Adafruit NeoTrellis M4 Express
Created by lady ada

Last updated on 2019-05-06 03:35:26 AM UTC

# 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)... 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).
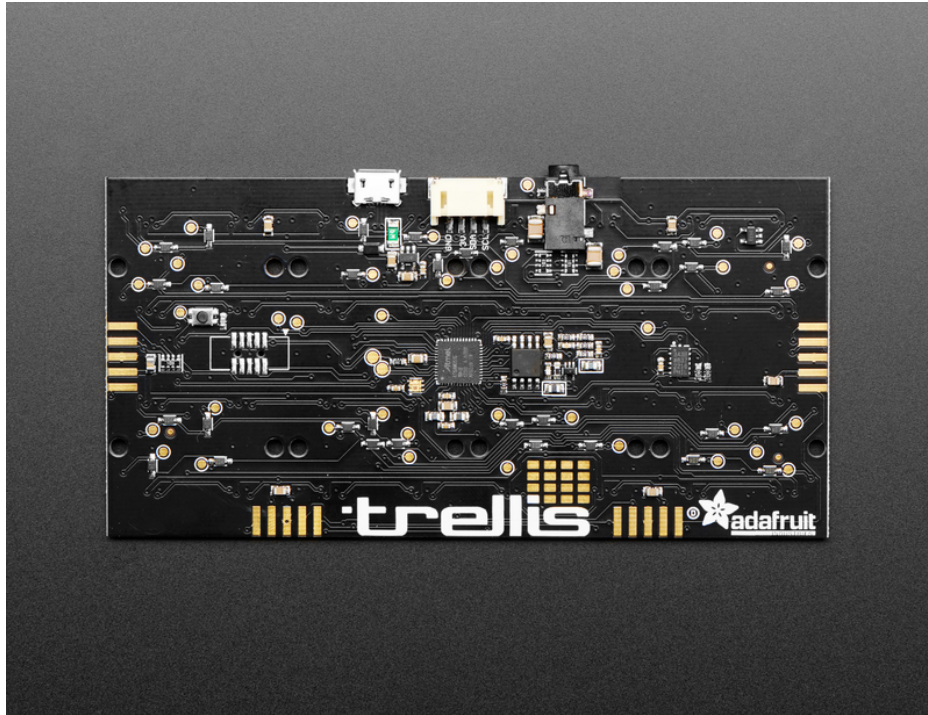

The native USB port can turn it into a MIDI USB device if you like - currently that's only supported in Arduino. Tether it to a computer or tablet, if you like. Or use it in standalone mode, as long as its powered from a USB power plug, it'll run whatever firmware is burned into it.

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.
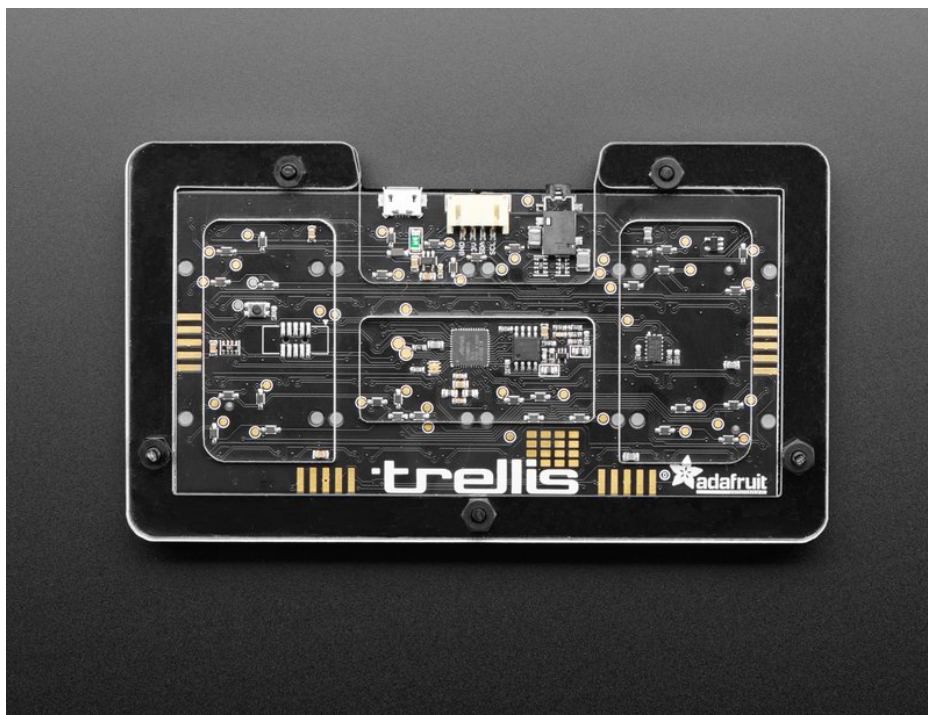

And if you want to have some audio inputs, we have you covered. The 4th pin on the headphone jack is for microphone or line in. If you have a classic 'mobile phone headset', the electret mic will go through an amplifier into an ADC pin. Again, it's not audiophile quality (we're talking about an electret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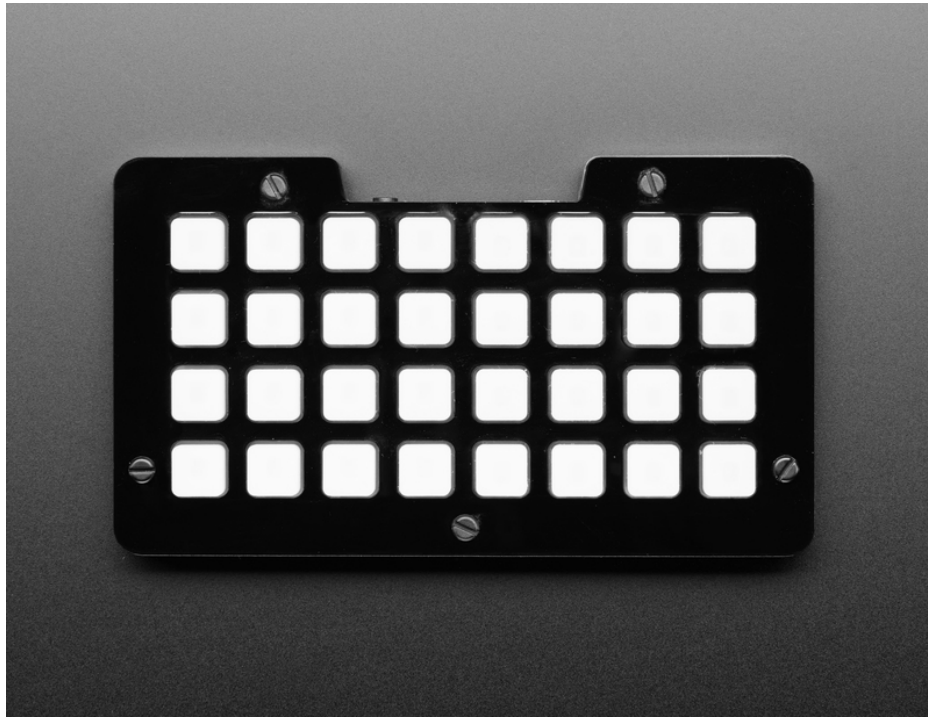

To add more interactivity, a precision triple-axis accelerometer from Analog Devices, the ADXL343, is included as well, and provides sensor information on tilt, motion, or tapping. Great for adding another dimension of data input in addition to the button pads.

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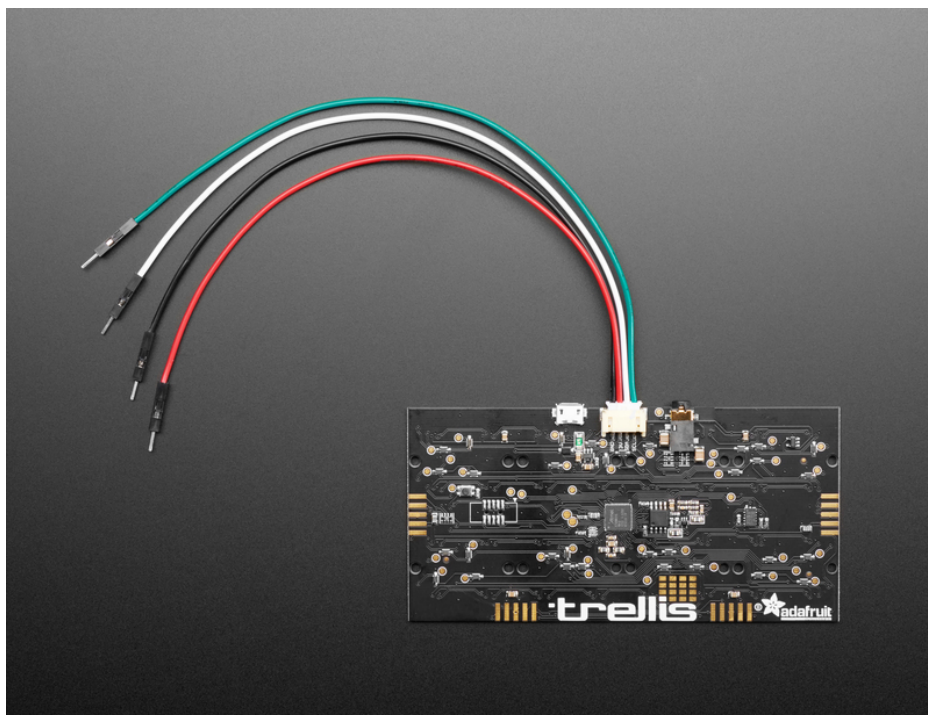
Here are some of the features you can look forward to when using NeoTrellis M4

- **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 (https://adafru.it/eQw) 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)
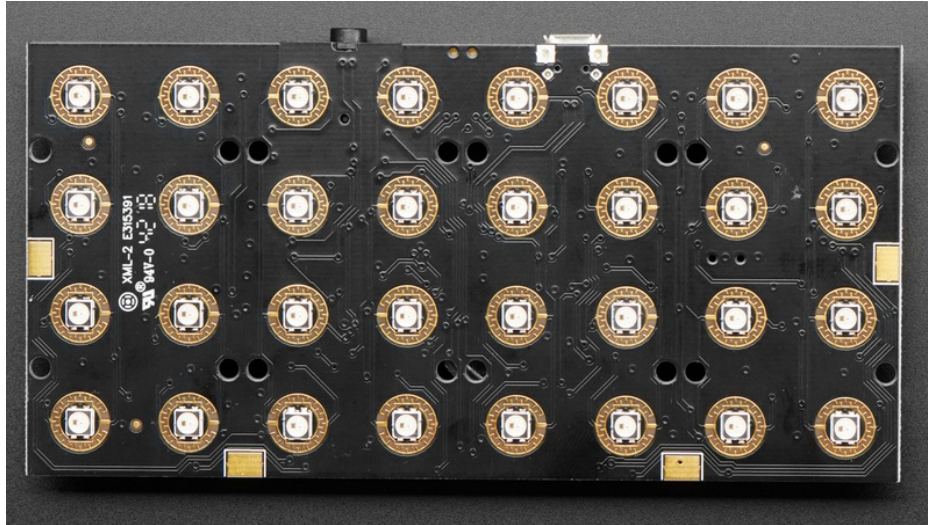- Really fun to press buttons and have sounds come out!
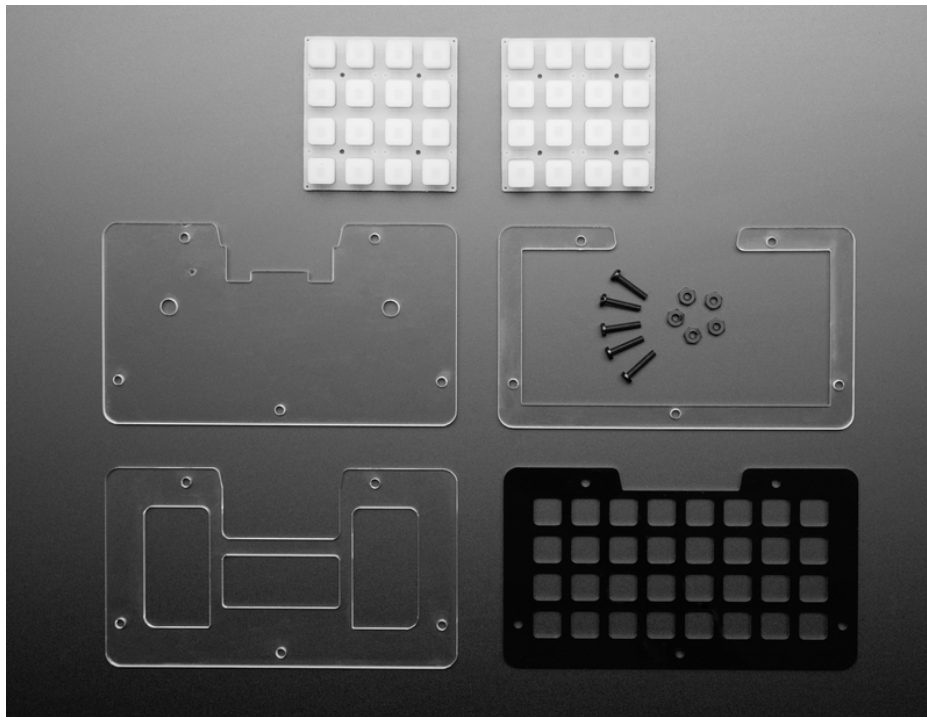
Board Tour

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

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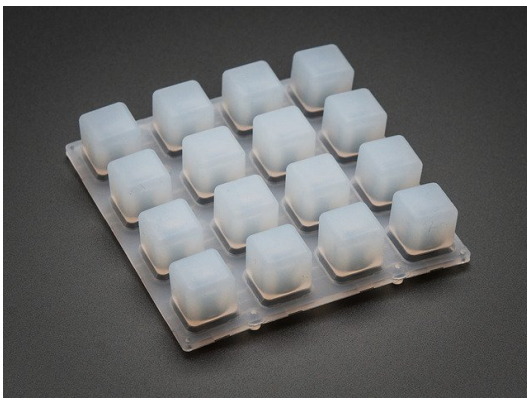
Silicone Elastomer 4x4 Button Keypad - for 3mm LEDs

$4.95
IN STOCK

ADD TO CART

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
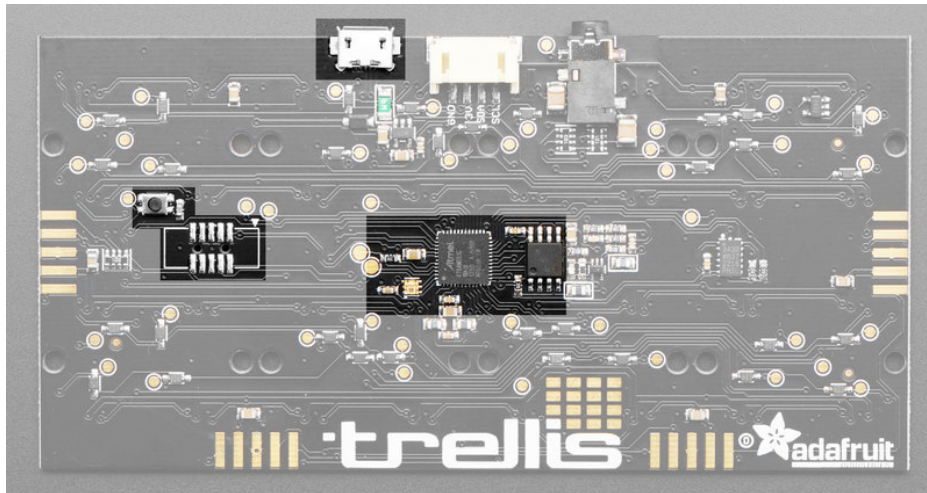
`board.COL0` , `board.COL1` , `board.COL2` , `board.COL3` , `board.COL4` , `board.COL5` , `board.COL6` , `board.COL7`

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
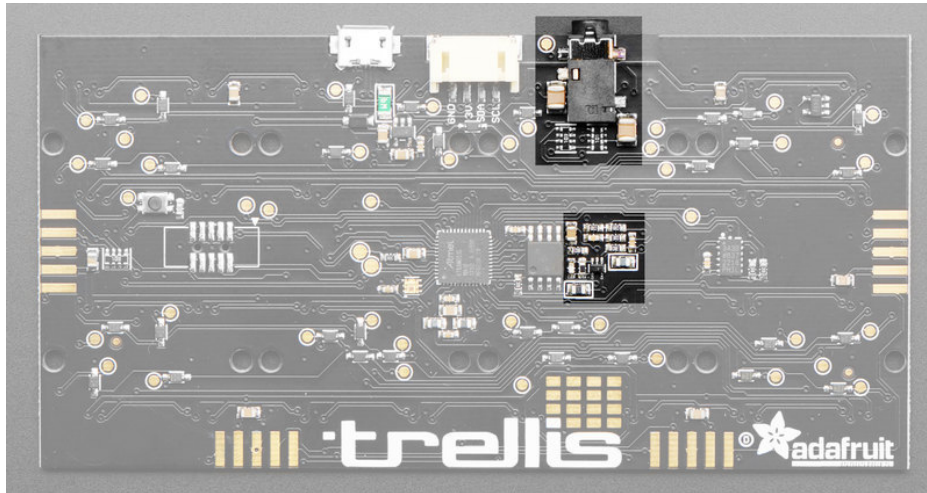
Stereo Audio Out

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.
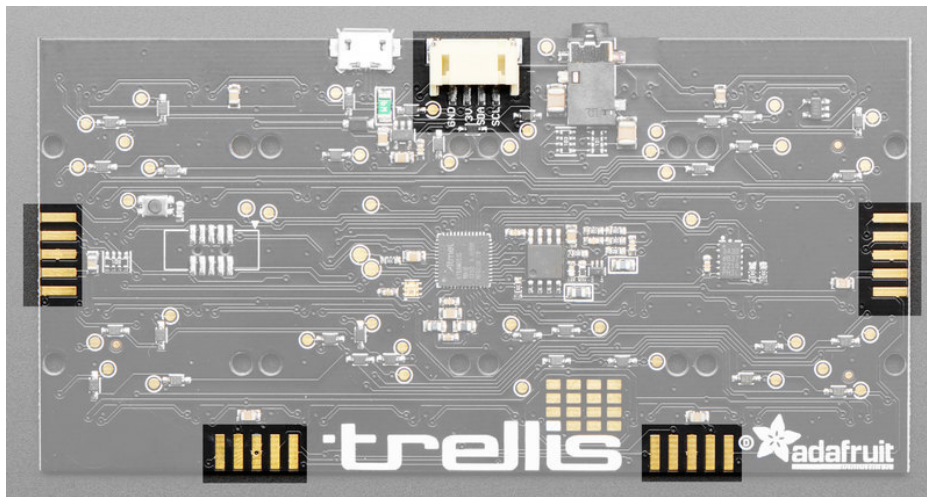
If you're using Arduino, the two GPIO are available on **PIN_WIRE_SDA** and **PIN_WIRE_SCL** for I2C usage, or **A4** and **A5** for analog reading.

On CircuitPython, you can use `board.SDA` and `board.SCL`

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.
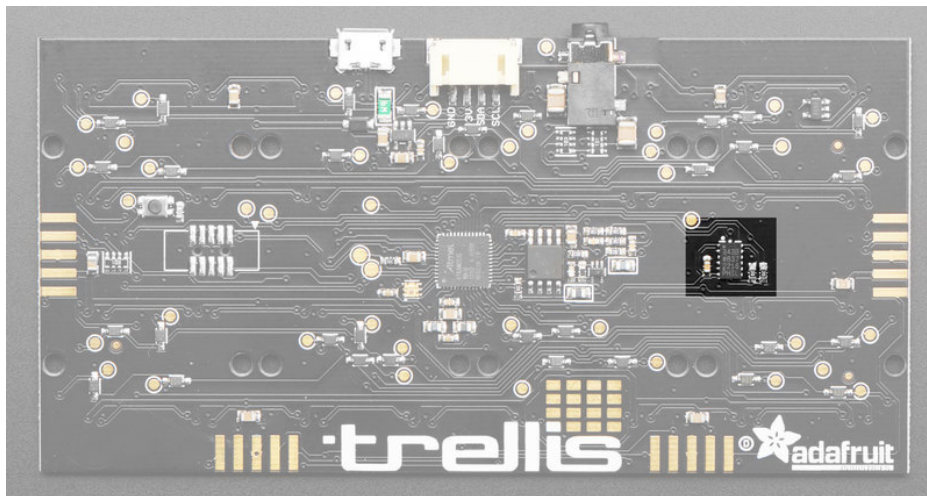

This sensor is on a separate I2C port, which is available in Arduino under**Wire1** or in CircuitPython under `board.ACCELEROMETER_SCL` and `board.ACCELEROMETER_SDA`

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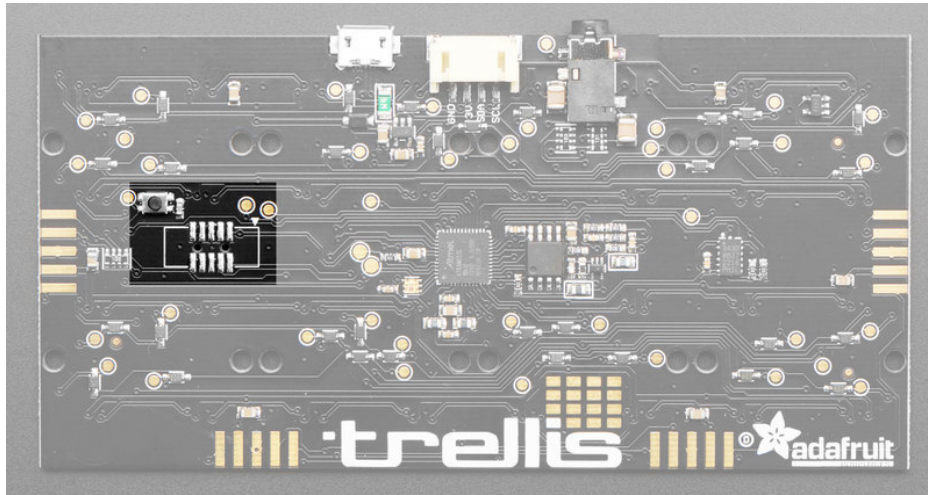
There's a small reset button on the back of the PCB, you can double-click it to enter the bootloader.

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
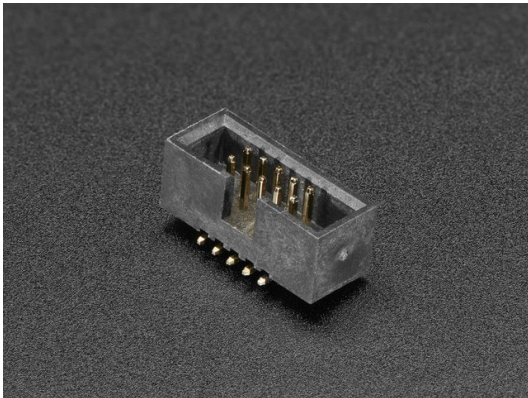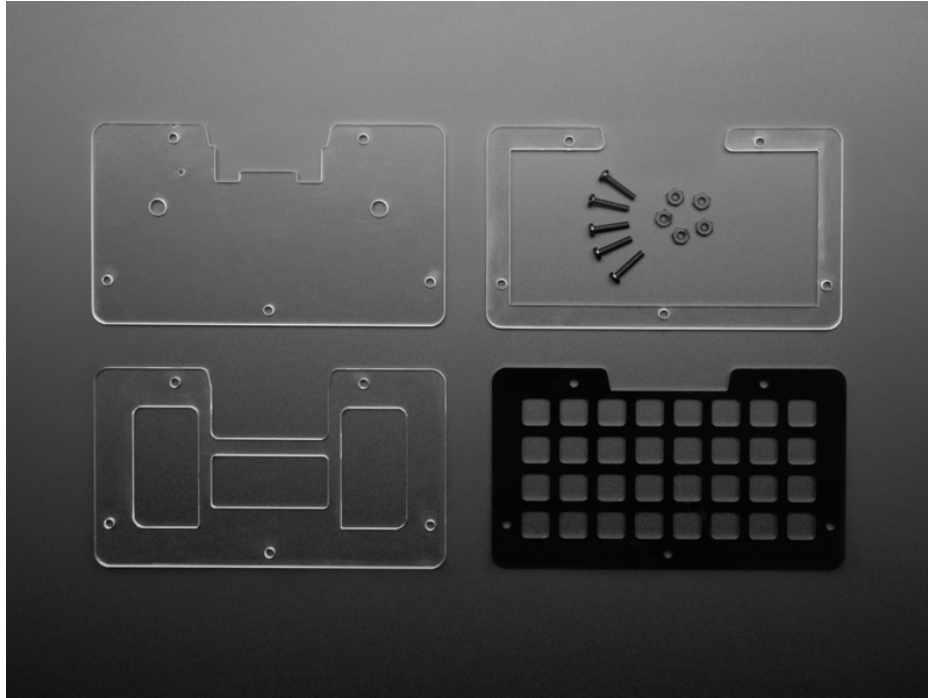

SWD 0.05" Pitch Connector - 10 Pin SMT Box Header
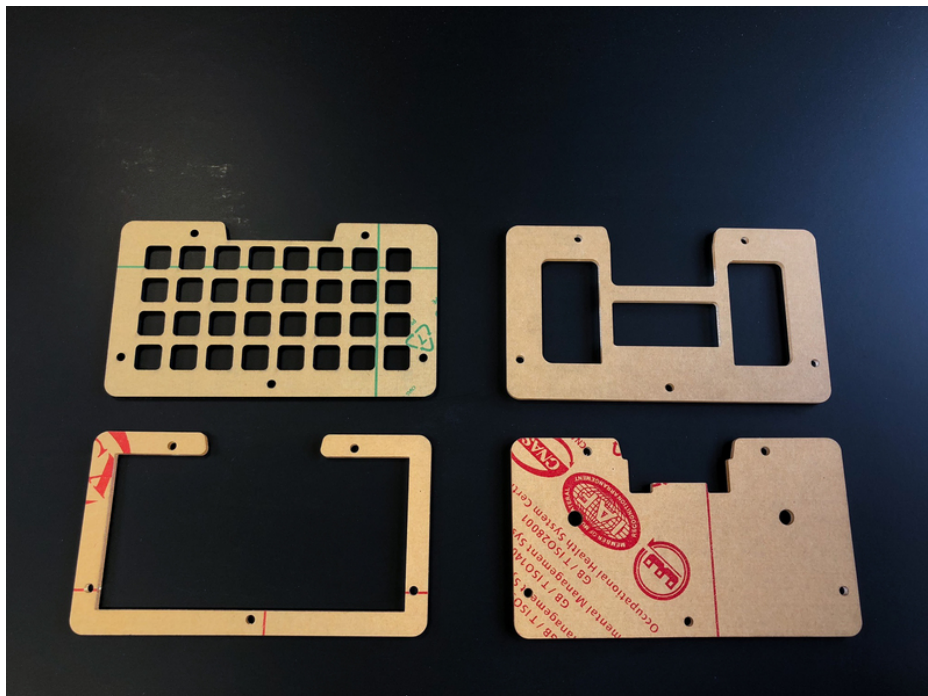
$1.50
IN STOCK

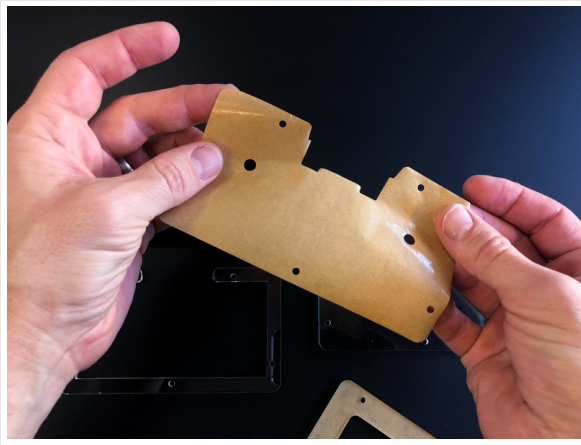ADD TO CART

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.
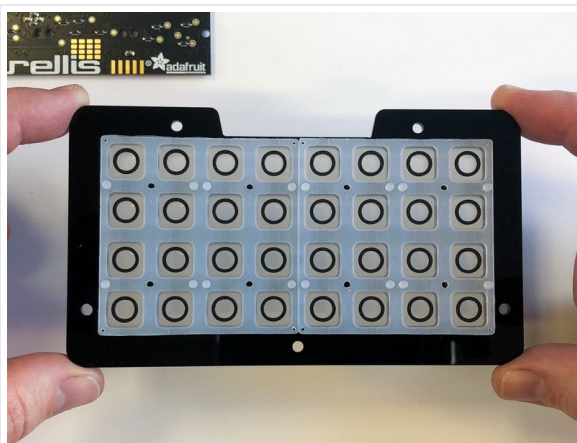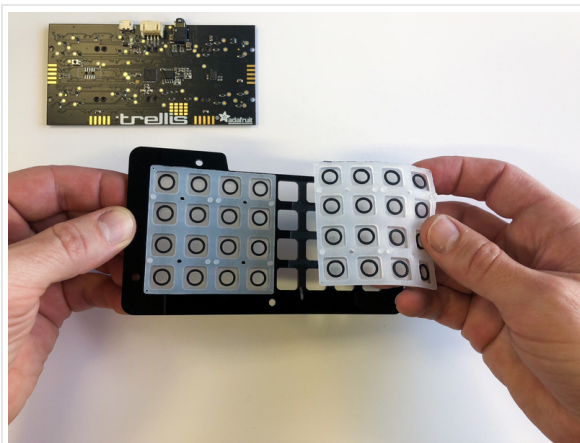
Prep

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.
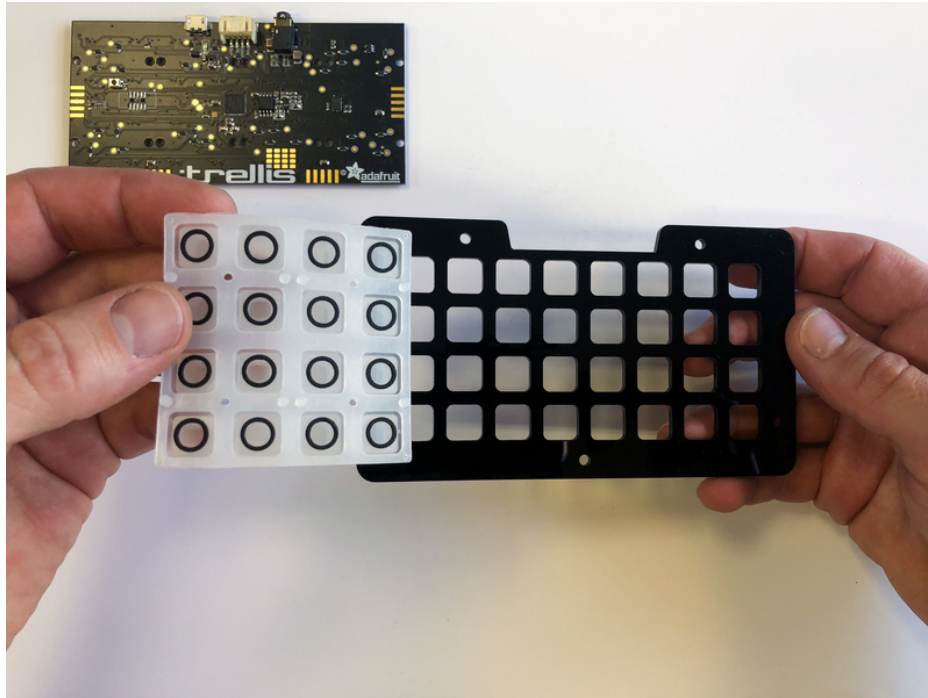
Place the Trellis M4

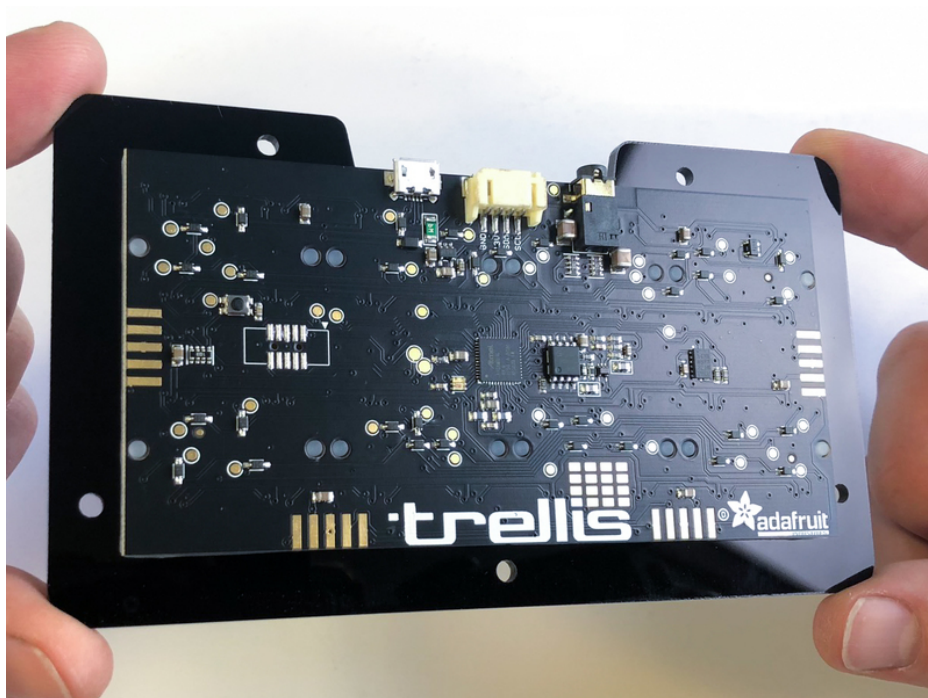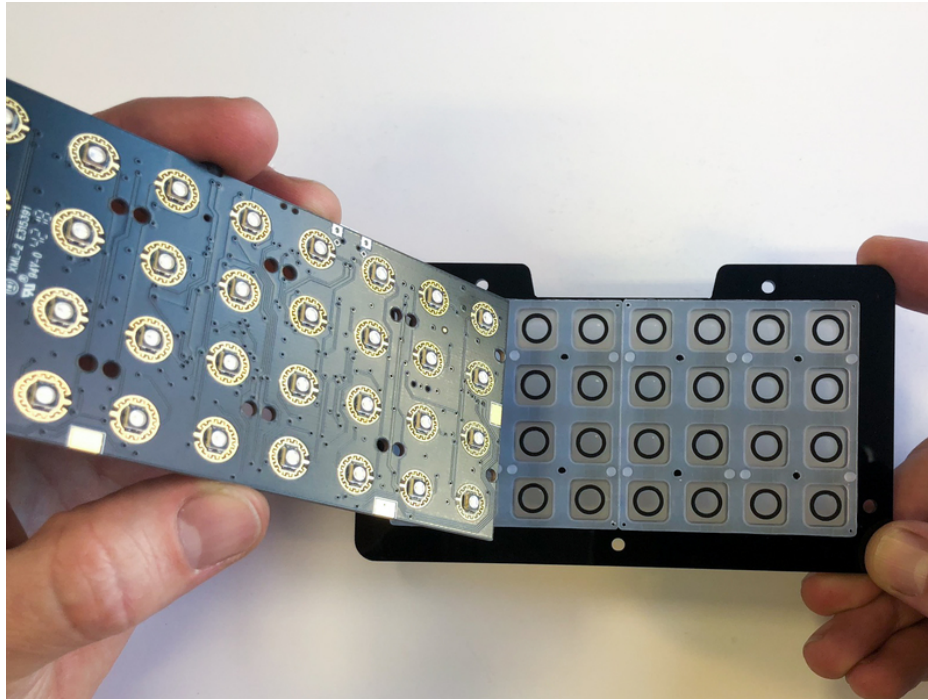Place the Trellis M4 board down onto the elastomer pads with the NeoPixels and button contacts aligned with the buttons as shown. Make sure all of the sixteen registration nubs fit into their associated holes on the PCB.

Frame Layer

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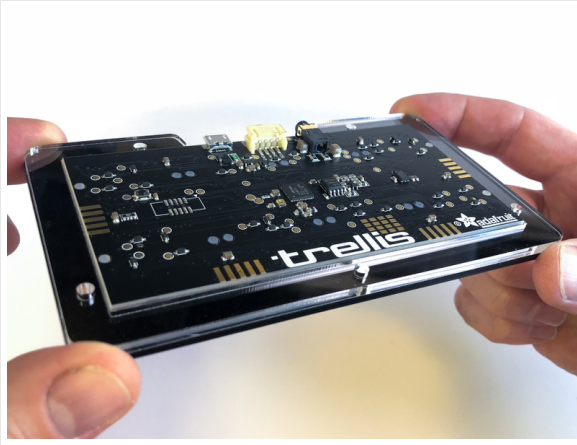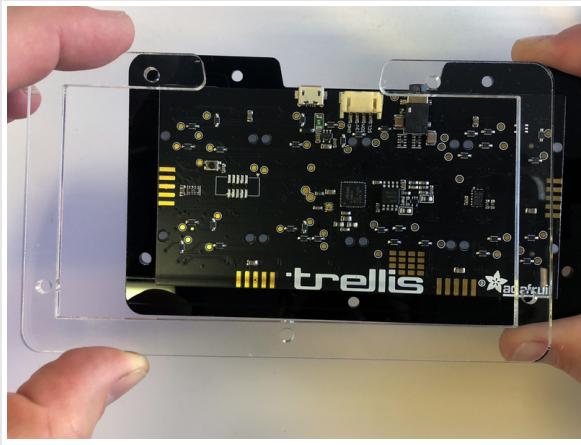
Penultimate Layer

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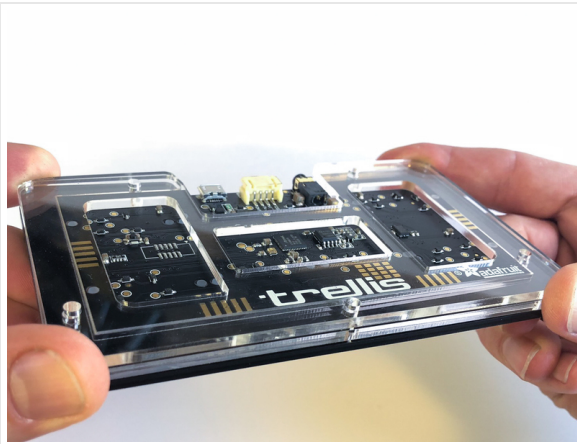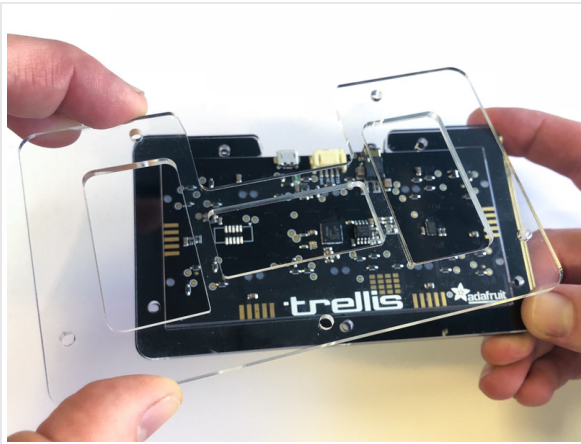
Back Layer

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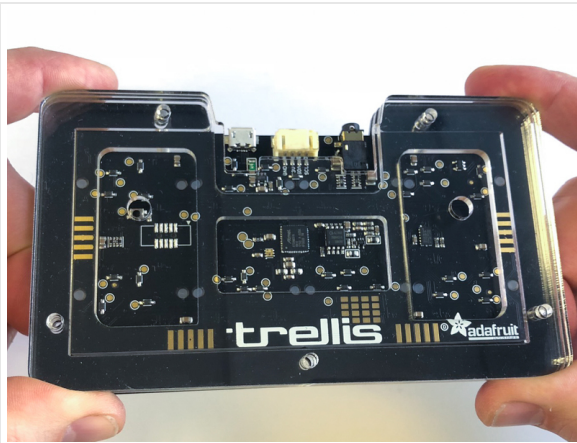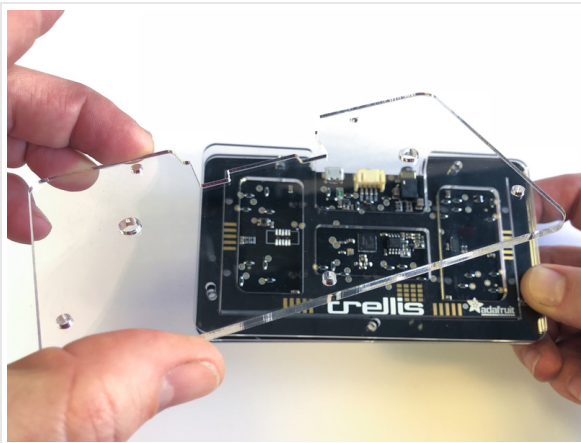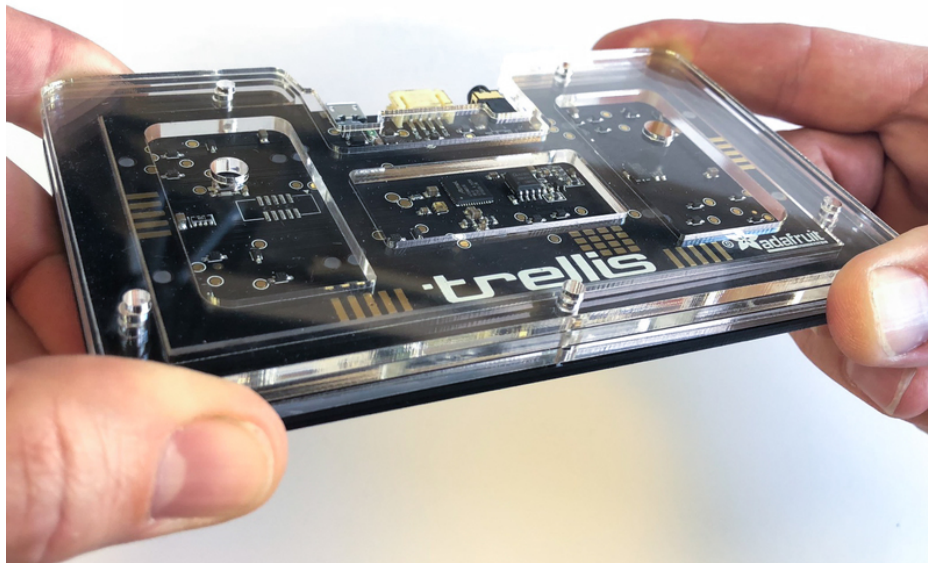
Fasteners

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.
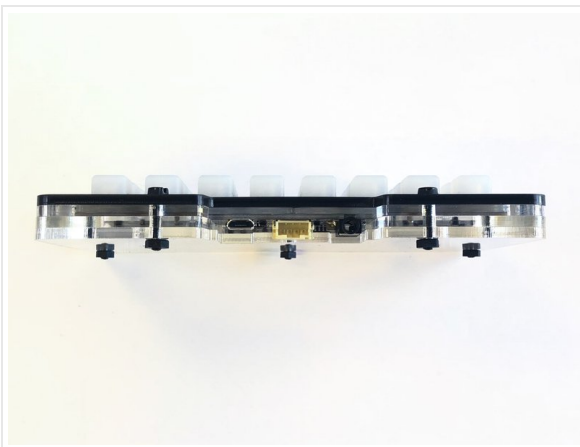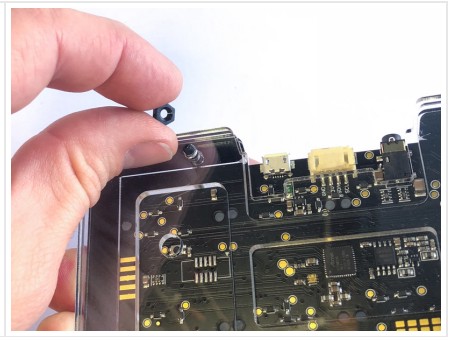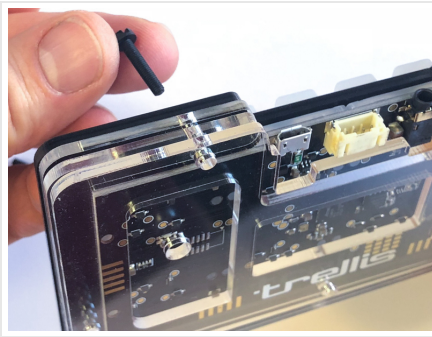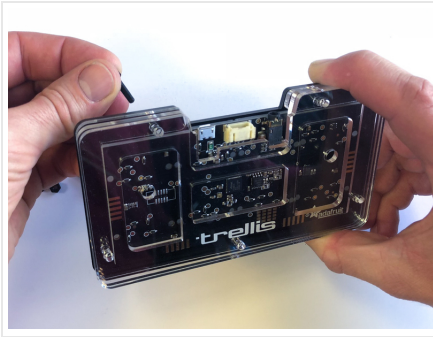
Your Trellis M4 is ready for use!

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). Use them on a laser cutter to created 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!

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

After you have downloaded and installed **the latest version of Arduino IDE**, you will need to start the IDE and navigate to the **Preferences** menu. You can access it from the **File** menu in *Windows* or *Linux*, or the **Arduino** menu on *OS X*.
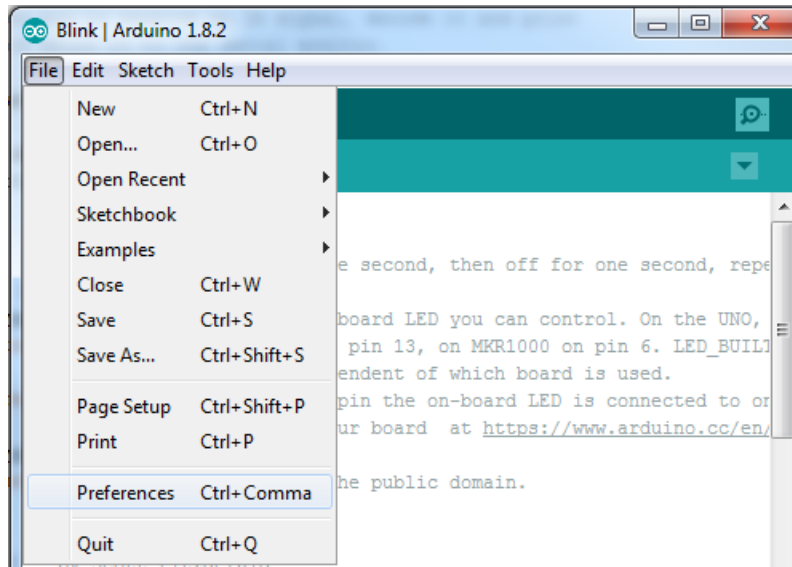
A dialog will pop up just like the one shown below.
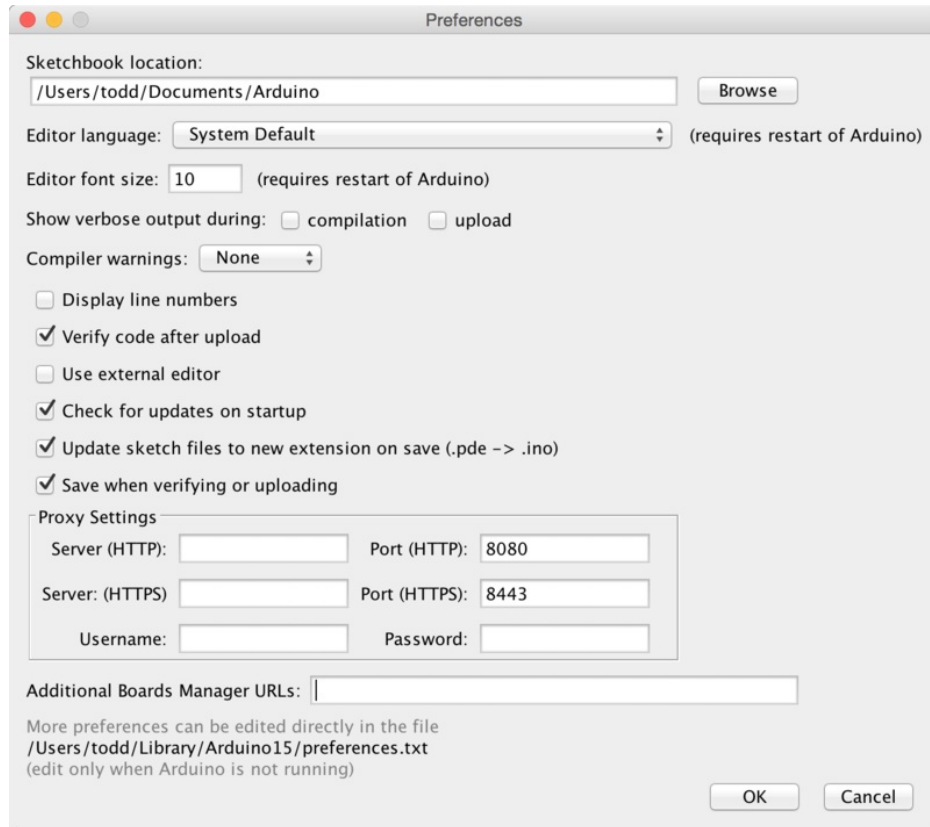
We will be adding a URL to the new **Additional Boards Manager URLs** option. The list of URLs is comma separated, and *you will only have to add each URL once.* New Adafruit boards and updates to existing boards will automatically be picked up by the Board Manager each time it is opened. The URLs point to index files that the Board Manager uses to build the list of available & installed boards.

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

https://adafruit.github.io/arduino-board-index/package_adafruit_index.json
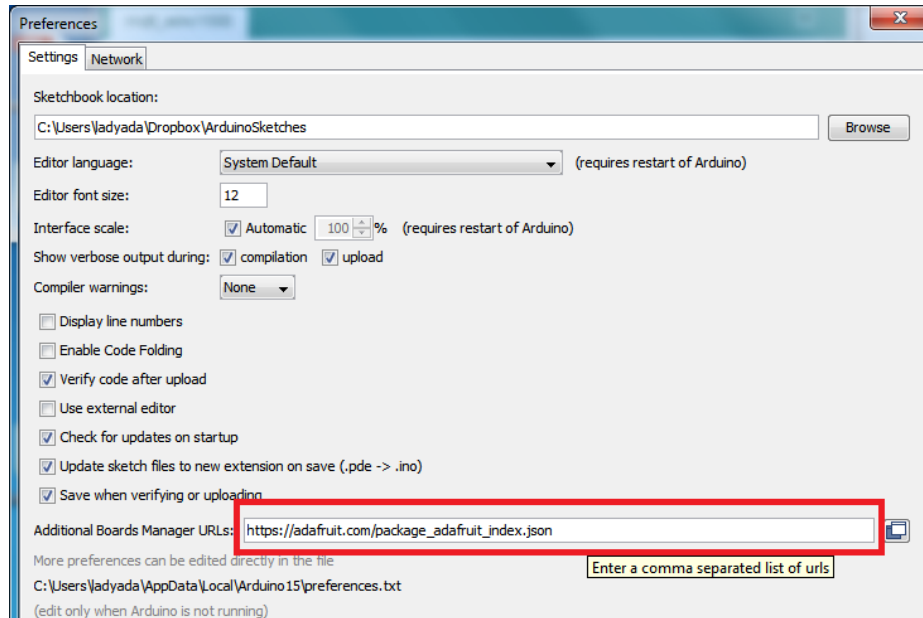
Here's a short description of each of the Adafruit supplied packages that will be available in the Board Manager when you add the URL:

- **Adafruit AVR Boards** - Includes support for Flora, Gemma, Feather 32u4, 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://adafru.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

The Feather/Metro/Gemma/Trinket M0 and M4 use an ATSAMD21 or ATSAMD51 chip, and you can pretty easily get it working with the Arduino IDE. Most libraries (including the popular ones like NeoPixels and display) will work with the M0 and M4, 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.
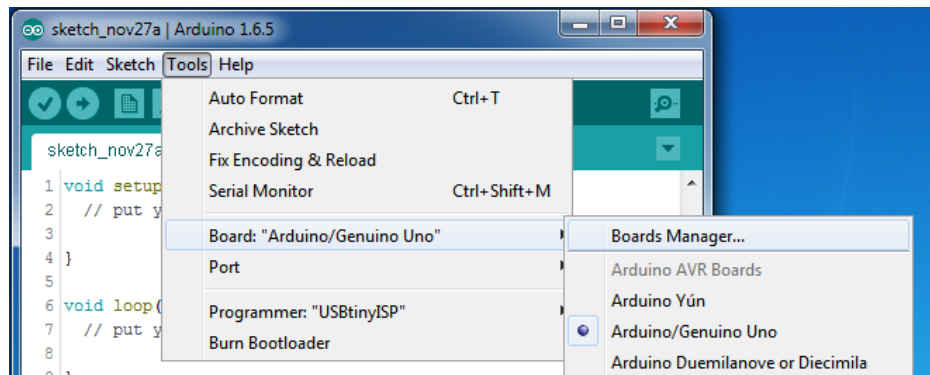
Once the Board Manager opens, click on the category drop down menu on the top left hand side of the window and select **All**. You will then be able to select and install the boards supplied by the URLs added to the preferences.

> 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**
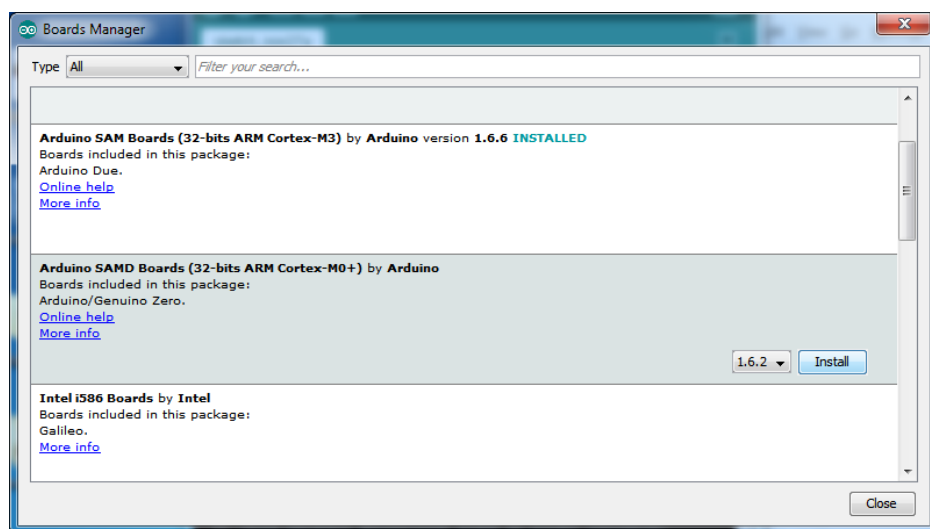
Install Adafruit SAMD

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**
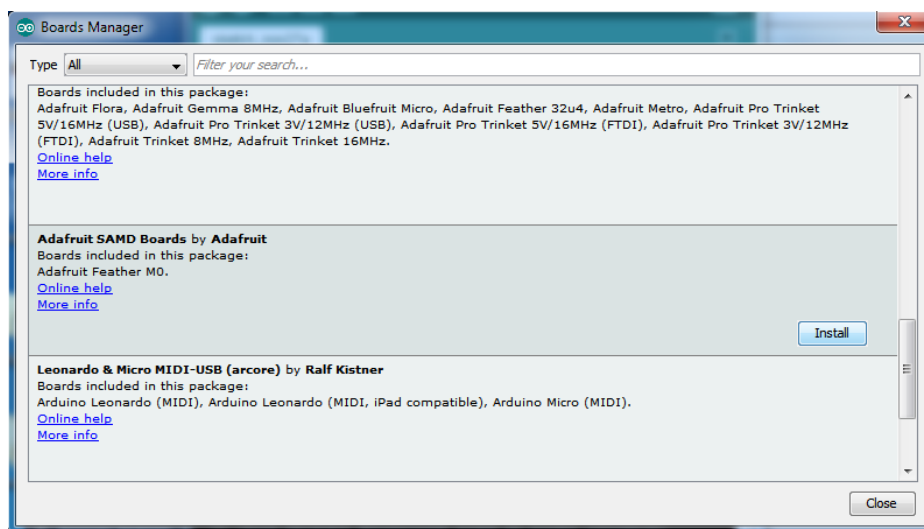
Even though in theory you don't need to - I recommend rebooting the IDE

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**
- **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**
- **ItsyBitsy M4 Express**
- **Feather M4 Express**
- **Trellis M4 Express**
- **Grand Central M4 Express**
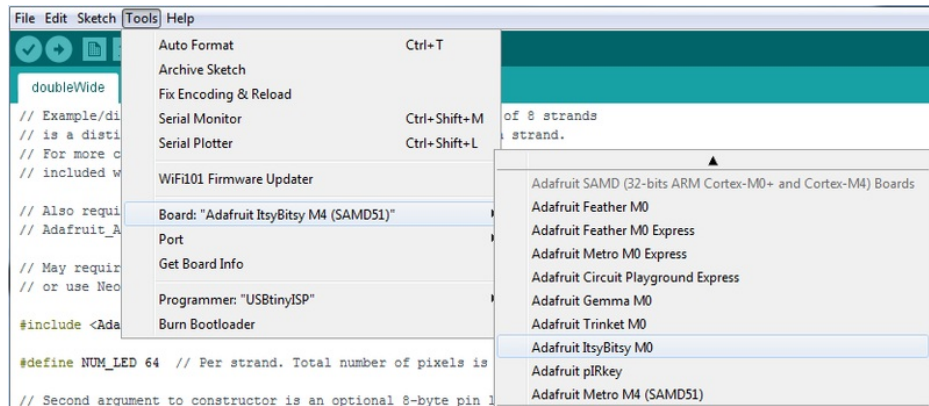
Install Drivers (Windows 7 & 8 Only)

When you plug in the board, you'll need to possibly install a driver

Click below to download our Driver Installer

https://adafru.it/EC0

https://adafru.it/EC0

Download and run the installer
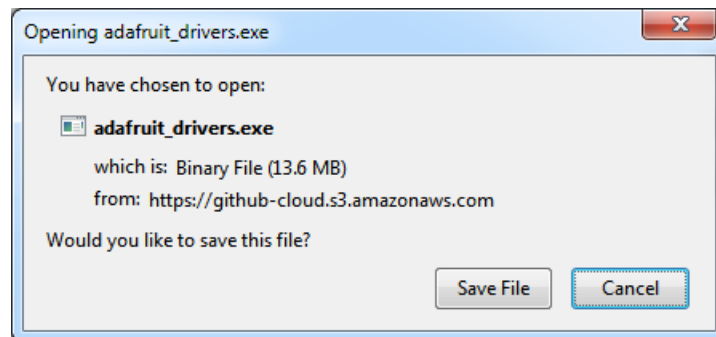
Run the installer! Since we bundle the SiLabs and FTDI drivers as well, you'll need to click through the license
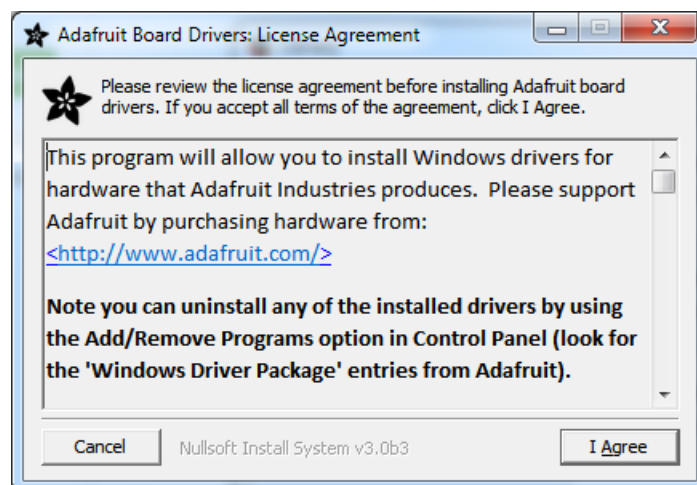
Select which drivers you want to install, the defaults will set you up with just about every Adafruit board!
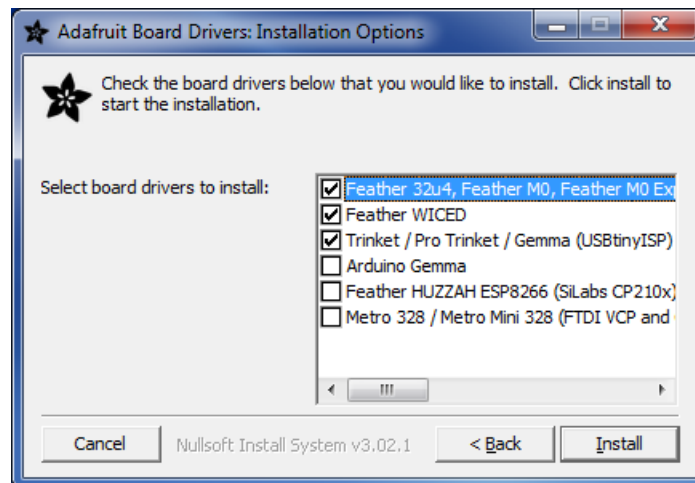
Click **Install** to do the installin'
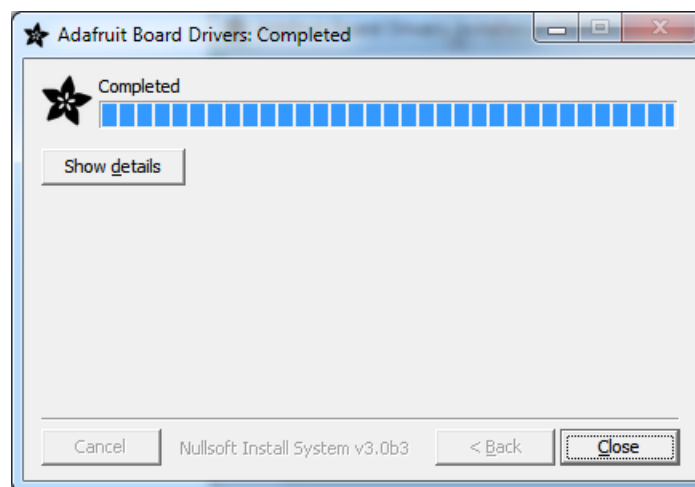
Blink

Now you can upload your first blink sketch!

Plug in the M0 or 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/Trellis!
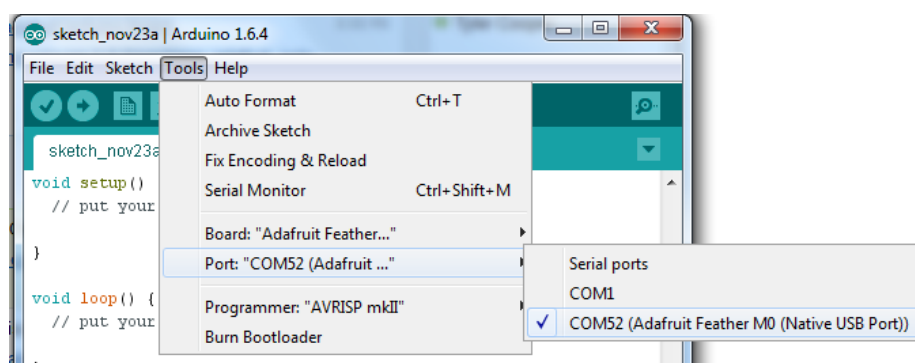
Now load up the Blink example

```
// 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're using **Trellis M4 Express**, you can go to the next page cause there's no pin 13 LED - so you won't see it blink. Still this is a good thing to test compile and upload!

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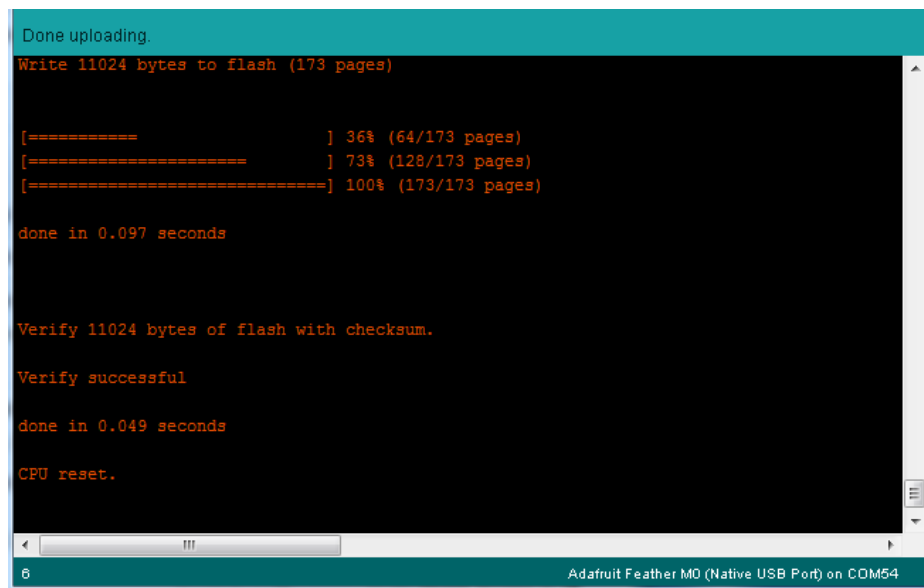
After uploading, you may see a message saying "Disk Not Ejected Properly" about the ...BOOT drive. You can ignore that message: it's an artifact of how the bootloader and uploading work.

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
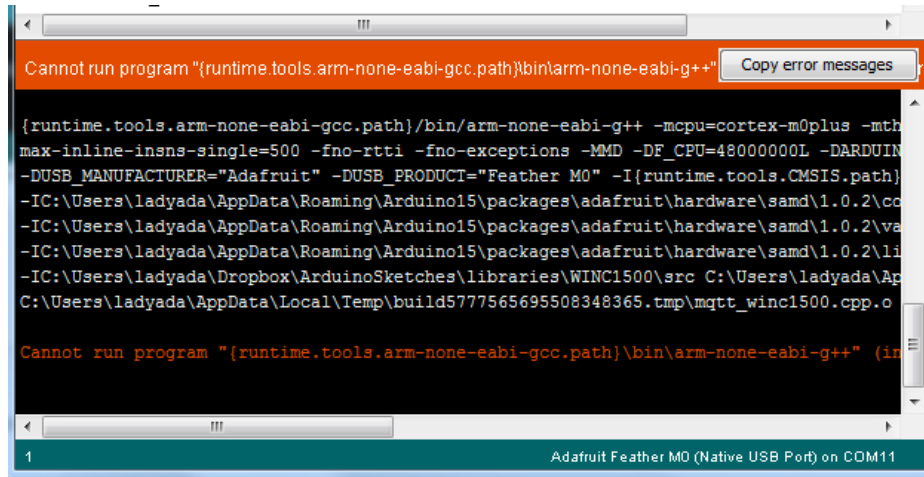
Manually bootloading

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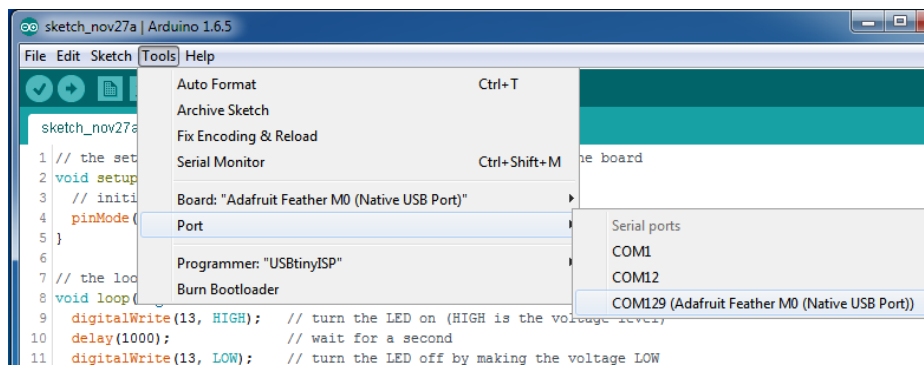
You may need to go back and reselect the 'normal' USB serial port next time you want to use the normal upload.

Ubuntu & Linux Issue Fix

Note if you're using Ubuntu 15.04 (or perhaps other more recent Linux distributions) there is an issue with the modem manager service which causes the Bluefruit LE micro to be difficult to program. If you run into errors like "device or resource busy", "bad file descriptor", or "port is busy" when attempting to program then you are hitting this issue. (https://adafru.it/sHE)

The fix for this issue is to make sure Adafruit's custom udev rules are applied to your system. One of these rules is made to configure modem manager not to touch the Feather board and will fix the programming difficulty issue. Follow the steps for installing Adafruit's udev rules on this page. (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...
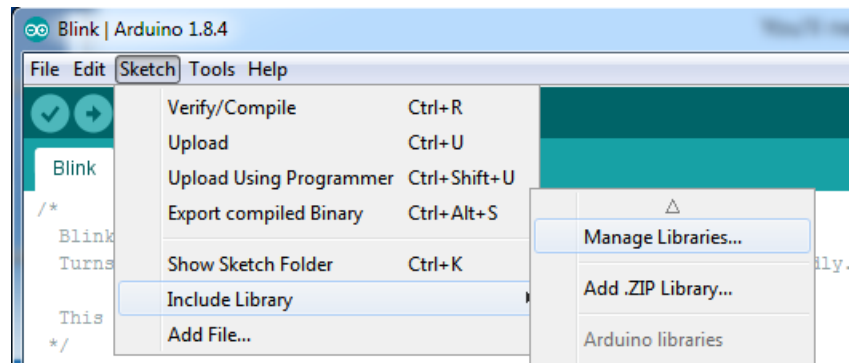
And install the following libraries:

Adafruit NeoPixel

This will let you light up the LEDs on the front
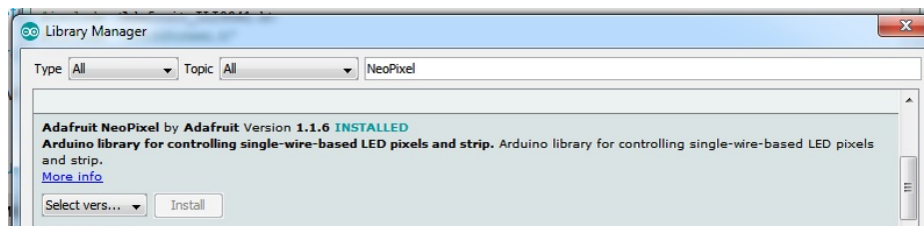
Adafruit DMA NeoPixel

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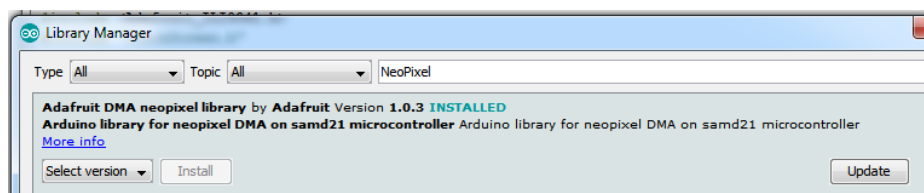
Adafruit Unified Sensor

The underlying sensor library for ADXL343 support
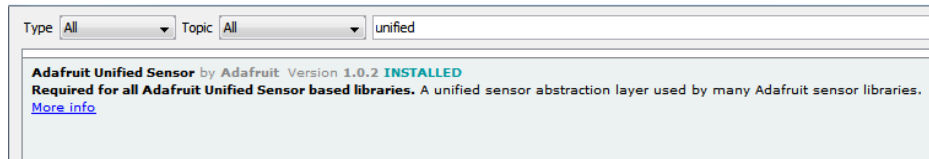
Adafruit QSPI

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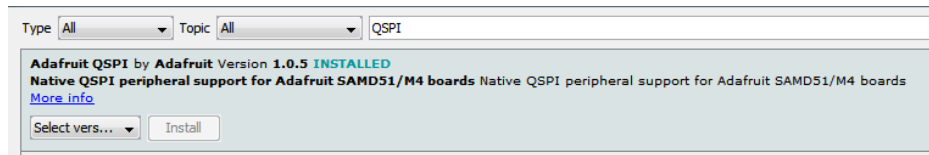
Adafruit Zero DMA

This is used by the PJRC Audio Library you'll install later.
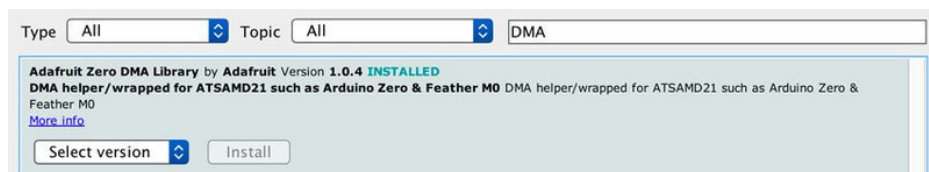
Adafruit SPIFlash

This is also used by the PJRC Audio Library.
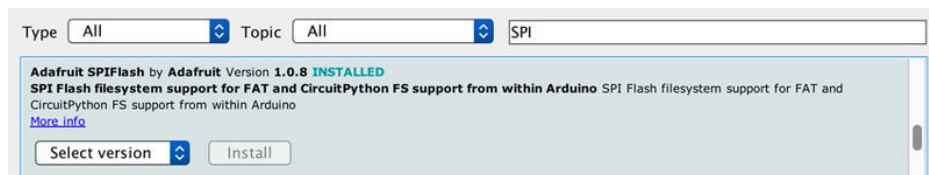
Adafruit Keypad

Our Keypad support library (for reading the button matrix)
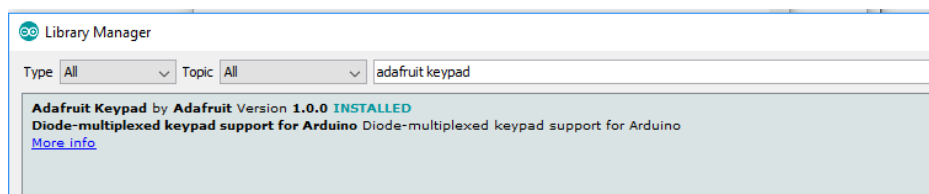
MIDI USB

So you can have the Trellis M4 act like a MIDI device over USB
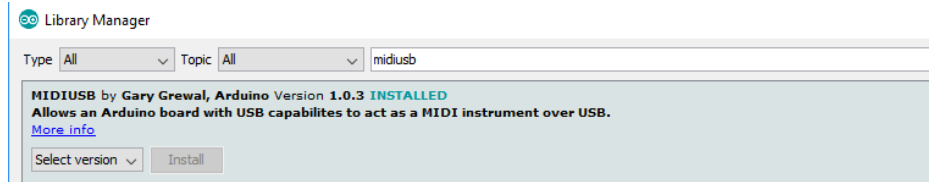
ADXL343

The ADXL343 Library which provides accelerometer support
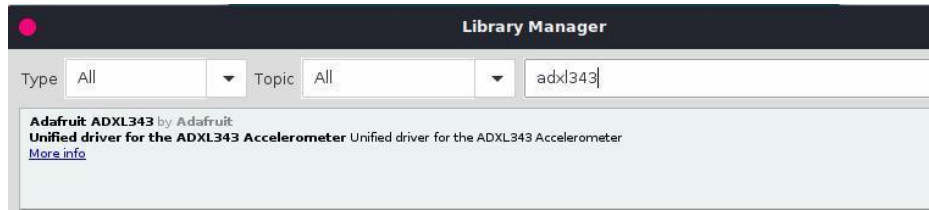
NeoTrellis M4

The NeoTrellis_M4 Library that handles MIDI, LEDs & button presses
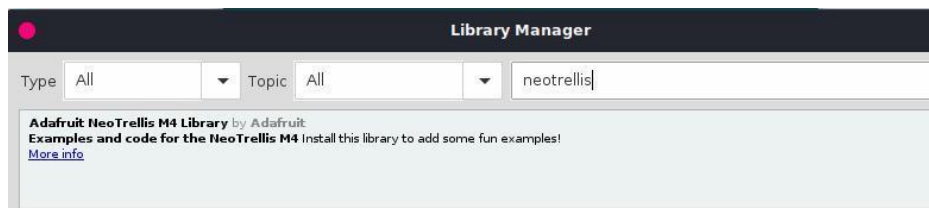
Next Install via Zip:
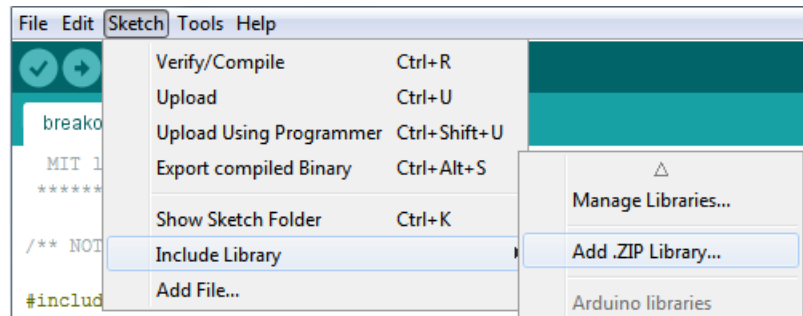
Our fork of the PJRC Audio library to support the SAMD51

https://adafru.it/CVX

https://adafru.it/CVX

Adapting Sketches to M0 & M4

The ATSAMD21 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)`.

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_AR
 #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 AVRs, 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 wont 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 ATSAMD21G18 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, its 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 dont 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, its 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!

https://github.com/drewfish/arduino-ZeroRegs (https://adafru.it/Bet)

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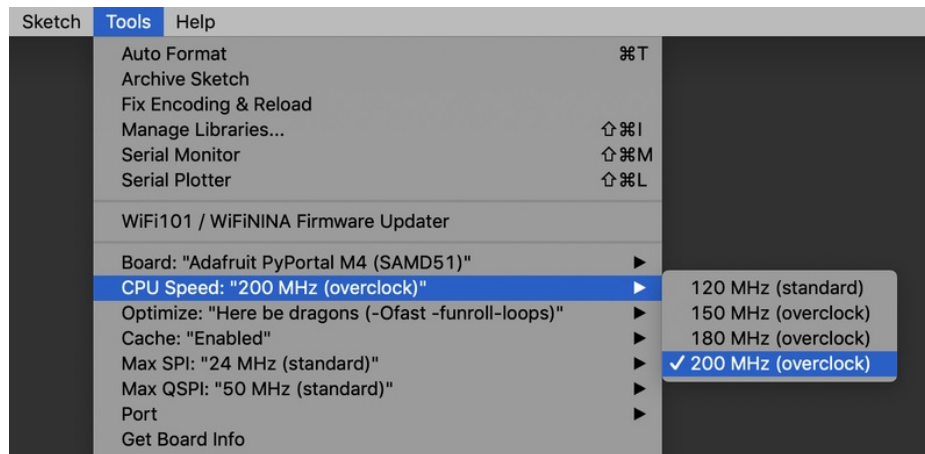
All of these performance tweaks involve a degree of uncertainty. There's *no guarantee* of improved performance in any given project, and *some may even be detrimental,* failing to work in part or in whole. If you encounter trouble, **select the default performance settings** and re-upload.

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) noticably 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) runs marginally better with it, if using the QSPI flash.

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://adafru.it/CW5)

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

NeoPixel Test

Load this example (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) 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) 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), 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). 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). 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.** There are 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. We welcome and encourage feedback from the community, and we incorporate this into how we are developing 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!
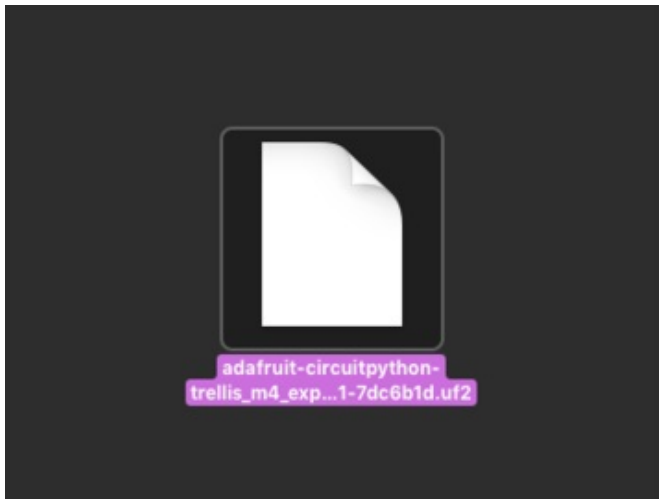
Click the link above and download the latest UF2 file.

Download and save it to your desktop (or wherever is handy).
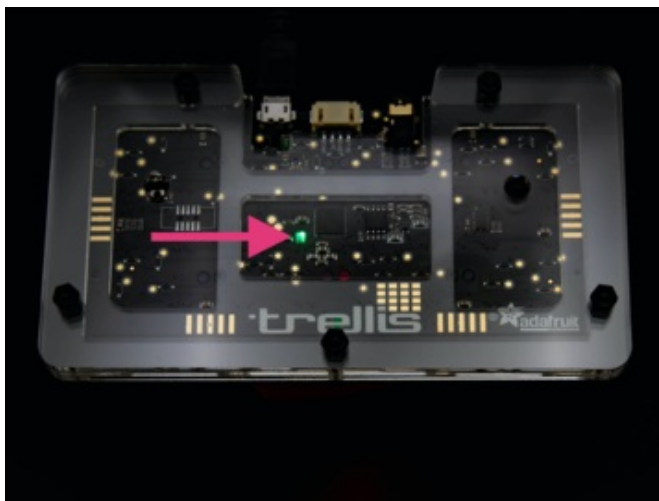
Plug your NeoTrellis M4 Express into your computer using a known-good USB cable.

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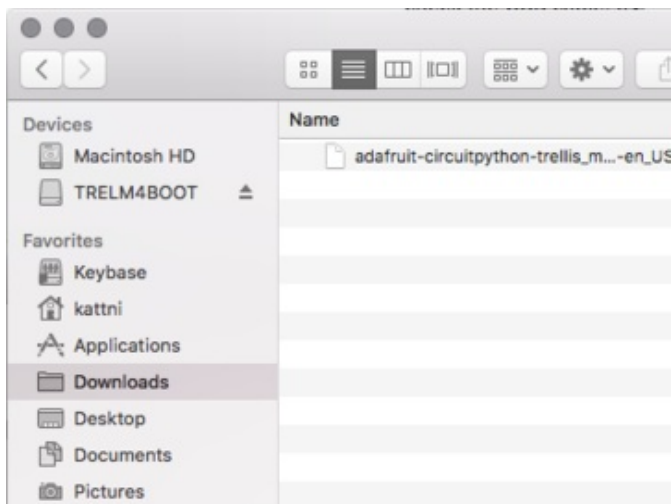
You will see a new flash disk drive appear called **TRELM4BOOT**.

Drag the **adafruit_circuitpython_etc.uf2** file to **TRELM4BOOT**.
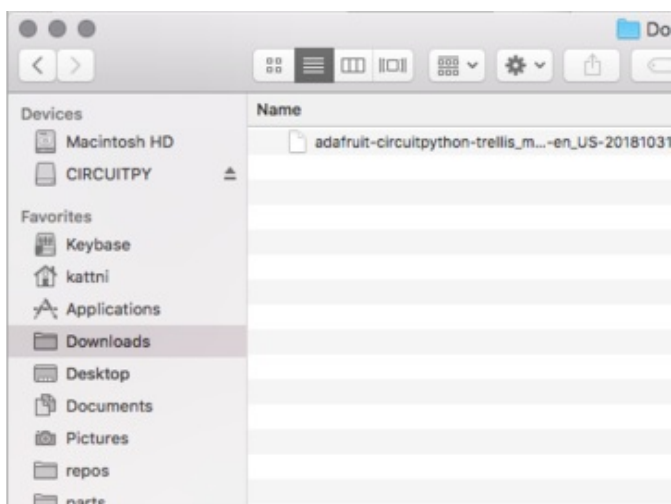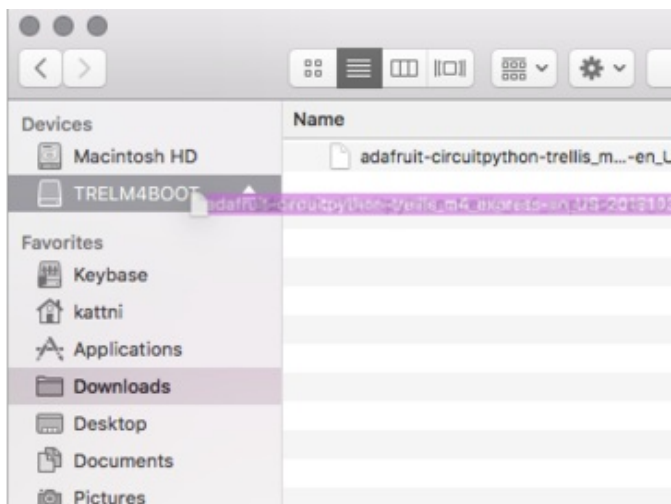
The LED will flash. Then, the **TRELM4BOOT** drive will disappear and a new disk drive called **CIRCUITPY** will appear.

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

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) 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.

<div style="background:green">https://adafru.it/y8E</div>

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).

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) is also an editor that will fully write files on save (https://adafru.it/Be7)
- vim (https://adafru.it/ek9) / **vi** safely writes all changes
- Sublime Text (https://adafru.it/xNB) safely writes all changes
- The PyCharm IDE (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), install the fsync-on-save package (https://adafru.it/E9m) so that it will always write out all changes to files on `CIRCUITPY`.
- Visual Studio Code (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...

<div>

☐   Don't Click Reset or Unplug!

</div>

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 int 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` let's 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).

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). 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).

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:
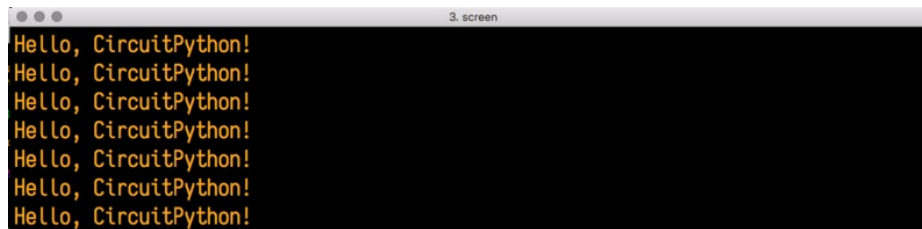
```
import time
import adafruit_trellism4

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.
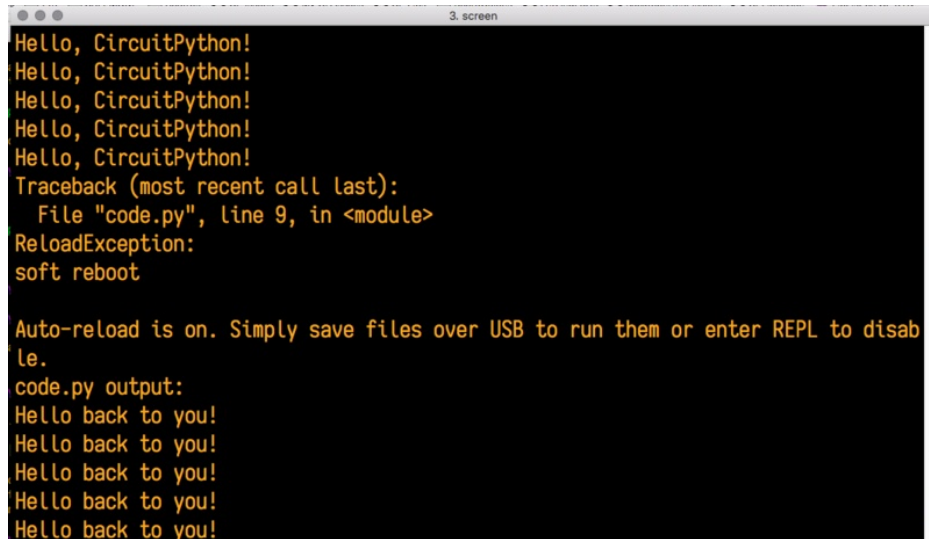

Excellent! Our print statement is showing up in our console! Try changing the printed text to something else.

```
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!

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 .
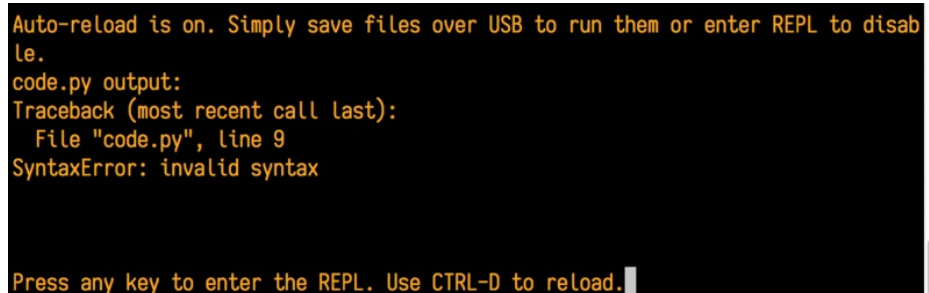
```
import time
import adafruit_trellism4

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!
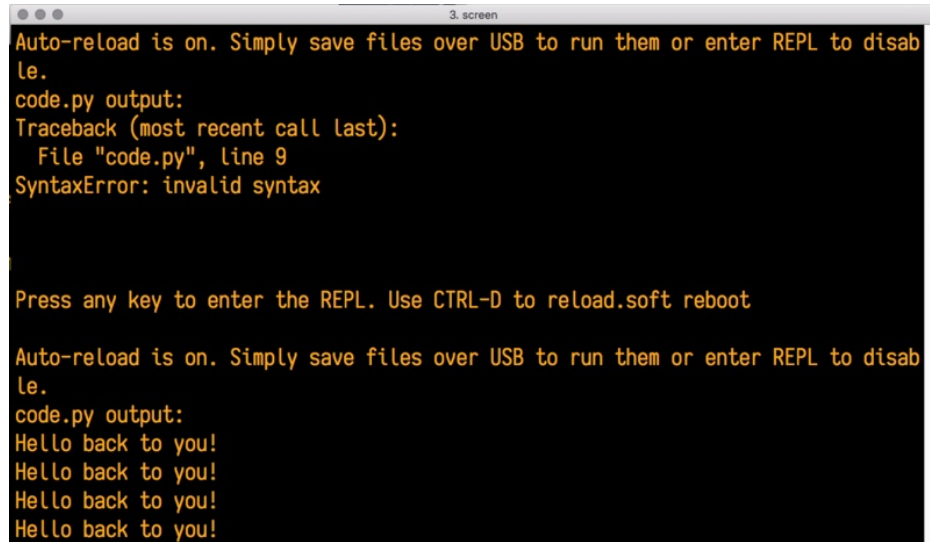

The Traceback (most recent call last): is telling you that the last thing it was able to run was line 9 in your code. The

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.

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!
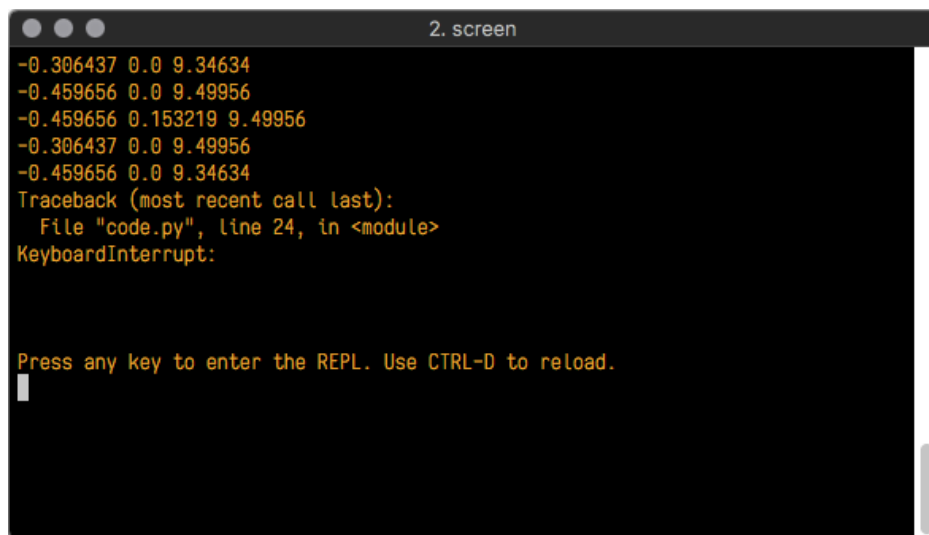
The REPL

The other feature of the serial connection is the **R**ead-**E**valuate-**P**rint-**L**oop, 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.

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, 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.

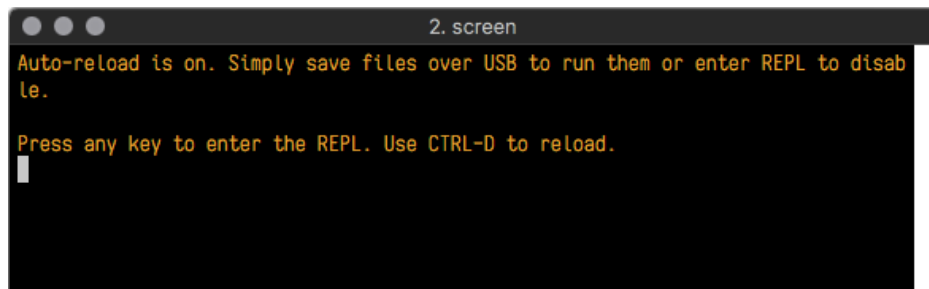The Traceback (most recent call last): is telling you the last thing your board was doing before you pressed Ctrl + C and interrupted it. The KeyboardInterrupt is you pressing Ctrl + C. This information can be handy when troubleshooting, but for now, don't worry about it. Just note that it is expected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

If there is no code running, you will enter the REPL immediately after pressing Ctrl + C. There is no information about what your board was doing before you interrupted it because there is no code running.

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

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

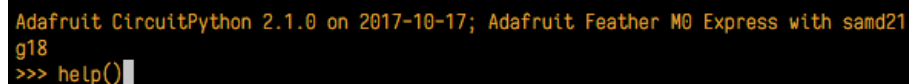The first thing you get from the REPL is information about your board.

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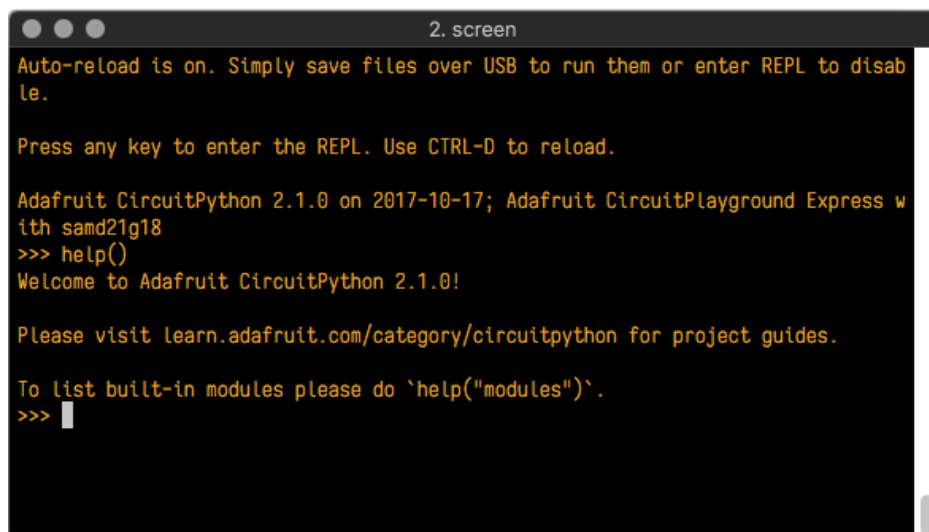
From this prompt you can run all sorts of commands and code. The first thing we'll do is run `help()`. This will tell us where to start exploring the REPL. To run code in the REPL, type it in next to the REPL prompt.

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

Then press enter. You should then see a message.

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, please do \`help("modules")\`.` Remember the libraries 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.

```
● ● ●                          3. screen
Adafruit CircuitPython 2.1.0 on 2017-10-17; Adafruit Feather M0 Express with sam
d21g18
>>> help()
Welcome to Adafruit CircuitPython 2.1.0!

Please visit learn.adafruit.com/category/circuitpython for project guides.

To list built-in modules please do `help("modules")`.
>>> help("modules")
__main__          busio           neopixel_write    time
analogio          digitalio       nvm               touchio
array             framebuf        os                ucollections
audiobusio        gamepad         pulseio           ure
audioio           gc              random            usb_hid
bitbangio         math            samd              ustruct
board             microcontroller storage
builtins          micropython     sys
Plus any modules on the filesystem
>>>
```
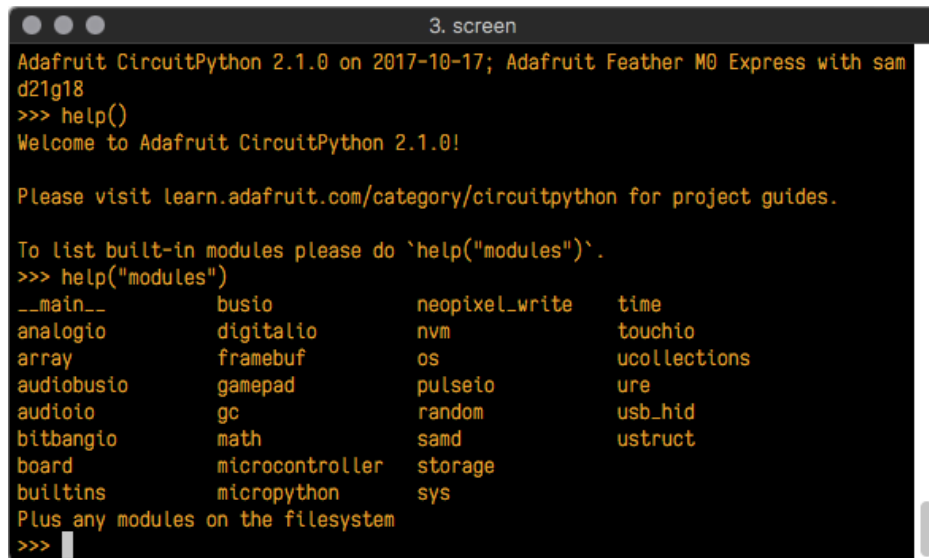
This is a list of all the core libraries built into CircuitPython. We discussed how board contains all of the pins on the board that you can use in your code. From the REPL, you are able to see that list!

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.

```
● ● ●                          3. screen
d21g18
>>> help()
Welcome to Adafruit CircuitPython 2.1.0!

Please visit learn.adafruit.com/category/circuitpython for project guides.

To list built-in modules please do `help("modules")`.
>>> help("modules")
__main__          busio           neopixel_write    time
analogio          digitalio       nvm               touchio
array             framebuf        os                ucollections
audiobusio        gamepad         pulseio           ure
audioio           gc              random            usb_hid
bitbangio         math            samd              ustruct
board             microcontroller storage
builtins          micropython     sys
Plus any modules on the filesystem
>>> import board
>>>
```

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

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 `D13` ? 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." We're going to say hello to something else. Type into the REPL:

`print("Hello, CircuitPython!")`

Then press enter.

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. As we said though, 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 libraries are available and explore those libraries.

Try typing more into the REPL to see what happens!

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!

CircuitPython Libraries

As we continue to develop CircuitPython and create new releases, we will stop supporting older releases. If you are running CircuitPython 2.x, you need to update to 3.x. You must download the CircuitPython Library Bundle that matches your version of CircuitPython. Please update to CircuitPython 3.x and then download the 3.x 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 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, it's in the base directory of the drive. If not, simply create the folder yourself.
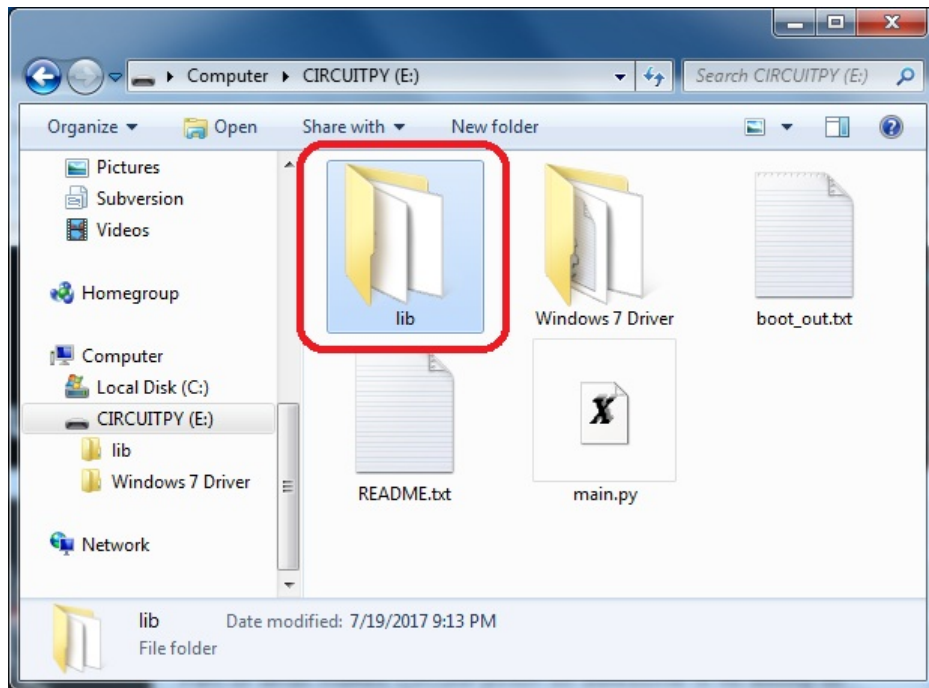
CircuitPython libraries work in the same was as regular Python modules so the Python docs (https://adafru.it/rar) are a great reference for how it all should work. In Python terms, we can place our library files in the **lib** directory because its part of the Python path by default.

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, we provide a bundle full of our libraries.

Our bundle and releases 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.

Installing the CircuitPython Library Bundle

We're constantly updating and improving our libraries, so we don't (at this time) ship our CircuitPython boards with the

full library bundle. Instead, you can find example code in the guides for your board that depends on external libraries. Some of these libraries may be available from us at Adafruit, some may be written by community members!

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

You can grab the latest Adafruit CircuitPython Bundle release by clicking the button below.

Note: Match up the bundle version with the version of CircuitPython you are running - 3.x library for running any version of CircuitPython 3, 4.x for running any version of CircuitPython 4, etc. If you mix libraries with major CircuitPython versions, you will most likely get errors due to changes in library interfaces possible during major version changes.

<div style="background-color:green; text-align:center;">https://adafru.it/Ayy</div>

https://adafru.it/Ayy

If you need another version, you can also visit the bundle release page (https://adafru.it/Ayy) which will let you select exactly what version you're looking for, as well as information about changes.

Either way, download the version that matches your CircuitPython firmware version. If you don't know the version, look at the initial prompt in the CircuitPython REPL, which reports the version. For example, if you're running v3.0.2, download the v3 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.

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.
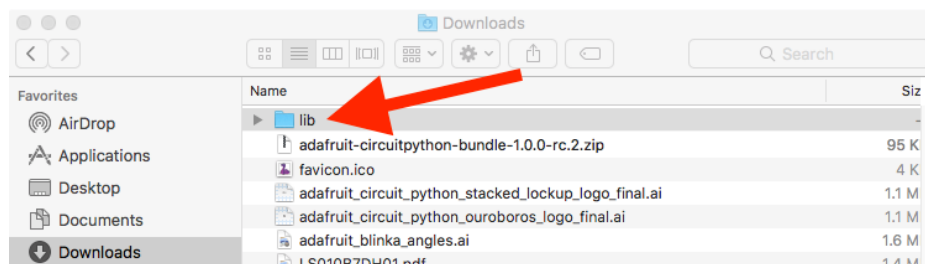
When you open the folder, you'll see a large number of **mpy** files and folders
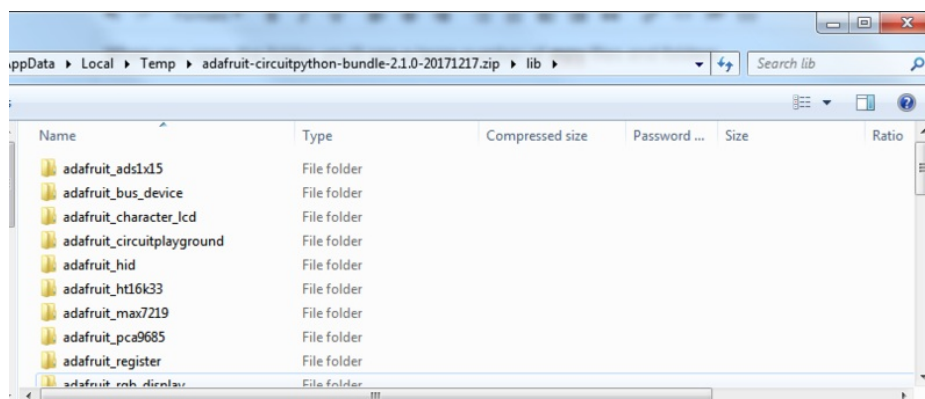
Example Files

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

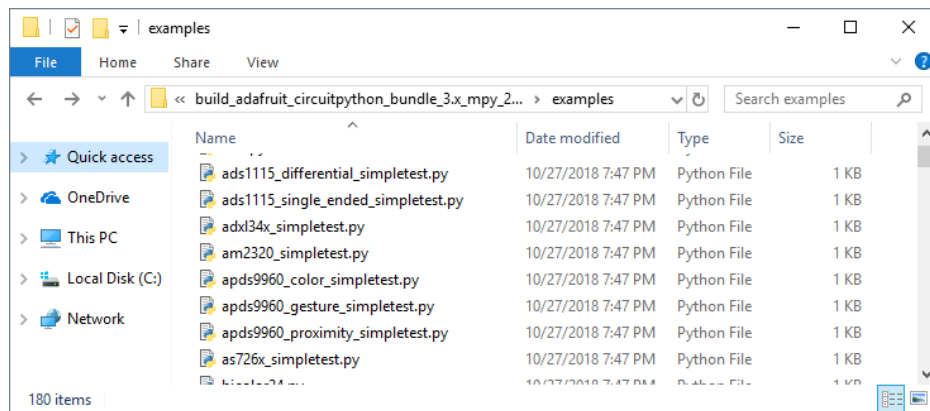- Allow for quick testing of devices.
- Provide an example base of code, that is easily built upon for individualized purposes.

Express Boards

If you are using a Feather M0 Express, Metro M0 Express or Circuit Playground Express (or any other "Express" board) your CircuitPython board comes with at least 2 MB of Flash storage. This is *plenty* of space for all of our library files so we recommend you just install them all! (If you have a Gemma M0 or Trinket M0 or other non-Express board, skip down to the next section)

On Express boards, the **lib** and **examples** directories can be copied directly to the **CIRCUITPY** drive.

Just drag the entire **lib** and **examples** (optional) folders into the **CIRCUITPY** drive, and 'replace' any old files if your operating system prompts you.

To use any of the supplied example files, from either the REPL or inside a **code.py/main.py**, simply import them using the filename:

```
import examples.adxl34x_simpletest
```

Non-Express Boards

If you are using Trinket M0 or Gemma M0, you will need to load the libraries individually, due to file space restrictions. If you are using a non-express board, or you would rather load libraries as you use them, you'll first want to create a **lib** folder on your **CIRCUITPY** drive. Open the drive, right click, choose the option to create a new folder, and call it **lib**. 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**.

This also applies to example files. They are only supplied as raw **.py** files, so they may need to be converted to **.mpy** using the **mpy-cross** utility if you encounter `MemoryErrors`. This is discussed in the CircuitPython Essentials Guide (https://adafru.it/CTw). Usage is the same as described above in the Express Boards section. Note: If you do not place examples in a separate folder, you would remove the examples from the `import` statement.

Example: `ImportError` Due to Missing Library

If you choose to load libraries as you need them, you may write up code that tries to use a library you haven't yet loaded. We're going to 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**.**
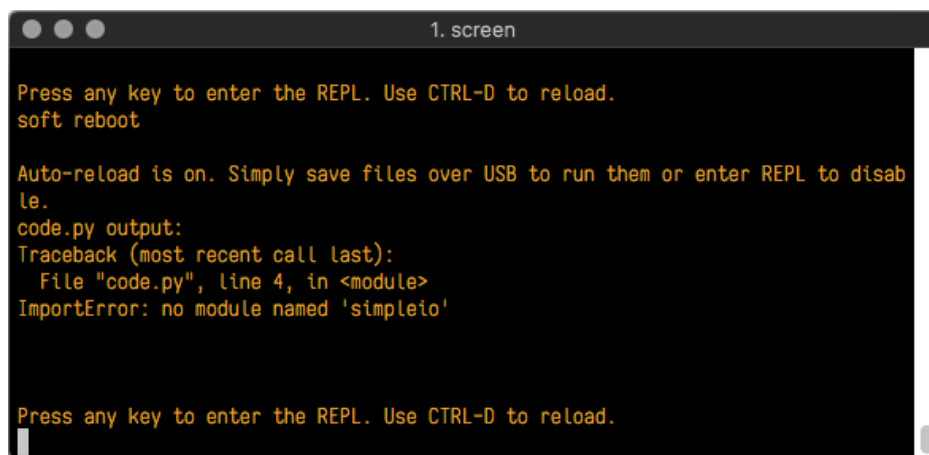
Let's use a modified version of the blinky example.

```
import board
import time
import simpleio

led = simpleio.DigitalOut(board.D13)

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.

We have an ImportError . It says there is no module named 'simpleio' . That's the one we just included in our 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 we're looking for! Follow the steps above to load an individual library file.

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

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 a Trinket M0 or Gemma M0, you'll want to follow the same steps in the example above to install libraries as you need them. You don't always need to wait for an `ImportError` as you probably know what library you added to your code. Simply open the **lib** folder you downloaded, find the library you need, and drag it to the **lib** folder on your **CIRCUITPY** drive.

For these boards, your internal storage is from the chip itself. So, these boards don't have enough space for all of the libraries. If you try to copy over the entire **lib** folder you won't have enough space on your **CIRCUITPY** drive.

You may end up running out of space on your Trinket M0 or Gemma M0 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 them in the Troubleshooting page in the Learn guides for your board.

Updating CircuitPython Libraries/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!

If you'd like to update the entire bundle at once, drag the lib and examples folder to your **CIRCUITPY** drive. Different operating systems will have a different dialog pop up. You want to tell it to replace the current folder. Then you're updated and ready to go!

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.

Frequently Asked Questions

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

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

How do I connect to the Internet with CircuitPython?

Is there asyncio support in CircuitPython

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

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 (REPL).

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) 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 the CircuitPython 2.x version of `mpy-cross` for your operating system from the CircuitPython Releases page (https://adafru.it/tBa) under the latest 2.x version.

You can build `mpy-cross` for CircuitPython 3.x by cloning the CircuitPython GitHub repo (https://adafru.it/tB7), and running `make` in the `circuitpython/mpy-cross/` directory. Then run `./mpy-cross path/to/foo.py` to create a `foo.mpy` in the same directory as the original file.

How do I check how much memory I have free?

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

Will give you the number of bytes available for use.

Does CircuitPython support interrupts?

No. CircuitPython does not currently support interrupts. We do not have an estimated time for when they will be included.

Does Feather M0 support WINC1500?

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

Can AVRs such as ATmega328 or ATmega2560 run CircuitPython?

No.

Commonly Used Acronyms

CP or CPy = CircuitPython (https://adafru.it/cpy-welcome)
CPC = Circuit Playground Classic (https://adafru.it/ncE)
CPX = Circuit Playground Express (https://adafru.it/wpF)

Advanced 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.
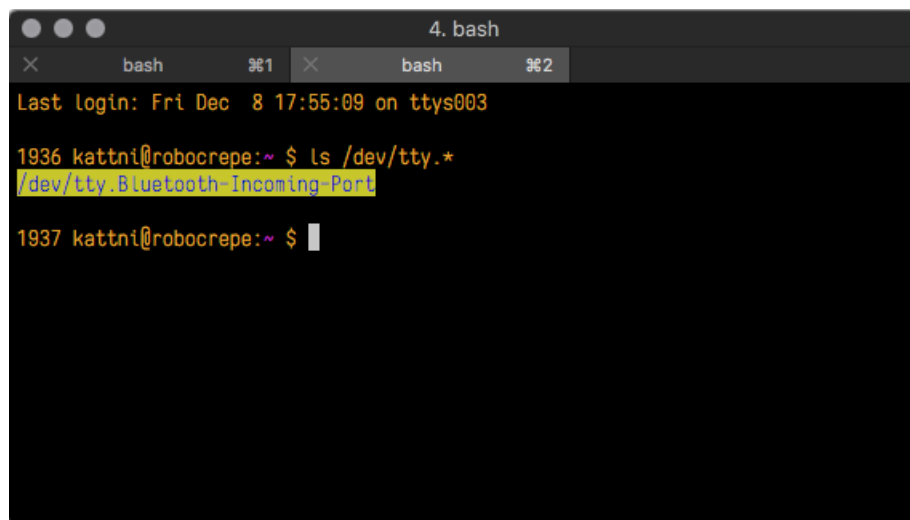
What's the Port?

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.

We're going to use Terminal to determine what 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. On Mac, 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, we'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.

For Linux, the procedure is the same, however, the name is slightly different. If you're using Linux, you'll type:

`ls /dev/ttyACM*`

The concept is the same with Linux. We are asking to see the listings in the `/dev/` folder, starting with `ttyACM` and ending with anything. This will show you the current serial connections. In the example below, the error is indicating that are no current serial connections starting with `ttyACM` .

Now, plug your board. Using Mac, type:

`ls /dev/tty.*`

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

Using Mac, 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, type:

`ls /dev/ttyACM*`

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

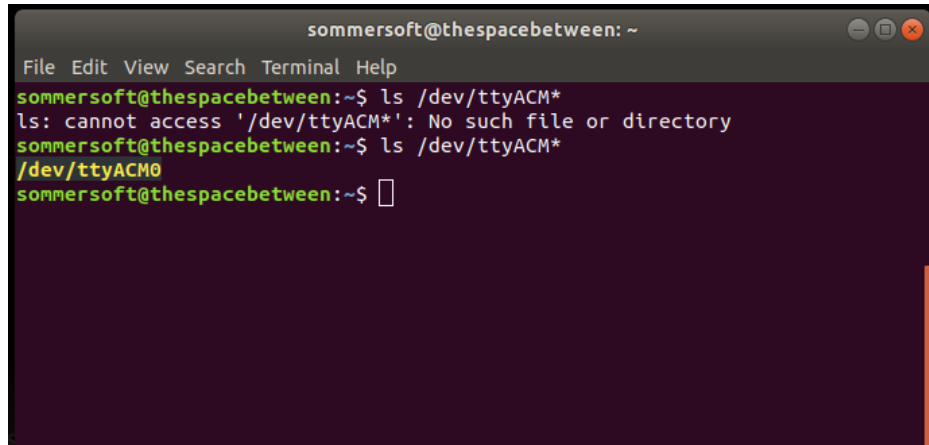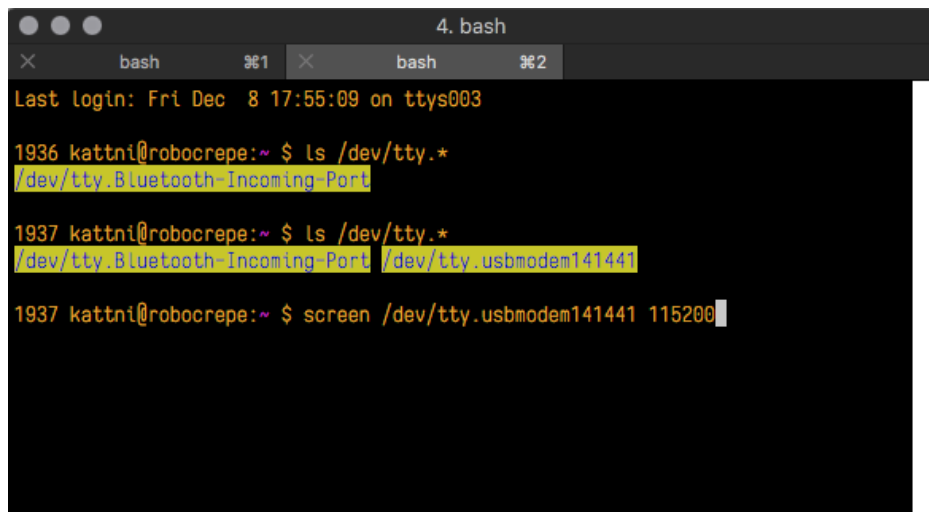Using Linux, a new listing has appeared called /dev/ttyACM0 . The ttyACM0 part of this listing is the name the example board is using. Yours will be called something similar.

Connect with screen

Now that you know the name your board is using, you're ready connect to the serial console. We're going to use a command called screen . The screen command is included with MacOS. Linux users may need to install it using their package manager. 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.

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!

Permissions on Linux

If you try to run `screen` and it doesn't work, then you may be running into an issue with permissions. Linux keeps track of users and groups and what they are allowed to do and not do, like access the hardware associated with the serial connection for running `screen` . So if you see something like this:

then you may need to grant yourself access. There are generally two ways you can do this. The first is to just run `screen` using the `sudo` command, which temporarily gives you elevated privileges.

Once you enter your password, you should be in:

The second way is to add yourself to the group associated with the hardware. To figure out what that group is, use the command `ls -l` as shown below. The group name is circled in red.

Then use the command `adduser` to add yourself to that group. You need elevated privileges to do this, so you'll need to use `sudo` . In the example below, the group is **adm** and the user is **ackbar**.
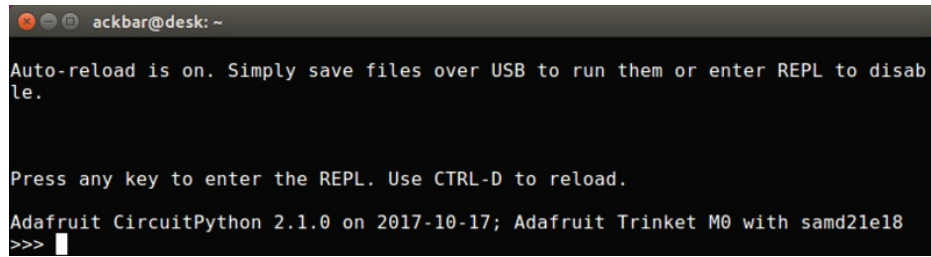
After you add yourself to the group, you'll need to logout and log back in, or in some cases, reboot your machine. After you log in again, verify that you have been added to the group using the command `groups` . If you are still not in the group, reboot and check again.

And now you should be able to run `screen` without using `sudo` .

And you're in:

```
Auto-reload is on. Simply save files over USB to run them or enter REPL to disab
le.


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

Adafruit CircuitPython 2.1.0 on 2017-10-17; Adafruit Trinket M0 with samd21e18
>>>
```

The examples above use screen , but you can also use other programs, such as putty or picocom , if you prefer.

Advanced Serial Console on Windows

Windows 7 Driver

If you're using Windows 7, use the link below to download the driver package. You will not need to install drivers on Mac, Linux or Windows 10.

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.

We'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.
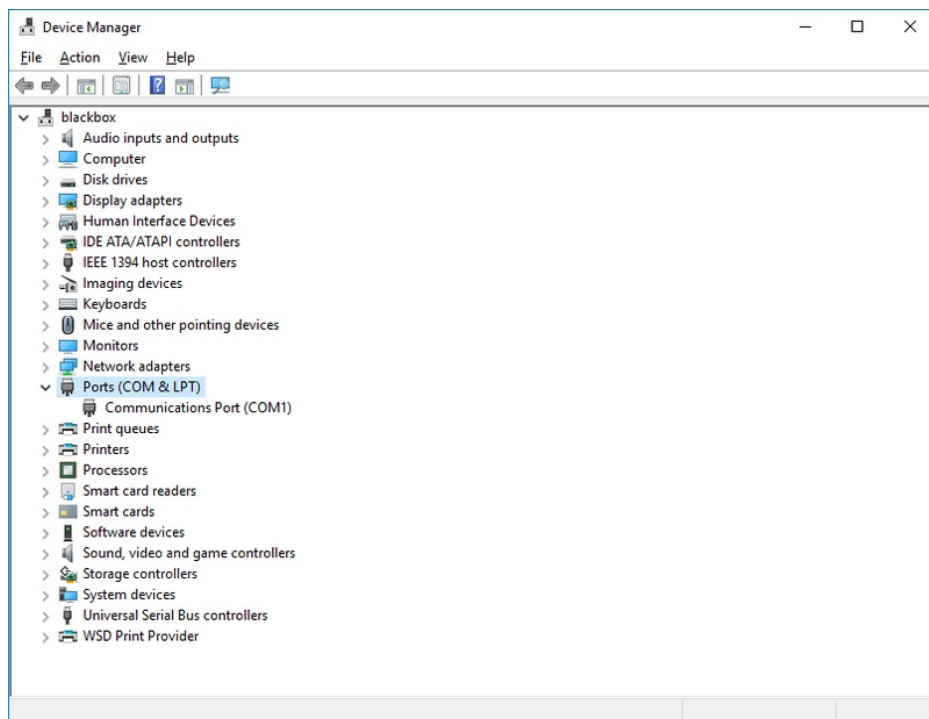
Now plug in your board. The Device Manager list will refresh and a new item will appear under Ports (COM & LPT). You'll find a different (COM#) after this item in the list.
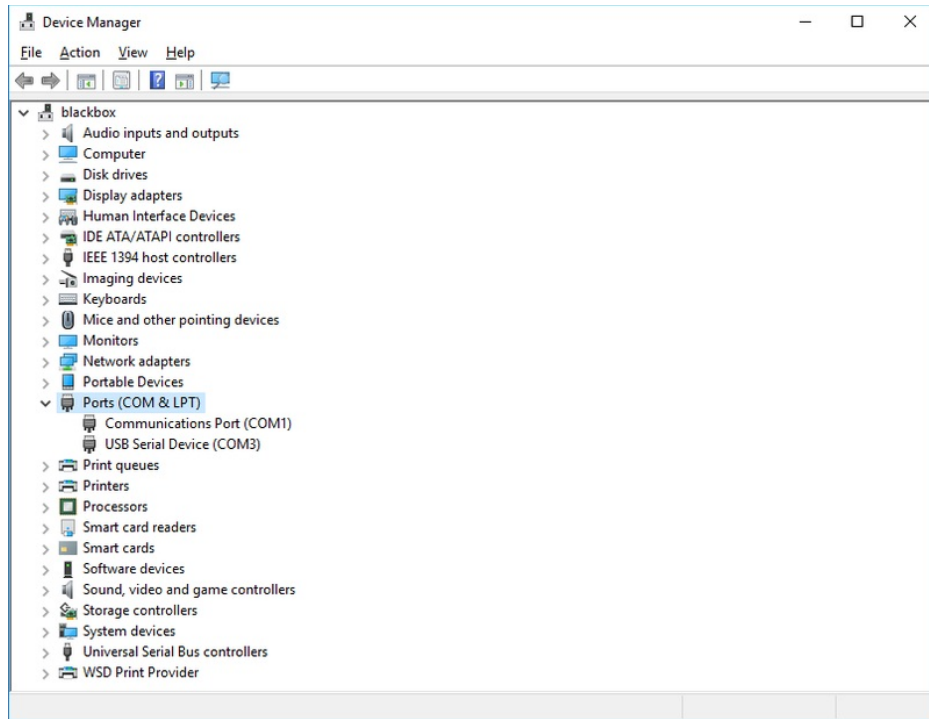
Sometimes the item will refer to the name of the board. Other times it may be called something like USB Serial Device, as seen in the image above. Either way, there is a new (COM#) following the name. This is the port your board is using.

Install Putty

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

The first thing to do is download the latest version of PuTTY (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.
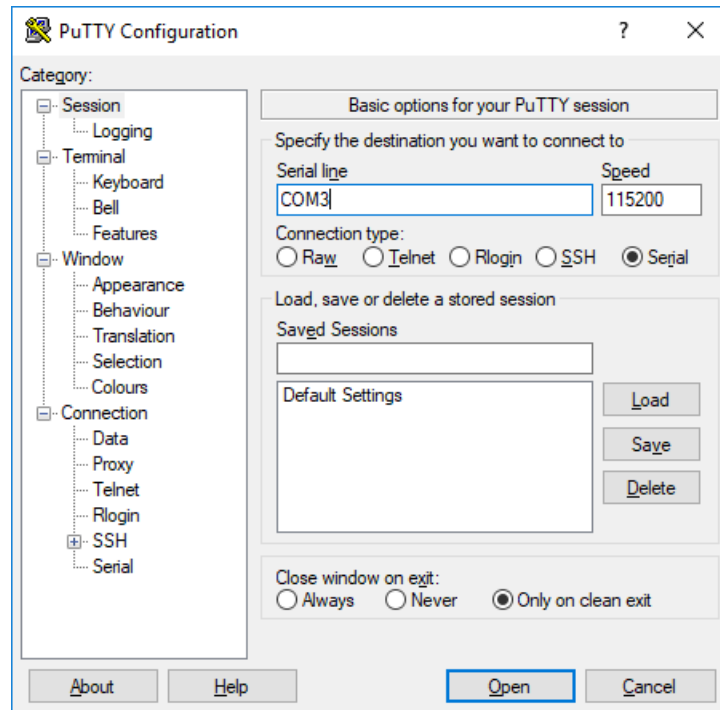
Once your settings are entered, you're ready to connect to the serial console. Click "Open" at the bottom of the window. A new window will open.
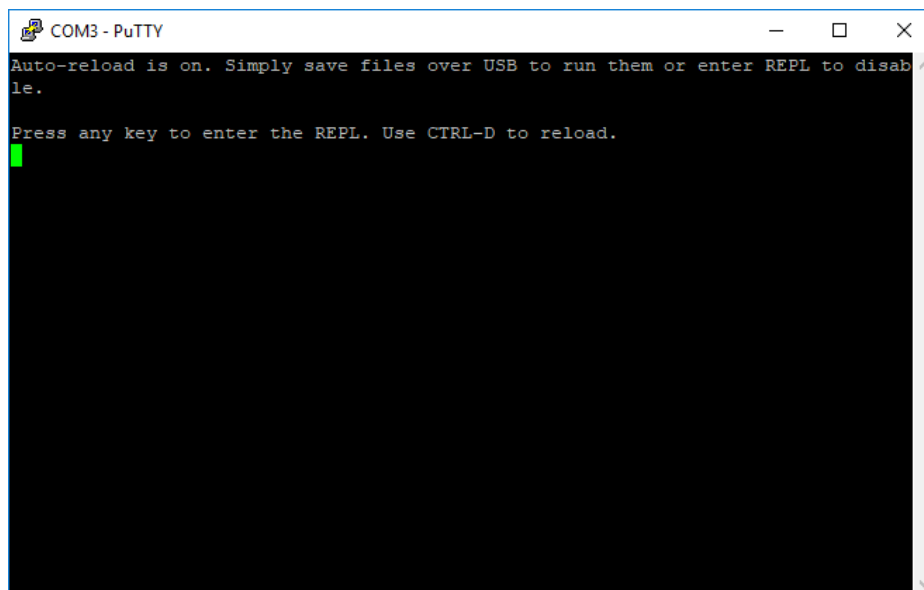
If no code is running, the window will either be blank or will look like the window above. Now you're ready to see the results of your code.

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 we continue to develop CircuitPython and create new releases, we will stop supporting older releases. If you are running CircuitPython 2.x, you need to update to 3.x. You must download the CircuitPython Library Bundle that matches your version of CircuitPython. Please update to CircuitPython 3.x and then download the 3.x bundle.

Always Run the Latest Version of CircuitPython and Libraries

As we continue to develop CircuitPython and create new releases, we will stop supporting older releases. **If you are running CircuitPython 2.x, you need to** update to 3.x (https://adafru.it/Amd)**.**

You need to download the CircuitPython Library Bundle that matches your version of CircuitPython. **Please update to CircuitPython 3.x and then** download the 3.x bundle (https://adafru.it/ABU).

We will soon stop providing the 2.x bundle as an automatically created download on the Adafruit CircuitPython Bundle repo. If you must continue to use 2.x, you can still download the 2.x version of `mpy-cross` from the 2.x release of CircuitPython on the CircuitPython repo and create your own 2.x compatible .mpy library files. **However, it is best to update to 3.x for both CircuitPython and the library bundle.**

CPLAYBOOT, TRINKETBOOT, FEATHERBOOT, or GEMMABOOT Drive Not Present

You may have a different board.

Only Adafruit Express boards and the Trinket M0 and Gemma M0 boards ship with the UF2 bootloader (https://adafru.it/zbX)installed. Feather M0 Basic, Feather M0 Adalogger, and similar boards use a regular Arduino-compatible bootloader, which does not show a *boardname*BOOT drive.

MakeCode

If you are running a MakeCode (https://adafru.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.

Windows 10

Did you install the Adafruit Windows Drivers package by mistake? 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

The latest version of the Adafruit Windows Drivers (version 2.0.0.0 or later) will fix the missing *boardname*BOOT drive problem on Windows 7. To resolve this, first uninstall the old versions of the drivers:

- Unplug any boards. In **Uninstall or Change a Program (Control Panel->Programs->Uninstall a program)**, uninstall everything named "Windows Driver Package - Adafruit Industries LLC ...".
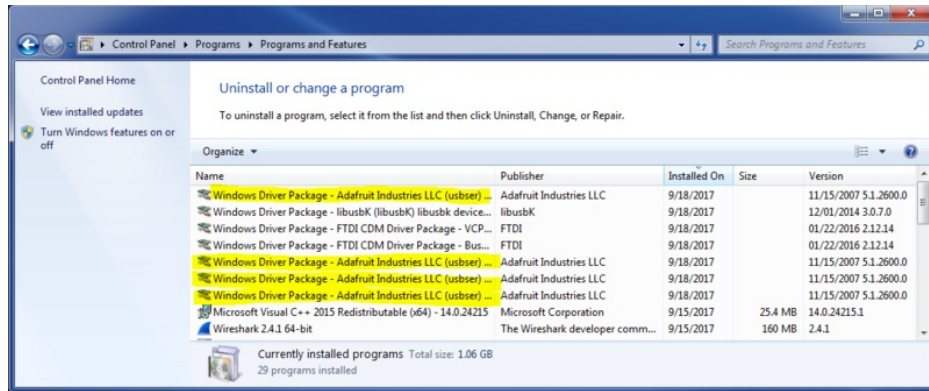
- Now install the new 2.3.0.0 (or higher) Adafruit Windows Drivers Package:

<div align="center">

https://adafru.it/AB0

https://adafru.it/AB0

</div>

- When running the installer, you'll be shown a list of drivers to choose from. You can check and uncheck the boxes to choose which drivers to install.
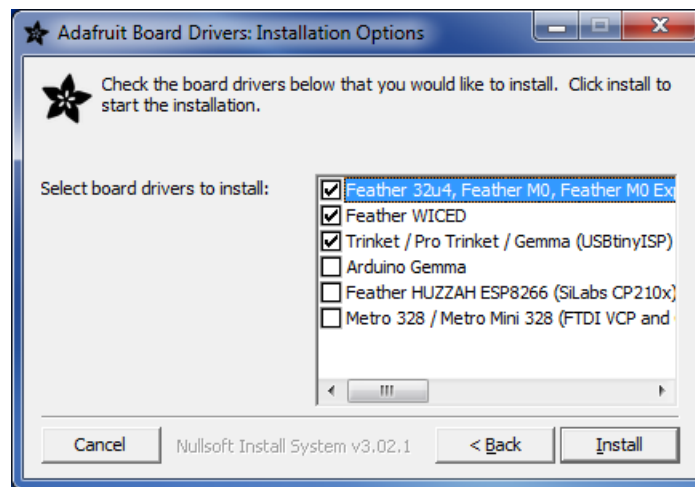
You should now be done! Test by unplugging and replugging the board. You should see the `CIRCUITPY` drive, and when you double-click the reset button (single click on Circuit Playground Express running MakeCode), you should see the appropriate `boardnameBOOT` drive.

Let us know in the Adafruit support forums (https://adafru.it/jIf) 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 we know of 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 thi.s

- **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.

CIRCUITPY Drive Does Not Appear

Kaspersky anti-virus can block the appearance of the `CIRCUITPY` drive. We haven't yet figured out a settings change 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.

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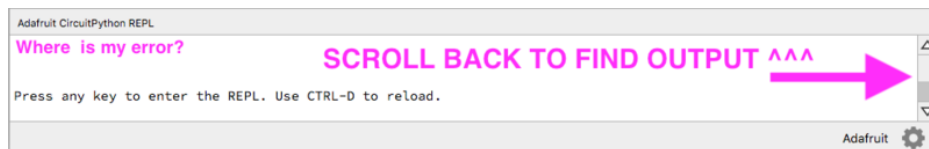
This applies to any kind of serial output whether it be error messages or print statements. So before you start trying to debug your problem on the hardware side, be sure to check that you haven't simply missed the serial messages due to serial output panel height.

CircuitPython RGB Status Light

The Feather M0 Express, Feather M4 Express, Metro M0 Express, Metro M4 Express, ItsyBitsy M0 Express, ItsyBitsy M4 Express, Gemma M0, and Trinket M0 all have a single NeoPixel or DotStar RGB LED on the board that indicates the

status of CircuitPython.

Circuit Playground Express does NOT have a status LED. The LEDs will pulse green when in the bootloader. They do NOT indicate any status while running CircuitPython.

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.

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 2.x and 3.x, as well as between 1.x and 2.x.

So, for instance, if you upgraded to CircuitPython 3.x from 2.x you'll need to download a newer version of the library that triggered the error on `import`. They are all available in the Adafruit bundle (https://adafru.it/y8E).

Make sure to download a version with 2.0.0 or higher in the filename if you're using CircuitPython version 2.2.4, and the version with 3.0.0 or higher in the filename if you're using CircuitPython version 3.0.

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.

First check - have you used Arduino to program your board? If so, CircuitPython is no longer able to provide the USB services. Reset the board so you get a *boardnameBOOT* drive rather than a `CIRCUITPY` drive, copy the latest version of CircuitPython (`.uf2`) back to the board, then Reset. This may restore `CIRCUITPY` functionality.

If still broken - 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.

In this situation, 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.

Easiest Way: Use `storage.erase_filesystem()`

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

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

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

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

Old Way: For the Circuit Playground Express, Feather M0 Express, and Metro M0 Express:

If you are running a version of CircuitPython before 2.3.0, and you don't want to upgrade, or you can't get to the REPL, you can do this.

1. Download the correct erase file:

> https://adafru.it/AdI

https://adafru.it/AdI

> https://adafru.it/AdJ

https://adafru.it/AdJ

> https://adafru.it/AdK

https://adafru.it/AdK

> https://adafru.it/EoM

https://adafru.it/EoM

> https://adafru.it/DjD

https://adafru.it/DjD

> https://adafru.it/DBA

https://adafru.it/DBA

2. Double-click the reset button on the board to bring up the *boardname*BOOT drive.

3. Drag the erase .uf2 file to the *boardname*BOOT drive.

4. The onboard NeoPixel will turn yellow or blue, indicating the erase has started.

5. After approximately 15 seconds, the mainboard NeoPixel 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 *boardname*BOOT drive.

7. Drag the appropriate latest release of CircuitPython (https://adafru.it/Amd) .uf2 file to the *boardname*BOOT 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). You'll also need to install your libraries and code!

Old Way: For Non-Express Boards with a UF2 bootloader (Gemma M0, Trinket M0):

If you are running a version of CircuitPython before 2.3.0, and you don't want to upgrade, or you can't get to the REPL, you can do this.

1. Download the erase file:

2. Double-click the reset button on the board to bring up the *boardname*BOOT drive.

3. Drag the erase .uf2 file to the *boardname*BOOT drive.

4. The boot LED will start flashing again, and the *boardname*BOOT drive will reappear.

5. Drag the appropriate latest release CircuitPython (https://adafru.it/Amd) .uf2 file to the *boardname*BOOT 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) You'll also need to install your libraries and code!

Old Way: For non-Express Boards without a UF2 bootloader (Feather M0 Basic Proto, Feather Adalogger, Arduino Zero):

If you are running a version of CircuitPython before 2.3.0, and you don't want to upgrade, or you can't get to the REPL, you can do this.

Just follow these directions to reload CircuitPython using bossac (https://adafru.it/Bed), which will erase and re-create CIRCUITPY .

Running Out of File Space on Non-Express 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 couple ways to free up space.

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. Its ~12KiB or so.

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.

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, we recommend that 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 we're counting bytes.

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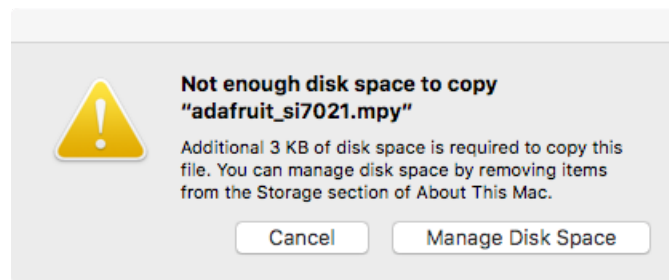
Luckily you can disable some of the extra hidden files that Mac OSX adds by running a few commands to disable search indexing and create zero byte placeholders. Follow the steps below to maximize the amount of space available on OSX:

Prevent & Remove Mac OSX 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) 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.

However there are still some cases where hidden files will be created by Mac OSX. In particular if you copy a file that was downloaded from the internet it will have special metadata that Mac OSX 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 Mac OSX Without Creating Hidden Files

Once you've disabled and removed hidden files with the above commands on Mac OSX 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 **foo.mpy** file to the board use a command like:

```
    cp -X foo.mpy /Volumes/CIRCUITPY
```
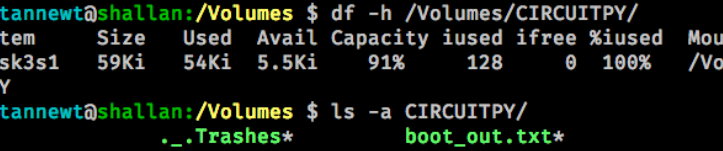
Or to copy a folder and all of its child files/folders use a command like:

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

Other Mac OSX 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 list the amount of space used on the CIRCUITPY drive with the **df** command:

Lets remove the `._` files first.

Whoa! We have 13Ki more than before! This space can now be used for libraries and code!

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 TrelllisM4 (https://adafru.it/CYr) library onto your board. Download the latest CircuitPython Library Bundle (https://adafru.it/uap) 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**

Check out the CircuitPython Libraries page of this guide (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.
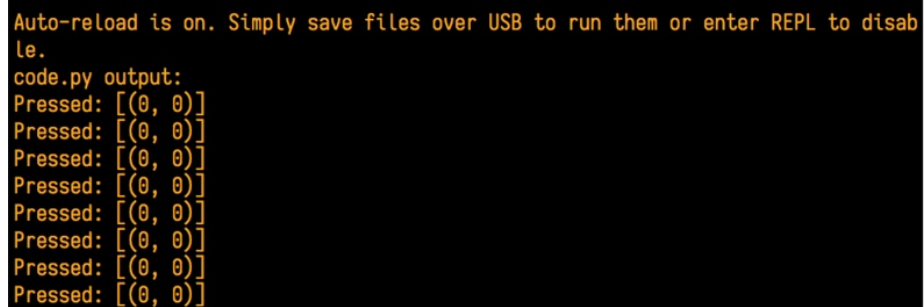
Let's take a look at a simple example. Copy the following code to the code.py file on your CIRCUITPY drive and save the file. Then connect to the serial console.
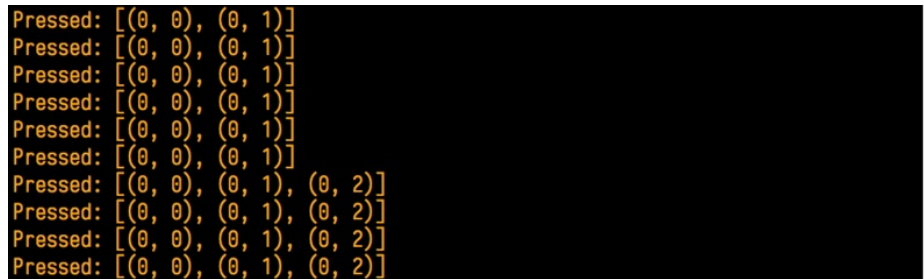
```
import adafruit_trellism4

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.

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

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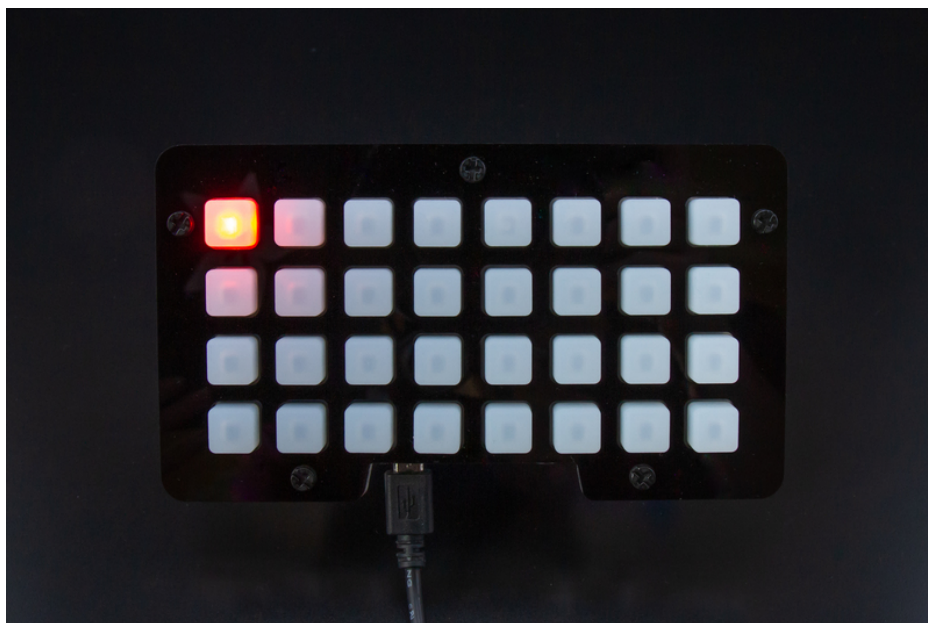
And another 90 degrees clockwise so the USB cord is pointing to the left, and update your code so `rotation=270` . Upper left is red!

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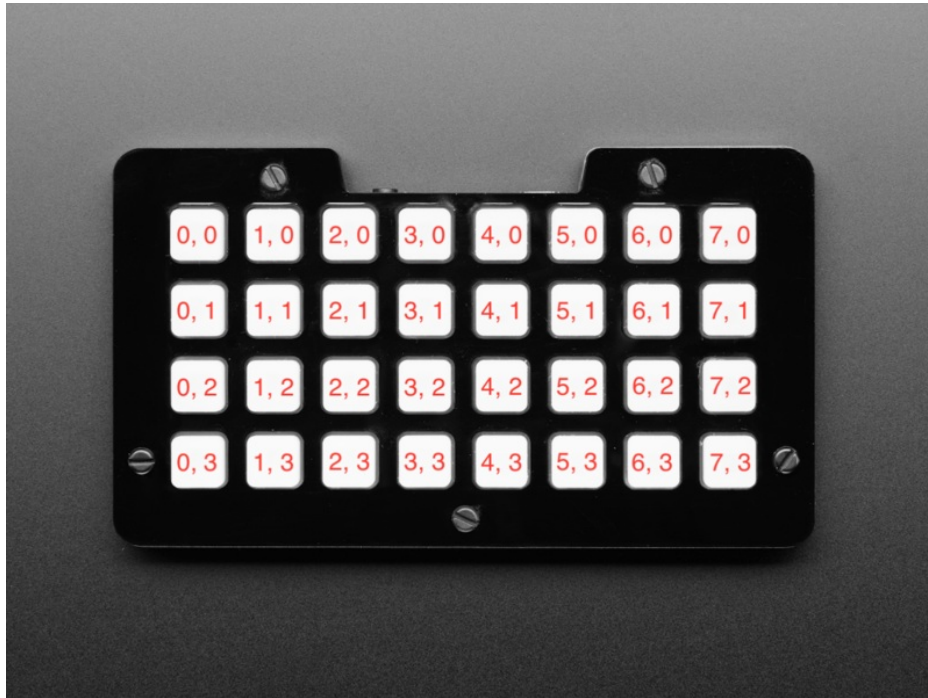
(0, 0) is always in the upper left corner and the rest of the grid follows standard positive (x, y) coordinates. So when the board is rotated 0 or 180 degrees, the board is oriented horizontally, and the lower right will be (7, 3).

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. Copy the following code to your code.py file and save it.

```
Temporarily unable to load content:
```

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.
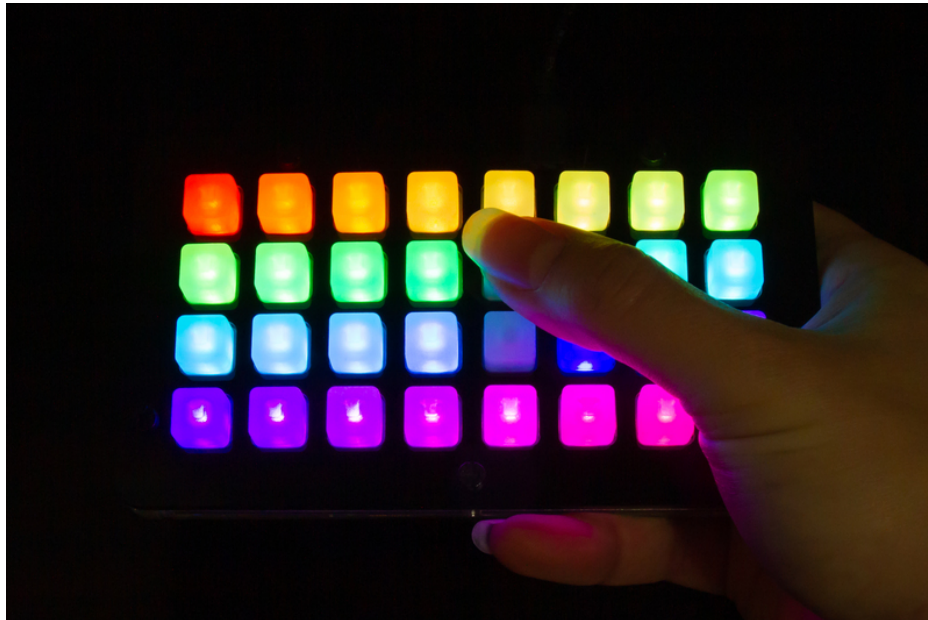
Imports and Setup

The first thing you need to do is import the library and then set it up for use. This is super simple. You being 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).

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 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!
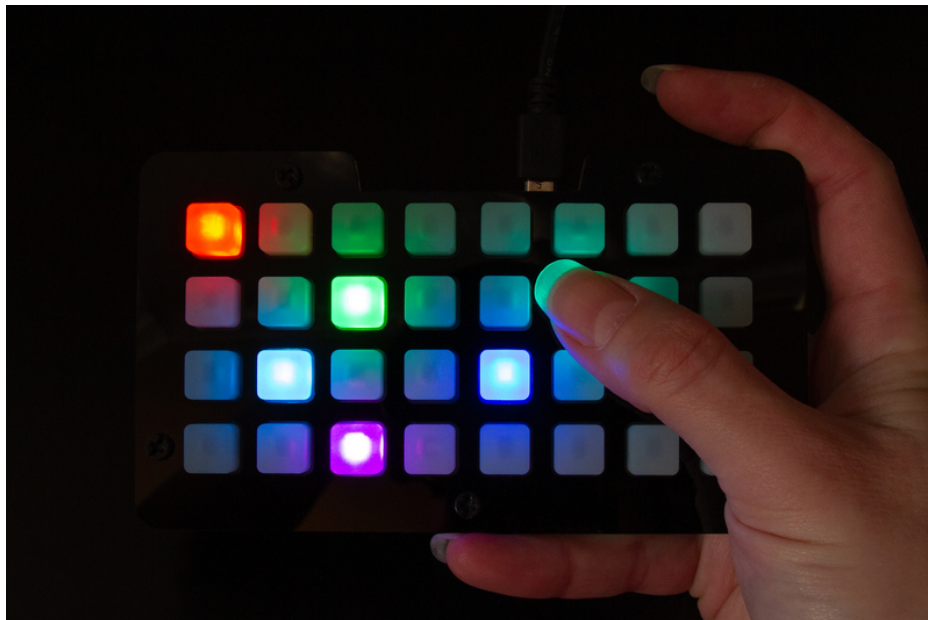
NeoPixels On: True or False?

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 it's 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. Copy it to **code.py** on your NeoTrellis M4 and start toggling!

```python
import adafruit_trellism4

trellis = adafruit_trellism4.TrellisM4Express()


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))


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] = 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

    current_press = pressed
```
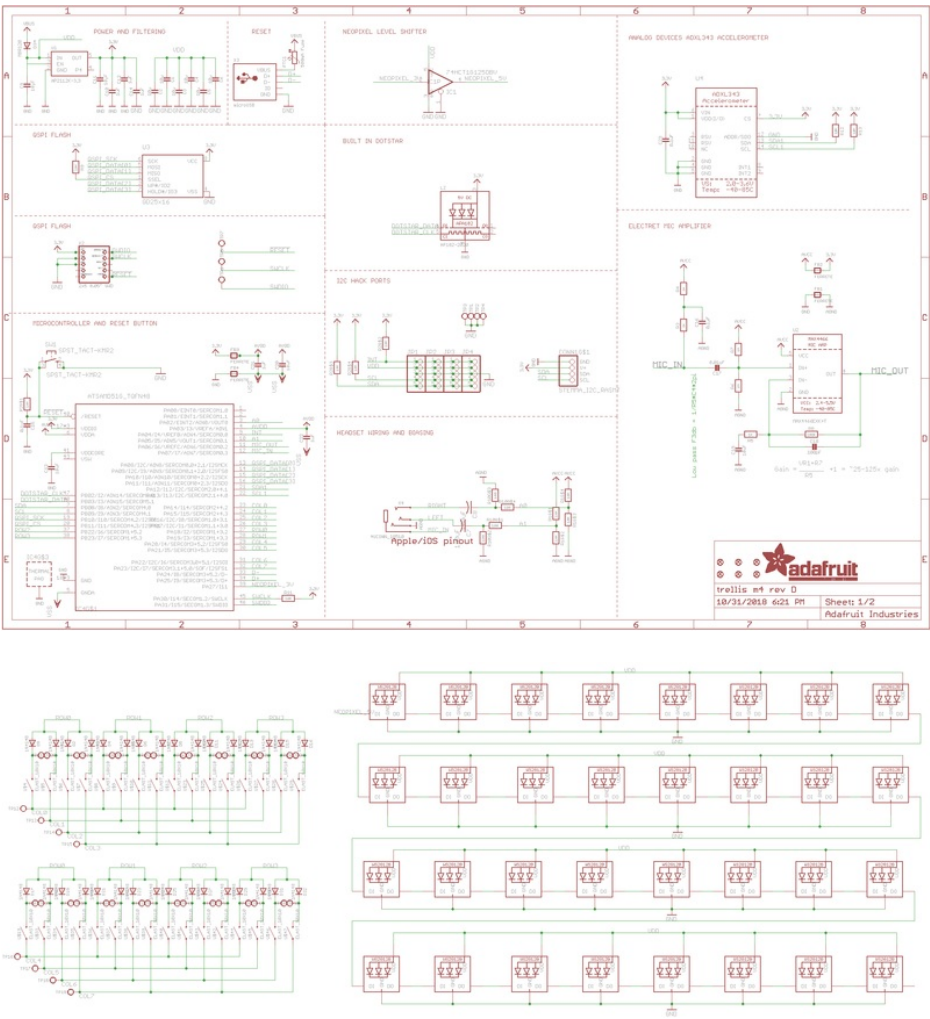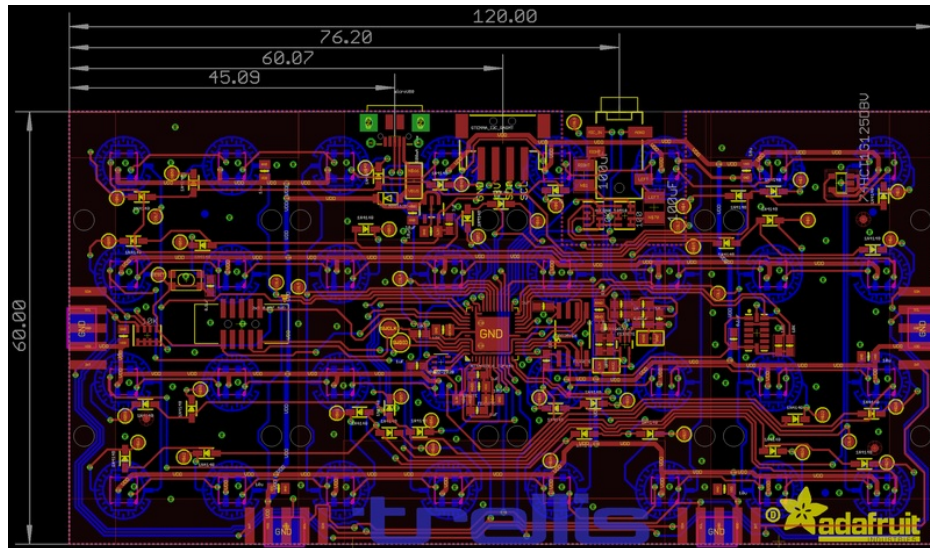
Downloads

Files

- PCB and Enclosure Files (https://adafru.it/D0i)

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 uf2tool, available here. (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), then check out the UF2 file format specification. (https://adafru.it/vPE)

Visit Adafruit's fork of the Microsoft UF2-samd bootloader GitHub repository (https://adafru.it/Beu) for source code and releases of pre-built bootloaders on CircuitPython.org (https://adafru.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.

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.
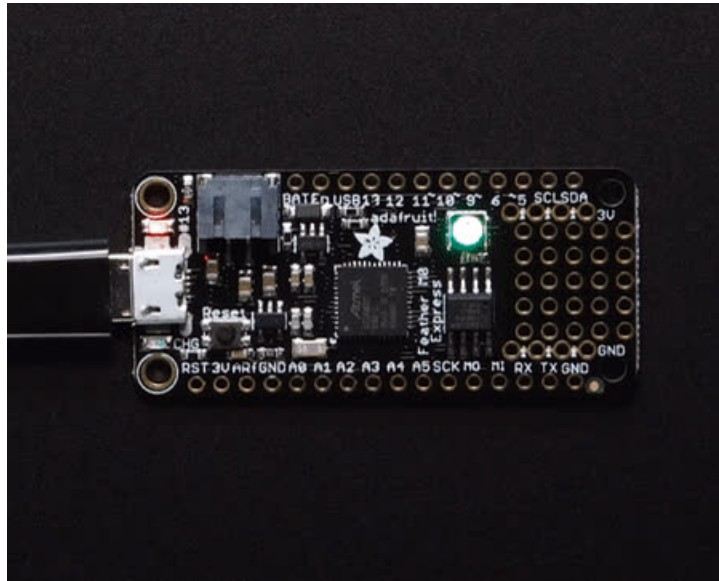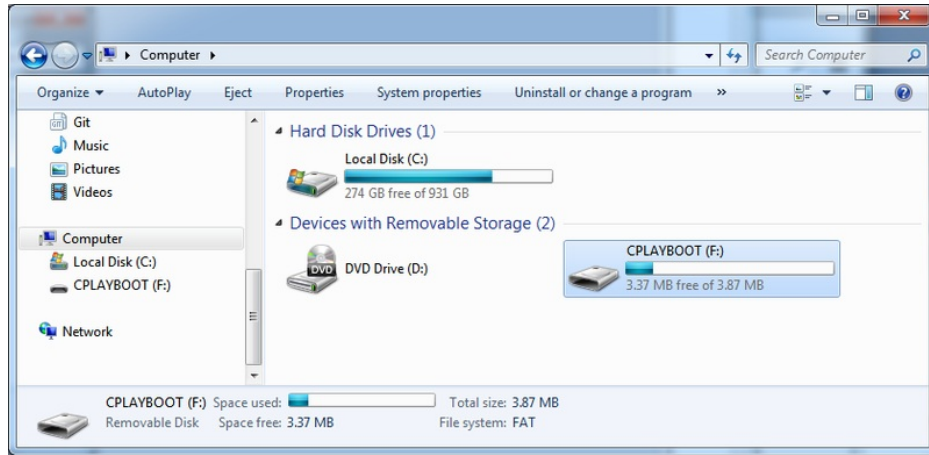
If the bootloader couldn't start, you will get a red NeoPixel LED.

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!
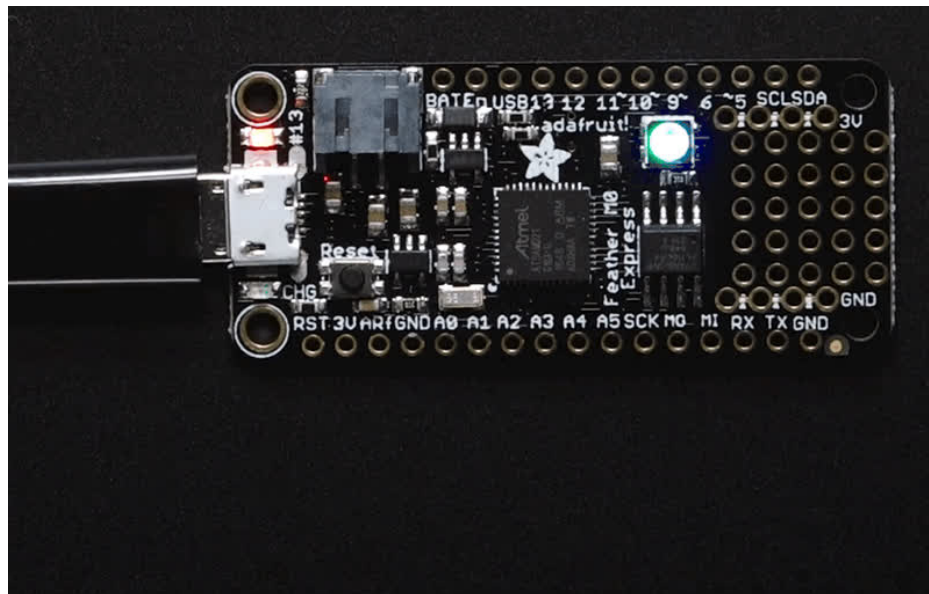

Once the bootloader is running, check your computer. You should see a USB Disk drive...

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.
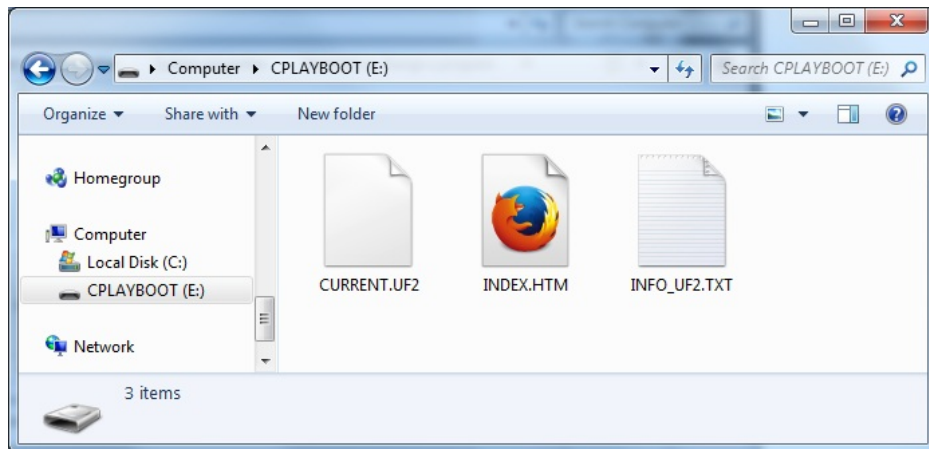

Using the Mass Storage Bootloader

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.
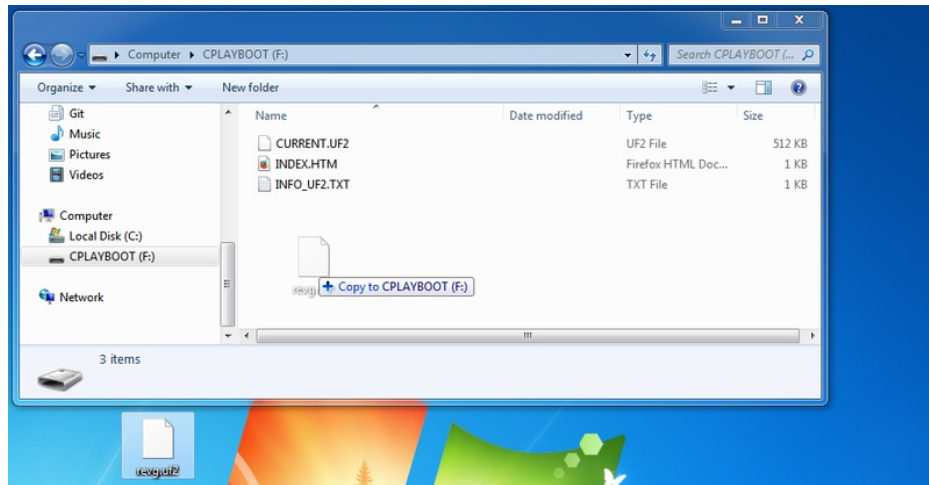
You may get an alert from the OS that the file is being copied without it's properties. You can just click **Yes**
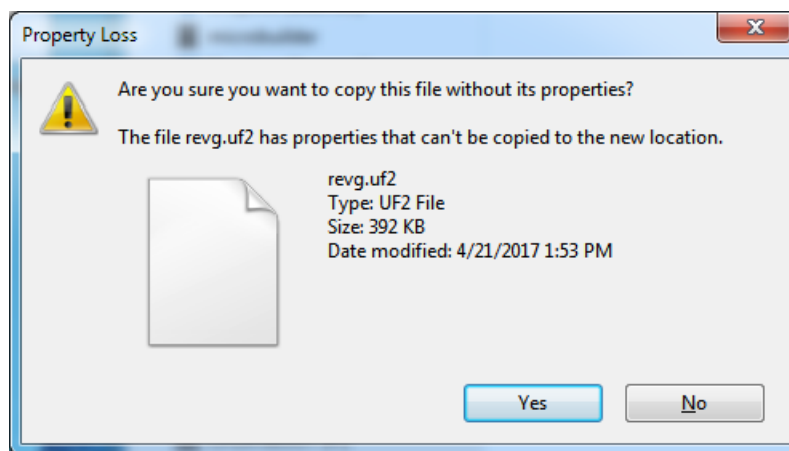
You may also get get a complaint that the drive was ejected without warning. Don't worry about this. The drive only ejects once the bootloader has verified and completed the process of writing the new code

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).

Windows 7 Drivers

If you are running Windows 7 (or, goodness, something earlier?) You will need a Serial Port driver file. Windows 10 users do not need this so skip this step.

You can download our full driver package here:

https://adafru.it/AB0

https://adafru.it/AB0

Download and run the installer. We recommend just selecting all the serial port drivers available (no harm to do so) and installing them.
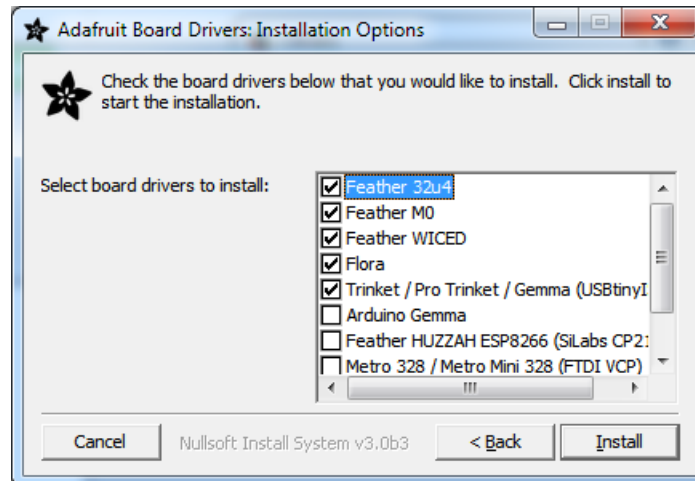
Verifying Serial Port in Device Manager

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!
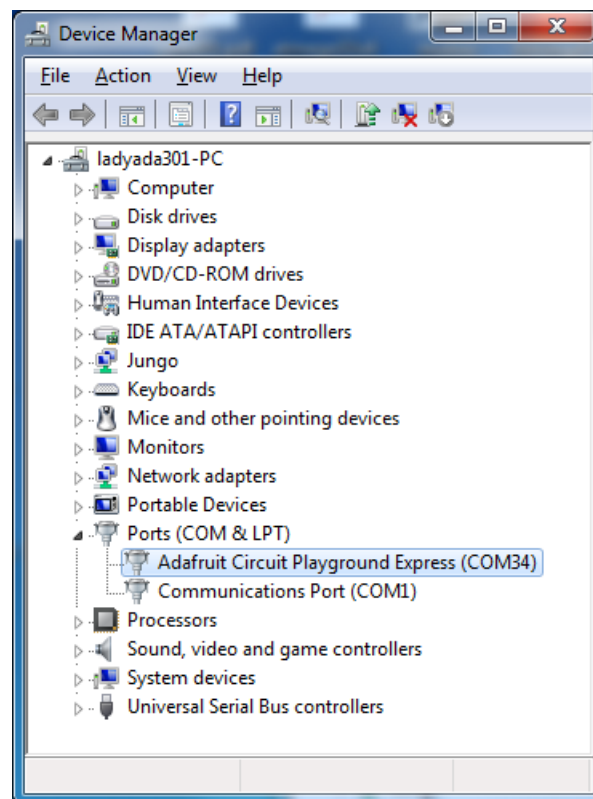
If you see something like this, it means you did not install the drivers. Go back and try again, then remove and re-plug the USB cable for your board

Running bossac on the command line

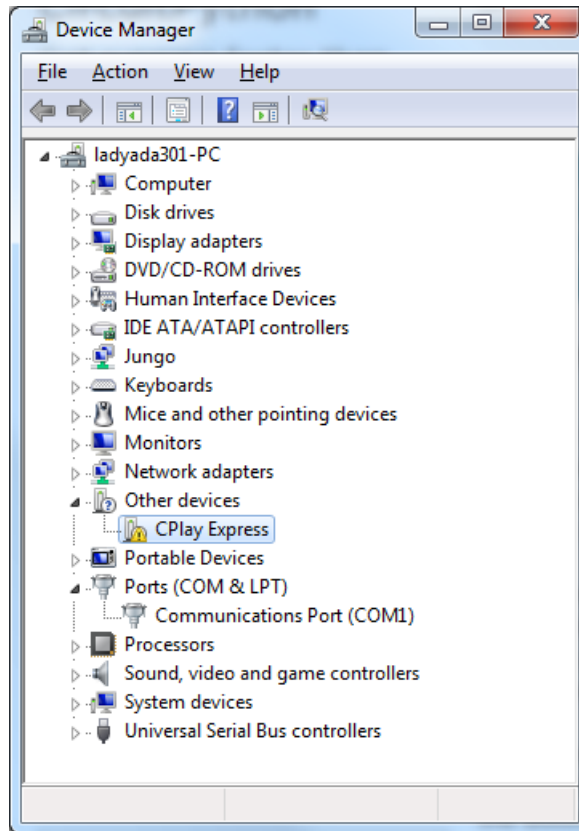If you are using the Arduino IDE, this step is not required. But sometimes you want to read/write custom binary files, say for loading CircuitPython or your own code. We recommend using bossac v 1.7.0 (or greater), which has been tested. The Arduino branch is most recommended (https://adafru.it/vQb).

You can download the latest builds here. (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
```
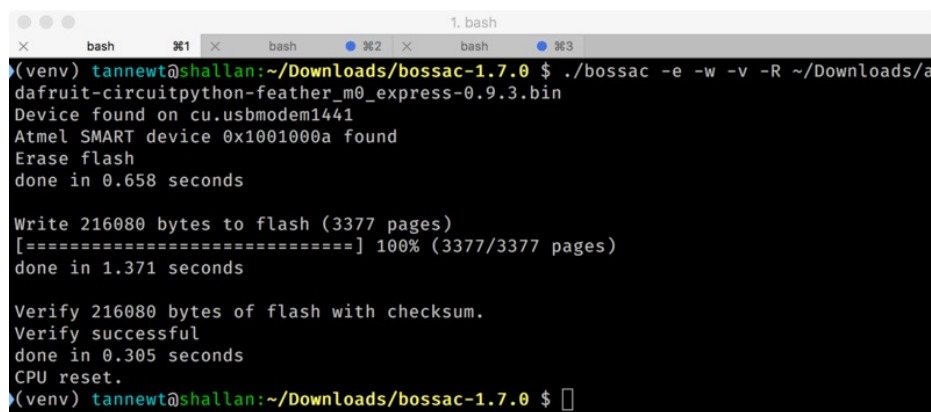
Using bossac Versions 1.9 or Later

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.

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).

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 SFHWRO
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).

Below are the latest updaters for various boards. The latest versions can always be found here (https://adafru.it/Bmg).
Look for the `update-bootloader...` files, not the `bootloader...` files.

<div style="background:#6eab43;color:white;">https://adafru.it/ECV</div>

https://adafru.it/ECV

<div style="background:#6eab43;color:white;">https://adafru.it/ECW</div>

https://adafru.it/ECW

<div style="background:#6eab43;color:white;">https://adafru.it/ECY</div>

https://adafru.it/ECY

<div style="background:#6eab43;color:white;">https://adafru.it/ED0</div>

https://adafru.it/ED0

<div style="background:#6eab43;color:white;">https://adafru.it/ED3</div>

https://adafru.it/ED3

<div style="background:#6eab43;color:white;">https://adafru.it/ED6</div>

https://adafru.it/ED6

<div style="background:#6eab43;color:white;">https://adafru.it/ED8</div>

https://adafru.it/ED8

<div style="background:#6eab43;color:white;">https://adafru.it/Bmg</div>
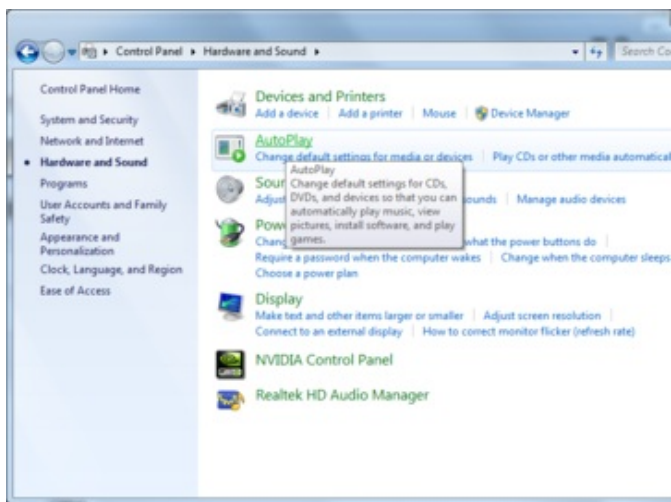
https://adafru.it/Bmg

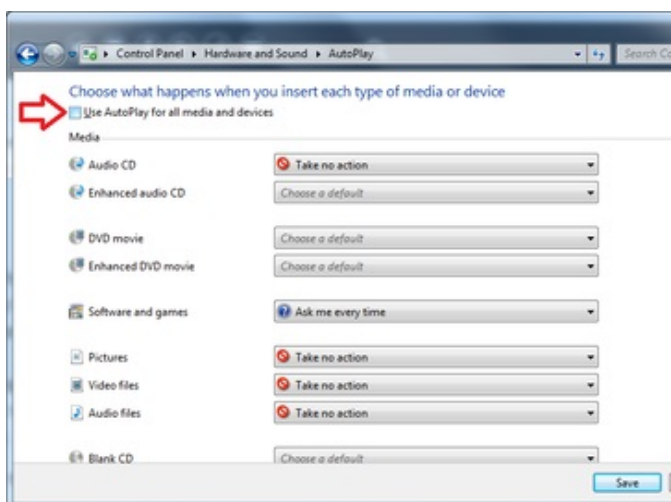## Getting Rid of Windows Pop-ups

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). 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) 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.bi

# 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 short writeup by turbinenreiter on how to do it for the Feather M4 (but adaptable to other boards) (https://adafru.it/ven)

Update Bootloader

> Trellises now come with a locked (protected) bootloader so you do not need to perform this step if you have an ADABOX Trellis!

On the very first batch of Neo Trellis M4 Express boards, the factory bootloader was not protected against accidental overwriting. Normally, the bootloader is unlikely to be damaged, but certain buggy programs and certain development environments might overwrite the bootloader. If you have one of these boards, we recommend you update the bootloader, which will protect it against overwriting as part of the updating process.

To see if your board needs updating, double-click the reset button, and look at the `INFO_UF2.TXT` file in `TRELM4BOOT`. If the bootloader version listed is earlier than " `UF2 Bootloader v2.0.0-adafruit.7`", you may want to update.

Download the bootloader updater below. Double-click the reset button and wait for the `TRELM4BOOT` drive to appear. Then drag or copy the updater `TRELM4BOOT`. Wait for `TRELM4BOOT` to reappear. Your bootloader is now protected against overwriting.

After updating, you'll need to reload CircuitPython or whatever other program was loaded on the board. Your `CIRCUITPY` drive is not affected by the updater on NeoTrellis M4 Express, because it's on a separate SPI flash chip.

> https://adafru.it/CYm

https://adafru.it/CYm

> Updating the bootloader means you will need to reinstall any older program/CircuitPython via
> https://learn.adafruit.com/adafruit-neotrellis-m4/circuitpython

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?
☐

When using the 16-step sequencer, if you press the audio sampler buttons, odd music will play

Why does the NeoTrellis sometimes 'hiccup' and restart after a few seconds?

After loading the 16-step sequencer, the MIDI arpeggiator or some other UF2 examples, my CIRCUITPY no longer appears!

A few seconds/minutes after playing a sound on the NeoTrellis there's a 'peeeeeeeeeewwwwwwww' sound - what's that?

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

When I press a button on the NeoTrellis a whole row is activated?