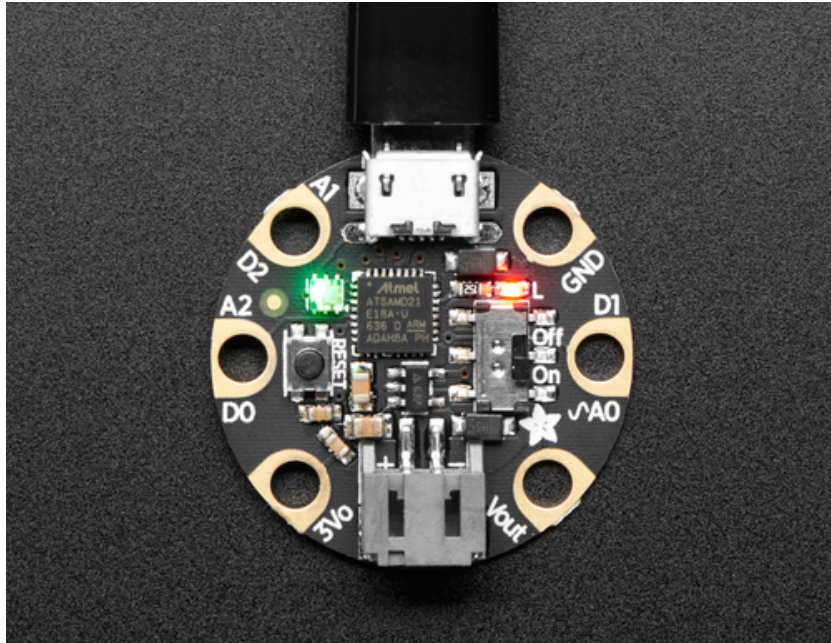


Adafruit Gemma M0

Created by lady ada



Last updated on 2017-08-01 09:22:36 PM UTC

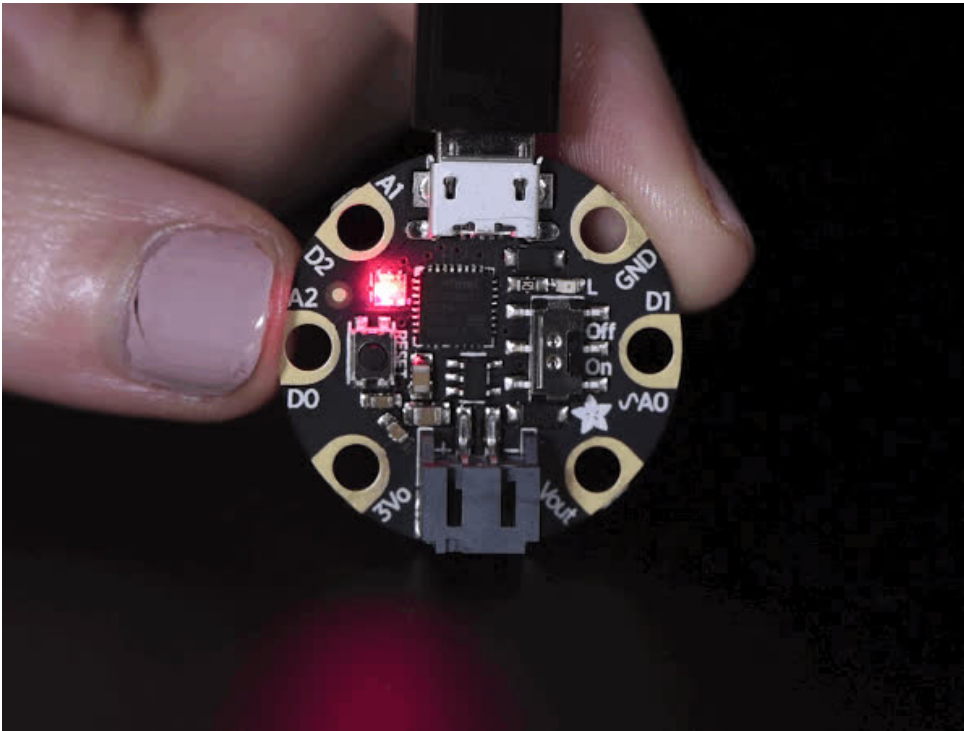
Guide Contents

Guide Contents	2
Overview	5
Guided Tour	8
Pinouts	10
JST Battery Input	10
Power Pads	10
Input/Output Pads	11
Common to all pads	11
Unique pad capabilities	11
Secret SWD and Reset Pads	11
Windows Driver Installation	13
Manual Driver Installation	14
CircuitPython	15
Reinstalling and Updating CircuitPython	15
Flashing	16
Flashing UF2	16
Flashing with BOSSAC	18
After flashing	18
CircuitPython Blinky	19
code.py	19
Status LED	19
Debugging	20
Libraries	20
More info	20
Serial Console (REPL)	21
Windows	21
Serial Drivers (for Windows 7)	21
Determine Your Serial Port	21
Install Serial Port Terminal Software	22
Mac OSX and Linux	23
Using the REPL	25
Installing Libraries	27
Installing on the Gemma	27

Out of space!	30
Delete something!	31
Use tabs	31
Mac OSX loves to add extra files.	31
Continuing after copy	32
CircuitPython Analog In	34
Creating analog inputs	34
GetVoltage Helper	34
Main Loop	34
CircuitPython Analog Out	36
Creating an analog output	36
Setting the analog output	36
Main Loop	36
CircuitPython Cap Touch	38
Creating an capacitive touch input	38
Main Loop	38
Copper Foil Tape with Conductive Adhesive - 6mm x 15 meter roll	39
Copper Foil Tape with Conductive Adhesive - 25mm x 15 meter roll	40
Small Alligator Clip Test Lead (set of 12)	40
Arduino IDE Setup	41
https://adafruit.github.io/arduino-board-index/package_adafruit_index.json	42
Using with Arduino IDE	44
Install SAMD Support	44
Install Adafruit SAMD	45
Install Drivers (Windows 7 Only)	45
Blink	47
Sucessful Upload	48
Compilation Issues	48
Manually bootloading	49
Ubuntu & Linux Issue Fix	49
Adapting Sketches to M0	50
Analog References	50
Pin Outputs & Pullups	50
Serial vs SerialUSB	50
AnalogWrite / PWM on Feather/Metro M0	51
Missing header files	52

Bootloader Launching	52
Aligned Memory Access	52
Floating Point Conversion	52
How Much RAM Available?	53
Storing data in FLASH	53
UF2 Bootloader Details	54
Entering Bootloader Mode	54
Using the Mass Storage Bootloader	56
Using the BOSSA Bootloader	57
Windows 7 Drivers	57
Verifying Serial Port in Device Manager	58
Running bossac on the command line	59
Updating the bootloader	60
Getting Rid of Windows Pop-ups	60
Making your own UF2	61
Downloads	63
Files:	63
Schematic & Fabrication Print	63

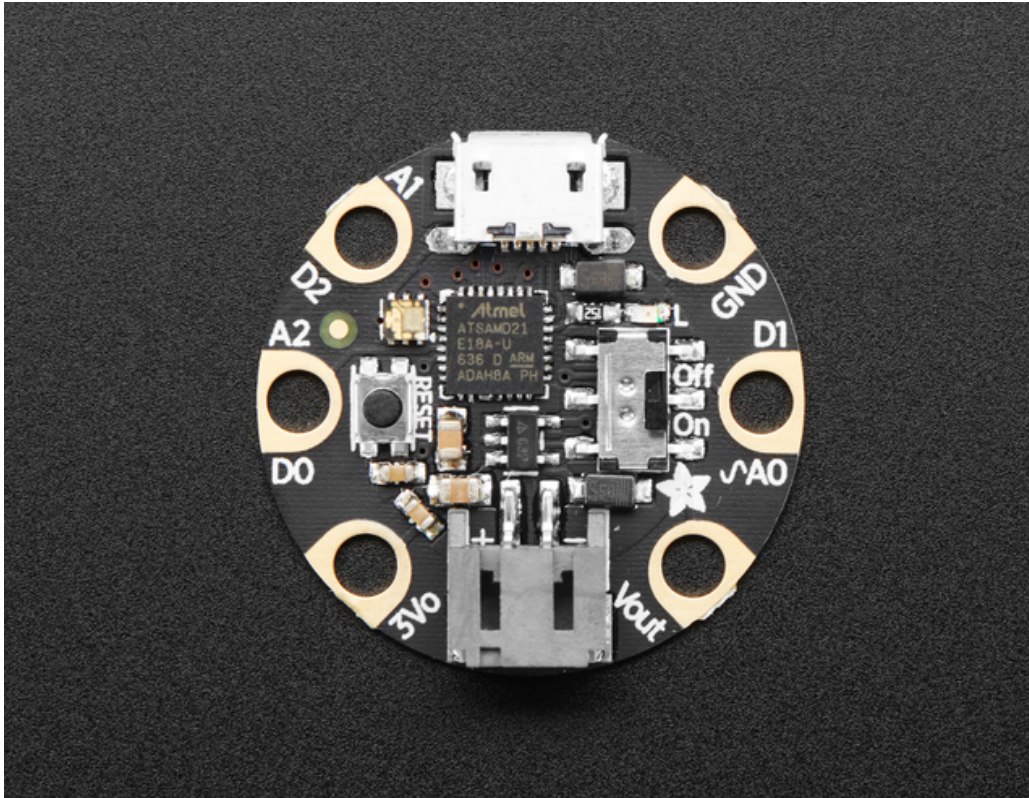
Overview



The Gemma M0 is a super small microcontroller board, with just enough to build many simple projects. It may look small and cute: round, about the size of a quarter, with friendly alligator-clip sew pads. But do not be fooled! The Gemma M0 is incredibly powerful! We've taken the same form factor we used for [the original ATtiny85-based Gemma](http://adafru.it/duB) (<http://adafru.it/duB>) and gave it a power up. The Gemma M0 has swapped out the lightweight ATtiny85 for a ATSAM21E18 powerhouse.

It will super-charge your wearables! It's just as small, and it's easier to use, so you can do more.

The most exciting part of the Gemma M0 is that while you can use it with the Arduino IDE, we are shipping it with CircuitPython on board. When you plug it in, it will show up as a very small disk drive with **main.py** on it. Edit **main.py** with your favorite text editor to build your project using Python, the most popular programming language. No installs, IDE or compiler needed, so you can use it on any computer, even ChromeBooks or computers you can't install software on. When you're done, unplug the Gemma M0 and your code will go with you.



Here are some of the updates you can look forward to when using Gemma M0:

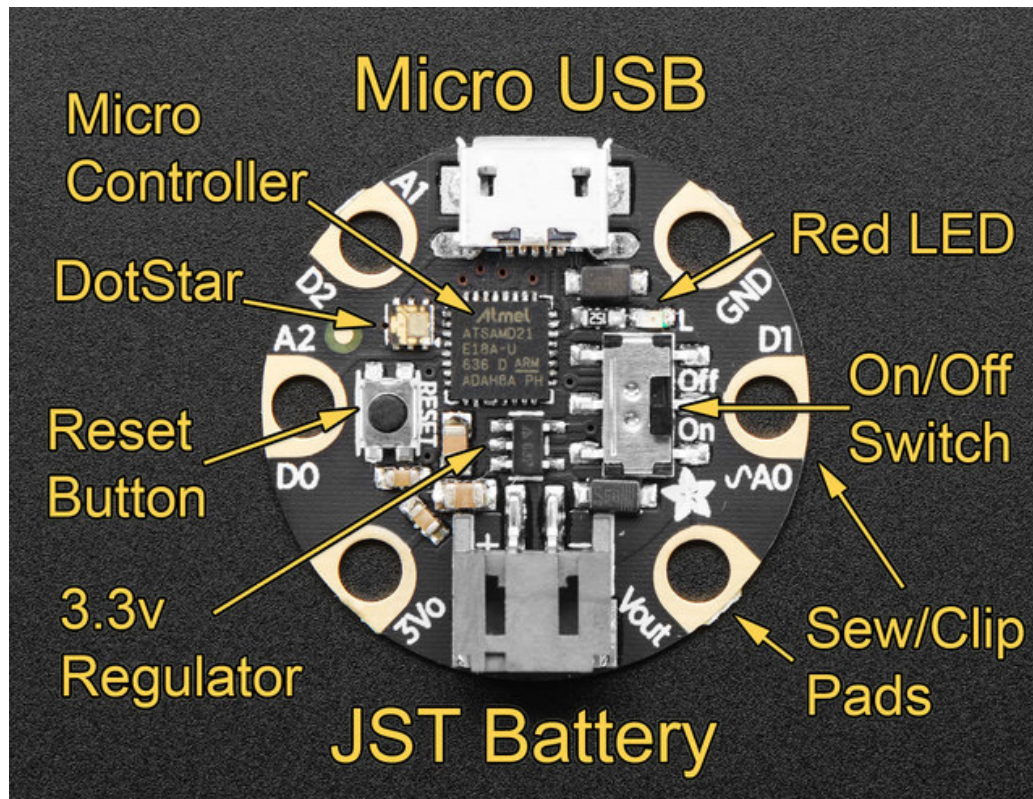
- Same size, form-factor, and pinout as classic Gemma
- Updating ATtiny85 8-bit AVR for ATSAM021E18 32-bit Cortex M0+
- **256KB Flash** - 8x as much as 8 KB on ATtiny85
- **32 KB RAM** - 64x as much as 512 bytes on ATtiny85
- **48 MHz 32 bit processor** - 6x as fast as ATtiny85 (not even taking into account 32-bit speedups)
- **Native USB supported by every OS** - can be used in Arduino or CircuitPython as USB serial console, Keyboard/Mouse HID, even a little disk drive for storing Python scripts. (ATtiny85 does not have native USB)
- Can be used with **Arduino IDE** or **CircuitPython**
- **Built in RGB DotStar LED**
- **Three big-hole sew-pads** can be used for conductive thread or alligator-clips for fast prototyping
 - Each I/O pad can be used for **12-bit analog input**, or **digital input/output** with internally connected pullups or pulldowns
 - We gave the M0 pads the exact same names as the original Gemma so all your existing Arduino code will work exactly the same as-is without changes
 - True **analog output** on one I/O pad - can be used to play 10-bit quality audio clips
 - **Two high speed PWM** outputs on other two I/O Pads - for servos, LEDs, etc
 - All three pads can also be used as **hardware capacitive touch sensors** with no additional components required
 - Can drive **NeoPixels or DotStars on any pins, with enough memory to drive 8000+ pixels DMA-NeoPixel support on one pin.** (<http://adafru.it/xYD>) so you can drive pixels without having to spend any processor time on it.
 - Native hardware I2C or Serial available on two pads so you can connect to any I2C or Serial device with true hardware support (no annoying bit-banging)
- Same **Reset switch** for starting your project code over
- **On/Off switch** built in
- **JST battery connector** for plugging in AAA's or LiPoly battery (no built-in LiPoly charging so it is safe to use with NiMH/Alkalines)



Each order comes with one fully assembled and tested Gemma M0 with CircuitPython & example code programmed in.

So what are you waiting for? Pick up a Gemma M0 today and be amazed at how easy and fast it is to get started with Gemma and CircuitPython!

Guided Tour



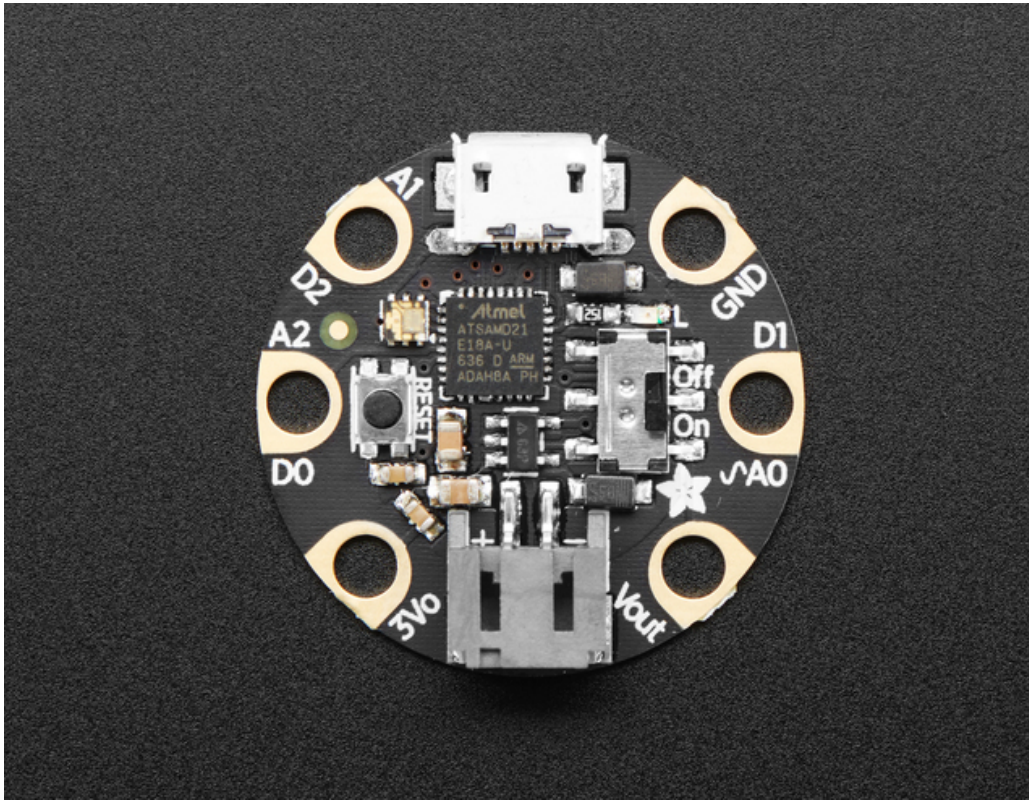
Let me take you on a tour of your Gemma M0! Each Gemma M0 is assembled here at Adafruit and comes chock-full of good design to make it a joy to use.

- **Micro B USB connector** - We went with the tried and true micro-B USB connector for power and/or USB communication (bootloader, serial, HID, etc). Use with any computer with a standard data/sync cable.
- **RGB DotStar LED** - Instead of an always-on green LED we provide a full RGB LED. You can set it to any color in the rainbow. It will also help you know when the bootloader is running (it will turn green) or if it failed to initialize USB when connected to a computer (it will turn green). By default after you boot up the Gemma M0 it will turn a lovely violet color.
- **Red #13 LED** - this LED does double duty. Its connected with a series resistor to the digital #13 GPIO pin. It pulses nicely when the Gemma is in bootloader mode, and its also handy for when you want an indicator LED.
- **JST Battery Input** - take your Gemma anywhere and power it from an external battery. This pin can take up 6V DC input, and has reverse-polarity, over-current and thermal protections. The circuitry inside will use either the battery or USB power, safely switching from one to the other. If both are connected, it will use whichever has the higher voltage. Works great with a Lithium Polymer battery or our 3xAAA battery packs with a JST connector on the end. There is no built in battery charging (so that you can use Alkaline or Lithium batteries safely)
- **Vout (Voltage Output)** - This pin will give you either the battery power or USB power, whichever has a higher voltage. Its great when you want to power something like NeoPixels, that might use more than the 500mA available from the onboard regulator
- **3V Regulator** - The on-board voltage regulator can supply up to 500mA at a steady 3.3V from up to 6VDC
- **Sewing and Alligator clip friendly pads** - You can easily sew to these pads, and they're gold plated so they wont corrode (oxidize). You can also use alligator clips or solder directly to them.
- **3 General Purpose I/O (GPIO) Pads!** - 3 GPIO pins, at 3V logic, check the next section for a detailed pinout

guide

- **Reset Button** - an onboard reset button will launch the bootloader when pressed and the Gemma is plugged into a computer. If it is not connected to a computer, it's smart enough to go straight to the program.
- **On/Off switch** - Lets you turn on/off your project, it will control both the Gemma and the Vout pad. This switch can switch up to about 500mA of current, so if you are driving a huge number of servos or NeoPixels, connect power to those power-greedy parts externally.

Pinouts



JST Battery Input

There is no battery **INPUT** pin on the Gemma. You can connect a battery via the JST jack. We have found that [Lipoly batteries](http://adafru.it/cFB) (<http://adafru.it/cFB>), [coin-cells](http://adafru.it/783) (<http://adafru.it/783>), and [AAA's](http://adafru.it/727) (<http://adafru.it/727>) work great. [You can also make your own battery input pack using a plain JST cable](http://adafru.it/261) (<http://adafru.it/261>). And use a [JST extension cable if necessary](http://adafru.it/1131) (<http://adafru.it/1131>).

You can plug anything from around 4 VDC up to 6 VDC. That means any single-cell LiPoly, or 3-4 AAA or AA batteries. This input is polarity protected. Gemma and DotStar LED light up, you're good to go. You can turn off the battery with the on/off switch, which will completely disconnect power on the Gemma M0.

Power Pads

Half of the pads on the Gemma M0 are related to power in and out **3Vo** , **Vout** and **GND**

- **Vout** - This is a voltage **OUTPUT** pin, it will be connected to *either* the USB power or the battery input, whichever has the higher voltage. This output does not connect to the regulator so you can draw as much current as your USB port / Battery can provide (in general, thats about 500mA)
- **3Vo** - This is the **3.3V OUTPUT** pad from the voltage regulator. It can provide up to 500mA at a steady 3.3V. Good for sensors or small LEDs or other 3V devices.
- **GND** is the common ground pin, used for logic and power. It is connected to the USB ground and the power regulator, etc. This is the pin you'll want to use for any and all ground connections

Input/Output Pads

Next we will cover the 3 GPIO (General Purpose Input Output) pins! For reference you may want to also check out the datasheet-reference in the downloads section for the core ATSAM21E18 pin. We picked pins that have *a lot* of capabilities.

Common to all pads

All the GPIO pads can be used as digital inputs, digital outputs, for LEDs, buttons and switchesIn addition, all can be used as analog inputs (12-bit ADC) or hardware capacitive touch. All pads can also be used as hardware interrupts inputs.

Each pad can provide up to ~20mA of current. Don't connect a motor or other high-power component directly to the pins! [Instead, use a transistor to power the DC motor on/off](http://adafru.it/aUD) (<http://adafru.it/aUD>)

On a Gemma M0, the GPIO are 3.3V output level, and should not be used with 5V inputs. In general, most 5V devices are OK with 3.3V output though.

The three pads are completely 'free' pins, they are not used by the USB connection, LEDs, DotStar, etc so you never have to worry about the USB interface interfering with them when programming

Unique pad capabilities

- **Pad #0 / A2** - this is connected to **PA04** on the ATSAM21. This pin can be used as a digital I/O with selectable pullup or pulldown, capacitive touch, analog input (use 'A2'), PWM output, and is also used for I2C data (SDA), and hardware Serial RX.
- **Pad #1 / A0** - this is connected to **PA02** on the ATSAM21. This pin can be used as a digital I/O with selectable pullup or pulldown, capacitive touch, analog input (use 'A0'), and true analog (10-bit DAC) output. It cannot be used as PWM output.
- **Pad #2 / A1** - this is connected to **PA05** on the ATSAM21. This pin can be used as a digital I/O with selectable pullup or pulldown, capacitive touch, analog input (use 'A1'), PWM output, and is also used for I2C clock (SCL), and hardware Serial TX.

Secret SWD and Reset Pads

On the bottom of the Gemma M0 you will see three small pads. These are used for our programming/test but you can use them too.



Starting from the pad closest to the edge there is:

- SWDIO
- SWCLK
- Reset

On the off chance you want to reprogram your Gemma M0 or debug it using a Cortex M0 debug/programmer, you will need to solder/connect to these pads. We use them for testing and you will likely never need it but they are there if you do!

Windows Driver Installation

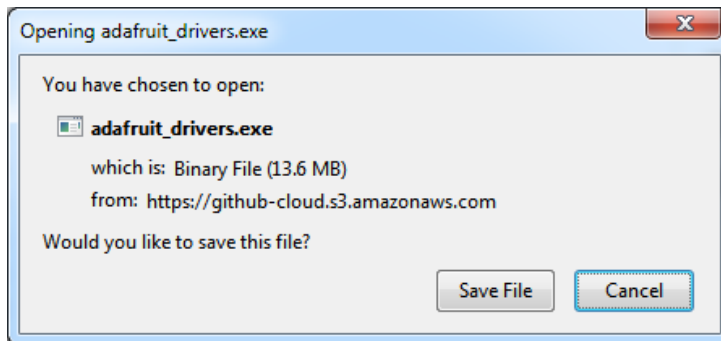
Mac and Linux do not require drivers, only Windows folks need to do this step

Before you plug in your board, you'll need to possibly install a driver!

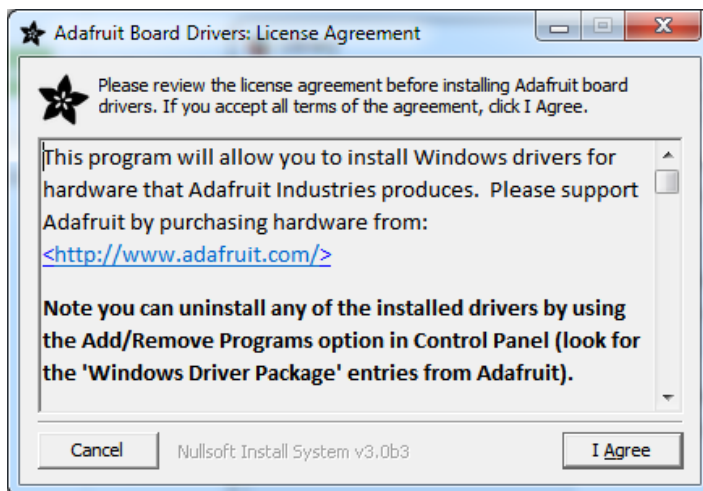
Click below to download our Driver Installer

[Download Adafruit Windows Driver Installer](http://adafru.it/mai)
<http://adafru.it/mai>

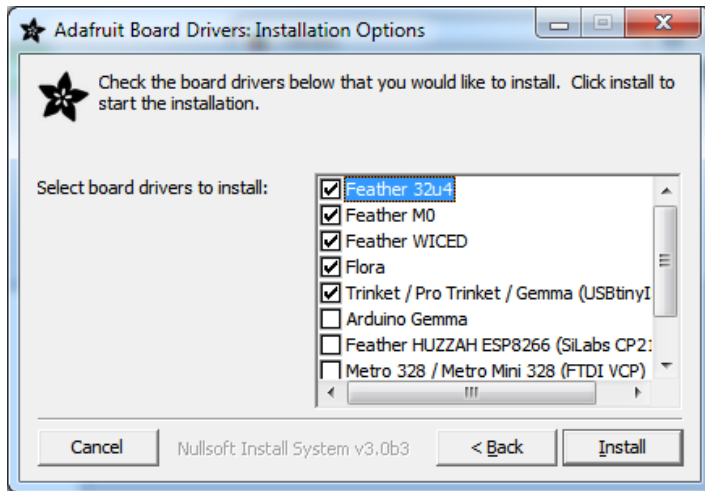
Download and run the installer



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



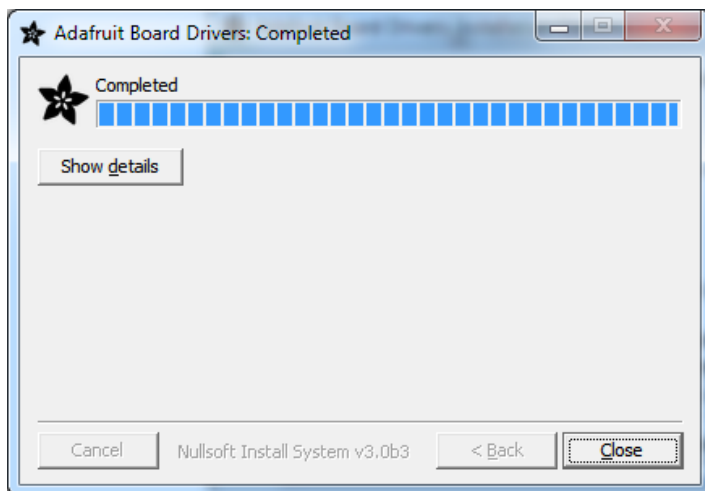
Select which drivers you want to install, we suggest selecting all of them so you don't have to do this again!



By default, we install the **Feather 32u4** , **Feather M0**, **Flora**, **Gemma M0**, **Circuit Playground** and **Trinket / Pro Trinket / Gemma / USBtinyISP** drivers.

You can also, optionally, install the **Arduino Gemma** (different than the Adafruit Gemma!), Huzzah and Metro drivers

Click **Install** to do the installin'



Manual Driver Installation

If windows needs the driver files (inf/cat) for some reason you can get all the drivers in a zip by clicking below:

[Adafruit Windows Drivers](http://adafruit.com/drivers)

<http://adafru.it/rEY>

And point windows to the **Drivers** folder when it asks for the driver location

CircuitPython

[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. The Gemma M0 is the first board that comes pre-loaded with CircuitPython. Simply copy and edit files on the CIRCUITPY drive to iterate.

Your Gemma M0 already comes with CircuitPython but maybe there's a new version, or you overwrote your Gemma M0 with Arduino code! In that case, see the below for how to reinstall or update CircuitPython. Otherwise you can skip this and go straight to the next page

Reinstalling and Updating CircuitPython

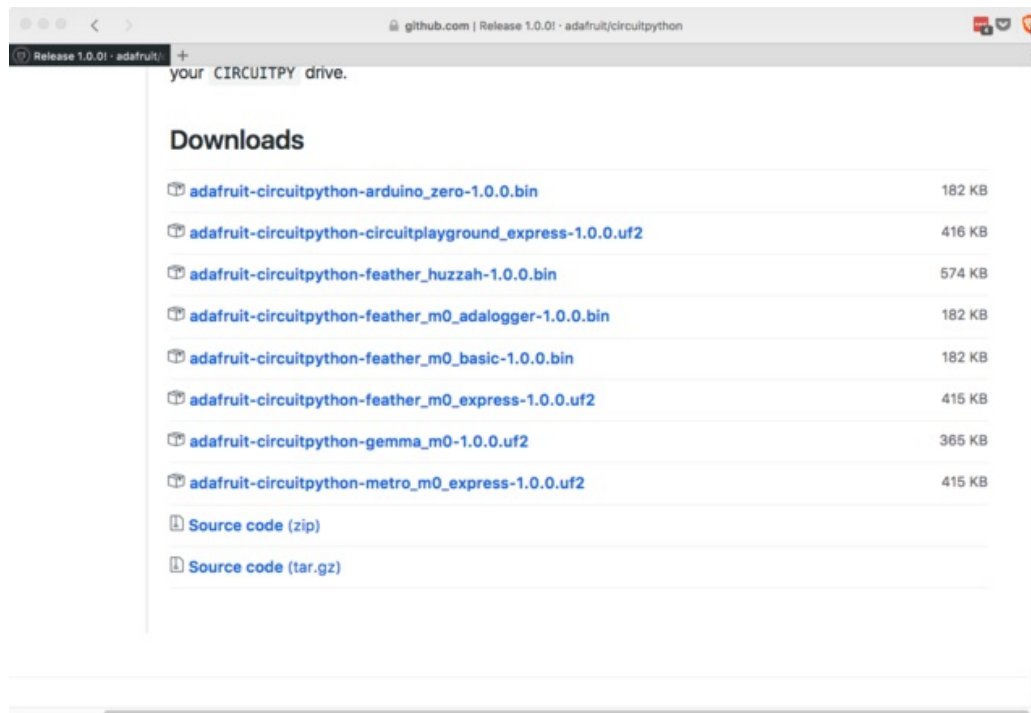
The latest builds of [CircuitPython](#) are available from the [GitHub release page](#). Binaries for different boards are listed under the Downloads section. Pick the one that matches your board such as `adafruit-circuitpython-gemma_m0-1.0.0.bin` for the Gemma M0.

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

[Click here to see the latest CircuitPython Release](#)

<http://adafru.it/v1F>

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!



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 newer boards including the **Gemma M0**. Flashing via bossac is possible with both the UF2 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.

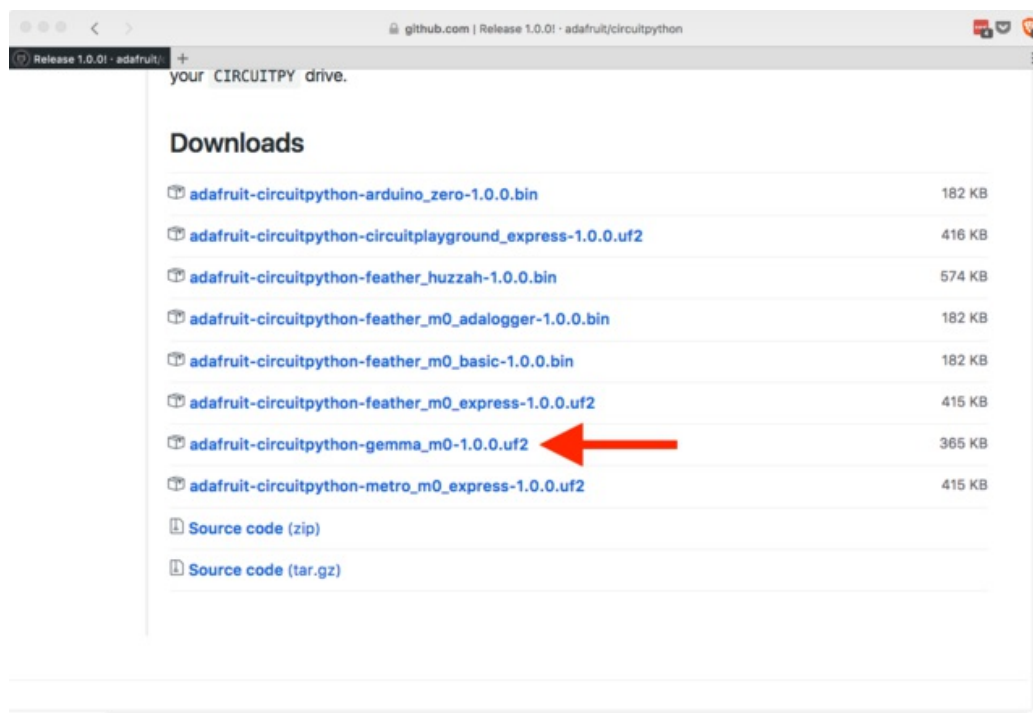
Flashing UF2

The Gemma M0 comes with a new 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.](http://adafru.it/vQd) (<http://adafru.it/vQd>)

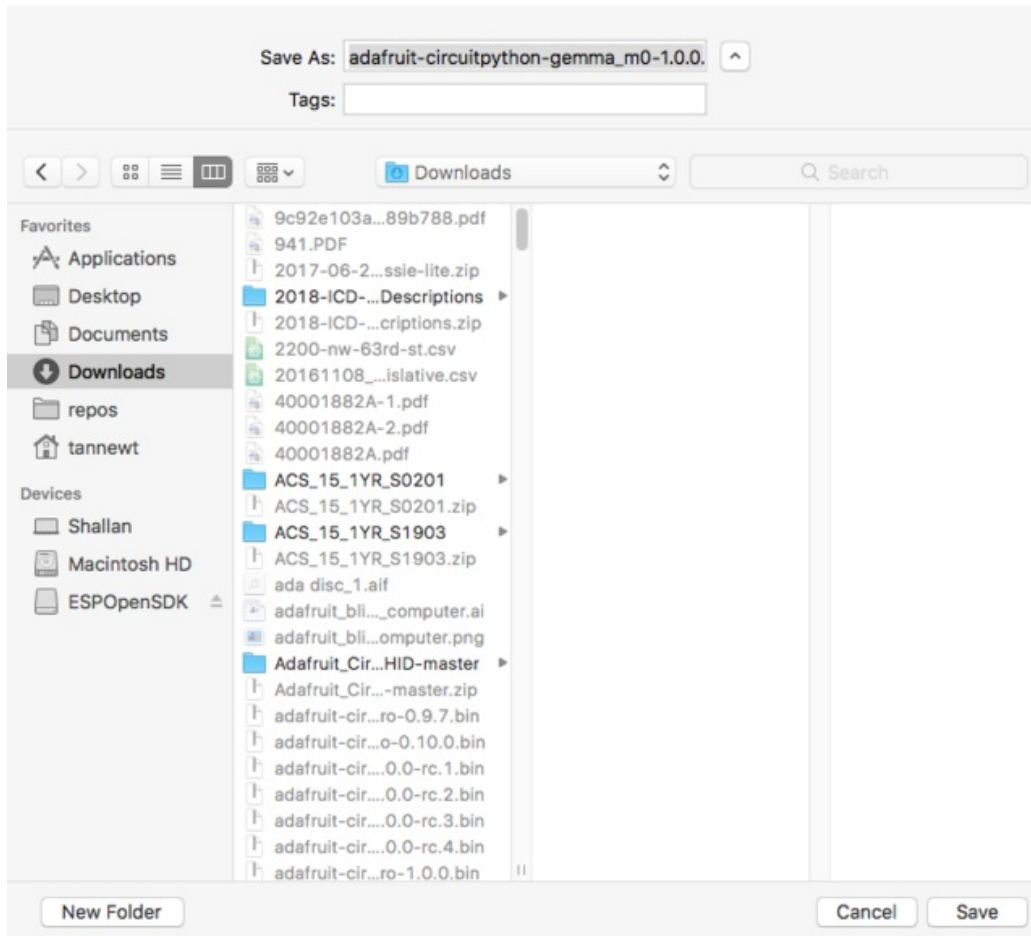
Start by ejecting or "safely remove" the CIRCUITPY drive if its present, then double-clicking the reset button while it is plugged into your computer. You should see a new disk drive 'pop up' called **GEMMABOOT** or similar, and the DotStar on your board glow green.

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 **Gemma M0 UF2** file in the github downloads list:

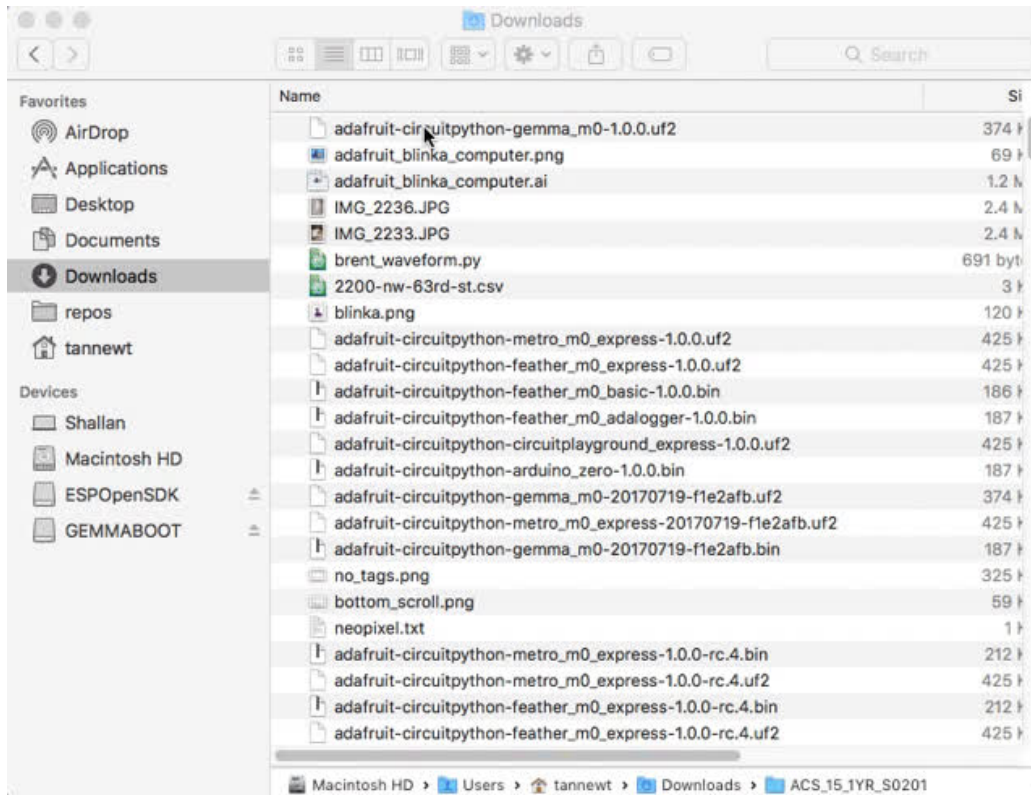


Click to download and save the file onto your Desktop or somewhere else you can find it



Then drag the **uf2** file into the GEMMABOOT 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-gemma_m0-1.0.0.bin
```

This will erase the chip, write the given file, verify the write and Reset the board. After reset, CircuitPython should be running. Newer boards with the UF2 bootloader may cause a warning of an early eject of a USB drive but just ignore it. Nothing important was being written to the drive.

After flashing

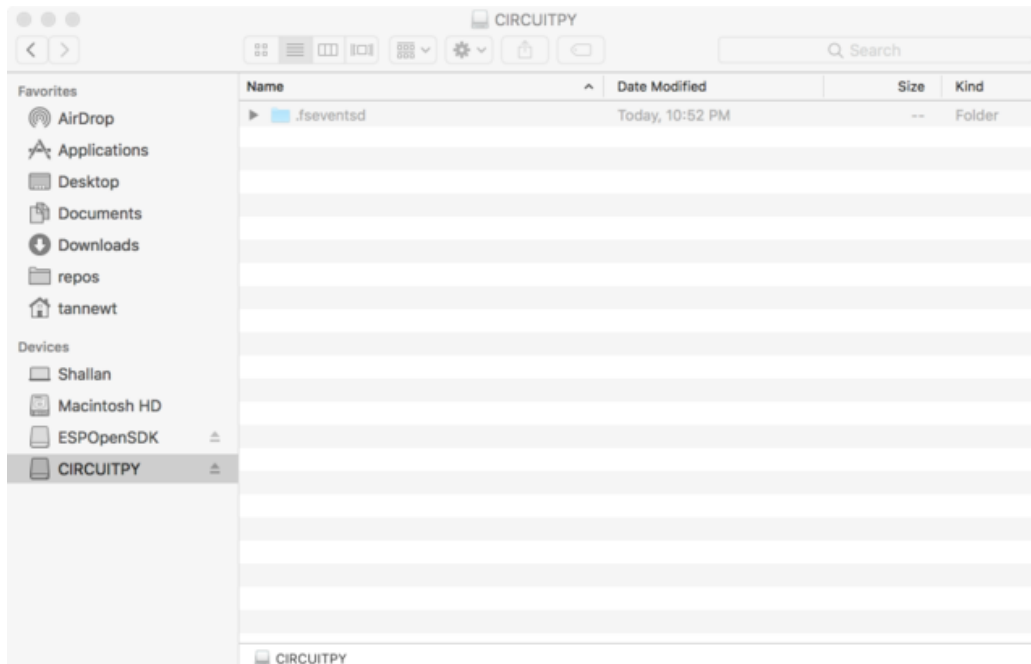
After a successful flash by bossac or UF2 you should see a CIRCUITPY drive appear.

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.txt (code.py also works). After a brief time, the board's red LED should begin to flash every second.

Status LED

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.

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](#)

Serial Console (REPL)

CircuitPython sends the output of the .py files it runs to the connected computer over USB serial. So, to view the output of your code from print statements and any errors that occur you'll need to connect to the serial console.

Also, because CircuitPython is a variant of Python, it too has a read-evaluate-print-loop or 'REPL' for short. The REPL lets you run individual commands and load up code interactively and is therefore great for testing a new idea. However, the code typed into the REPL isn't saved anywhere so make sure and save it elsewhere (like code.py for example.)

Windows

Serial Drivers (for Windows 7)

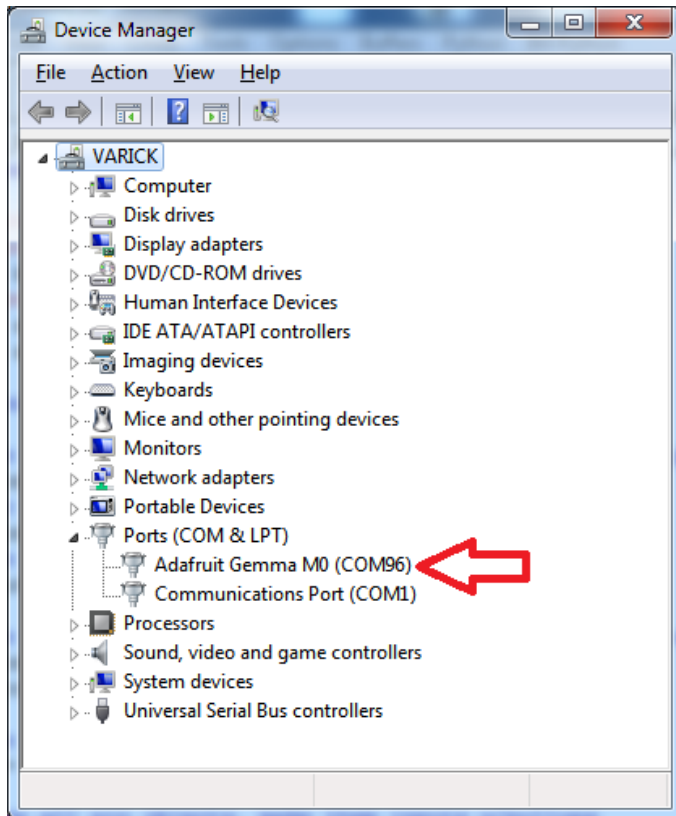
If you are using Windows 7 you will need to install drivers. Click below to download the driver package and install it! This is not necessary for Mac, Linux or Windows 10+.

[Download Adafruit Windows Driver Installer](#)

<http://adafru.it/mai>

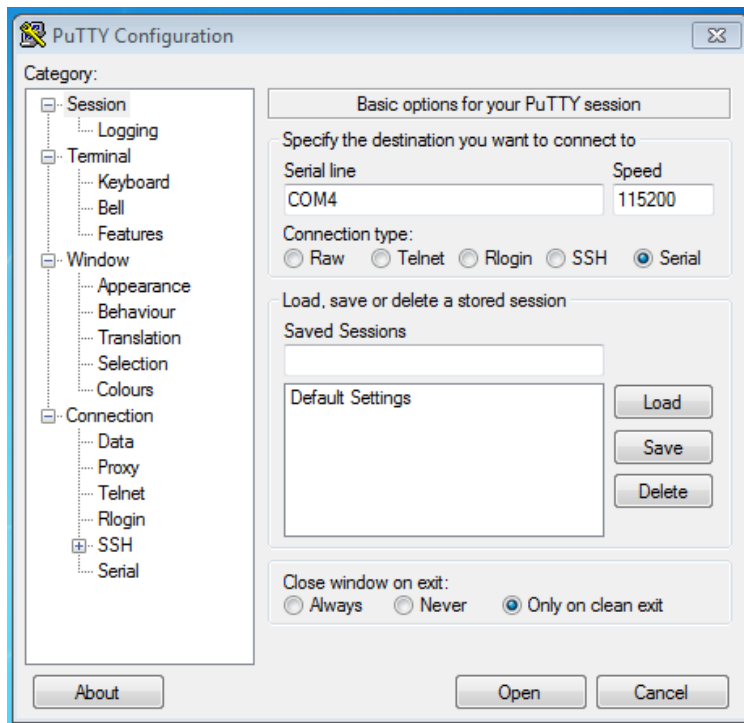
Determine Your Serial Port

Next you must determine the name of the serial port for your board. It's easiest to look at the serial ports with the board disconnected (on Windows check **Device Manager** under the **Ports (COM/LPT)** node

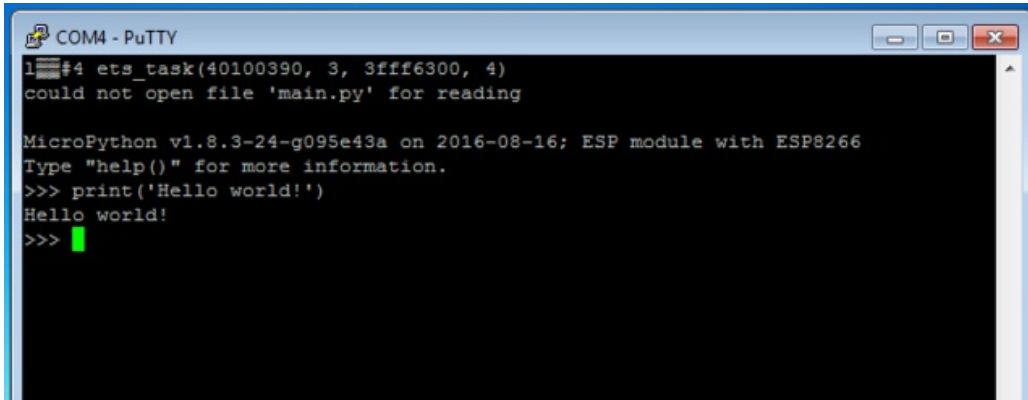


Install Serial Port Terminal Software

On Windows you'll want to use a tool like [PuTTY](http://adafru.it/pNe) (<http://adafru.it/pNe>) to connect to the serial port. Download and run PuTTY, then configure it to use a serial connection to the board's COM port at 115200 baud similar to as shown below:



After clicking **Open** you should see a new window pop up with the current output from the running code. If no code is running, it may be blank so hit **Ctrl - C** to get to the REPL prompt.

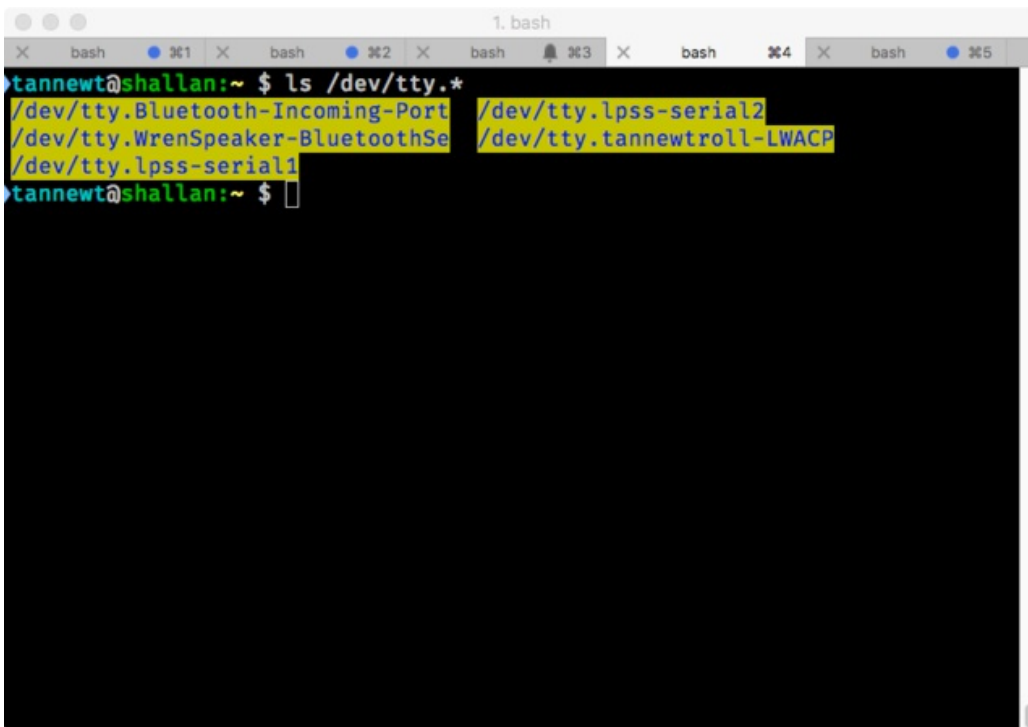


Mac OSX and Linux

Connecting to the serial terminal on is more straightforward than on Windows. Neither OS requires additional drivers.

First open a terminal program. On Mac OSX, Terminal comes installed and [Term2](http://adafru.it/xZd) (<http://adafru.it/xZd>) can be downloaded. On Linux there are a variety available such as `gnome-terminal` (called Terminal) and `Konsole` on KDE.

Now before plugging in the board, type `ls /dev/tty.*` to view existing serial connections.



Next, plug in the board. There should be one new serial connection that is for your board. Typically on Mac OSX its something like `/dev/tty.usbmodem*` and on Linux its `/dev/ttyACM*`.

```
1. bash
bash  #1  bash  #2  bash  #3  bash  #4  bash  #5
tannewt@shallan:~$ ls /dev/tty.*
/dev/tty.Blueetooth-Incoming-Port  /dev/tty.lpsserial2
/dev/tty.WrenSpeaker-BluetoothSe  /dev/tty.tannewtroll-LWACP
/dev/tty.lpsserial1
tannewt@shallan:~$ ls /dev/tty.*
/dev/tty.Blueetooth-Incoming-Port  /dev/tty.lpsserial2
/dev/tty.WrenSpeaker-BluetoothSe  /dev/tty.tannewtroll-LWACP
/dev/tty.lpsserial1                /dev/tty.usbmodem145111
tannewt@shallan:~$
```

Now that we know the device name, the `screen` command can be used to connect to the serial port. Its installed on Mac OSX by default but Linux users may need to install it using their package manager. Run the following command to connect at 115200 baud:

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

Where `/dev/tty.board_name` is the name of the board's serial port.

When you're done using screen most versions of it allow you to exit by pressing **Ctrl-a** then **k** then **y** or pressing **Ctrl-a** then typing `:quit` and pressing **enter**.


```
1. bash
bash x1 x bash x2 x bash x3 x bash x4 x bash x5
tannewt@shallan:~$ ls /dev/tty.*
/dev/tty.Blueetooth-Incoming-Port /dev/tty.lpsserial2
/dev/tty.WrenSpeaker-BluetoothSe /dev/tty.tannewtroll-LWACP
/dev/tty.lpsserial1
tannewt@shallan:~$ ls /dev/tty.*
/dev/tty.Blueetooth-Incoming-Port /dev/tty.lpsserial2
/dev/tty.WrenSpeaker-BluetoothSe /dev/tty.tannewtroll-LWACP
/dev/tty.lpsserial1 /dev/tty.usbmodem145111
tannewt@shallan:~$ screen /dev/tty.usbmodem145111 115200
[screen is terminating]
tannewt@shallan:~$
```

Using the REPL

After you're connected to the serial REPL try pressing enter to confirm you see the >>> prompt. You can also type `help()` and press enter on most boards to see basic usage information.

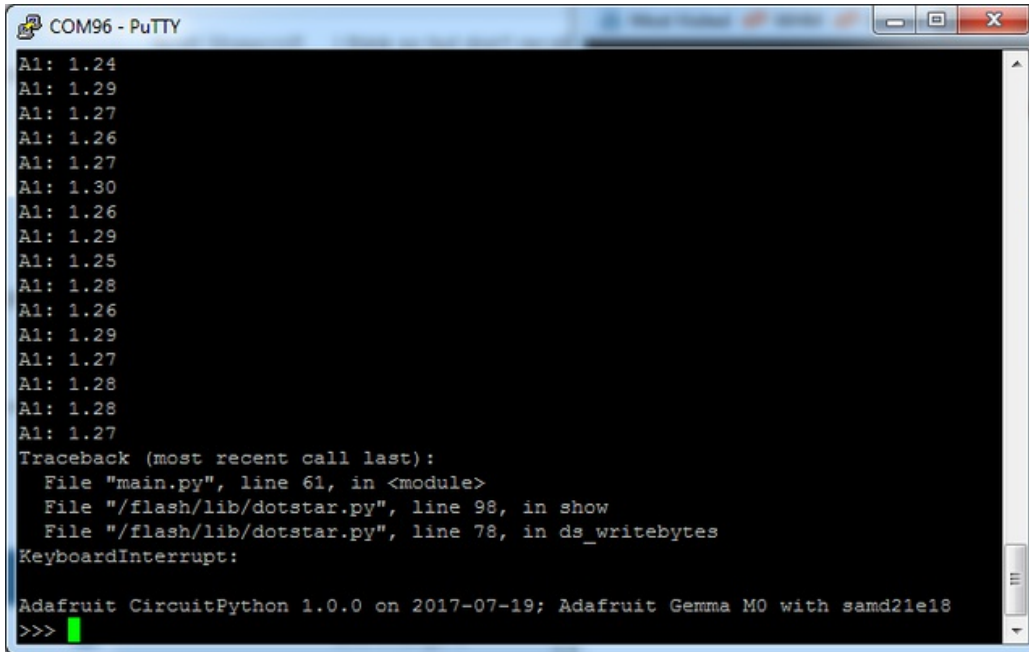
```
COM96 - PuTTY
>>> help()
Welcome to Adafruit CircuitPython 1.0.0!

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

Built in modules:
  __main__
  builtins
  micropython
  array
  ucollections
  ustruct
  math
  sys
  gc
  urandom
  touchio
  microcontroller
  analogio
  digitalio
  busio
  board
  uos
  time
```

If you can't get a >>> prompt to appear try pressing **Ctrl-c** a couple times to interrupt any running program on the board.

You might get a **Traceback** and **KeyboardInterrupt** that lets you know the current Python program has stopped, and you'll get a prompt:



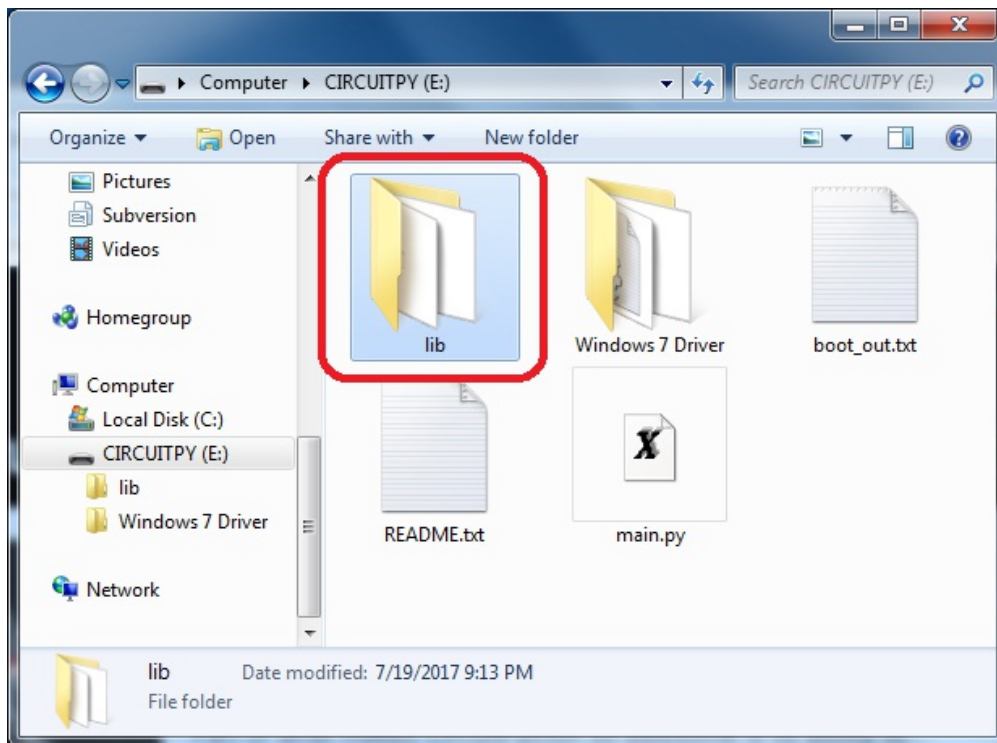
```
COM96 - PuTTY
A1: 1.24
A1: 1.29
A1: 1.27
A1: 1.26
A1: 1.27
A1: 1.30
A1: 1.26
A1: 1.29
A1: 1.25
A1: 1.28
A1: 1.26
A1: 1.29
A1: 1.27
A1: 1.28
A1: 1.28
A1: 1.27
Traceback (most recent call last):
  File "main.py", line 61, in <module>
  File "/flash/lib/dotstar.py", line 98, in show
  File "/flash/lib/dotstar.py", line 78, in ds_writebytes
KeyboardInterrupt:
Adafruit CircuitPython 1.0.0 on 2017-07-19; Adafruit Gemma M0 with samd21e18
>>>
```

That's all there is to connecting to the board's serial REPL, you're ready to start typing in and running CircuitPython code!

Installing Libraries

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. So, instead of compiling libraries in, you simply place them into your `lib` directory on the CIRCUITPY drive.

Your Gemma M0 should come with a `lib` folder already, its in the base directory of the drive:



CircuitPython libraries work in the same way 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 on the Gemma

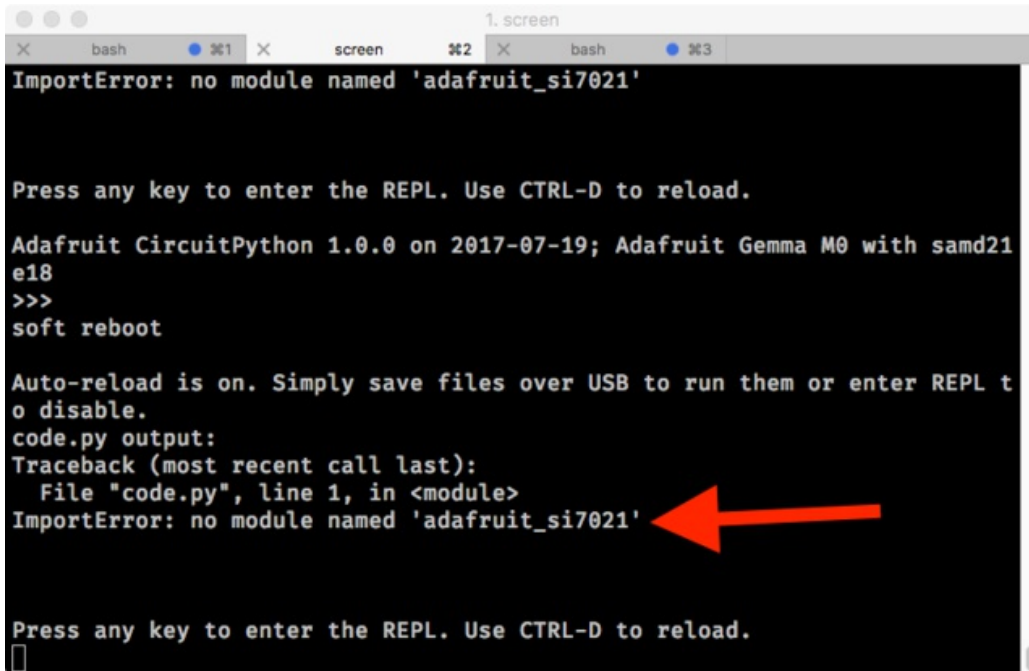
The Gemma M0's tiny size means we weren't able to fit extra flash memory on the board. That's why the drive is less than 64kb total! It's the microcontroller itself that's saving the data, in the internal 256KB Flash space - about 192KB for the bootloader and Python interpreter and about 64KB for user code.

We need to be selective about what libraries we load on the Gemma M0 because we can't simply fit them all like we can on an Express board. So, let's take a look at this silly example below which uses a SI7021 I2C temperature sensor.

```
import adafruit_si7021
import busio
import board

i2c = busio.I2C(board.SCL, board.SDA)
sensor = adafruit_si7021.SI7021(i2c)
print("Temperature:", sensor.temperature)
print("Humidity:", sensor.relative_humidity)
```

After saving that as code.py on the drive we see the status NeoPixel flashes that an error has occurred. Opening up the serial console we see that an ImportError has occurred.

A screenshot of a terminal window titled "1. screen" with three tabs: "bash", "screen", and "bash". The terminal output shows an ImportError: no module named 'adafruit_si7021'. Below the error, it says "Press any key to enter the REPL. Use CTRL-D to reload." and "Adafruit CircuitPython 1.0.0 on 2017-07-19; Adafruit Gemma M0 with samd21e18". It then shows a soft reboot and auto-reload status. The error message is repeated, and a red arrow points to it. The terminal ends with another "Press any key to enter the REPL. Use CTRL-D to reload." prompt.

```
ImportError: no module named 'adafruit_si7021'

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

Adafruit CircuitPython 1.0.0 on 2017-07-19; Adafruit Gemma M0 with samd21e18
>>>
soft reboot

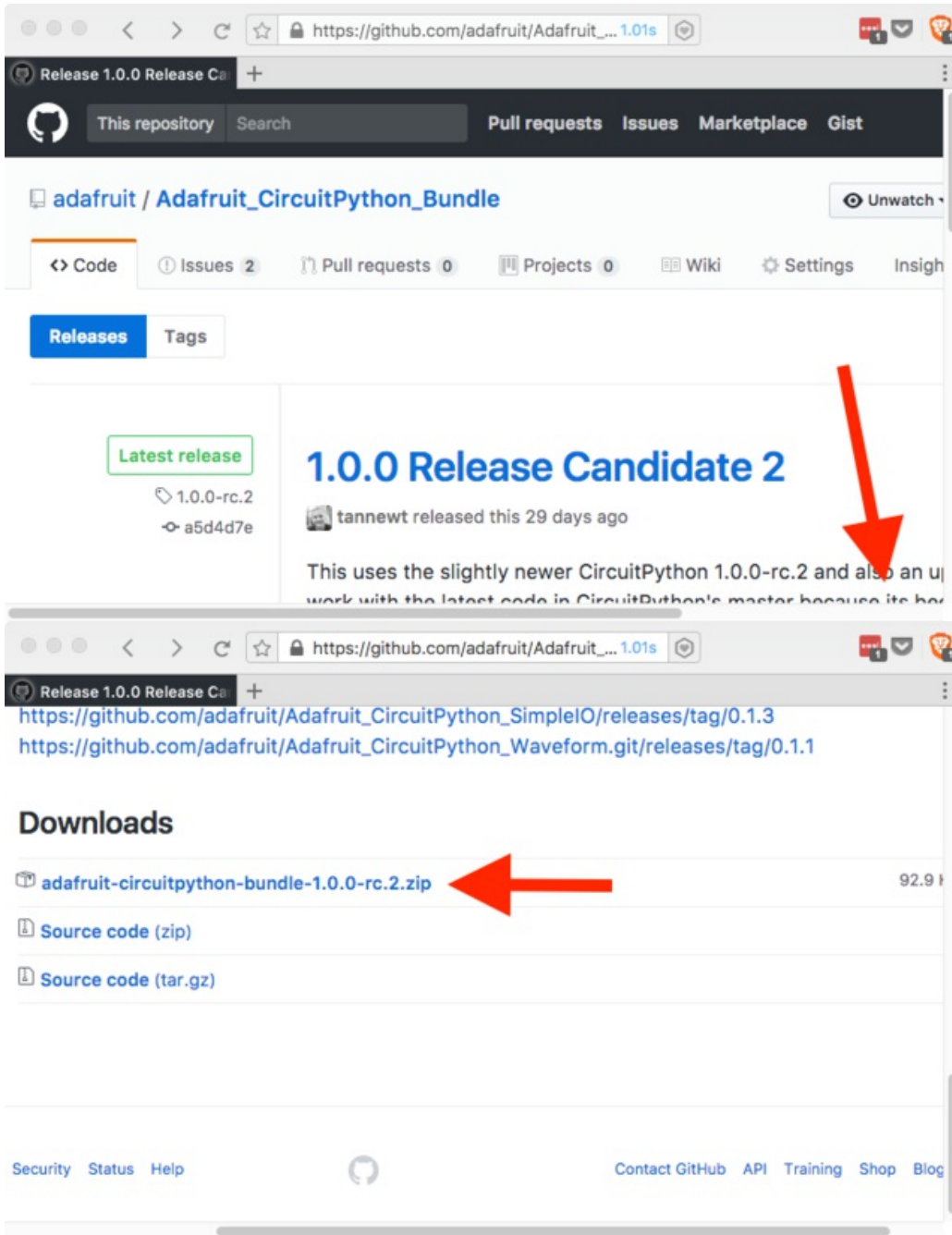
Auto-reload is on. Simply save files over USB to run them or enter REPL to
disable.
code.py output:
Traceback (most recent call last):
  File "code.py", line 1, in <module>
ImportError: no module named 'adafruit_si7021'

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

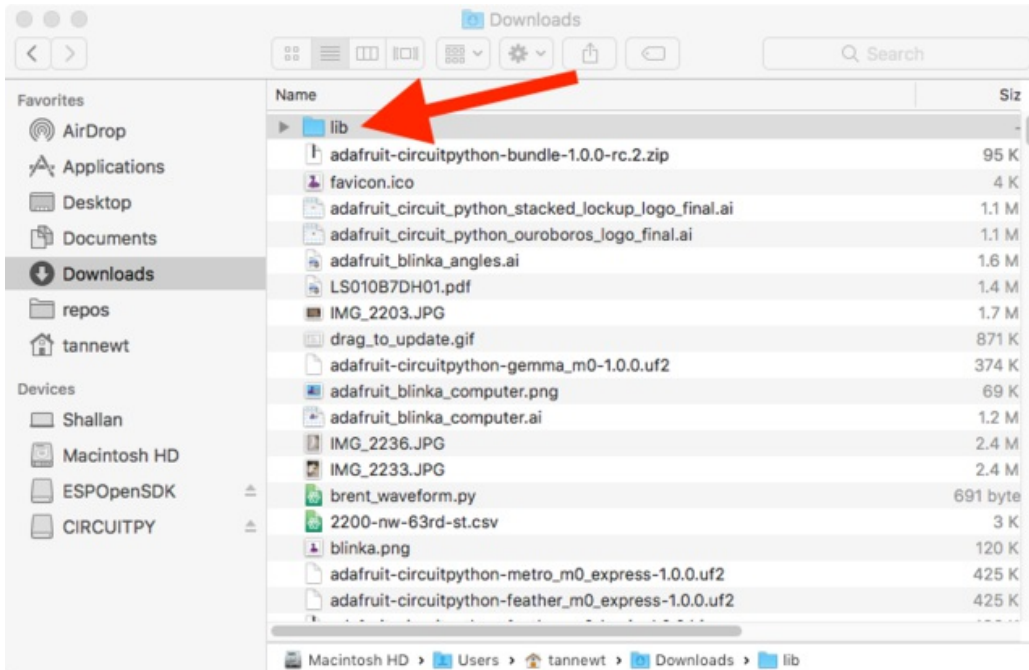
It says that no module exists named `adafruit_si7021`. That's the library we need to download! Since we bought the sensor from Adafruit it's likely there is a library for in [the official Adafruit bundle](#). If it's not an Adafruit part or it's missing, we can also check [the Community bundle](#) which has libraries contributed by the community.

[Click for the latest Adafruit Bundle release](#)
<http://adafru.it/y8E>

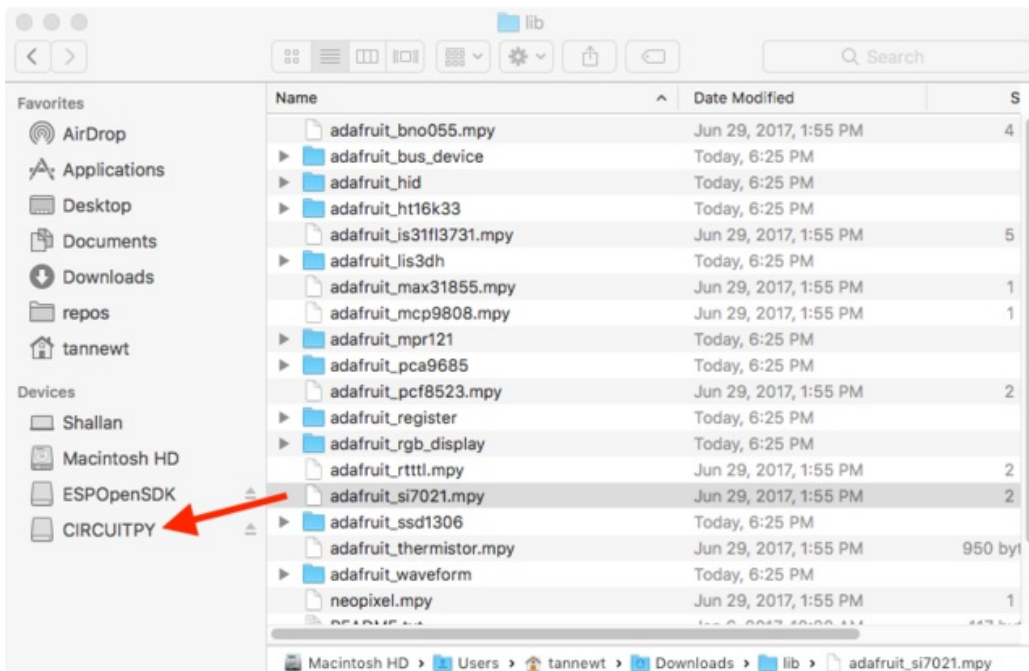
Visiting the bundle release page will show us information on the latest release including a list of all the versions of the included drivers. Scrolling to the bottom of the page will reveal the downloads. We'll download the first zip file which starts with `adafruit-circuitpython-bundle`.



After downloading the zip, extract its contents. This is usually done by double clicking on the zip. On Mac OSX as I'm using, it places the file in the same directory as the zip. I usually sort my Downloads by file data so the lib directory that was contained in the zip ends up next to the zip file.



On Express boards, the `lib` directory can be copied directly to the CIRCUITPY drive. However, the Gemma M0 doesn't have enough space for all of the libraries. So, we'll copy over just what we need as we need it. In the `lib` folder there is an `adafruit_si7021.mpy` file. That matches the missing module! Python imports modules based on the filename so they will always match up. Lets drag it over. If this works, skip to [the next section](#). Keep reading if you have an error.



Out of space!

The file system on the Gemma M0 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 Gemma M0 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.

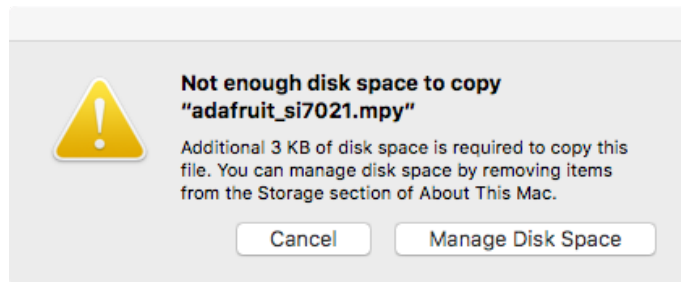
Delete something!

The simplest way of freeing up space is to delete files from the drive. Perhaps there are libraries in there 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.



Mac OSX hasn't really supported floppy sized file systems in decades. As a result, it loves to store extra information on the drive. We'd turn this off if we could but we can't. So, if you are on Mac with your Gemma at all, be on the lookout for extra files. You'll need to use another OS or the command line to successfully delete them.

Here is how to see how much space is free and what all of the files are.

```
1. bash
(venv) tannewt@shallan:/Volumes $ 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/CIRCUITPY
(venv) tannewt@shallan:/Volumes $ ls -a CIRCUITPY/
./
../
.TemporaryItems/
.Trashes/
..TemporaryItems*
.fsevents/
README.txt*
Windows 7 Driver/
boot_out.txt*
code.py*
lib/
original_code.py*
```

Lets remove the `._` files first.

```
1. bash
bash #1 bash #2 bash #3
(venv) tannewt@shallan:/Volumes $ 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/CIRCUITPY
(venv) tannewt@shallan:/Volumes $ ls -a CIRCUITPY/
./                ..                .TemporaryItems/ .Trashes/
..                .original_code.py* .fseventsd/      README.txt*
lib/              original_code.py*
Windows 7 Driver/
(venv) tannewt@shallan:/Volumes $ rm CIRCUITPY/.*
(venv) tannewt@shallan:/Volumes $ df -h /Volumes/CIRCUITPY/
Filesystem      Size  Used Avail Capacity iused ifree %iused  Mounted on
/dev/disk3s1    59Ki  42Ki  18Ki   71%    128     0  100%  /Volumes/CIRCUITPY
(venv) tannewt@shallan:/Volumes $ ls -a CIRCUITPY/
./                ..                .TemporaryItems/ .Trashes/
..                .fseventsd/      Windows 7 Driver/ lib/
README.txt*      boot_out.txt*    original_code.py*
code.py*
(venv) tannewt@shallan:/Volumes $
```

Whoa! We have 13Ki more than before! Lets continue!

Continuing after copy

Woohoo! Everything copied over just fine. Lets check the serial terminal to see how things are going.

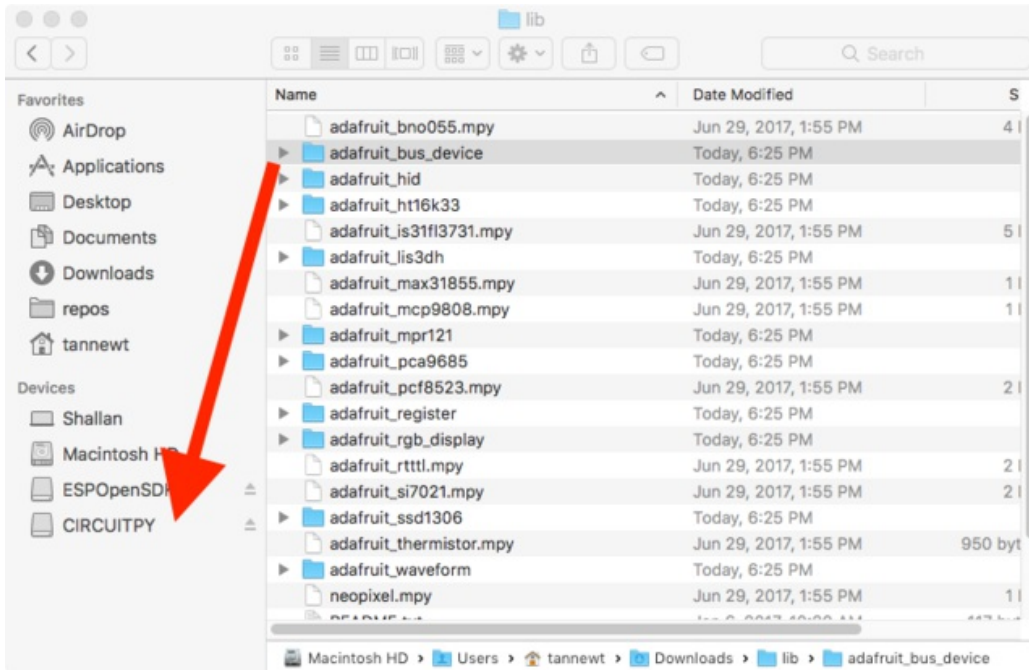
```
1. screen
bash #1 screen #2 bash #3
Auto-reload is on. Simply save files over USB to run them or enter REPL to disable.

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:
Traceback (most recent call last):
  File "code.py", line 1, in <module>
    File "libraries/drivers/si7021/adafruit_si7021.py", line 36, in <module>
>
ImportError: no module named 'adafruit_bus_device'
Press any key to enter the REPL. Use CTRL-D to reload.

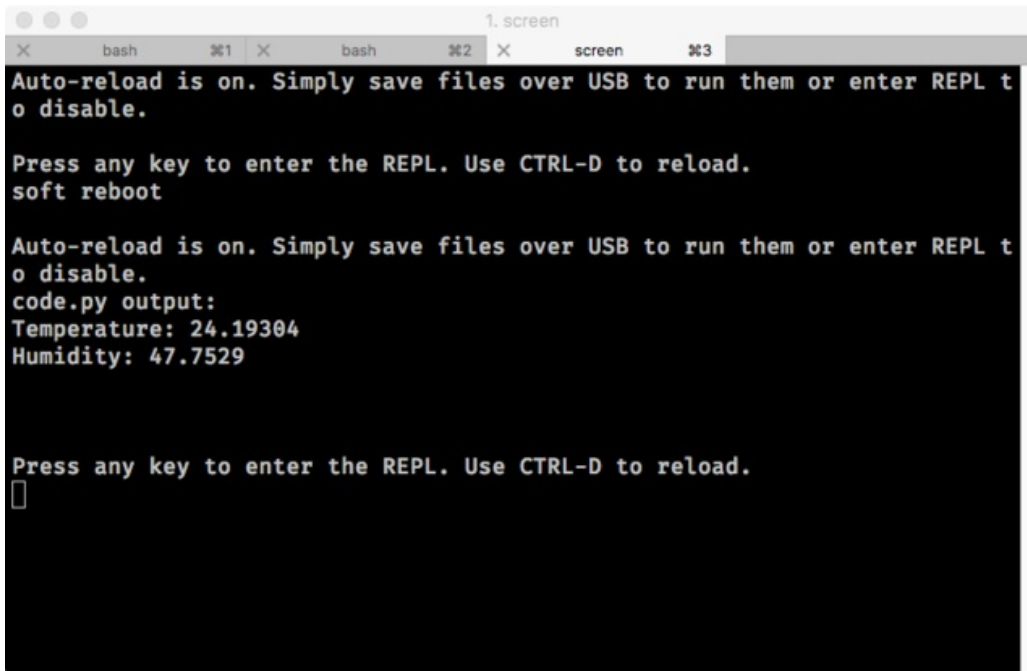
```

Oops! Another ImportError! Libraries can depend on other libraries so copying one file over may not be enough. Looking in the bundle, there is an adafruit_bus_device directory. In Python terms this is a package. It contains module files. Lets copy the folder over to make sure we get everything.



If that fails due to out of space, [see above](#). If not, continue on.

Lets check the serial connection again. Looks like it worked! We don't have any more `ImportErrors` and we can see the temperature (in Celsius) and the relative humidity.



CircuitPython Analog In

This quick-start example shows how you can read the analog voltages on all three Gemma M0 pads.

Copy and paste the code block into **main.py** using your favorite text editor, and save the file, to run the demo

```
# Gemma IO demo - analog inputs

from digitalio import *
from analogio import *
from board import *
import time

led = DigitalInOut(L)
led.direction = Direction.OUTPUT

analog0in = AnalogIn(A0)
analog1in = AnalogIn(A1)
analog2in = AnalogIn(A2)

def getVoltage(pin):
    return (pin.value * 3.3) / 65536

while True:
    print("A0: %f \t\t A1: %f \t\t A2: %f" %
          (getVoltage(analog0in),
           getVoltage(analog1in),
           getVoltage(analog2in)))

    time.sleep(0.1)
```

Creating analog inputs

```
analog0in = AnalogIn(A0)
analog1in = AnalogIn(A1)
analog2in = AnalogIn(A2)
```

Creates three objects, one for each pad, and connects the objects to A0, A1 and A2 as analog inputs.

GetVoltage Helper

`getVoltage(pin)` is our little helper program. By default, analog readings will range from 0 (minimum) to 65535 (maximum). This helper will convert the 0-65535 reading from **pin.value** and convert it to a 0-3.3V voltage reading.

Main Loop

The main loop is simple, it will just print out the three voltages as floating point values (the `%f` indicates to print as floating point) by calling `getVoltage` on each of our analog objects.

If you connect to the serial port REPL, you'll see the voltages printed out. By default the pins are *floating* so the voltages will vary. Try touching a wire from **A0** to the **GND** or **3Vo** pad to see the voltage change!

```
COM96 - PuTTY
A0: 1.297925      A1: 1.283473      A2: 3.299143
A0: 1.279595      A1: 1.274661      A2: 3.299193
A0: 1.259706      A1: 1.239463      A2: 3.299193
A0: 1.257490      A1: 1.267712      A2: 3.299143
A0: 1.255728      A1: 1.253008      A2: 3.299193
A0: 1.284983      A1: 1.251095      A2: 3.299193
A0: 1.282919      A1: 1.283120      A2: 3.299193
A0: 1.270028      A1: 1.274762      A2: 3.299143
A0: 1.272093      A1: 1.273251      A2: 3.299193
A0: 1.282365      A1: 1.291429      A2: 3.299193
A0: 1.257288      A1: 1.278488      A2: 3.299193
A0: 1.282717      A1: 1.257339      A2: 3.299193
A0: 1.261116      A1: 1.261166      A2: 3.299193
A0: 1.293846      A1: 1.294550      A2: 3.299193
A0: 1.275064      A1: 1.269172      A2: 3.299193
A0: 1.267309      A1: 1.285940      A2: 3.299193
A0: 1.286847      A1: 1.275618      A2: 3.299193
A0: 1.260511      A1: 1.291277      A2: 3.299193
A0: 1.279696      A1: 1.282868      A2: 3.299143
```

CircuitPython Analog Out

This quick-start example shows how you can set the DAC (true analog voltage output) on all Gemma M0 pad **A0**.

Copy and paste the code block into **main.py** using your favorite text editor, and save the file, to run the demo

```
# Gemma IO demo - analog output

from digitalio import *
from analogio import *
from board import *
import time

aout = AnalogOut(A0)

while True:
    # Count up from 0 to 65535, with 64 increment
    # which ends up corresponding to the DAC's 10-bit range
    for i in range (0,65535,64):
        aout.value = i
```

Creating an analog output

```
aout = AnalogOut(A0)
```

Creates an object **aout** that is connected to the only DAC pin available - A0.

Setting the analog output

The DAC on the Gemma M0 is a 10-bit output, from 0-3.3V. So in theory you will have a resolution of 0.0032 Volts per bit. To allow CircuitPython to be general-purpose enough that it can be used with chips with anything from 8 to 16-bit DACs, the DAC takes a 16-bit value and divides it down internally.

E.g. writing 0 will be the same as setting it to 0 - 0 Volts out

Writing 5000 is the same as setting it to $5000 / 64 = 78$
And $78 / 1024 * 3.3V = 0.25V$ output

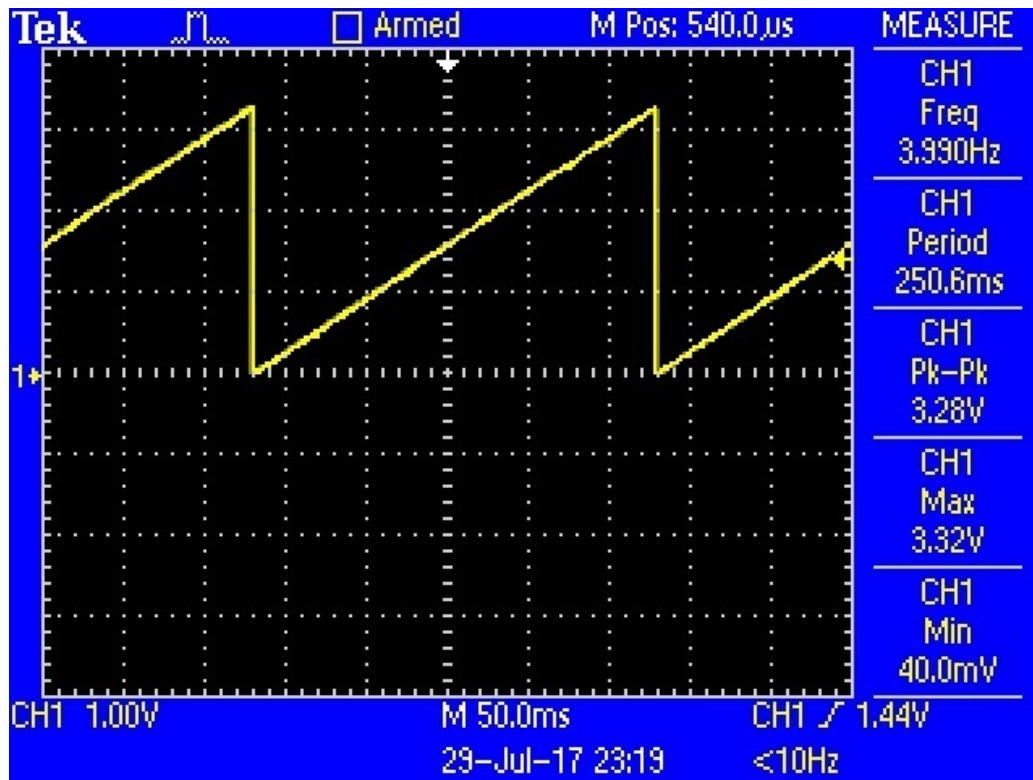
Writing 65535 is the same as 1023 which is the top range and you'll get 3.3V output

Main Loop

The main loop is fairly simple, it just goes through the entire range of the DAC, from 0 to 65535, but increments 64 at a time so it ends up clicking up one bit for each of the 10-bits of range available.

CircuitPython is not terribly fast, so at the fastest update loop you'll get 4 Hz. The DAC isn't good for audio outputs as-is.

Bigger boards like the Metro or Feather M0 have more code space and can perform audio playback capabilities via the DAC.



CircuitPython Cap Touch

This quick-start example shows how you can read the capacitive touch sensors built into on all three Gemma M0 pads.

Copy and paste the code block into **main.py** using your favorite text editor, and save the file, to run the demo

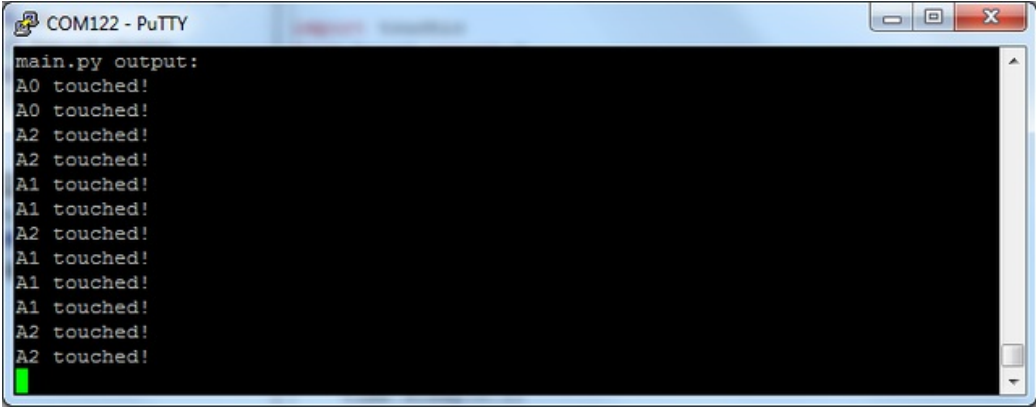
```
# Gemma IO demo - captouch

import touchio
from board import *
import time

touch0 = touchio.TouchIn(A0)
touch1 = touchio.TouchIn(A1)
touch2 = touchio.TouchIn(A2)

while True:
    if touch0.value:
        print("A0 touched!")
    if touch1.value:
        print("A1 touched!")
    if touch2.value:
        print("A2 touched!")
    time.sleep(0.01)
```

You can open up the serial console to see the touches detected and printed out.



```
COM122 - PuTTY
main.py output:
A0 touched!
A0 touched!
A2 touched!
A2 touched!
A1 touched!
A1 touched!
A2 touched!
A1 touched!
A1 touched!
A1 touched!
A2 touched!
A2 touched!
```

Creating an capacitive touch input

All three pads can be used as capacitive TouchIn devices:

```
touch0 = touchio.TouchIn(A0)
touch1 = touchio.TouchIn(A1)
touch2 = touchio.TouchIn(A2)
```

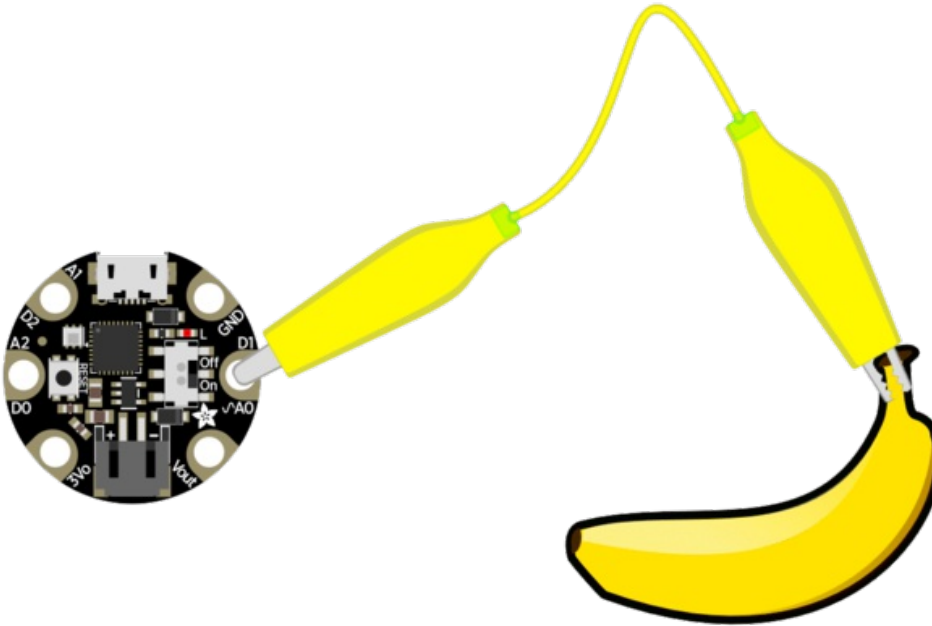
Creates three objects, one connected to each of the Gemma M0 pads.

Main Loop

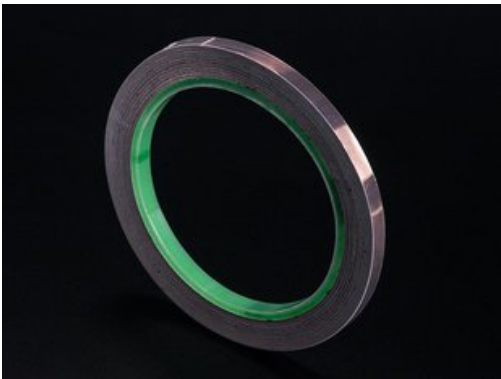
The main loop checks each sensor one after the other, to determine if it has been touched. If `touch0.value` returns True, that means that that pad, A0, detected a touch. For each pad, if it has been touched, a message will print.

A small sleep delay is added at the end so the loop doesn't run too fast. You may want to change the delay from 0.1 seconds to 0 seconds to slow it down or speed it up.

Note that no extra hardware is required, you can touch the pads directly, but you may want to attach alligator clips or foil tape to metallic or conductive objects. Try silverware, fruit or other food, liquid, aluminum foil, and items around your desk!



You may need to restart your code/board after changing the attached item because the capacitive touch code 'calibrates' based on what it sees when it first starts up. So if you get too many touch-signals or not enough, hit that reset button!



Copper Foil Tape with Conductive Adhesive - 6mm x 15 meter roll

PRODUCT ID: 1128

Copper tape can be an interesting addition to your toolbox. The tape itself is made of thin pure copper so its extremely flexible and can take on nearly any shape. You can easily solder...

<http://adafruit.it/eNZ>

\$5.95

IN STOCK



Copper Foil Tape with Conductive Adhesive - 25mm x 15 meter roll

PRODUCT ID: 1127

Copper tape can be an interesting addition to your toolbox. The tape itself is made of thin pure copper so its extremely flexible and can take on nearly any shape. You can easily solder...

<http://adafru.it/y8F>

\$19.95

IN STOCK



Small Alligator Clip Test Lead (set of 12)

PRODUCT ID: 1008

Connect this to that without soldering using these handy mini alligator clip test leads. 15" cables with alligator clip on each end, color coded. You get 12 pieces in 6 colors....

<http://adafru.it/dWJ>

\$3.95

OUT OF STOCK

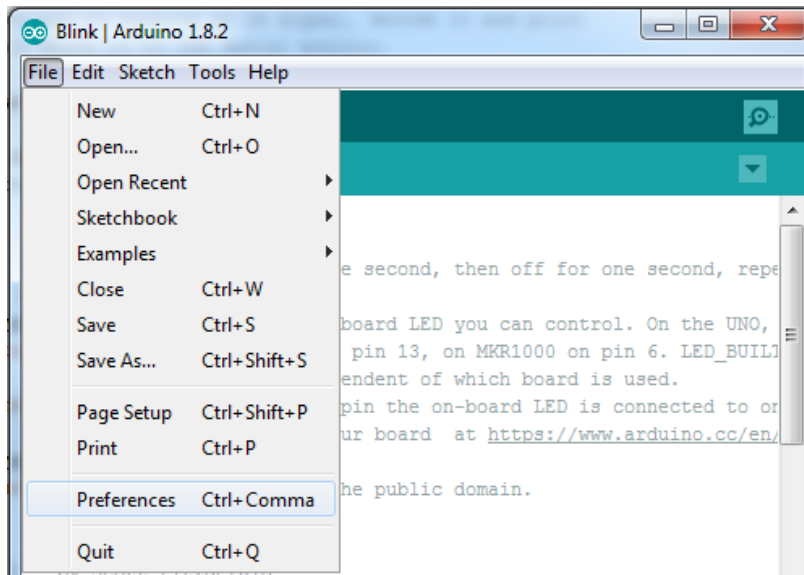
Arduino IDE Setup

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

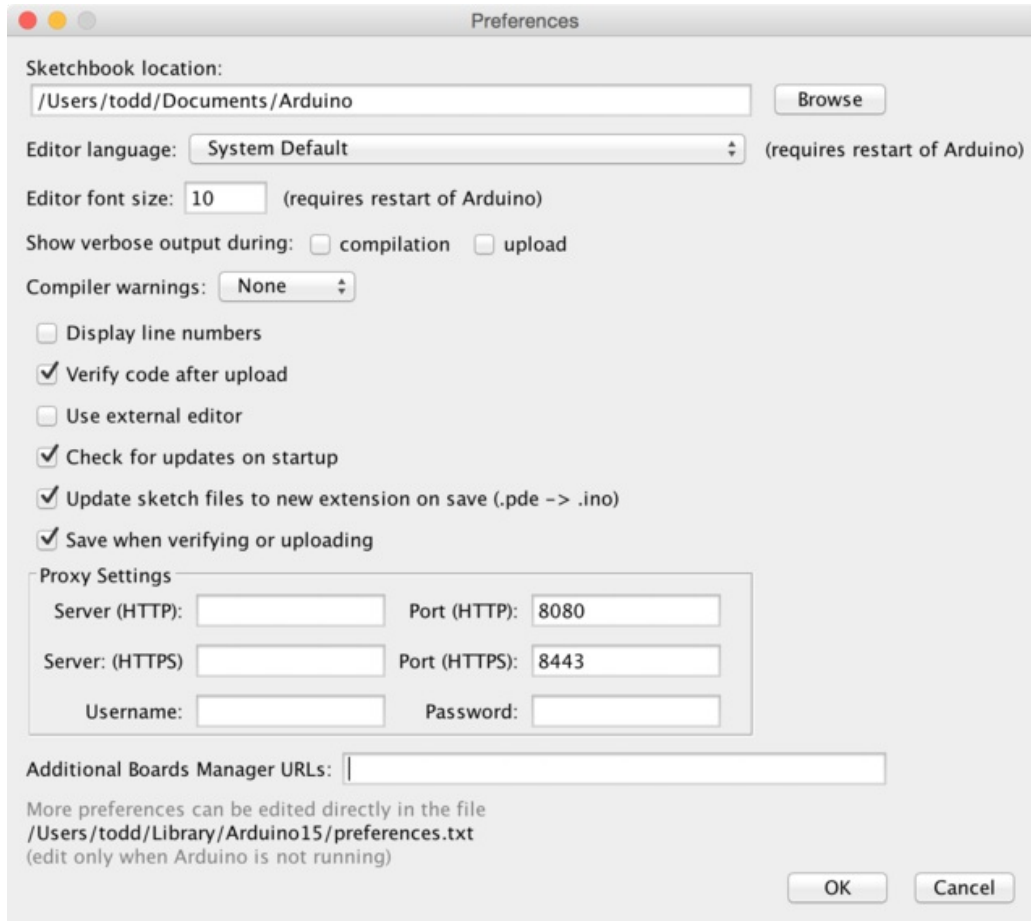
[Arduino IDE Download](http://adafru.it/f1P)

<http://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*.



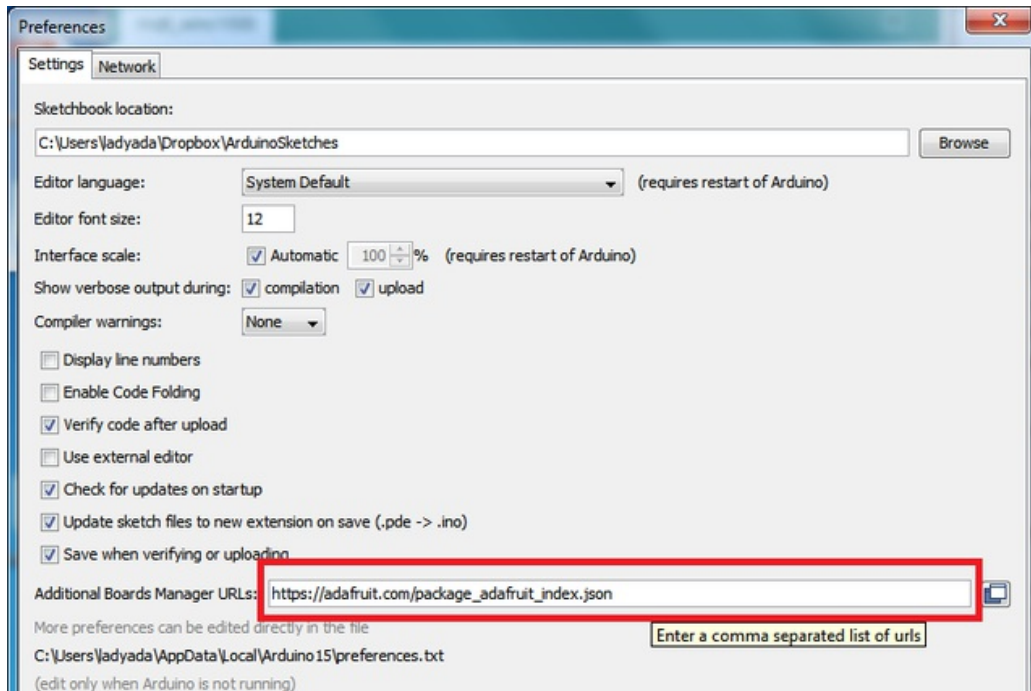
A dialog will pop up just like the one shown below.



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 \(http://adafru.it/f7U\)](http://adafru.it/f7U). We will only need to add one URL to the IDE in this example, but **you can add multiple URLs by separating them with commas**. Copy and paste the link below into the **Additional Boards Manager URLs** option in the Arduino IDE preferences.

https://adafruit.github.io/arduino-board-index/package_adafruit_index.json



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

- **Adafruit AVR Boards** - Includes support for Flora, Gemma, Feather 32u4, Trinket, & Trinket Pro.
- **Adafruit SAMD Boards** - Includes support for Feather M0, 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 \(http://adafru.it/eSI\)](http://adafru.it/eSI).

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

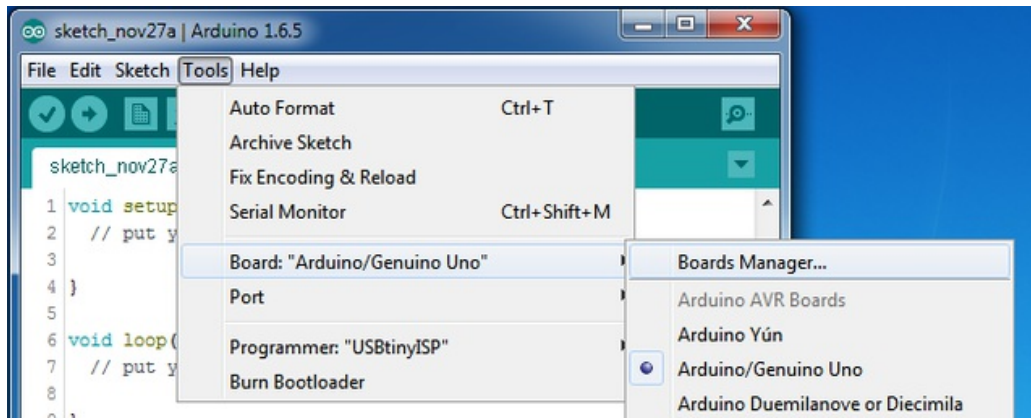
Once done click **OK** to save the new preference settings. Next we will look at installing boards with the Board Manager.

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

Using with Arduino IDE

Since the Feather/Metro/Gemma/Trinket M0 use an ATSAM21 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.

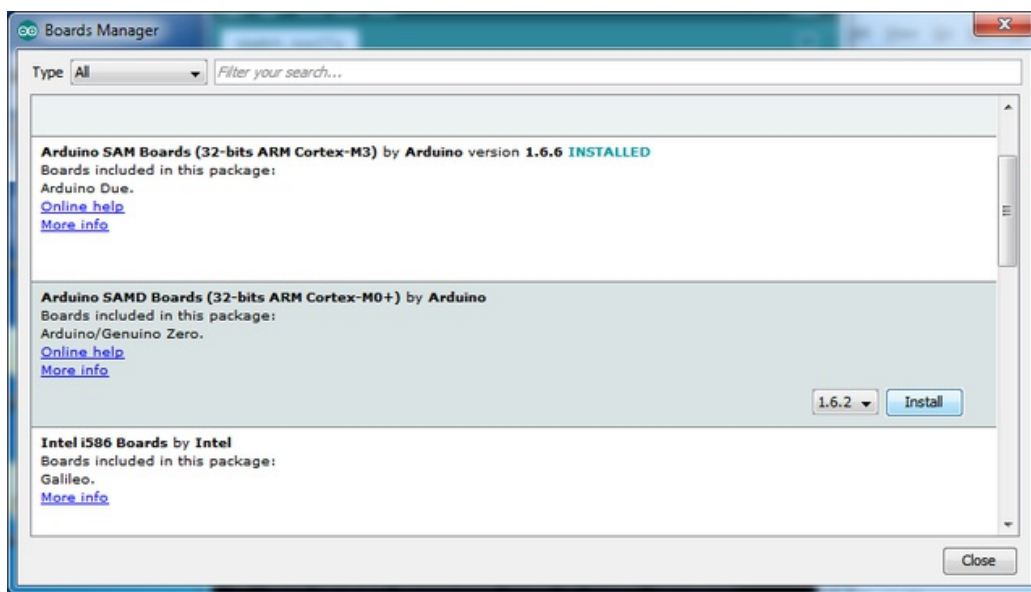


Once the Board Manager opens, click on the category drop down menu on the top left hand side of the window and select **Contributed**. You will then be able to select and install the boards supplied by the URLs added to the preferences.

Install SAMD Support

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

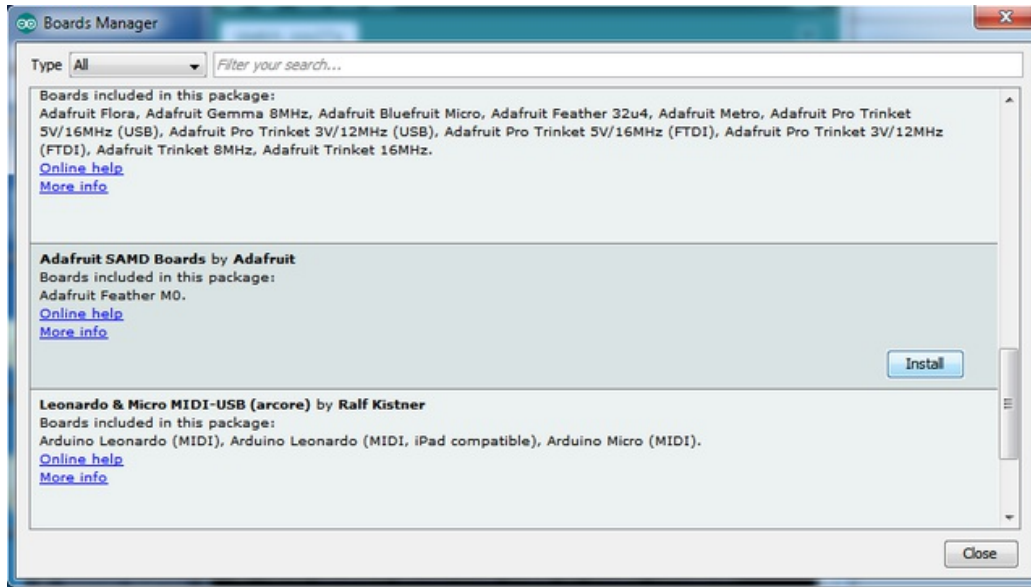
You can type **Arduino SAMD** in the top search bar, then when you see the entry, click **Install**



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**

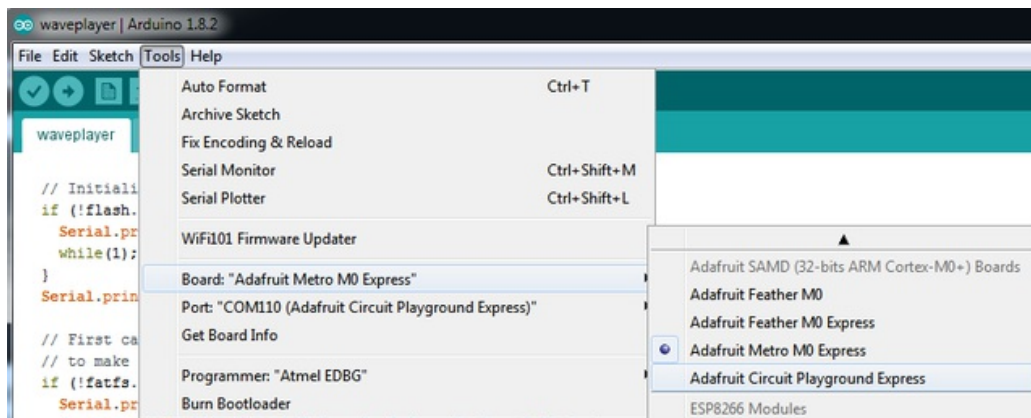


Even though in theory you don't need to - I recommend rebooting the IDE

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**



Install Drivers (Windows 7 Only)

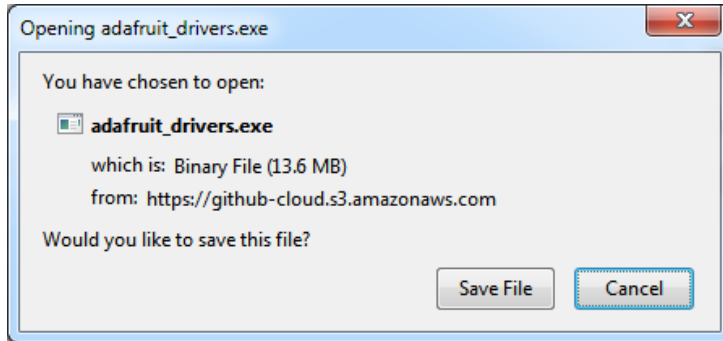
When you plug in the board, you'll need to possibly install a driver

Click below to download our Driver Installer

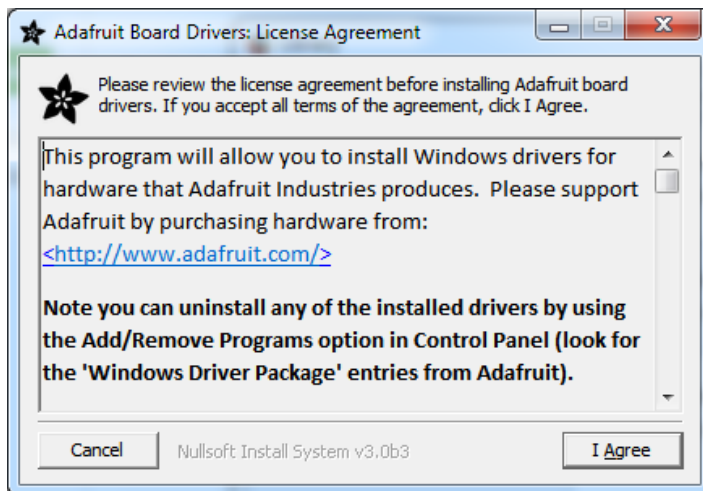
[Download Adafruit Driver Installer v1.3](#)

<http://adafru.it/x2d>

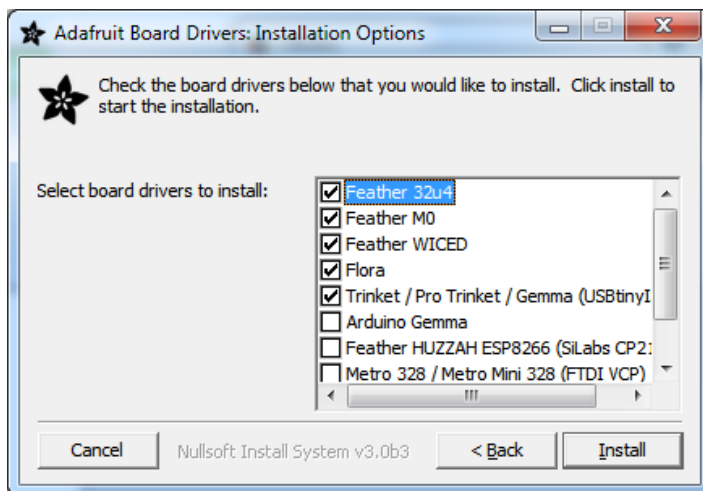
Download and run the installer



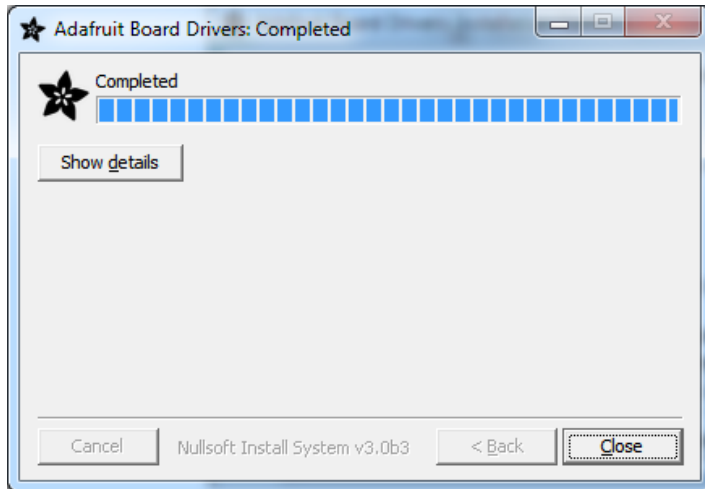
Run the installer! Since we bundle the SiLabs and FTDI drivers as well, you'll need to click through the license



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



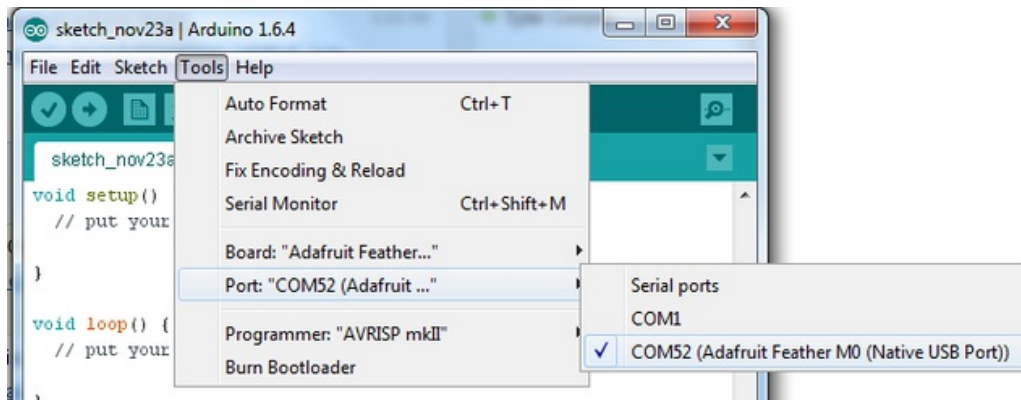
Click **Install** to do the installin'



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 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);
}

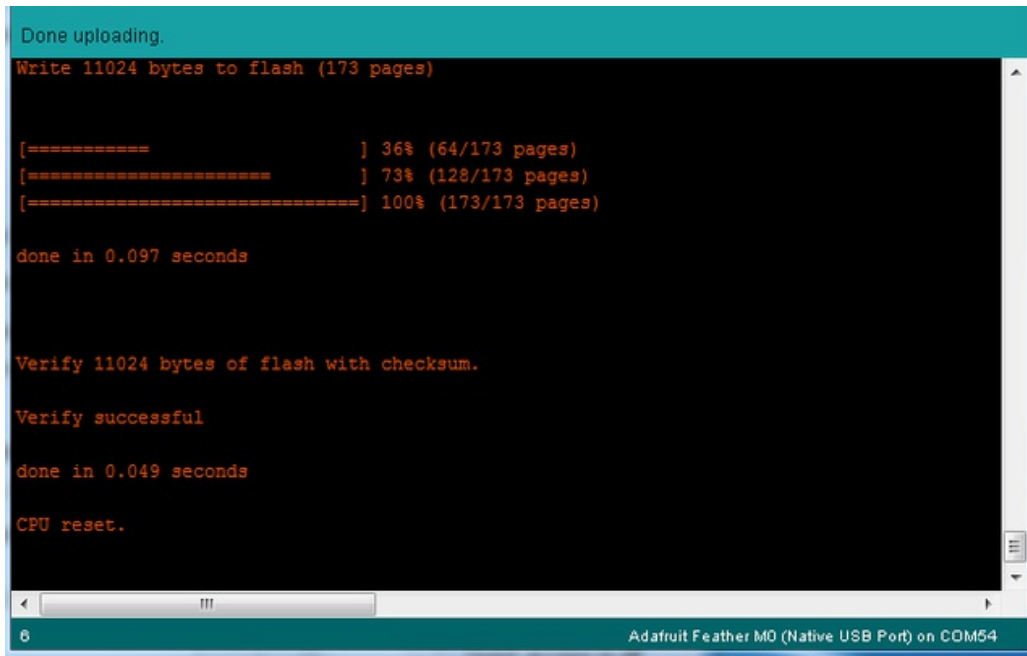
// the loop function runs over and over again forever
void loop() {
  digitalWrite(13, HIGH); // turn the LED on (HIGH is the voltage level)
  delay(1000);           // wait for a second
  digitalWrite(13, LOW); // turn the LED off by making the voltage LOW
  delay(1000);          // wait for a second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

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.

Successful Upload

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



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

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

done in 0.097 seconds

Verify 11024 bytes of flash with checksum.

Verify successful

done in 0.049 seconds

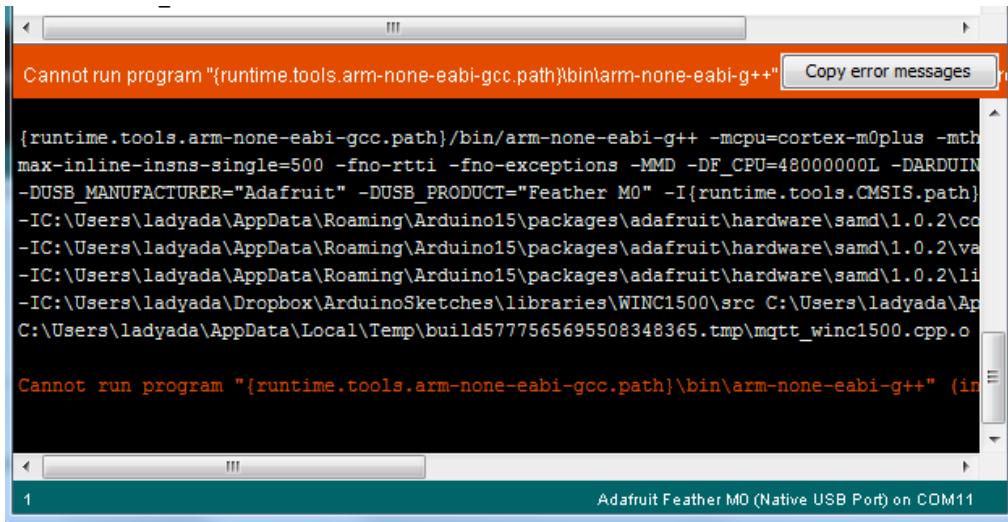
CPU reset.
```

Compilation Issues

If you get an alert that looks like

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

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

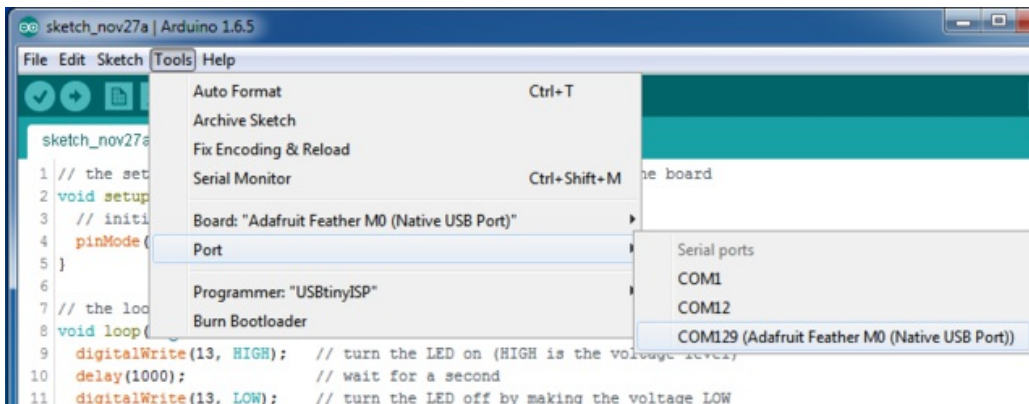


Manually bootloading

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

The red LED will pulse, so you know that its in bootloader mode.

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



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

Ubuntu & Linux Issue Fix

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

The fix for this issue is to make sure Adafruit's custom udev rules are applied to your system. One of these rules is made to configure modem manager not to touch the Feather board and will fix the programming difficulty issue.

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

Adapting Sketches to M0

The ATSAM21 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 `isAnalogReference(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:

A screenshot of the Arduino IDE window titled "datecalc | Arduino 1.6.5". The menu bar includes "File", "Edit", "Sketch", "Tools", and "Help". Below the menu bar is a toolbar with icons for saving, opening, and other functions. The main text area shows a C++ sketch for date calculations. The code includes comments and preprocessor directives for SAMD-based boards, and a function named showDate that prints text and a date-time object to the serial console.

```
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 DateTime dt) {
12     Serial.print(txt);
13     Serial.print(' ');
```

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)**

Missing header files

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

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

you'll get an error that says

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

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

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

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

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

Bootloader Launching

For most other AVR's, clicking **reset** while plugged into USB will launch the bootloader manually, the bootloader will time out after a few seconds. For the M0, 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> (<http://adafru.it/IFS>)

How Much RAM Available?

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

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

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

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

Storing data in FLASH

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

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

That string is now in FLASH. You can manipulate the string just like RAM data, the compiler will automatically read from FLASH so you dont need special progmem-knowledgeable functions.

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

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

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

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 `avrdude`), one can simply **drag a file onto a removable drive**.

The format of the file is a little special. Due to 'operating system woes' you cannot just drag a binary or hex file (trust us, we tried it, it isn't cross-platform compatible). Instead, the format of the file has extra information to help the bootloader know where the data goes. The format is called UF2 (USB Flashing Format). Microsoft MakeCode generates UF2s for flashing and CircuitPython releases are also available as UF2. [You can also create your own UF2s from binary files using uf2tool, available here. \(http://adafru.it/vPE\)](http://adafru.it/vPE)

The bootloader is **also BOSSA compatible**, so it can be used with the Arduino IDE which expects a BOSSA bootloader on ATSAMD-based boards

For more information about UF2, [you can read a bunch more at the MakeCode blog \(http://adafru.it/w5A\)](http://adafru.it/w5A), then [check out the UF2 file format specification \(http://adafru.it/vPE\)](http://adafru.it/vPE) and to build your *own* bootloader for ATSAMD-based boards, visit [Microsoft UF2-SAMD github repository \(http://adafru.it/vPF\)](http://adafru.it/vPF).

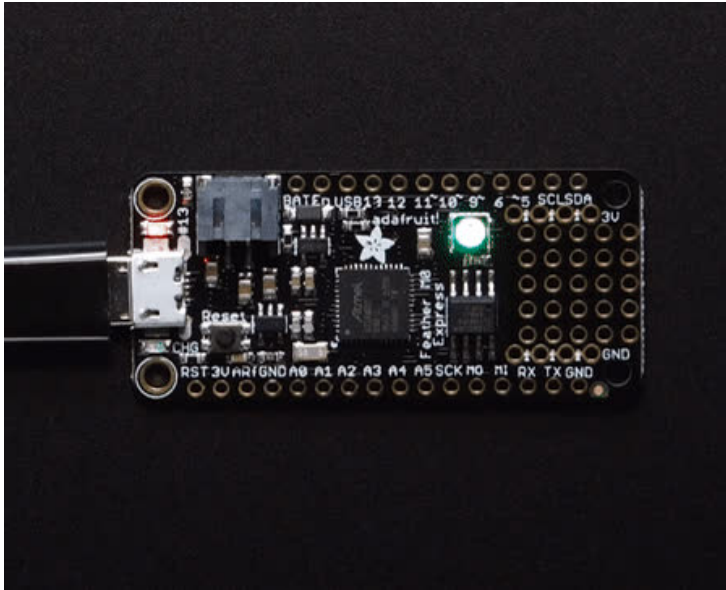
The bootloader is not needed when changing your CircuitPython code. Its only needed when upgrading the CircuitPython core or changing between CircuitPython, Arduino and Microsoft MakeCode.

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**, and Gemma M0 will show up as **GEMMABOOT**

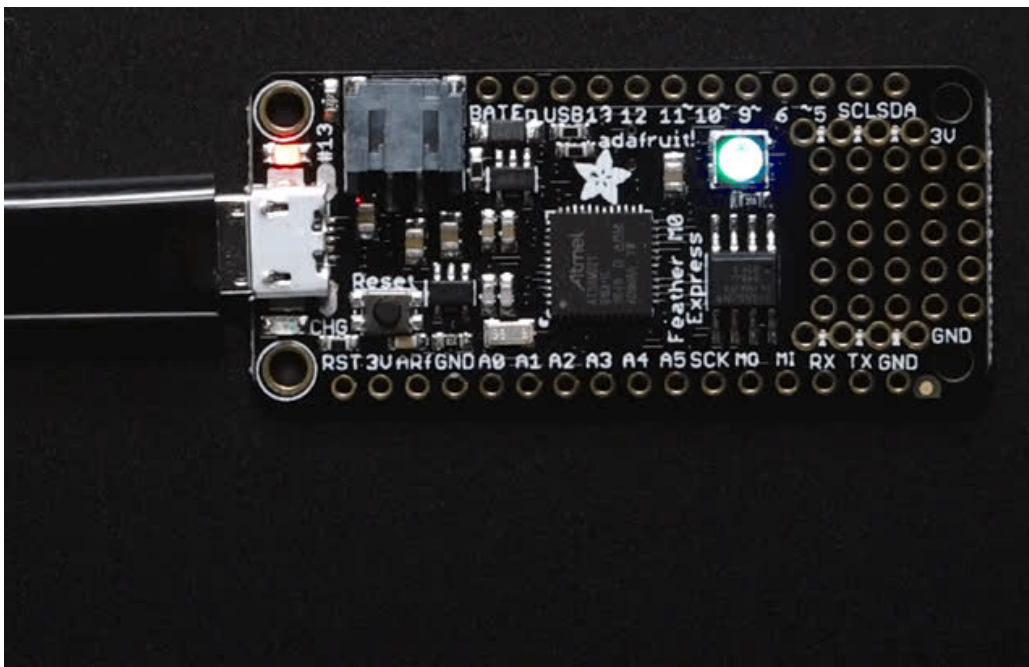
Furthermore, when the bootloader is active, it will change the color of one or more onboard neopixels to indicate the connection status, red for disconnected and green for connected. If the board is plugged in but still showing that its disconnected, try a different USB cable. Some cables only provide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



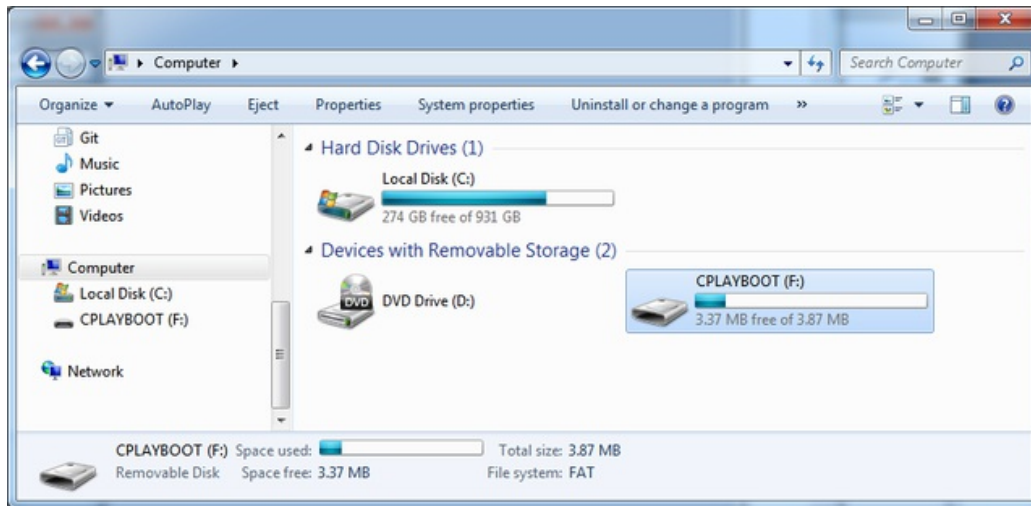
If the bootloader couldn't start, you will get a red NeoPixel LED.

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!



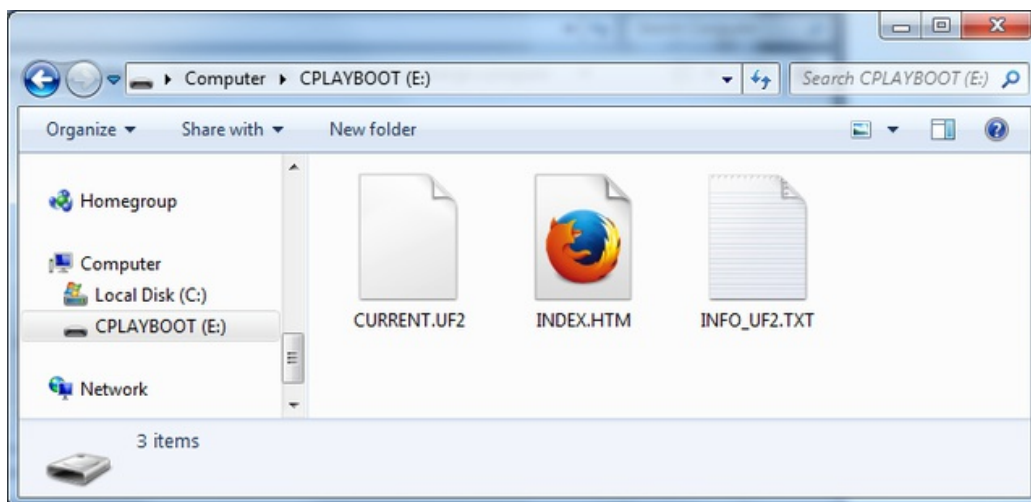
The first versions of the Feather M0 Express have a bootloader that may be unreliable, especially when used through a USB Hub. If the board doesn't connect through a USB hub or connects but then disconnects, then please update the bootloader with the instructions below.

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



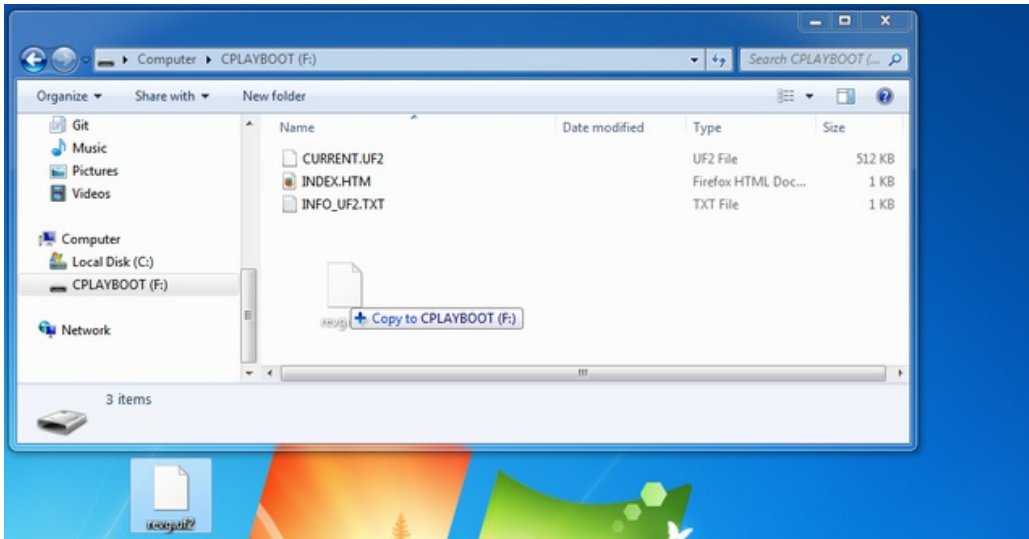
Once the bootloader is successfully connected you can open the drive and browse the virtual filesystem. This isn't the same filesystem as you use with CircuitPython or Arduino. It should have three files:

- **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.

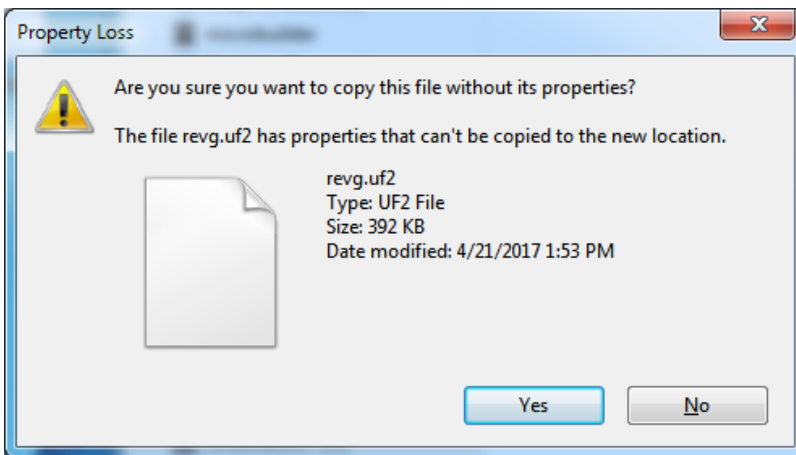


Using the Mass Storage Bootloader

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 \(http://adafru.it/vQa\)](http://adafru.it/vQa).

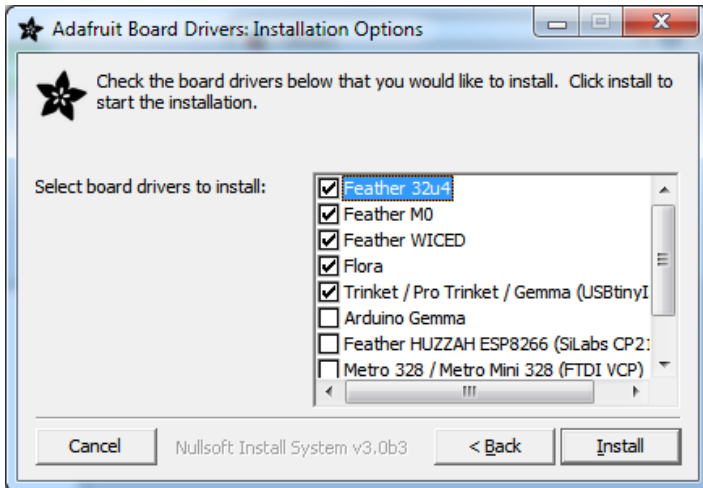
Windows 7 Drivers

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

You can download our full driver package here:

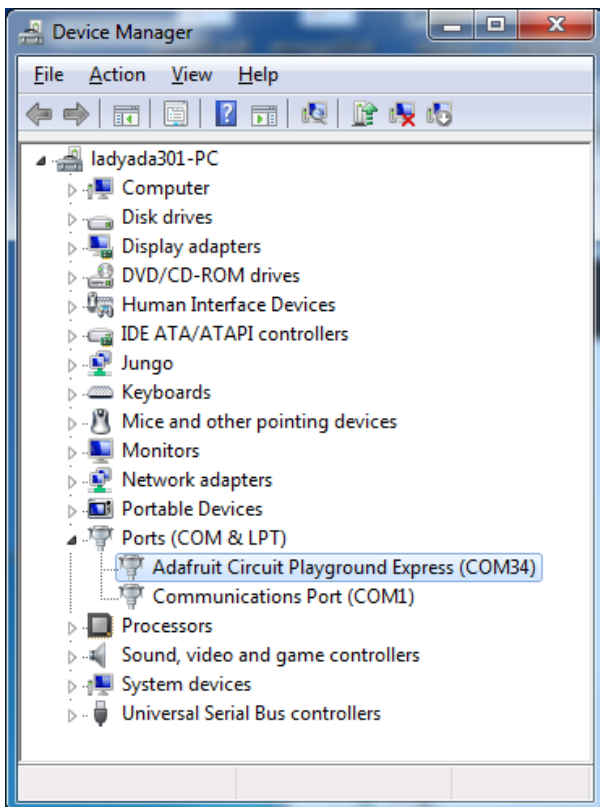
[Download Adafruit Driver Installer v1.1](http://adafru.it/vA7)
<http://adafru.it/vA7>

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

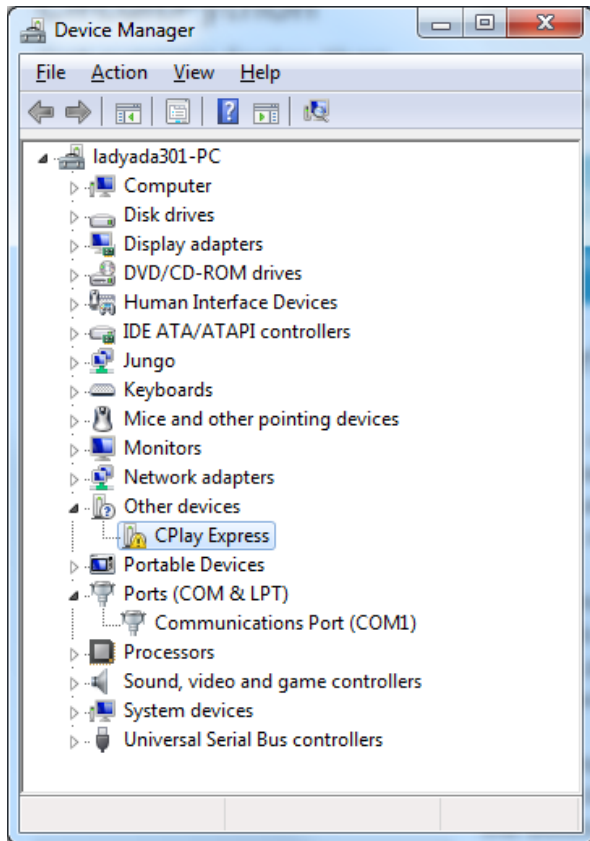


Verifying Serial Port in Device Manager

If you're running Windows, its a good idea to verify the device showed up. Open your Device Manager from the control panel and look under **Ports (COM & LPT)** for a device called **Feather M0** or **Circuit Playground** or whatever!



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



Running bossac on the command line

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

[You can download the latest builds here \(http://adafru.it/s1B\)](http://adafru.it/s1B) The mingw32 version is for Windows, apple-darwin for Mac OSX and various linux options for Linux. Once downloaded, extract the files from the zip and open the command line to the directory with bossac

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 -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. A hard power-reset is also recommended after **bossac**, just in case.

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

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

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

Updating the bootloader

The UF2 bootloader is still in beta, 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 \(http://adafru.it/tBa\)](http://adafru.it/tBa).

The latest updater for Feather M0 Express:

- [Feather M0 v1.22.0 Update UF2](http://adafru.it/wJf)
- [Metro M0 v1.22.0 Update UF2](http://adafru.it/wJA)

Getting Rid of Windows Pop-ups

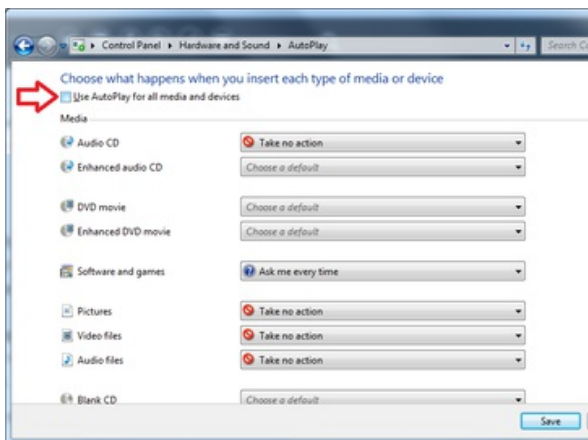
If you do a *lot* of development on Windows with the UF2 bootloader, you may get annoyed by the constant "Hey you inserted a drive what do you want to do" pop-ups.



Go to the Control Panel. Click on the **Hardware and Sound** header



Click on the **AutoPlay** header



Uncheck the box at the top, labeled **Use AutoPlay for all devices**

Making your own UF2

Making your own UF2 is easy! All you need is a .bin file of a program you wish to flash and [the Python conversion script](http://adafru.it/vZb) (<http://adafru.it/vZb>). Make sure that your program was compiled to start at 0x2000 (8k) because the bootloader takes the first 8k. CircuitPython's [linker script](http://adafru.it/vZc) (<http://adafru.it/vZc>) 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.

Downloads

Files:

- [ATSAMD21 Datasheet \(http://adafru.it/xZe\)](http://adafru.it/xZe)
- [Webpage for the ATSAMD21E18 \(main chip used\) \(http://adafru.it/xZf\)](http://adafru.it/xZf)
- [EagleCAD files on GitHub \(http://adafru.it/xZA\)](http://adafru.it/xZA)

[releasefiles-55c8eb5.zip](#)

<http://adafru.it/yb8>

Schematic & Fabrication Print

