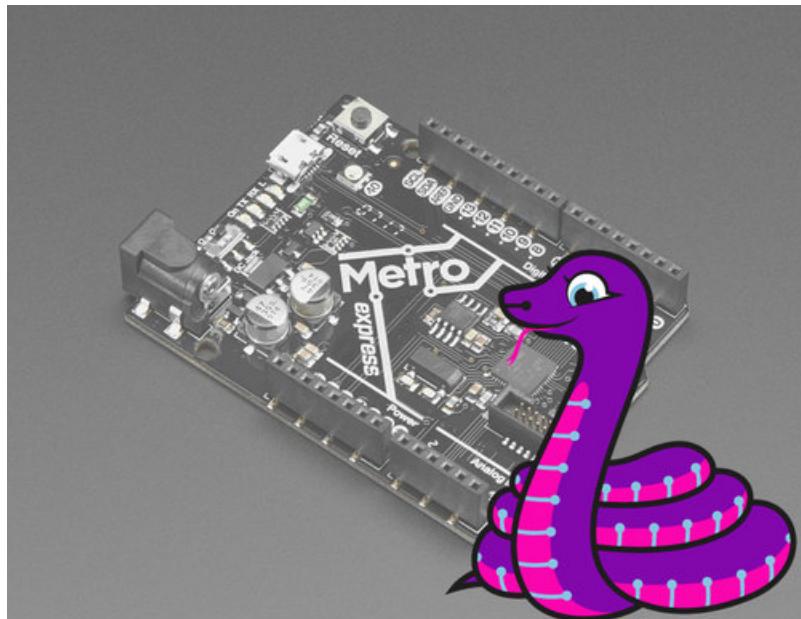




## Adafruit Metro M0 Express - Designed for CircuitPython

Created by lady ada



Last updated on 2018-01-04 11:40:15 PM UTC

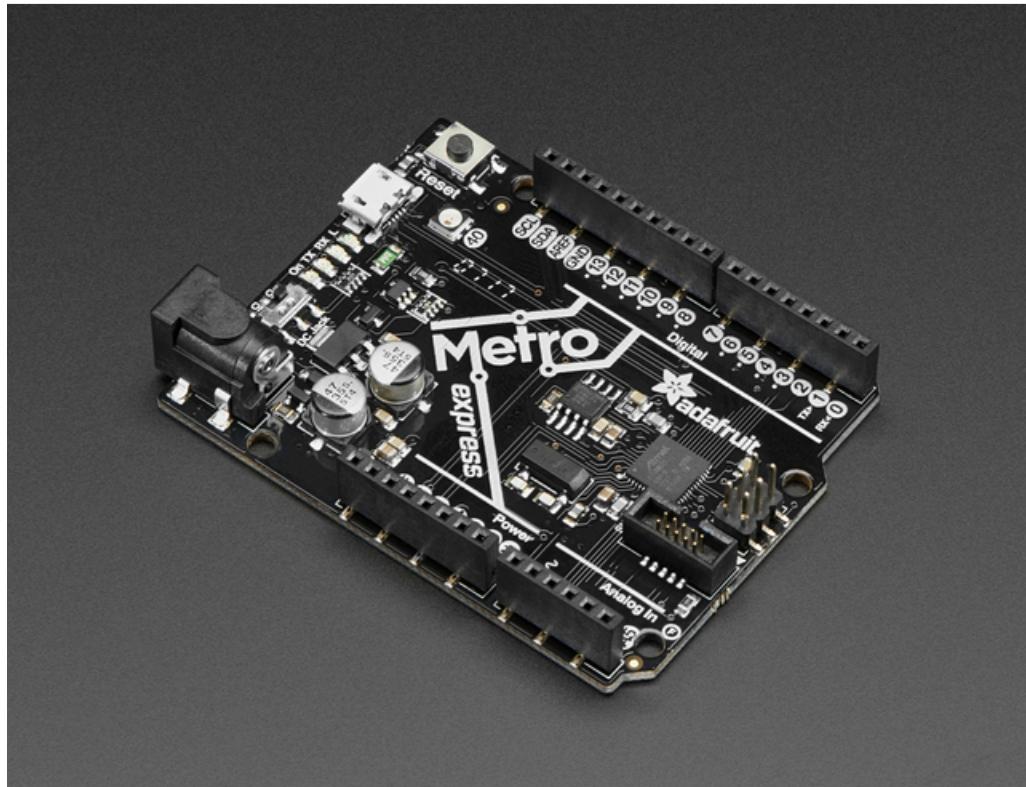
## Guide Contents

Guide Contents	2
Overview	5
Pinouts	9
Power Connections	9
Logic pins	10
Top Row	11
Bottom Row	11
Right side	11
Additional analog inputs	11
SPI Flash and NeoPixel	12
Other Pins!	13
Debug Interface	13
SEGGER J-Link EDU - JTAG/SWD Debugger	14
SEGGER J-Link BASE - JTAG/SWD Debugger	15
JTAG (2x10 2.54mm) to SWD (2x5 1.27mm) Cable Adapter Board	15
10-pin 2x5 Socket-Socket 1.27mm IDC (SWD) Cable - 150mm long	15
UF2 Bootloader Details	16
Entering Bootloader Mode	16
Using the Mass Storage Bootloader	18
Using the BOSSA Bootloader	19
Windows 7 Drivers	19
Verifying Serial Port in Device Manager	20
Running bossac on the command line	21
Updating the bootloader	22
Getting Rid of Windows Pop-ups	23
Making your own UF2	24
Arduino IDE Setup	25
<a href="https://adafruit.github.io/arduino-board-index/package_adafruit_index.json">https://adafruit.github.io/arduino-board-index/package_adafruit_index.json</a>	26
Using with Arduino IDE	28
Install SAMD Support	28
Install Adafruit SAMD	28
Install Drivers (Windows 7 Only)	29
Blink	31
Successful Upload	32
Compilation Issues	32
Manually bootloading	33
Ubuntu & Linux Issue Fix	33
Adapting Sketches to M0	34
Analog References	34
Pin Outputs & Pullups	34
Serial vs SerialUSB	34
AnalogWrite / PWM on Feather/Metro M0	35

analogWrite() PWM range	36
Missing header files	36
Bootloader Launching	36
Aligned Memory Access	36
Floating Point Conversion	37
How Much RAM Available?	37
Storing data in FLASH	37
Using SPI Flash	38
Read & Write CircuitPython Files	38
Format Flash Memory	40
Datalogging Example	41
Reading and Printing Files	42
Full Usage Example	42
Accessing SPI Flash	43
Metro M0 HELP!	45
My Metro M0 stopped working when I unplugged the USB!	45
My Metro never shows up as a COM or Serial port in the Arduino IDE	45
Ack! I "did something" and now when I plug in the Metro, it doesn't show up as a device anymore so I can't upload to it or fix it...	45
I can't get the Metro USB device to show up - I get "USB Device Malfunctioning" errors!	45
I'm having problems with COM ports and my Metro M0	45
I don't understand why the COM port disappears, this does not happen on my Arduino UNO!	46
I'm trying to upload to my 32u4, getting "avrdude: butterfly_recv(): programmer is not responding" errors	46
I'm trying to upload to my Metro M0, and I get this error "Connecting to programmer: avrdude: butterfly_recv(): programmer is not responding"	46
I'm trying to upload to my Metro and i get this error "avrdude: ser_recv(): programmer is not responding"	46
CircuitPython Setup	47
Downloading	47
Flashing	48
Flashing UF2	48
Flashing with BOSSAC	50
After flashing	51
Welcome to the Community!	52
Adafruit Discord	52
Adafruit Forums	53
Adafruit Github	54
ReadTheDocs	55
CircuitPython Blinky	56
code.py	56
Status LED (Gemma/Trinket/Metro/Feather)	56
Debugging	57
Libraries	57
More info	57
Connecting to the Serial Console	58
Are you using Mu?	58

Using Something Else?	59
Interacting with the Serial Console	60
The REPL	63
Returning to the serial console	66
CircuitPython Libraries	68
Installing the CircuitPython Library Bundle	68
Express Boards	69
Non-Express Boards	70
Example: ImportError Due to Missing Library	70
Library Install on Non-Express Boards	71
Updating CircuitPython Libraries	71
CircuitPython Built-Ins	72
Things that are Built In and Work	72
flow control	72
math	72
tuples, lists, arrays, and dictionaries	72
classes/objects and functions	72
lambdas	72
Things to watch out for!	72
Troubleshooting	74
CPLAYBOOT, TRINKETBOOT, FEATHERBOOT, or GEMMABOOT Drive Not Present	74
You may have a different board.	74
MakeCode	74
Windows 10	74
Windows 7	74
CircuitPython RGB Status Light	75
CIRCUITPY Drive Issues	76
For the Circuit Playground Express, Feather M0 Express, and Metro M0 Express:	76
For the Gemma M0, Trinket M0, Feather M0: Basic (Proto) and Feather Adalogger:	76
Running Out of File Space on Non-Express Boards	77
Delete something!	77
Use tabs	77
Mac OSX loves to add extra files.	77
Prevent & Remove Mac OSX Hidden Files	77
Copy Files on Mac OSX Without Creating Hidden Files	78
Other Mac OSX Space-Saving Tips	78
Downloads	80
Files	80
Schematic & Fabrication Print	80

## Overview



Metro is our series of microcontroller boards for use with the Arduino IDE. This new Metro board looks a whole lot like our [original Metro 328](#), but with a huge upgrade. Instead of the ATmega328, this Metro features a ATSAMD21G18 chip, an ARM Cortex M0+. It's our first Metro that is designed for use with CircuitPython! CircuitPython is our beginner-oriented flavor of MicroPython - and as the name hints at, it's a small but full-featured version of the popular Python programming language specifically for use with circuitry and electronics.

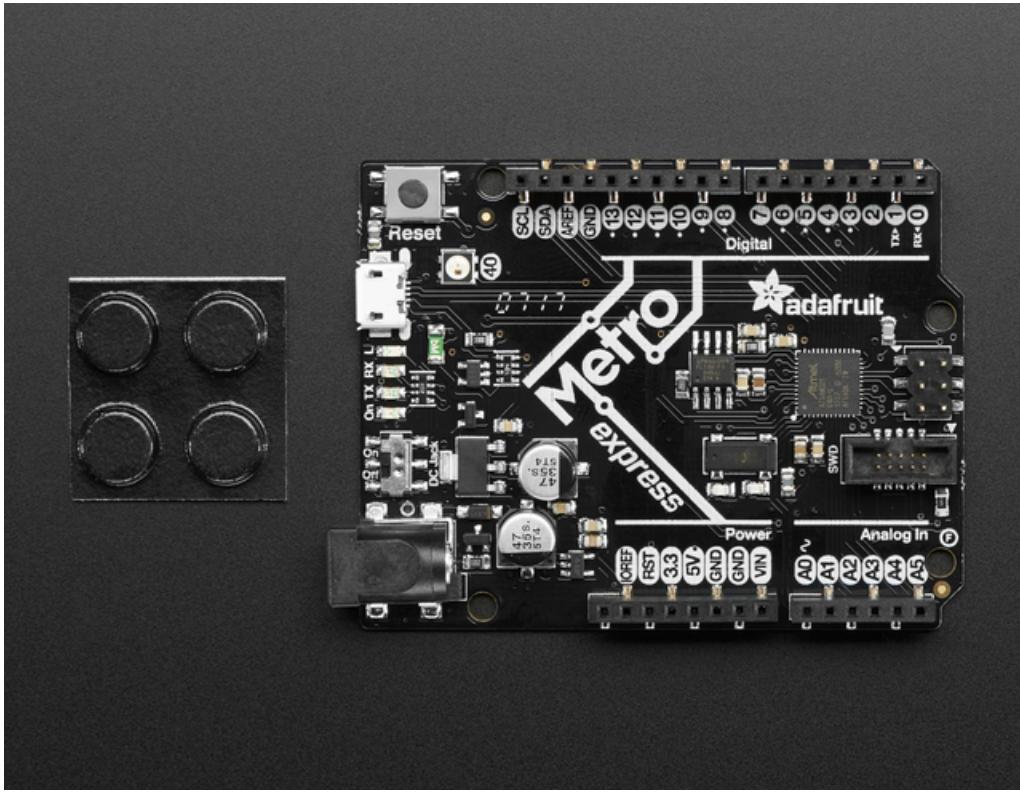
Not only can you use CircuitPython, but the Metro M0 is also usable in the Arduino IDE.



At the Metro M0's heart is an ATSAMD21G18 ARM Cortex M0 processor, clocked at 48 MHz and at 3.3V logic, the same one used in the new [Arduino Zero](#). This chip has a whopping 256K of FLASH (8x more than the Atmega328) and 32K of RAM (16x as much)! This chip comes with built in USB so it has USB-to-Serial program & debug capability built in with no need for an FTDI-like chip.

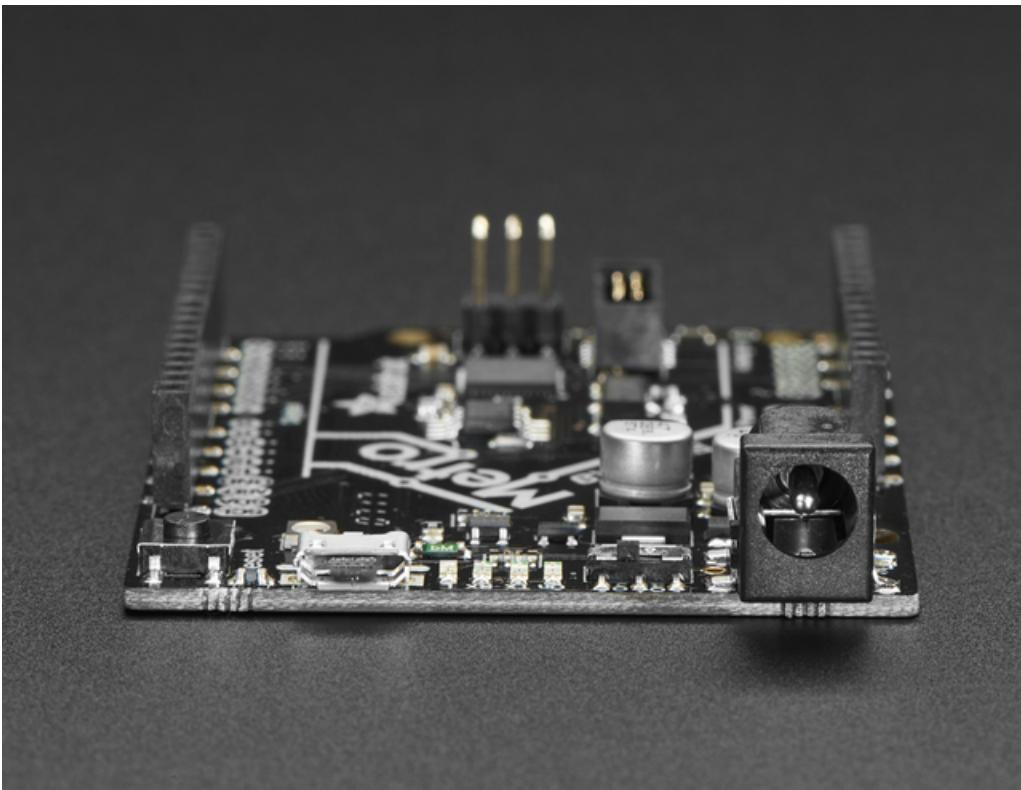


- **Power the METRO** with 7-9V polarity protected DC or the micro USB connector to any 5V USB source. The 2.1mm DC jack has an on/off switch next to it so you can turn off your setup easily. The METRO will automagically switch between USB and DC.
- **METRO has 25 GPIO pins**, 12 of which are analog in, and one of which is a true analog out. There's a hardware SPI port, hardware I2C port and hardware UART. Logic level is 3.3V
- **Native USB**, there's no need for a hardware USB to Serial converter as the Metro M0 has built in USB support. When used to act like a serial device, the USB interface can be used by any computer to listen/send data to the METRO, and can also be used to launch and update code via the bootloader. It can also act like a keyboard, mouse or MIDI device as well.
- **Four indicator LEDs and one NeoPixel**, on the front edge of the PCB, for easy debugging. One green power LED, two RX/TX LEDs for data being sent over USB, and a red LED connected. Next to the reset button there is an RGB NeoPixel that can be used for any purpose.
- **2 MB SPI Flash** storage chip is included on board. You can use the SPI Flash storage like a very tiny hard drive. When used in Circuit Python, the 2 MB flash acts as storage for all your scripts, libraries and files. When used in Arduino, you can read/write files to it, like a little datalogger or SD card, and then with our helper program, access the files over USB.
- **Easy reprogramming**, comes pre-loaded with the [UF2 bootloader](#), which looks like a USB key. Simply drag firmware on to program, no special tools or drivers needed! It can be used by MakeCode or Arduino IDE (in bossa compatibility)



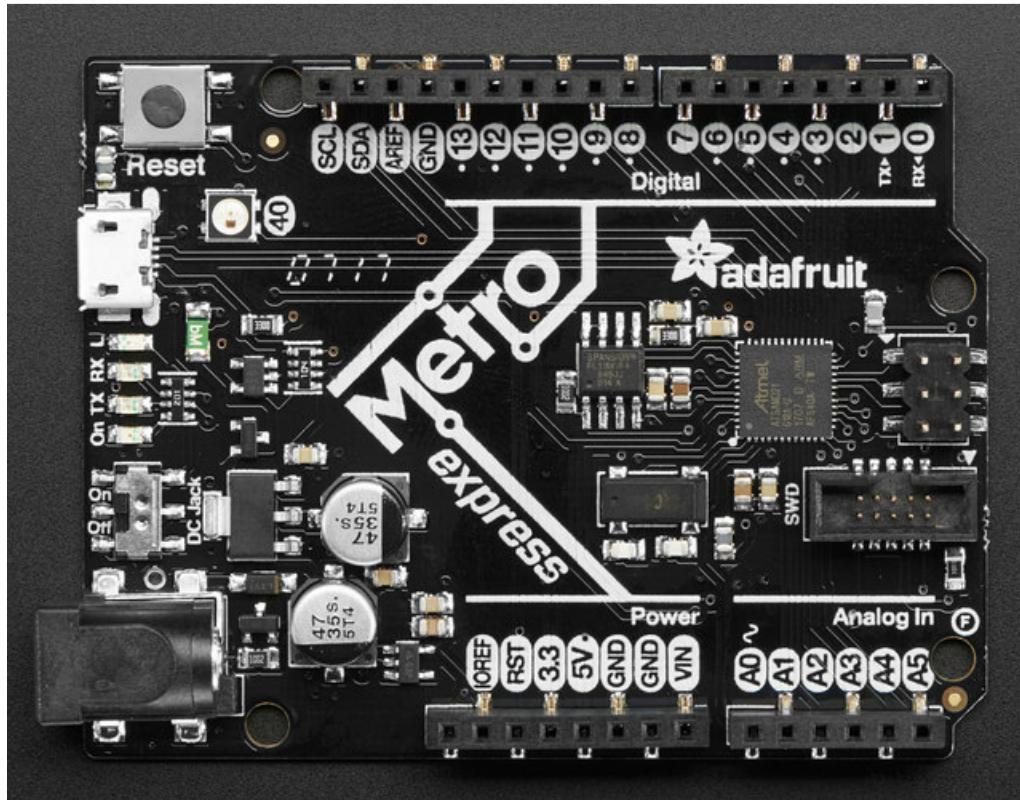
Here's some handy specs!

- Measures 2.8" x 2.1" x 0.28"
- ATSAMD21G18 @ 48MHz with 3.3V logic/power
- 256KB of FLASH + 32KB of RAM
- 4 MB SPI Flash chip
- No EEPROM
- 32.768 KHz crystal for clock generation & RTC
- 3.3V regulator with 500mA peak current output
- USB native support, comes with USB bootloader and serial port debugging
- You also get tons of pins - 25 GPIO pins, 5 more than the Metro 328
- Hardware Serial, hardware I2C, hardware SPI support
- PWM outputs on almost all pins
- 6 x 12-bit analog inputs
- 1 x 10-bit analog output (DAC)
- Built in NeoPixel on pin #40
- Pin #13 red LED for general purpose blinking
- Power on/off switch
- 4 mounting holes
- We also include 4 rubber bumpers to keep it from slipping off your desk
- Reset button



## Pinouts

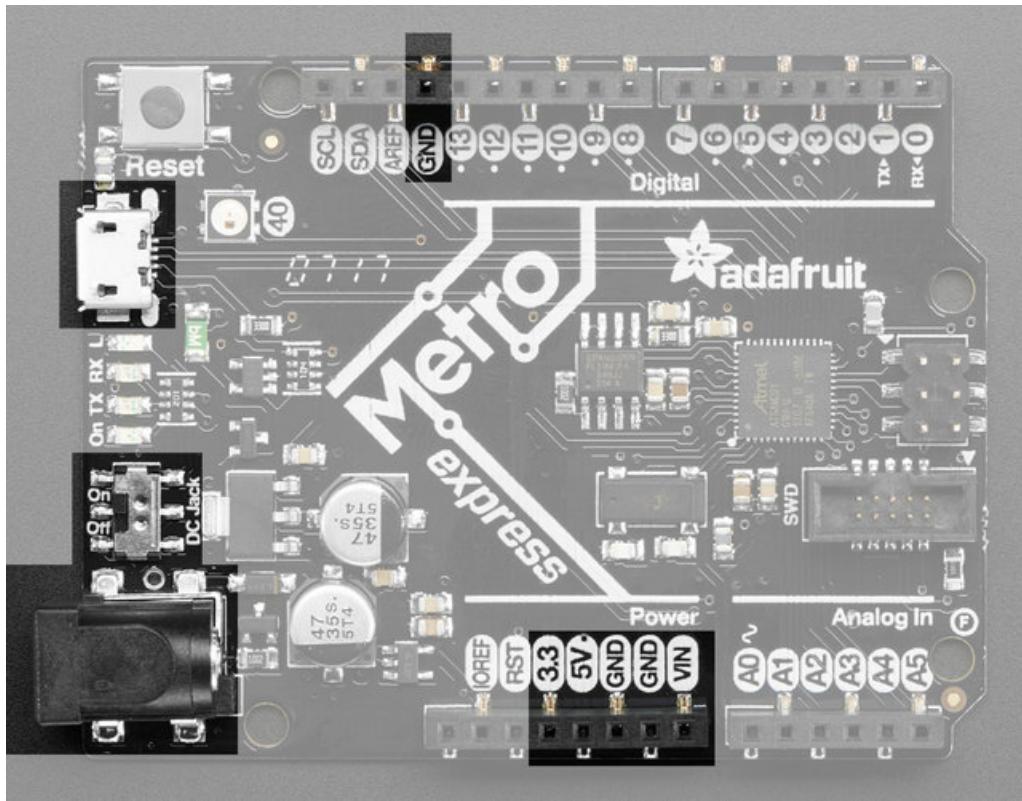
---



The Metro M0 is chock-full of microcontroller goodness. There's also a lot of pins and ports. We'll take you a tour of them now!

## Power Connections

---



There's a lot of ways to power the Metro M0 Express, and a lot of ways to *get* power out as well.

There are two primary ways to power the Metro:

- Through the Micro USB port up at the top left
- Through the DC jack at the bottom left

The MicroUSB jack provides 5V at 500mA or so, there is a fuse that will shut off temporarily when more than 1000mA is drawn, this is to protect a computer USB port. You can plug this into any computer or USB charger with a USB cable. You can draw up to 500mA between the Vin, 5V and 3.3V supplies (combined).

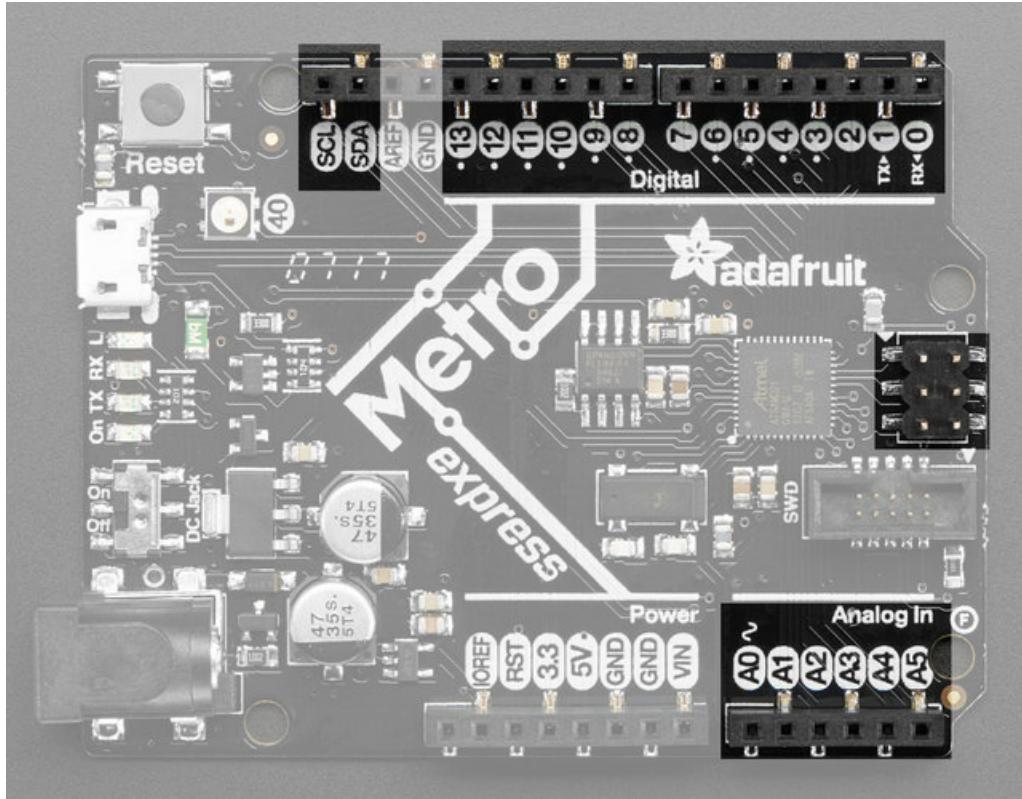
The DC Jack is a 5.5mm/2.1mm center-positive DC connector, which is the most common available. Provide about 6V-12V here to power the Metro. There is no fuse on this connection so you can draw more current, up to 800mA between the **5V** and **3.3V** supplies, and 2A from **Vin**.

Onboard regulators take the USB or DC power and linearly convert it to **3.3V** and **5V**:

- **3V** - this is the output from the 3.3V regulator, it can supply 500mA peak
- **5V** - this is the output from the 5V regulator (when DC jack is used), or from USB. It can supply ~500mA peak from USB and ~800mA peak from DC
- **GND** - this is the common ground for all power and logic
- **Vin** - this is the *higher* of the DC jack or USB voltage. So if the DC jack is plugged in and 9V, Vin is 9V. If only USB connected, this will be 5V.

There is also an on/off switch. This switch is only for the DC jack and does not affect powering via USB

## Logic pins



This is the general purpose I/O pin set for the microcontroller.

All logic is 3.3V

Most pins can do PWM output

All pins can be interrupt inputs

### Top Row

- #0 / RX - GPIO #0, also receive (input) pin for **Serial1** (hardware UART)
- #1 / TX - GPIO #1, also transmit (output) pin for **Serial1**
- #2 through #12 - These are general purpose GPIO. If there's a dot next to the pad it can act as a PWM output.
- #13 - GPIO #13 and is connected to the **red LED** marked L next to the USB jack
- **SDA** - the I2C (Wire) data pin. There's no pull up on this pin by default so when using with I2C, you may need a 2.2K-10K pullup.
- **SCL** - the I2C (Wire) clock pin. There's no pull up on this pin by default so when using with I2C, you may need a 2.2K-10K pullup.

### Bottom Row

- **A0** - This pin is analog *input A0* but is also an analog *output* due to having a DAC (digital-to-analog converter). You can set the raw voltage to anything from 0 to 3.3V, unlike PWM outputs this is a true analog output
- **A1 thru A5** - These are each analog input as well as digital I/O pins.

### Right side

- **SCK/MOSI/MISO** - These are the hardware SPI pins, are connected to the 2x3 header on the right hand side. you can use them as everyday GPIO pins (but recommend keeping them free as they are best used for hardware SPI connections for high speed.)

### Additional analog inputs

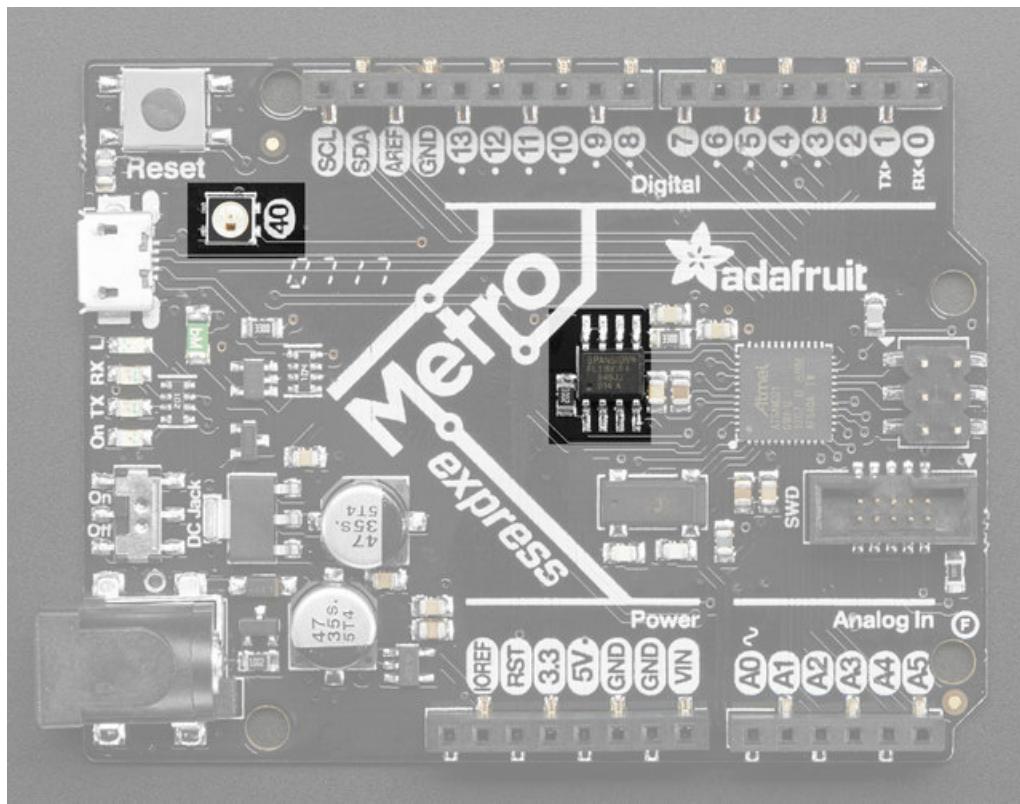
In addition to the A0-A5 pins, there are extra analog inputs available

- Digital #0 is also A6
- Digital #1 is also A7
- Digital #4 is also A8
- Digital #5 is also A9
- Digital #8 is also A10
- Digital #9 is also A11

These pins are available in CircuitPython under the `board` module. Names that start with # are prefixed with D and other names are as is. So `#0 / RX` above is available as `board.D0` and `board.RX` for example.

## SPI Flash and NeoPixel

As part of the 'Express' series of boards, the Metro M0 Express is designed for use with CircuitPython. To make that easy, we have added two extra parts to this Metro M0: a mini NeoPixel (RGB LED) and a 2 MB SPI Flash chip



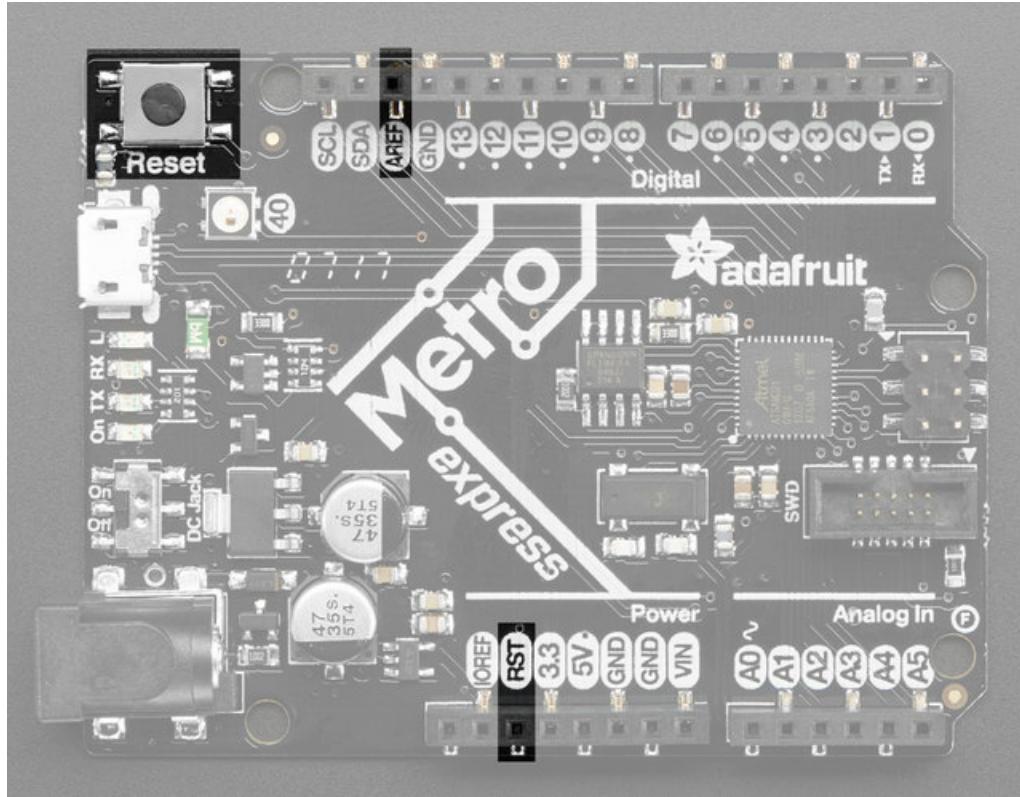
The **NeoPixel** is connected to pin #40 in Arduino, so [just use our NeoPixel library](#) and set it up as a single-LED strand on pin 40. In CircuitPython, the NeoPixel is `board.NEOPIXEL` and the library for it is [here](#) and in [the bundle](#). The NeoPixel is powered by the 3.3V power supply but that hasn't shown to make a big difference in brightness or color. The NeoPixel is also used by the bootloader to let you know if the device has enumerated correctly (green) or USB failure (red). In CircuitPython, the LED is used to indicate the runtime status.

The SPI Flash is connected to 4 pins that are not brought out on the GPIO pads. This way you don't have to worry about the SPI flash colliding with other devices on the main SPI connection. Under Arduino, the **FLASH SCK** pin is #38, **MISO** is #36, **MOSI** is #37, and **CS** is #39. If you use **Metro M0 Express** as your board type, you'll be able to access the Flash SPI port under **SPI1** - this is a fully new hardware SPI device separate from the GPIO pins on the outside edge of the Feather. In CircuitPython, the SPI flash is used natively by the interpreter and is read-only to user code, instead the

Flash just shows up as the writeable disk drive!

## Other Pins!

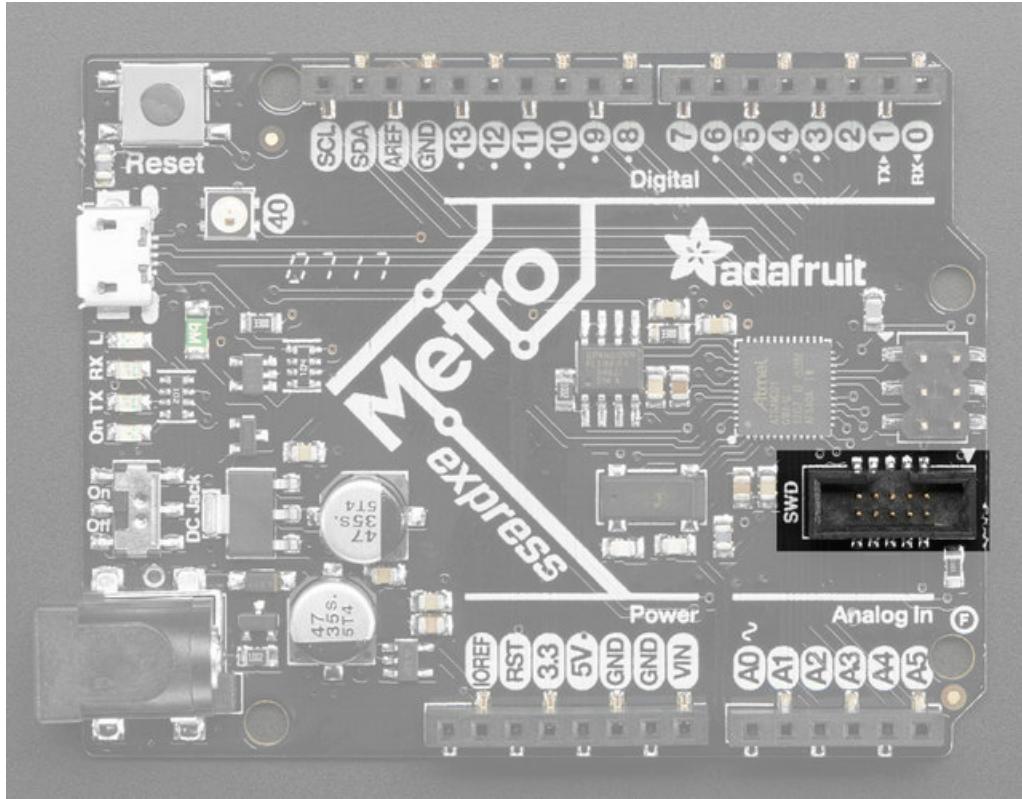
---



- **RST** - this is the Reset pin, tie to ground to manually reset the ATSAMD21, as well as launch the bootloader manually
- **ARef** - the analog reference pin. Normally the reference voltage is the same as the chip logic voltage (3.3V) but if you need an alternative analog reference, connect it to this pin and select the external AREF in your firmware.  
Can't go higher than 3.3V!

## Debug Interface

---



If you'd like to do more advanced development, trace-debugging, or not use the bootloader, we have the SWD interface exposed.

You can use any 2x5 0.05" pitch SWD interface to connect. We suggest a J-Link. Since the SWCLK pin is shared between the NeoPixel, and the bootloader takes control of the pin, you need to reset the board *right before* beginning debug. OpenOCD and some other debug interfaces may not be able to do this. That's why we really really suggest a JLink!



SEGGER J-Link EDU - JTAG/SWD Debugger

PRODUCT ID: 1369

<https://adafru.it/e9G>

\$69.95  
IN STOCK



### SEGGER J-Link BASE - JTAG/SWD Debugger

PRODUCT ID: 2209

<https://adafru.it/e5q>

\$399.95  
IN STOCK

---

You'll need an adapter and cable to convert the 2x10 JTAG cable to SWD



### JTAG (2x10 2.54mm) to SWD (2x5 1.27mm) Cable Adapter Board

PRODUCT ID: 2094

<https://adafru.it/wbz>

\$4.95  
IN STOCK



### 10-pin 2x5 Socket-Socket 1.27mm IDC (SWD) Cable - 150mm long

PRODUCT ID: 1675

<https://adafru.it/wbA>

\$2.95  
IN STOCK

## 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 Express and Gemma/Trinket M0 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 `avrduude`), 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.](#)

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](#), then [check out the UF2 file format specification](#). Visit [Adafruit's fork of the Microsoft UF2-samd bootloader GitHub repository](#) for source code and [releases of pre-built bootloaders](#).

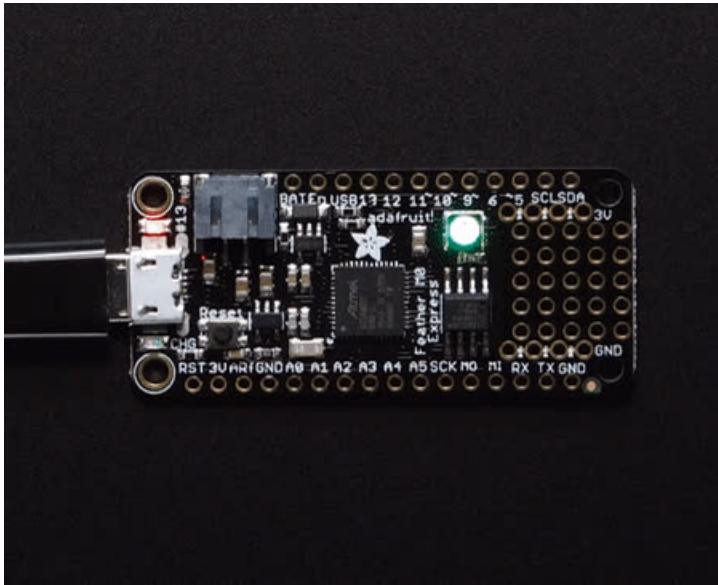
The bootloader is not needed when changing your CircuitPython code. Its only needed when upgrading the CircuitPython core or changing between CircuitPython, Arduino and Microsoft MakeCode.

## 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**

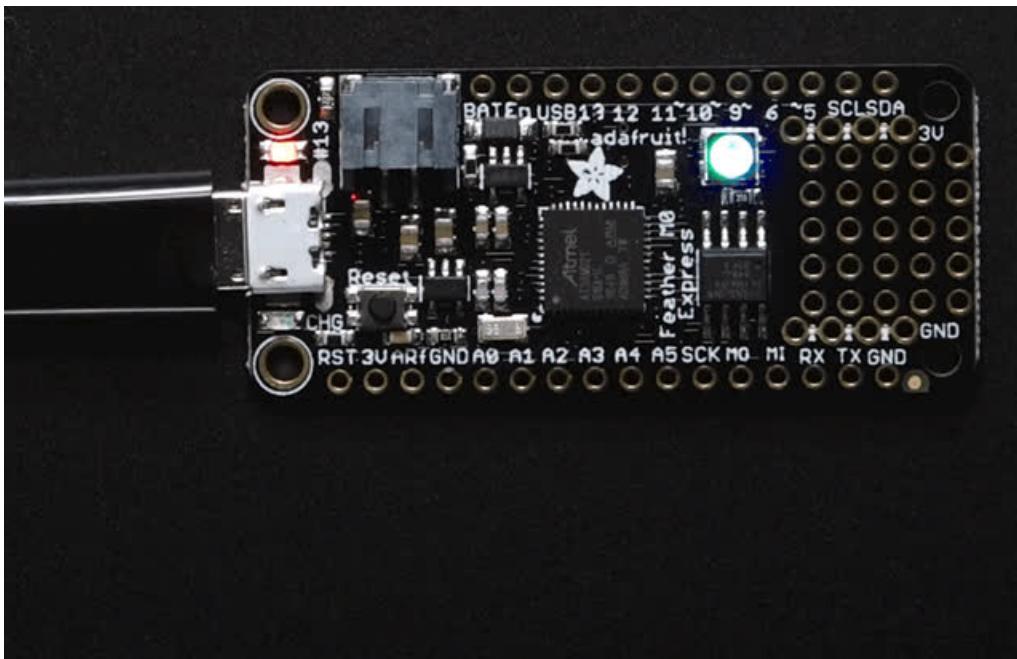
Furthermore, when the bootloader is active, it will change the color of one or more onboard neopixels to indicate the connection status, red for disconnected and green for connected. If the board is plugged in but still showing that its disconnected, try a different USB cable. Some cables only provide 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.

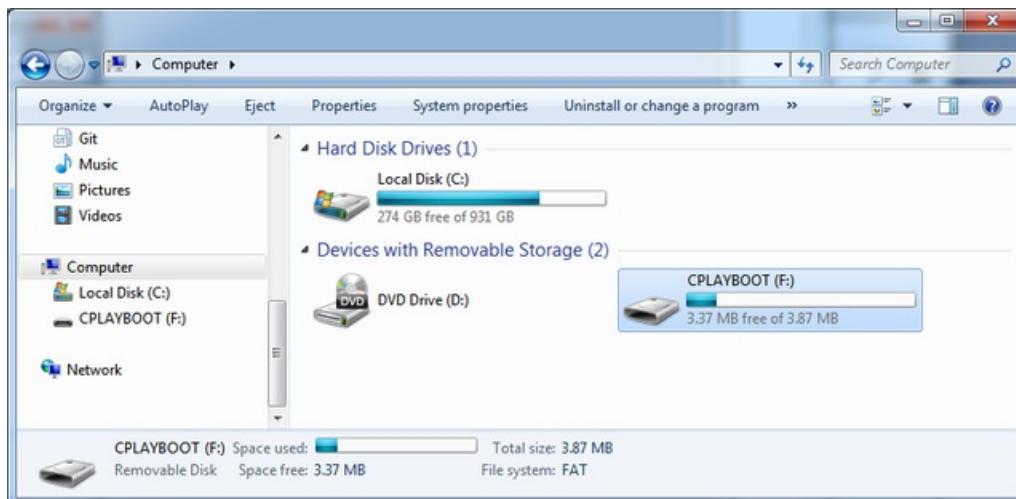


If the bootloader couldn't start, you will get a red NeoPixel LED.

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!

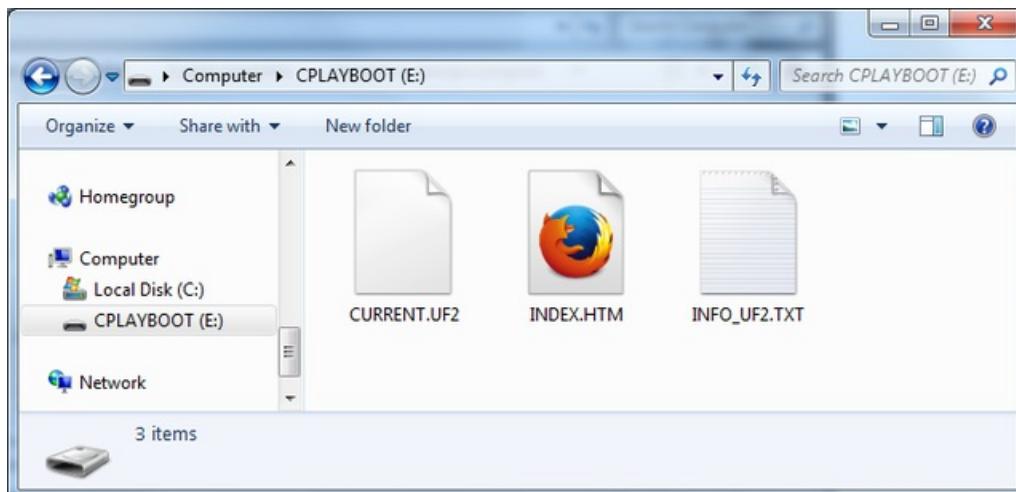


Once the bootloader is running, check your computer. You should see a USB Disk drive...



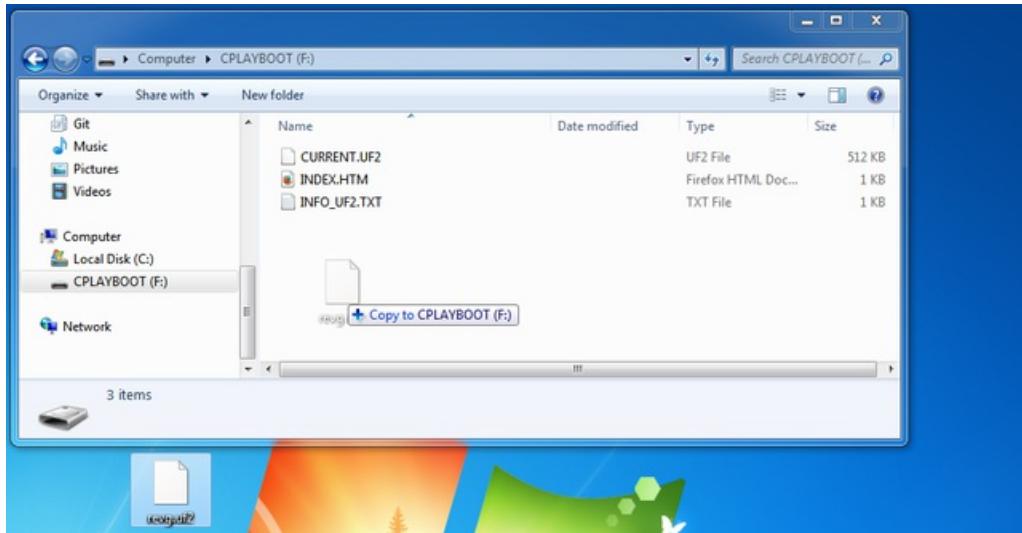
Once the bootloader is successfully connected you can open the drive and browse the virtual filesystem. This isn't the same filesystem as you use with CircuitPython or Arduino. It should have three files:

- **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.

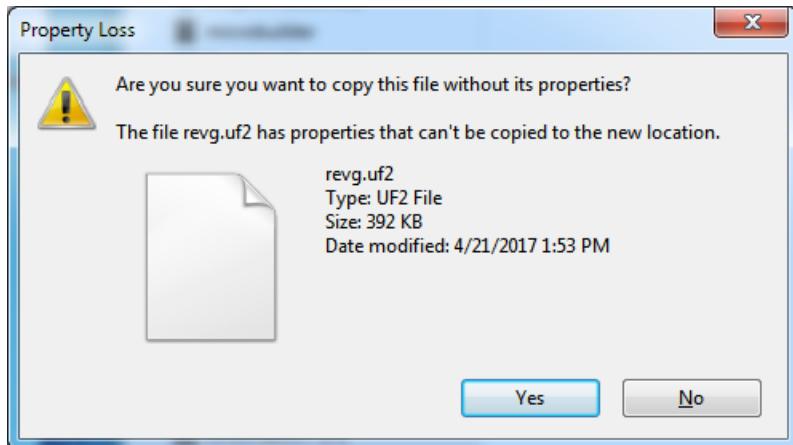


## Using the Mass Storage Bootloader

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 its properties. You can just click **Yes**



You may also get a complaint that the drive was ejected without warning. Don't worry about this. The drive only ejects once the bootloader has verified and completed the process of writing the new code

## Using the BOSSA Bootloader

As mentioned before, the bootloader is also compatible with BOSSA, which is the standard method of updating boards when in the Arduino IDE. It is a command-line tool that can be used in any operating system. We won't cover the full use of the **bossac** tool, suffice to say it can do quite a bit! More information is available at [ShumaTech](#).

### Windows 7 Drivers

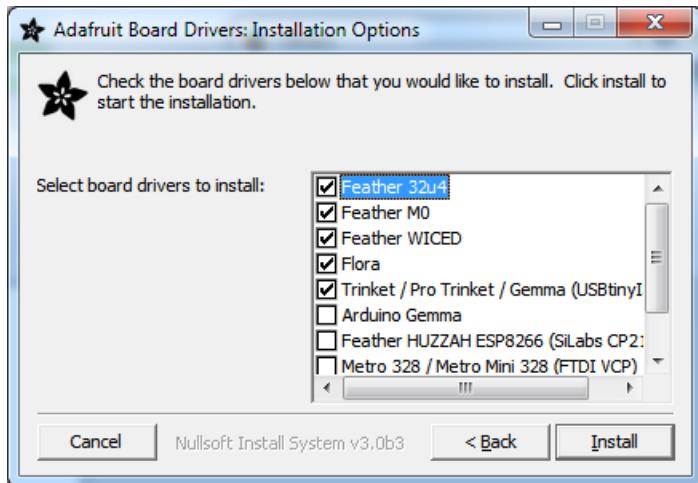
If you are running Windows 7 (or, goodness, something earlier?) You will need a Serial Port driver file. Windows 10 users do not need this so skip this step.

You can download our full driver package here:

Download Latest Adafruit Driver Installer

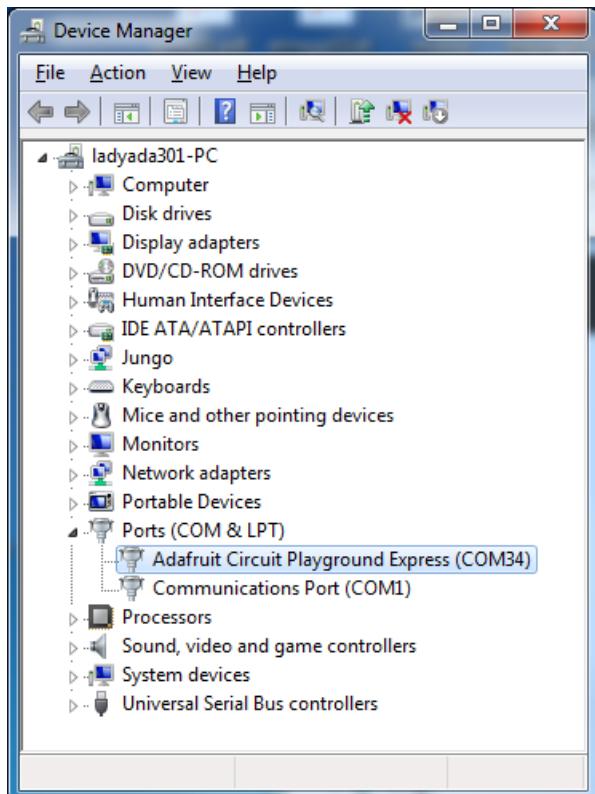
<https://adafru.it/A0N>

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

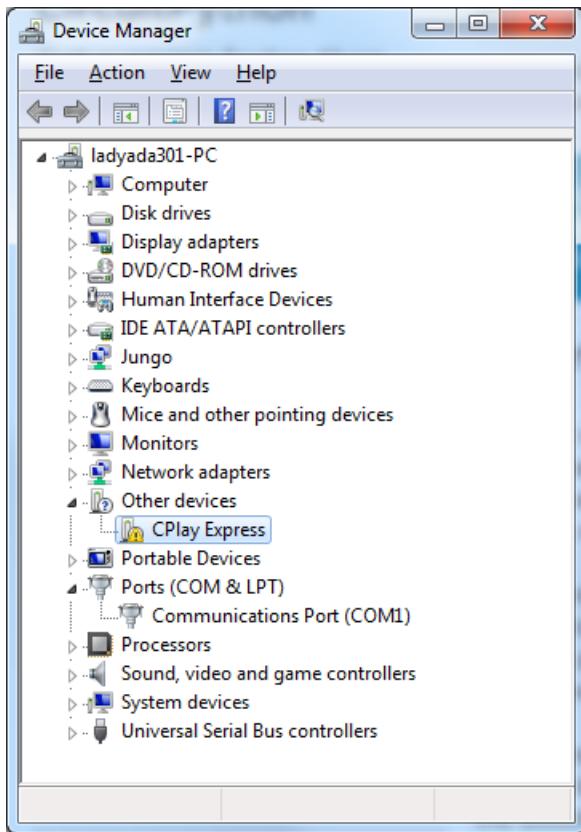


### Verifying Serial Port in Device Manager

If you're running Windows, it's 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

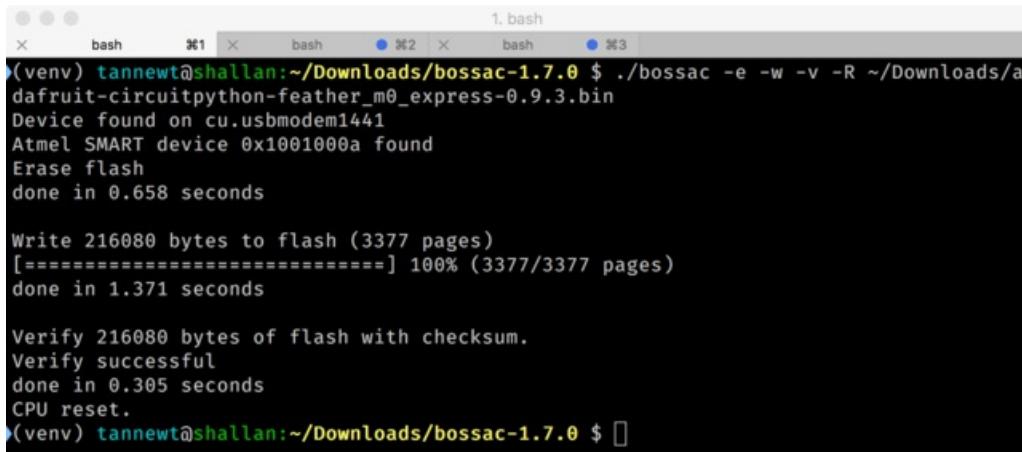
If you are using the Arduino IDE, this step is not required. But sometimes you want to read/write custom binary files, say for loading CircuitPython or your own code. We recommend using bossac v 1.7.0 (or greater), which has been tested. [The Arduino branch is most recommended.](#)

You can download the latest builds [here](#). 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`

For example here's the command line you probably want to run:

```
bossac -e -w -v -R ~/Downloads/adafruit-circuitpython-feather_m0_express-0.9.3.bin
```

This will `-e`rase the chip, `-w`rite the given file, `-v`erify the write and `-R`eset the board. After reset, CircuitPython should be running. Express boards may cause a warning of an early eject of a USB drive but just ignore it. Nothing important was being written to the drive. A hard power-reset is also recommended after `bossac`, just in case.



```
(venv) tannewt@shallan:~/Downloads/bossac-1.7.0 $ ./bossac -e -w -v -R ~/Downloads/adafruit-circuitpython-feather_m0_express-0.9.3.bin
Device found on cu.usbmodem1441
Atmel SMART device 0x1001000a found
Erase flash
done in 0.658 seconds

Write 216080 bytes to flash (3377 pages)
[=====] 100% (3377/3377 pages)
done in 1.371 seconds

Verify 216080 bytes of flash with checksum.
Verify successful
done in 0.305 seconds
CPU reset.
(venv) tannewt@shallan:~/Downloads/bossac-1.7.0 $
```

## Updating the bootloader

---

The UF2 bootloader is a new bootloader, 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.

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 v1.20.0 SFHR  
Model: Adafruit Feather M0  
Board-ID: SAMD21G18A-Feather-v0

Lastly, reload your code from Arduino or MakeCode or flash the [latest CircuitPython core](#).

The latest updaters for various boards:

Circuit Playground Express v1.23 update-bootloader.uf2

<https://adafru.it/yDv>

Feather M0 Express v1.23 update-bootloader.uf2

<https://adafru.it/yDw>

Metro M0 Express v1.23 update-bootloader.uf2

<https://adafru.it/yDx>

Gemma M0 v1.23 update-bootloader.uf2

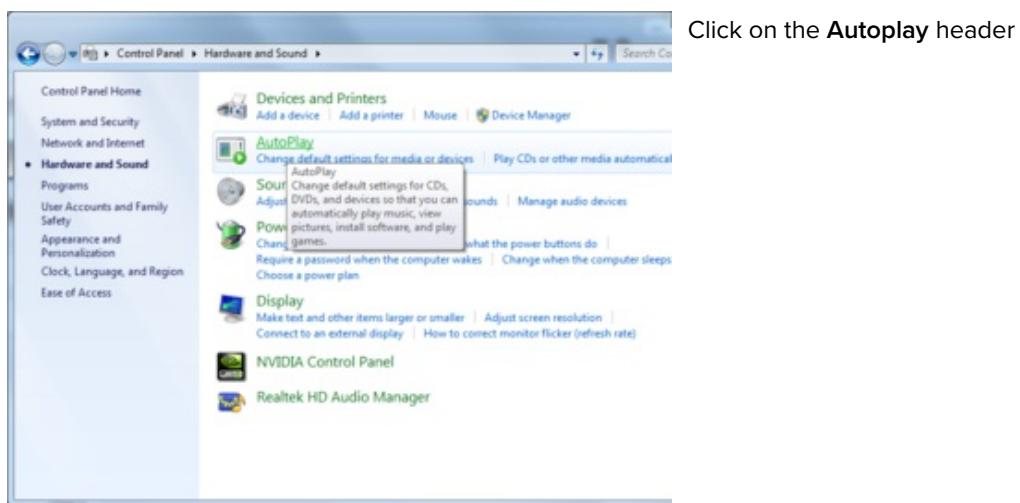
<https://adafru.it/yDy>

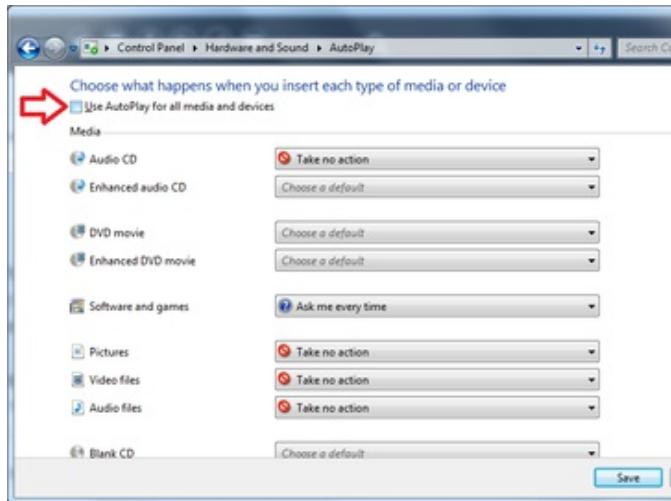
Trinket M0 v1.23 update-bootloader.uf2

<https://adafru.it/yDz>

## 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.





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](#). Make sure that your program was compiled to start at 0x2000 (8k) because the bootloader takes the first 8k. CircuitPython's [linker script](#) 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:

```
uf2conv.py -c -o build-circuitplayground_express/revg.uf2 build-circuitplayground_express/revg.bin
```

This will produce a revg.uf2 file in the same directory as the source revg.bin. The uf2 can then be flashed in the same way as above.

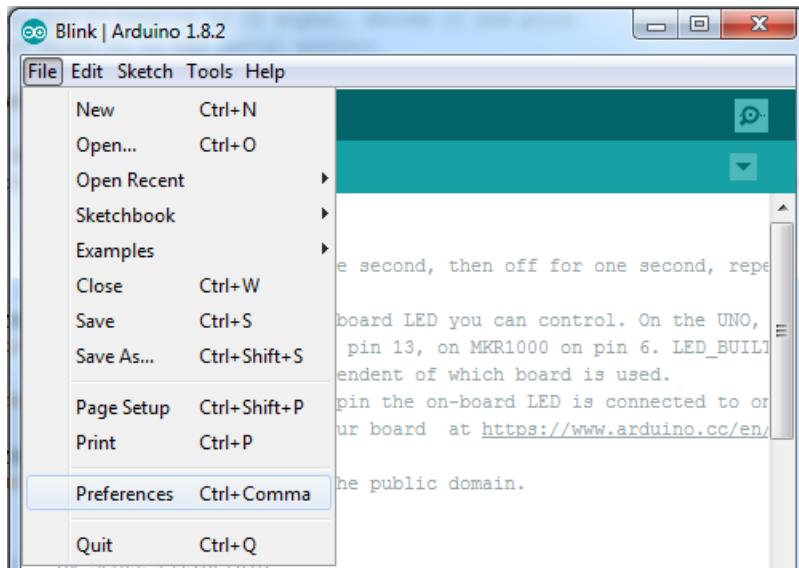
## Arduino IDE Setup

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

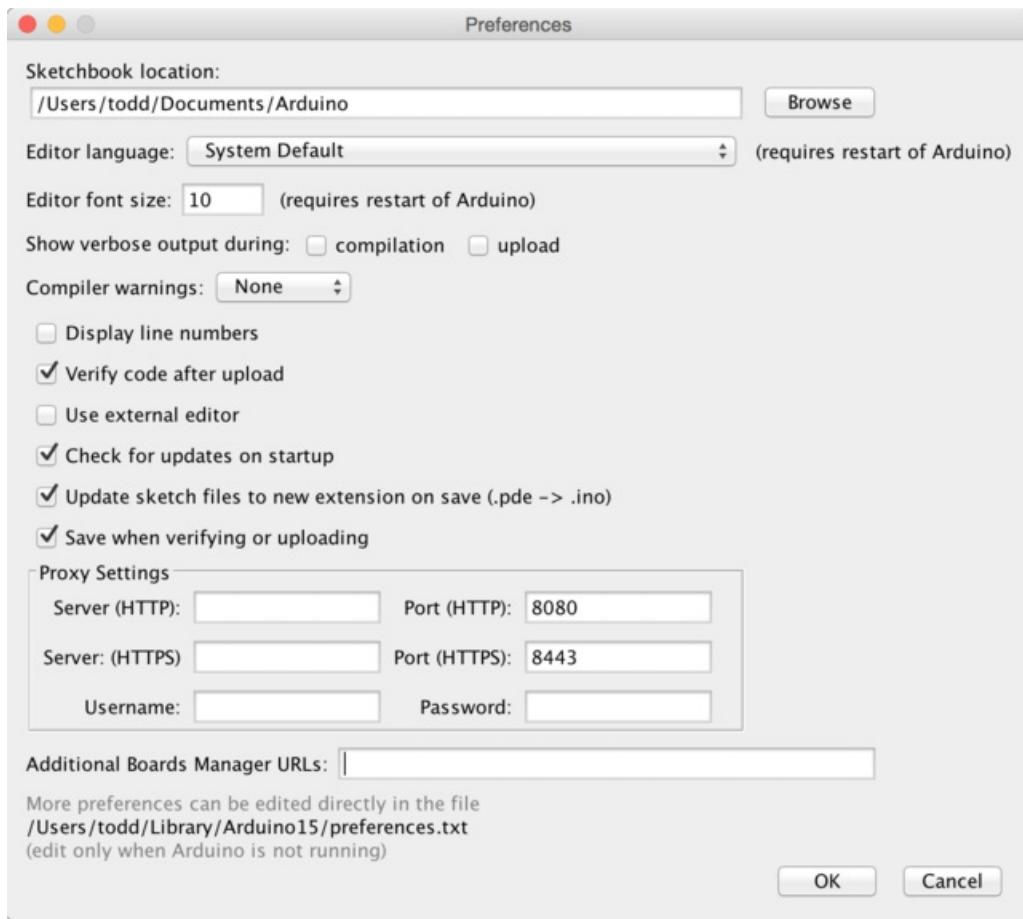
Arduino IDE Download

<https://adafru.it/f1P>

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*.



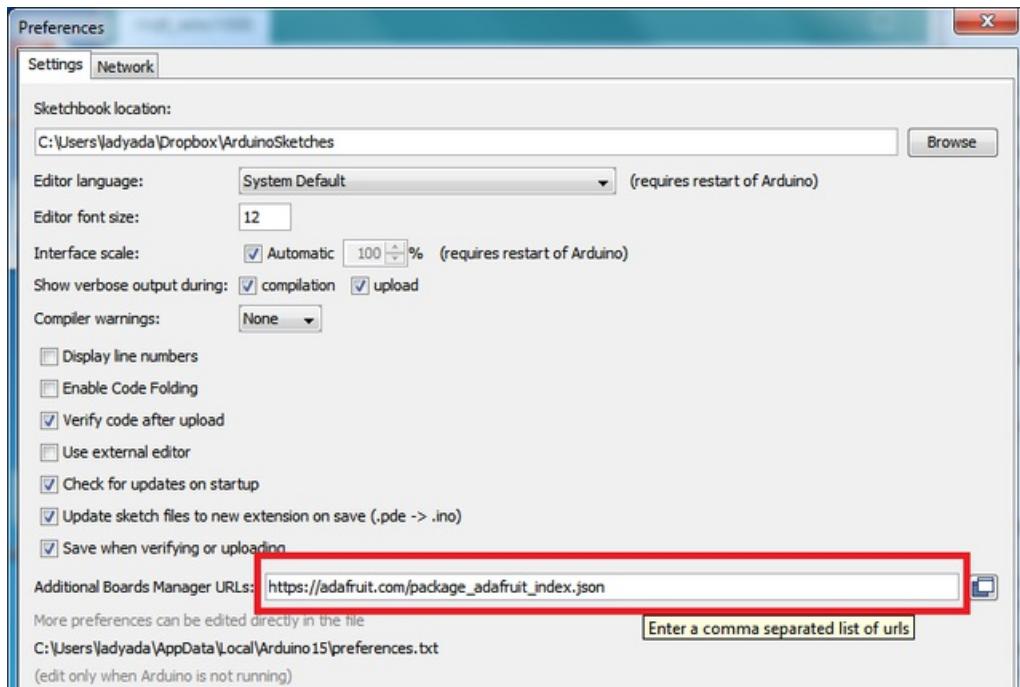
A dialog will pop up just like the one shown below.



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](#). We will only need to add one URL to the IDE in this example, but *you can add multiple URLs by separating them with commas*. Copy and paste the link below into the **Additional Boards Manager URLs** option in the Arduino IDE preferences.

[https://adafruit.github.io/arduino-board-index/package\\_adafruit\\_index.json](https://adafruit.github.io/arduino-board-index/package_adafruit_index.json)



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

- **Adafruit AVR Boards** - Includes support for Flora, Gemma, Feather 32u4, Trinket, & Trinket Pro.
- **Adafruit SAMD Boards** - Includes support for Feather M0, Metro M0, 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](#).

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

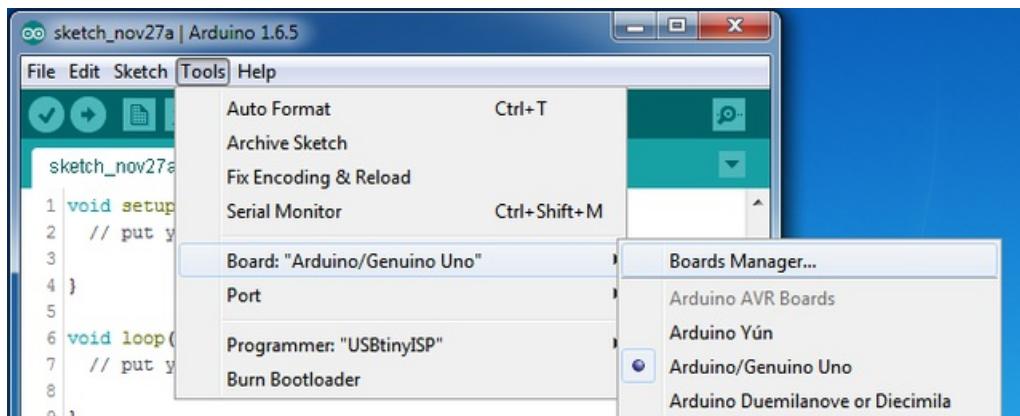
Once done click **OK** to save the new preference settings. Next we will look at installing boards with the Board Manager.

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

## Using with Arduino IDE

Since the Feather/Metro/Gemma/Trinket M0 use an ATSAMD21 chip running at 48 MHz, 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, 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.

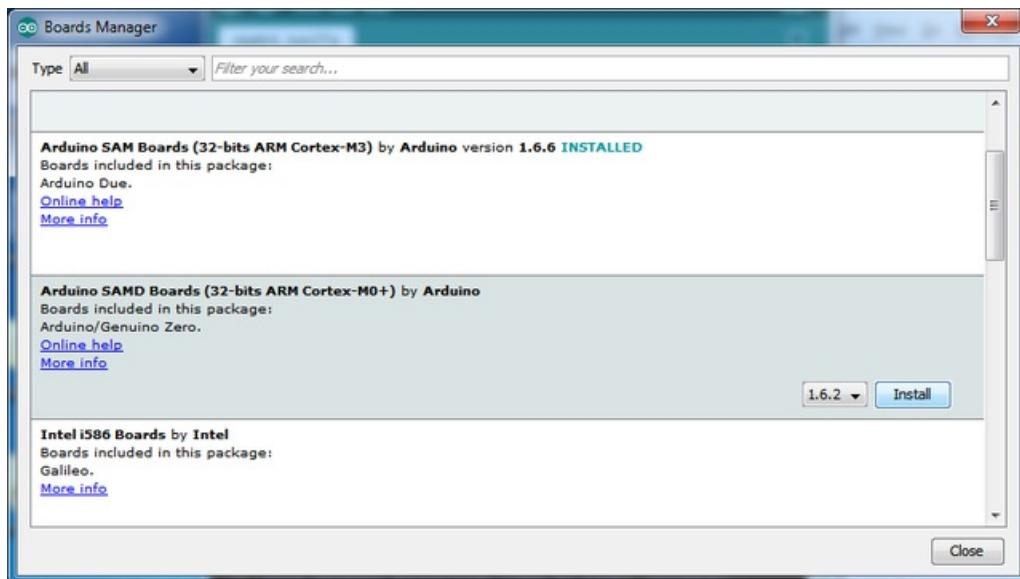


Once the Board Manager opens, click on the category drop down menu on the top left hand side of the window and select **Contributed**. You will then be able to select and install the boards supplied by the URLs added to the preferences.

## Install SAMD Support

First up, install the **Arduino SAMD Boards** version **1.6.15** or later

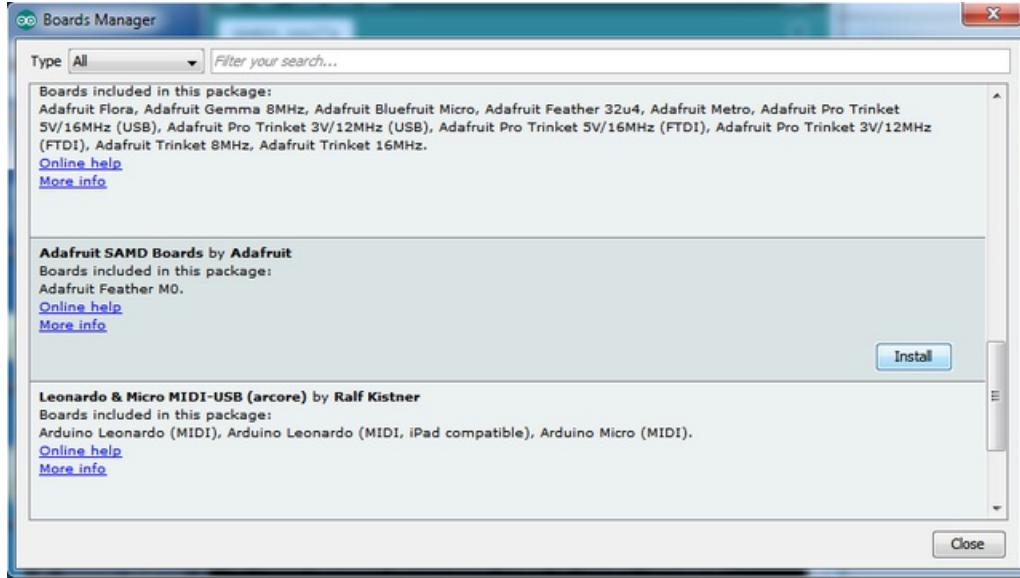
You can type **Arduino SAMD** in the top search bar, then when you see the entry, click **Install**



## Install Adafruit SAMD

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

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

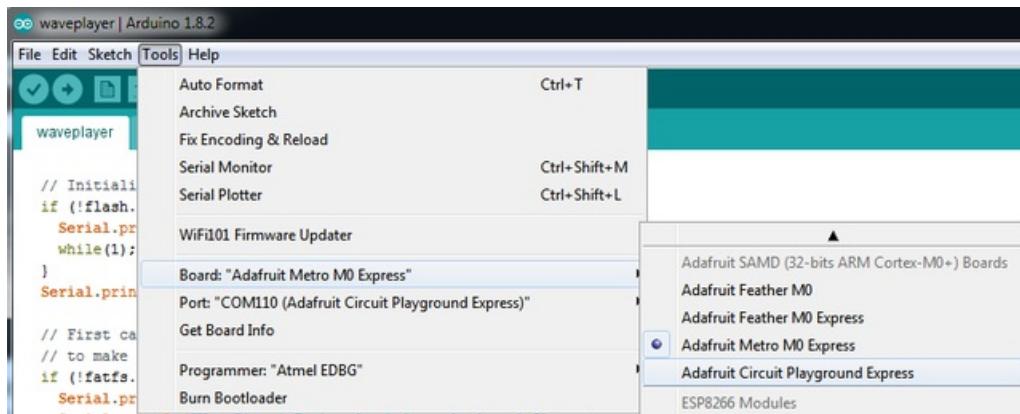


Even though in theory you don't need to - I recommend rebooting the IDE

**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**



## Install Drivers (Windows 7 Only)

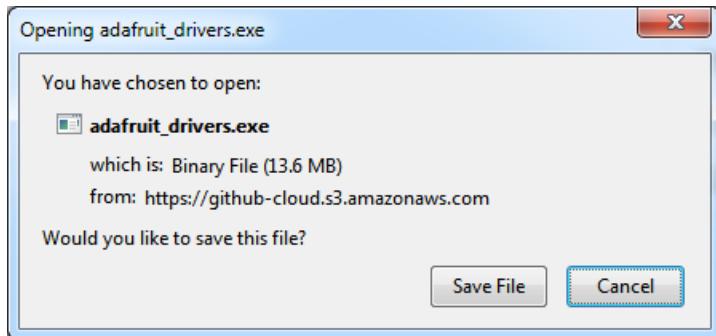
When you plug in the board, you'll need to possibly install a driver

Click below to download our Driver Installer

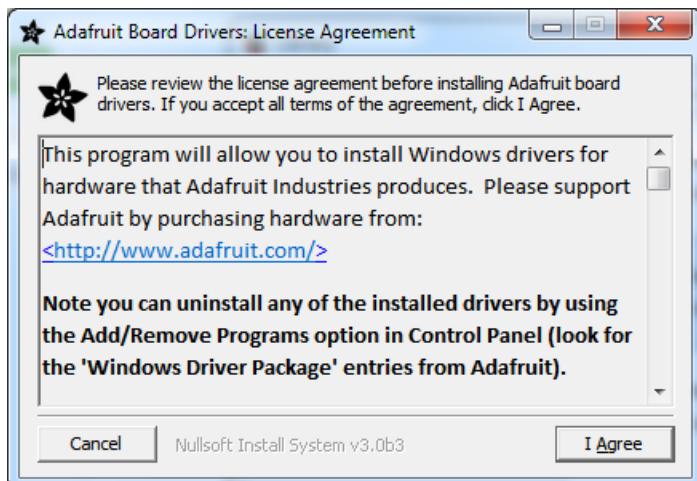
## Download Adafruit Driver Installer v2.0.0.0

<https://adafru.it/zek>

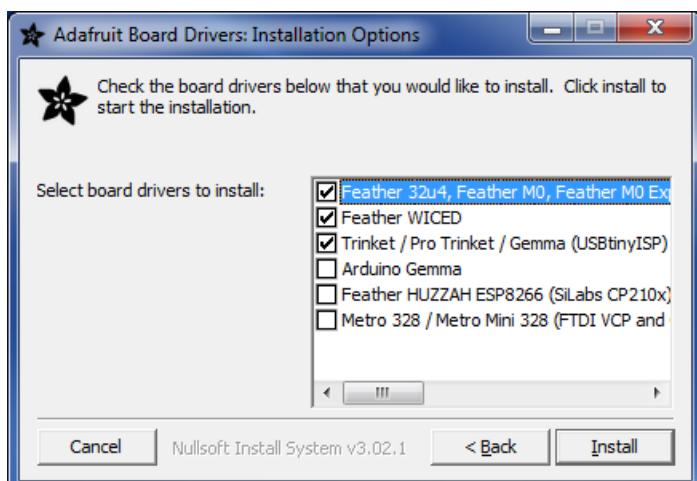
Download and run the installer



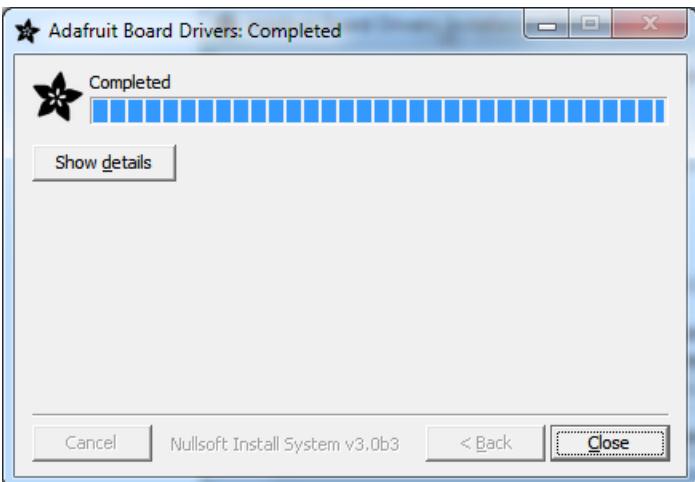
Run the installer! Since we bundle the SiLabs and FTDI drivers as well, you'll need to click through the license



Select which drivers you want to install, the defaults will set you up with just about every Adafruit board!



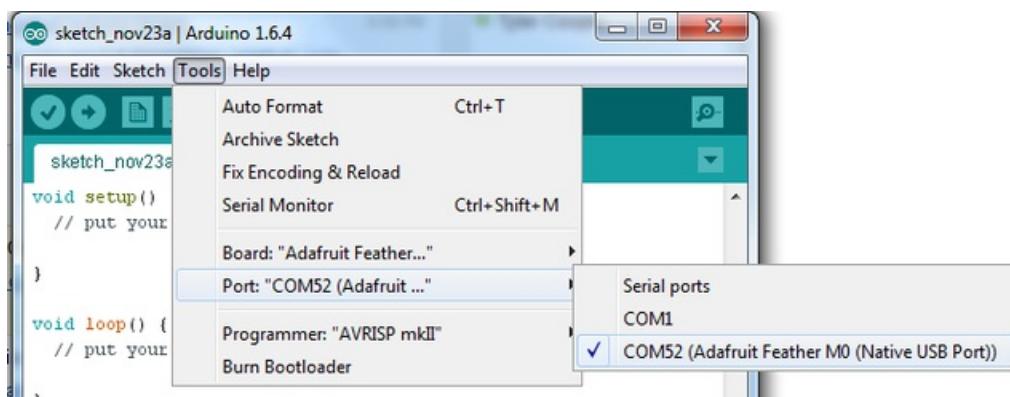
Click **Install** to do the installin'



## Blink

Now you can upload your first blink sketch!

Plug in the Gemma M0, Trinket M0, Metro M0 or Feather M0 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 dropdown, it'll even be 'indicated' as Trinket/Gemma/Metro/Feather M0!



Now load up the Blink example

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

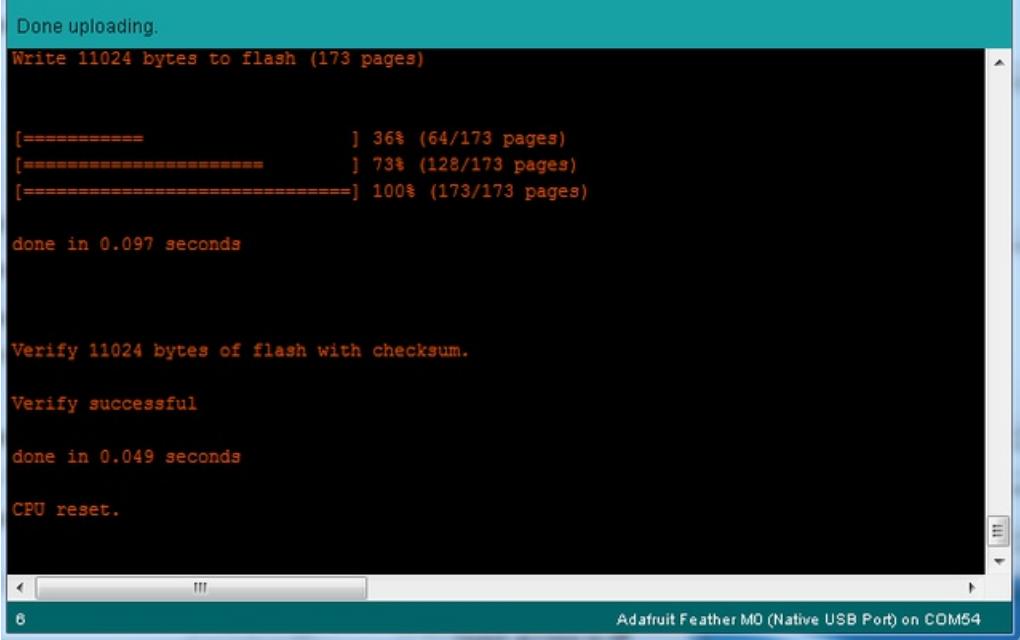
// the loop function runs over and over again forever
void loop() {
    digitalWrite(13, HIGH);      // turn the LED on (HIGH is the voltage level)
    delay(1000);                // wait for a second
    digitalWrite(13, LOW);       // turn the LED off by making the voltage LOW
    delay(1000);    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

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

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

## Sucessful Upload

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



```
Done uploading.  
Write 11024 bytes to flash (173 pages)  
  
[=====] 36% (64/173 pages)  
[=====] 73% (128/173 pages)  
[=====] 100% (173/173 pages)  
  
done in 0.097 seconds  
  
  
Verify 11024 bytes of flash with checksum.  
Verify successful  
  
done in 0.049 seconds  
  
CPU reset.
```

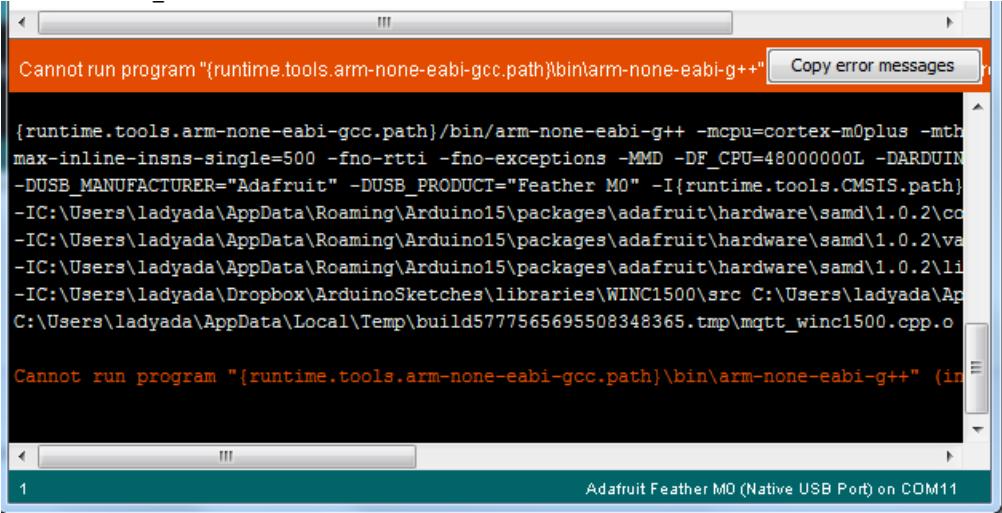
The screenshot shows a terminal window with a dark background and light-colored text. It displays the output of an Arduino upload process. The text includes progress bars indicating the write and verify steps, and a final CPU reset message. The window title bar reads "Adafruit Feather M0 (Native USB Port) on COM54".

## Compilation Issues

If you get an alert that looks like

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

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



```
Cannot run program "{runtime.tools.arm-none-eabi-gcc.path}\bin\arm-none-eabi-g++" Copy error messages  
  
{runtime.tools.arm-none-eabi-gcc.path}/bin/arm-none-eabi-g++ -mcpu=cortex-m0plus -mthumb -fno-rtti -fno-exceptions -MMD -DF_CPU=4800000L -DARDUIN  
-DUSB_MANUFACTURER="Adafruit" -DUSB_PRODUCT="Feather M0" -I{runtime.tools\CMSIS.path}  
-IC:\Users\ladyada\AppData\Roaming\Arduino15\packages\adafruit\hardware\samd\1.0.2\co  
-IC:\Users\ladyada\AppData\Roaming\Arduino15\packages\adafruit\hardware\samd\1.0.2\va  
-IC:\Users\ladyada\AppData\Roaming\Arduino15\packages\adafruit\hardware\samd\1.0.2\li  
-IC:\Users\ladyada\Dropbox\ArduinoSketches\libraries\WINC1500\src C:\Users\ladyada\Ap  
C:\Users\ladyada\AppData\Local\Temp\build5777565695508348365.tmp\mqtt_winc1500.cpp.o  
  
Cannot run program "{runtime.tools.arm-none-eabi-gcc.path}\bin\arm-none-eabi-g++" (in
```

The screenshot shows a terminal window with a dark background and light-colored text. It displays a long error message starting with "Cannot run program" followed by a path to the compiler executable. The message continues with various compiler flags and paths. The window title bar reads "Adafruit Feather M0 (Native USB Port) on COM11".

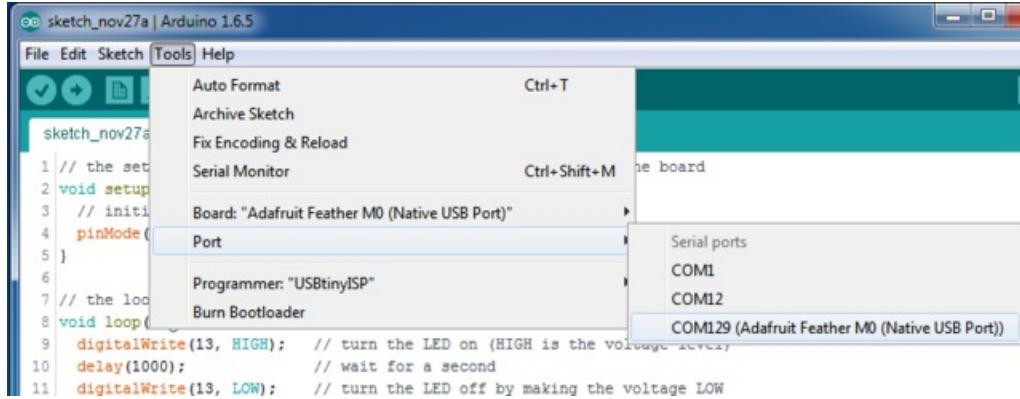
## Manually bootloading

---

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](#).

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.](#)

## Adapting Sketches to M0

---

The ATSAMD21 is a very nice little chip but its fairly new as Arduino-compatible cores go. **Most** sketches & libraries will work but here's a few things we noticed!

The below note are for all M0 boards, but not all may apply (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 is the same as the output-selection register.

For the M0, you can't do this anymore! Instead, use

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

which has the benefit of being backwards compatible with AVR.

### 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 Core, we fixed it so that Serial goes to USB when you use a Feather M0 so it will automatically work just fine.

However, on the off chance you are using the official Arduino SAMD core not the Adafruit version (which really, we recommend you use our version because as you can see it can vary) & 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:



The screenshot shows the Arduino IDE interface with a sketch titled "datecalc". The code includes #includes for Wire.h and RTClib.h, defines the Serial port, and contains a showDate function that prints a date/time string to the serial port.

```
datecalc | Arduino 1.6.5
File Edit Sketch Tools Help
datecalc $ 
1 // Simple date conversions and calculations
2
3 #include <Wire.h>
4 #include "RTClib.h"
5
6 #if defined(ARDUINO_ARCH_SAMD)
7 // for Zero, output on USB Serial console, remove line below if using programming port to program the Zero!
8 #define Serial SerialUSB
9 #endif
10
11 void showDate(const char* txt, const DateTimes dt) {
12   Serial.print(txt);
13   Serial.print(" ");
14 }
```

## 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)`

## 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, you'll need to *double click* the button. You will see a pulsing red LED to let you know you're in bootloader mode. Once in that mode, it won't time out! Click reset again if you want to go back to launching code

## Aligned Memory Access

---

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

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

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

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

## Floating Point Conversion

---

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

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

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

<http://forum.arduino.cc/index.php?topic=368720.0>

## 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> 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 \$3FFF Then it is in FLASH

## Using SPI Flash

---

One of the best features of the M0 express board is a small SPI flash memory chip built into the board. This memory can be used for almost any purpose like storing data files, Python code, and more. Think of it like a little SD card that is always connected to the board, and in fact with Arduino you can access the memory using a library that is very similar to the [Arduino SD card library](#). You can even read and write files that CircuitPython stores on the flash chip!

To use the flash memory with Arduino you'll need to install the [Adafruit SPI Flash Memory library](#) in the Arduino IDE. Click the button below to download the source for this library, open the zip file, and then copy it into an **Adafruit\_SPIFlash** folder (remove the -master GitHub adds to the downloaded zip and folder) [in the Arduino library folder on your computer](#):

[Download Adafruit\\_SPIFlash](#)

<https://adafru.it/wbu>

Once the library is installed open the Arduino IDE and look for the following examples in the library:

- **fatfs\_circuitpython**
- **fatfs\_datalogging**
- **fatfs\_format**
- **fatfs\_full\_usage**
- **fatfs\_print\_file**
- **flash\_erase**

These examples allow you to format the flash memory with a FAT filesystem (the same kind of filesystem used on SD cards) and read and write files to it just like a SD card.

## Read & Write CircuitPython Files

---

The **fatfs\_circuitpython** example shows how to read and write files on the flash chip so that they're accessible from CircuitPython. This means you can run a CircuitPython program on your board and have it store data, then run an Arduino sketch that uses this library to interact with the same data.

Note that before you use the **fatfs\_circuitpython** example you **must** have loaded CircuitPython on your board. [Load the latest version of CircuitPython as explained in this guide](#) first to ensure a CircuitPython filesystem is initialized and written to the flash chip. Once you've loaded CircuitPython then you can run the **fatfs\_circuitpython** example sketch.

To run the sketch load it in the Arduino IDE and upload it to the Feather M0 board. Then open the serial monitor at 115200 baud. You should see the serial monitor display messages as it attempts to read files and write to a file on the flash chip. Specifically the example will look for a **boot.py** and **main.py** file (like what CircuitPython runs when it starts) and print out their contents. Then it will add a line to the end of a **data.txt** file on the board (creating it if it doesn't exist already). After running the sketch you can reload CircuitPython on the board and open the **data.txt** file to read it from CircuitPython!

To understand how to read & write files that are compatible with CircuitPython let's examine the sketch code. First notice an instance of the **Adafruit\_M0\_Express\_CircuitPython** class is created and passed an instance of the flash chip class in the last line below:

```

#define FLASH_SS      SS1           // Flash chip SS pin.
#define FLASH_SPI_PORT SPI1        // What SPI port is Flash on?

Adafruit_SPIFlash flash(FLASH_SS, &FLASH_SPI_PORT);    // Use hardware SPI

// Alternatively you can define and use non-SPI pins!
//Adafruit_SPIFlash flash(SCK1, MISO1, MOSI1, FLASH_SS);

// Finally create an Adafruit_M0_Express_CircuitPython object which gives
// an SD card-like interface to interacting with files stored in CircuitPython's
// flash filesystem.
Adafruit_M0_Express_CircuitPython pythonfs(flash);

```

By using this **Adafruit\_M0\_Express\_CircuitPython** class you'll get a filesystem object that is compatible with reading and writing files on a CircuitPython-formatted flash chip. This is very important for interoperability between CircuitPython and Arduino as CircuitPython has specialized partitioning and flash memory layout that isn't compatible with simpler uses of the library (shown in the other examples).

Once an instance of the **Adafruit\_M0\_Express\_CircuitPython** class is created (called **pythonfs** in this sketch) you can go on to interact with it just like if it were the [SD card library in Arduino](#). You can open files for reading & writing, create directories, delete files and directories and more. Here's how the sketch checks if a **boot.py** file exists and prints it out a character at a time:

```

// Check if a boot.py exists and print it out.
if (pythonfs.exists("boot.py")) {
    File bootPy = pythonfs.open("boot.py", FILE_READ);
    Serial.println("Printing boot.py...");
    while (bootPy.available()) {
        char c = bootPy.read();
        Serial.print(c);
    }
    Serial.println();
}
else {
    Serial.println("No boot.py found...");
}

```

Notice the **exists** function is called to check if the **boot.py** file is found, and then the **open** function is used to open it in read mode. Once a file is opened you'll get a reference to a **File** class object which you can read and write from as if it were a **Serial** device (again just like the **SD card library**, [all of the same File class functions are available](#)). In this case the **available** function will return the number of bytes left to read in the file, and the **read** function will read a character at a time to print it to the serial monitor.

Writing a file is just as easy, here's how the sketch writes to **data.txt**:

```
// Create or append to a data.txt file and add a new line
// to the end of it. CircuitPython code can later open and
// see this file too!
File data = pythonfs.open("data.txt", FILE_WRITE);
if (data) {
    // Write a new line to the file:
    data.println("Hello CircuitPython from Arduino!");
    data.close();
    // See the other fatfs examples like fatfs_full_usage and fatfs_datalogging
    // for more examples of interacting with files.
    Serial.println("Wrote a new line to the end of data.txt!");
}
else {
    Serial.println("Error, failed to open data file for writing!");
}
```

Again the **open** function is used but this time it's told to open the file for writing. In this mode the file will be opened for appending (i.e. data added to the end of it) if it exists, or it will be created if it doesn't exist. Once the file is open print functions like **print** and **println** can be used to write data to the file (just like writing to the serial monitor). Be sure to **close** the file when finished writing!

That's all there is to basic file reading and writing. Check out the **fatfs\_full\_usage** example for examples of even more functions like creating directories, deleting files & directories, checking the size of files, and more! Remember though to interact with CircuitPython files you need to use the **Adafruit\_Feather\_M0\_CircuitPython** class as shown in the **fatfs\_circuitpython** example above!

## Format Flash Memory

The **fatfs\_format** example will format the SPI flash with a new blank filesystem. **Be warned this sketch will delete all data on the flash memory, including any Python code or other data you might have stored!** The format sketch is useful if you'd like to wipe everything away and start fresh, or to help get back in a good state if the memory should get corrupted for some reason.

**Be aware too the fatfs\_format and examples below are not compatible with a CircuitPython-formatted flash chip!** If you need to share data between Arduino & CircuitPython check out the **fatfs\_circuitpython** example above.

To run the format sketch load it in the Arduino IDE and upload it to the Feather M0 board. Then open the serial monitor at 115200 baud. You should see the serial monitor display a message asking you to confirm formatting the flash. If you don't see this message then close the serial monitor, press the board's reset button, and open the serial monitor again.

```
Adafruit SPI Flash FatFs Format Example
Flash chip JEDEC ID: 0x1401501
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
This sketch will ERASE ALL DATA on the flash chip and format it with a new filesystem!
Type OK (all caps) and press enter to continue.
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
Partitioning flash with 1 primary partition...
Partitioned flash!
Creating and formatting FAT filesystem (this takes ~60 seconds)...
Formatted flash!
Flash chip successfully formatted with new empty filesystem!
```

Autoscroll      Both NL & CR      230400 baud

Type **OK** and press enter in the serial monitor input to confirm that you'd like to format the flash memory. **You need to enter OK in all capital letters!**

Once confirmed the sketch will format the flash memory. The format process takes about a minute so be patient as the data is erased and formatted. You should see a message printed once the format process is complete. At this point the flash chip will be ready to use with a brand new empty filesystem.

## Datalogging Example

One handy use of the SPI flash is to store data, like datalogging sensor readings. The **fatfs\_datalogging** example shows basic file writing/datalogging. Open the example in the Arduino IDE and upload it to your Feather M0 board. Then open the serial monitor at 115200 baud. You should see a message printed every minute as the sketch writes a new line of data to a file on the flash filesystem.

To understand how to write to a file look in the loop function of the sketch:

```
// Open the datalogging file for writing. The FILE_WRITE mode will open
// the file for appending, i.e. it will add new data to the end of the file.
File dataFile = fatfs.open(FILE_NAME, FILE_WRITE);
// Check that the file opened successfully and write a line to it.
if (dataFile) {
    // Take a new data reading from a sensor, etc. For this example just
    // make up a random number.
    int reading = random(0,100);
    // Write a line to the file. You can use all the same print functions
    // as if you're writing to the serial monitor. For example to write
    // two CSV (commas separated) values:
    dataFile.print("Sensor #1");
    dataFile.print(",");
    dataFile.print(reading, DEC);
    dataFile.println();
    // Finally close the file when done writing. This is smart to do to make
    // sure all the data is written to the file.
    dataFile.close();
    Serial.println("Wrote new measurement to data file!");
}
```

Just like using the Arduino SD card library you create a **File** object by calling an **open** function and pointing it at the name of the file and how you'd like to open it (**FILE\_WRITE** mode, i.e. writing new data to the end of the file). Notice however instead of calling open on a global SD card object you're calling it on a **fatfs** object created earlier in the sketch (look at the top after the `#define` configuration values).

Once the file is opened it's simply a matter of calling **print** and **println** functions on the file object to write data inside of it. This is just like writing data to the serial monitor and you can print out text, numeric, and other types of data. Be sure to close the file when you're done writing to ensure the data is stored correctly!

## Reading and Printing Files

The **fatfs\_print\_file** example will open a file (by default the `data.csv` file created by running the **fatfs\_datalogging** example above) and print all of its contents to the serial monitor. Open the **fatfs\_print\_file** example and load it on your Feather M0 board, then open the serial monitor at 115200 baud. You should see the sketch print out the contents of `data.csv` (if you don't have a file called `data.csv` on the flash look at running the datalogging example above first).

To understand how to read data from a file look in the **setup** function of the sketch:

```
// Open the file for reading and check that it was successfully opened.  
// The FILE_READ mode will open the file for reading.  
File dataFile = fatfs.open(FILE_NAME, FILE_READ);  
if (dataFile) {  
    // File was opened, now print out data character by character until at the  
    // end of the file.  
    Serial.println("Opened file, printing contents below:");  
    while (dataFile.available()) {  
        // Use the read function to read the next character.  
        // You can alternatively use other functions like readUntil, readString, etc.  
        // See the fatfs_full_usage example for more details.  
        char c = dataFile.read();  
        Serial.print(c);  
    }  
}
```

Just like when writing data with the datalogging example you create a **File** object by calling the **open** function on a **fatfs** object. This time however you pass a file mode of **FILE\_READ** which tells the filesystem you want to read data.

After you open a file for reading you can easily check if data is available by calling the **available** function on the file, and then read a single character with the **read** function. This makes it easy to loop through all of the data in a file by checking if it's available and reading a character at a time. However there are more advanced read functions you can use too--see the **fatfs\_full\_usage** example or even the [Arduino SD library documentation](#) (the SPI flash library implements the same functions).

## Full Usage Example

For a more complete demonstration of reading and writing files look at the **fatfs\_full\_usage** example. This examples uses every function in the library and demonstrates things like checking for the existence of a file, creating directories, deleting files, deleting directories, and more.

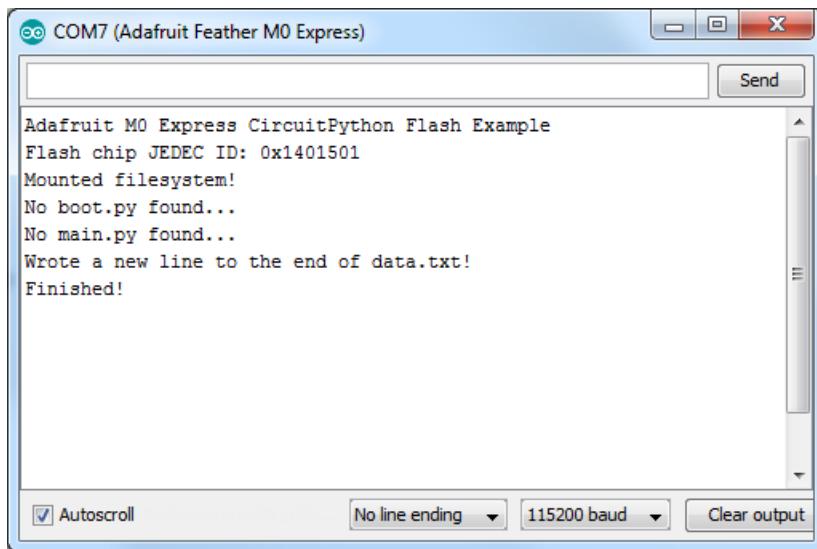
Remember the SPI flash library is built to have the same functions and interface as the [Arduino SD library](#) so if you have code or examples that store data on a SD card they should be easy to adapt to use the SPI flash library, just create a **fatfs** object like in the examples above and use its **open** function instead of the global SD object's **open** function. Once you have a reference to a file all of the functions and usage should be the same between the SPI flash

and SD libraries!

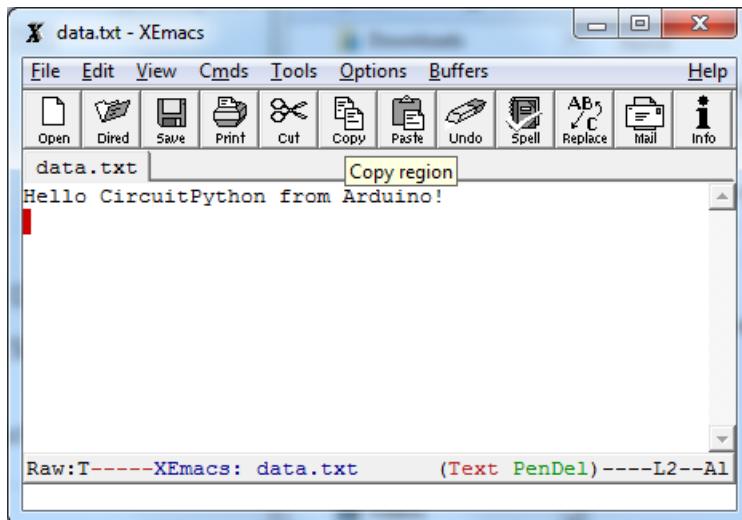
## Accessing SPI Flash

Arduino doesn't have the ability to show up as a 'mass storage' disk drive. So instead we must use CircuitPython to do that part for us. Here's the full technique:

- Start the bootloader on the Express board. Drag over the latest **circuitpython.uf2** file
- After a moment, you should see a **CIRCUITPY** drive appear on your hard drive with **boot\_out.txt** on it
- Now go to Arduino and upload the **fatfs\_circuitpython** example sketch from the Adafruit SPI library. Open the serial console. It will successfully mount the filesystem and write a new line to **data.txt**



- Back on your computer, re-start the Express board bootloader, and re-drag **circuitpython.uf2** onto the **BOOT** drive to reinstall circuitpython
- Check the **CIRCUITPY** drive, you should now see **data.txt** which you can open to read!



Once you have your Arduino sketch working well, for datalogging, you can simplify this procedure by dragging **CURRENT.UF2** off of the **BOOT** drive to make a backup of the current program before loading circuitpython on. Then once you've accessed the file you want, re-drag **CURRENT.UF2** back onto the **BOOT** drive to re-install the Arduino

sketch!

## Metro M0 HELP!

---

**My Metro M0 stopped working when I unplugged the USB!**

A lot of our example sketches have a

```
while (!Serial);
```

line in setup(), to keep the board waiting until the USB is opened. This makes it a lot easier to debug a program because you get to see all the USB data output. If you want to run your Metro M0 without USB connectivity, delete or comment out that line

**My Metro never shows up as a COM or Serial port in the Arduino IDE**

**A vast number of Metro 'failures' are due to charge-only USB cables**

We get upwards of 5 complaints a day that turn out to be due to charge-only cables!

Use only a cable that you **know** is for data syncing

If you have any charge-only cables, cut them in half throw them out. We are serious! They tend to be low quality in general, and will only confuse you and others later, just get a good data+charge USB cable

**Ack! I "did something" and now when I plug in the Metro, it doesn't show up as a device anymore so I cant upload to it or fix it...**

No problem! You can 'repair' a bad code upload easily. Note that this can happen if you set a watchdog timer or sleep mode that stops USB, or any sketch that 'crashes' your Metro

1. Turn on **verbose upload** in the Arduino IDE preferences
2. Plug in Metro M0, it won't show up as a COM/serial port that's ok
3. Open up the Blink example (Examples->Basics->Blink)
4. Select the correct board in the Tools menu, e.g. Metro M0 (check your board to make sure you have the right one selected!)
5. Compile it (make sure that works)
6. Click Upload to attempt to upload the code
7. The IDE will print out a bunch of COM Ports as it tries to upload. **During this time, double-click the reset button, you'll see the red pulsing LED and the NeoPixel will be green that tells you its now in bootloading mode**
8. The Metro will show up as the Bootloader COM/Serial port
9. The IDE should see the bootloader COM/Serial port and upload properly

**I can't get the Metro USB device to show up - I get "USB Device Malfunctioning" errors!**

This seems to happen when people select the wrong board from the Arduino Boards menu.

If you have a Metro M0 (look on the board to read what it is you have) Make sure you select **Metro M0** - do not use Feather M0 or Arduino Zero

**I'm having problems with COM ports and my Metro M0**

Theres **two** COM ports you can have with the M0, one is the **user port** and one is the **bootloader port**. They are not the same COM port number!

When you upload a new user program it will come up with a user com port, particularly if you use Serial in your user program.

If you crash your user program, or have a program that halts or otherwise fails, the user com port can disappear.

When the user COM port disappears, Arduino will not be able to automatically start the bootloader and upload new software.

So you will need to help it by performing the click-during upload procedure to re-start the bootloader, and upload something that is known working like "Blink"

**I don't understand why the COM port disappears, this does not happen on my Arduino UNO!**

UNO-type Arduinos have a *seperate* serial port chip (aka "FTDI chip" or "Prolific PL2303" etc etc) which handles all serial port capability separately than the main chip. This way if the main chip fails, you can always use the COM port.

M0 and 32u4-based Arduinos do not have a separate chip, instead the main processor performs this task for you. It allows for a lower cost, higher power setup...but requires a little more effort since you will need to 'kick' into the bootloader manually once in a while

**I'm trying to upload to my 32u4, getting "avrdude: butterfly\_recv(): programmer is not responding" errors**

This is likely because the bootloader is not kicking in and you are accidentally **trying to upload to the wrong COM port**

The best solution is what is detailed above: manually upload Blink or a similar working sketch by hand by manually launching the bootloader

**I'm trying to upload to my Metro M0, and I get this error "Connecting to programmer: .avrdude: butterfly\_recv(): programmer is not responding"**

You probably don't have Metro M0 selected in the boards drop-down. Make sure you selected **Metro M0**.

**I'm trying to upload to my Metro and i get this error "avrdude: ser\_recv(): programmer is not responding"**

You probably don't have Metro M0 selected in the boards drop-down. Make sure you selected **Metro M0**

## CircuitPython Setup

---



CircuitPython is a derivative of [MicroPython](#) 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 [download CircuitPython](#) and drag it onto the drive that appears (only available on Express boards currently). Once installed, simply copy and edit files on the drive to iterate.

### Downloading

---

The latest builds of [CircuitPython](#) are available from the [GitHub release page](#). Binaries for different boards are listed under the Downloads section. Pick the one that matches your board such as [adafruit-circuitpython-feather\\_m0\\_express-0.9.3.bin](#) for the Feather M0 Express or [adafruit-circuitpython-metro\\_m0\\_express-0.9.3.bin](#) for the Metro M0 Express.

Files that end with `.bin` can be flashed with `esptool.py` or `bossac`. Files ending in `.uf2` can be flashed onto a virtual drive when in bootloader mode.

Click here to see the latest CircuitPython  
Releases

<https://adafru.it/vIF>

You will see a list of all available *flavors* of CircuitPython. Since we support a lot of different hardware, we have a long list of available downloads!

### Downloads

<a href="#"> adafruit-circuitpython-arduino_zero-0.9.3.bin</a>	184 KB
<a href="#"> adafruit-circuitpython-feather_huzzah-0.9.3.bin</a>	574 KB
<a href="#"> adafruit-circuitpython-feather_m0_adalogger-0.9.3.bin</a>	184 KB
<a href="#"> adafruit-circuitpython-feather_m0_basic-0.9.3.bin</a>	184 KB
<a href="#"> adafruit-circuitpython-feather_m0_express-0.9.3.bin</a>	211 KB
<a href="#"> adafruit-circuitpython-feather_m0_express-0.9.3.uf2</a>	423 KB
<a href="#"> Source code (zip)</a>	
<a href="#"> Source code (tar.gz)</a>	

See below for which file to download!

# Flashing

---

Flashing is the process of updating the CircuitPython core. It isn't needed for updating your own code. **There are two available methods:** **UF2** and **bossac** UF2 flashing is only available on Express boards, they have a UF2-capable beta bootloader. Flashing via bossac is possible with both the Express bootloader and the original "Arduino" one. We recommend using UF2 if you can. If UF2 fails, or is not available, try bossac.

Regardless of what method you use, you must first get the board into the bootloader mode. This is done by double clicking the reset button. The board is in bootloader mode when the red led fades in and out. Boards with the status neopixel will also show USB status while the red led fades. Green means USB worked while red means the board couldn't talk to the computer. The first step to troubleshooting a red neopixel is trying a different USB cable to make sure its not a charge-only cable.

## Flashing UF2

---

Adafruit Express boards come with a new beta bootloader called UF2 that makes flashing CircuitPython even easier than before. This beta bootloader allows you to drag so-called ".uf2" type files onto the BOOT drive. [For more information, check out our UF2 bootloader page.](#)

Start by double-clicking the reset button while it is plugged into your computer. You should see a new disk drive 'pop up' called **METROBOOT** or **FEATHERBOOT** or similar, and the NeoPixel on your board glow green.

The drive will contain a few files. If you want to make a 'backup' of the current firmware on the device, drag-off and save the **CURRENT.UF2** file. Other than that, you can ignore the index.htm and info\_uf2.txt files. They cannot be deleted and are only for informational purposes.

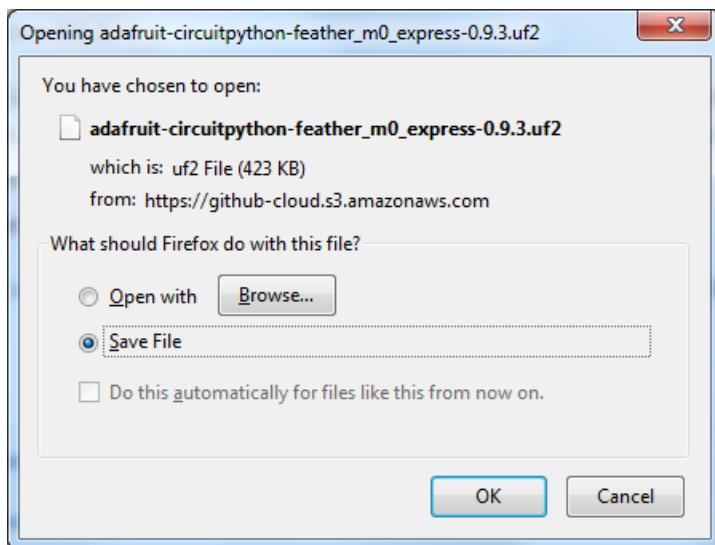
Next up, find the **Feather M0 Express UF2** or **Metro M0 Express UF2** file in the github downloads list:

## Downloads

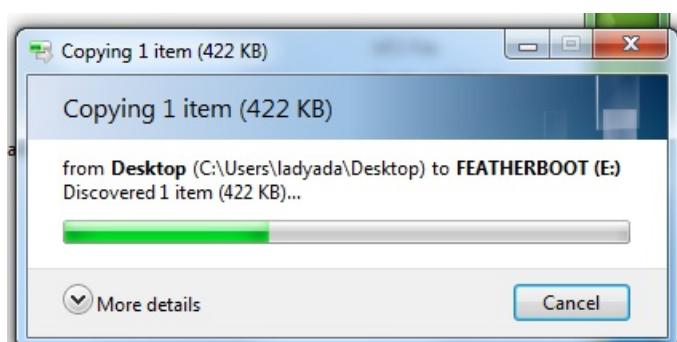
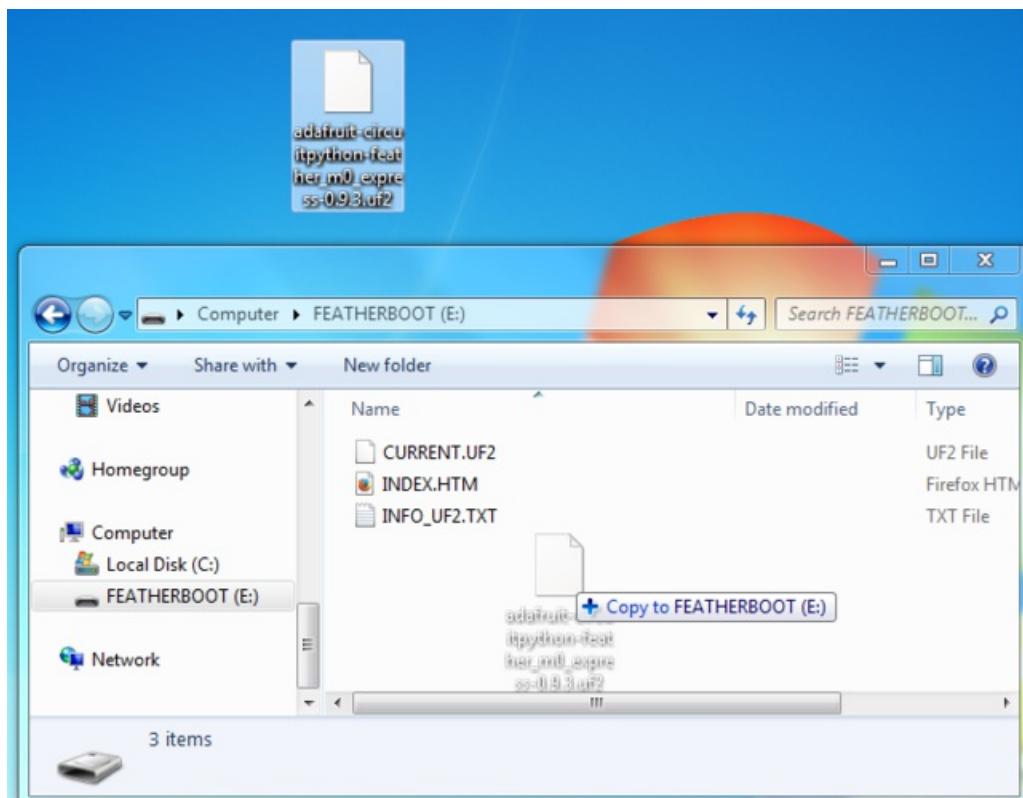
---

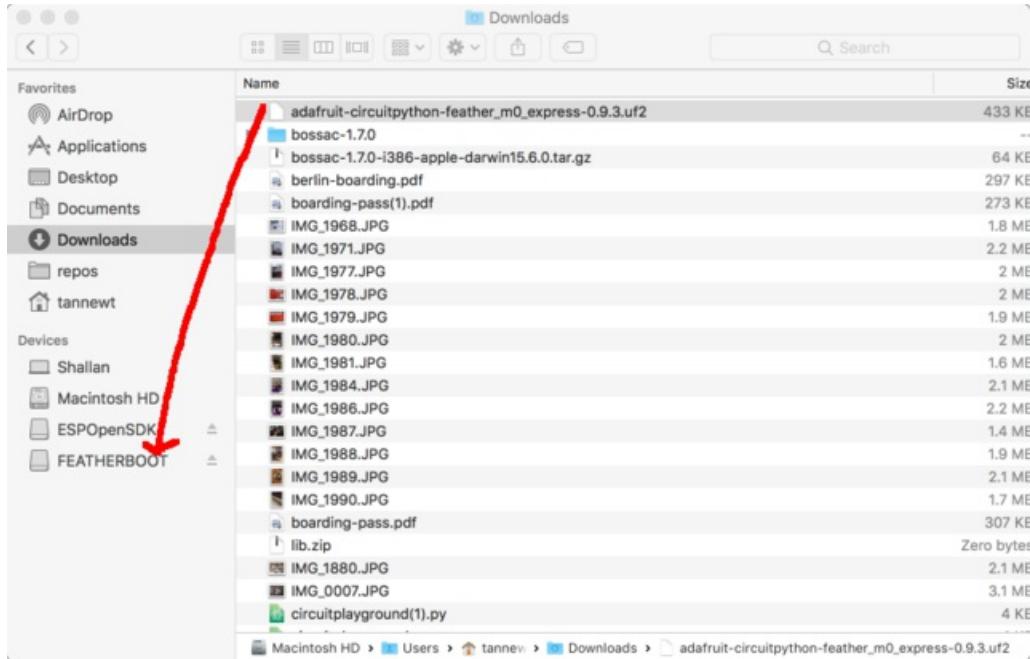
-  [adafruit-circuitpython-arduino\\_zero-0.9.7.bin](#)
-  [adafruit-circuitpython-feather\\_huzzah-0.9.7.bin](#)
-  [adafruit-circuitpython-feather\\_m0\\_adalogger-0.9.7.bin](#)
-  [adafruit-circuitpython-feather\\_m0\\_basic-0.9.7.bin](#)
-  [adafruit-circuitpython-feather\\_m0\\_express-0.9.7.bin](#)
-  [adafruit-circuitpython-feather\\_m0\\_express-0.9.7.uf2](#)
-  [adafruit-circuitpython-metro\\_m0\\_express-0.9.7.bin](#)
-  [adafruit-circuitpython-metro\\_m0\\_express-0.9.7.uf2](#)

Click to download and save the file onto your Desktop or somewhere else you can find it



Then drag the **uf2** file into the **BOOT** drive





Once the full file has been received, the board will automatically restart into CircuitPython. Your computer may warn about ejecting the drive early, if it does, simply ignore it because the board made sure the file was received ok.

## Flashing with BOSSAC

This method is only recommended if you can't use UF2 for some reason!

To flash with bossac (BOSSA's command line tool) first download the latest version from [here](#). 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`.

```
bossac -e -w -v -R ~/Downloads/adafruit-circuitpython-feather_m0_express-0.9.3.bin
```

This will erase the chip, write the given file, verify the write and reset the board. After reset, CircuitPython should be running. Express boards may cause a warning of an early eject of a USB drive but just ignore it. Nothing important was being written to the drive.

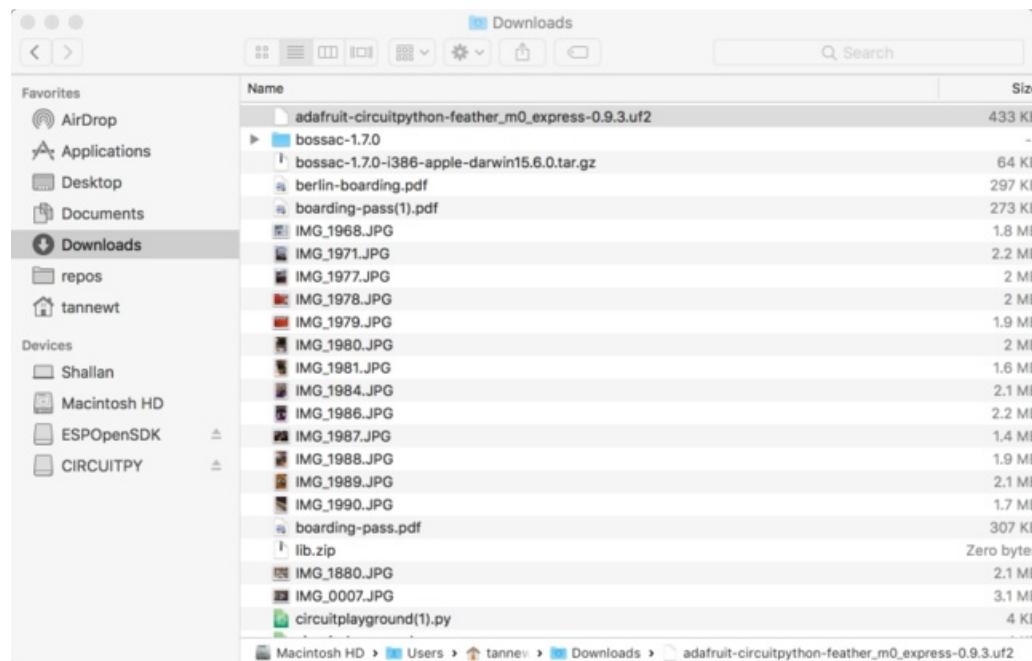
```
(venv) tannewt@shallan:~/Downloads/bossac-1.7.0 $ ./bossac -e -w -v -R ~/Downloads/adafruit-circuitpython-feather_m0_express-0.9.3.bin
Device found on cu.usbmodem1441
Atmel SMART device 0x1001000a found
Erase flash
done in 0.658 seconds

Write 216080 bytes to flash (3377 pages)
[=====] 100% (3377/3377 pages)
done in 1.371 seconds

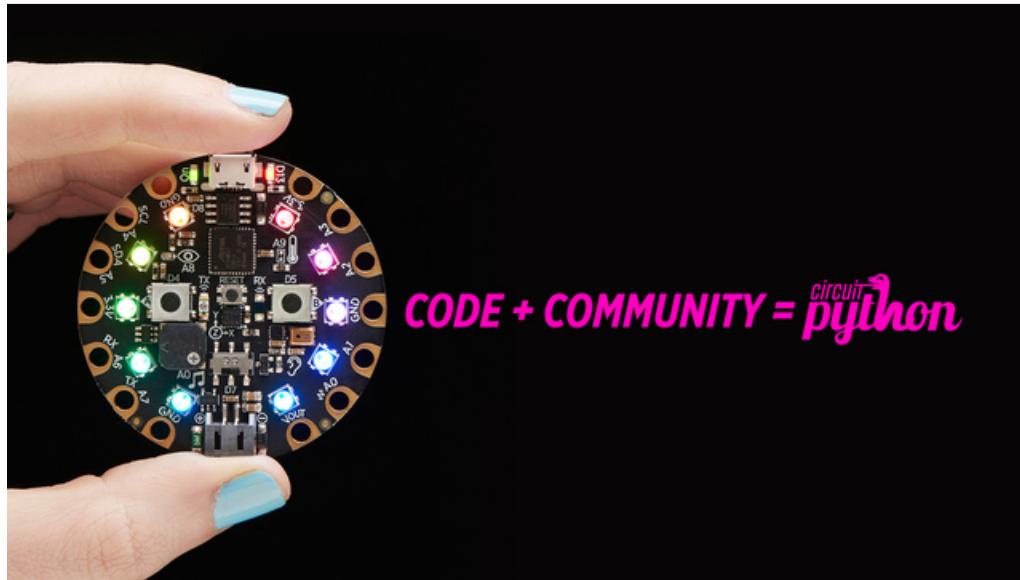
Verify 216080 bytes of flash with checksum.
Verify successful
done in 0.305 seconds
CPU reset.
(venv) tannewt@shallan:~/Downloads/bossac-1.7.0 $
```

## After flashing

After a successful flash by bossac or UF2 you should see a CIRCUITPY drive appear.



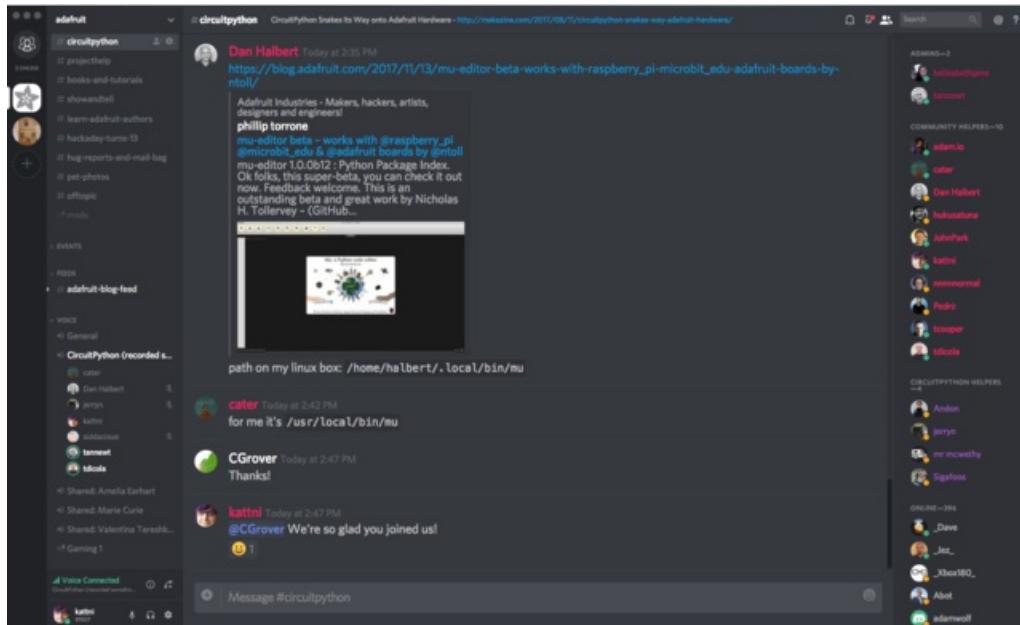
# Welcome to the Community!



CircuitPython is a programming language that's super simple to get started with and great for learning. It runs on microcontrollers and works out of the box. You can plug it in and get started with any text editor. The best part? CircuitPython comes with an amazing, supportive community.

Everyone is welcome! CircuitPython is Open Source. This means it's available for anyone to use, edit, copy and improve upon. This also means CircuitPython becomes better because of you being a part of it. It doesn't matter whether this is your first microcontroller board or you're a computer engineer, you have something important to offer the Adafruit CircuitPython community. We're going to highlight some of the many ways you can be a part of it!

## Adafruit Discord



The Adafruit Discord server is the best place to start. Discord is where the community comes together to volunteer and provide live support of all kinds. From general discussion to detailed problem solving, and everything in between,

Discord is a digital maker space with makers from around the world.

There are many different channels so you can choose the one best suited to your needs. Each channel is shown on Discord as "#channelname". There's the #projecthelp channel for assistance with your current project or help coming up with ideas for your next one. There's the #showandtell channel for showing off your newest creation. Don't be afraid to ask a question in any channel! If you're unsure, #general is a great place to start. If another channel is more likely to provide you with a better answer, someone will guide you.

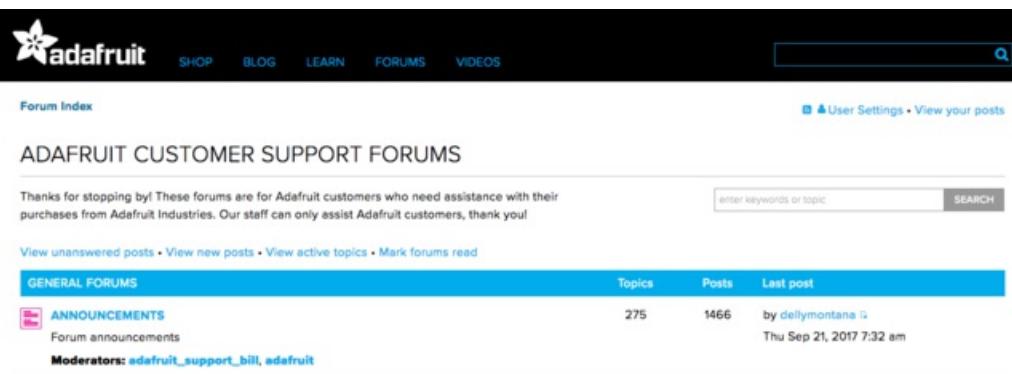
The CircuitPython channel is where to go with your CircuitPython questions. #circuitpython is there for new users and developers alike so feel free to ask a question or post a comment! Everyone of any experience level is welcome to join in on the conversation. We'd love to hear what you have to say!

The easiest way to contribute to the community is to assist others on Discord. Supporting others doesn't always mean answering questions. Join in celebrating successes! Celebrate your mistakes! Sometimes just hearing that someone else has gone through a similar struggle can be enough to keep a maker moving forward.

The Adafruit Discord is the 24x7x365 hackerspace that you can bring your granddaughter to.

Visit <https://adafru.it/discord> to sign up for Discord. We're looking forward to meeting you!

## Adafruit Forums



The screenshot shows the Adafruit Customer Support Forums homepage. At the top, there's a navigation bar with links for SHOP, BLOG, LEARN, FORUMS, and VIDEOS. On the right side of the header is a search bar. Below the header, there's a "Forum Index" link and a "User Settings" link with a "View your posts" option. The main title "ADAFRUIT CUSTOMER SUPPORT FORUMS" is centered above a message thanking visitors for stopping by. To the right of the message is a search bar and a "SEARCH" button. Below the message, there are links for "View unanswered posts", "View new posts", "View active topics", and "Mark forums read". A table lists forum categories: "GENERAL FORUMS" (Topics: 275, Posts: 1466, Last post: by delliymontana on Thu Sep 21, 2017 7:32 am), "ANNOUNCEMENTS" (Topics: 275, Posts: 1466, Last post: by delliymontana on Thu Sep 21, 2017 7:32 am), and "Moderators: adafruit\_support\_bill, edefruit".

The [Adafruit Forums](#) are the perfect place for support. Adafruit has wonderful paid support folks to answer any questions you may have. Whether your hardware is giving you issues or your code doesn't seem to be working, the forums are always there for you to ask. You need an Adafruit account to post to the forums. You can use the same account you use to order from Adafruit.

While Discord may provide you with quicker responses than the forums, the forums are a more reliable source of information. If you want to be certain you're getting an Adafruit-supported answer, the forums are the best place to be.

There are forum categories that cover all kinds of topics, including everything Adafruit. The [Adafruit CircuitPython](#) and [MicroPython](#) category under "Supported Products & Projects" is the best place to post your CircuitPython questions.

## Adafruit CircuitPython and MicroPython

Moderators: [adafruit\\_support\\_bill](#), [adafruit](#)

## Forum rules

Adafruit MicroPython is currently EXPERIMENTAL and BETA - Please visit <https://learn.adafruit.com/category/micropython> and <http://forum.micropython.org/> in addition to our section here!

[POST A TOPIC](#)

Search this forum...

[SEARCH](#)[Mark topics read](#) • 179 topics • Page 1 of 4 • [1234](#)

Please be positive and constructive with your questions and comments.

ANNOUNCEMENTS	Replies	Views	Last post
<a href="#">CIRCUITYTHON 2.1.0 RELEASED!</a> by <a href="#">danhalbert</a> » Wed Oct 18, 2017 12:47 am	1	111	by <a href="#">danhalbert</a> Fri Oct 20, 2017 2:43 am

Be sure to include the steps you took to get to where you are. If it involves wiring, post a picture! If your code is giving you trouble, include your code in your post! These are great ways to make sure that there's enough information to help you with your issue.

You might think you're just getting started, but you definitely know something that someone else doesn't. The great thing about the forums is that you can help others too! Everyone is welcome and encouraged to provide constructive feedback to any of the posted questions. This is an excellent way to contribute to the community and share your knowledge!

## Adafruit Github

CircuitPython - a Python implementation for teaching coding with microcontrollers

circuitpython

9,856 commits 32 branches 73 releases 206 contributors

Whether you're just beginning or are life-long programmer who would like to contribute, there are ways for everyone to be a part of building CircuitPython. GitHub is the best source of ways to contribute to [CircuitPython](#) itself. If you need an account, visit <https://github.com/> and sign up.

If you're new to GitHub or programming in general, there are great opportunities for you. Head over to [adafruit/circuitpython](#) on GitHub, click on "[Issues](#)", and you'll find a list that includes issues labeled "[good first issue](#)". These are things we've identified as something that someone with any level of experience can help with. These issues include options like updating documentation, providing feedback, and fixing simple bugs.

<input type="checkbox"/> ⓘ OneWire BusDevice driver <a href="#">good first issue</a> #338 opened 29 days ago by tannewt	Long term	2
<input type="checkbox"/> ⓘ Feather M0 Adalogger does not have D8 or D7 <a href="#">good first issue</a> #323 opened on Oct 13 by ladyada	3.0	7
<input type="checkbox"/> ⓘ Audit and fix native API for methods that accept and ignore extra args. <a href="#">good first issue</a> #321 opened on Oct 12 by tannewt	Long term	

Already experienced and looking for a challenge? Checkout the rest of the issues list and you'll find plenty of ways to contribute. You'll find everything from new driver requests to core module updates. There's plenty of opportunities for everyone at any level!

When working with CircuitPython, you may find problems. If you find a bug, that's great! We love bugs! Posting a detailed issue to GitHub is an invaluable way to contribute to improving CircuitPython. Be sure to include the steps to replicate the issue as well as any other information you think is relevant. The more detail, the better!

Testing new software is easy and incredibly helpful. Simply load the newest version of CircuitPython or a library onto your CircuitPython hardware, and use it. Let us know about any problems you find by posting a new issue to GitHub. Software testing on both current and beta releases is a very important part of contributing CircuitPython. We can't possibly find all the problems ourselves! We need your help to make CircuitPython even better.

On GitHub, you can submit feature requests, provide feedback, report problems and much more. If you have questions, remember that Discord and the Forums are both there for help!

## ReadTheDocs

The screenshot shows the Adafruit CircuitPython ReadTheDocs website. The left sidebar has sections for 'API AND USAGE' and 'Core Modules'. Under 'Core Modules', there is a 'Support Matrix' and a 'Modules' section which lists several modules: 'analogio' (Analog hardware support), 'audiobusio' (Support for audio input and output over digital bus), 'audioio' (Support for audio input and output), and 'bitbangio' (Digital protocols implemented by the CPU). The main content area is titled 'audioio – Support for audio input and output'. It contains a brief description of the module, a 'Libraries' section with a single item '• [AudioOut](#) – Output an analog audio signal', and a note about class lifetime. Navigation buttons for 'Previous' and 'Next' are at the bottom.

[ReadTheDocs](#) is an excellent resource for a more in-depth look at CircuitPython. This is where you'll find things like API documentation and details about core modules. There is also a Design Guide that includes contribution guidelines for CircuitPython.

RTD gives you access to a low-level look at CircuitPython. There are details about each of the [core modules](#). Each module lists the available libraries. Each module library page lists the available parameters and an explanation for each. In many cases, you'll find quick code examples to help you understand how the modules and parameters work, however it won't have detailed explanations like the Learn Guides. If you want help understanding what's going on behind the scenes in any CircuitPython code you're writing, ReadTheDocs is there to help!

Here is `blink.py`:

```
import digitalio
from board import *
import time

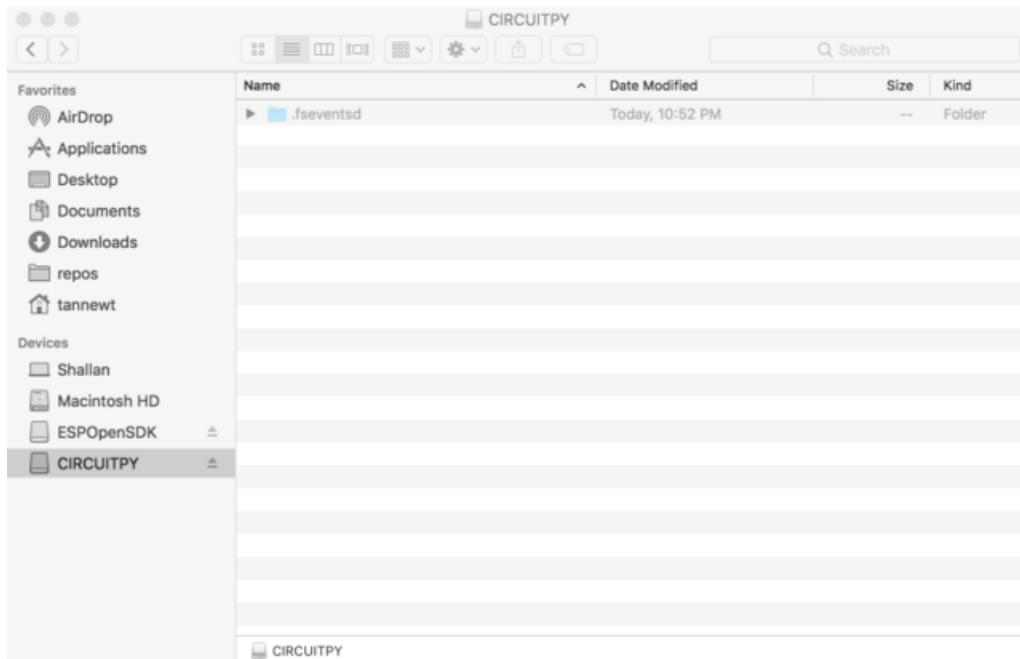
led = digitalio.DigitalInOut(D13)
led.direction = digitalio.Direction.OUTPUT
while True:
    led.value = True
    time.sleep(0.1)
    led.value = False
    time.sleep(0.1)
```

## CircuitPython Blinky

Let's get blinky going with CircuitPython to explore the way we can write code and confirm everything is working as expected.

### code.py

After plugging in a board with CircuitPython into your computer a CIRCUITPY drive will appear. At first, the drive may be empty but you can create and edit files on it just like you would on a USB drive. On here, you can save a **code.py** (**code.txt** and **main.py** also work) file to run every time the board resets. This is the CircuitPython equivalent of an Arduino sketch. However, all of the compiling is done on the board itself. *All you need to do is edit the file.*



So, fire up your favorite text editor, such as Notepad on Windows,TextEdit on Mac or [download Atom](#) (my favorite), and create a new file. In the file copy this:

```
import digitalio
import board
import time

led = digitalio.DigitalInOut(board.D13)
led.direction = digitalio.Direction.OUTPUT
while True:
    led.value = not led.value
    time.sleep(0.5)
```

Now, save the file to the drive as `code.py` (`main.py` or `code.txt` also works). After a brief time, the board's red LED should begin to flash every second.

Do not click the RESET button after saving your code file! It could cause the computer to not-finish writing your code to disk. Just wait a few seconds and it should automatically restart the python code for you!

### Status LED (Gemma/Trinket/Metro/Feather)

If you have a Gemma, Trinket, Metro or Feather running CircuitPython, there's a single RGB LED on the board to help you know what's up. While `code.py` is running the status neopixel will be solid green. After it is finished, the neopixel will fade green on success or flash an error code on failure. Red flashes happen when data is written to the drive.

Circuit Playground Express does not have this status LED

## Debugging

Did the status LED flash a bunch of colors at you? You may have an error in your code. Don't worry it happens to everyone. Python code is checked when you run it rather than before like Arduino does when it compiles. To see the CircuitPython error you'll need to connect to the serial output (like Arduino's serial monitor).

See [this guide](#) for detailed instructions.

If you are new to Python try googling the error first, if that doesn't find an answer feel free to drop by the [support forum](#).

## Libraries

Using libraries with CircuitPython is also super easy. Simply drag and drop libraries onto the CIRCUITPY drive or into a `lib` folder on the drive to keep it tidy.

Find CircuitPython libraries on GitHub using the [topic](#) and through our [tutorials](#).

Make sure the libraries are for CircuitPython and not MicroPython. There are some differences that may cause it to not work as expected.

## More info

---

- [Guides and Tutorials](#)
- [API Reference](#)
- [Adafruit forum](#)
- [Libraries](#)

## 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.

## Are you using Mu?

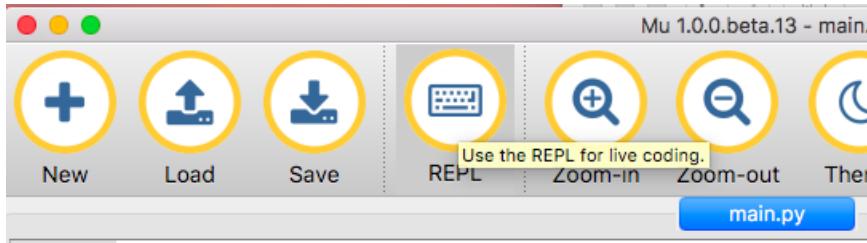
If so, good news! The serial console is built into Mu and will autodetect your board making using the REPL *really really easy*.

Please note that Mu does yet not work with nRF52 or ESP8266-based CircuitPython boards, skip down to the next section for details on using a terminal program.



First, make sure your CircuitPython board is plugged in. If you are using Windows 7, make sure you installed the drivers (<https://adafru.it/Amd>).

Once in Mu, look for the **REPL** button in the menu and click it



```
1 import board
2 import digitalio
3 import time
4
5 led = digitalio.DigitalInOut(board.D13)
6 led.direction = digitalio.Direction.OUTPUT
7
8 while True:
9     led.value = True
```

Traceback (most recent call last):
File "code.py", line 10, in <module>
KeyboardInterrupt:
Auto-reload is on. Simply save files over USB to run them or enter REPL to disable.
Press any key to enter the REPL. Use CTRL-D to reload.
soft reboot

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

The editor window will split in half.

The bottom half is your serial output/input. You can see text *from* the CircuitPython board as well as send text *to* the board.

## Using Something Else?

If you're not using Mu to edit, are using ESP8266 or nRF52 CircuitPython, or if for some reason you are not a fan of the built in serial console, you can run the serial console as a separate program.

[Windows requires you to download a terminal program, check out this page for more details](#)

[Mac and Linux both have one built in, though other options are available for download, check this page for more details](#)

## 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:

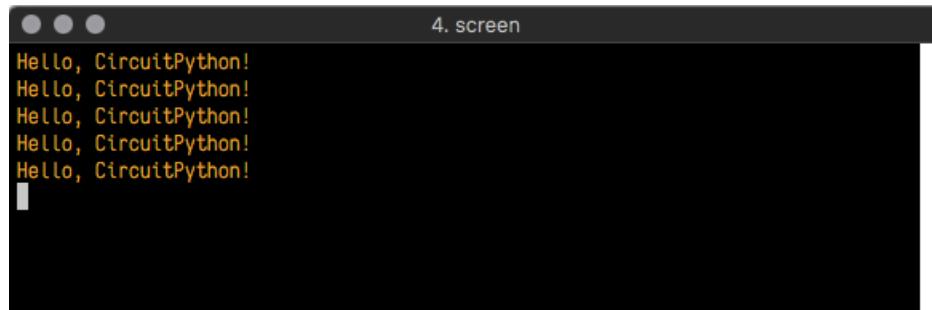
```
import board
import digitalio
import time

led = digitalio.DigitalInOut(board.D13)
led.direction = digitalio.Direction.OUTPUT

while True:
    print("Hello, CircuitPython!")
    led.value = True
    time.sleep(1)
    led.value = False
    time.sleep(1)
```

Save your file.

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



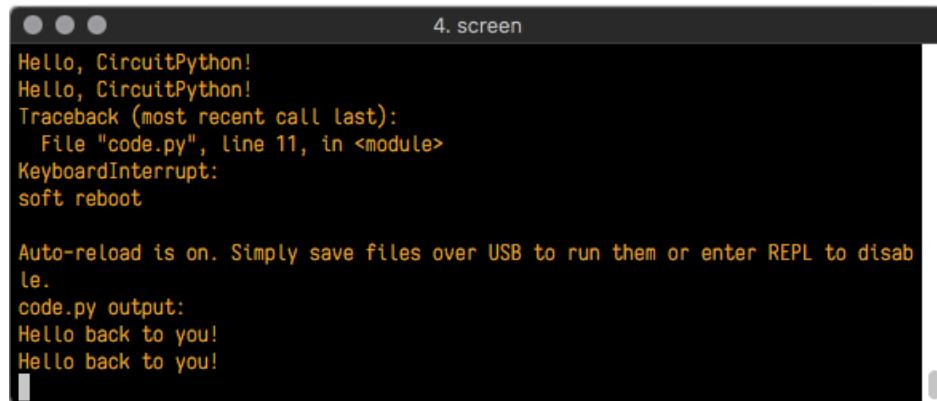
Excellent! Our print statement is showing up in our console! Try changing the printed text to something else.

```
code.py
1 import board
2 import digitalio
3 import time
4
5 led = digitalio.DigitalInOut(board.D13)
6 led.direction = digitalio.Direction.OUTPUT
7
8 while True:
9     print("Hello back to you!")
10    led.value = True
11    time.sleep(1)
12    led.value = False
13    time.sleep(1)
14
```

A screenshot of a code editor showing the code.py file. The code is identical to the previous example, except for the print statement which now says "Hello back to you!". The code editor has a dark background and a black border.

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!



```
Hello, CircuitPython!
Hello, CircuitPython!
Traceback (most recent call last):
  File "code.py", line 11, in <module>
KeyboardInterrupt:
soft reboot

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

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 `e` at the end of `True` from the line `led.value = True` so that it says `led.value = Tru`



```
code.py
1 import board
2 import digitalio
3 import time
4
5 led = digitalio.DigitalInOut(board.D13)
6 led.direction = digitalio.Direction.OUTPUT
7
8 while True:
9     print("Hello back to you!")
10    led.value = Tru
11    time.sleep(1)
12    led.value = False
13    time.sleep(1)
14
```

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!

```
Hello back to you!
Traceback (most recent call last):
  File "code.py", line 13, in <module>
KeyboardInterrupt:
soft reboot

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

code.py output:
Hello back to you!
Traceback (most recent call last):
  File "code.py", line 10, in <module>
NameError: name 'Tru' is not defined

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

The `Traceback (most recent call last):` is telling you that the last thing it was able to run was line 10 in your code. The next line is your error: `NameError: name 'Tru' is not defined`. This error might not mean a lot to you, but combined with knowing the issue is on line 10, it gives you a great place to start!

Go back to your code, and take a look at line 10. Obviously, you know what the problem is already. But if you didn't, you'd want to look at line 10 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 spelled True wrong. Fix the typo and save your file.

```
le.
code.py output:
Hello back to you!
Traceback (most recent call last):
  File "code.py", line 10, in <module>
NameError: name 'Tru' is not defined

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

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

code.py output:
Hello back to you!
Hello back to you!
```

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

The serial console will display any output generated by your code. Some sensors, such as a humidity sensor or a thermistor, receive data and you can use print statements to display that information. You can also use print statements for troubleshooting. If your code isn't working, and you want to know where it's failing, you can put print statements in various places to see where it stops printing.

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

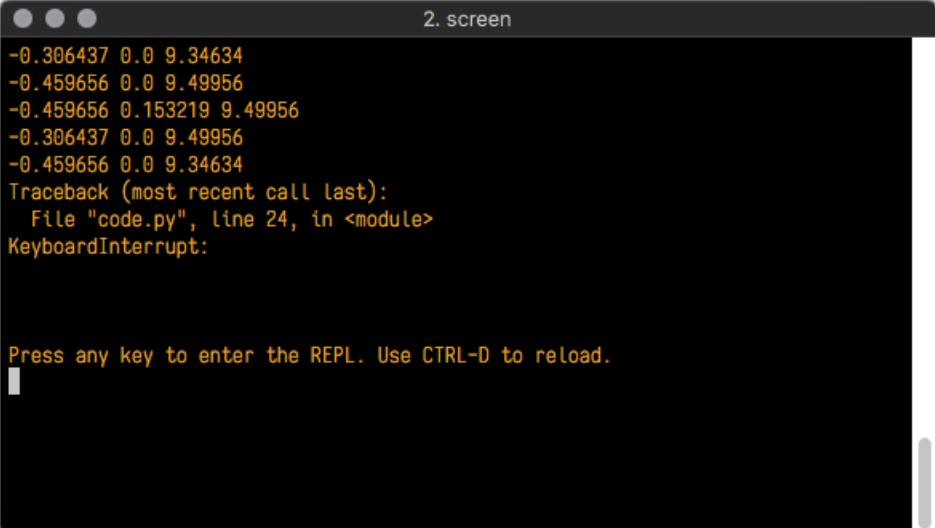
## The REPL

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

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.

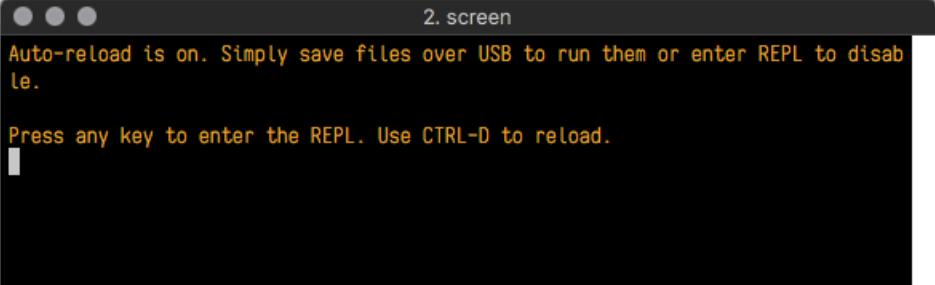
The **Traceback (most recent call last):** is telling you the last thing your board was doing before you pressed Ctrl + C and interrupted it. The **KeyboardInterrupt** is you pressing Ctrl + C. This information can be handy when troubleshooting, but for now, don't worry about it. Just note that it is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



```
-0.306437 0.0 9.34634
-0.459656 0.0 9.49956
-0.459656 0.153219 9.49956
-0.306437 0.0 9.49956
-0.459656 0.0 9.34634
Traceback (most recent call last):
  File "code.py", line 24, in <module>
KeyboardInterrupt:

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

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.



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

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

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

```
Adafruit CircuitPython 2.1.0 on 2017-10-17; Adafruit CircuitPlayground Express with samd21g18
>>> 
```

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

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

```
Adafruit CircuitPython 2.1.0 on 2017-10-17; Adafruit CircuitPlayground Express with samd21g18
```

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.

```
Adafruit CircuitPython 2.1.0 on 2017-10-17; Adafruit Feather M0 Express with samd21
g18
>>> help() 
```

Then press enter. You should then see a message.

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

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

Adafruit CircuitPython 2.1.0 on 2017-10-17; Adafruit CircuitPlayground Express with samd21g18
>>> 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")`.

>>> 
```

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        framebuffer 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        framebuffer 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.



```
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            framebuffer  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
>>> dir(board)
['A0', 'A1', 'A2', 'A3', 'A4', 'A5', 'SCK', 'MOSI', 'MISO', 'D0', 'RX', 'D1', 'TX',
 'SDA', 'SCL', 'D5', 'D6', 'D9', 'D10', 'D11', 'D12', 'D13', 'NEOPIXEL']
>>> 
```

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.

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

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

You can write single lines of code that run stand-alone. You can also write entire programs into the REPL to test them. 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

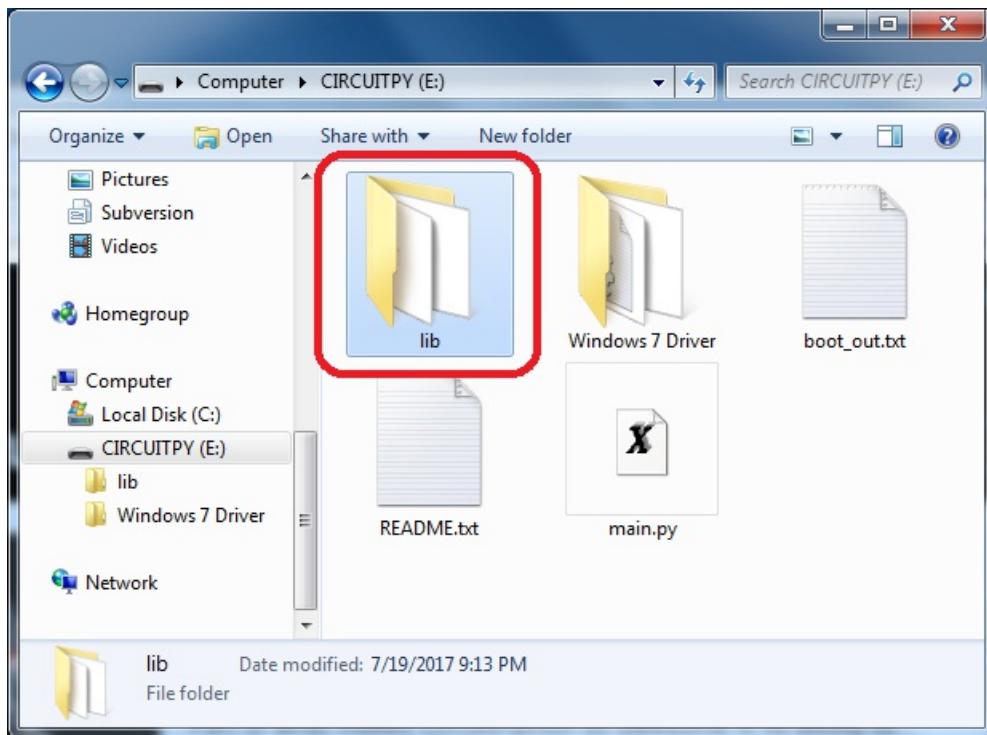
When you're ready to leave the REPL and return to the serial console, simply press **Ctrl + D**. This will reload your board and reenter the serial console. You will restart the program you had running before entering the REPL. In the console window, you'll see any output from the program you had running. 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

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, its 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](#) 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 2.x Bundle release by clicking this button:

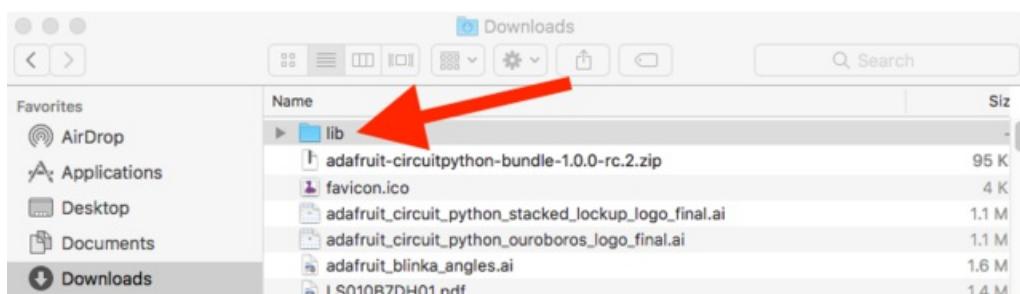
Click for the Latest Adafruit CircuitPython Library  
Bundle Release

<https://adafru.it/AgR>

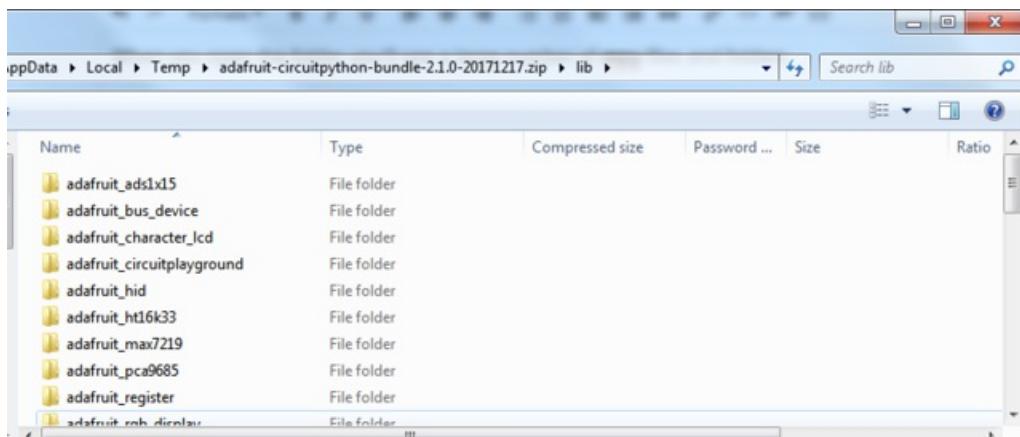
If you need another version, [you can also visit the bundle release page](#) which will let you select exactly what version you're looking for, as well as information about changes.

Either way, download the version that matches your CircuitPython run-time. For example, if you're running v2.2 download the v2 bundle. If you're running 3.0, download the v3 bundle. There's also a `py` bundle which contains the uncompressed python files, you probably *don't* want that!

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



## 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` directory can be copied directly to the CIRCUITPY drive.

Just drag the entire `lib` folder into the CIRCUITPY drive, and 'replace' any old files if your operating system prompts you

## 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`.

### 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.**

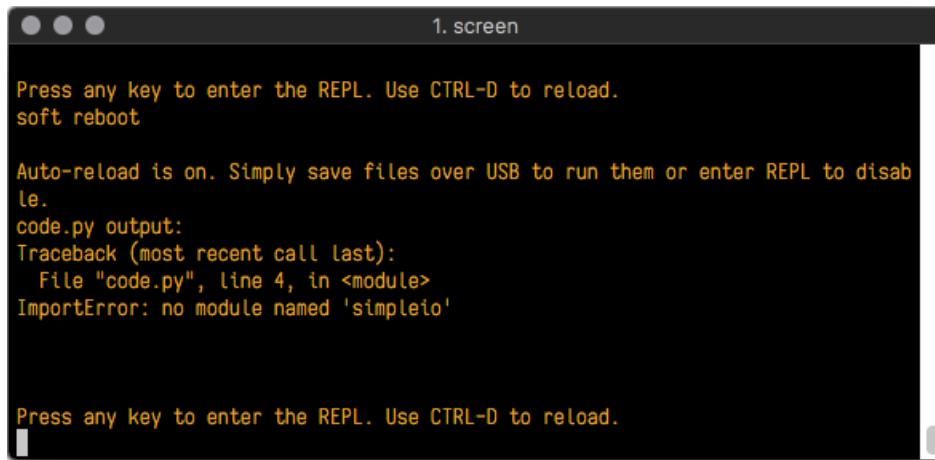
Let's use a modified version of the blinky example.

```
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.



```
1. screen

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

Auto-reload is on. Simply save files over USB to run them or enter REPL to disable.
code.py output:
Traceback (most recent call last):
  File "code.py", line 4, in <module>
    ImportError: no module named 'simpleio'

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

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.



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

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

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

## Library Install on Non-Express Boards

If you have 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

Libraries 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, 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 folder to your CIRUCITPY 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.

## CircuitPython Built-Ins

CircuitPython comes 'with the kitchen sink' - *a lot* of the things you know and love about classic Python 3 (sometimes called CPython) already work. There are a few things that don't but we'll try to keep this list updated as we add more capabilities!

This is not an exhaustive list! It's just some of the many features you can use

### Things that are Built In and Work

#### flow control

All the usual `if`, `elif`, `else`, `for`, `while` ... work just as expected

#### math

`import math` will give you a range of handy mathematical functions

```
>>> dir(math)
['__name__', 'e', 'pi', 'sqrt', 'pow', 'exp', 'log', 'cos', 'sin', 'tan', 'acos', 'asin', 'atan', 'atan2', 'ceil', 'copysign', 'fabs', 'floor', 'fmod', 'frexp',
 'ldexp', 'modf', 'isfinite', 'isinf', 'isnan', 'trunc', 'radians', 'degrees']
```

CircuitPython supports 30-bit wide floating point values so you can use int's and float's whenever you expect

#### tuples, lists, arrays, and dictionaries

You can organize data in (), []'s , and {}'s including strings, objects, floats, etc

#### classes/objects and functions

We use objects and functions extensively in our libraries so check out one of our many examples like this [MCP9808 library](#) for class examples

#### lambdas

Yep! You can create function-functions with lambda just the way you like em:

```
>>> g = lambda x: x**2
>>> g(8)
64
```

### Things to watch out for!

- The wide body of python libraries have not been ported over, so while we wish you could `import numpy` , numpy isn't available. So you may have to port some code over yourself!
- For the ATSAMD21 based boards (Feather M0, Metro M0, Trinket M0, Gemma M0, Circuit PlayGround Express) there's a limited amount of RAM, we've found you can have about 250-ish lines of python (that's with various libraries) before you hit MemoryErrors. The upcoming SAMD51 chipset will help with that a ton but its not yet available)
- Non-Express boards like Trinket M0 and Gemma M0 and non-Express Feathers do not include all of the hardware support. For example, `audioio` and `bitbangio` are not included.
- Integers can only be up to 31 bits. Integers of unlimited size are not supported.

- We keep up with MicroPython stable releases, so check out the core 'differences' they document here.

## 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.

### 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](#) installed. Feather M0 Basic, Feather M0 Adalogger, and similar boards use a regular Arduino-compatible bootloader, which does not show a `boardnameBOOT` drive.

#### MakeCode

If you are running a [MakeCode](#) 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.

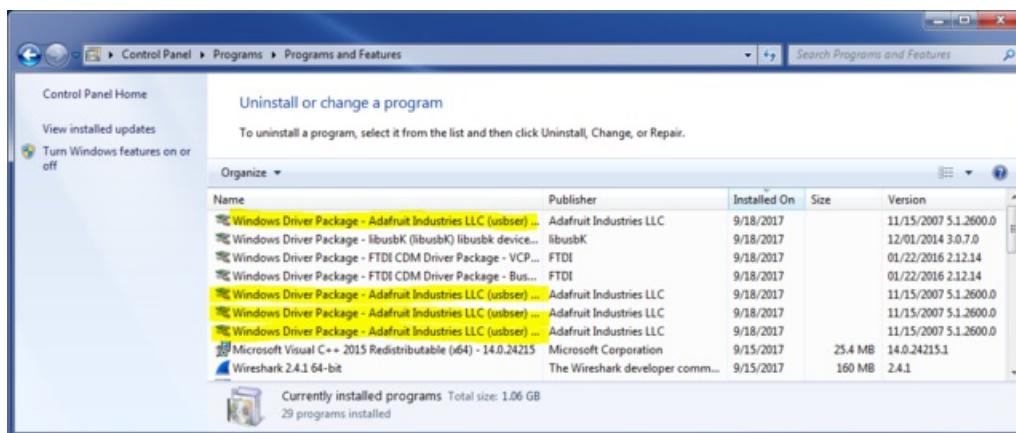
#### Windows 10

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 `boardnameBOOT` drive problem on Windows 7. To resolve this, first uninstall the old versions of the drivers:

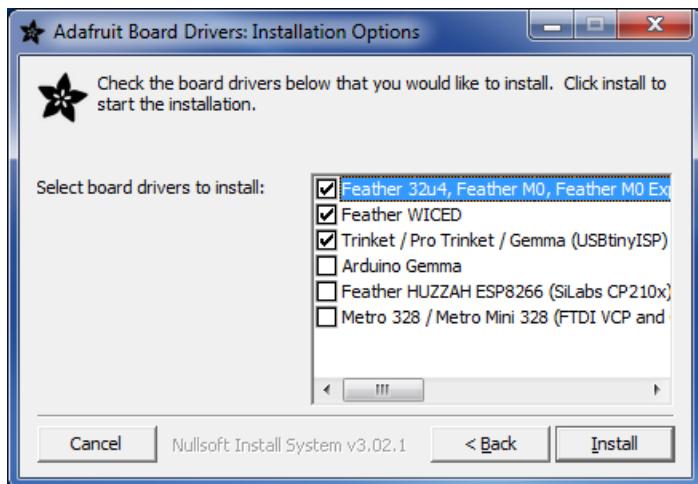
- Unplug any boards. In **Uninstall or Change a Program (Control Panel->Programs->Uninstall a program)**, uninstall everything named "Windows Driver Package - Adafruit Industries LLC ...".



- Now install the new 2.0.0.0 (or higher) Adafruit Windows Drivers Package:

[Download Latest Drivers](#)

- 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](#) or on the [Adafruit Discord](#) if this does not work for you!

## CircuitPython RGB Status Light

The Feather M0 Express, Metro M0 Express, Gemma M0, and Trinket M0 all have a single NeoPixel or DotStar RGB LED on the board that indicates the status of 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
- YELLOW**: Circuit Python is in safe mode: it crashed and restarted
- WHITE**: REPL is running
- BLUE**: Circuit Python is starting up

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**: `OSSError`
- 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.

## 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 become corrupted.

This happens most often when the CIRCUITPY disk is not safely ejected before being reset by the button or being disconnected from USB. 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.

### For the Circuit Playground Express, Feather M0 Express, and Metro M0 Express:

1. Download the correct erase file:

Circuit Playground Express

<https://adafruit.it/AdI>

Feather M0 Express

<https://adafruit.it/AdJ>

Metro M0 Express

<https://adafruit.it/AdK>

2. Double-click the reset button on the board to bring up the *boardnameBOOT* drive.
3. Drag the erase .uf2 file to the *boardnameBOOT* drive.
4. The onboard NeoPixel will turn blue, indicating the erase has started.
5. After approximately 15 seconds, the NeoPixel will start flashing green.
6. Double-click the reset button on the board to bring up the *boardnameBOOT* drive.
7. Drag the appropriate latest release of CircuitPython .uf2 file to the *boardnameBOOT* drive.

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

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

If you haven't already downloaded the latest release of CircuitPython for your board, you can find it [here](#).

### For the Gemma M0, Trinket M0, Feather M0: Basic (Proto) and Feather Adalogger:

1. Download the erase file:

Gemma M0, Trinket M0, Feather M0 Basic,  
Feather Adalogger

2. Double-click the reset button on the board to bring up the `boardnameBOOT` drive.
3. Drag the erase `.uf2` file to the `boardnameBOOT` drive.
4. The boot LED will start flashing again, and the `boardnameBOOT` drive will reappear.
5. Drag the appropriate latest release CircuitPython `.uf2` file to the `boardnameBOOT` drive.

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

If you haven't already downloaded the latest version of CircuitPython for your board, you can find it [here](#).

## 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.

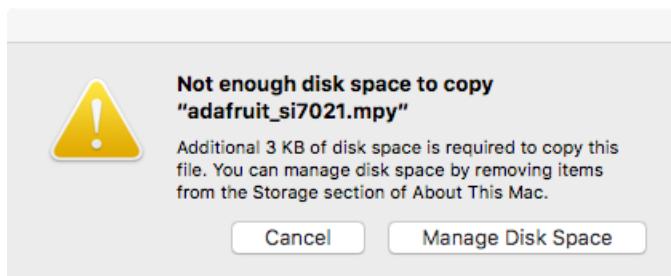
Delete something!

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](#) 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 .{,_}{{fsevents,Spotlight-V*,Trashes}
mkdir fsevents
touch fsevents/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 **afoo.mpy** file to the board use a command like:

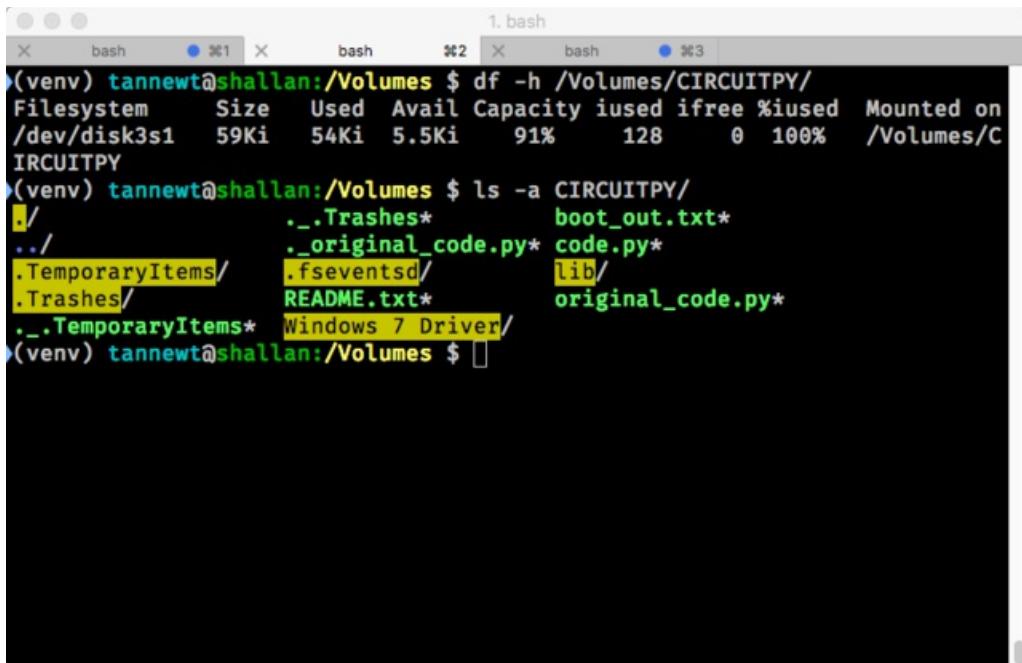
```
cp -X foo.mpy /Volumes/CIRCUITPY
```

Or to copy a folder and all of its child files/folders use a command like:

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

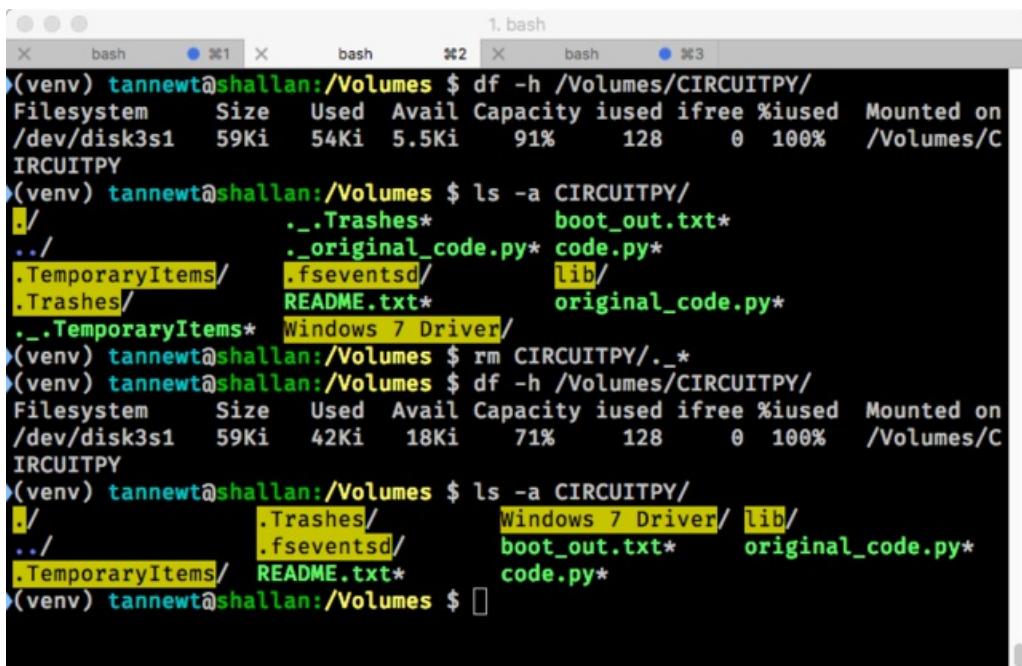
### Other Mac OSX Space-Saving Tips

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:



```
(venv) tannewt@shallan:/Volumes/CIRCUITPY/ $ df -h /Volumes/CIRCUITPY/
Filesystem      Size   Used  Avail Capacity iused ifree %iused  Mounted on
/dev/disk3s1    59Ki   54Ki   5.5Ki   91%     128     0  100%   /Volumes/C
IRCUITPY
(venv) tannewt@shallan:/Volumes/CIRCUITPY/ $ ls -a
./              ._.Trashes*      boot_out.txt*
../             ._.original_code.py* code.py*
.TemporaryItems/ fsevents.d/ lib/
.Trashes/        README.txt*    original_code.py*
._.TemporaryItems* Windows 7 Driver/
(venv) tannewt@shallan:/Volumes/CIRCUITPY/ $
```

Lets remove the `..` files first.



```
(venv) tannewt@shallan:/Volumes/CIRCUITPY/ $ df -h /Volumes/CIRCUITPY/
Filesystem      Size   Used  Avail Capacity iused ifree %iused  Mounted on
/dev/disk3s1    59Ki   54Ki   5.5Ki   91%     128     0  100%   /Volumes/C
IRCUITPY
(venv) tannewt@shallan:/Volumes/CIRCUITPY/ $ ls -a
./              ._.Trashes*      boot_out.txt*
../             ._.original_code.py* code.py*
.TemporaryItems/ fsevents.d/ lib/
.Trashes/        README.txt*    original_code.py*
._.TemporaryItems* Windows 7 Driver/
(venv) tannewt@shallan:/Volumes/CIRCUITPY/ $ rm CIRCUITPY/..
(venv) tannewt@shallan:/Volumes/CIRCUITPY/ $ df -h /Volumes/CIRCUITPY/
Filesystem      Size   Used  Avail Capacity iused ifree %iused  Mounted on
/dev/disk3s1    59Ki   42Ki   18Ki    71%     128     0  100%   /Volumes/C
IRCUITPY
(venv) tannewt@shallan:/Volumes/CIRCUITPY/ $ ls -a
./              .Trashes/      Windows 7 Driver/ lib/
../             fsevents.d/    boot_out.txt*   original_code.py*
.TemporaryItems/ README.txt*  code.py*
(venv) tannewt@shallan:/Volumes/CIRCUITPY/ $
```

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

# Downloads

## Files

- [ATSAMD21 Datasheet](#) (the main chip on the Metro M0)
- Fritzing object in the Adafruit Fritzing Library

CircuitPython 2.2 Metro M0 Demo Files.zip

<https://adafru.it/Aql>

## Schematic & Fabrication Print

