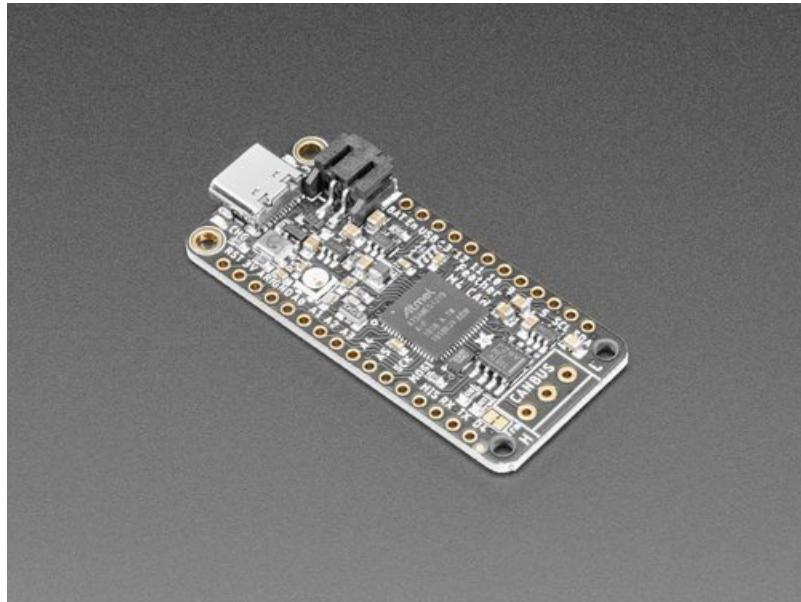




Adafruit Feather M4 CAN Express

Created by Kattni Rembor



Last updated on 2021-05-19 05:50:44 PM EDT

Guide Contents

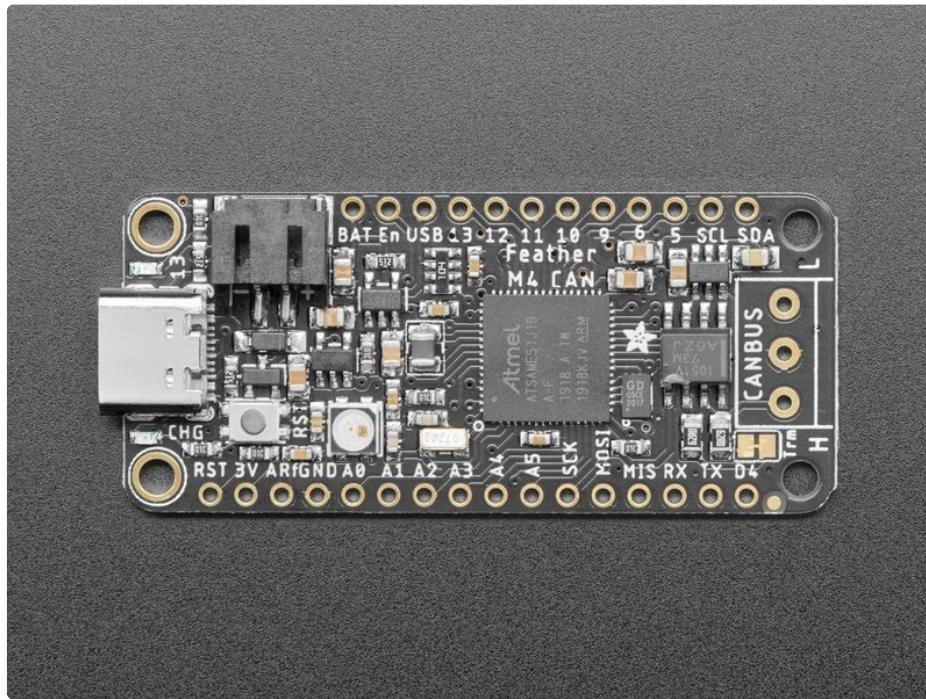
| | |
|---|----|
| Guide Contents | 2 |
| Overview | 6 |
| Pinouts | 11 |
| Power | 11 |
| CAN Bus | 12 |
| Logic Pins | 13 |
| I2S Pins: | 14 |
| Parallel Capture Peripheral | 14 |
| NeoPixel and LEDs | 15 |
| QSPI Flash | 16 |
| Other Pins | 16 |
| Debugging Interface | 17 |
| Arduino IDE Setup | 18 |
| https://adafruit.github.io/arduino-board-index/package_adafruit_index.json | 19 |
| Using with Arduino IDE | 21 |
| Install SAMD Support | 21 |
| Install Adafruit SAMD | 22 |
| Install Drivers (Windows 7 & 8 Only) | 23 |
| Blink | 25 |
| Successful Upload | 26 |
| Compilation Issues | 27 |
| Manually bootloading | 27 |
| Ubuntu & Linux Issue Fix | 28 |
| Adapting Sketches to M0 & M4 | 29 |
| Analog References | 29 |
| Pin Outputs & Pullups | 29 |
| Serial vs SerialUSB | 29 |
| AnalogWrite / PWM on Feather/Metro M0 | 30 |
| analogWrite() PWM range | 31 |
| analogWrite() DAC on A0 | 31 |
| Missing header files | 31 |
| Bootloader Launching | 32 |
| Aligned Memory Access | 32 |
| Floating Point Conversion | 32 |
| How Much RAM Available? | 33 |
| Storing data in FLASH | 33 |
| Pretty-Printing out registers | 33 |
| M4 Performance Options | 34 |
| CPU Speed (overclocking) | 34 |
| Optimize | 35 |
| Cache | 35 |
| Max SPI and Max QSPI | 35 |
| Enabling the Buck Converter on some M4 Boards | 36 |
| Arduino CAN Examples | 37 |
| Install the Arduino Library | 37 |

| | |
|--|----|
| Transmission & Reception Examples | 38 |
| Bi-Directional Communication Demo | 42 |
| CircuitPython on Feather M4 CAN | 47 |
| Set up CircuitPython Quick Start! | 47 |
| Installing Mu Editor | 49 |
| Download and Install Mu | 49 |
| Using Mu | 49 |
| Creating and Editing Code | 51 |
| Creating Code | 51 |
| Editing Code | 53 |
| Your code changes are run as soon as the file is done saving. | 54 |
| 1. Use an editor that writes out the file completely when you save it. | 54 |
| 2. Eject or Sync the Drive After Writing | 55 |
| Oh No I Did Something Wrong and Now The CIRCUITPY Drive Doesn't Show Up!!! | 55 |
| Back to Editing Code... | 56 |
| Exploring Your First CircuitPython Program | 57 |
| Imports & Libraries | 57 |
| Setting Up The LED | 57 |
| Loop-de-loops | 58 |
| What Happens When My Code Finishes Running? | 58 |
| What if I don't have the loop? | 59 |
| More Changes | 60 |
| Naming Your Program File | 60 |
| Connecting to the Serial Console | 61 |
| Are you using Mu? | 61 |
| Setting Permissions on Linux | 62 |
| Using Something Else? | 63 |
| Interacting with the Serial Console | 64 |
| The REPL | 68 |
| Returning to the serial console | 72 |
| CircuitPython Libraries | 73 |
| Installing the CircuitPython Library Bundle | 74 |
| Example Files | 75 |
| Copying Libraries to Your Board | 76 |
| Example: ImportError Due to Missing Library | 76 |
| Library Install on Non-Express Boards | 77 |
| Updating CircuitPython Libraries/Examples | 78 |
| Frequently Asked Questions | 79 |
| I have to continue using an older version of CircuitPython; where can I find compatible libraries? | 79 |
| Is ESP8266 or ESP32 supported in CircuitPython? Why not? | 79 |
| How do I connect to the Internet with CircuitPython? | 80 |
| Is there asyncio support in CircuitPython? | 81 |
| My RGB NeoPixel/DotStar LED is blinking funny colors - what does it mean? | 82 |
| What is a MemoryError? | 83 |
| What do I do when I encounter a MemoryError? | 83 |
| Can the order of my import statements affect memory? | 84 |
| How can I create my own .mpy files? | 84 |
| How do I check how much memory I have free? | 84 |
| Does CircuitPython support interrupts? | 84 |
| Does Feather M0 support WINC1500? | 85 |
| Can AVR's such as ATmega328 or ATmega2560 run CircuitPython? | 85 |

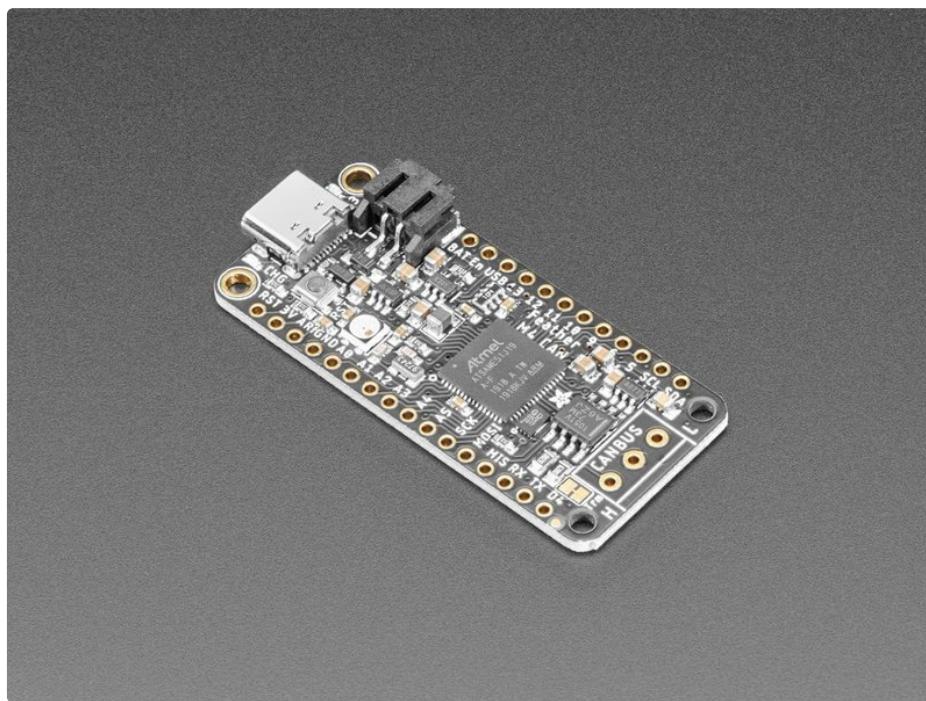
| | |
|---|-----|
| Commonly Used Acronyms | 85 |
| Welcome to the Community! | 86 |
| Adafruit Discord | 86 |
| Adafruit Forums | 87 |
| Adafruit Github | 88 |
| ReadTheDocs | 89 |
| Advanced Serial Console on Windows | 91 |
| Windows 7 Driver | 91 |
| What's the COM? | 91 |
| Install Putty | 92 |
| Advanced Serial Console on Mac and Linux | 94 |
| What's the Port? | 94 |
| Connect with screen | 96 |
| Permissions on Linux | 97 |
| Uninstalling CircuitPython | 100 |
| Backup Your Code | 100 |
| Moving Circuit Playground Express to MakeCode | 100 |
| Moving to Arduino | 101 |
| Troubleshooting | 104 |
| Always Run the Latest Version of CircuitPython and Libraries | 104 |
| I have to continue using CircuitPython 5.x, 4.x, 3.x or 2.x, where can I find compatible libraries? | 104 |
| CPLAYBOOT, TRINKETBOOT, FEATHERBOOT, or GEMMABOOT Drive Not Present | 104 |
| You may have a different board. | 105 |
| MakeCode | 105 |
| MacOS | 105 |
| Windows 10 | 105 |
| Windows 7 or 8.1 | 105 |
| Windows Explorer Locks Up When Accessing boardnameBOOT Drive | 106 |
| Copying UF2 to boardnameBOOT Drive Hangs at 0% Copied | 107 |
| CIRCUITPY Drive Does Not Appear | 107 |
| Windows 7 and 8.1 Problems | 107 |
| Serial Console in Mu Not Displaying Anything | 107 |
| CircuitPython RGB Status Light | 108 |
| ValueError: Incompatible .mpy file. | 109 |
| CIRCUITPY Drive Issues | 109 |
| Easiest Way: Use storage.erase_filesystem() | 110 |
| Old Way: For the Circuit Playground Express, Feather M0 Express, and Metro M0 Express: | 110 |
| Old Way: For Non-Express Boards with a UF2 bootloader (Gemma M0, Trinket M0): | 112 |
| Old Way: For non-Express Boards without a UF2 bootloader (Feather M0 Basic Proto, Feather Adalogger, Arduino Zero): | 112 |
| Running Out of File Space on Non-Express Boards | 112 |
| Delete something! | 112 |
| Use tabs | 113 |
| MacOS loves to add extra files. | 113 |
| Prevent & Remove MacOS Hidden Files | 113 |
| Copy Files on MacOS Without Creating Hidden Files | 114 |
| Other MacOS Space-Saving Tips | 115 |
| Device locked up or boot looping | 116 |

| | |
|-----------------------------|-----|
| CircuitPython CANio Library | 117 |
| Downloads | 118 |
| Files: | 118 |
| Schematic | 118 |
| Fab Print | 118 |

Overview



One of our favorite Feathers, the Feather M4 Express, gets a glow-up here with an upgrade to the SAME51 chipset which has built-in CAN bus support! Like its SAMD51 cousin, the **ATSAME5J19** comes with a **120MHz Cortex M4** with floating point support and **512KB Flash and 192KB RAM**. Your code will zig and zag and zoom, and with a bunch of extra peripherals for support, this will for sure be your favorite new chipset for CAN interfacing projects.

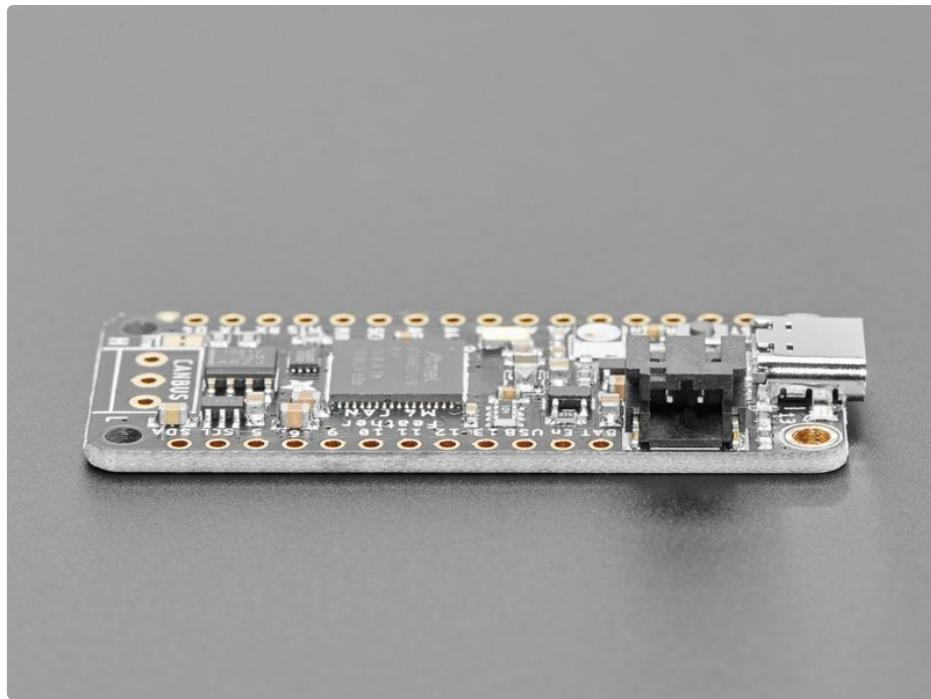


At the end of the board we have placed a CAN transceiver chip as well as a 5V converter to generate 5V power to the transceiver even when running on battery. The two CAN signal lines and ground reference signal are available on a 3-pin 3.5mm terminal block. The chip and booster can be put to sleep for power saving. The built in CAN can read or write packets and has support in both Arduino and CircuitPython.

And best of all, it's a Feather - so you know it will work with all our FeatherWings! What a great way to quickly get up and running. It's even pin-compatible with the original Feather M4.

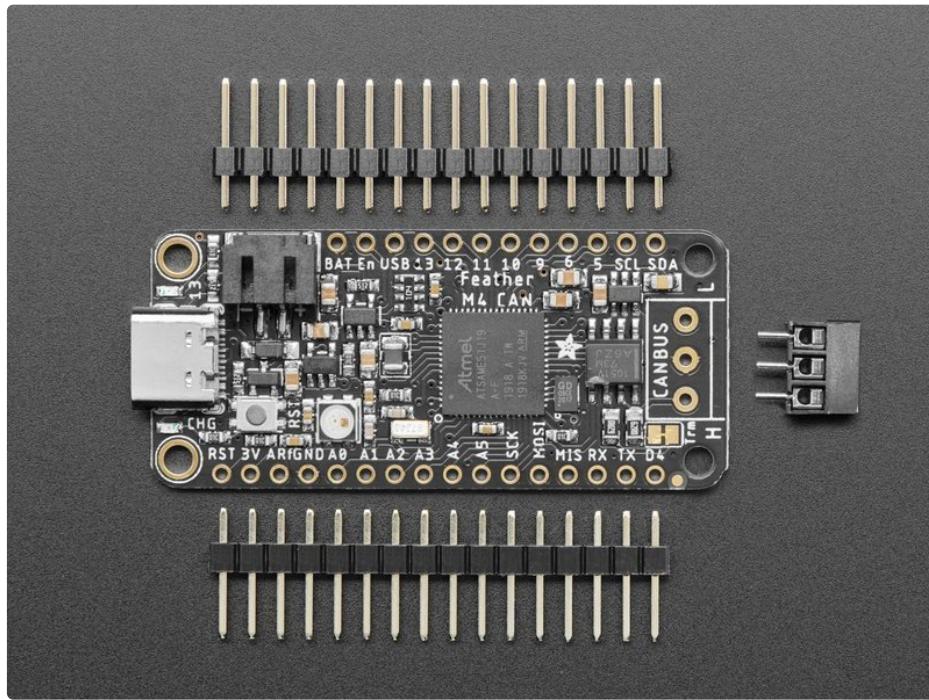


The most exciting part of the Feather M4 CAN is that while you can use it with the Arduino IDE - and it's bonkers fast when you do, you can use it with CircuitPython as well. When you plug it in, it will show up as a very small disk drive with `main.py` on it. Edit `main.py` with your favorite text editor 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 Feather and your code will go with you.

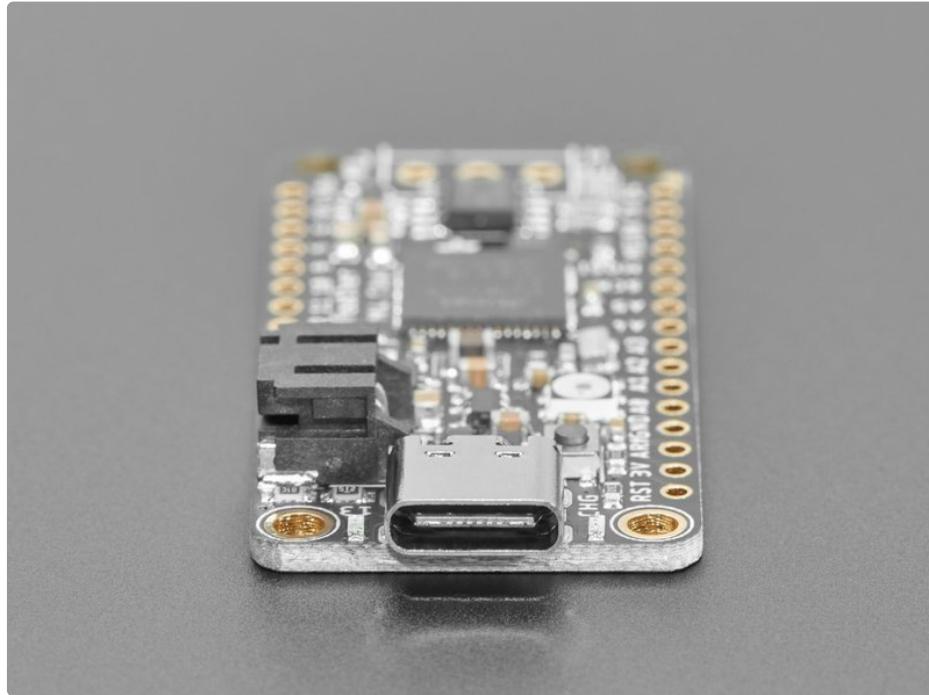


Here are some of the updates you can look forward to when using Feather M4 CAN:

- Measures 2.0" x 0.9" x 0.28" (50.8mm x 22.8mm x 7mm) without headers soldered in
- Light as a (large?) feather - 5 grams
- ATSAME51 32-bit Cortex M4 core running at 120 MHz, 32-bit, 3.3V logic and power
- **Hardware CAN bus support with built-in transceiver, 5V booster and terminal connection.**
- [Floating point support with Cortex M4 DSP instructions. \(https://adafru.it/ENz\)](https://adafru.it/ENz)
- 512 KB flash, 192 KB RAM
- 2 MB SPI FLASH chip for storing files and CircuitPython code storage.
- No EEPROM
- 32.768 KHz crystal for clock generation & RTC
- 3.3V regulator with 500mA peak current output
- **USB Type C** connector for USB native support, comes with USB bootloader and serial port debugging
- Built in crypto engines with AES (256 bit), true RNG, Pubkey controller
- Tons of GPIO! 21 x GPIO pins with following capabilities:
 - Dual 1 MSPS 12 bit true analog DAC (A0 and A1) - can be used to play 12-bit stereo audio clips
 - Dual 1 MSPS 12 bit ADC (6 analog pins some on ADC1 and some on ADC2)
 - 6 x hardware SERCOM - Native hardware SPI, I2C and Serial all available
 - 16 x PWM outputs - for servos, LEDs, etc
 - I2S input and output
 - 8-bit Parallel capture controller (for camera/video in)
- Built in 100mA lipoly charger with charging status indicator LED
- Pin #13 red LED for general purpose blinking
- Power/enable pin
- 4 mounting holes
- Reset button



The **Feather M4 Can Express** also comes with a **Mini NeoPixel** and **2 MB SPI Flash** storage. You can use the SPI Flash storage like a very tiny hard drive. When used in CircuitPython, 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.

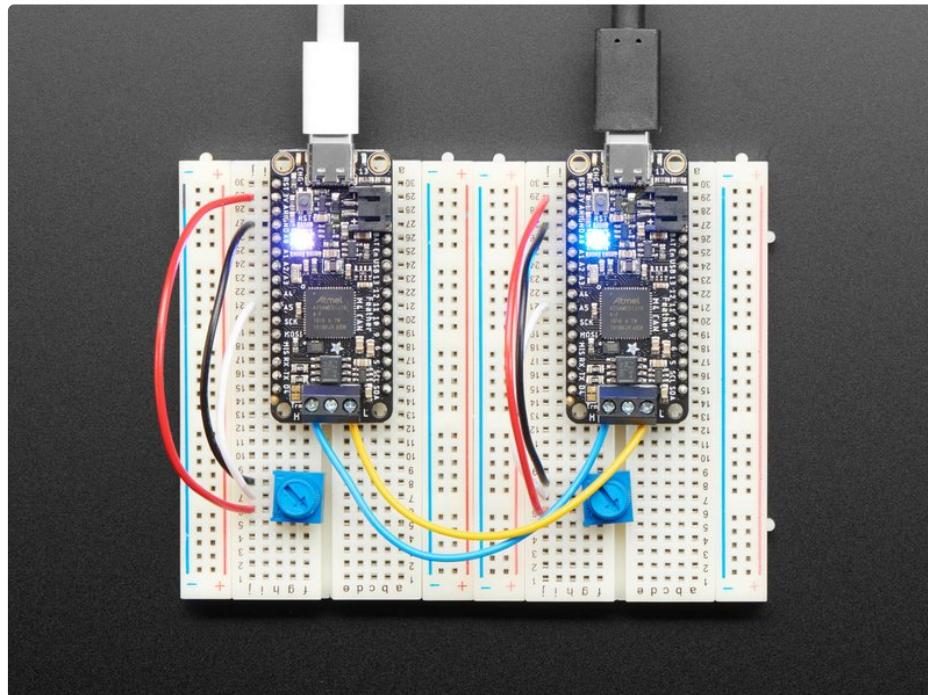


Easy reprogramming: the Feather M4 CAN comes pre-loaded with the [UF2 bootloader](#) (<https://adafru.it/vQd>), which looks like a USB storage key. Simply drag firmware on to program,

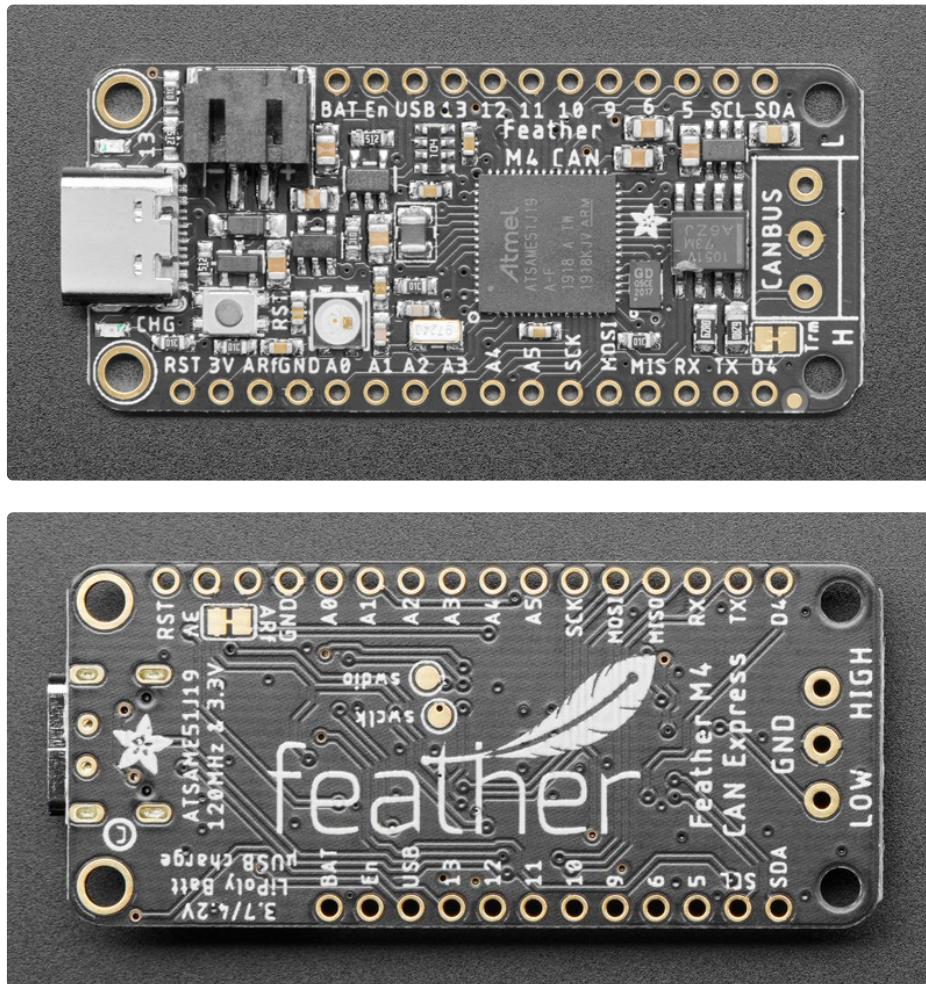
no special tools or drivers needed! It can be used to load up CircuitPython or Arduino IDE (it is bossa-compatible)

Comes fully assembled and tested, with the UF2 USB bootloader. We also toss in some headers so you can solder it in and plug into a solderless breadboard.

Lipoly battery and USB cable not included (but we do have lots of options in the shop if you'd like!)



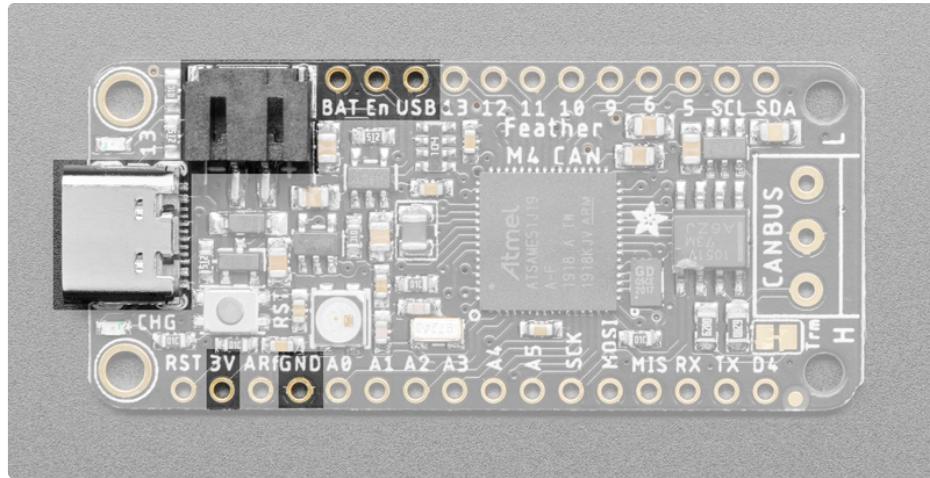
Pinouts



The Feather M4 CAN Express is loaded with microcontroller and CAN goodness. It also has many pins and ports. Let's take a tour!

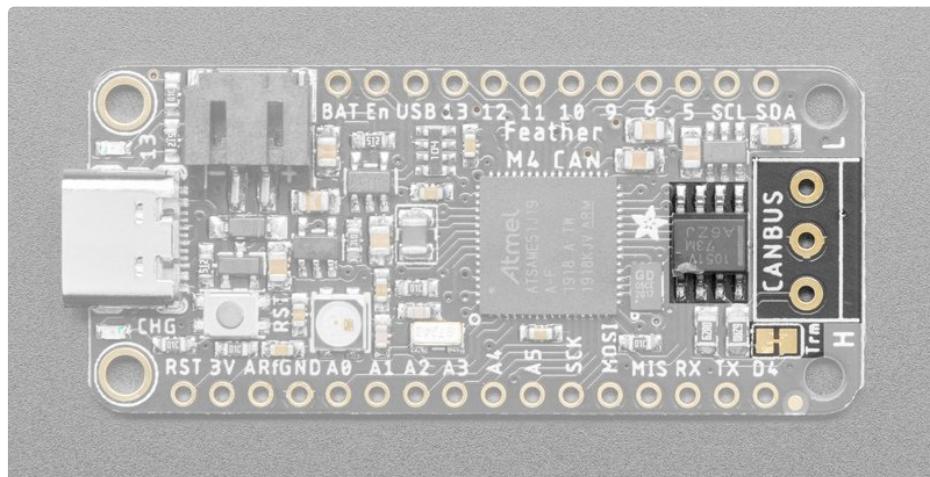
If you've used the Feather M4 (non-CAN version) - please note that the pinouts for the CAN version use the same exact GPIO's for all exposed pins. This makes the CAN Feather M4 work with all of the GPIO-specific accessories like RGB matrix drivers, NeoPXL8 wings, etc.

Power



- **USB C port** - This is used for both powering and programming the board. You can power it with any USB C cable and will request 5V from a USB C PD. When USB is plugged in it will charge the Lipoly battery.
- **GND** - this is the common ground for all power and logic
- **BAT** - this is the positive voltage to/from the JST jack for the optional Lipoly battery
- **USB** - this is the positive voltage to/from the USB C jack if connected
- **EN** - this is the 3.3V regulator's enable pin. It's pulled up, so connect to ground to disable the 3.3V regulator
- **3V** - this is the output from the 3.3V regulator, it can supply 500mA peak

CAN Bus



The SAMDE51 has built in hardware CAN bus support, but you still need a transceiver.

We use a 3V-logic compatible transceiver (the 8-SOIC chip).

Right above it, there is a small 3V->5V switched-capacitor boost converter so that the output CAN signals

are +5V.

By default both the boost converter and the transceiver are disabled. The boost-enable pin is connected to pin # 4 (**BOOST_ENABLE** in CircuitPython), and the transceiver enable (standby) is on a pin named **PIN_CAN_STANDBY**

Here is Arduino code to turn on both the transceiver and booster:

```
pinMode(PIN_CAN_STANDBY, OUTPUT);
digitalWrite(PIN_CAN_STANDBY, false); // turn off STANDBY
pinMode(4, OUTPUT);
digitalWrite(4, true); // turn on booster
```

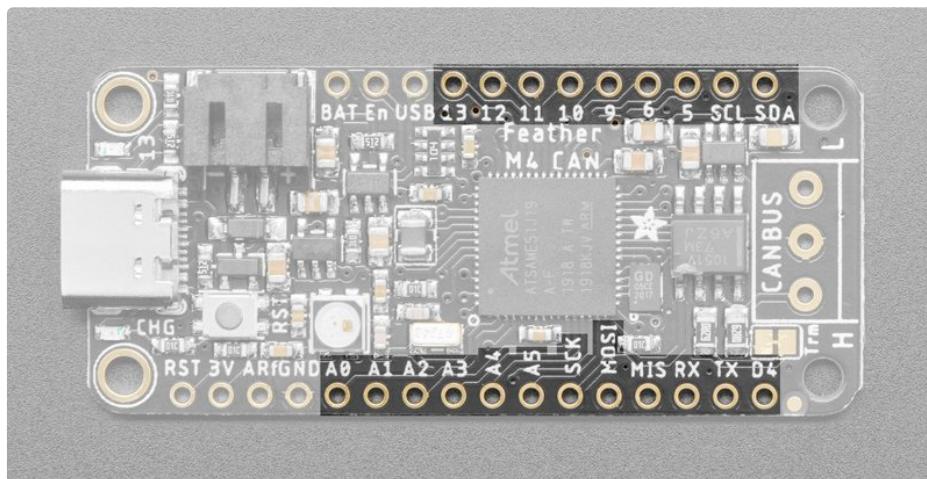
In CircuitPython you can use this snippet to enable

```
# If the CAN transceiver has a standby pin, bring it out of standby mode
if hasattr(board, 'CAN_STANDBY'):
    standby = digitalio.DigitalInOut(board.CAN_STANDBY)
    standby.switch_to_output(False)

# If the CAN transceiver is powered by a boost converter, turn on its supply
if hasattr(board, 'BOOST_ENABLE'):
    boost_enable = digitalio.DigitalInOut(board.BOOST_ENABLE)
    boost_enable.switch_to_output(True)
```

The board has a 120 ohm terminator connected between H and L. You can disable (remove) the terminator by **cutting** the Trm jumper, if your bus is already terminated!

Logic Pins



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

All logic is 3.3V

Nearly all pins can do PWM output

All pins can be interrupt inputs

- **#0 / RX** - GPIO #0, also receive (input) pin for **Serial1** (hardware UART). Also PWM out
- **#1 / TX** - GPIO #1, also transmit (output) pin for **Serial1**. PWM out
- **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. Also GPIO **#21**. Also PWM out
- **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. Also GPIO **#22**. Also PWM out
- **#4** - GPIO #4, PWM out
- **#5** - GPIO #5, PWM out
- **#6** - GPIO #6, PWM out
- **#9** - GPIO #9, PWM out
- **#10** - GPIO #10, PWM out
- **#11** - GPIO #11, PWM out
- **#12** - GPIO #12, PWM out
- **#13** - GPIO #13, PWM out and is connected to the **red LED** next to the USB jack
- **SCK/MOSI/MISO** - These are the hardware SPI pins, you can use them as everyday GPIO
#**25/#24/#23** pins (but recommend keeping them free as they are best used for hardware SPI connections for high speed.)

Analog Pins:

- **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** - This pin is analog *input A1* but is also an analog *output* due to having a DAC (digital-to-analog converter). This is the second DAC, and is 'independent' of A0. You can set the raw voltage to anything from 0 to 3.3V, unlike PWM outputs this is a true analog output. Also can do PWM.
- **A2 thru A5** - These are each analog input as well as digital I/O pins. These pins can also do PWM.

I2S Pins:

- **#1/Tx** - I2S **bit_clock** pin.
- **#10** - I2S **word_select** pin.
- **#11** - I2S **data** pin.

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.

Parallel Capture Peripheral

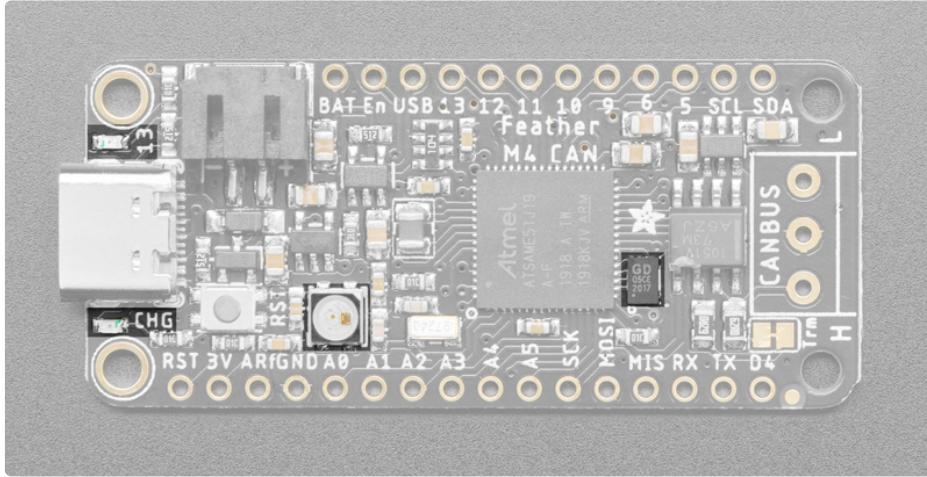
There's a 'camera' input peripheral you can use with some camera chips to capture video with 8-bit data width. We thought this was neat so we made sure all those pins were available. Here are the PCC pins

(left) and the Feather M4 pins it's mapped to. Unlike other peripherals, you cannot mux these signals to other pins!

- DEN1: SDA
- DEN2: SCL
- CLK: D6
- D0: D11
- D1: D13
- D2: D10
- D3: D12
- D4: MISO
- D5: D5
- D6: MOSI
- D7: SCK

NeoPixel and LEDs

As part of the 'Express' series of boards, the Feather M4 CAN Express is designed for use with CircuitPython. To make that easy, we have added two extra parts to this Feather M4: a mini NeoPixel (addressable RGB LED) and a 2 MB QSPI (Quad SPI) Flash chip. There is also the little red LED, and a yellow charge LED.



NeoPixel LED - connected to pin #8 in Arduino, so [just use our NeoPixel library](https://adafru.it/dhw) (<https://adafru.it/dhw>) and set it up as a single-LED strand on pin 8. In CircuitPython, the NeoPixel is `board.NEOPIXEL` and the library for it is [here](https://adafru.it/wby) (<https://adafru.it/wby>) and in [the bundle](https://adafru.it/uap) (<https://adafru.it/uap>).

Unlike most other boards the NeoPixel **is not powered by the 3.3V power supply** - instead we power it from a separate GPIO pin. This lets us turn off the NeoPixel completely, removing the ~1mA quiescent current. To enable the NeoPixel, set the `NEOPIXEL_POWER` (CircuitPython) or `PIN_NEOPixel_POWER` (Arduino) to be an OUTPUT and HIGH. To disable it, set it to be an OUTPUT and LOW. (When used as a

status LED, CircuitPython will automatically enable the power)

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.

- **Red LED** - connected to pin #13. It is user-controllable through code.
- **CHG LED** - This LED is lit up when a battery is charging. If there is no battery attached, the yellow LED will flicker (it's looking for a battery!)

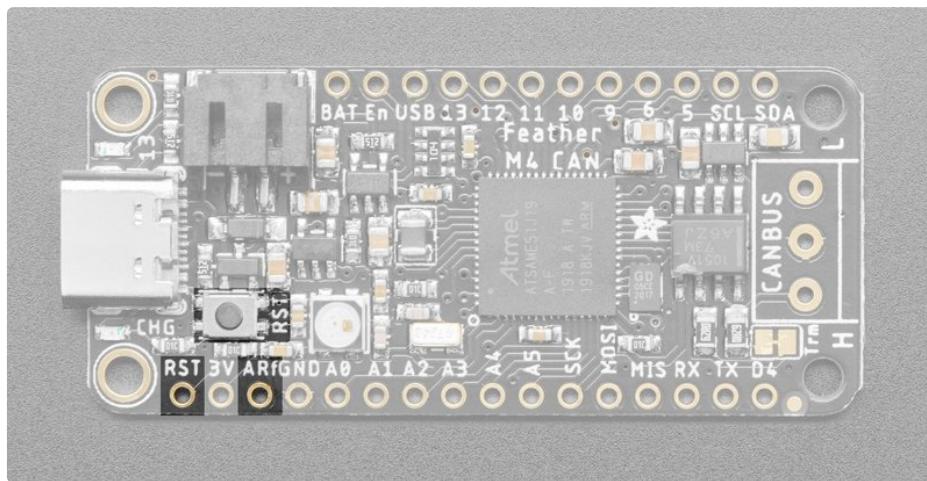
QSPI Flash

- **QSPI Flash** - connected to 6 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.

QSPI is neat because it allows you to have 4 data in/out lines instead of just SPI's single line in and single line out. This means that QSPI is *at least* 4 times faster. But in reality is at least 10x faster because you can clock the QSPI peripheral much faster than a plain SPI peripheral

However, the QSPI port is *not* also on an SERCOM. So, you have to either use the QSPI peripheral or bitbang SPI if you want to talk to the chip. [We have an Arduino library here which provides QSPI interfacing for Arduino \(https://adafru.it/BeX\)](https://adafru.it/BeX). In CircuitPython, the QSPI 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 ATSAMD51, 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!

On the Feather M4 CAN Express, at least for now, AREF is tied to 3.3V due to a silicon v0 bug that does not allow the DACs to work unless AREF is connected to 3.3V. You cut the bottom jumper if you need a different AREF voltage but note that this may change DAC range!

Debugging Interface

If you'd like to do more advanced development, trace-debugging, or not use the bootloader, we have the SWD interface exposed. You'll have to solder to the two SWDIO/SWCLK pads on the bottom:



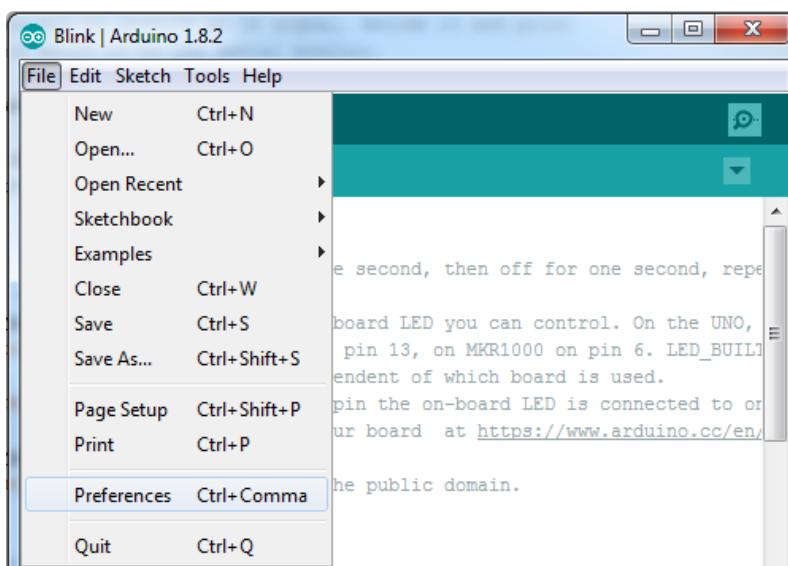
Arduino IDE Setup

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

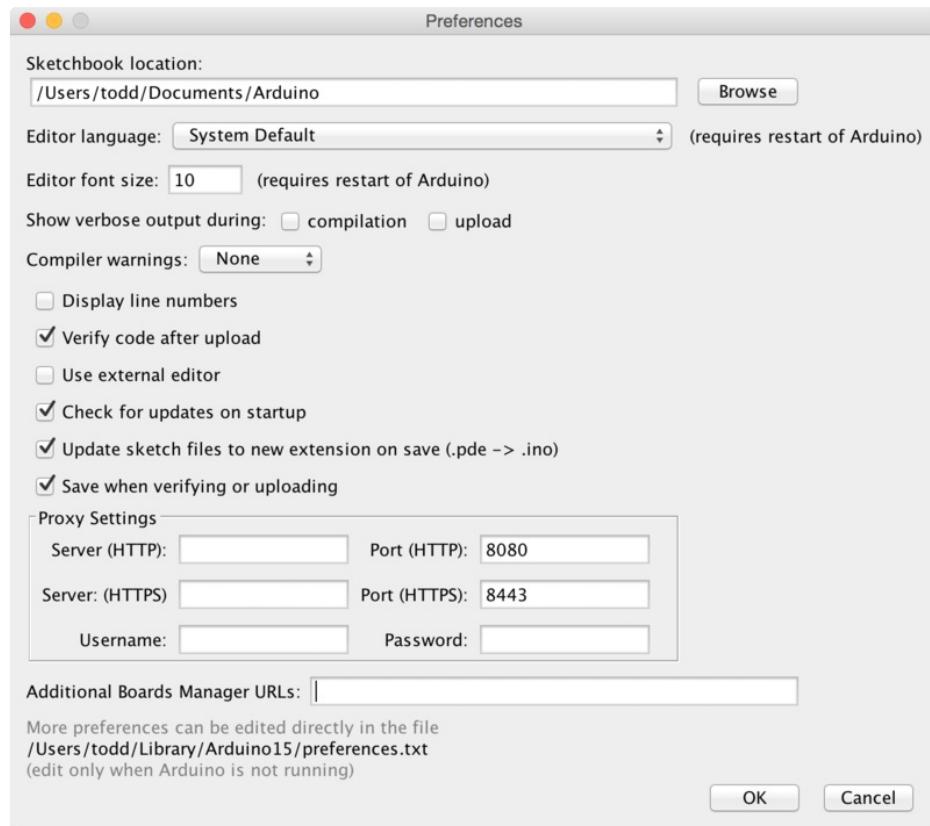
<https://adafru.it/f1P>

<https://adafru.it/f1P>

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



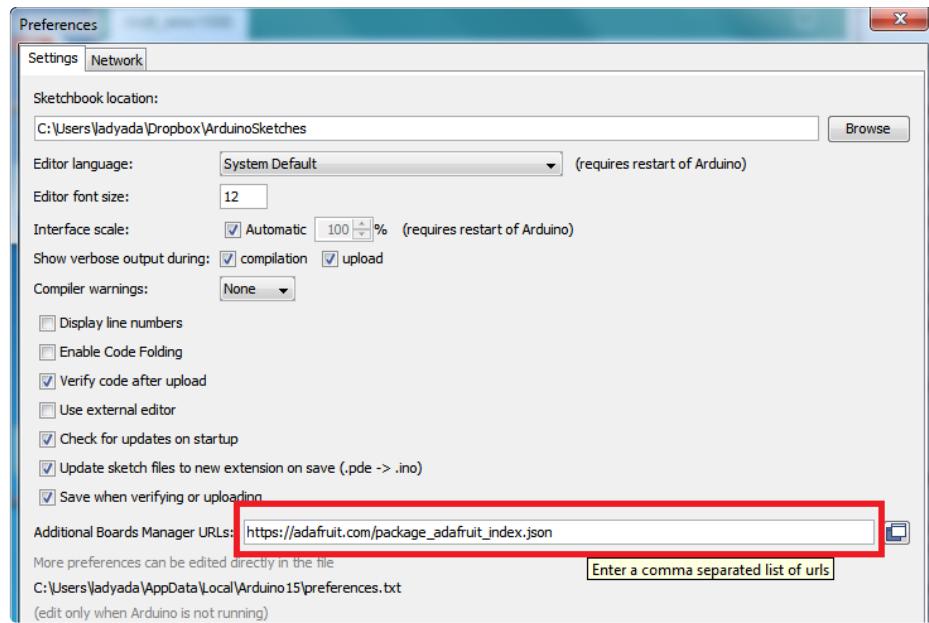
A dialog will pop up just like the one shown below.



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

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

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



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

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

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

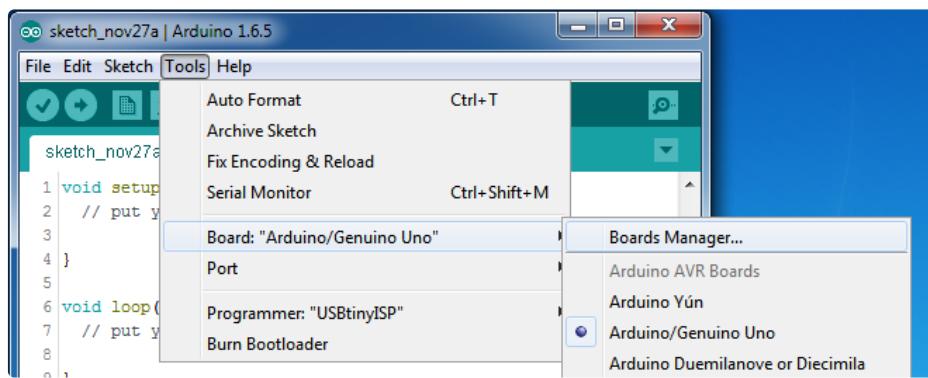
Once done click **OK** to save the new preference settings. Next we will look at installing boards with the Board Manager.

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

Using with Arduino IDE

The Feather/Metro/Gemma/QTPy/Trinket M0 and M4 use an ATSAMD21 or ATSAMD51 chip, and you can pretty easily get it working with the Arduino IDE. Most libraries (including the popular ones like NeoPixels and display) will work with the M0 and M4, especially devices & sensors that use I2C or SPI.

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



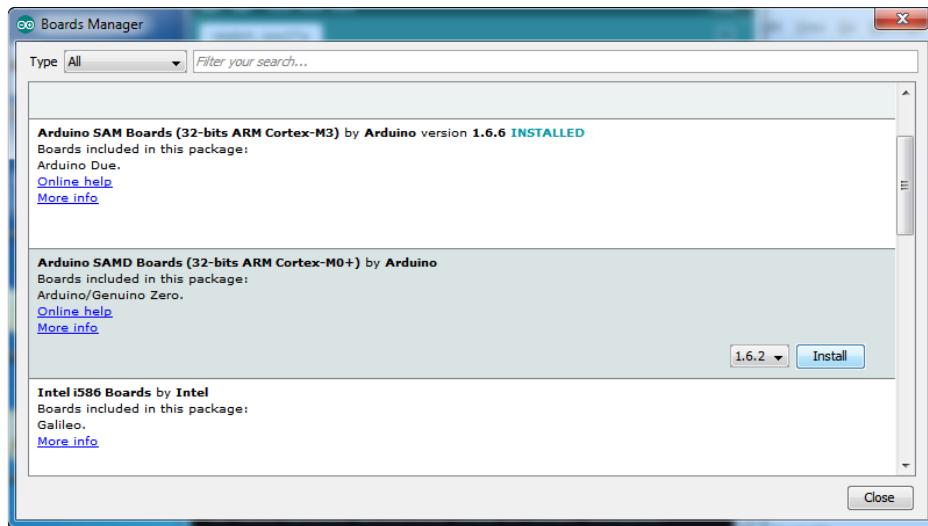
Once the Board Manager opens, click on the category drop down menu on the top left hand side of the window and select **All**. You will then be able to select and install the boards supplied by the URLs added to the preferences.

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

Install SAMD Support

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

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

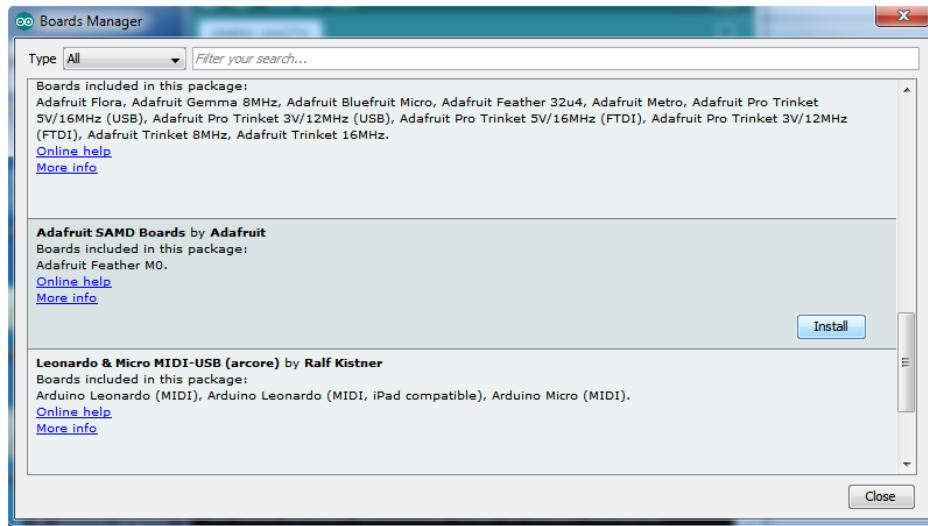


Install Adafruit SAMD

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

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

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



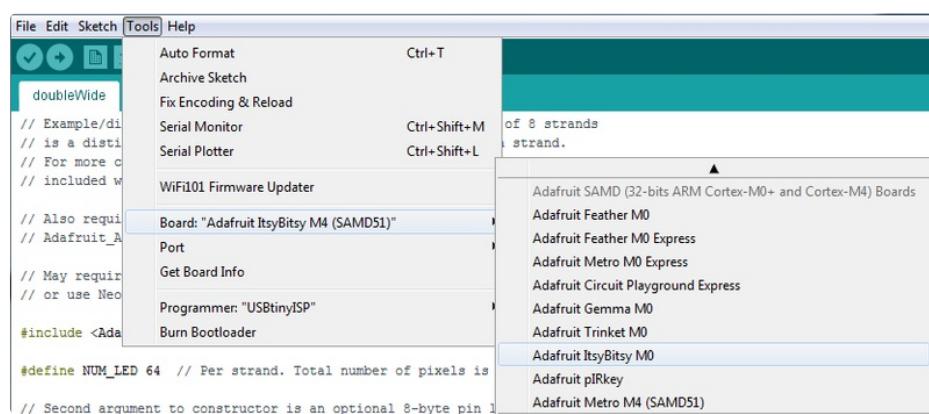
Even though in theory you don't need to - I recommend rebooting the IDE

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

Select the matching board, the current options are:

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

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



Install Drivers (Windows 7 & 8 Only)

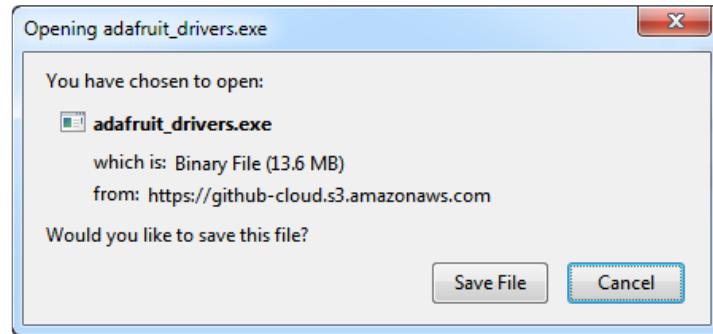
When you plug in the board, you'll need to possibly install a driver

Click below to download our Driver Installer

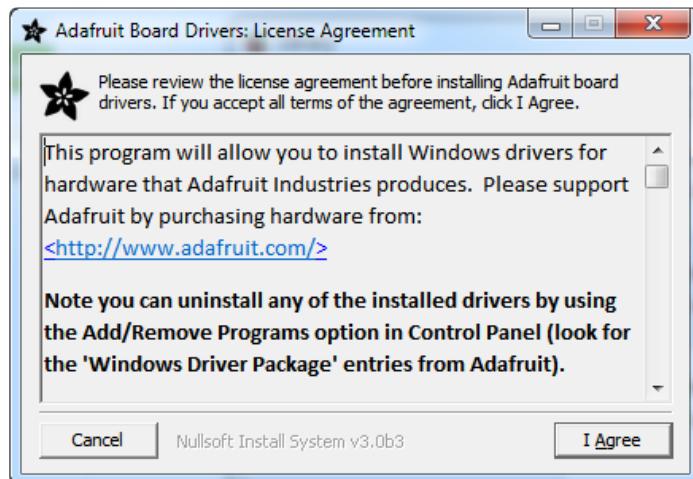
<https://adafru.it/EC0>

<https://adafru.it/EC0>

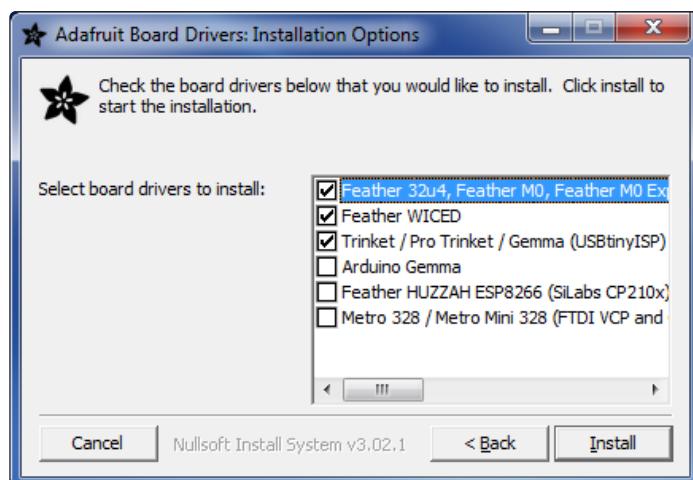
Download and run the installer



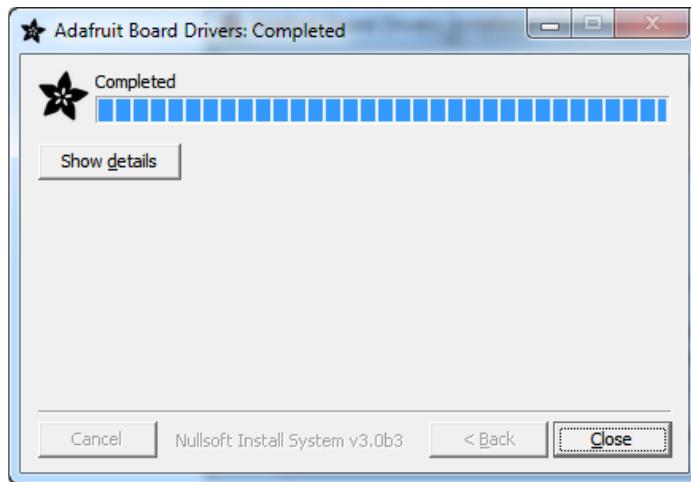
Run the installer! Since we bundle the SiLabs and FTDI drivers as well, you'll need to click through the license



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



Click **Install** to do the installin'

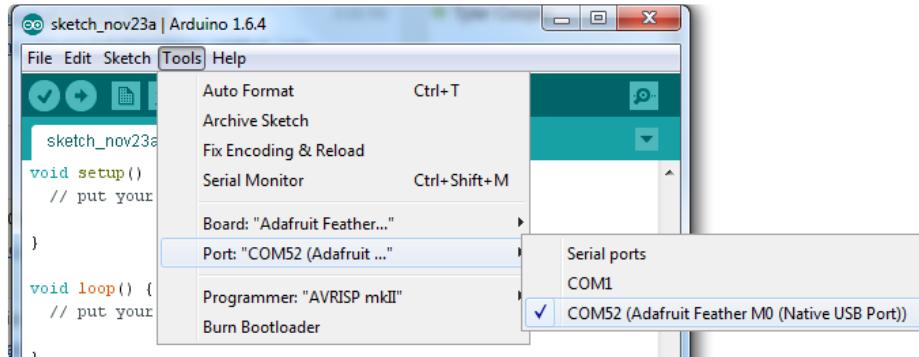


Blink

Now you can upload your first blink sketch!

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

Please note, the QT Py and Trellis M4 Express are two of our very few boards that does not have an onboard pin 13 LED so you can follow this section to practice uploading but you wont see an LED blink!



Now load up the Blink example

```

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

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

```

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

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

Successful Upload

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

```

Done uploading.

Write 11024 bytes to flash (173 pages)

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

done in 0.097 seconds

Verify 11024 bytes of flash with checksum.

Verify successful

done in 0.049 seconds

CPU reset.

Adafruit Feather M0 (Native USB Port) on COM54

```

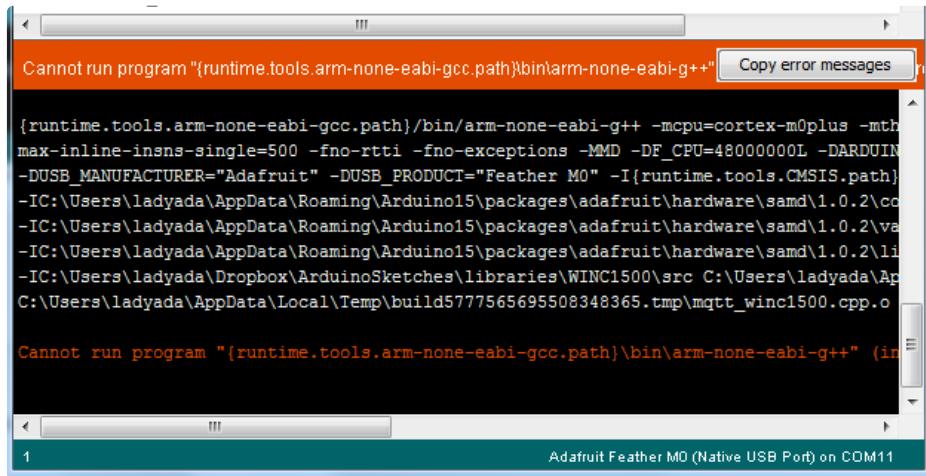
After uploading, you may see a message saying "Disk Not Ejected Properly" about the ...BOOT drive. You can ignore that message: it's an artifact of how the bootloader and uploading work.

Compilation Issues

If you get an alert that looks like

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

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

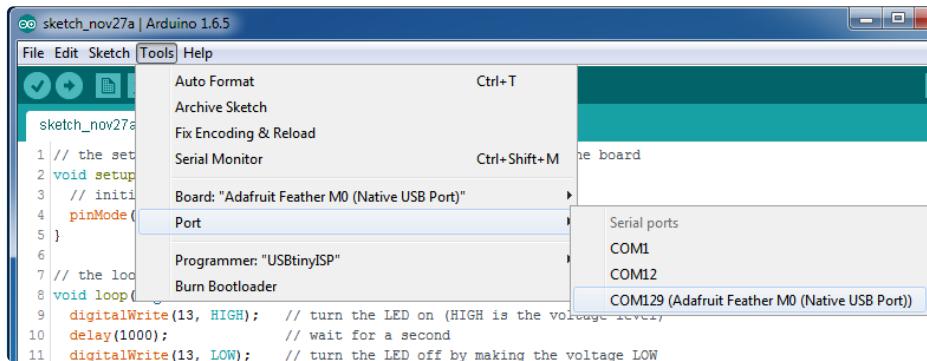


Manually bootloading

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

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

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



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

Ubuntu & Linux Issue Fix

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

Adapting Sketches to M0 & M4

The ATSAMD21 and 51 are very nice little chips, but fairly new as Arduino-compatible cores go. Most sketches & libraries will work but here's a collection of things we noticed.

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

Analog References

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

Pin Outputs & Pullups

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

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

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

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

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

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

Serial vs SerialUSB

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

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

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

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

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

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



The screenshot shows the Arduino IDE interface with a sketch titled "datecalc". The code in the editor is as follows:

```
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(' ');
```

AnalogWrite / PWM on Feather/Metro M0

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

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

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

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

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

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

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

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

TCC instances with eight output channels:

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

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

- Analog pin A5

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

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

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

- TX and SDA (Digital pins 1 and 20)

analogWrite() PWM range

On AVR, if you set a pin's PWM with `analogWrite(pin, 255)` it will turn the pin fully HIGH. On the ARM cortex, it will set it to be 255/256 so there will be very slim but still-existing pulses-to-0V. If you need the pin to be fully on, add test code that checks if you are trying to `analogWrite(pin, 255)` and, instead, does a `digitalWrite(pin, HIGH)`

analogWrite() DAC on A0

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

Missing header files

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

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

you'll get an error that says

```
fatal error: util/delay.h: No such file or directory
#include <util/delay.h>
```

^

compilation terminated.

Error compiling.

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

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

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

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

Bootloader Launching

For most other AVR's, clicking **reset** while plugged into USB will launch the bootloader manually, the bootloader will time out after a few seconds. For the M0/M4, you'll need to **double click** the button. You will see a pulsing red LED to let you know you're in bootloader mode. Once in that mode, it wont time out! Click reset again if you want to go back to launching code.

Aligned Memory Access

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

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

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

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

Floating Point Conversion

Like the AVR Arduinos, the M0 library does not have full support for converting floating point numbers to ASCII strings. Functions like sprintf will not convert floating point. Fortunately, the standard AVR-LIBC

library includes the dtostrf function which can handle the conversion for you.

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

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

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

How Much RAM Available?

The ATSAMD21G18 has 32K of RAM, but you still might need to track it for some reason. You can do so with this handy function:

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

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

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

Storing data in FLASH

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

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

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

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

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

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

Pretty-Printing out registers

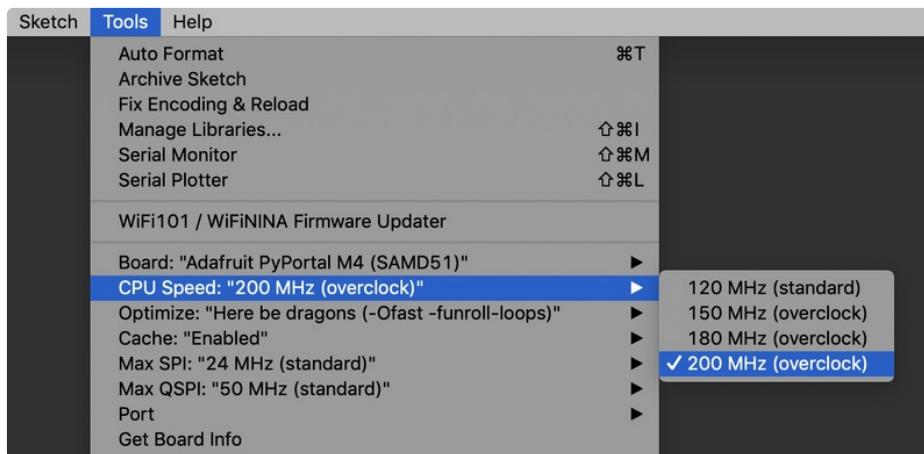
There's a *lot* of registers on the SAMD21, and you often are going through ASF or another framework to

get to them. So having a way to see exactly what's going on is handy. This library from drewfish will help a ton!

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

M4 Performance Options

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



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

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

CPU Speed (overclocking)

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

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

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

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

depending how it's written. **If you encounter such code or libraries, set the CPU speed to the default 120 MHz and re-upload.**

Optimize

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

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

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

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

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

Cache

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

Max SPI and Max QSPI

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

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

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

Enabling the Buck Converter on some M4 Boards

If you want to reduce power draw, some of our boards have an inductor so you can use the 1.8V buck converter instead of the built in linear regulator. If the board does have an inductor (see the schematic) you can add the line `SUPC->VREG.bit SEL = 1;` to your code to switch to it. Note it will make ADC/DAC reads a bit noisier so we don't use it by default. [You'll save ~4mA](https://adafru.it/F0H) (<https://adafru.it/F0H>).

Arduino CAN Examples

To use the built in CAN hardware, we have adapted ('forked') the most common CAN Arduino library and made changes so that it works seamlessly on a SAME51

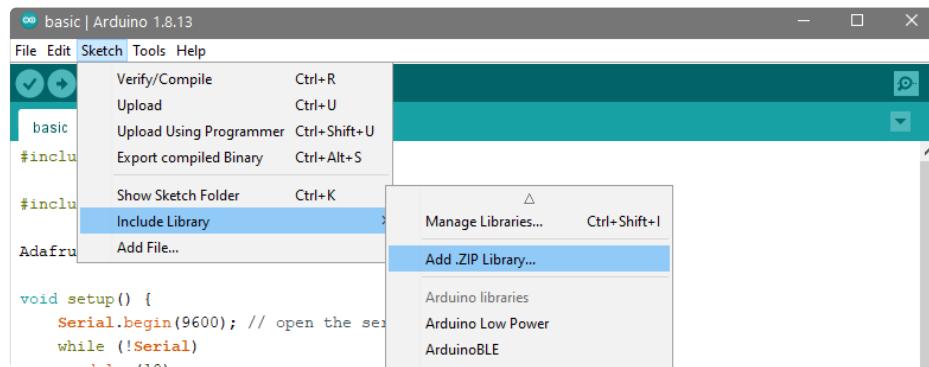
The new library lives on github at <https://github.com/adafruit/arduino-CAN> (<https://adafru.it/QbX>)

Other than having to enable the transceiver and boost supply, you can use existing code for the CAN library 'as is'!

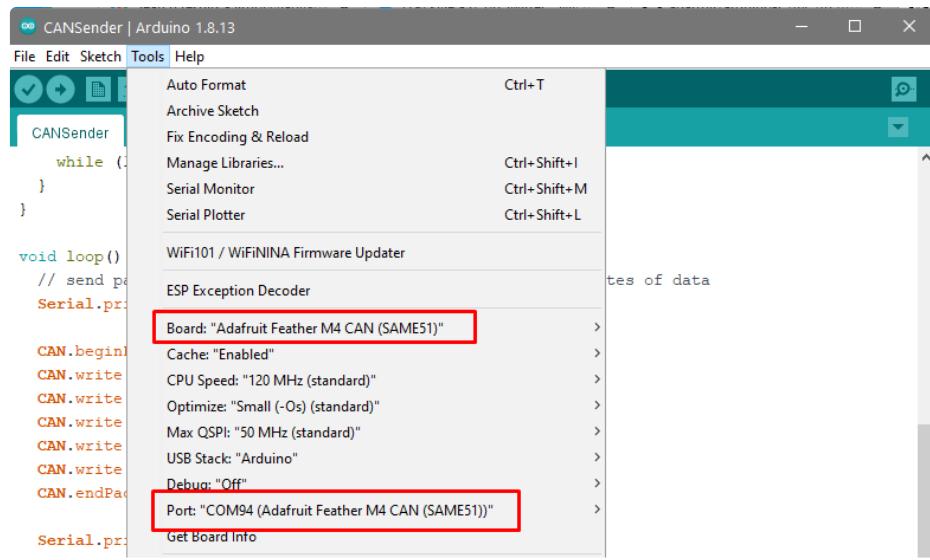
Install the Arduino Library

Right now the new library is not in the library manager. You will have to install it manually To do so, download the zip of the library from <https://github.com/adafruit/arduino-CAN/archive/master.zip> (<https://adafru.it/QbY>)

In the Arduino IDE under the **Sketch** menu, select **Include Library** and **Add .ZIP Library**, then select the zip you just downloaded

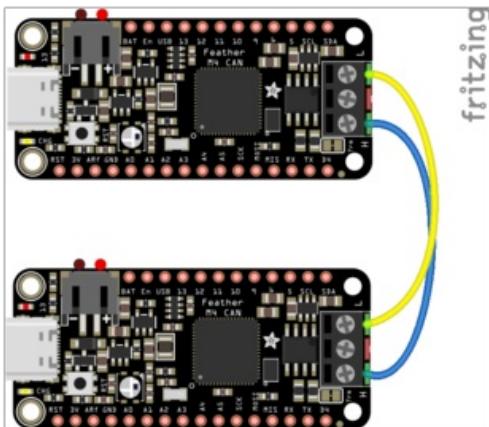


when uploading code make sure you have **Feather M4 CAN (SAME51)** selected, not the Feather M4 Express!



Transmission & Reception Examples

Our first examples will transmit packets from one board to another. In this example we're using two CAN Feather boards but of course you can replace either side with a different CAN-enabled platform.



Wire up L to L and H to H on the bus. You do not need to connect Ground (but it is available)

On one Feather load the transmitter example:

```
// Copyright (c) Sandeep Mistry. All rights reserved.  
// Licensed under the MIT license. See LICENSE file in the project root for full license  
information.  
  
#include <CAN.h>  
  
void setup() {  
    Serial.begin(9600);  
    while (!Serial);  
  
    Serial.println("CAN Sender");  
    pinMode(PIN_CAN_STANDBY, OUTPUT);  
    digitalWrite(PIN_CAN_STANDBY, false); // turn off STANDBY  
    pinMode(PIN_CAN_BOOSTEN, OUTPUT);  
    digitalWrite(PIN_CAN_BOOSTEN, true); // turn on booster  
  
    // start the CAN bus at 250 kbps  
    if (!CAN.begin(250000)) {  
        Serial.println("Starting CAN failed!");  
        while (1);  
    }  
}  
  
void loop() {  
    // send packet: id is 11 bits, packet can contain up to 8 bytes of data  
    Serial.print("Sending packet ... ");  
  
    CAN.beginPacket(0x12);  
    CAN.write('h');  
    CAN.write('e');  
    CAN.write('l');  
    CAN.write('l');  
    CAN.write('o');  
    CAN.endPacket();  
  
    Serial.println("done");  
  
    delay(1000);  
  
    // send extended packet: id is 29 bits, packet can contain up to 8 bytes of data  
    Serial.print("Sending extended packet ... ");  
  
    CAN.beginExtendedPacket(0xabcd);  
    CAN.write('w');  
    CAN.write('o');  
    CAN.write('r');  
    CAN.write('l');  
    CAN.write('d');  
    CAN.endPacket();  
  
    Serial.println("done");  
  
    delay(1000);  
}
```

On the other Feather load the receiver example

```
// Copyright (c) Sandeep Mistry. All rights reserved.
// Licensed under the MIT license. See LICENSE file in the project root for full license
// information.

#include <CAN.h>

void setup() {
    Serial.begin(9600);
    while (!Serial);

    Serial.println("CAN Receiver");

    pinMode(PIN_CAN_STANDBY, OUTPUT);
    digitalWrite(PIN_CAN_STANDBY, false); // turn off STANDBY
    pinMode(PIN_CAN_BOOSTEN, OUTPUT);
    digitalWrite(PIN_CAN_BOOSTEN, true); // turn on booster

    // start the CAN bus at 250 kbps
    if (!CAN.begin(250000)) {
        Serial.println("Starting CAN failed!");
        while (1);
    }
}

void loop() {
    // try to parse packet
    int packetSize = CAN.parsePacket();

    if (packetSize) {
        // received a packet
        Serial.print("Received ");

        if (CAN.packetExtended()) {
            Serial.print("extended ");
        }

        if (CAN.packetRtr()) {
            // Remote transmission request, packet contains no data
            Serial.print("RTR ");
        }

        Serial.print("packet with id 0x");
        Serial.print(CAN.packetId(), HEX);

        if (CAN.packetRtr()) {
            Serial.print(" and requested length ");
            Serial.println(CAN.packetDlc());
        } else {
            Serial.print(" and length ");
            Serial.println(packetSize);

            // only print packet data for non-RTR packets
            while (CAN.available()) {
                Serial.print((char)CAN.read());
            }
        }
    }
}
```

```
        }
        Serial.println();
    }

    Serial.println();
}
}
```

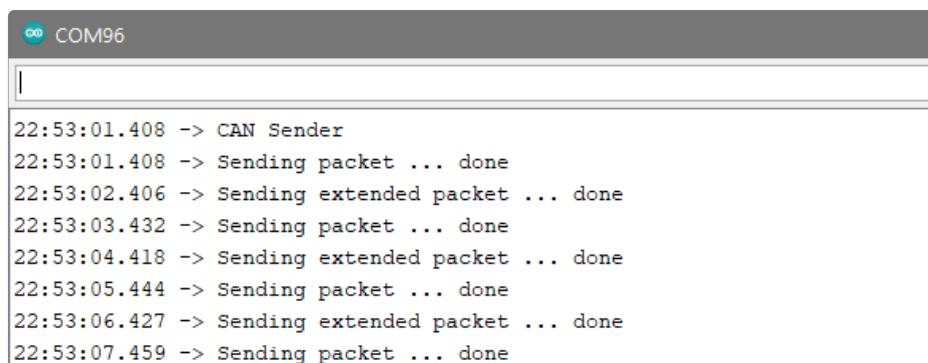
<https://adafru.it/QqC>

<https://adafru.it/QqC>

Make sure whenever you load CAN read/write code you match the bus speed. Some examples are 500000 baud but we recommend 250000 baud! Whatever you use, all boards must have the same

Now open the Serial console for both CAN Feathers (you may need to run Arduino two times, or use another serial port monitor so you can see both)

On the sender you'll see:



```
22:53:01.408 -> CAN Sender
22:53:01.408 -> Sending packet ... done
22:53:02.406 -> Sending extended packet ... done
22:53:03.432 -> Sending packet ... done
22:53:04.418 -> Sending extended packet ... done
22:53:05.444 -> Sending packet ... done
22:53:06.427 -> Sending extended packet ... done
22:53:07.459 -> Sending packet ... done
```

On the receiver you'll see the data getting received!

```
COM94  
22:55:47.040 -> CAN Receiver  
22:55:47.040 -> Received packet with id 0x12 and length 5  
22:55:47.040 -> hello  
22:55:47.040 ->  
22:55:48.023 -> Received packet with id 0x12 and length 5  
22:55:48.023 -> hello  
22:55:48.023 ->  
22:55:49.005 -> Received extended packet with id 0xABCD and length 5  
22:55:49.005 -> world  
22:55:49.005 ->  
22:55:50.029 -> Received packet with id 0x12 and length 5  
22:55:50.029 -> hello  
22:55:50.029 ->  
22:55:51.006 -> Received extended packet with id 0xABCD and length 5  
22:55:51.006 -> world
```

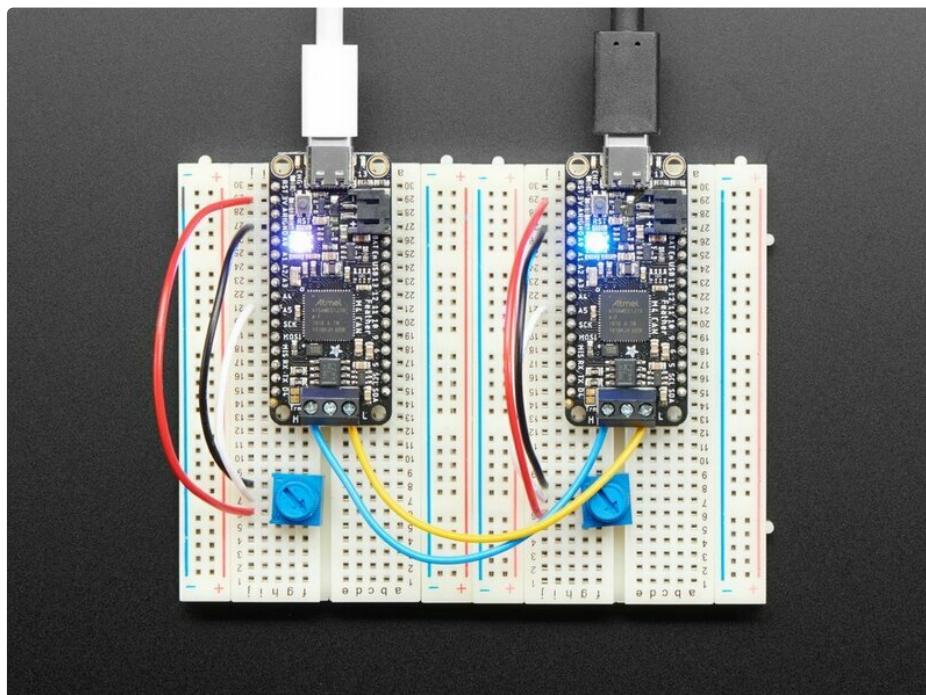
If not, check that **L** and **H** wires are both solidly connected

For more details on the [Arduino CAN library usage, check the documentation](https://adafruit.it/QbZ) (<https://adafruit.it/QbZ>)

Bi-Directional Communication Demo

OK now you can send and receive, but you'd like to do both? No problem, try this example!

On the two boards wired together, connect a potentiometer to each board, with the middle pin going to **A5** and one side to ground and the other side to 3V



Upload the same code to both boards

```

// Copyright (c) ladyada. Public domain

#include <CAN.h>
#include <Adafruit_NeoPixel.h>

Adafruit_NeoPixel strip(1, PIN_NEOPIXEL, NEO_GRB + NEO_KHZ800);

#define MY_PACKET_ID 0xAF

uint32_t timestamp;

void setup() {
  Serial.begin(9600);

  Serial.println("CAN NeoPixel Potentiometer RX/TX demo");

  pinMode(PIN_CAN_STANDBY, OUTPUT);
  digitalWrite(PIN_CAN_STANDBY, false); // turn off STANDBY
  pinMode(PIN_CAN_BOOSTEN, OUTPUT);
  digitalWrite(PIN_CAN_BOOSTEN, true); // turn on booster
  strip.begin();
  strip.setBrightness(50);

  // start the CAN bus at 250 kbps
  if (!CAN.begin(250000)) {
    Serial.println("Starting CAN failed!");
    while (1);
  }

  timestamp = millis();
}

void loop() {
  // every 100 ms send out a packet
  if ((millis() - timestamp) > 100) {
    uint16_t pot = analogRead(A5);
    // send a packet with the potentiometer value
    Serial.print("Sending packet with value ");
    Serial.print(pot);

    CAN.beginPacket(MY_PACKET_ID);
    CAN.write(pot >> 8);
    CAN.write(pot & 0xFF);
    CAN.endPacket();

    Serial.println("...sent!");
    timestamp = millis();
  }

  // try to parse any incoming packet
  int packetSize = CAN.parsePacket();

  if (packetSize) {
    // received a packet
    Serial.print("Received ");

```

```

    Serial.print(" received ");

    if (CAN.packetExtended()) {
        Serial.print("extended ");
    }

    if (CAN.packetRtr()) {
        // Remote transmission request, packet contains no data
        Serial.print("RTR ");
    }

    Serial.print("packet with id 0x");
    Serial.print(CAN.packetId(), HEX);

    if (CAN.packetRtr()) {
        Serial.print(" and requested length ");
        Serial.println(CAN.packetDlc());
    } else {
        Serial.print(" and length ");
        Serial.println(packetSize);

        uint8_t receivedData[packetSize];
        for (int i=0; i<packetSize; i++) {
            receivedData[i] = CAN.read();
            Serial.print("0x");
            Serial.print(receivedData[i], HEX);
            Serial.print(", ");
        }
        Serial.println();
    }

    uint16_t value = (uint16_t)receivedData[0] << 8 | receivedData[1];
    strip.setPixelColor(0, Wheel(value / 4));
    strip.show();
}

Serial.println();
}
}

uint32_t Wheel(byte WheelPos) {
    WheelPos = 255 - WheelPos;
    if(WheelPos < 85) {
        return strip.Color(255 - WheelPos * 3, 0, WheelPos * 3);
    }
    if(WheelPos < 170) {
        WheelPos -= 85;
        return strip.Color(0, WheelPos * 3, 255 - WheelPos * 3);
    }
    WheelPos -= 170;
    return strip.Color(WheelPos * 3, 255 - WheelPos * 3, 0);
}

```

Ten times a second, each board sends out the analog reading from A5 as two bytes of data. If you want to have more than two boards, you can give each board a unique ID by editing `#define MY_PACKET_ID 0xAF`

to change it to a different byte value

```
uint16_t pot = analogRead(A5);
//...
CAN.beginPacket(MY_PACKET_ID);
CAN.write(pot >> 8);
CAN.write(pot & 0xFF);
CAN.endPacket();
```

We also check to read any packets, there are a few different types of CAN packets (not covered in this guide) so we make sure its a non-return-request type...

```
// try to parse any incoming packet
int packetSize = CAN.parsePacket();

if (packetSize) {
    // received a packet
    Serial.print("Received ");
    //...
    Serial.print("packet with id 0x");
    Serial.print(CAN.packetId(), HEX);

    if (CAN.packetRtr()) {
        Serial.print(" and requested length ");
        Serial.println(CAN.packetDlc());
    } else {
        Serial.print(" and length ");
        Serial.println(packetSize);
```

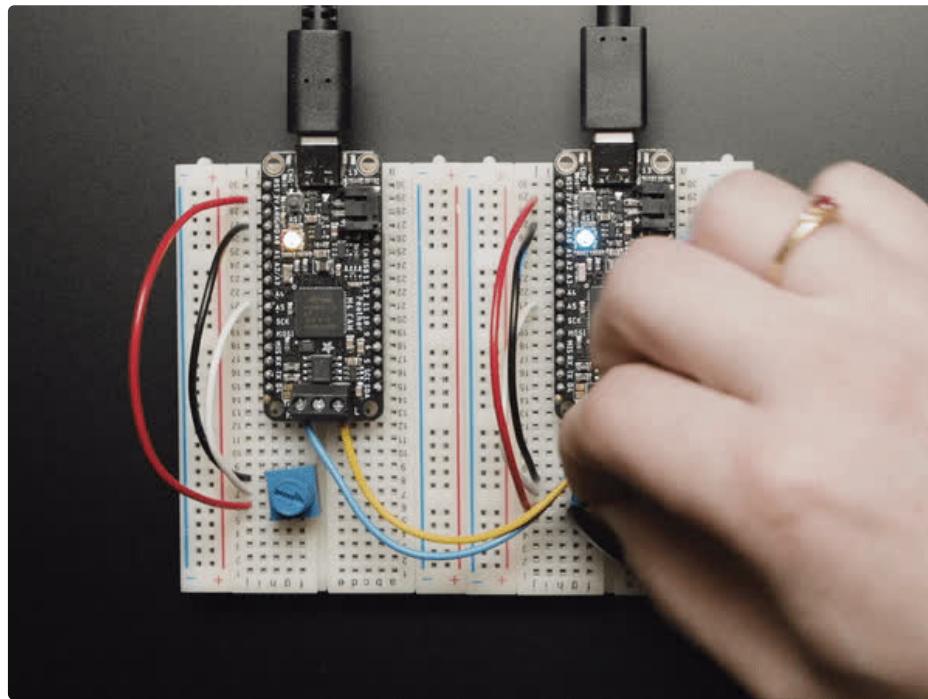
We then read the data from the CAN buffer, convert the first two bytes back into a 16-bit number. That number is going to range from 0 to 1023 (because the analog input signal from the sender is 10-bit). We divide it by 4 so it ranges from 0-255 and then use our color Wheel function to convert into a rainbow color.

Note we do very little checking to verify the packet data and length, this is a very barebones example!

```
uint8_t receivedData[packetSize];
for (int i=0; i<packetSize; i++) {
    receivedData[i] = CAN.read();
    Serial.print("0x");
    Serial.print(receivedData[i], HEX);
    Serial.print(", ");
}
Serial.println();

uint16_t value = (uint16_t)receivedData[0] << 8 | receivedData[1];
strip.setPixelColor(0, Wheel(value / 4));
strip.show();
}
```

Twist the knob on one Feather breadboard to see the NeoPixel on the *other* Feather change colors!



CircuitPython on Feather M4 CAN

[CircuitPython](https://adafru.it/tB7) (<https://adafru.it/tB7>) is a derivative of [MicroPython](https://adafru.it/BeZ) (<https://adafru.it/BeZ>) designed to simplify experimentation and education on low-cost microcontrollers. It makes it easier than ever to get prototyping by requiring no upfront desktop software downloads. Simply copy and edit files on the **CIRCUITPY** drive to iterate.

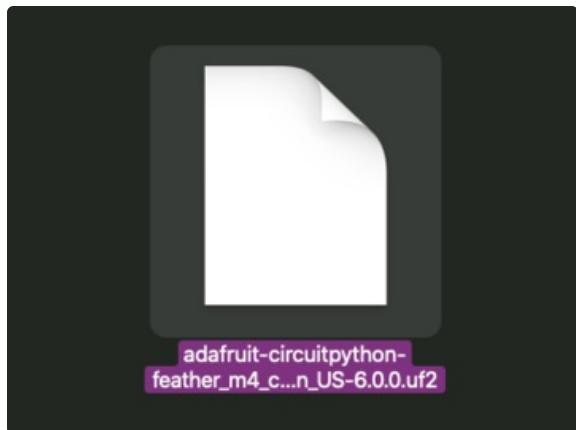
The following instructions will show you how to install CircuitPython. If you've already installed CircuitPython but are looking to update it or reinstall it, the same steps work for that as well!

Set up CircuitPython Quick Start!

Follow this quick step-by-step for super-fast Python power :)

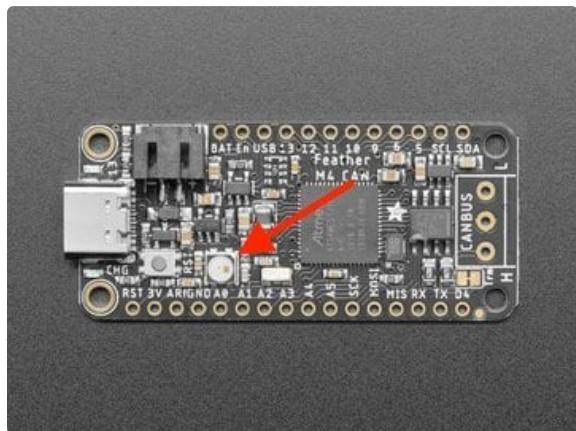
<https://adafru.it/QbB>

<https://adafru.it/QbB>



Click the link above and download the latest UF2 file.

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

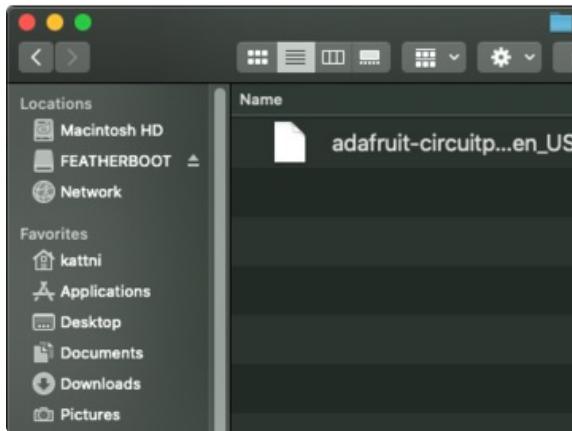


Plug your Feather M4 into your computer using a known-good USB cable.

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

Double-click the **Reset** button next to the USB connector on your board, and you will see the **NeoPixel RGB LED** (as indicated by the red arrow in the image) turn green. If it turns red, check the USB cable, try another USB port, etc. **Note:** The little red LED next to the USB connector will pulse red. That's ok!

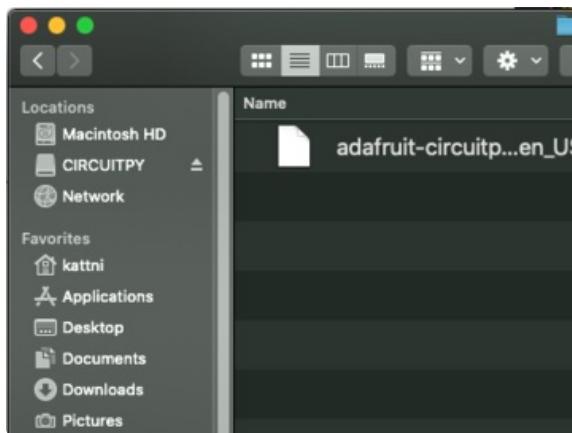
If double-clicking doesn't work the first time, try again. Sometimes it can take a few tries to get the rhythm right!



You will see a new disk drive appear called **FEATHERBOOT**.



Drag the **adafruit_circuitpython_etc.uf2** file to **FEATHERBOOT**.



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

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

Installing Mu Editor

Mu is a simple code editor that works with the Adafruit CircuitPython boards. It's written in Python and works on Windows, MacOS, Linux and Raspberry Pi. The serial console is built right in so you get immediate feedback from your board's serial output!

Mu is our recommended editor - please use it (unless you are an experienced coder with a favorite editor already!)

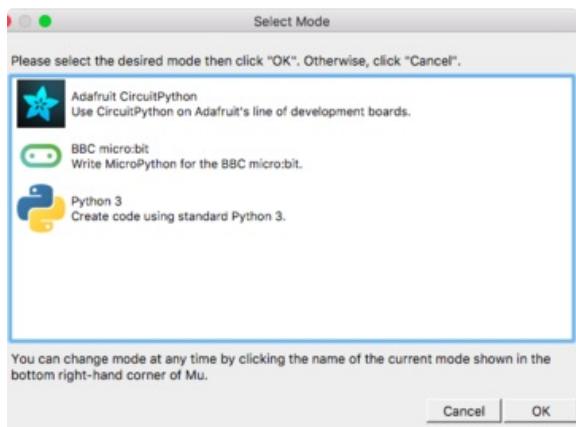
Download and Install Mu



Download Mu

from <https://codewith.mu> (<https://adafru.it/Be6>). Click the **Download** or **Start Here** links there for downloads and installation instructions. The website has a wealth of other information, including extensive tutorials and how-to's.

Using Mu



The first time you start Mu, you will be prompted to select your 'mode' - you can always change your mind later. For now please select **CircuitPython**!

The current mode is displayed in the lower right corner of the window, next to the "gear" icon. If the mode says "Microbit" or something else, click the **Mode** button in the upper left, and then choose "CircuitPython" in the dialog box that appears.



Mu attempts to auto-detect your board, so please plug in your CircuitPython device and make sure it shows up as a **CIRCUITPY** drive before starting Mu

You can now explore Mu! The three main sections of the window are labeled below; the button bar, the text editor, and the serial console / REPL.



Now you're ready to code! Let's keep going...

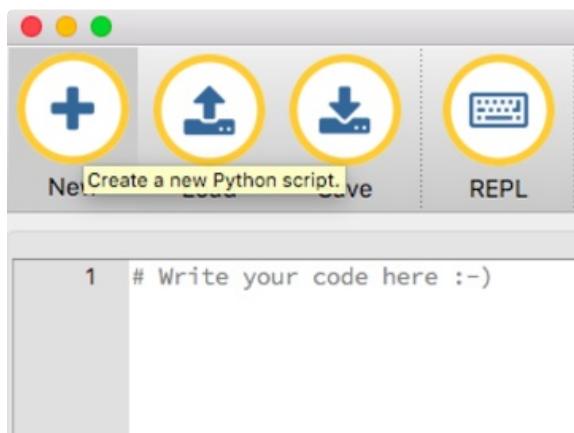
Creating and Editing Code

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

To create and edit code, all you'll need is an editor. There are many options. **We strongly recommend using Mu! It's designed for CircuitPython, and it's really simple and easy to use, with a built in serial console!**

If you don't or can't use Mu, there are basic text editors built into every operating system such as Notepad on Windows,TextEdit on Mac, and gedit on Linux. However, many of these editors don't write back changes immediately to files that you edit. That can cause problems when using CircuitPython. See the [Editing Code \(<https://adafru.it/id3>\)](https://adafru.it/id3) section below. If you want to skip that section for now, make sure you do "Eject" or "Safe Remove" on Windows or "sync" on Linux after writing a file if you aren't using Mu. (This is not a problem on MacOS.)

Creating Code



Open your editor, and create a new file. If you are using Mu, click the **New** button in the top left

Copy and paste the following code into your editor:

```
import board
import digitalio
import time

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

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

The QT Py and the Trinkeys not have a built-in little red LED! There is an addressable RGB NeoPixel LED. The above example will NOT work on the QT Py or the Trinkeys!

If you're using QT Py or a Trinkey, please download the [NeoPixel blink example](https://adafru.it/PE0) (<https://adafru.it/PE0>).

The NeoPixel blink example uses the onboard NeoPixel, but the time code is the same. You can use the linked NeoPixel Blink example to follow along with this guide page.

If you are using Adafruit CLUE, you will need to edit the code to use `board.D17` as shown below!

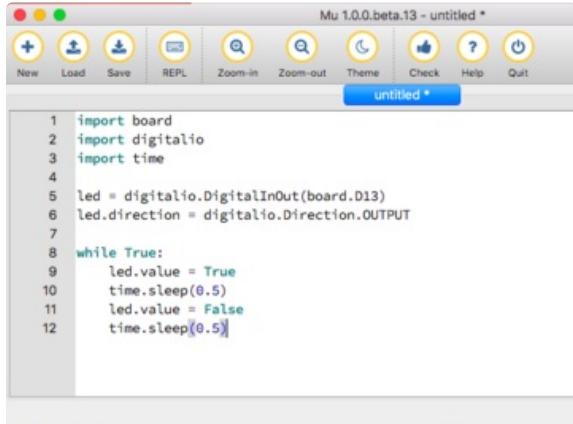
For Adafruit CLUE, you'll need to use `board.D17` instead of `board.LED`. The rest of the code remains the same. Make the following change to the `led =` line:

```
led = digitalio.DigitalInOut(board.D17)
```

If you are using Adafruit ItsyBitsy nRF52840, you will need to edit the code to use `board.BLUE_LED` as shown below!

For Adafruit ItsyBitsy nRF52840, you'll need to use `board.BLUE_LED` instead of `board.LED`. The rest of the code remains the same. Make the following change to the `led =` line:

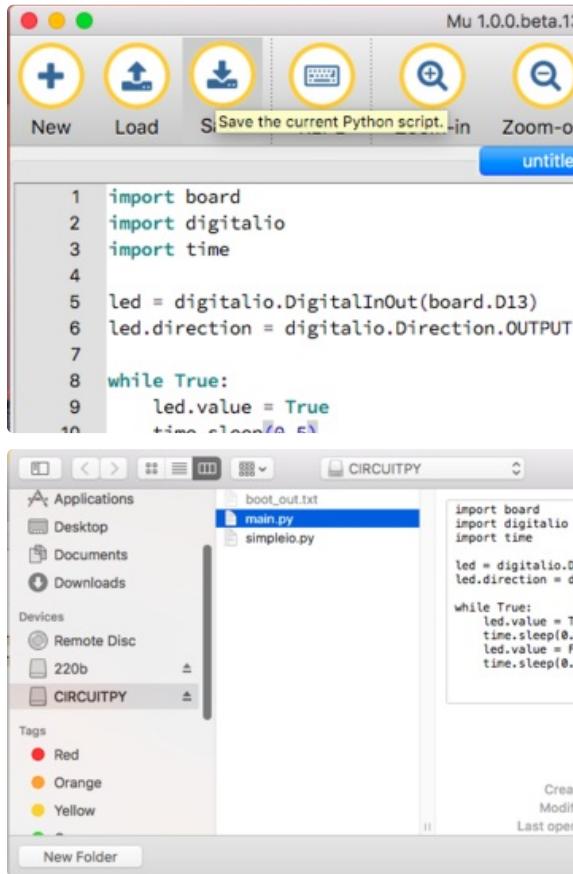
```
led = digitalio.DigitalInOut(board.BLUE_LED)
```



A screenshot of the Mu 1.0.0 beta 13 Python editor. The window title is "Mu 1.0.0.beta.13 - untitled *". The toolbar includes icons for New, Load, Save, REPL, Zoom-in, Zoom-out, Theme, Check, Help, and Quit. The status bar shows "untitled *". The code area contains the following Python script:

```
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
10    time.sleep(0.5)
11    led.value = False
12    time.sleep(0.5)
```

It will look like this - note that under the `while True:` line, the next four lines have spaces to indent them, but they're indented exactly the same amount. All other lines have no spaces before the text.



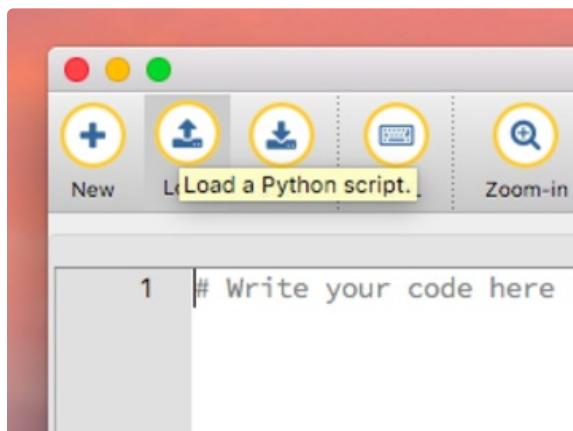
Save this file as **code.py** on your CIRCUITPY drive.

On each board (except the ItsyBitsy nRF52840) you'll find a tiny red LED. On the ItsyBitsy nRF52840, you'll find a tiny blue LED.

The little LED should now be blinking. Once per second.

Congratulations, you've just run your first CircuitPython program!

Editing Code



To edit code, open the **code.py** file on your CIRCUITPY drive into your editor.

Make the desired changes to your code. Save the file. That's it!

Your code changes are run as soon as the file is done saving.

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

Don't Click Reset or Unplug!

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

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

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

There are a few ways to avoid this:

1. Use an editor that writes out the file completely when you save it.

Recommended editors:

- [mu](https://adafruit.it/Be6) (<https://adafruit.it/Be6>) is an editor that safely writes all changes (it's also our recommended editor)
- [emacs](https://adafruit.it/xNA) (<https://adafruit.it/xNA>) is also an editor that will [fully write files on save](https://adafruit.it/Be7) (<https://adafruit.it/Be7>)
- [Sublime Text](https://adafruit.it/xNB) (<https://adafruit.it/xNB>) safely writes all changes
- [Visual Studio Code](https://adafruit.it/Be9) (<https://adafruit.it/Be9>) appears to safely write all changes
- [gedit](#) on Linux appears to safely write all changes
- [IDLE](https://adafruit.it/IWB) (<https://adafruit.it/IWB>), in Python 3.8.1 or later, [was fixed](https://adafruit.it/IWD) (<https://adafruit.it/IWD>) to write all changes immediately
- [thonny](https://adafruit.it/Qb6) (<https://adafruit.it/Qb6>) fully writes files on save

Recommended *only* with particular settings or with add-ons:

- [vim](https://adafruit.it/ek9) (<https://adafruit.it/ek9>) / [vi](#) safely writes all changes. But set up [vim](#) to not write [swapfiles](#) (<https://adafruit.it/ELO>) (.swp files: temporary records of your edits) to CIRCUITPY. Run vim with `vim -n`, set the `no swapfile` option, or set the `directory` option to write swapfiles elsewhere. Otherwise the swapfile writes trigger restarts of your program.
- The [PyCharm IDE](https://adafruit.it/xNC) (<https://adafruit.it/xNC>) is safe if "Safe Write" is turned on in Settings->System Settings->Synchronization (true by default).
- If you are using [Atom](https://adafruit.it/fMG) (<https://adafruit.it/fMG>), install the [fsync-on-save package](#) (<https://adafruit.it/E9m>) so that it will always write out all changes to files on [CIRCUITPY](#).
- [SlickEdit](https://adafruit.it/DdP) (<https://adafruit.it/DdP>) works only if you [add a macro to flush the disk](#) (<https://adafruit.it/ven>).

We *don't* recommend these editors:

- **notepad** (the default Windows editor) and **Notepad++** can be slow to write, so we recommend the editors above! If you are using notepad, be sure to eject the drive (see below)
- **IDLE** in Python 3.8.0 or earlier does not force out changes immediately
- **nano** (on Linux) does not force out changes
- **geany** (on Linux) does not force out changes
- **Anything else** - we haven't tested other editors so please use a recommended one!

If you are dragging a file from your host computer onto the CIRCUITPY drive, you still need to do step 2. Eject or Sync (below) to make sure the file is completely written.

2. Eject or Sync the Drive After Writing

If you are using one of our not-recommended-editors, not all is lost! You can still make it work.

On Windows, you can **Eject or Safe Remove** the CIRCUITPY drive. It won't actually eject, but it will force the operating system to save your file to disk. On Linux, use the **sync** command in a terminal to force the write to disk.

You also need to do this if you use Windows Explorer or a Linux graphical file manager to drag a file onto CIRCUITPY

Oh No I Did Something Wrong and Now The CIRCUITPY Drive Doesn't Show Up!!!

Don't worry! Corrupting the drive isn't the end of the world (or your board!). If this happens, follow the steps found on the [Troubleshooting \(<https://adafruit.it/Den>\)](https://adafruit.it/Den) page of every board guide to get your board up and running again.

Back to Editing Code...

Now! Let's try editing the program you added to your board. Open your `code.py` file into your editor. We'll make a simple change. Change the first `0.5` to `0.1`. The code should look like this:

```
import board
import digitalio
import time

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

while True:
    led.value = True
    time.sleep(0.1)
    led.value = False
    time.sleep(0.5)
```

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

Exploring Your First CircuitPython Program

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

Here is the original code again:

```
import board
import digitalio
import time

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

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

Imports & Libraries

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

```
import board
import digitalio
import time
```

The `import` statements tells the board that you're going to use a particular library in your code. In this example, we imported three modules: `board`, `digitalio`, and `time`. All three of these modules are built into CircuitPython, so no separate library files are needed. That's one of the things that makes this an excellent first example. You don't need any thing extra to make it work! `board` gives you access to the *hardware on your board*, `digitalio` lets you *access that hardware as inputs/outputs* and `time` let's you pass time by 'sleeping'

Setting Up The LED

The next two lines setup the code to use the LED.

```
led = digitalio.DigitalInOut(board.LED)
led.direction = digitalio.Direction.OUTPUT
```

Your board knows the red LED as `LED`. So, we initialise that pin, and we set it to output. We set `led` to equal the rest of that information so we don't have to type it all out again later in our code.

Loop-de-loops

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

Inside our loop, we have four items:

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

First, we have `led.value = True`. This line tells the LED to turn on. On the next line, we have `time.sleep(0.5)`. This line is telling CircuitPython to pause running code for 0.5 seconds. Since this is between turning the led on and off, the led will be on for 0.5 seconds.

The next two lines are similar. `led.value = False` tells the LED to turn off, and `time.sleep(0.5)` tells CircuitPython to pause for another 0.5 seconds. This occurs between turning the led off and back on so the LED will be off for 0.5 seconds too.

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

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

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

What Happens When My Code Finishes Running?

When your code finishes running, CircuitPython resets your microcontroller board to prepare it for the next run of code. That means any set up you did earlier no longer applies, and the pin states are reset.

For example, try reducing the above example to `led.value = True`. The LED will flash almost too quickly to see, and turn off. This is because the code finishes running and resets the pin state, and the LED is no longer receiving a signal.

To that end, most CircuitPython programs involve some kind of loop, infinite or otherwise

What if I don't have the loop?

If you don't have the loop, the code will run to the end and exit. This can lead to some unexpected behavior in simple programs like this since the "exit" also resets the state of the hardware. This is a different behavior than running commands via REPL. So if you are writing a simple program that doesn't seem to work, you may need to add a loop to the end so the program doesn't exit.

The simplest loop would be:

```
while True:
```

```
    pass
```

And remember - you can press to exit the loop.

See also the [Behavior section in the docs](#) (<https://adafru.it/Bvz>).

More Changes

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

```
while True:  
    led.value = True  
    time.sleep(0.1)  
    led.value = False  
    time.sleep(0.1)
```

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

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

Well done! You're doing great! You're ready to start into new examples and edit them to see what happens! These were simple changes, but major changes are done using the same process. Make your desired change, save it, and get the results. That's really all there is to it!

Naming Your Program File

CircuitPython looks for a code file on the board to run. There are four options: `code.txt`, `code.py`, `main.txt` and `main.py`. CircuitPython looks for those files, in that order, and then runs the first one it finds. While we suggest using `code.py` as your code file, it is important to know that the other options exist. If your program doesn't seem to be updating as you work, make sure you haven't created another code file that's being read instead of the one you're working on.

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.

If you're on Linux, and are seeing multi-second delays connecting to the serial console, or are seeing "AT" and other gibberish when you connect, then the modemmanager service might be interfering. Just remove it; it doesn't have much use unless you're still using dial-up modems. To remove, type this command at a shell:

```
sudo apt purge modemmanager
```

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 **Serial** button in the menu and click it.



Setting Permissions on Linux

On Linux, if you see an error box something like the one below when you press the **Serial** button, you need to add yourself to a user group to have permission to connect to the serial console.



On Ubuntu and Debian, add yourself to the **dialout** group by doing:

```
sudo adduser $USER dialout
```

After running the command above, reboot your machine to gain access to the group. On other Linux distributions, the group you need may be different. See [Advanced Serial Console on Mac and](#)

[Linux \(https://adafru.it/AI\)](https://adafru.it/AI) for details on how to add yourself to the right group.

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 \(https://adafru.it/AH\)](https://adafru.it/AH)

[Mac and Linux both have one built in, though other options are available for download, check this page for more details \(https://adafru.it/AI\)](https://adafru.it/AI)

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:

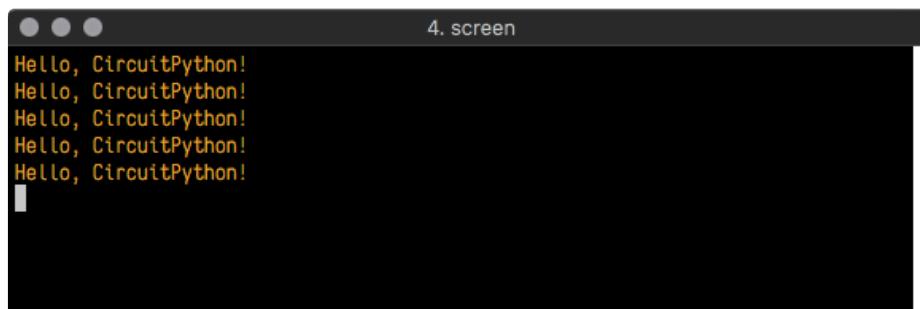
```
import board
import digitalio
import time

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.

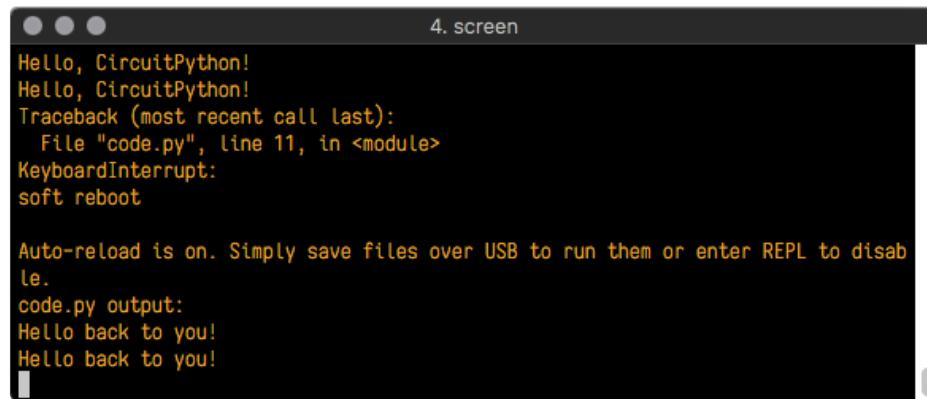
Now, let's go take a look at the window with our connection to the serial console.



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
```

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!
```

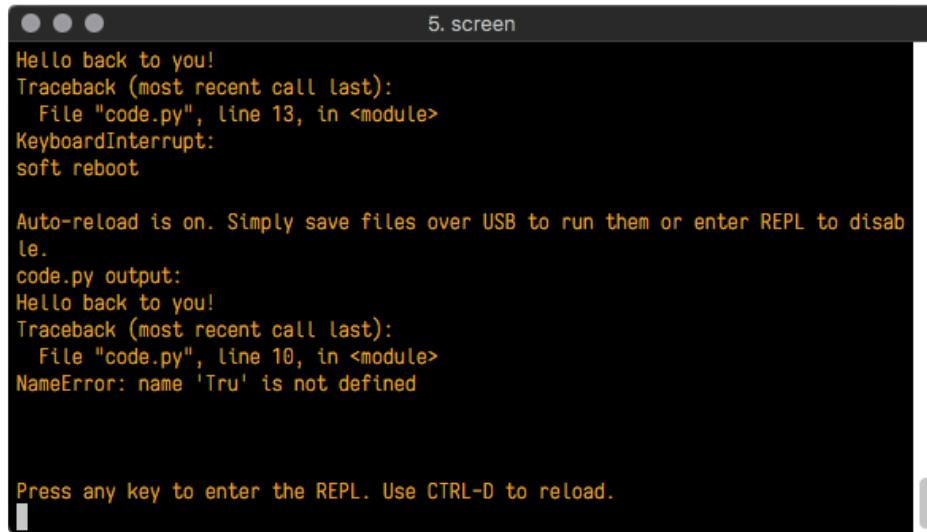
The **Traceback (most recent call last):** is telling you the last thing your board was doing before you saved your file. This is normal behavior and will happen every time the board resets. This is really handy for troubleshooting. Let's introduce an error so we can see how it is used.

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!



A screenshot of a serial console window titled "5. screen". The window shows the following text:

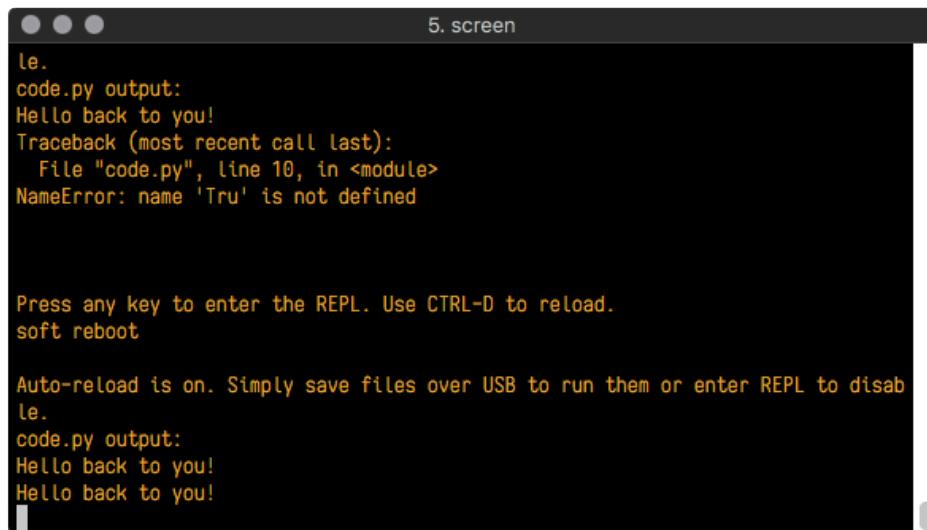
```
Hello back to you!
Traceback (most recent call last):
  File "code.py", line 13, in <module>
KeyboardInterrupt:
soft reboot

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.



A screenshot of a serial console window titled "5. screen". The window shows the following text:

```
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.

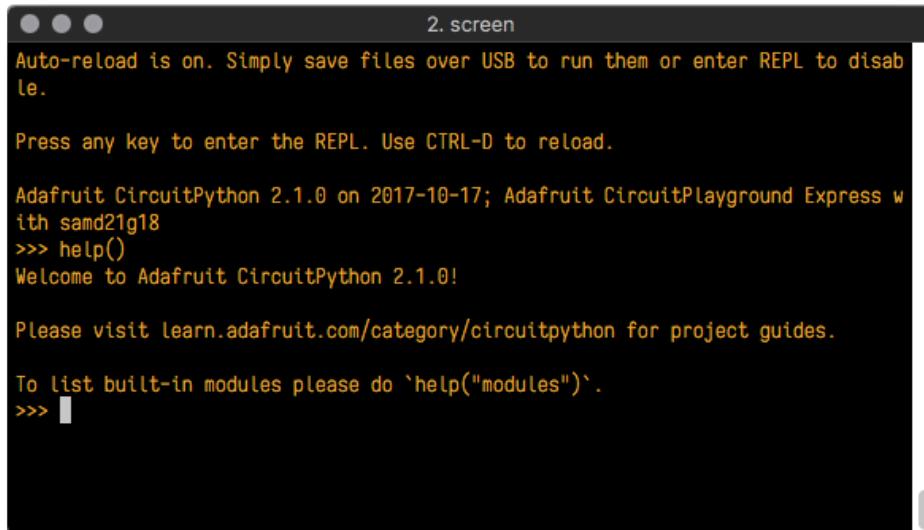
```
>>>
```

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 samd21g18
>>> 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.



The screenshot shows a terminal window titled "3. screen". The output of the command `help("modules")` is displayed. It starts with the Adafruit Feather M0 Express board information, followed by a welcome message, and then a list of built-in modules. The modules listed are: __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. A note at the bottom says "Plus any modules on the filesystem". The REPL prompt "`>>>` " is visible at the bottom.

```
Adafruit CircuitPython 2.1.0 on 2017-10-17; Adafruit Feather M0 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")`.
>>> 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.



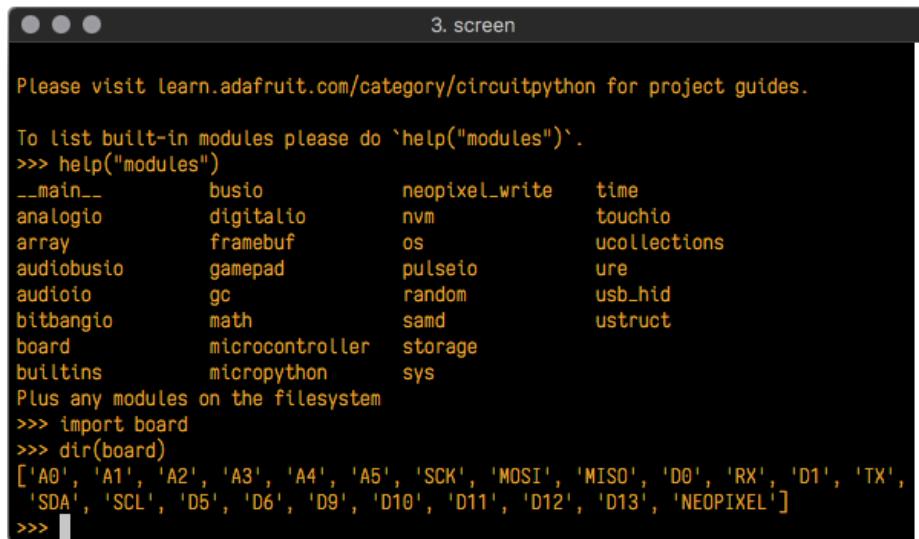
The screenshot shows a terminal window titled "3. screen". The output of the command `import board` is displayed. It starts with the Adafruit Feather M0 Express board information, followed by a welcome message, and then a list of built-in modules. The modules listed are: __main__, busio, neopixel_write, 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. A note at the bottom says "Plus any modules on the filesystem". The REPL prompt "`>>>` " is visible at the bottom.

```
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.

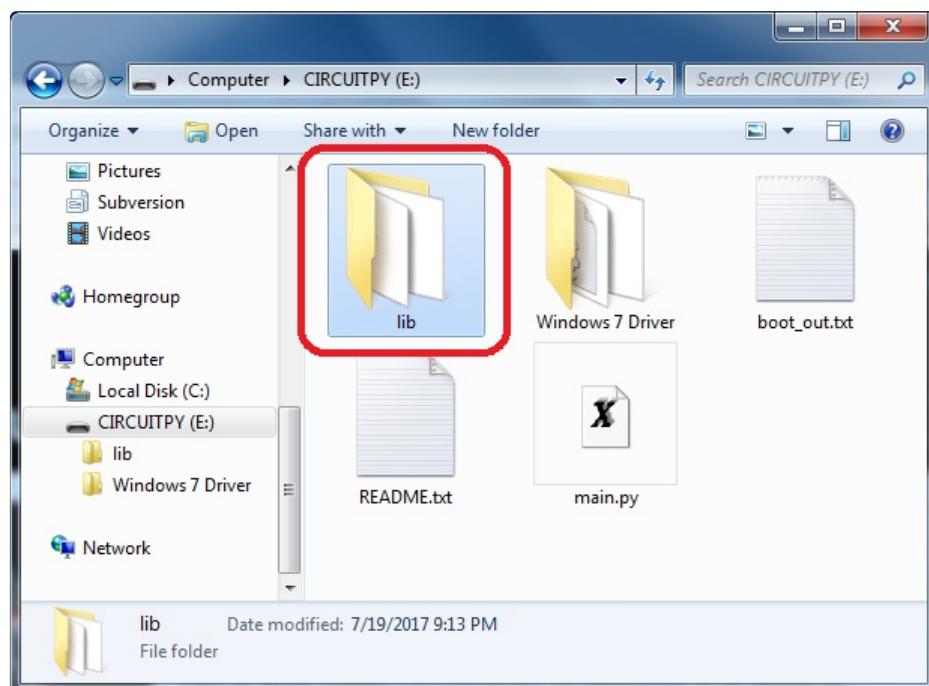
You can return to the REPL at any time!

CircuitPython Libraries

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

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

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



CircuitPython libraries work in the same way as regular Python modules so the [Python docs](https://adafruit.it/rar) (<https://adafruit.it/rar>) are a great reference for how it all should work. In Python terms, we can place our library files in the **lib** directory because its part of the Python path by default.

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

libraries.

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

Installing the CircuitPython Library Bundle

We're constantly updating and improving our libraries, so we don't (at this time) ship our CircuitPython boards with the full library bundle. Instead, you can find example code in the guides for your board that depends on external libraries. Some of these libraries may be available from us at Adafruit, some may be written by community members!

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

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

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

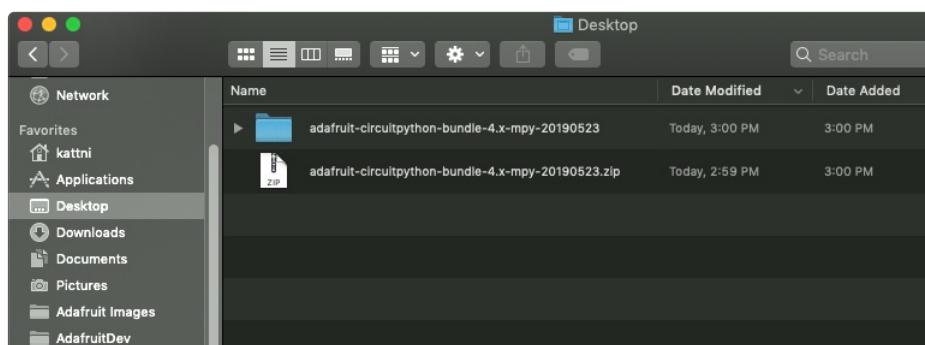
<https://adafru.it/ENC>

<https://adafru.it/ENC>

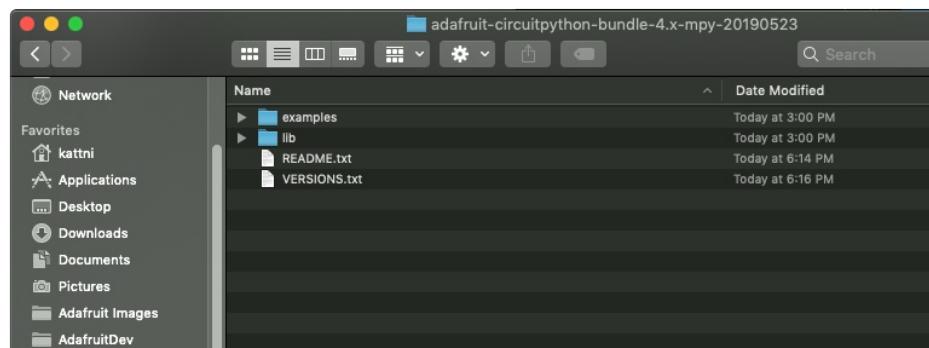
If you need another version, [you can also visit the bundle release page](#) (<https://adafru.it/Ayy>) which will let you select exactly what version you're looking for, as well as information about changes.

Either way, download the version that matches your CircuitPython firmware version. If you don't know the version, look at the initial prompt in the CircuitPython REPL, which reports the version. For example, if you're running v4.0.1, download the 4.x library bundle. There's also a `py` bundle which contains the uncompressed python files, you probably *don't* want that unless you are doing advanced work on libraries.

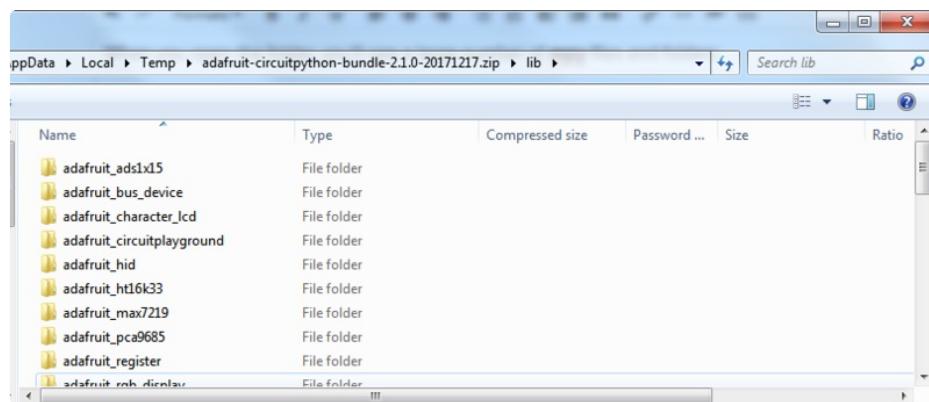
After downloading the zip, extract its contents. This is usually done by double clicking on the zip. On Mac OSX, it places the file in the same directory as the zip.



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



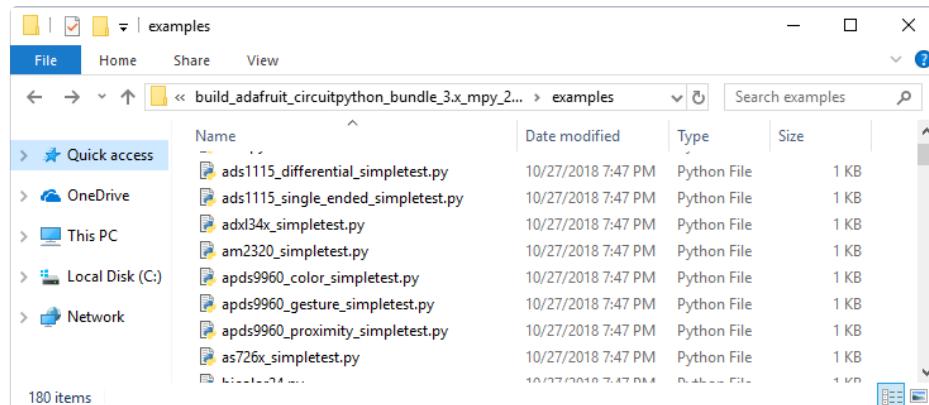
Now open the lib folder. When you open the folder, you'll see a large number of `mpy` files and folders



Example Files

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

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



Copying Libraries to Your Board

First you'll want to create a `lib` folder on your **CIRCUITPY** drive. Open the drive, right click, choose the option to create a new folder, and call it `lib`. Then, open the `lib` folder you extracted from the downloaded zip. Inside you'll find a number of folders and `.mpy` files. Find the library you'd like to use, and copy it to the `lib` folder on **CIRCUITPY**.

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

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

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.

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.

You may end up running out of space on your Trinket M0 or Gemma M0 even if you only load libraries as you need them. There are a number of steps you can use to try to resolve this issue. You'll find them in the Troubleshooting page in the Learn guides for your board.

Updating CircuitPython Libraries/Examples

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

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

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

Frequently Asked Questions

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

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

I have to continue using an older version of CircuitPython; where can I find compatible libraries?

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

- [2.x \(https://adafru.it/FJA\)](https://adafru.it/FJA)
- [3.x \(https://adafru.it/FJB\)](https://adafru.it/FJB)
- [4.x \(https://adafru.it/QDL\)](https://adafru.it/QDL)
- [5.x \(https://adafru.it/QDJ\)](https://adafru.it/QDJ)

Is ESP8266 or ESP32 supported in CircuitPython? Why not?

We dropped ESP8266 support as of 4.x - For more information please read about it here!

[https://learn.adafruit.com/welcome-to-circuitpython/circuitpython-for-esp8266 \(https://adafru.it/CiG\)](https://learn.adafruit.com/welcome-to-circuitpython/circuitpython-for-esp8266 (https://adafru.it/CiG))

We do not support ESP32 because it does not have native USB. We do support ESP32-S2, which does.

How do I connect to the Internet with CircuitPython?

If you'd like to add WiFi support, check out our guide on ESP32/ESP8266 as a co-processor. (<https://adafru.it/Dwa>)

Is there asyncio support in CircuitPython?

We do not have asyncio support in CircuitPython at this time. However, `async` and `await` are turned on in many builds, and we are looking at how to use event loops and other constructs effectively and easily.

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

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

What is a **MemoryError**?

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

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

Try resetting your board. Each time you reset the board, it reallocates the memory. While this is unlikely to

resolve your issue, it's a simple step and is worth trying.

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

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

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

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

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

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

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

You can download `mpy-cross` for your operating system from <https://adafruit-circuit-python.s3.amazonaws.com/index.html?prefix=bin/mpy-cross/> (<https://adafru.it/QDK>). Almost any version will do. The format for `.mpy` files has not changed since CircuitPython 4.x.

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

How do I check how much memory I have free?

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

Will give you the number of bytes available for use.

Does CircuitPython support interrupts?

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

Does Feather M0 support WINC1500?

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

Can AVRs such as ATmega328 or ATmega2560 run CircuitPython?

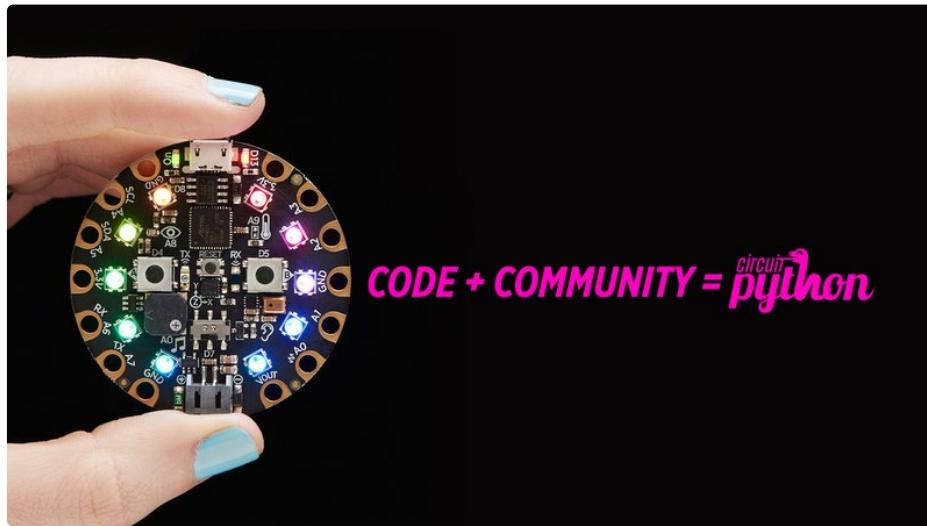
No.

Commonly Used Acronyms

CP or CPy = [CPC = \[Circuit Playground Classic\]\(https://adafruit.it/ncE\)](https://adafruit.it/cpy>Welcome</p></div><div data-bbox=)

CPX = [Circuit Playground Express](https://adafruit.it/wpF)

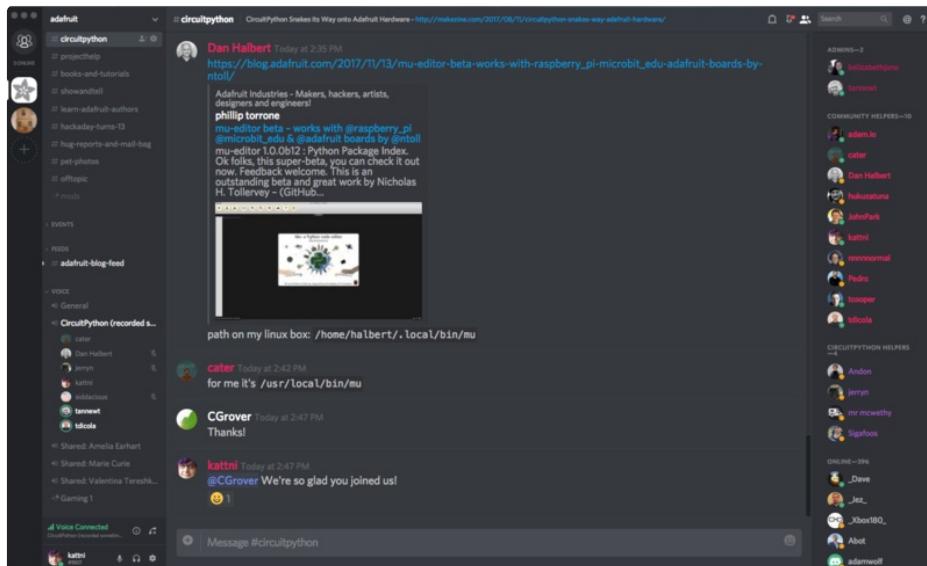
Welcome to the Community!



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 #help-with-projects 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.

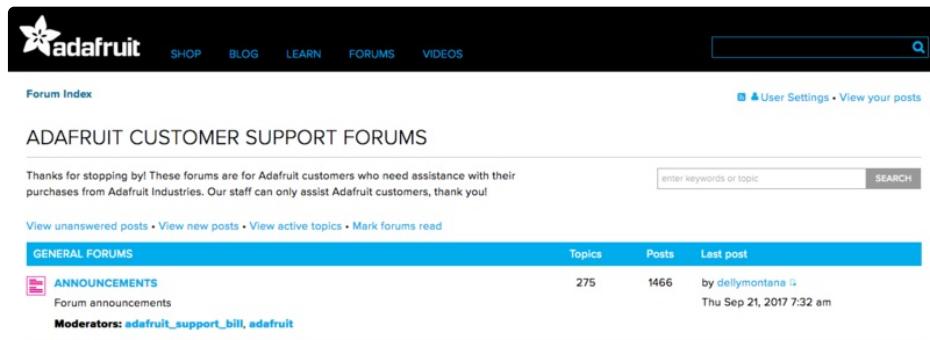
The help with CircuitPython channel is where to go with your CircuitPython questions. #help-with-circuitpython is there for new users and developers alike so feel free to ask a question or post a comment! Everyone of any experience level is welcome to join in on the conversation. We'd love to hear what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

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

A screenshot of the Adafruit Customer Support Forums homepage. At the top, there's a navigation bar with links for SHOP, BLOG, LEARN, FORUMS (which is highlighted in blue), and VIDEOS. To the right of the navigation is a search bar. Below the navigation, there's a link to 'Forum Index' and another to 'User Settings • View your posts'. The main title 'ADAFRUIT CUSTOMER SUPPORT FORUMS' is centered above a horizontal line. Below the line, a message says 'Thanks for stopping by! These forums are for Adafruit customers who need assistance with their purchases from Adafruit Industries. Our staff can only assist Adafruit customers, thank you!' To the right of this message is a search bar with the placeholder 'Enter keywords or topic' and a 'SEARCH' button. Below the message, there are links for 'View unanswered posts • View new posts • View active topics • Mark forums read'. A table follows, showing forum categories: 'GENERAL FORUMS' (Topics: 275, Posts: 1466, Last post: by dellymontana on Thu Sep 21, 2017 7:32 am), 'ANNOUNCEMENTS' (Forum announcements, Moderators: adafruit_support_bill, adafruit).

| | Topics | Posts | Last post |
|----------------|--------|-------|---|
| GENERAL FORUMS | 275 | 1466 | by dellymontana on Thu Sep 21, 2017 7:32 am |
| ANNOUNCEMENTS | | | |

The [Adafruit Forums](https://adafru.it/jlf) (<https://adafru.it/jlf>) 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](#) (<https://adafru.it/xXA>) category under "Supported Products & Projects" is the best place to post your CircuitPython questions.

The screenshot shows the Adafruit CircuitPython and MicroPython forum page. At the top, there's a navigation bar with links to 'Forum Index', 'Supported Products & Projects', and the current category. On the right, there are 'User Settings' and 'View your posts' links. Below the navigation, the category name 'Adafruit CircuitPython and MicroPython' is displayed, along with 'Moderators: adafruit_support_bill, adafruit'. A 'FORUM RULES' section follows, with a note that Adafruit MicroPython is currently EXPERIMENTAL and BETA. It also links to 'https://learn.adafruit.com/category/micropython' and 'http://forum.micropython.org/'. Below the rules, there are buttons for 'POST A TOPIC', 'SEARCH', and a link to 'Mark topics read'. A message encourages users to be positive and constructive. A table lists a single announcement: 'CIRCUITPYTHON 2.1.0 RELEASED!' by danhalbert on Oct 18, 2017, with 1 reply, 111 views, and the last post on Oct 20, 2017 at 2:47 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

The screenshot shows the GitHub repository page for 'adafruit/circuitpython'. The header includes a repository icon, the full repository name 'adafruit / circuitpython', a 'Search' bar, and navigation links for 'Pull requests', 'Issues', 'Marketplace', and 'Explore'. On the right, there are buttons for 'Unwatch', 'Star', 'Fork', and a counter showing 1,357 forks. Below the header, there are tabs for 'Code', 'Issues 73', 'Pull requests 4', and 'Insights'. The main content area displays the repository description: 'CircuitPython - a Python implementation for teaching coding with microcontrollers'. It shows the repository has been forked from 'micropython/micropython'. At the bottom, there are summary statistics: 9,856 commits, 32 branches, 73 releases, and 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](#) (<https://adafru.it/tB7>) itself. If you need an account, visit <https://github.com/> (<https://adafru.it/d6C>) and sign up.

If you're new to GitHub or programming in general, there are great opportunities for you. Head over to [adafruit/circuitpython](#) (<https://adafru.it/tB7>) on GitHub, click on "[Issues](#) (<https://adafru.it/Bee>)", and you'll find a list that includes issues labeled "[good first issue](#) (<https://adafru.it/Bef>)". 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 | good first issue | #338 opened 29 days ago by tannewt | | Long term | | 2 |
| <input type="checkbox"/> | | Feather M0 Adalogger | does not have D8 or D7 | good first issue | #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. | | good first issue | #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 interface. On the left is a sidebar with navigation links like 'Search docs', 'API AND USAGE', 'Core Modules' (which is expanded), 'Support Matrix', 'Modules' (which is expanded), and 'analogio', 'audiobusio', 'audiointerface', 'bitsbangio'. The main content area is titled 'audioio – Support for audio input and output'. It contains a brief description: 'The `audioio` module contains classes to provide access to audio IO.' Below this is a 'Libraries' section with a single item: '• `AudioOut` – Output an analog audio signal'. At the bottom, there are 'Previous' and 'Next' navigation buttons.

[ReadTheDocs](https://adafru.it/Beg) (<https://adafru.it/Beg>) 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](https://adafru.it/Beh) (<https://adafru.it/Beh>). 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 blinky:

```
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)
```

Advanced Serial Console on Windows

Windows 7 Driver

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

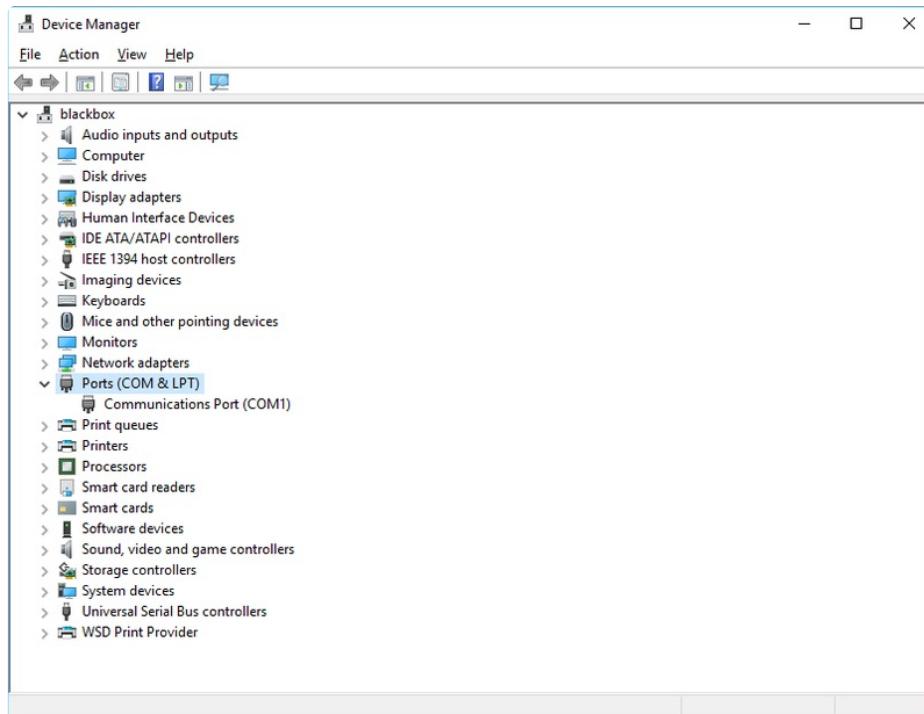
<https://adafru.it/AB0>

<https://adafru.it/AB0>

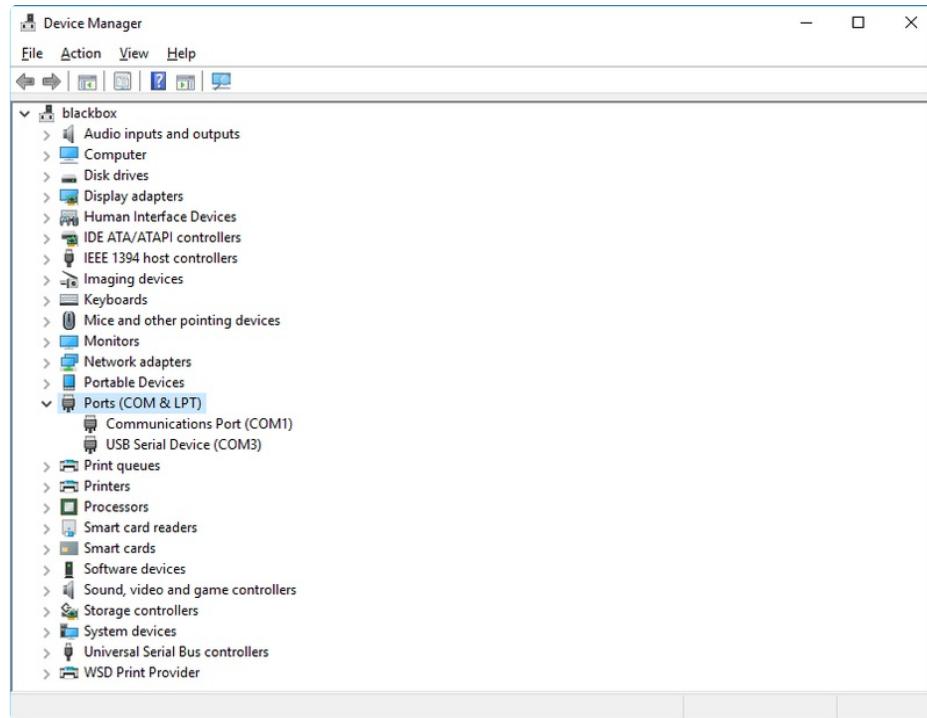
What's the COM?

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

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



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



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

Install Putty

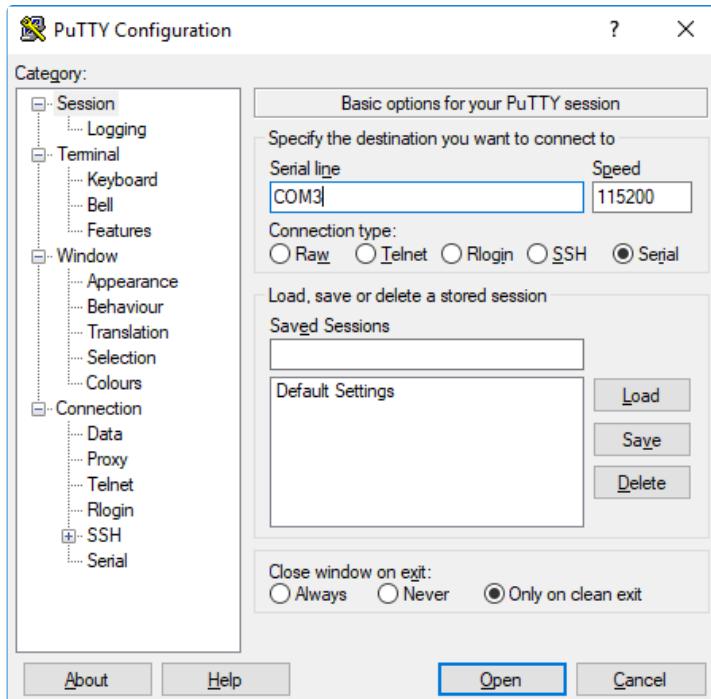
If you're using Windows, you'll need to download a terminal program. We're going to use PuTTY.

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

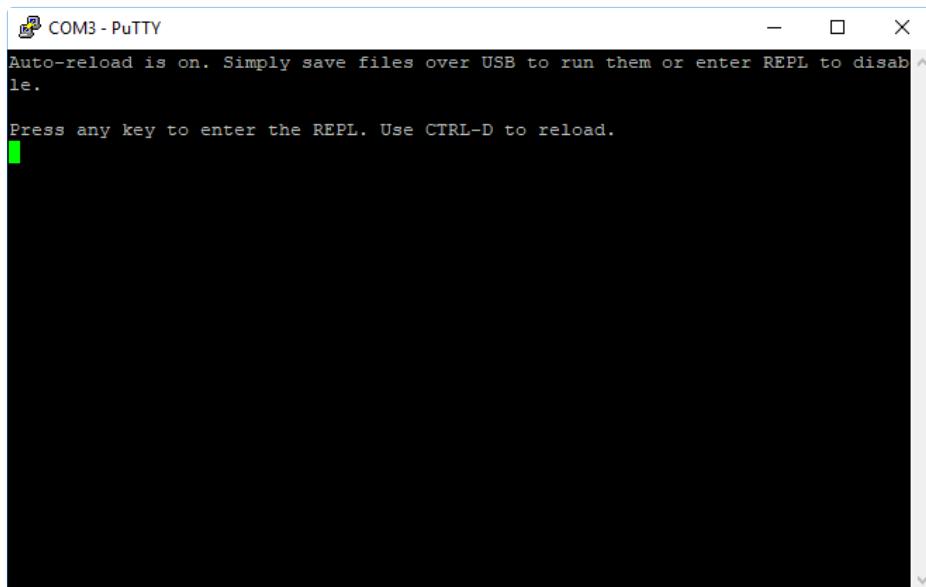
Now you need to open PuTTY.

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

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



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



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

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

Advanced Serial Console on Mac and Linux

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

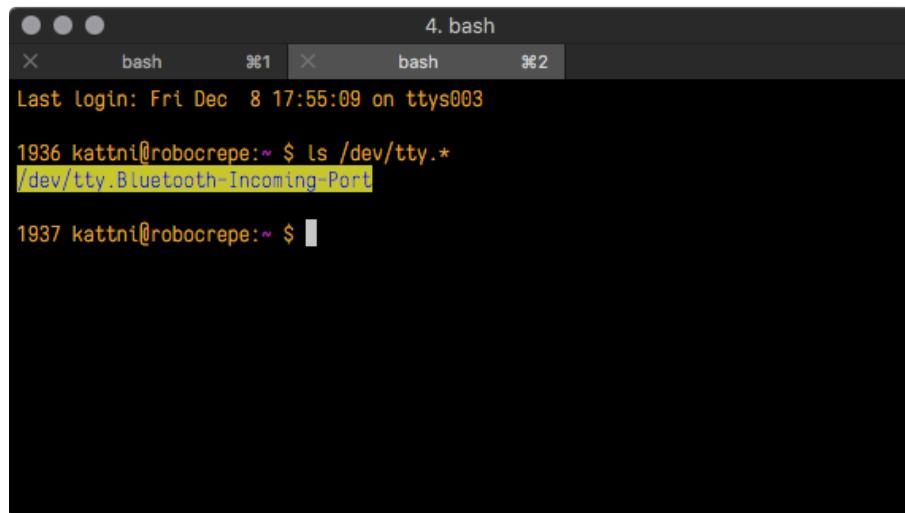
What's the Port?

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

We're going to use Terminal to determine what port the board is using. The easiest way to determine which port the board is using is to first check **without** the board plugged in. On Mac, open Terminal and type the following:

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

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

A screenshot of a Mac OS X desktop showing a Terminal window titled "4. bash". The window has two tabs: "bash" and "%2". The terminal output shows the user's last login information ("Last Login: Fri Dec 8 17:55:09 on ttys003") followed by the command "ls /dev/tty.*" and its output, which includes "/dev/ttys003" and "/dev/ttys004". The prompt "1936 kattni@robocrepe:~ \$" is visible at the bottom.

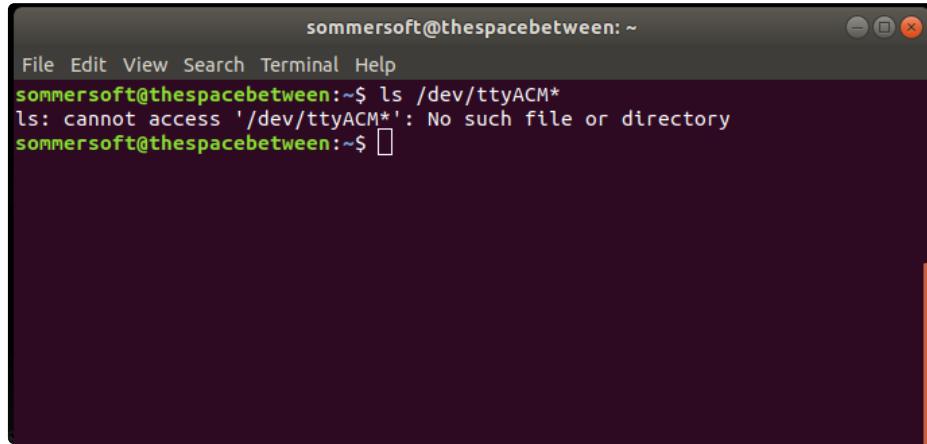
```
Last Login: Fri Dec 8 17:55:09 on ttys003
1936 kattni@robocrepe:~ $ ls /dev/tty.*
/dev/ttys003
/dev/ttys004
1937 kattni@robocrepe:~ $
```

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

```
ls /dev/ttyACM*
```

The concept is the same with Linux. We are asking to see the listings in the `/dev/` folder, starting with `ttyACM` and ending with anything. This will show you the current serial connections. In the example below,

the error is indicating that there are no current serial connections starting with `ttyACM`.

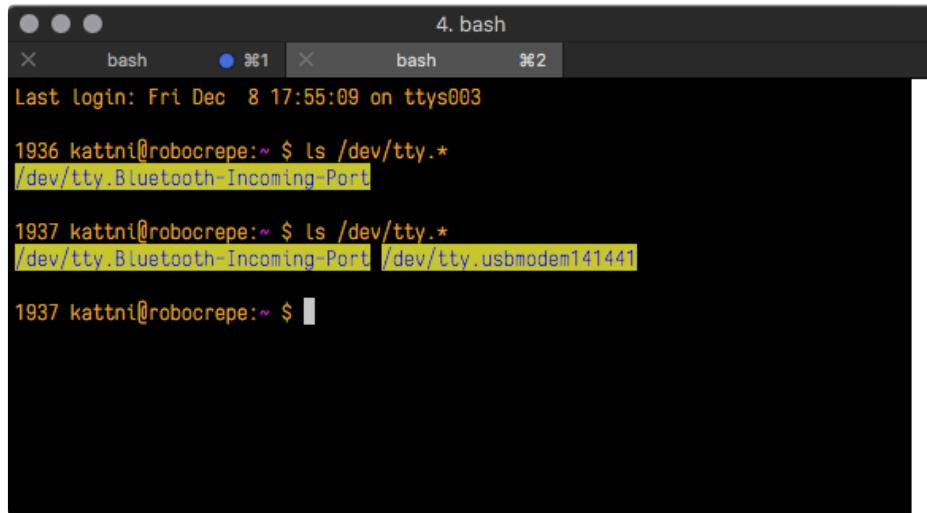


A screenshot of a terminal window titled "sommersoft@thespacebetween: ~". The window has a dark background and light-colored text. It shows the command `ls /dev/ttyACM*` being run, followed by an error message: "ls: cannot access '/dev/ttyACM*': No such file or directory". The cursor is shown as a small square at the end of the command line.

Now, plug your board. Using Mac, type:

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

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



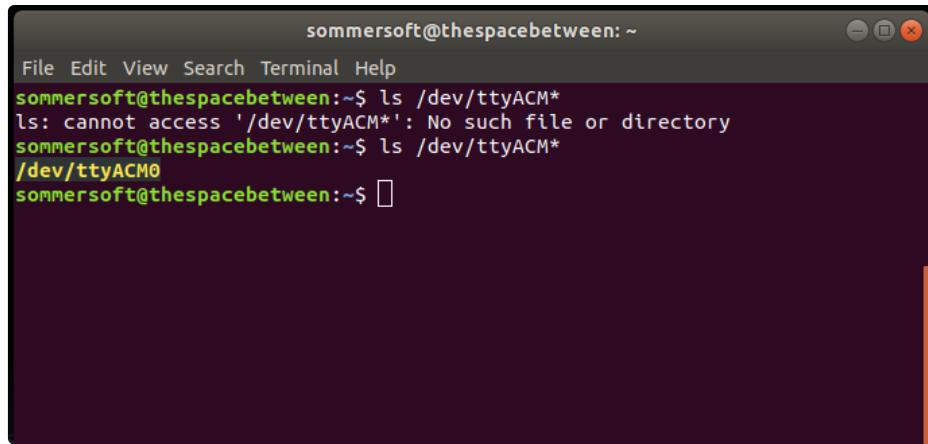
A screenshot of a terminal window titled "4. bash". The window has a dark background and light-colored text. It shows the command `ls /dev/tty.*` being run twice. The first run shows a connection named `/dev/tty.Bluetooth-Incoming-Port`. The second run shows two connections: `/dev/tty.Bluetooth-Incoming-Port` and `/dev/tty.usbmodem141441`. The cursor is shown as a small square at the end of the command line.

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

Using Linux, type:

```
ls /dev/ttyACM*
```

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



```
sommersoft@thespacebetween:~$ ls /dev/ttyACM*
ls: cannot access '/dev/ttyACM*': No such file or directory
sommersoft@thespacebetween:~$ ls /dev/ttyACM*
/dev/ttyACM0
sommersoft@thespacebetween:~$ 
```

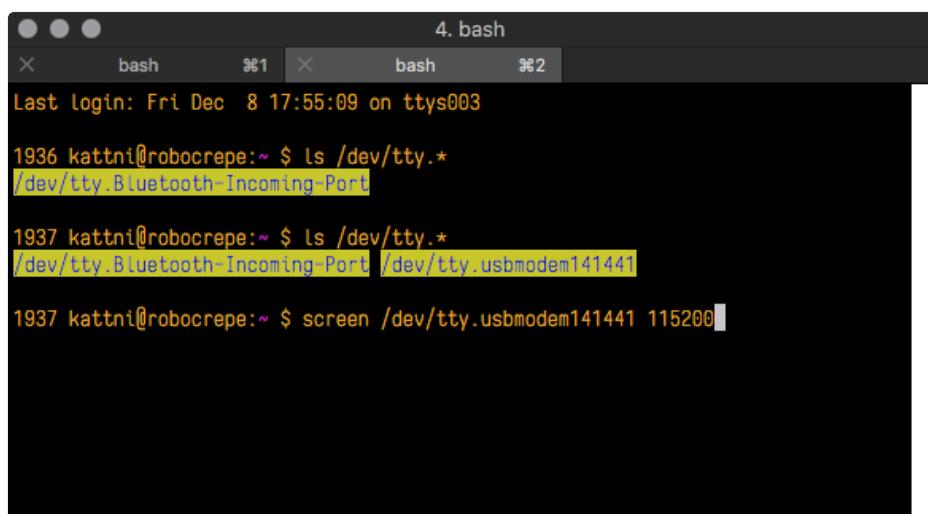
Using Linux, a new listing has appeared called `/dev/ttyACM0`. The `ttyACM0` part of this listing is the name the example board is using. Yours will be called something similar.

Connect with screen

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

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

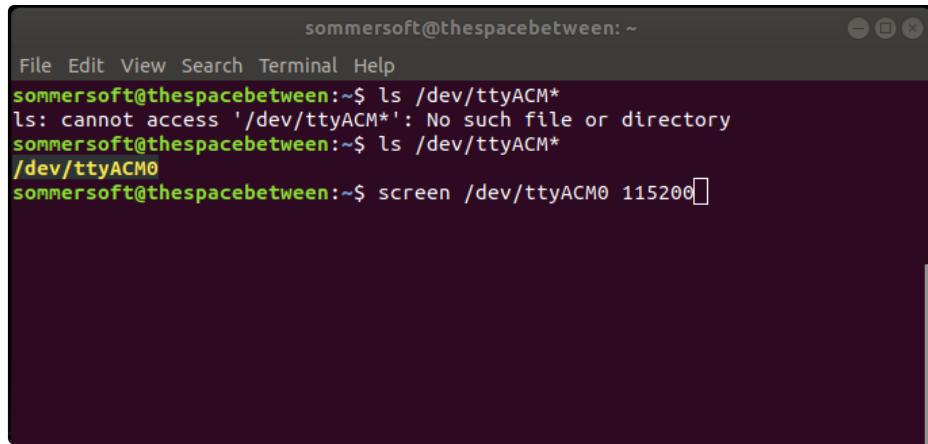
The first part of this establishes using the `screen` command. The second part tells `screen` the name of the board you're trying to use. The third part tells `screen` what baud rate to use for the serial connection. The baud rate is the speed in bits per second that data is sent over the serial connection.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



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

1937 kattni@robocrepe:~ $ ls /dev/tty.*
/dev/tty.Bluetooth-Incoming-Port /dev/tty.usbmodem141441

1937 kattni@robocrepe:~ $ screen /dev/tty.usbmodem141441 115200
```



A screenshot of a terminal window titled "sommersoft@thespacebetween: ~". The window has a dark background and light-colored text. It shows the following command-line session:

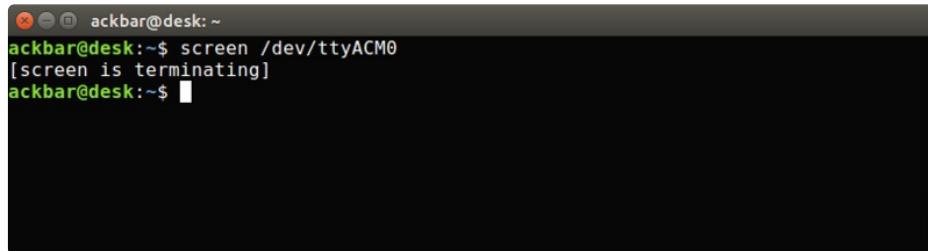
```
sommersoft@thespacebetween:~$ ls /dev/ttyACM*
ls: cannot access '/dev/ttyACM*': No such file or directory
sommersoft@thespacebetween:~$ ls /dev/ttyACM*
/dev/ttyACM0
sommersoft@thespacebetween:~$ screen /dev/ttyACM0 115200
```

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

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

Permissions on Linux

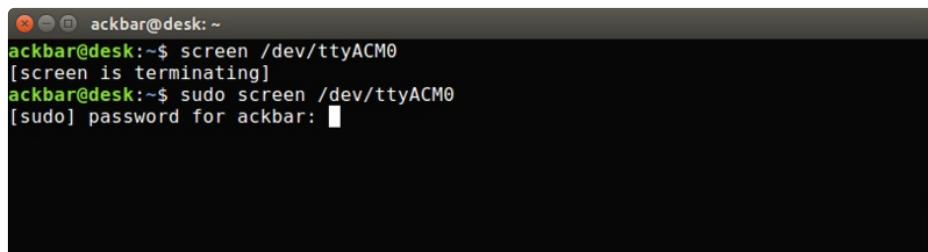
If you try to run `screen` and it doesn't work, then you may be running into an issue with permissions. Linux keeps track of users and groups and what they are allowed to do and not do, like access the hardware associated with the serial connection for running `screen`. So if you see something like this:



A screenshot of a terminal window titled "ackbar@desk: ~". The window has a dark background and light-colored text. It shows the following command-line session:

```
ackbar@desk:~$ screen /dev/ttyACM0
[screen is terminating]
ackbar@desk:~$
```

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



A screenshot of a terminal window titled "ackbar@desk: ~". The window has a dark background and light-colored text. It shows the following command-line session:

```
ackbar@desk:~$ screen /dev/ttyACM0
[screen is terminating]
ackbar@desk:~$ sudo screen /dev/ttyACM0
[sudo] password for ackbar:
```

Once you enter your password, you should be in:

```
ackbar@desk: ~
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 Trinket M0 with samd21e18
>>> 
```

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

Then use the command `adduser` to add yourself to that group. You need elevated privileges to do this, so you'll need to use `sudo`. In the example below, the group is `adm` and the user is `ackbar`.

```
ackbar@desk: ~
ackbar@desk:~$ ls -l /dev/ttyACM0
crw-rw---- 1 root adm 166, 0 Dec 21 08:29 /dev/ttyACM0
ackbar@desk:~$ sudo adduser ackbar adm
Adding user 'ackbar' to group `adm' ...
Adding user ackbar to group adm
Done.
ackbar@desk:~$ 
```

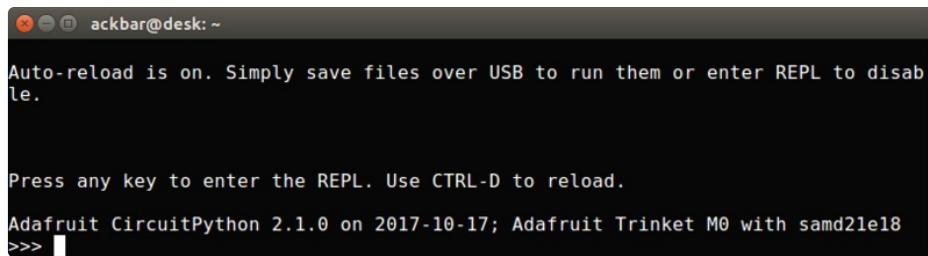
After you add yourself to the group, you'll need to logout and log back in, or in some cases, reboot your machine. After you log in again, verify that you have been added to the group using the command `groups`. If you are still not in the group, reboot and check again.

```
ackbar@desk: ~
ackbar@desk:~$ groups
ackbar adm sudo
ackbar@desk:~$ 
```

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

```
ackbar@desk: ~
ackbar@desk:~$ groups
ackbar adm sudo
ackbar@desk:~$ screen /dev/ttyACM0 115200
```

And you're in:



A screenshot of a terminal window titled "ackbar@desk: ~". The window displays the following text:

```
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 Trinket M0 with samd2le18
>>> █
```

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

Uninstalling CircuitPython

A lot of our boards can be used with multiple programming languages. For example, the Circuit Playground Express can be used with MakeCode, Code.org CS Discoveries, CircuitPython and Arduino.

Maybe you tried CircuitPython and want to go back to MakeCode or Arduino? Not a problem

You can always remove/re-install CircuitPython *whenever you want!* Heck, you can change your mind every day!

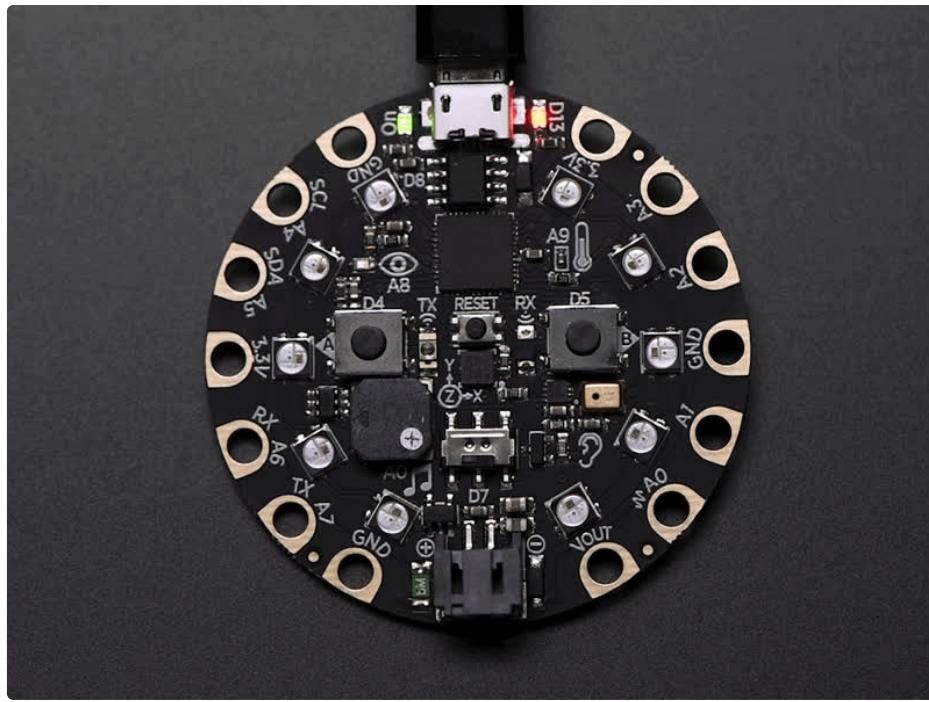
Backup Your Code

Before uninstalling CircuitPython, don't forget to make a backup of the code you have on the little disk drive. That means your `main.py` or `code.py` any other files, the `lib` folder etc. You may lose these files when you remove CircuitPython, so backups are key! Just drag the files to a folder on your laptop or desktop computer like you would with any USB drive.

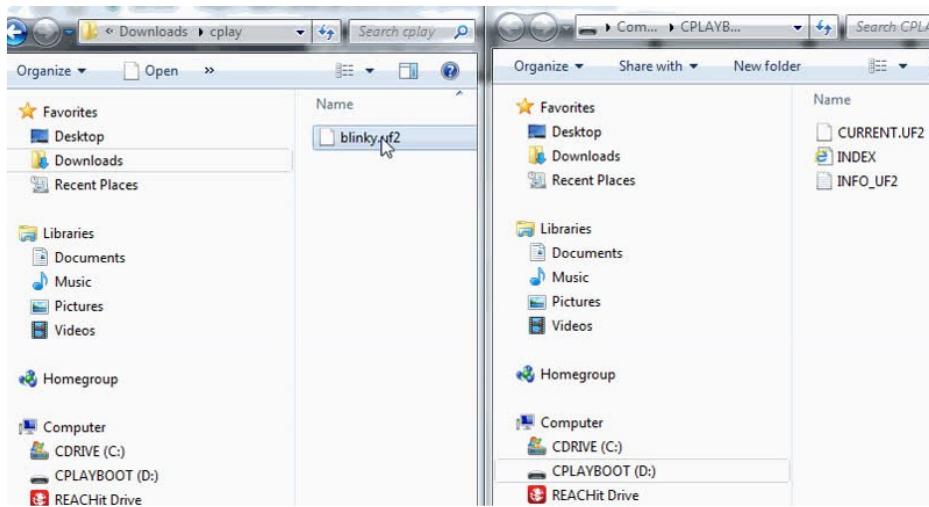
Moving Circuit Playground Express to MakeCode

On the Circuit Playground Express (**this currently does NOT apply to Circuit Playground Bluefruit**), if you want to go back to using MakeCode, it's really easy. Visit makecode.adafruit.com (<https://adafru.it/wpC>) and find the program you want to upload. Click Download to download the `.uf2` file that is generated by MakeCode.

Now double-click your CircuitPython board until you see the onboard LED(s) turn green and the `...BOOT` directory shows up.



Then find the downloaded MakeCode .uf2 file and drag it to the ...BOOT drive.



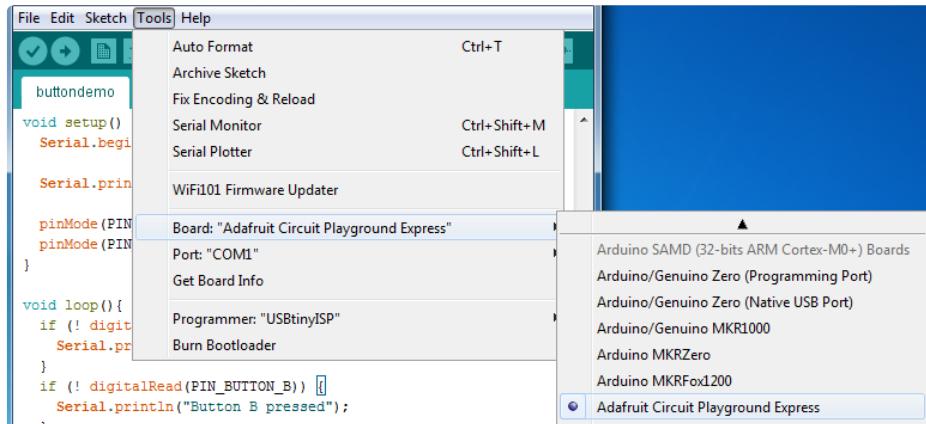
Your MakeCode is now running and CircuitPython has been removed. Going forward you only have to **single click** the reset button

Moving to Arduino

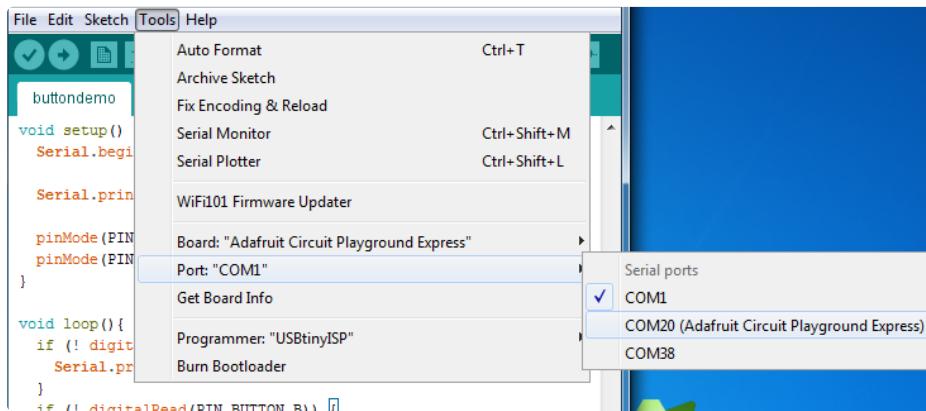
If you want to change your firmware to Arduino, it's also pretty easy.

Start by plugging in your board, and double-clicking reset until you get the green onboard LED(s) - just like with MakeCode

Within Arduino IDE, select the matching board, say Circuit Playground Express



Select the correct matching Port:



Create a new simple Blink sketch 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
}
```

Make sure the LED(s) are still green, then click **Upload** to upload Blink. Once it has uploaded successfully, the serial Port will change so **re-select the new Port!**

Once Blink is uploaded you should no longer need to double-click to enter bootloader mode, Arduino will

automatically reset when you upload

Troubleshooting

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

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

Always Run the Latest Version of CircuitPython and Libraries

As we continue to develop CircuitPython and create new releases, we will stop supporting older releases. You need to [update to the latest CircuitPython](#). (<https://adafru.it/Em8>).

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

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

I have to continue using CircuitPython 5.x, 4.x, 3.x or 2.x, where can I find compatible libraries?

We are no longer building or supporting the CircuitPython 2.x, 3.x, 4.x or 5.x library bundles. We highly encourage you to [update CircuitPython to the latest version](#) (<https://adafru.it/Em8>) and use [the current version of the libraries](#) (<https://adafru.it/ENC>). However, if for some reason you cannot update, you can find [the last available 2.x build here](#) (<https://adafru.it/FJA>), [the last available 3.x build here](#) (<https://adafru.it/FJB>), [the last available 4.x build here](#) (<https://adafru.it/QDL>), and [the last available 5.x build here](#) (<https://adafru.it/QDJ>).

CPLAYBOOT, TRINKETBOOT, FEATHERBOOT, or GEMMABOOT Drive Not Present

You may have a different board.

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

MakeCode

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

MacOS

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

Windows 10

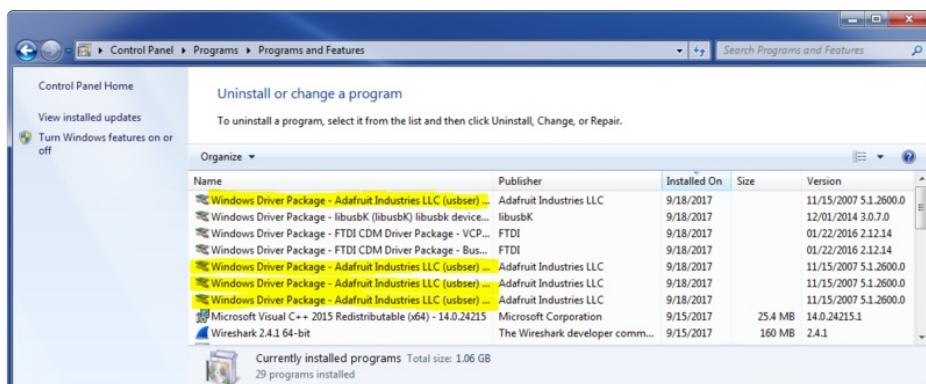
Did you install the Adafruit Windows Drivers package by mistake, or did you upgrade to Windows 10 with the driver package installed? You don't need to install this package on Windows 10 for most Adafruit boards. The old version (v1.5) can interfere with recognizing your device. Go to **Settings -> Apps**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

Windows 7 or 8.1

Version 2.5.0.0 or later of the Adafruit Windows Drivers will fix the missing `boardnameBOOT` drive problem on Windows 7 and 8.1. 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 ...".

We [recommend](#) (<https://adafru.it/Amd>) that you upgrade to Windows 10 if possible; an upgrade is probably still free for you: see the link.

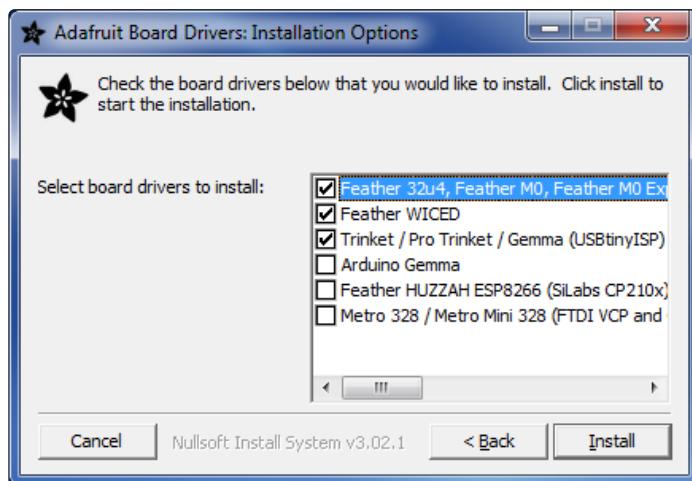


- Now install the new 2.5.0.0 (or higher) Adafruit Windows Drivers Package:

<https://adafru.it/ABO>

<https://adafru.it/ABO>

- When running the installer, you'll be shown a list of drivers to choose from. You can check and uncheck the boxes to choose which drivers to install.



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

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

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

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

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

uninstallation.

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

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

CIRCUITPY Drive Does Not Appear

Kaspersky anti-virus can block the appearance of the **CIRCUITPY** drive. We haven't yet figured out a settings change that prevents this. Complete uninstallation of Kaspersky fixes the problem.

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

Windows 7 and 8.1 Problems

Windows 7 and 8.1 can become confused about USB device installations. We [recommend](https://adafruit.it/Amd) (<https://adafruit.it/Amd>) that you upgrade to Windows 10 if possible; an upgrade is probably still free for you: see the link. If not, try cleaning up your USB devices with your board unplugged. Use [Uwe Sieber's Device Cleanup Tool](https://adafruit.it/RWd) (<https://adafruit.it/RWd>), which you must run as Administrator.

Serial Console in Mu Not Displaying Anything

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

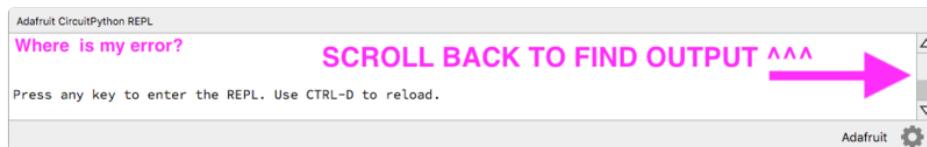
Depending on the size of your screen or Mu window, when you open the serial console, the serial console panel may be very small. This can be a problem. A basic CircuitPython error takes 10 lines to display!

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

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

More complex errors take even more lines!

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



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

CircuitPython RGB Status Light

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

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

Here's what the colors and blinking mean:

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

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

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

These are followed by flashes indicating the line number, including place value. **WHITE** flashes are thousands' place, **BLUE** are hundreds' place, **YELLOW** are tens' place, and **CYAN** are one's place. So for example, an error on line 32 would flash **YELLOW** three times and then **CYAN** two times. Zeroes are indicated by an extra-long dark gap.

ValueError: Incompatible .mpy file.

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

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

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

CIRCUITPY Drive Issues

You may find that you can no longer save files to your `CIRCUITPY` drive. You may find that your `CIRCUITPY` stops showing up in your file explorer, or shows up as `NO_NAME`. These are indicators that your filesystem has issues.

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

If still broken - When the `CIRCUITPY` disk is not safely ejected before being reset by the button or being disconnected from USB, it may corrupt the flash drive. It can happen on Windows, Mac or Linux.

In this situation, the board must be completely erased and CircuitPython must be reloaded onto the board.

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

Easiest Way: Use `storage.erase_filesystem()`

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

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

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

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

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

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

1. Download the correct erase file:

<https://adafru.it/Adl>

<https://adafru.it/Adl>

<https://adafru.it/AdJ>

<https://adafru.it/AdJ>

<https://adafru.it/EVK>

<https://adafru.it/EVK>

<https://adafru.it/AdK>

<https://adafru.it/AdK>

<https://adafru.it/EoM>

<https://adafru.it/EoM>

<https://adafru.it/DjD>

<https://adafru.it/DjD>

<https://adafru.it/DBA>

<https://adafru.it/DBA>

<https://adafru.it/Eca>

<https://adafru.it/Eca>

<https://adafru.it/Gnc>

<https://adafru.it/Gnc>

<https://adafru.it/GAN>

<https://adafru.it/GAN>

<https://adafru.it/GAO>

<https://adafru.it/GAO>

<https://adafru.it/Jat>

<https://adafru.it/Jat>

<https://adafru.it/Q5B>

<https://adafru.it/Q5B>

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

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

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

[If you haven't already downloaded the latest release of CircuitPython for your board, check out the](#)

[installation page](https://adafru.it/Amd) (<https://adafru.it/Amd>). You'll also need to install your libraries and code!

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

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

1. Download the erase file:

<https://adafru.it/AdL>

<https://adafru.it/AdL>

2. Double-click the reset button on the board to bring up the `boardnameBOOT` drive.
3. Drag the erase `.uf2` file to the `boardnameBOOT` drive.
4. The boot LED will start flashing again, and the `boardnameBOOT` drive will reappear.
5. [Drag the appropriate latest release CircuitPython](#) (<https://adafru.it/Amd>) `.uf2` file to the `boardnameBOOT` drive.

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

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

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

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

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

Running Out of File Space on Non-Express Boards

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

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

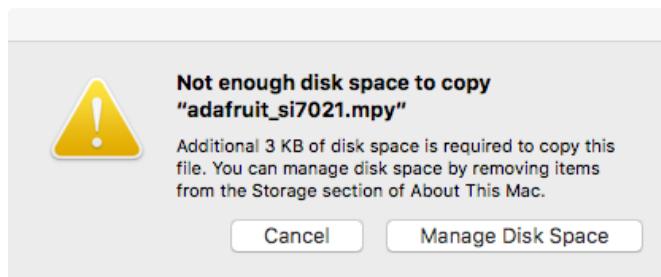
Delete something!

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

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.

MacOS loves to add extra files.



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

Prevent & Remove MacOS Hidden Files

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

```
ls -l /Volumes
```

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

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

```
mdutil -i off /Volumes/CIRCUITPY
cd /Volumes/CIRCUITPY
rm -rf .{,_}{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.

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

```
>>> import storage
>>> storage.erase_filesystem
```

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

Copy Files on MacOS Without Creating Hidden Files

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

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

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

(Replace `foo.mpy` with the name of the file you want to copy.) Or to copy a folder and all of its child files/folders use a command like:

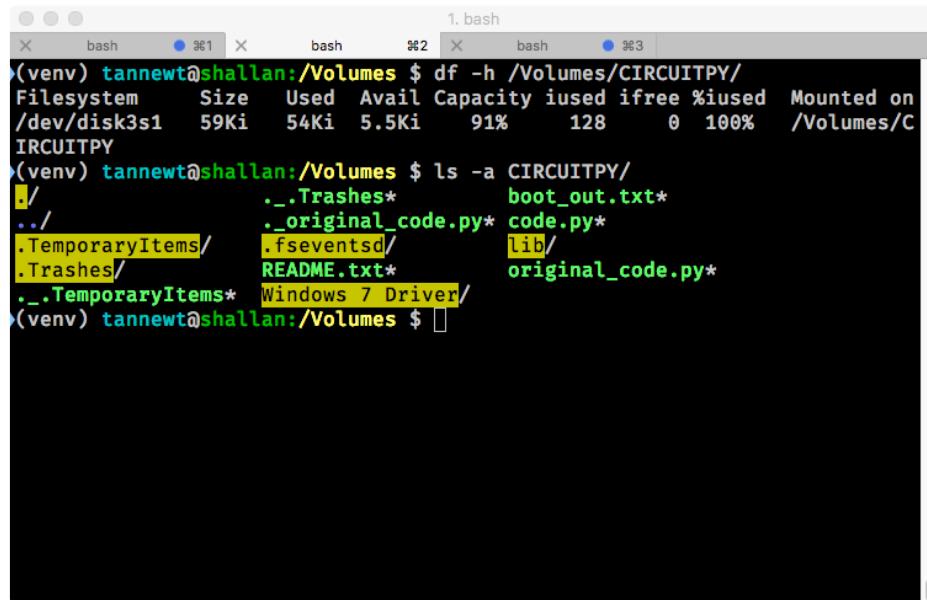
```
cp -rX folder_to_copy /Volumes/CIRCUITPY
```

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

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

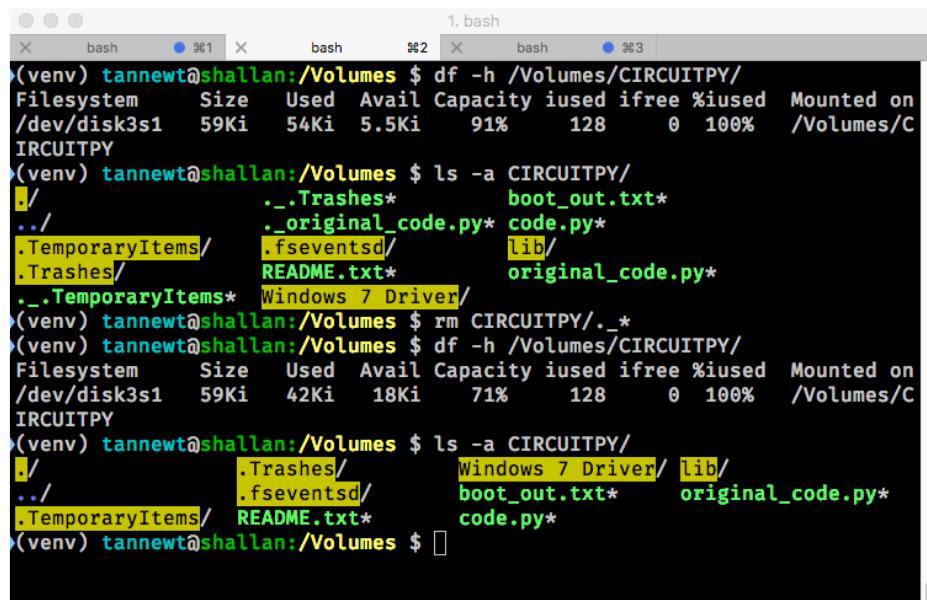
Other MacOS Space-Saving Tips

If you'd like to see the amount of space used on the drive and manually delete hidden files here's how to do so. First list the amount of space used on the **CIRCUITPY** drive with the **df** command:



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

Lets remove the **._** files first.



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

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

Device locked up or boot looping

In rare cases, it may happen that something in your `code.py` or `boot.py` files causes the device to get locked up, or even go into a boot loop. These are not your everyday Python exceptions, typically it's the result of a deeper problem within CircuitPython. In this situation, it can be difficult to recover your device if **CIRCUITPY** is not allowing you to modify the `code.py` or `boot.py` files. Safe mode is one recovery option. When the device boots up in safe mode it will not run the `code.py` or `boot.py` scripts, but will still connect the **CIRCUITPY** drive so that you can remove or modify those files as needed.

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

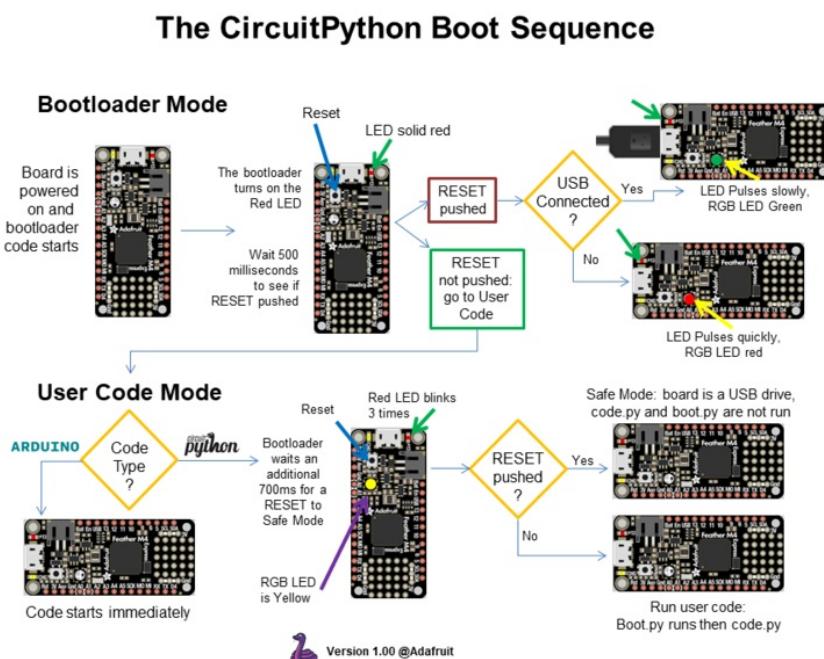
For most devices:

Press the reset button, and then when the RGB status LED is yellow, press the reset button again.

For ESP32-S2 based devices:

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

Refer to the following diagram for boot sequence details:



CircuitPython CANio Library

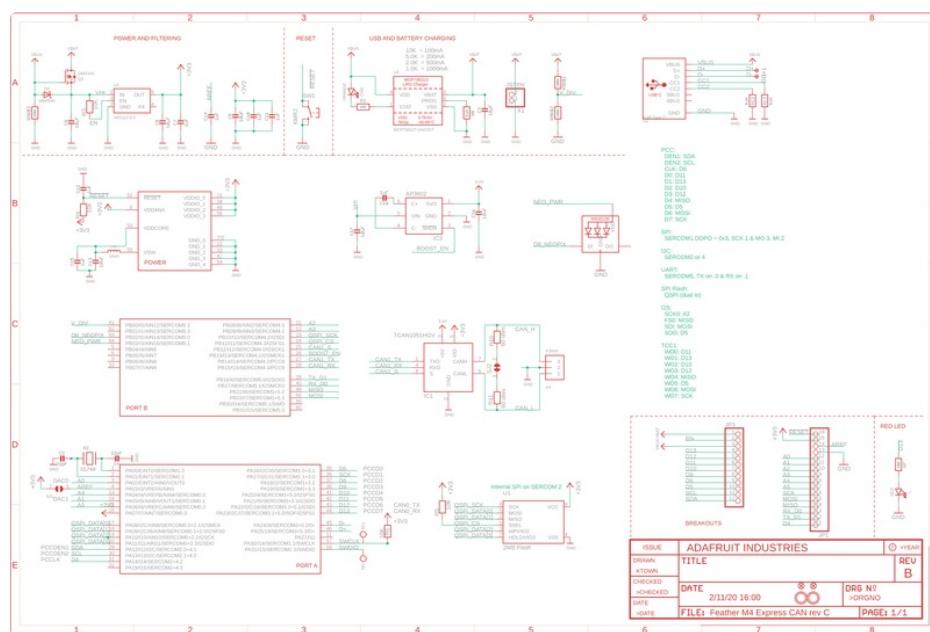
[CircuitPython CANio Library \(https://adafru.it/Qb-\)](https://adafru.it/Qb-)

Downloads

Files:

- [ATSAME51J19 datasheet](https://adafruit.it/Pbs) (<https://adafruit.it/Pbs>)
- [TCAN1051-Q1 datasheet](https://adafruit.it/QCf) (<https://adafruit.it/QCf>)
- [AP3602A datasheet](https://adafruit.it/QCg) (<https://adafruit.it/QCg>)
- [Fritzing object in the Adafruit Fritzing Library](https://adafruit.it/Pbt) (<https://adafruit.it/Pbt>)
- [EagleCAD PCB files on GitHub](https://adafruit.it/Pbu) (<https://adafruit.it/Pbu>)

Schematic



Fab Print

