the following image:



```
# SPDX-FileCopyrightText: 2021 ladyada for Adafruit Industries
# SPDX-License-Identifier: MIT

import adafruit_trellism4

trellis = adafruit_trellism4.TrellisM4Express()

while True:
    pressed = trellis.pressed_keys
    if pressed:
        print("Pressed:", pressed)
```

Try pressing a button. The coordinates are printed to the serial output.

```
Auto-reload is on. Simply save files over USB to run them or enter REPL to disable.
code.py output:
Pressed: [(0, 0)]
Pressed: [(0, 0)]
Pressed: [(0, 0)]
Pressed: [(0, 0)]
Pressed: [(0, 0)]
Pressed: [(0, 0)]
Pressed: [(0, 0)]
Pressed: [(0, 0)]
```

Try pressing more than one button. It prints the coordinates of all buttons pressed at any given point in time.

```
Pressed: [(0, 0), (0, 1)]
Pressed: [(0, 0), (0, 1)]
Pressed: [(0, 0), (0, 1)]
Pressed: [(0, 0), (0, 1)]
Pressed: [(0, 0), (0, 1)]
Pressed: [(0, 0), (0, 1)]
Pressed: [(0, 0), (0, 1), (0, 2)]
Pressed: [(0, 0), (0, 1), (0, 2)]
Pressed: [(0, 0), (0, 1), (0, 2)]
Pressed: [(0, 0), (0, 1), (0, 2)]
```

As you can see, `pressed_keys` returns a list (the outer brackets, i.e. `[]`) of coordinate tuples (the sets of numbers in parentheses, i.e. `(x, y)`).

This example is useful to test the buttons and give you an idea of what the coordinates are for each one.

Now we'll explore the other features of this library using the NeoPixels found on your NeoTrellis M4 Express.

## Now Let's Rotate the Board!

Copy the following code to the `code.py` file found on your CIRCUITPY drive and save it.

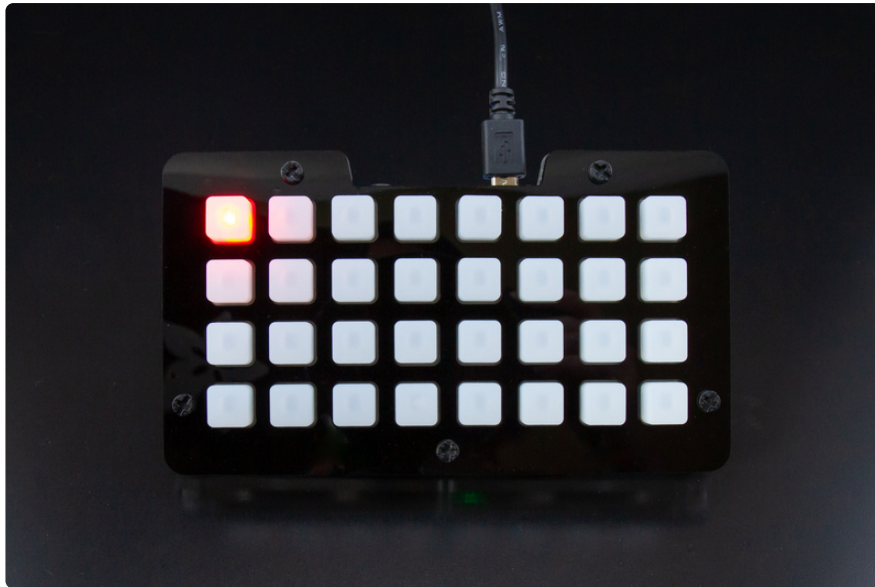
```
import adafruit_trellism4

trellis = adafruit_trellism4.TrellisM4Express()

while True:
    trellis.pixels[0, 0] = (255, 0, 0)
```

We have setup the library for use with: `trellis = adafruit_trellism4.TrellisM4Express()`

With the board oriented with the USB port on the opposite side of the board from you, the top-left button will light up red. This is the default orientation and is considered to be a 0 degree rotation.



The TrellisM4 library has the ability to specify four different orientations in 90 degree increments by setting `rotation` in the setup line equal to: **0**, **90**, **180**, or **270**. This allows you to rotate your board with ease to the position that works best for your project, without requiring complicated code to compensate. We do that for you! The coordinates of the buttons change with the rotation, so `(0, 0)` is always top-left. Let's look at some examples.

Rotate your board 90 degrees clockwise so the USB port is to the right, and change the setup line to update your code to the following:

```
import adafruit_trellism4

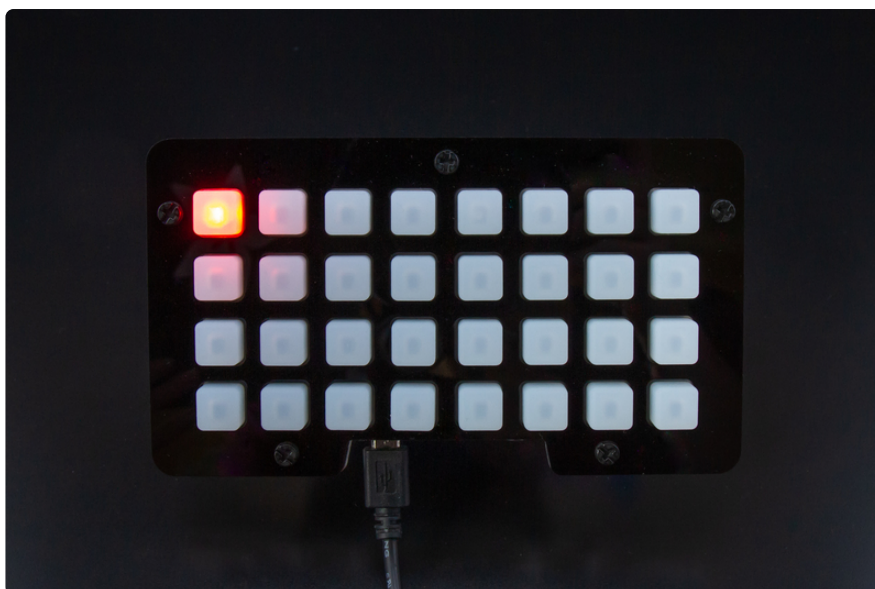
trellis = adafruit_trellism4.TrellisM4Express(rotation=90)

while True:
    trellis.pixels[0, 0] = (255, 0, 0)
```

Here we have told the code `rotation=90`. Again, the top-left button is red, without needing to change the rest of your code.



Try rotating your board another 90 degrees clockwise, until the USB cord is pointing down or towards you, and updating your code so `rotation=180` . Same result.



And another 90 degrees clockwise so the USB cord is pointing to the left, and update your code so `rotation=270` . Upper left is red!





These examples show that `(0, 0)` will always be in the upper left corner if you orient the board to match the `rotation` you provide. However, we've only specified pixel `(0, 0)` in these examples, so the code continues to work at all rotations. What happens if you have all the pixel coordinates specified in your code and provide a different rotation?

## Trellis M4 Coordinate Layout

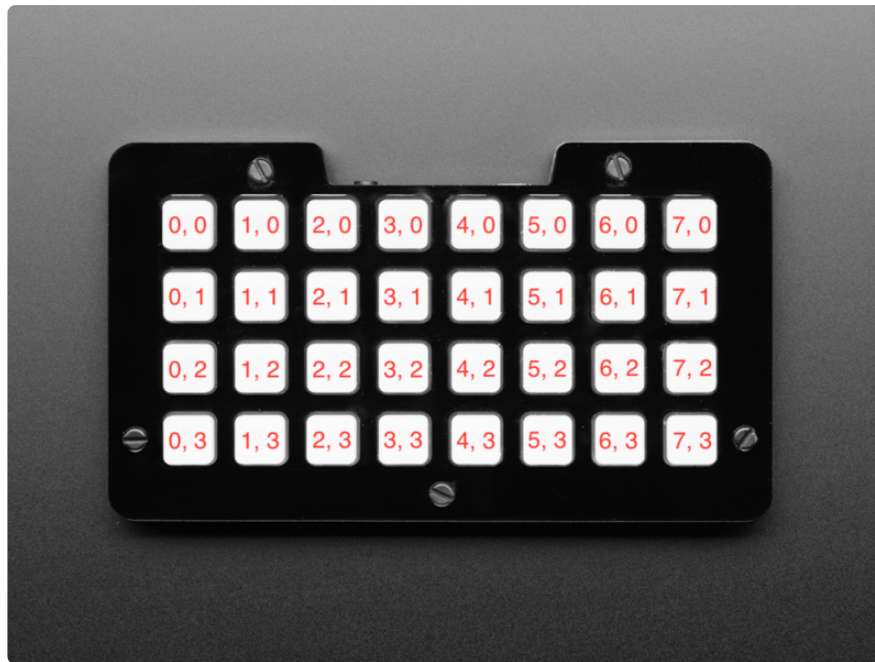
The rest of the available coordinates also update to match the `rotation` you specify on setup.

Therefore, if your code uses specific coordinates for either the buttons or the NeoPixels, and you provide a new `rotation` and rotate your board, you may need to change your code. This is because the available coordinates are different for different rotations.

For example, if you light up the NeoPixel at `(7, 3)` and then set `rotation=90`, your code will fail because that coordinate is not available at a 90 degree rotation. The images below show you what the coordinate grids look like at each possible rotation. Let's take a look.

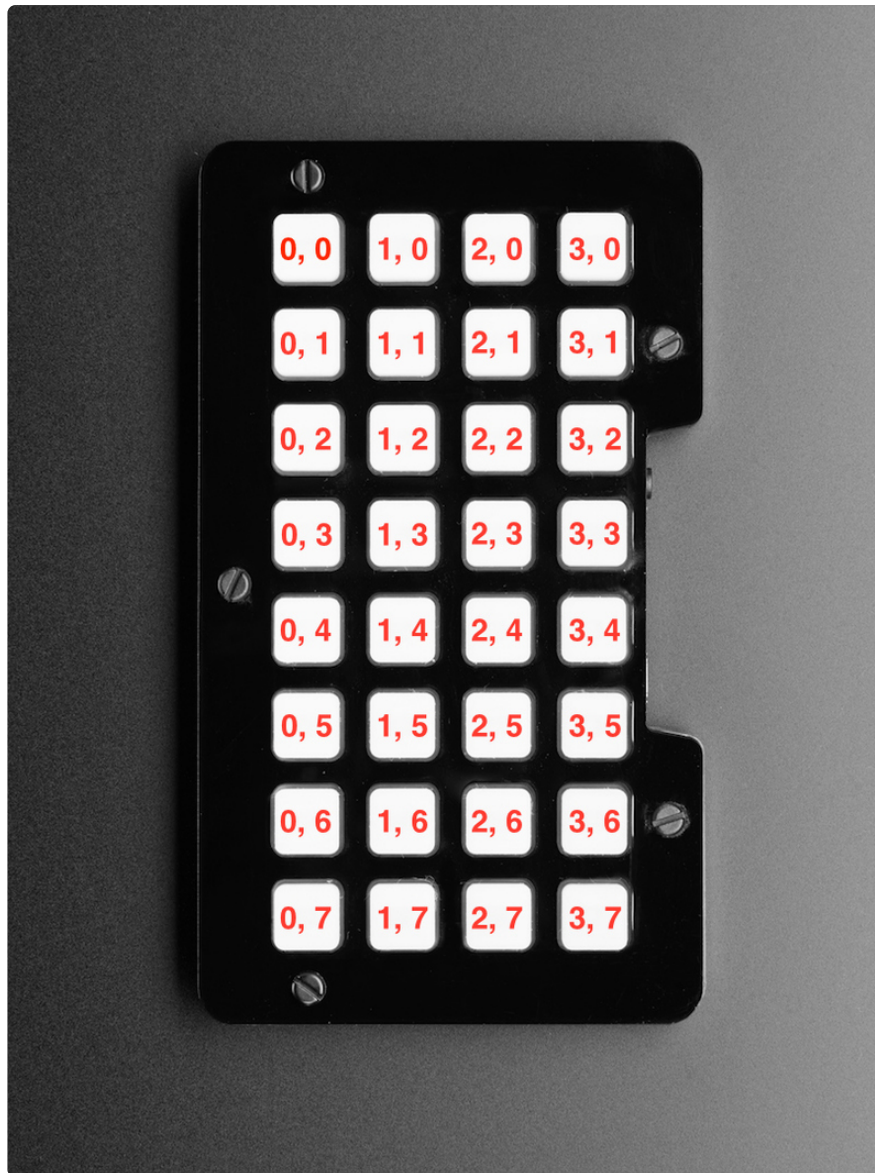
Here are the coordinates available when the board is at the `0` or `180` degree rotations:





$(0, 0)$  is always in the upper left corner and the rest of the grid follows standard positive  $(x, y)$  coordinates. So when the board is rotated  $0$  or  $180$  degrees, the board is oriented horizontally, and the lower right will be  $(7, 3)$ .

However, when you rotate the board to  $90$  or  $270$  degrees, the available coordinates change, as the grid is now oriented vertically:



The top left is still `(0, 0)`, but the bottom right is now `(3, 7)`.

So, remember, if you have specific coordinates for either the NeoPixels or the buttons in your code that are not available in the opposite orientation, and you provide a `rotation` for an opposite orientation, your code will error and stop running. This is important to keep in mind as a troubleshooting step if your code works in one orientation but fails if you rotate it 90 degrees in either direction.

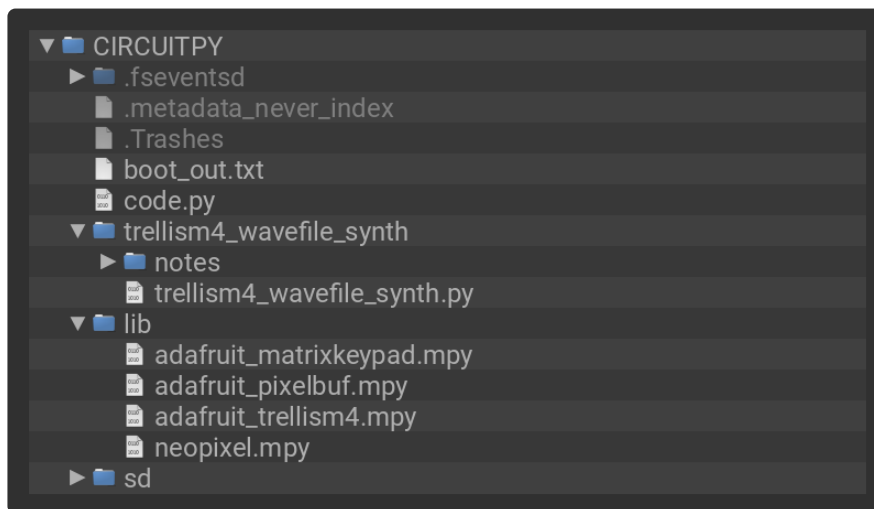
## Width and Height Can Make Rainbows!

This library provides a two-dimensional representation of the NeoPixel grid so you can use coordinates to address the different pixels. If you want to be able to iterate over the grid, you need to use the `pixels.width` and `pixels.height` properties.

The following example uses `pixels.width` and `pixels.height` to spread a rainbow over all the NeoPixels.

In the example below, click the **Download Project Bundle** button below to download the necessary libraries and the `code.py` file in a zip file. Extract the contents of the zip file, open the directory `examples/` and then click on the directory that matches the version of CircuitPython you're using and copy the contents of that directory to your **CIRCUITPY** drive.

Your **CIRCUITPY** drive should now look similar to the following image:



```
# SPDX-FileCopyrightText: 2021 ladyada for Adafruit Industries
# SPDX-License-Identifier: MIT

"""Test your Trellis M4 Express without needing the serial output.
Press any button and the rest will light up the same color!"""
import time
from rainbowio import colorwheel
import adafruit_trellism4

trellis = adafruit_trellism4.TrellisM4Express()

for x in range(trellis.pixels.width):
    for y in range(trellis.pixels.height):
        pixel_index = ((y * 8) + x) * 256 // 32
        trellis.pixels[x, y] = colorwheel(pixel_index & 255)

current_press = set()
while True:
    pressed = set(trellis.pressed_keys)
    for press in pressed - current_press:
        if press:
            print("Pressed:", press)
            pixel = (press[1] * 8) + press[0]
            pixel_index = pixel * 256 // 32
            trellis.pixels.fill(colorwheel(pixel_index & 255))
    for release in current_press - pressed:
        if release:
            print("Released:", release)
            for x in range(trellis.pixels.width):
                for y in range(trellis.pixels.height):
```

```
        pixel_index = ((y * 8) + x) * 256 // 32
        trellis.pixels[x, y] = colorwheel(pixel_index & 255)
    time.sleep(0.08)
    current_press = pressed
```

Rainbow! When you press a button, it lights up all the pixels to match the color of the button pressed and then returns to the rainbow layout when the button is released. It's an easy way to test whether your buttons are working without being connected to the serial console.

## More To Come!

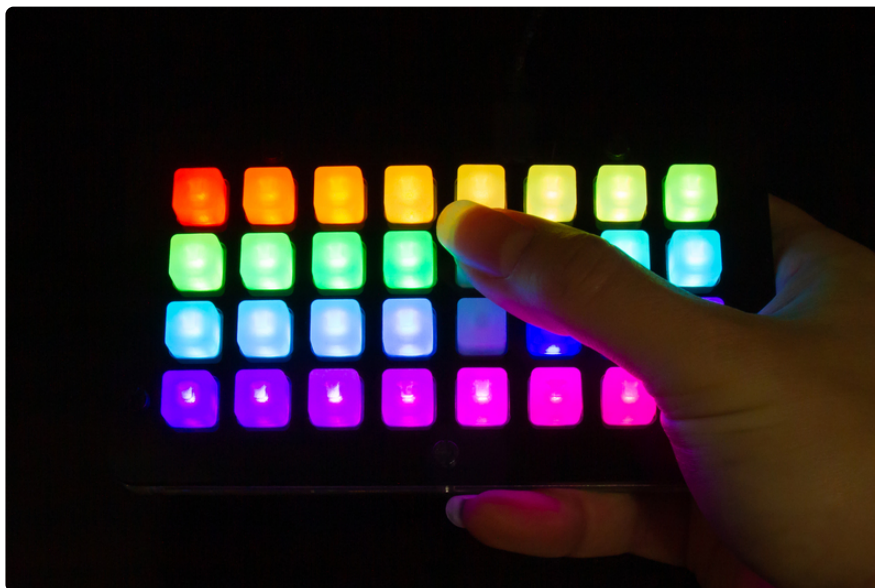
We will be adding more features to this library. Check back for updates and make sure you're always using the latest version of the library!

---

## Trellis M4 CircuitPython Demo

We've explained how to use the features in the Adafruit CircuitPython TrellisM4 library. Now we'll show how to use the library to write a CircuitPython program. In this section, you'll learn how to initialise the library for use in your code, and then use Python lists and sets to keep track of LED states and button presses.

This program begins with all the NeoPixels turned off. You can then press the buttons to toggle the associated NeoPixels on and off in a rainbow pattern spread over all of the LEDs.



## Imports and Setup

The first thing you need to do is import the library and then set it up for use. This is super simple. You begin your program with the following code:

```
import adafruit_trellism4
trellis = adafruit_trellism4.TrellisM4Express()
```

This is how you'll begin any program that will be using the TrellisM4 library. You can include other libraries in your code by adding them after the `import` line and setting them up after the `trellis` line.

## `wheel` for the Rainbow

Next, a helper function called `wheel` is defined. This code allows us to generate the rainbow pattern seen when you toggle all of the buttons on.

```
def wheel(pos):
    if pos < 0 or pos > 255:
        return 0, 0, 0
    if pos < 85:
        return int(255 - pos * 3), int(pos * 3), 0
    if pos < 170:
        pos -= 85
        return 0, int(255 - pos * 3), int(pos * 3)
    pos -= 170
    return int(pos * 3), 0, int(255 - (pos * 3))
```

For more information on how `wheel` is used, check out [the Wheel Explained section here \(https://adafru.it/Bek\)](https://adafru.it/Bek).

## A Set of Button Presses

With basic code, when you press a button, the button press gets sent over and over as long as you hold the button. This happens very quickly, too quickly for you to easily press it once and have the button press sent only once. Since we're trying to use the button as a toggle, we need it to respond only once until it is pressed again. So, we're going to use sets to keep track of which buttons have been pressed.

A set is an unordered collection with no duplicate elements. Since we're not keeping track of the order in which we press the buttons, we don't care that it's unordered. Since a set eliminates duplicates, we don't have to worry about it including the same button press twice. You can use math on sets to do things including subtracting one set from another. Subtracting one set from another provides you with the elements in

one set that are not in another. This allows you to determine the difference between the two sets. We will be comparing two sets. So sets are perfect for our needs!

This code creates two sets and then compares them. This allows for us to keep track of the buttons currently being pressed and compare them to the buttons previously pressed to determine what buttons have been released.

First, we create an empty `set` called `current_press`. Then we begin our loop and create a `set` called `pressed` that contains all the buttons being pressed (`trellis.pressed_keys`) at the beginning of the loop. If no buttons are pressed, `pressed` will be empty.

```
current_press = set()

while True:
    pressed = set(trellis.pressed_keys)
    for press in pressed - current_press:
        ... # LED toggle code lives here.
    current_press = pressed
```

At the end of the loop, we set `current_press = pressed` so `current_press` contains all of the buttons currently pressed when the loop ends. Then, the loop begins again, and we check to see which buttons are being pressed by checking `pressed`. This means we now have two sets: one containing the buttons pressed when the loop ended, and one containing the buttons pressed when the loop begins.

Now, we do some math so we can compare the two. `press - current_press` will provide a collection of buttons still being pressed, and we can now apply the LED toggle code to only those buttons!

## A Multidimensional List of NeoPixel States

This example toggles the NeoPixels on or off when each button is pressed. Whether the pixel is turned on or off is dependent on the pixel's current state; if the pixel is on, the button press turns it off, and if the pixel is off, the button press turns it on. We need to track the current state of every pixel to be able to determine what state to toggle it to. For that, we're going to use a multidimensional list.

A list is a collection of items that are both ordered and changeable. We want to use the `(x, y)` coordinates of the pixels to keep track of which pixel is which, so we want to keep all of the list entries in a particular order. We are going to be tracking two states for each pixel, so we need to be able to change each entry in the list to reflect the current state. Lists are exactly what we need!

Since we are going to be using `(x, y)` coordinates of the pixels to determine which pixel we are addressing, we need to create a list that is a representation of the NeoPixel grid. For this, we need to create a multidimensional list, which is a list made up of lists.

First we create the initial list and call it `led_on`.

```
led_on = []
```

If you were to `print(led_on)`, it would look like this:

```
[]
```

Then we create the multidimensional aspect of the list. We use `append` to add `width` number of empty lists to the initial list, and then we populate them with `height` number of entries, which we're setting to `False` because the LEDs are not on to begin with. As we initialised the Trellis without specifying a rotation value, width is equal to 8 and height is equal to 4.

```
for x in range(trellis.pixels.width):  
    led_on.append([])  
    for y in range(trellis.pixels.height):  
        led_on[x].append(False)
```

Let's take a look at the two parts of this section of code.

We append `x`, or 8, empty lists to our initial list:

```
for x in range(trellis.pixels.width):  
    led_on.append([])
```

If you were to `print(led_on)` now, it would look like this:

```
[[], [], [], [], [], [], [], []]
```

Then we append four entries of `False` to each of the `x` empty lists:

```
for y in range(trellis.pixels.height):  
    led_on[x].append(False)
```

Finally, if you were to `print(led_on)` now, it would look like this:

```
[[False, False, False, False], [False, False, False, False], [False,  
False, False, False], [False, False, False, False], [False, False,
```



```
False, False], [False, False, False, False], [False, False, False, False], [False, False, False, False]] [False, False, False, False]]
```

Each four member list represents an `x` coordinate, 0 - 7. Each `False` represents a `y` coordinate, 0 - 3. For example, if `(3, 2)` were set to `True`, the list would look like this:

```
[[False, False, False, False], [False, False, False, False], [False, False, False, False], [False, False, True, False], [False, False, False, False], [False, False, False, False], [False, False, False, False], [False, False, False, False]] [False, False, False, False]]
```

Now we have our representation of the NeoPixel grid and we can start toggling!

## Extra Credit: List Comprehensions

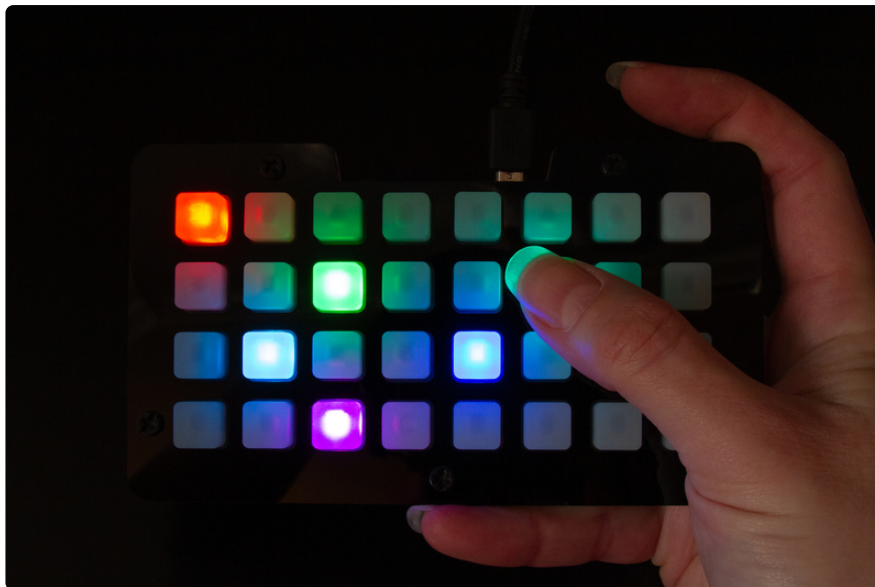
We used five lines of code to create our list. It was easy to see what was happening in each section of code, and to explain how it was affecting the creation of our list. It turns out, however, that there's a way to do it in one line of code!

List comprehensions provide a more concise way to create lists. They consist of brackets containing an expression followed by a `for` clause, potentially followed by further `for` or `if` clauses. As you can see, we used two `for` clauses in our list creation code. Our list comprehension consists of two `for` clauses as well.

You can replace every line of code discussed in the previous section with the following line:

```
led_on = [[False for y in range(trellis.pixels.height)] for x in range(trellis.pixels.width)]
```

It does exactly the same thing as the code in the previous section: creates a multidimensional list made up of `x` lists, each containing `y` entries of `False`. And, it's much shorter. Handy!



## NeoPixels On: True or False?

Now that we have our button presses tracked and our multidimensional list, we can easily toggle the NeoPixels. We're going to use the list to track whether they are on or off. Each time we press a button, we'll check to see whether it's associated list entry is `True` or `False`. If it's `False`, that means the LED is currently off. If it's `True`, it means the LED is currently on. Based on these results, that button press will change the list entry and turn the LED to the appropriate status. Let's take a look!

Here is our LED toggle code:

```
x, y = press
if not led_on[x][y]:
    print("Turning on:", press)
    pixel_index = ((x + (y * 8)) * 256 // 32)
    trellis.pixels[x, y] = wheel(pixel_index & 255)
    led_on[x][y] = True

else:
    print("Turning off:", press)
    trellis.pixels[x, y] = (0, 0, 0)
    led_on[x][y] = False
```

`press` is already an `x, y` coordinate, but we need to have access to `x` and `y` separately. So, we first unpack `press` with `x, y = press`.

We need access to `x` and `y` to use our list. Remember, we're using a multidimensional list containing `x` number of lists made up of `y` number of entries. So, to access them, we'll use `led_on[x][y]`.

Next, we have an `if/else` block. The `if not led_on[x][y]:` says if the `led_on` list entry found at `[x][y]` is `False`, run the code indented below it. The `else` says,

"otherwise," which is to say, if the `led_on` list entry found at `[x][y]` is `True`, run the code indented below it.

If the list entry is `False` when we press a button, we send a `print` statement, turn on the LED associated with that button to the rainbow color that matches its location in the rainbow spread over all the buttons. Then, we set the list entry to `True`.

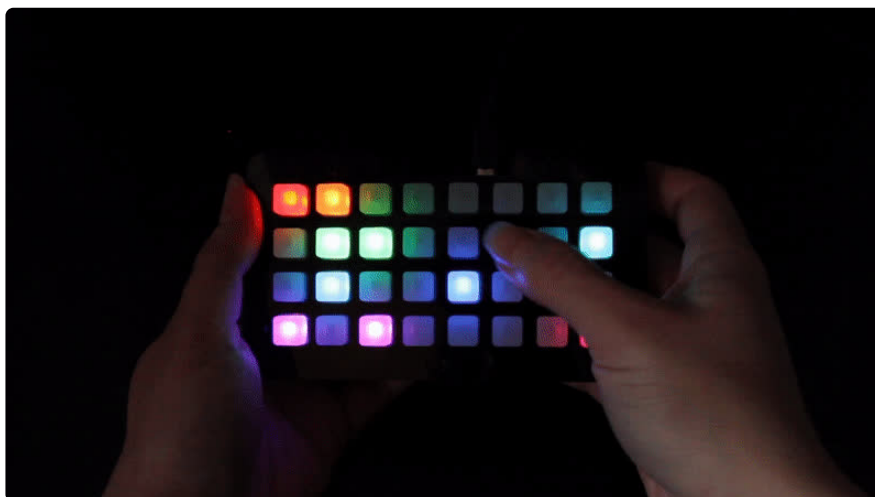
```
if not led_on[x][y]:
    print("Turning on:", press)
    pixel_index = ((x + (y * 8)) * 256 // 32)
    trellis.pixels[x, y] = wheel(pixel_index & 255)
    led_on[x][y] = True
```

If the list entry is `True` when we press a button, we send a `print` statement, turn the LED associated with that button off. Then, we set the list entry to `False`.

```
else:
    print("Turning off:", press)
    trellis.pixels[x, y] = (0, 0, 0)
    led_on[x][y] = False
```

You may have noticed that to address the pixels, we use `trellis.pixels[x, y]`, but to address the list we use `led_on[x][y]`. This is because the NeoPixel code is specially designed to work with an `[x, y]` coordinate assignment. The multidimensional list, however, only works by addressing `[x]` and `[y]` separately.

Whew! That's it! Now it's time to press, press, press!

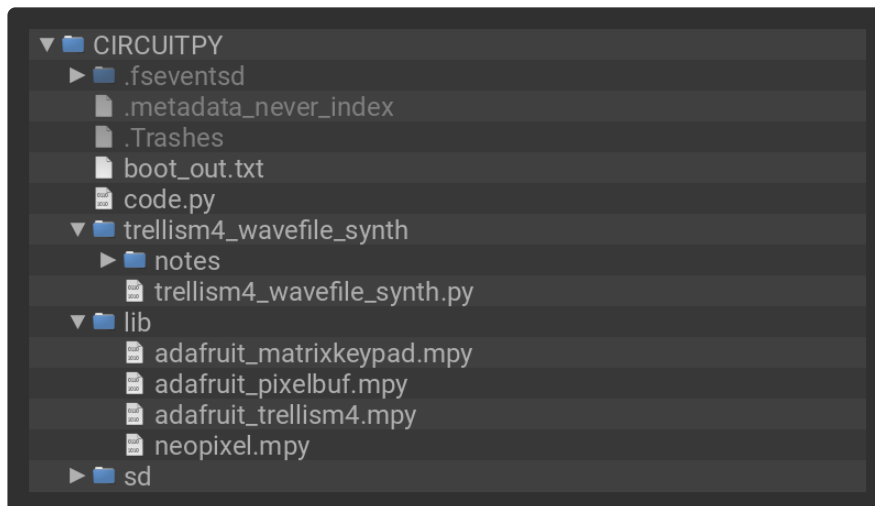


## NeoTrellis M4 NeoPixel Toggle Code

Here's the full program. To use with CircuitPython, you need to first install a few libraries, into the lib folder on your **CIRCUITPY** drive. Then you need to update **code.py** with the example script.

Thankfully, we can do this in one go. In the example below, click the **Download Project Bundle** button below to download the necessary libraries and the **code.py** file in a zip file. Extract the contents of the zip file, open the directory **examples/** and then click on the directory that matches the version of CircuitPython you're using and copy the contents of that directory to your **CIRCUITPY** drive.

Your **CIRCUITPY** drive should now look similar to the following image:



```
# SPDX-FileCopyrightText: 2021 ladyada for Adafruit Industries
# SPDX-License-Identifier: MIT

from rainbowio import colorwheel
import adafruit_trellism4

trellis = adafruit_trellism4.TrellisM4Express()

led_on = []

for x in range(trellis.pixels.width):
    led_on.append([])
    for y in range(trellis.pixels.height):
        led_on[x].append(False)

trellis.pixels.fill((0, 0, 0))

current_press = set()

while True:
    pressed = set(trellis.pressed_keys)

    for press in pressed - current_press:
        x, y = press

        if not led_on[x][y]:
            print("Turning on:", press)
            pixel_index = (x + (y * 8)) * 256 // 32
            trellis.pixels[x, y] = colorwheel(pixel_index & 255)
            led_on[x][y] = True

        else:
            print("Turning off:", press)
            trellis.pixels[x, y] = (0, 0, 0)
            led_on[x][y] = False
```

```
current_press = pressed
```

## Downloads

## Files

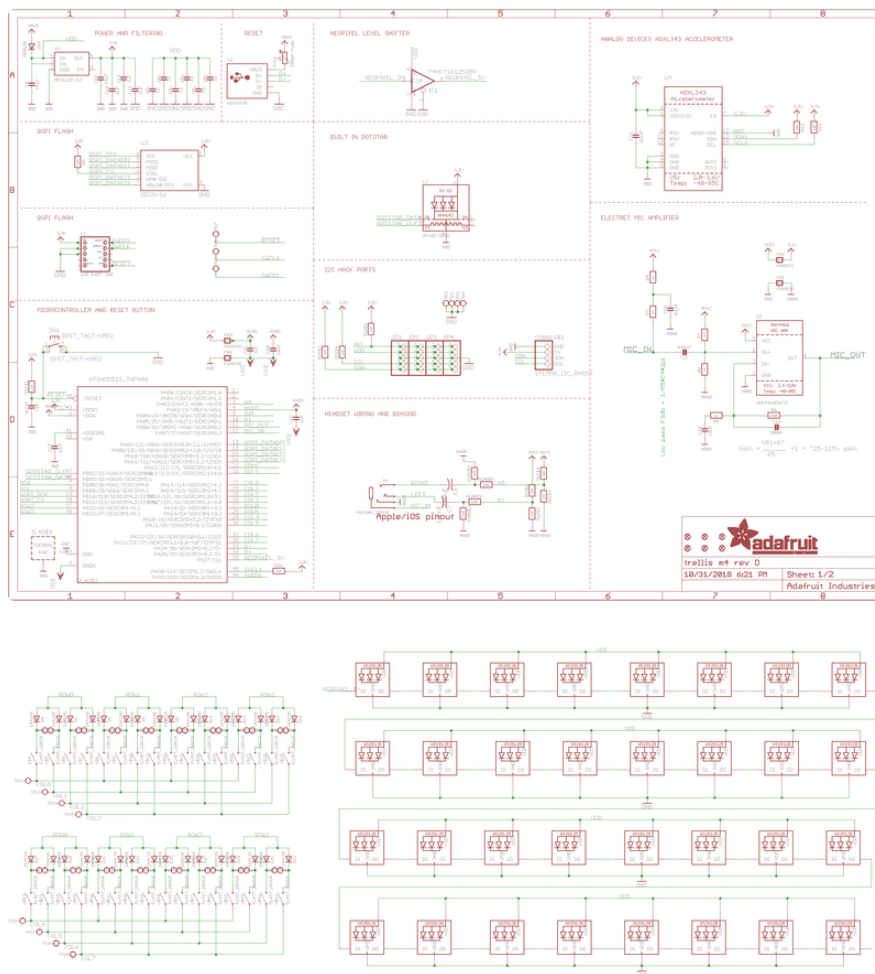
- [PCB and Enclosure Files \(https://adafru.it/D0i\)](https://adafru.it/D0i)
- [PDF for NeoTrellis Board Diagram on GitHub \(https://adafru.it/-At\)](https://adafru.it/-At)

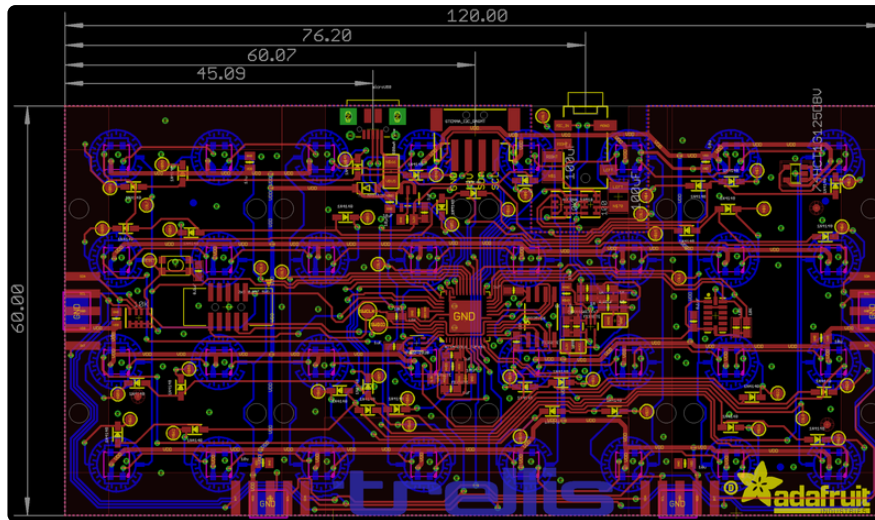
SVG for NeoTrellis Board Diagram

<https://adafru.it/-Au>

## Schematics and Fabrication Prints

Click to Embiggen





## UF2 Bootloader Details

This is an information page for advanced users who are curious how we get code from your computer into your Express board!

Adafruit SAMD21 (M0) and SAMD51 (M4) boards feature an improved bootloader that makes it easier than ever to flash different code onto the microcontroller. This bootloader makes it easy to switch between Microsoft MakeCode, CircuitPython and Arduino.

Instead of needing drivers or a separate program for flashing (say, `bossac`, `jlink` or `avrdude`), one can simply **drag a file onto a removable drive**.

The format of the file is a little special. Due to 'operating system woes' you cannot just drag a binary or hex file (trust us, we tried it, it isn't cross-platform compatible). Instead, the format of the file has extra information to help the bootloader know where the data goes. The format is called UF2 (USB Flashing Format). Microsoft MakeCode generates UF2s for flashing and CircuitPython releases are also available as UF2. [You can also create your own UF2s from binary files using uf2conv.py, available here. \(https://adafru.it/vPE\)](https://adafru.it/vPE)

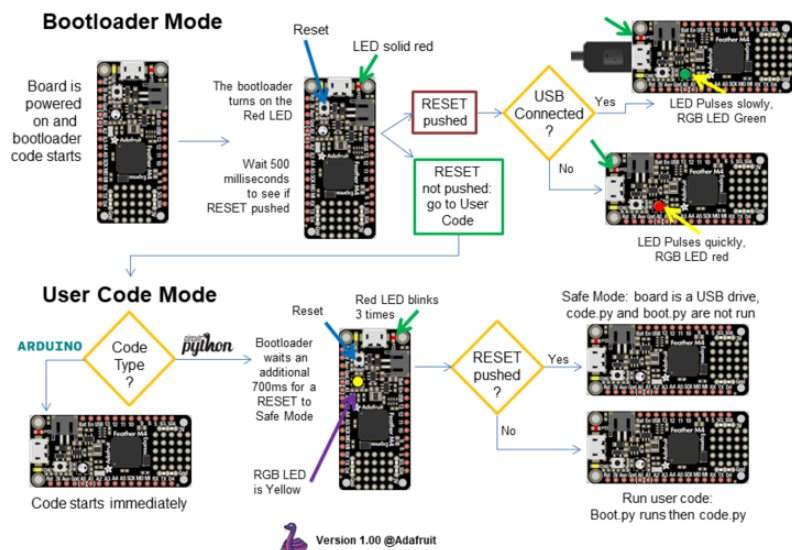
The bootloader is **also BOSSA compatible**, so it can be used with the Arduino IDE which expects a BOSSA bootloader on ATSAMD-based boards

For more information about UF2, [you can read a bunch more at the MakeCode blog \(https://adafru.it/w5A\)](https://adafru.it/w5A), then [check out the UF2 file format specification. \(https://adafru.it/vPE\)](https://adafru.it/vPE)

Visit [Adafruit's fork of the Microsoft UF2-samd bootloader GitHub repository](https://adafruit.it/Beu) (<https://adafruit.it/Beu>) for source code and releases of pre-built bootloaders on [CircuitPython.org](https://adafruit.it/Em8) (<https://adafruit.it/Em8>).

The bootloader is not needed when changing your CircuitPython code. Its only needed when upgrading the CircuitPython core or changing between CircuitPython, Arduino and Microsoft MakeCode.

### The CircuitPython Boot Sequence



## Entering Bootloader Mode

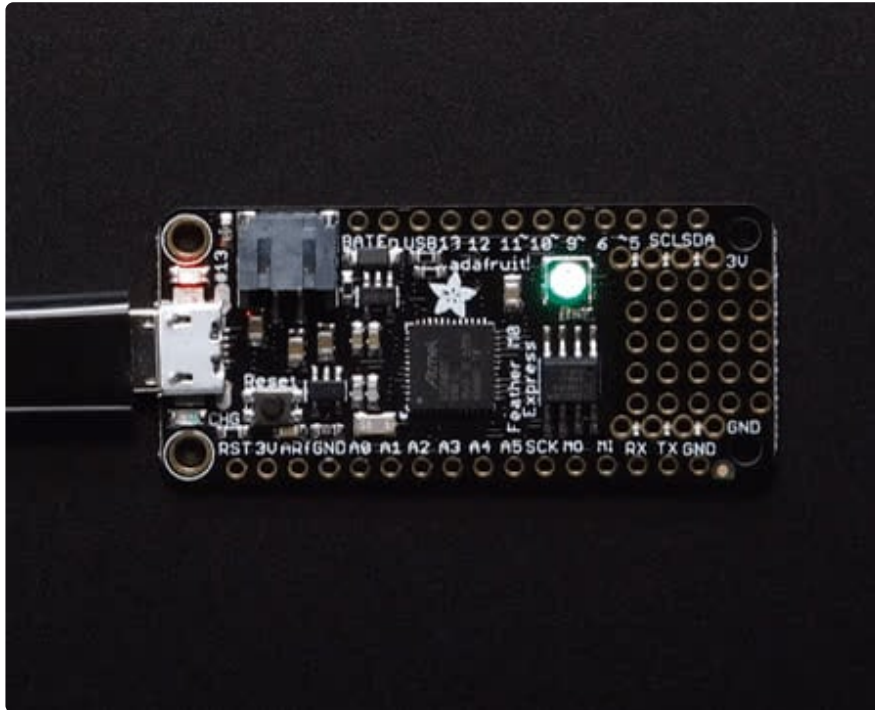
The first step to loading new code onto your board is triggering the bootloader. It is easily done by double tapping the reset button. Once the bootloader is active you will see the small red LED fade in and out and a new drive will appear on your computer with a name ending in **BOOT**. For example, feathers show up as **FEATHERBOOT**, while the new CircuitPlayground shows up as **CPLAYBOOT**, Trinket M0 will show up as **TRINKETBOOT**, and Gemma M0 will show up as **GEMMABOOT**

Furthermore, when the bootloader is active, it will change the color of one or more onboard neopixels to indicate the connection status, red for disconnected and green for connected. If the board is plugged in but still showing that its disconnected, try a different USB cable. Some cables only provide power with no communication.

For example, here is a Feather M0 Express running a colorful Neopixel swirl. When the reset button is double clicked (about half second between each click) the

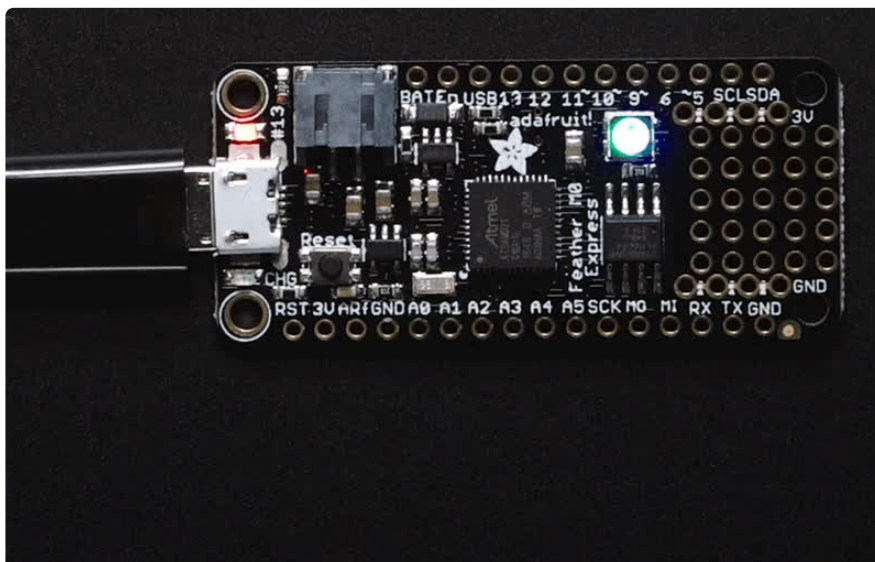


NeoPixel will stay green to let you know the bootloader is active. When the reset button is clicked once, the 'user program' (NeoPixel color swirl) restarts.

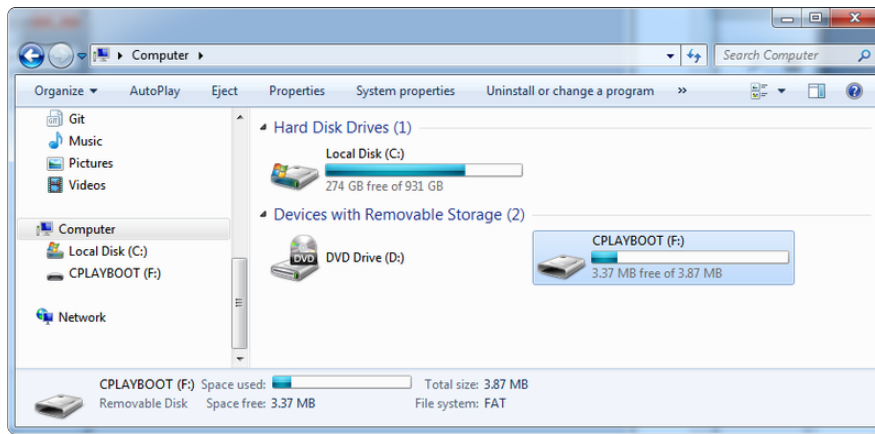


If the bootloader couldn't start, you will get a red NeoPixel LED.

That could mean that your USB cable is no good, it isn't connected to a computer, or maybe the drivers could not enumerate. Try a new USB cable first. Then try another port on your computer!

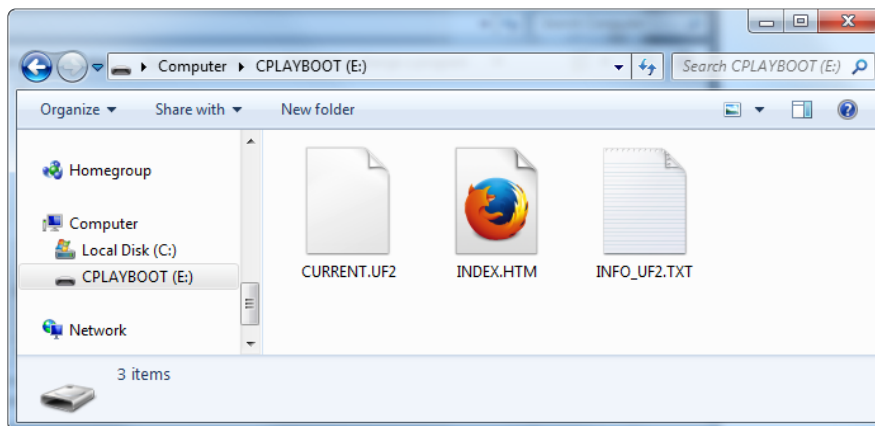


Once the bootloader is running, check your computer. You should see a USB Disk drive...



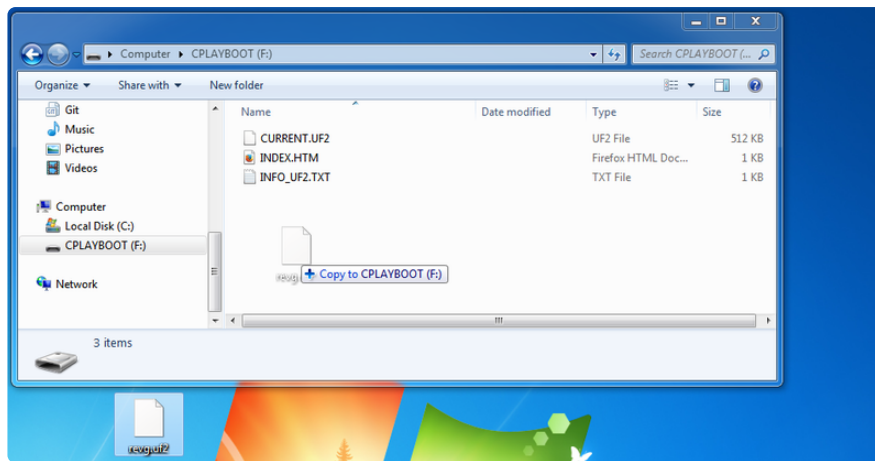
Once the bootloader is successfully connected you can open the drive and browse the virtual filesystem. This isn't the same filesystem as you use with CircuitPython or Arduino. It should have three files:

- **CURRENT.UF2** - The current contents of the microcontroller flash.
- **INDEX.HTM** - Links to Microsoft MakeCode.
- **INFO\_UF2.TXT** - Includes bootloader version info. Please include it on bug reports.

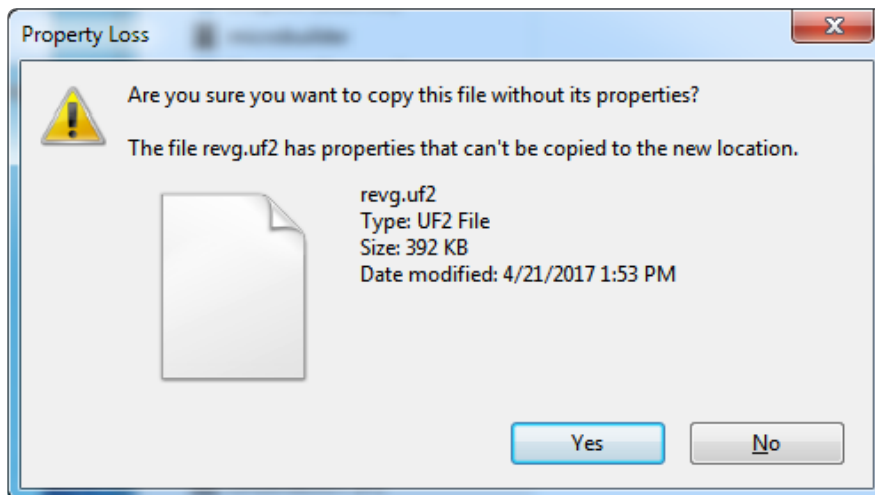


## Using the Mass Storage Bootloader

To flash something new, simply drag any UF2 onto the drive. After the file is finished copying, the bootloader will automatically restart. This usually causes a warning about an unsafe eject of the drive. However, its not a problem. The bootloader knows when everything is copied successfully.



You may get an alert from the OS that the file is being copied without its properties. You can just click **Yes**



You may also get a complaint that the drive was ejected without warning. Don't worry about this. The drive only ejects once the bootloader has verified and completed the process of writing the new code

## Using the BOSSA Bootloader

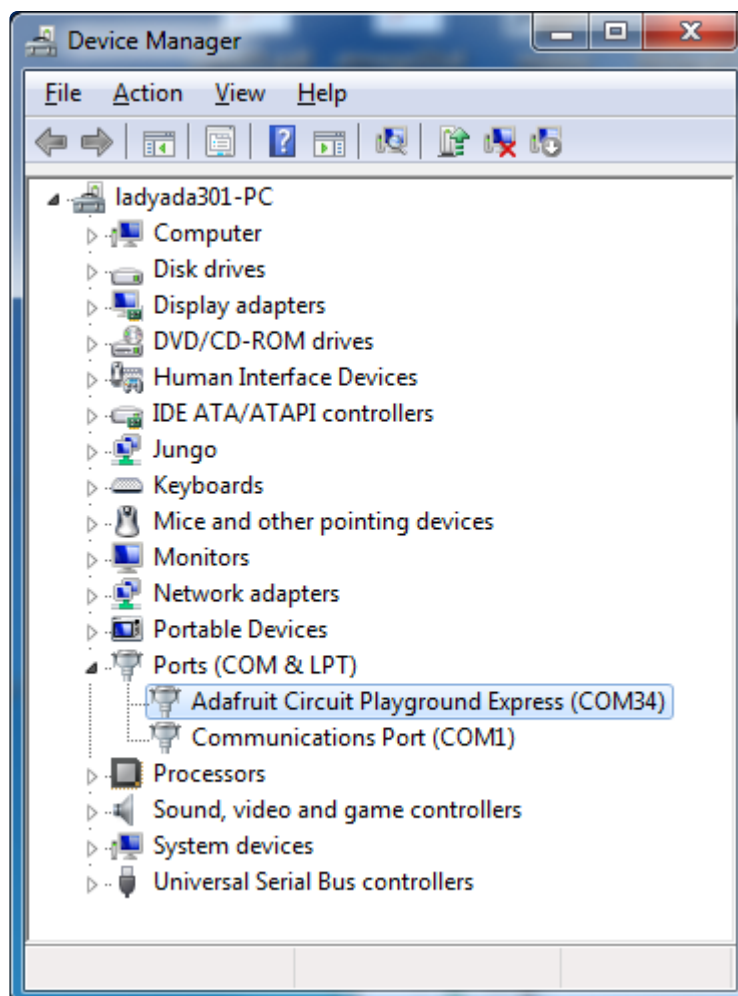
As mentioned before, the bootloader is also compatible with BOSSA, which is the standard method of updating boards when in the Arduino IDE. It is a command-line tool that can be used in any operating system. We won't cover the full use of the **bossac** tool, suffice to say it can do quite a bit! More information is available at [ShumaTech \(https://adafru.it/vQa\)](https://adafru.it/vQa).

If you are using Windows 7 or Windows 8.1 , note they have reached end-of-life and are no longer supported. They required driver installation. A limited set of

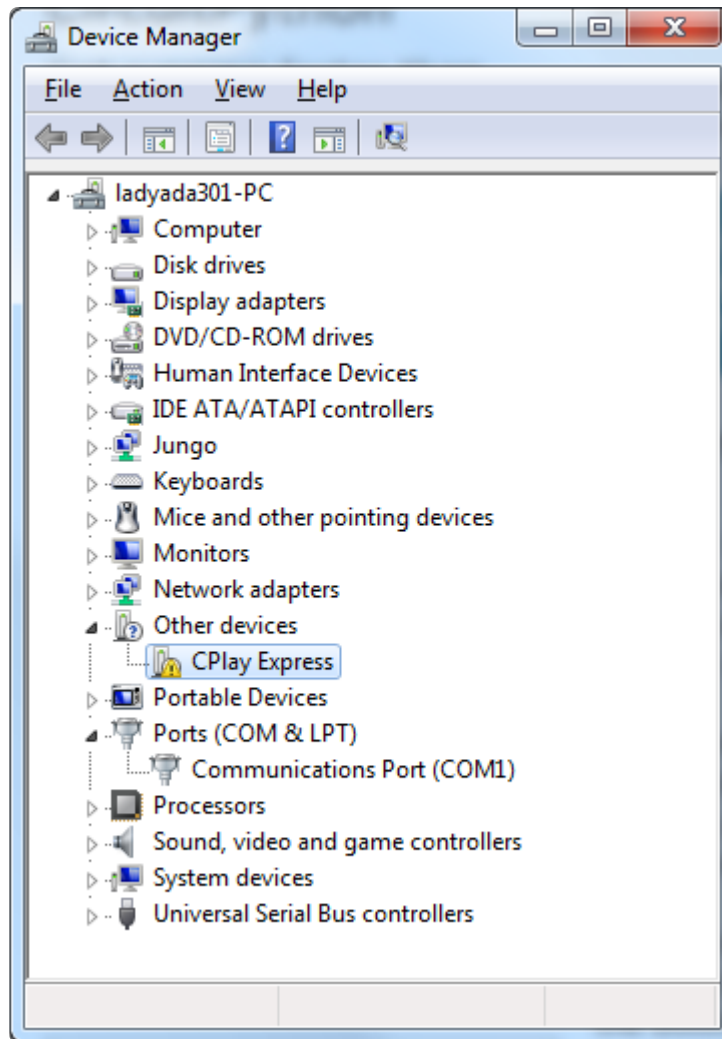
drivers is available for older boards, but drivers for most newer boards are not available.

## Verifying Serial Port in Device Manager

If you're running Windows, its a good idea to verify the device showed up. Open your Device Manager from the control panel and look under **Ports (COM & LPT)** for a device called **Feather M0** or **Circuit Playground** or whatever!



If you see something like this, it means you did not install the drivers. Go back and try again, then remove and re-plug the USB cable for your board



## Running bossac on the command line

If you are using the Arduino IDE, this step is not required. But sometimes you want to read/write custom binary files, say for loading CircuitPython or your own code. We recommend using bossac v 1.7.0 (or greater), which has been tested. [The Arduino branch is most recommended \(https://adafru.it/vQb\)](https://adafru.it/vQb).

[You can download the latest builds here. \(https://adafru.it/s1B\)](https://adafru.it/s1B) The `mingw32` version is for Windows, `apple-darwin` for Mac OSX and various `linux` options for Linux. Once downloaded, extract the files from the zip and open the command line to the directory with `bossac`.

With bossac versions 1.9 or later, you **must** use the `--offset` parameter on the command line, and it must have the correct value for your board.

With bossac version 1.9 or later, you must give an `--offset` parameter on the command line to specify where to start writing the firmware in flash memory. This parameter was added in bossac 1.8.0 with a default of `0x2000`, but starting in 1.9, the default offset was changed to `0x0000`, which is not what you want in most cases. If you omit the argument for bossac 1.9 or later, you will probably see a "Verify Failed" error from bossac. Remember to change the option for `-p` or `--port` to match the port on your Mac.

Replace the filename below with the name of your downloaded `.bin`: it will vary based on your board!

### Using bossac Versions 1.7.0, 1.8

There is no `--offset` parameter available. Use a command line like this:

```
bossac -p=/dev/cu.usbmodem14301 -e -w -v -R adafruit-circuitpython-boardname-version.bin
```

For example,

```
bossac -p=/dev/cu.usbmodem14301 -e -w -v -R adafruit-circuitpython-feather_m0_express-3.0.0.bin
```

### Using bossac Versions 1.9 or Later

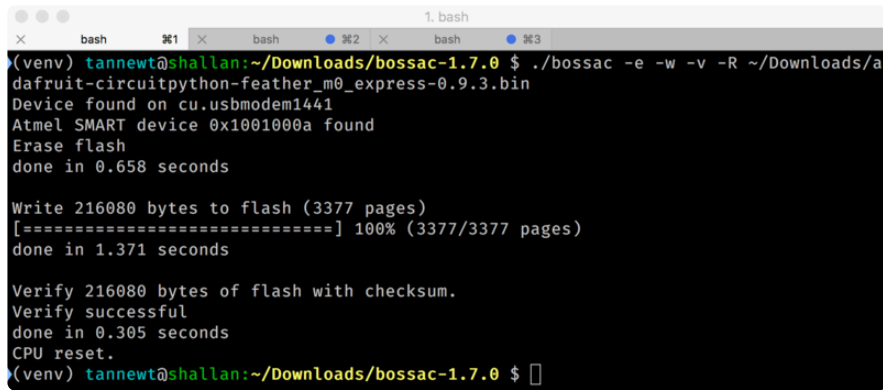
For M0 boards, which have an 8kB bootloader, you must specify `-offset=0x2000`, for example:

```
bossac -p=/dev/cu.usbmodem14301 -e -w -v -R --offset=0x2000 adafruit-circuitpython-feather_m0_express-3.0.0.bin
```

For M4 boards, which have a 16kB bootloader, you must specify `-offset=0x4000`, for example:

```
bossac -p=/dev/cu.usbmodem14301 -e -w -v -R --offset=0x4000 adafruit-circuitpython-feather_m4_express-3.0.0.bin
```

This will **e**rase the chip, **w**rite the given file, **v**erify the write and **R**eset the board. On Linux or MacOS you may need to run this command with `sudo ./bossac ...`, or add yourself to the **dialout** group first.



```
(venv) tannewt@shallan:~/Downloads/bossac-1.7.0 $ ./bossac -e -w -v -R ~/Downloads/a
dafruit-circuitpython-feather_m0_express-0.9.3.bin
Device found on cu.usbmodem1441
Atmel SMART device 0x1001000a found
Erase flash
done in 0.658 seconds

Write 216080 bytes to flash (3377 pages)
[=====] 100% (3377/3377 pages)
done in 1.371 seconds

Verify 216080 bytes of flash with checksum.
Verify successful
done in 0.305 seconds
CPU reset.
(venv) tannewt@shallan:~/Downloads/bossac-1.7.0 $
```

## Updating the bootloader

The UF2 bootloader is relatively new and while we've done a ton of testing, it may contain bugs. Usually these bugs effect reliability rather than fully preventing the bootloader from working. If the bootloader is flaky then you can try updating the bootloader itself to potentially improve reliability.

If you're using MakeCode on a Mac, you need to make sure to upload the bootloader to avoid a serious problem with newer versions of MacOS. See instructions and more details [here \(https://adafru.it/ECU\)](https://adafru.it/ECU).

In general, you shouldn't have to update the bootloader! If you do think you're having bootloader related issues, please post in the forums or discord.

Updating the bootloader is as easy as flashing CircuitPython, Arduino or MakeCode. Simply enter the bootloader as above and then drag the update bootloader uf2 file below. This uf2 contains a program which will unlock the bootloader section, update the bootloader, and re-lock it. It will overwrite your existing code such as CircuitPython or Arduino so make sure everything is backed up!

After the file is copied over, the bootloader will be updated and appear again. The **INFO\_UF2.TXT** file should show the newer version number inside.

For example:

```
UF2 Bootloader v2.0.0-adafruit.5 SFHWR0
Model: Metro M0
Board-ID: SAMD21G18A-Metro-v0
```

Lastly, reload your code from Arduino or MakeCode or flash the [latest CircuitPython core \(https://adafru.it/Em8\)](https://adafru.it/Em8).



Below are the latest updaters for various boards. The latest versions can always be found [here](https://adafru.it/Bmg) (<https://adafru.it/Bmg>). Look for the `update-bootloader...` files, not the `bootloader...` files.

**Circuit Playground Express V3.7.0  
update-bootloader.uf2**

<https://adafru.it/JcN>

**Feather M0 Express v3.7.0 update-  
bootloader.uf2**

<https://adafru.it/JcO>

**Metro M0 Express v3.7.0 update-  
bootloader.uf2**

<https://adafru.it/JcR>

**Gemma M0 v3.7.0 update-  
bootloader.uf2**

<https://adafru.it/JcU>

**Trinket M0 v3.7.0 update-  
bootloader.uf2**

<https://adafru.it/JcX>

**Itsy Bitsy M0 v3.7.0 update-  
bootloader.uf2**

<https://adafru.it/Jc->

**Grand Central M4 v3.7.0 update-  
bootloader.uf2**

<https://adafru.it/Jd2>

**Latest version of update-  
bootloader.uf2 for other boards.  
Make sure you pick the right one.**

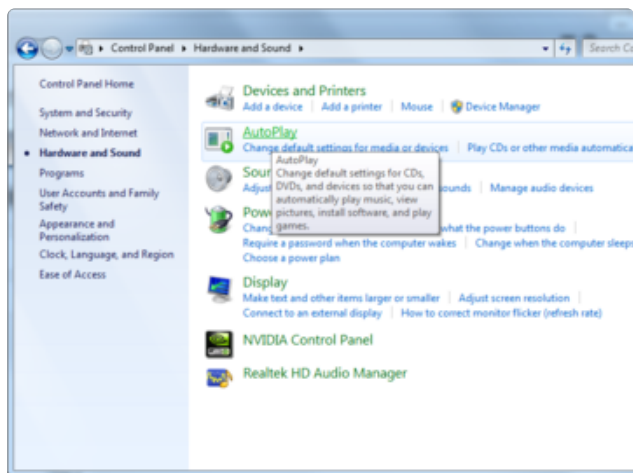
<https://adafru.it/Bmg>

# Getting Rid of Windows Pop-ups

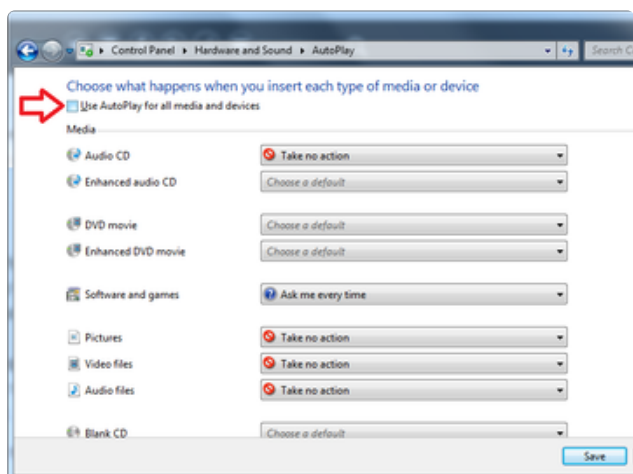
If you do a lot of development on Windows with the UF2 bootloader, you may get annoyed by the constant "Hey you inserted a drive what do you want to do" pop-ups.



Go to the Control Panel. Click on the **Hardware and Sound** header



Click on the **AutoPlay** header



Uncheck the box at the top, labeled **Use AutoPlay for all devices**

# Making your own UF2

Making your own UF2 is easy! All you need is a .bin file of a program you wish to flash and [the Python conversion script \(https://adafru.it/vZb\)](https://adafru.it/vZb). Make sure that your program was compiled to start at 0x2000 (8k) for M0 boards or 0x4000 (16kB) for M4 boards. The bootloader takes up the first 8kB (M0) or 16kB (M4). CircuitPython's [linker script \(https://adafru.it/CXh\)](https://adafru.it/CXh) is an example on how to do that.

Once you have a .bin file, you simply need to run the Python conversion script over it. Here is an example from the directory with **uf2conv.py**. This command will produce a **firmware.uf2** file in the same directory as the source **firmware.bin**. The uf2 can then be flashed in the same way as above.

```
# For programs with 0x2000 offset (default)
uf2conv.py -c -o build-circuitplayground_express/firmware.uf2 build-
circuitplayground_express/firmware.bin

# For programs needing 0x4000 offset (M4 boards)
uf2conv.py -c -b 0x4000 -o build-metro_m4_express/firmware.uf2 build-
metro_M4_express/firmware.bin
```

## Installing the bootloader on a fresh/bricked board

If you somehow damaged your bootloader or maybe you have a new board, you can use a JLink to re-install it.

[Here's a Learn Guide explaining how to fix the bootloader on a variety of boards using Atmel Studio \(https://adafru.it/F5f\)](https://adafru.it/F5f)

[Here's a short writeup by turbinenreiter on how to do it for the Feather M4 \(but adaptable to other boards\) \(https://adafru.it/ven\)](https://adafru.it/ven)

---

## FAQ/Troubleshooting

---

While playing audio back from the NeoTrellis M4 over the TRRS jack, I get an odd 'hissing' sound along with whatever is being played. Anything I should do to address this?

If your NeoTrellis M4 is powered by your computer USB, first thing to try is decoupling any ground loops by using headphones or a powered speaker running from a battery pack.

If it persists, there's a chance you're hearing serial data being transferred over the USB port, which can be noisy. This is the trade off of going with plain DAC output as opposed to I2S. It is meant to be a simple audio playback toy, not audiophile quality :)

You may also want to try powering the NeoTrellis M4 over a USB battery pack instead of your computer.

---

## When using the 16-step sequencer, if you press the audio sampler buttons, odd music will play

The 16-step sequencer 'sampler' demo where you can record audio onto the board depends on audio being stored in the flash memory, which is **also** where circuitpython stores files, so if you play one of the 'prerecorded' audio clips it will play files on the file system!

You can solve this by loading CircuitPython back onto the board, then removing all the WAV files.

---

## Why does the NeoTrellis sometimes 'hiccup' and restart after a few seconds?

Some computers, especially Mac's, will write a small amount of data after a few seconds, this causes CircuitPython to (correctly) reboot, and you'll get the audio re-start after a few seconds.

---

After loading the 16-step sequencer, the MIDI arpeggiator or some other UF2 examples, my CIRCUITPY no longer appears!

Correct, these are **Arduino** programs. Once you're done with these examples, [re-visit the CircuitPython for NeoTrellis M4 installation page and re-install CircuitPython \(<https://adafru.it/C-O>\)](#)

---

A few seconds/minutes after playing a sound on the NeoTrellis there's a 'peeeeeeeeeeeewwwwwwww' sound - what's that?

You're hearing the output capacitors discharge after not playing audio for a while. This normal, not harmful, and occurs with some headphones/speakers

---

When I press a button on the NeoTrellis a whole row is activated?

It's possible to accidentally "press" a column of four buttons even when you intend to only press one. This can happen when you press a button at an angle other than straight down on top of the button pad. The elastomer conductive part touches the NeoPixel ground pin against the column which makes the chip think the whole row/column is pressed. It only happens on some buttons and some angles, and is not common

It's not harmful, and there's a few ways to avoid it:

- Press straight down on buttons not at an angle
  - Cut a thicker top panel from wood/plastic/3D printing so that the buttons cannot be pressed at an angle
  - Read buttons via the [https://github.com/adafruit/Adafruit\\_NeoTrellisM4](https://github.com/adafruit/Adafruit_NeoTrellisM4) (<https://adafru.it/CW5>) library which will do a software filtering to remove the duplicate button presses
- 

Why do sometimes all the buttons get dimmer and the Trellis stops working?

If many of the NeoPixels are at close to full brightness, you will draw too much power from the USB port on your computer. Try lowering the brightness. If you are using the `adafruit_trellism4` library, you can use the `.brightness` setter to lower the brightness (1.0 is max; try 0.2 or 0.3).