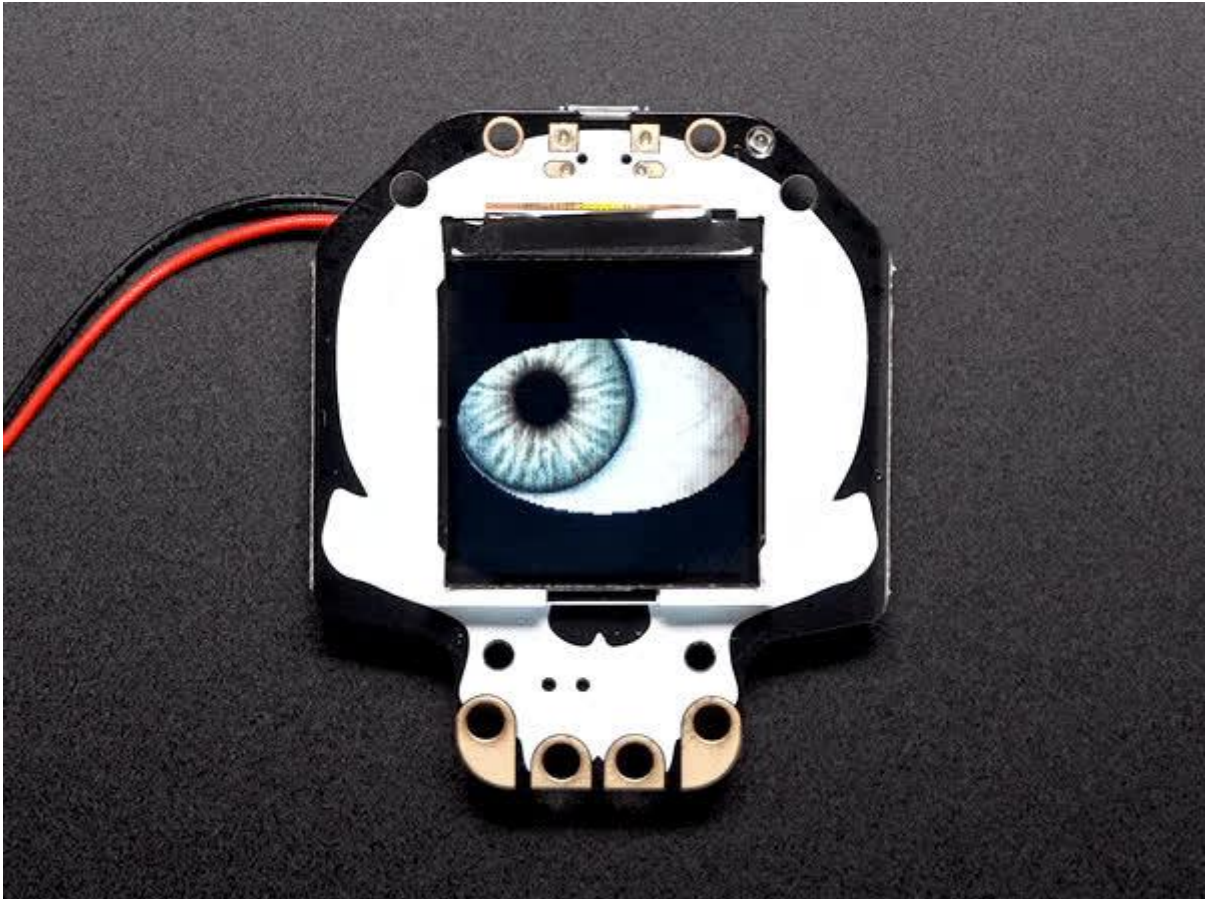




Adafruit Hallowing

Created by lady ada



<https://learn.adafruit.com/adafruit-hallowing>

Last updated on 2022-01-01 02:32:15 PM EST

Table of Contents

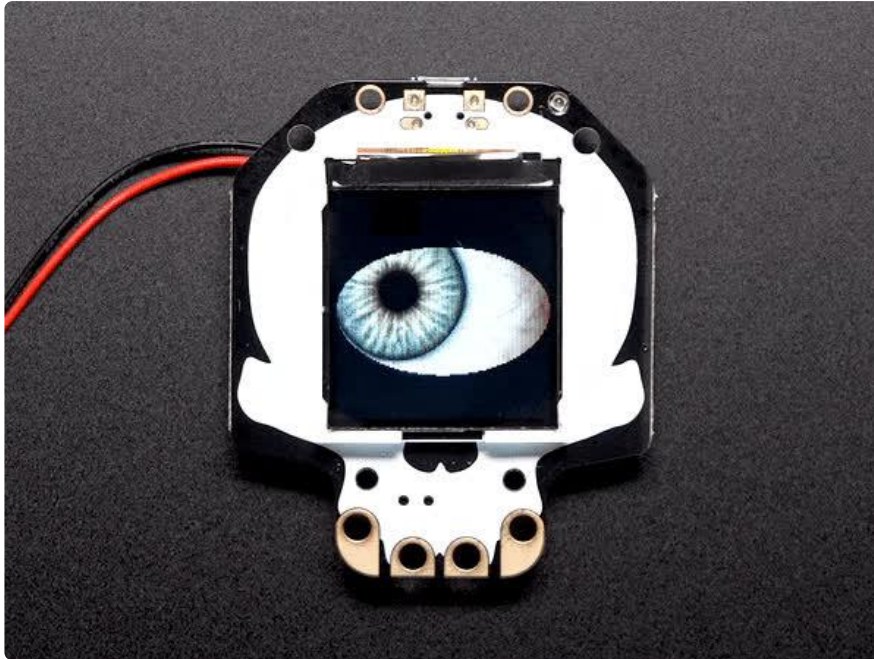
Overview	7
Pinouts	10
• Power Pins & Ports	11
• Chip & Flash	12
• Sensors	12
• Speaker	13
• LEDs	13
• TFT	14
Arduino IDE Setup	14
Using with Arduino IDE	17
• Install SAMD Support	17
• Install Adafruit SAMD	18
• Install Drivers (Windows 7 & 8 Only)	19
• Blink	21
• Successful Upload	22
• Compilation Issues	23
• Manually bootloading	24
• Ubuntu & Linux Issue Fix	24
Adapting Sketches to M0 & M4	24
• Analog References	25
• Pin Outputs & Pullups	25
• Serial vs SerialUSB	25
• AnalogWrite / PWM on Feather/Metro M0	26
• analogWrite() PWM range	27
• analogWrite() DAC on A0	28
• Missing header files	28
• Bootloader Launching	28
• Aligned Memory Access	29
• Floating Point Conversion	29
• How Much RAM Available?	29
• Storing data in FLASH	30
• Pretty-Printing out registers	30
• M4 Performance Options	31
• CPU Speed (overclocking)	31
• Optimize	32
• Cache	32
• Max SPI and Max QSPI	32
• Enabling the Buck Converter on some M4 Boards	33
Using SPI Flash	33
• Read & Write CircuitPython Files	35
• Format Flash Memory	37
• Datalogging Example	38
• Reading and Printing Files	39
• Full Usage Example	39
• Accessing SPI Flash	40
Feather HELP!	41

Arcada Libraries	46
• Install Libraries	46
• Adafruit Arcada	47
• If you aren't running Arduino IDE 1.8.10 or later, you'll need to install all of the following!	47
• Adafruit NeoPixel	47
• Adafruit FreeTouch	48
• Adafruit Touchscreen	48
• Adafruit SPIFlash	48
• Adafruit Zero DMA	48
• Adafruit GFX	48
• Adafruit ST7735	49
• Adafruit ILI9341	49
• Adafruit LIS3DH	49
• Adafruit Sensor	49
• Adafruit ImageReader	50
• ArduinoJson	50
• Adafruit ZeroTimer	50
• Adafruit TinyUSB	50
• Adafruit WavePlayer	51
• SdFat (Adafruit Fork)	51
• Audio - Adafruit Fork	51
Using the TFT	51
• Install Libraries	51
• Setup	52
• Graphics Test Code	53
Full Test Sketch	57
• Install Libraries	57
Arcada Library	58
• Initialization	58
• Joystick & Buttons	59
• Backlight, Speaker and Sensors	59
• Alert Boxes	59
Arcada Library Docs	60
Spooky Eyes	61
• Customizing the Spooky Eye Demo	62
Synchronized Eyes	62
• Synchronized Eyes with Two HalloWings	62
What is CircuitPython?	62
• CircuitPython is based on Python	63
• Why would I use CircuitPython?	63
CircuitPython	64
• Set up CircuitPython Quick Start!	64
Installing the Mu Editor	66
• Download and Install Mu	67
• Starting Up Mu	67
• Using Mu	68

Creating and Editing Code	68
• Creating Code	69
• Editing Code	71
• Back to Editing Code...	72
• Naming Your Program File	73
Connecting to the Serial Console	73
• Are you using Mu?	74
• Serial Console Issues or Delays on Linux	75
• Setting Permissions on Linux	75
• Using Something Else?	76
Interacting with the Serial Console	76
The REPL	79
• Entering the REPL	80
• Interacting with the REPL	81
• Returning to the Serial Console	83
CircuitPython Libraries	84
• The Adafruit CircuitPython Library Bundle	85
• Downloading the Adafruit CircuitPython Library Bundle	85
• The CircuitPython Community Library Bundle	86
• Downloading the CircuitPython Community Library Bundle	86
• Understanding the Bundle	87
• Example Files	88
• Copying Libraries to Your Board	88
• Understanding Which Libraries to Install	88
• Example: ImportError Due to Missing Library	91
• Library Install on Non-Express Boards	92
• Updating CircuitPython Libraries and Examples	93
CircuitPython Pins and Modules	93
• CircuitPython Pins	93
• import board	94
• I2C, SPI, and UART	95
• What Are All the Available Names?	96
• Microcontroller Pin Names	97
• CircuitPython Built-In Modules	98
Troubleshooting	98
• Always Run the Latest Version of CircuitPython and Libraries	99
• I have to continue using CircuitPython 5.x or earlier. Where can I find compatible libraries?	99
• Bootloader (boardnameBOOT) Drive Not Present	100
• Windows Explorer Locks Up When Accessing boardnameBOOT Drive	101
• Copying UF2 to boardnameBOOT Drive Hangs at 0% Copied	101
• CIRCUITPY Drive Does Not Appear	102
• Device Errors or Problems on Windows	102
• Serial Console in Mu Not Displaying Anything	103
• code.py Restarts Constantly	104
• CircuitPython RGB Status Light	104
• CircuitPython 7.0.0 and Later	104
• CircuitPython 6.3.0 and earlier	105
• Serial console showing ValueError: Incompatible .mpy file	107
• CIRCUITPY Drive Issues	107

• Safe Mode	107
• To erase CIRCUITPY: storage.erase_filesystem()	109
• Erase CIRCUITPY Without Access to the REPL	110
• For the specific boards listed below:	110
• For SAMD21 non-Express boards that have a UF2 bootloader:	112
• For SAMD21 non-Express boards that do not have a UF2 bootloader:	112
• Running Out of File Space on SAMD21 Non-Express Boards	113
• Delete something!	113
• Use tabs	113
• On MacOS?	114
• Prevent & Remove MacOS Hidden Files	114
• Copy Files on MacOS Without Creating Hidden Files	115
• Other MacOS Space-Saving Tips	115
• Device Locked Up or Boot Looping	116
 "Uninstalling" CircuitPython	117
• Backup Your Code	117
• Moving Circuit Playground Express to MakeCode	118
• Moving to Arduino	119
 Welcome to the Community!	120
• Adafruit Discord	121
• CircuitPython.org	122
• Adafruit GitHub	126
• Adafruit Forums	128
• Read the Docs	129
 UF2 Bootloader Details	129
• Entering Bootloader Mode	131
• Using the Mass Storage Bootloader	133
• Using the BOSSA Bootloader	134
• Running bossac on the command line	137
• Updating the bootloader	139
• Getting Rid of Windows Pop-ups	141
• Making your own UF2	142
• Installing the bootloader on a fresh/bricked board	142
 Downloads	142
• Files	142
• Schematic & Fabrication Print	143
 Troubleshooting	144
• TFT Screen Adhesive	144
• Double Stick Tape	146
• E6000 Glue	147
• Sugru	149
• Diagnostics	150

Overview



[This is Hallowing..this is Hallowing... Hallowing! Hallowing! \(https://adafru.it/C8m\)](https://adafru.it/C8m)

Are you the kind of person who doesn't like taking down the skeletons and spiders until after January? Well, we've got the development board for you. This is electronics at its most spooky! The Adafruit HalloWing is a skull-shaped ATSAM21 board with a ton of extras built in to make for an adorable wearable, badge, development kit, or the engine for your next cosplay or prop.



On the front is a cute 1.44" sized 128x128 full color TFT. Our default example code has our spooky eye demo running but you can use it for anything you like to display in glorious color.

There's also 4 fang-teeth below the display, these are analog/capacitive touch inputs with big alligator-clip holes.

On the reverse is a smorgasbord of electronic goodies:

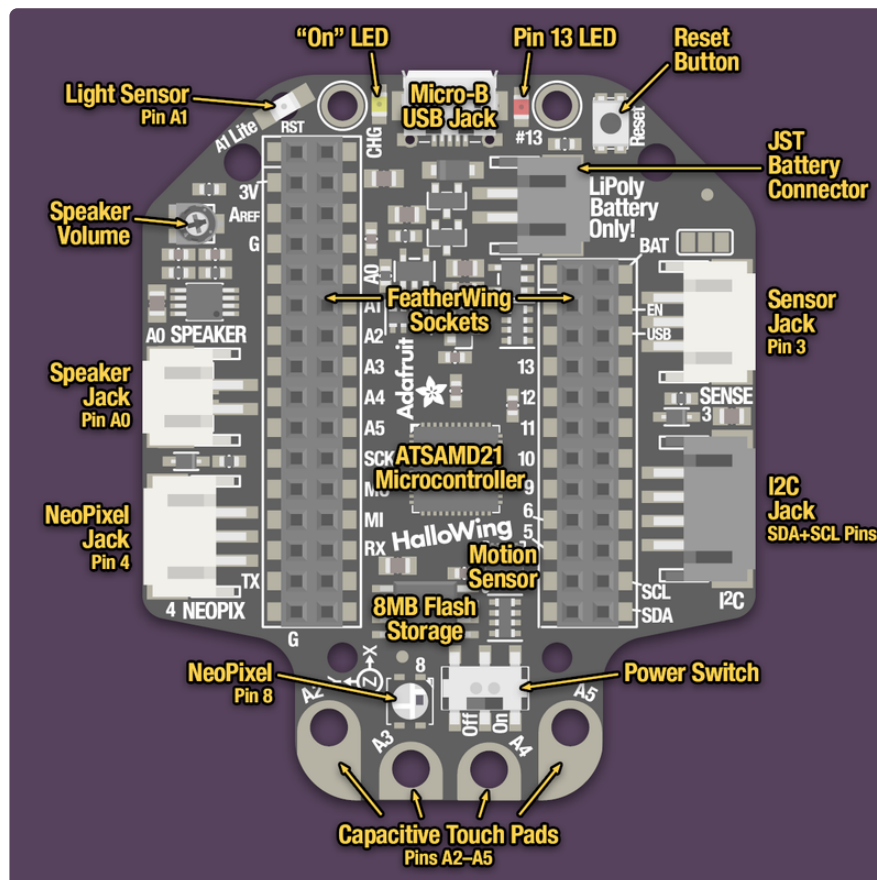
- ATSAM21G18 @ 48MHz with 3.3V logic/power - 256KB of FLASH + 32KB of RAM
- 8 MB of SPI Flash for storing images, sounds, animations, whatever!
- 3-axis accelerometer (motion sensor)
- Light sensor, reverse-mount so that it points out the front
- LiPoly battery port with built in recharging capability
- USB port for battery charging, programming and debugging
- Two female header strips with Feather-compatible pinout so you can plug any FeatherWings in
- Mono Class-D speaker driver for 4-8 ohm speakers, up to 2 Watts, with mini volume pot
- JST ports for Neopixels, sensor input, and I2C (you can fit I2C Grove connectors in here)
- 3.3V regulator with 500mA peak current output
- Reset button
- On-Off switch

OK so technically it's more like a really tricked-out Feather than a Wing but we simply could not resist the Halloween pun.

On each side of the HalloWing are JST-PH plugs for connecting external devices. The 3-pin JSTs connect to analog pins on the SAMD21, so you can use them for analog inputs. We label one for Neopixel and one for Sensors since we think most people will have one of each. The 4-pin JST connector connects to the I2C port and you can fit Grove connectors in it for additional hardware support.

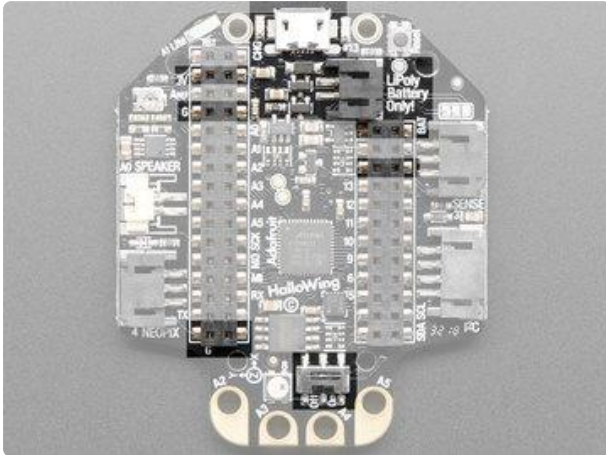
Comes fully assembled and ready to be your spooky friend. We install the UF2 bootloader on it so updating code and converting it to CircuitPython is easy.

Pinouts



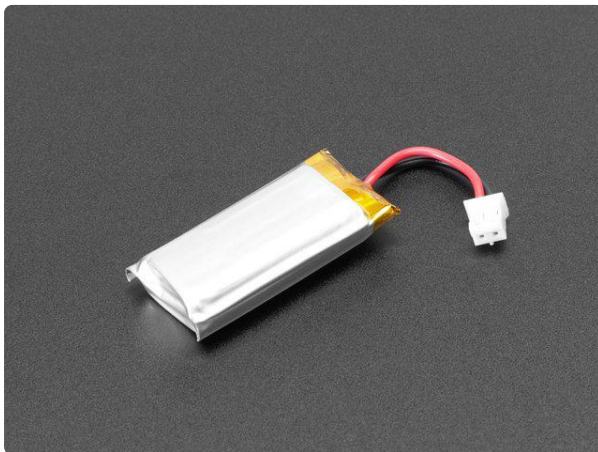
We put a ton of stuff on this HalloWing, above you can see a 'guided tour' of whats available!

Power Pins & Ports



There's two ways to power your Hallowing. The best way is to plug in a 3.7/4.2V Lipoly battery into the JST 2-PH port. You can then recharge the battery over the Micro USB jack. You can also just run the board directly from Micro USB, it will automatically 'switch over' to USB power when that's plugged in

The Hallowing JST battery port is expecting a LiPo with the 'standard' Adafruit polarity wiring. Using other battery packs with opposite wiring or voltages may destroy your Hallowing!



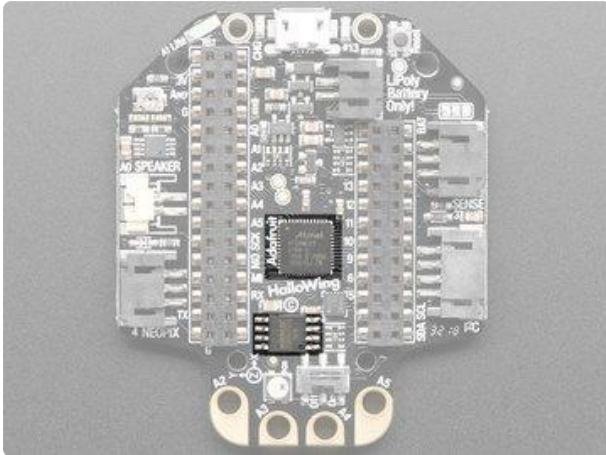
[Lithium Ion Polymer Battery Ideal For Feathers - 3.7V 400mAh](https://www.adafruit.com/product/3898)

Lithium-ion polymer (also known as 'lipo' or 'lipoly') batteries are thin, light, and powerful. The output ranges from 4.2V when completely charged to 3.7V. This...
<https://www.adafruit.com/product/3898>

You can turn off power completely with the on/off switch at the bottom of the board.

If you need access to the power pins, the 'Feather Headers' have 3.3V regulated out, GND (labeled G) on the left. On the right there's the BAT pin (connects directly to lipoly) and two pins below that is the USB pin. You can measure the voltage on the battery by reading analog pin A6 - this is divided by two with resistors so don't forget to x2 once you do the reading. The voltage, after doubling, will range from about 3.5 (empty) to 4.2V (charged)

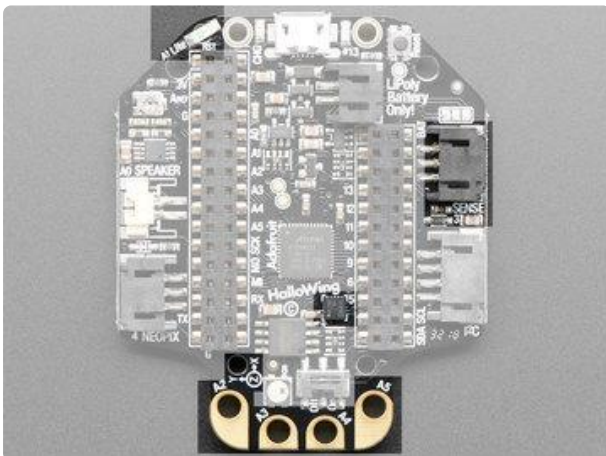
Chip & Flash



The main processor chip is the ATSAM21G18 Cortex M0+ running at 48MHz with 3.3V logic/power. It has 256KB of FLASH + 32KB of RAM and can run Arduino or CircuitPython

We also include 8 MB of SPI Flash for storing images, sounds, animations, whatever!

Sensors



There's a few built in sensors.

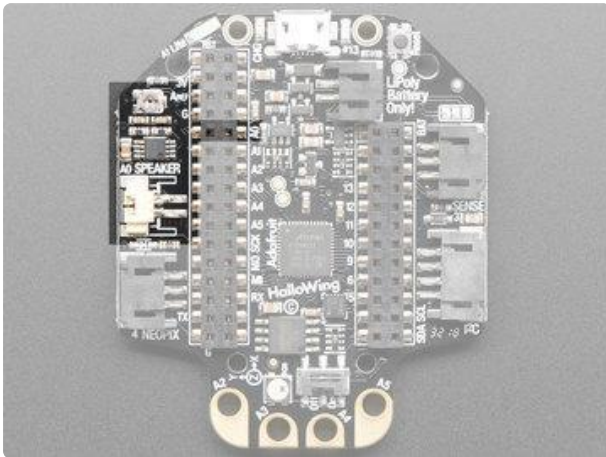
On the top there's a light sensor, connected to pin A1 - it's reverse mount so you can read light levels from the front.

There's also a LIS3DH 3-axis accelerometer connected to the I2C pins for detection motion, tilt or taps

On the bottom of the board are four pads designed for capacitive touch. They are connected to A2, A3, A4 and A5

On the right is a SENSE port, this is a JST 3-PH for connecting an external sensor. From the top to bottom the pads are GND, V+, D3 (in Arduino this is also A11). V+ is either LiPoly or USB power, whichever is plugged in and higher. There's a 470 ohm+3.6V zener diode connection to protect against voltages higher than 3.3V coming in.

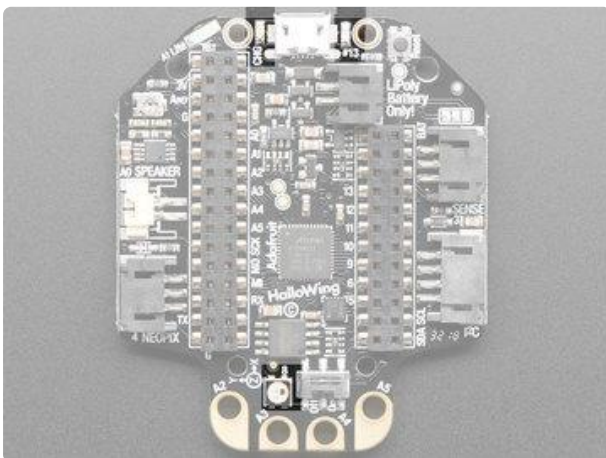
Speaker



We have a mono 2 Watt class D audio amp connected to A0 - that's the DAC output on the SAMD21, so you can get 10-bit audio output, good for many simple sound effects or musical output. There's a small smt trimpot you can adjust if you want, but the default 50% setting is pretty good.

The connector for the speaker is a Molex PicoBlade (<https://adafru.it/C8p>), but there's large pads you can solder too if you want to connect a custom speaker

LEDs

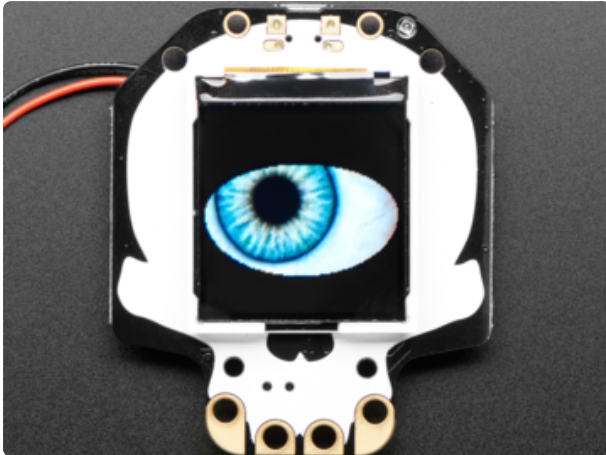


There are three LEDs - a red LED on pin D13, a CHG LED that will let you know when the battery is charging, and a NeoPixel on D8 (in Arduino) or `board.NEOPIXEL` (in CircuitPython)

It's normal for the yellow CHG LED to flicker when no battery is in place, that's the charge circuitry trying to detect whether a battery is there or not. If you are powering only over USB, you can cover it with tape

The charge LED is automatically driven by the Lipoly charger circuit. It will try to detect a battery and is expecting one to be attached. If there isn't one it may flicker once in a while when you use power because it's trying to charge a (non-existent) battery. It's not harmful, and its totally normal!

TFT



On the front is a 128x128 TFT. The TFT is connected to the SPI pins: SCK and MOSI (MISO is not connected)

We also use pin #37 for TFT Reset, #38 for TFT DC, #39 for TFT CS, and #7 for the backlight which is default-off.

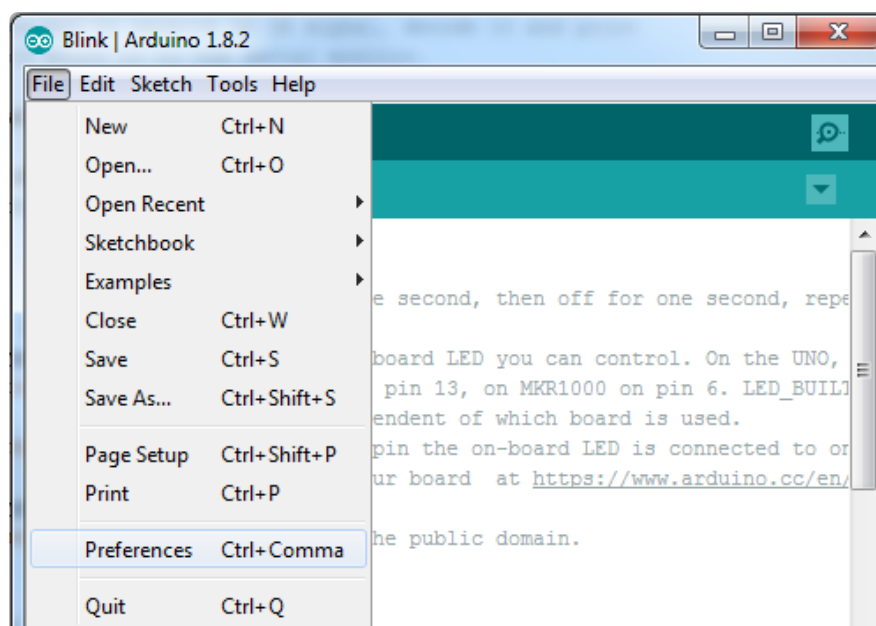
Arduino IDE Setup

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

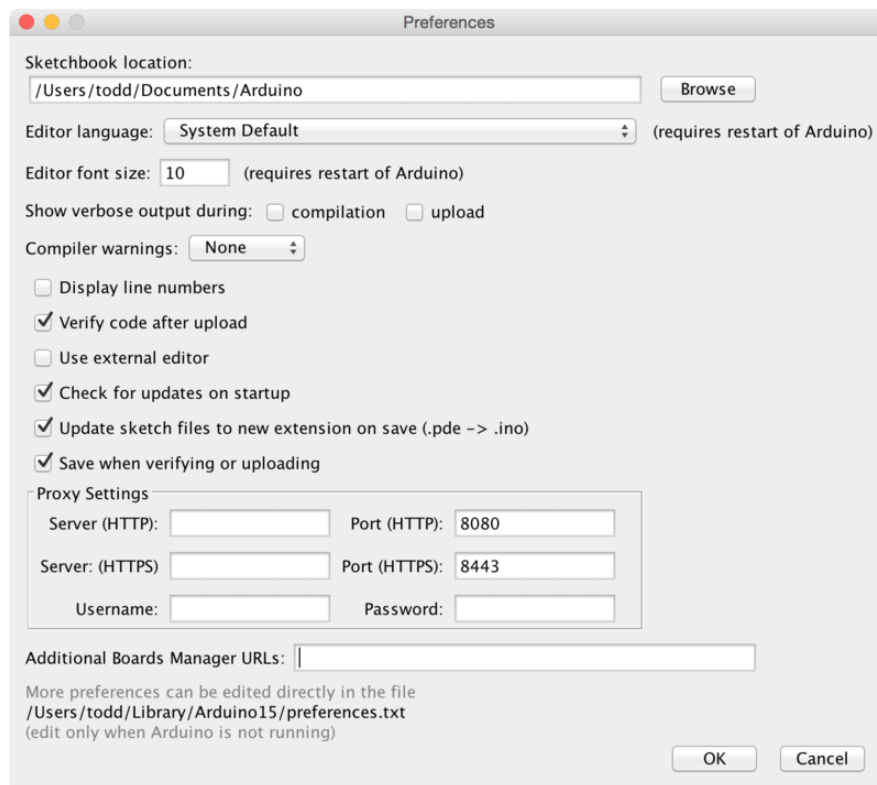
Arduino IDE Download

<https://adafru.it/f1P>

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



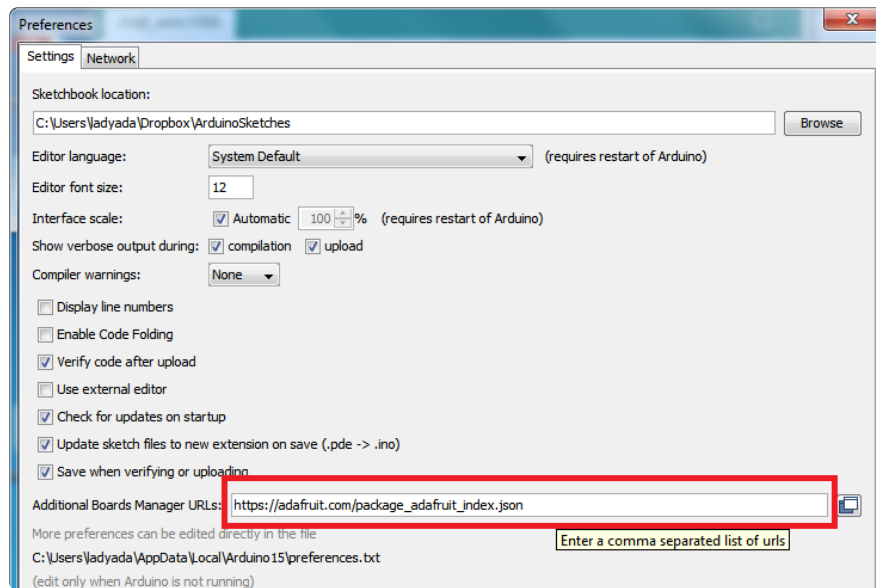
A dialog will pop up just like the one shown below.



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

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

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



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

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

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

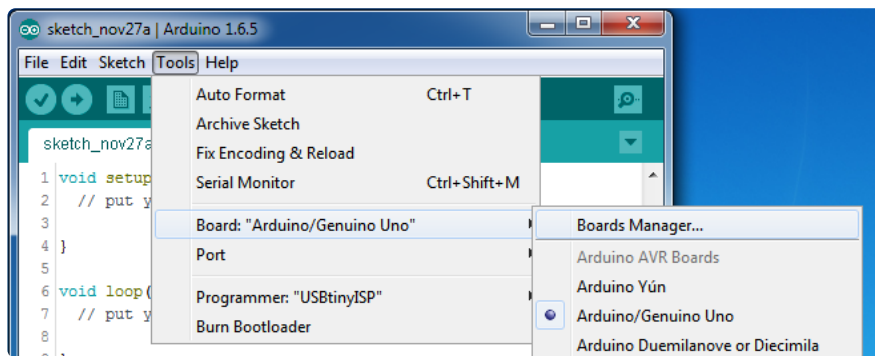
Once done click OK to save the new preference settings. Next we will look at installing boards with the Board Manager.

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

Using with Arduino IDE

The Feather/Metro/Gemma/QTPy/Trinket M0 and M4 use an ATSAM21 or ATSAM51 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.



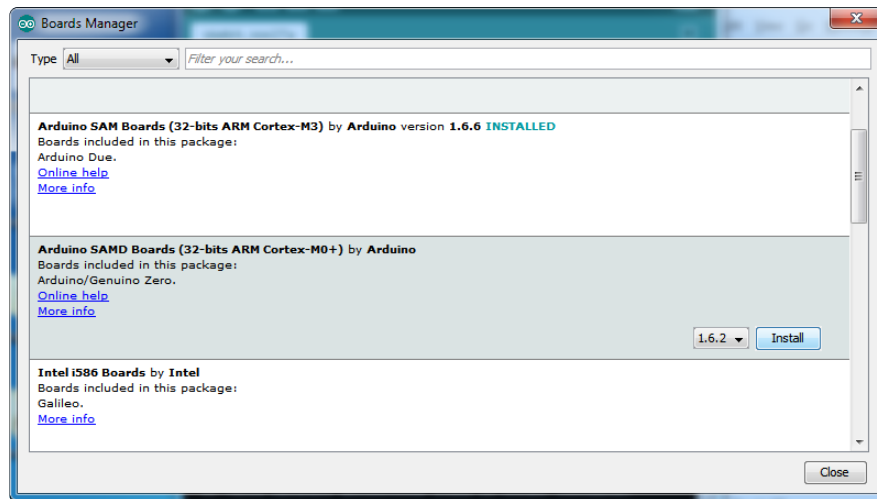
Once the Board Manager opens, click on the category drop down menu on the top left hand side of the window and select All. You will then be able to select and install the boards supplied by the URLs added to the preferences.

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 I nstall

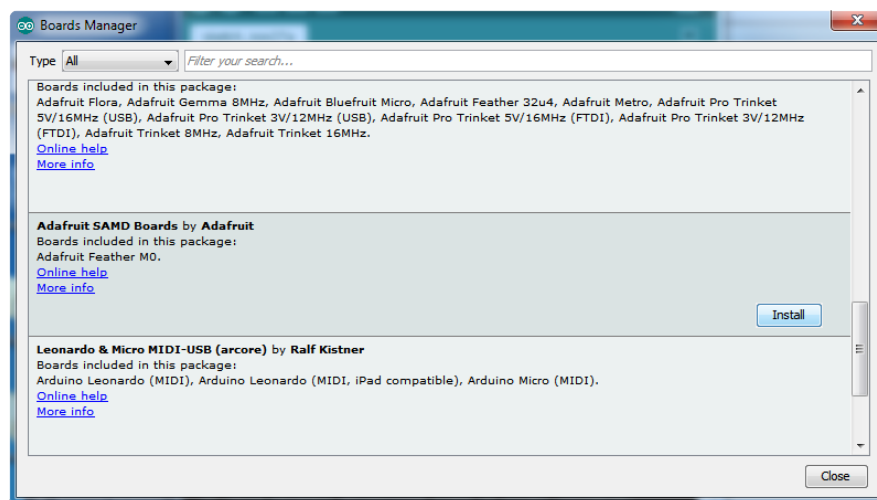


Install Adafruit SAMD

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

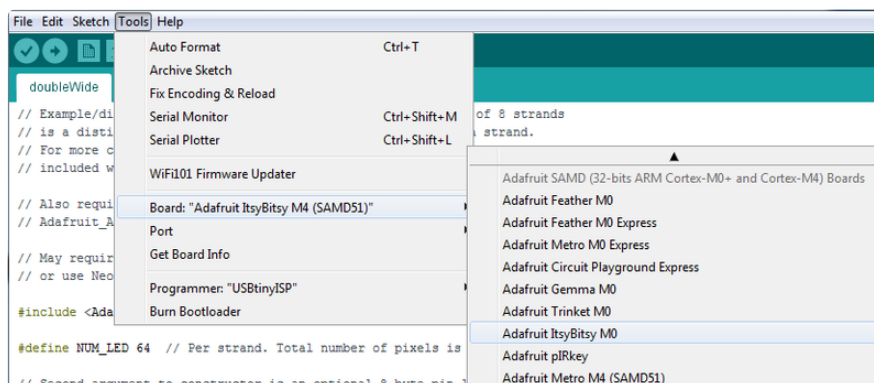


Even though in theory you don't need to - I recommend rebooting the IDE

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)

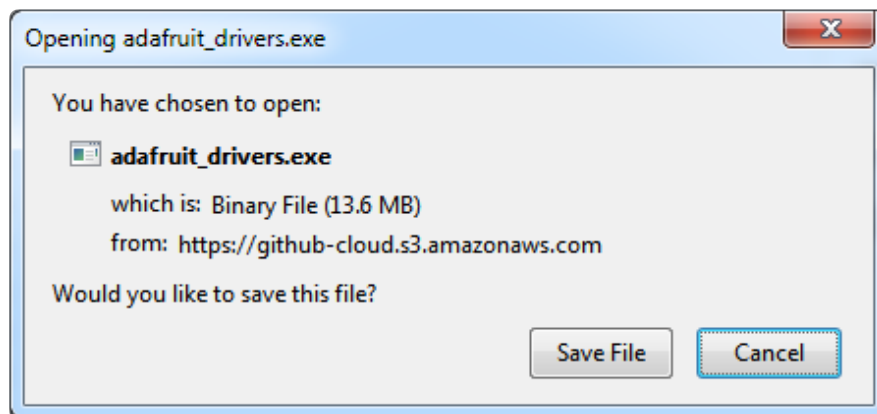
When you plug in the board, you'll need to possibly install a driver

Click below to download our Driver Installer

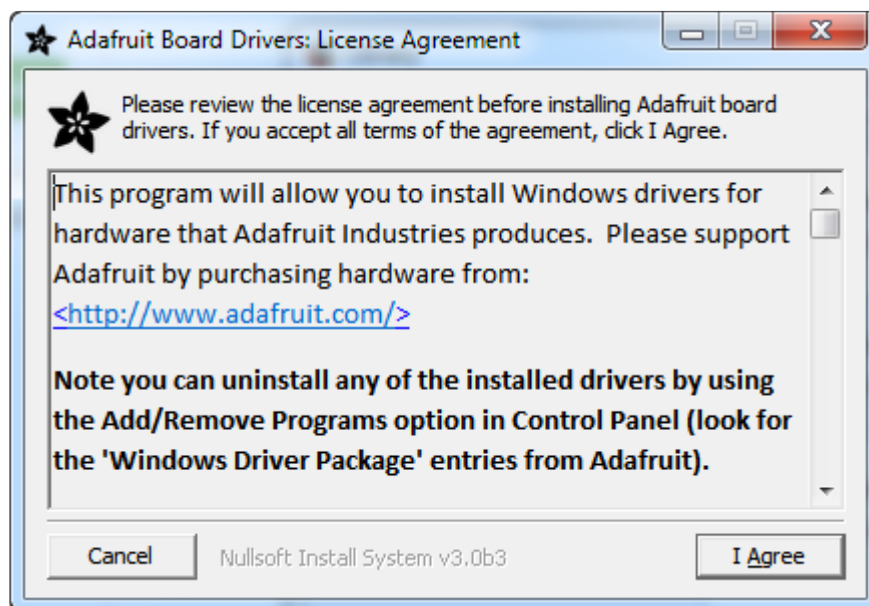
Download Latest Adafruit Drivers
package

<https://adafru.it/mb8>

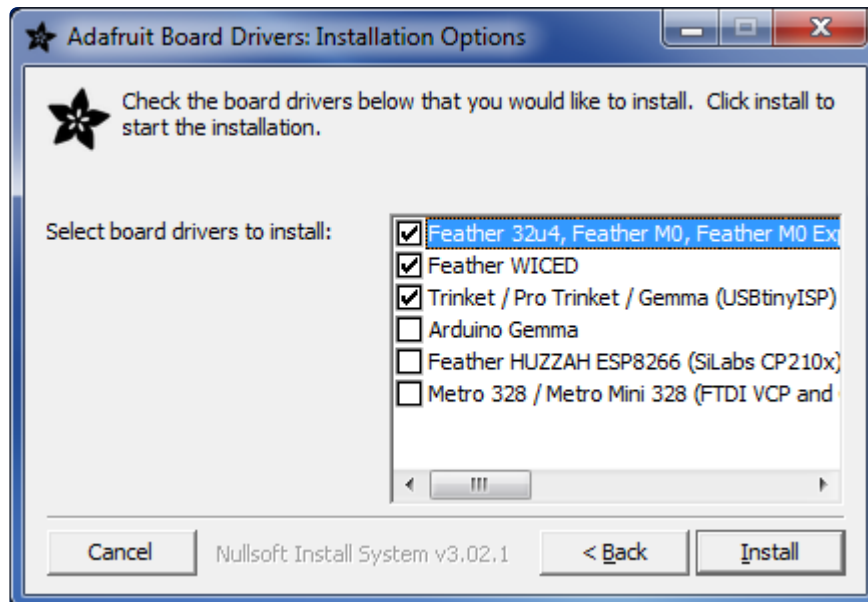
Download and run the installer



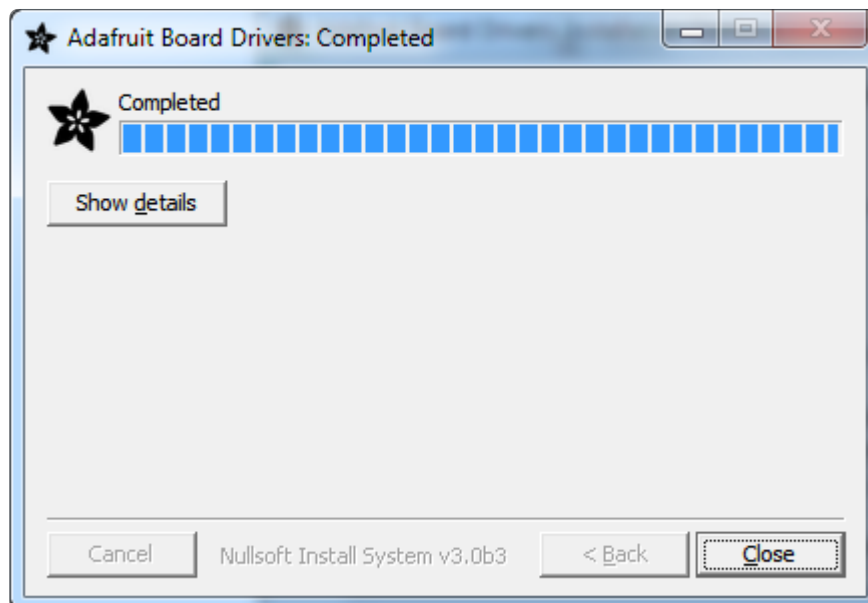
Run the installer! Since we bundle the SiLabs and FTDI drivers as well, you'll need to click through the license



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'

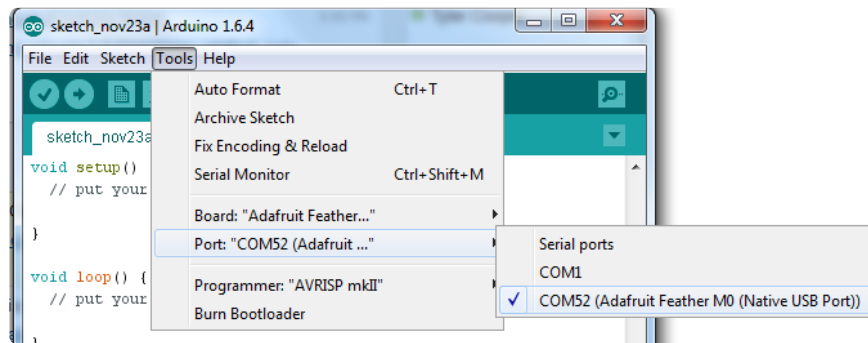


Blink

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

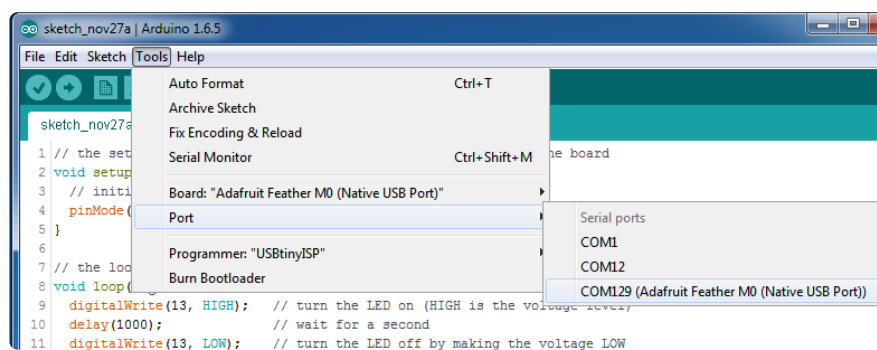
If you have a successful upload, you'll get a bunch of red text that tells you that the device was found and it was programmed, verified & reset

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

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

Analog References

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

Pin Outputs & Pullups

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

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

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

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

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

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

Serial vs SerialUSB

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

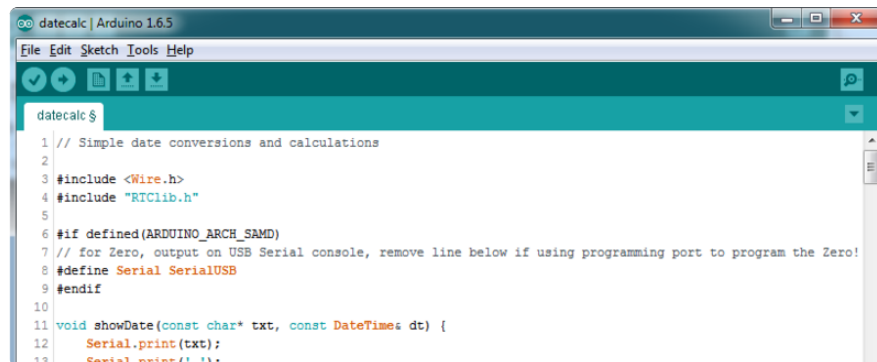
In the Adafruit M0/M4 Core, we fixed it so that `Serial` goes to USB so it will automatically work just fine.

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

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

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

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



AnalogWrite / PWM on Feather/Metro M0

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

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

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

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

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

- TC[0-4],WO[0-1]

- TCC[0-2],WO[0-7]

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

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

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

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

- Analog pin A5

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

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

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

- TX and SDA (Digital pins 1 and 20)

analogWrite() PWM range

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

analogWrite() DAC on A0

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

Missing header files

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

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

you'll get an error that says

```
fatal error: util/delay.h: No such file or directory
```

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

^

```
compilation terminated.
```

```
Error compiling.
```

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

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

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

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

Bootloader Launching

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

Aligned Memory Access

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

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

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

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

Floating Point Conversion

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

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

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

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

How Much RAM Available?

The 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, it's a little easier, simply add const before the variable name:

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

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

You can verify where data is stored by printing out the address:
`Serial.print("Address of str "); Serial.println((int)&str, HEX);`

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

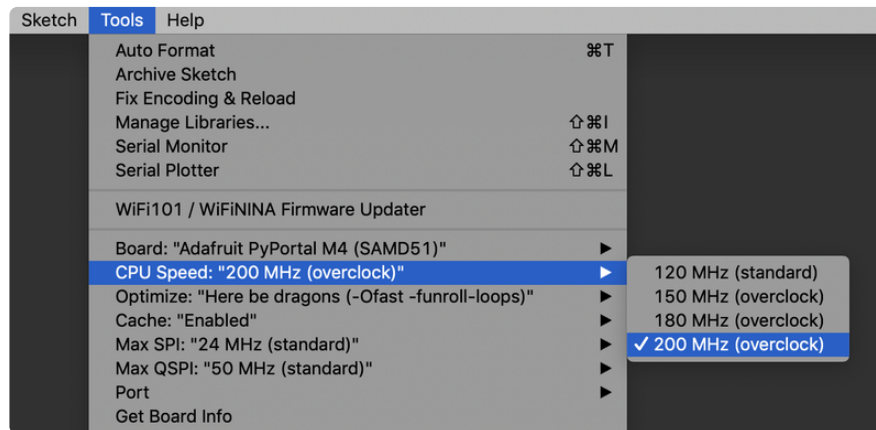
Pretty-Printing out registers

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

<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

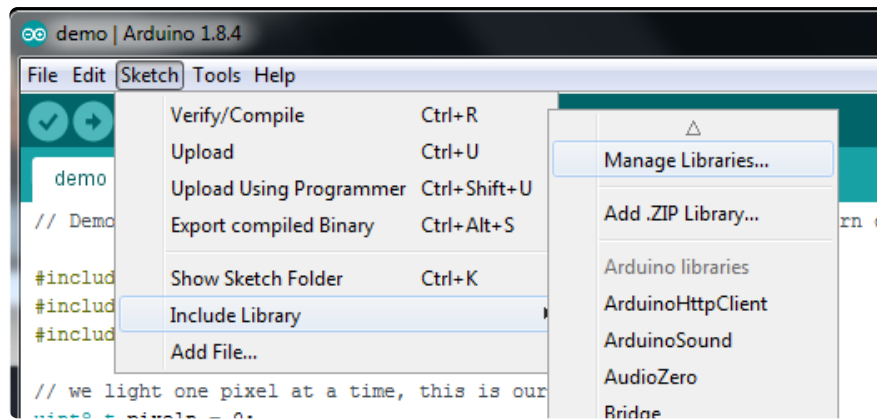
If you want to reduce power draw, some of our boards have an inductor so you can use the 1.8V buck converter instead of the built in linear regulator. If the board does have an inductor (see the schematic) you can add the line `SUPC->VREG.bit.SEL = 1;` to your code to switch to it. Note it will make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

Using SPI Flash

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

To use the flash memory with Arduino you'll need to install the [Adafruit SPI Flash Memory library \(https://adafru.it/wbt\)](https://adafru.it/wbt) in the Arduino IDE.

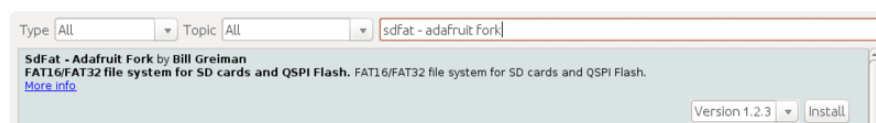
Open up the Arduino library manager



Search for the Adafruit SPIFlash library and install it



Search for the SdFat - Adafruit Fork library and install it



We also have a great tutorial on Arduino library installation at:

<http://learn.adafruit.com/adafruit-all-about-arduino-libraries-install-use> (<https://adafru.it/aYM>)

Once the library is installed look for the following examples in the library:

- fatfs_circuitpython
- fatfs_datalogging
- fatfs_format
- fatfs_full_usage
- fatfs_print_file
- flash_erase

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

Read & Write CircuitPython Files

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

Note that before you use the `fatfs_circuitpython` example you must have loaded CircuitPython on your board. [Load the latest version of CircuitPython as explained in this guide \(https://adafru.it/BeN\)](https://adafru.it/BeN) first to ensure a CircuitPython filesystem is initialized and written to the flash chip. Once you've loaded CircuitPython then you can run the `fatfs_circuitpython` example sketch.

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

To understand how to read & write files that are compatible with CircuitPython let's examine the sketch code. First notice an instance of the `Adafruit_M0_Express_CircuitPython` class is created and passed an instance of the flash chip class in the last line below:

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

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

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

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

By using this `Adafruit_M0_Express_CircuitPython` class you'll get a filesystem object that is compatible with reading and writing files on a CircuitPython-formatted flash chip. This is very important for interoperability between CircuitPython and Arduino as CircuitPython has specialized partitioning and flash memory layout that isn't compatible with simpler uses of the library (shown in the other examples).

Once an instance of the `Adafruit_M0_Express_CircuitPython` class is created (called `pythonfs` in this sketch) you can go on to interact with it just like if it were the [SD card library in Arduino \(https://adafru.it/wbw\)](https://adafru.it/wbw). You can open files for reading & writing, create directories, delete files and directories and more. Here's how the sketch checks if a `boot.py` file exists and prints it out a character at a time:

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

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

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

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

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

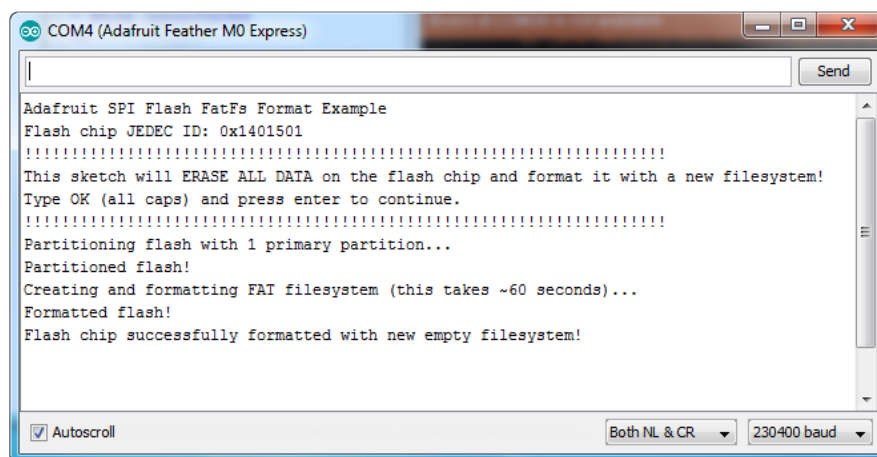
That's all there is to basic file reading and writing. Check out the `fatfs_full_usage` example for examples of even more functions like creating directories, deleting files & directories, checking the size of files, and more! Remember though to interact with CircuitPython files you need to use the `Adafruit_Feather_M0_CircuitPython` class as shown in the `fatfs_circuitpython` example above!

Format Flash Memory

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

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

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



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

Once confirmed the sketch will format the flash memory. The format process takes about a minute so be patient as the data is erased and formatted. You should see a

message printed once the format process is complete. At this point the flash chip will be ready to use with a brand new empty filesystem.

Datalogging Example

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

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

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

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

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

Reading and Printing Files

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

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

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

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

After you open a file for reading you can easily check if data is available by calling the `available` function on the file, and then read a single character with the `read` function.

This makes it easy to loop through all of the data in a file by checking if it's available and reading a character at a time. However there are more advanced read functions you can use too--see the `fatfs_full_usage` example or even the [Arduino SD library documentation](https://adafruit.it/ucu) (<https://adafruit.it/ucu>) (the SPI flash library implements the same functions).

Full Usage Example

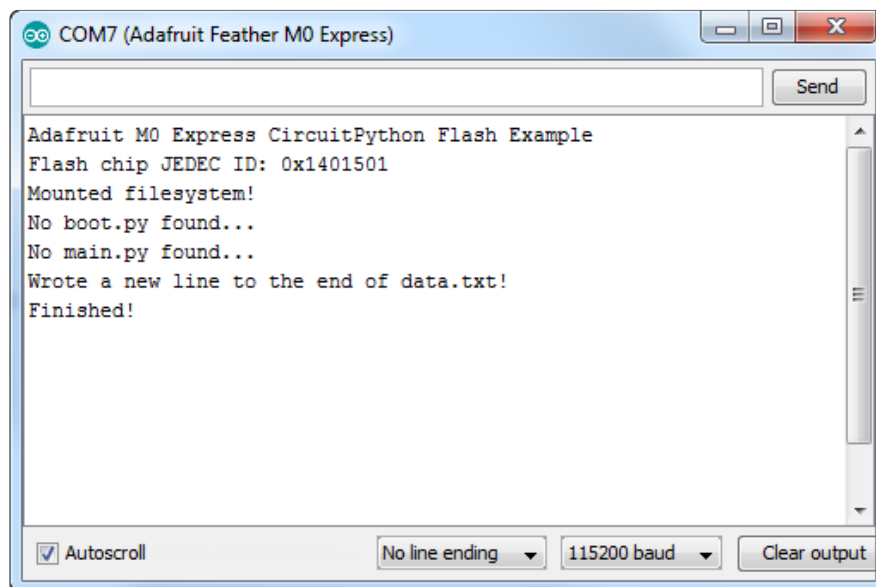
For a more complete demonstration of reading and writing files look at the `fatfs_full_usage` example. This examples uses every function in the library and demonstrates things like checking for the existence of a file, creating directories, deleting files, deleting directories, and more.

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

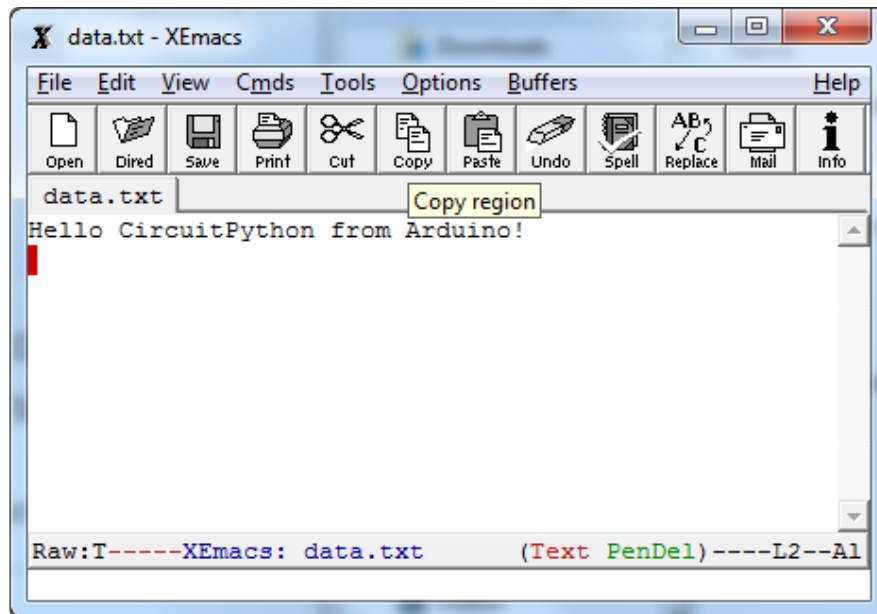
Accessing SPI Flash

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

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



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



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

Feather HELP!

Even though this FAQ is labeled for Feather, the questions apply to ItsyBitsy's as well!

My ItsyBitsy/Feather stopped working when I unplugged the USB!

A lot of our example sketches have a

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

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

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

A vast number of Itsy/Feather 'failures' are due to charge-only USB cables

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

Use only a cable that you know is for data syncing

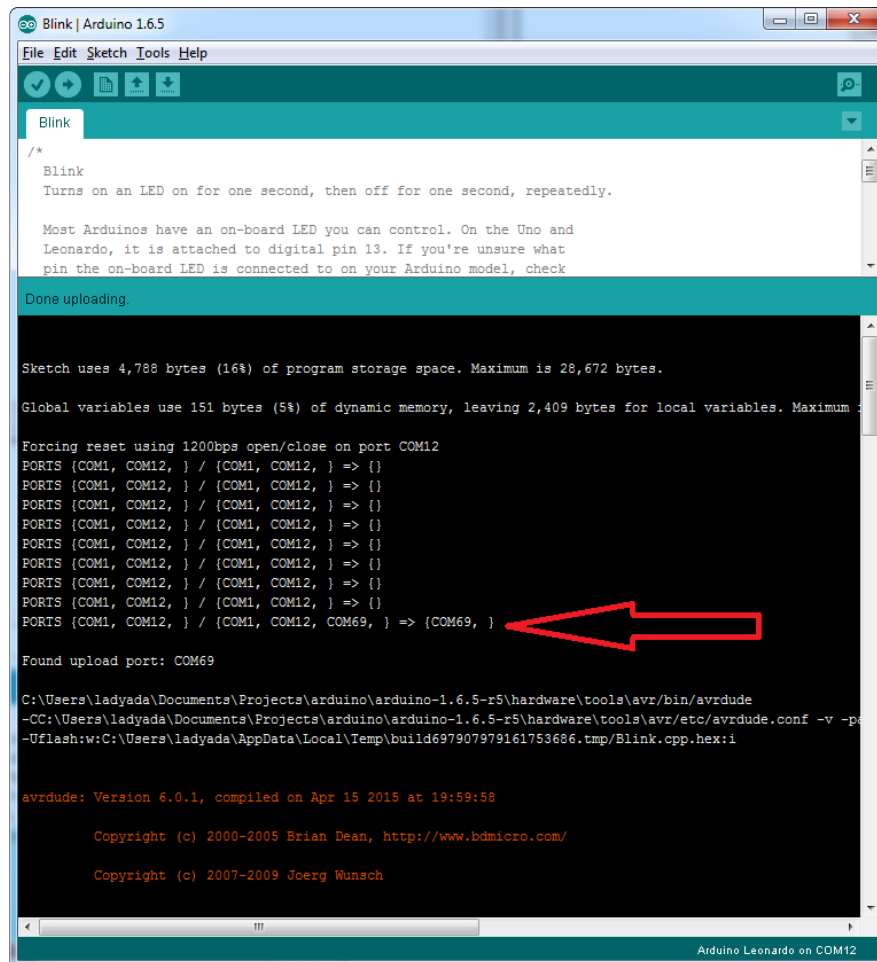
If you have any charge-only cables, cut them in half throw them out. We are serious! They tend to be low quality in general, and will only confuse you and others later, just get a good data+charge USB cable.

A quality USB port is critical. Avoid plugging into USB keyboards and when possible use a USB-2 HUB to avoid USB3 issues.

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

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

1. Turn on verbose upload in the Arduino IDE preferences
 2. Plug in Itsy or Feather 32u4/M0, it won't show up as a COM/serial port that's ok
 3. Open up the Blink example (Examples->Basics->Blink)
 4. Select the correct board in the Tools menu, e.g. Feather 32u4, Feather M0, Itsy 32u4 or M0 (physically check your board to make sure you have the right one selected!)
 5. Compile it (make sure that works)
 6. Click Upload to attempt to upload the code
 7. The IDE will print out a bunch of COM Ports as it tries to upload. During this time, double-click the reset button, you'll see the red pulsing LED that tells you its now in bootloading mode
 8. The board will show up as the Bootloader COM/Serial port
 9. The IDE should see the bootloader COM/Serial port and upload properly
-



I can't get the Itsy/Feather USB device to show up - I get "USB Device Malfunctioning" errors!

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

If you have a Feather 32u4 (look on the board to read what it is you have) Make sure you select Feather 32u4 for ATmega32u4 based boards! Do not use anything else, do not use the 32u4 breakout board line.

If you have a Feather M0 (look on the board to read what it is you have) Make sure you select Feather M0 - do not use 32u4 or Arduino Zero

If you have a ItsyBitsy M0 (look on the board to read what it is you have) Make sure you select ItsyBitsy M0 - do not use 32u4 or Arduino Zero

I'm having problems with COM ports and my Itsy/Feather 32u4/M0

There's two COM ports you can have with the 32u4/M0, one is the user port and one is the bootloader port. They are not the same COM port number!

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

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

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

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

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

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

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

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

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

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

I'm trying to upload to my Feather M0, and I get this error
"Connecting to programmer: .avrdude: butterfly_recv():
programmer is not responding"

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

I'm trying to upload to my Feather and i get this error
"avrdude: ser_recv(): programmer is not responding"

You probably don't have Feather M0 / Feather 32u4 selected in the boards drop-down. Make sure you selected Feather M0 (or Feather 32u4).

I attached some wings to my Feather and now I can't read the battery voltage!

Make sure your Wing doesn't use pin #9 which is the analog sense for the lipo battery!

The yellow LED Is flickering on my Feather, but no battery is plugged in, why is that?

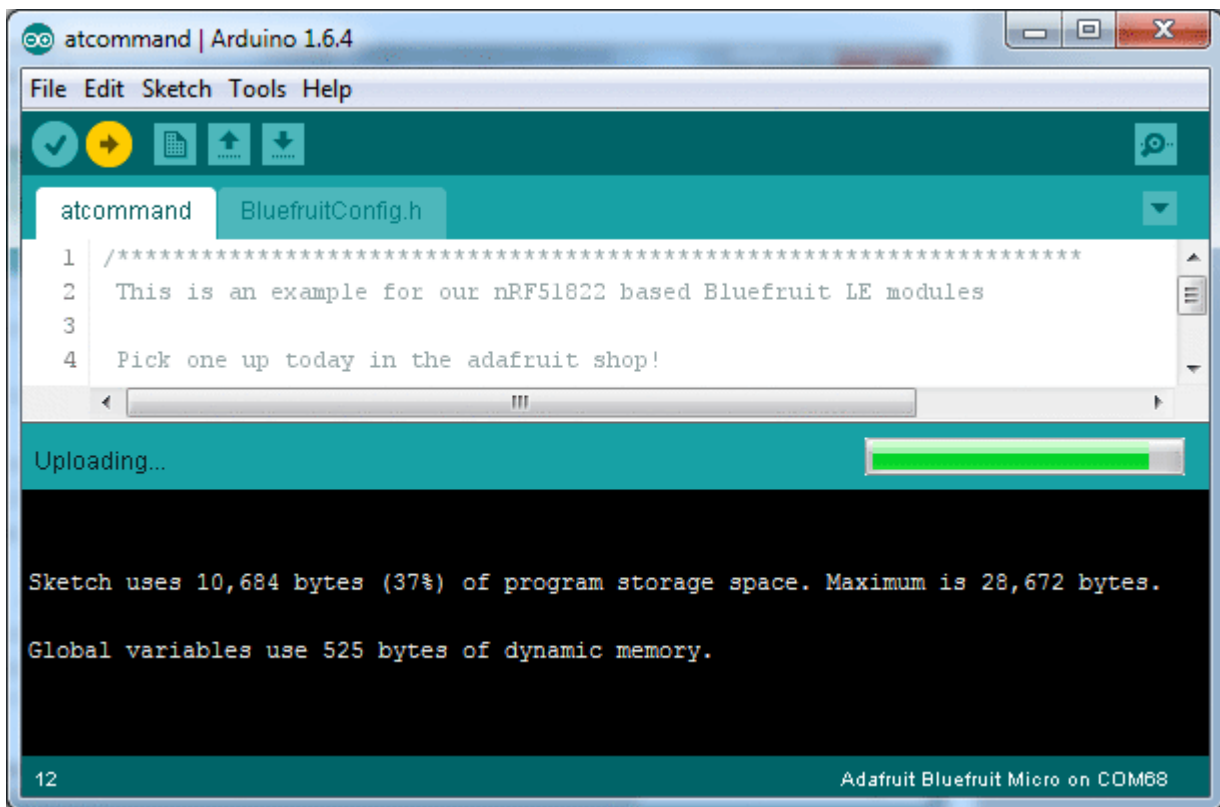
The charge LED is automatically driven by the Lipoly charger circuit. It will try to detect a battery and is expecting one to be attached. If there isn't one it may flicker once in a while when you use power because it's trying to charge a (non-existent) battery.

It's not harmful, and its totally normal!

The Arduino IDE gives me "Device Descriptor Request Failed"

This can require "manual 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, double-click the RST button to get back into the bootloader. The red LED will pulse, so you know that its in bootloader mode. Do the reset button double-press right as the Arduino IDE says its attempting to upload the sketch, when you see the Yellow Arrow lit and the Uploading... text in the status bar.



(h

<https://adafru.it/UJA>

Don't click the reset button before uploading, unlike other bootloaders you want this one to run at the time Arduino is trying to upload

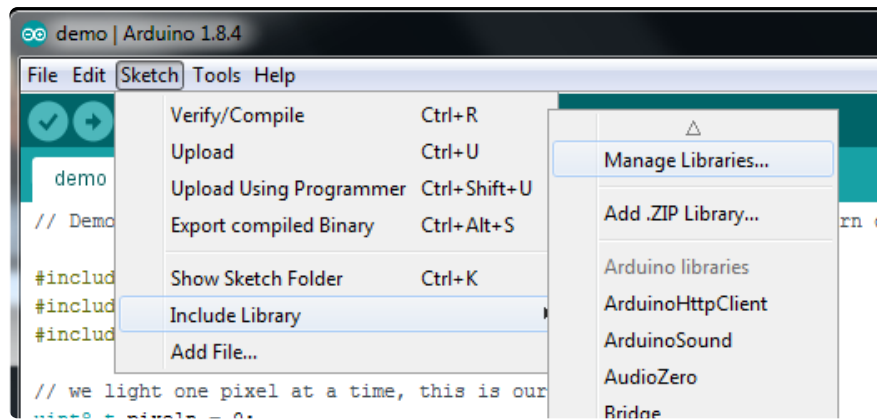
Arcada Libraries

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

There's a lot of libraries!

Install Libraries

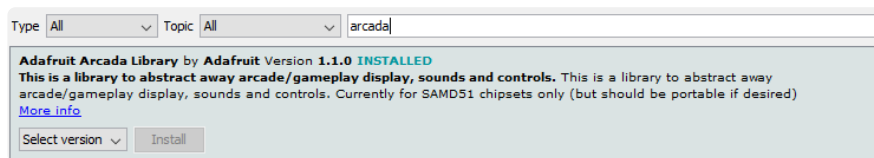
Open up the library manager...



And install the following libraries:

Adafruit Arcada

This library generalizes the hardware for you so you can read the joystick, draw to the display, read files, etc. without having to worry about the underlying methods

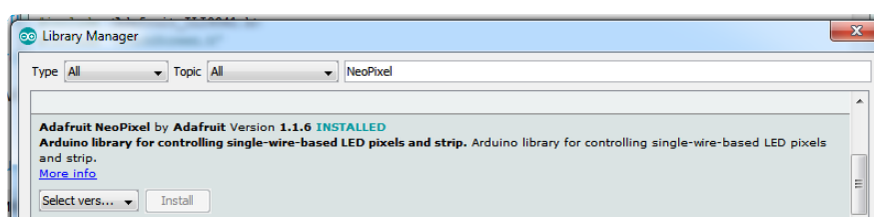


If you use Arduino 1.8.10 or later, the IDE will automatically install all the libraries you need to run all the Arcada demos when you install Arcada. We strongly recommend using the latest IDE so you don't miss one of the libraries!

If you aren't running Arduino IDE 1.8.10 or later, you'll need to install all of the following!

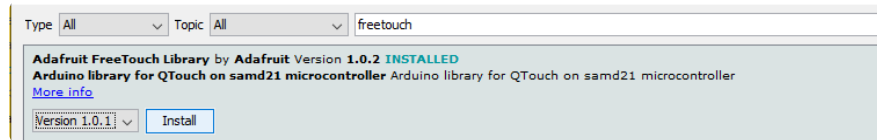
Adafruit NeoPixel

This will let you light up the status LEDs on the front/back



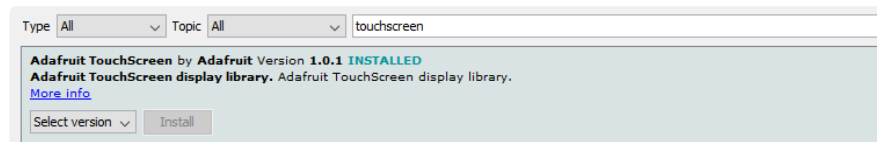
Adafruit FreeTouch

This is the open source version of QTouch for SAMD21 boards



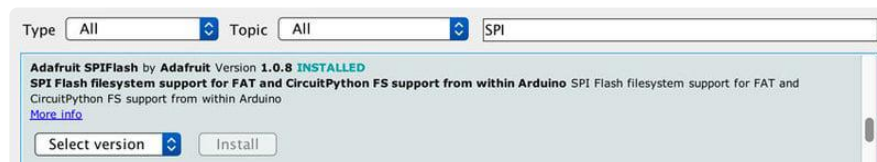
Adafruit Touchscreen

Used by Adafruit Arcada for touchscreen input (required even if your Arcada board does not have a touchscreen)



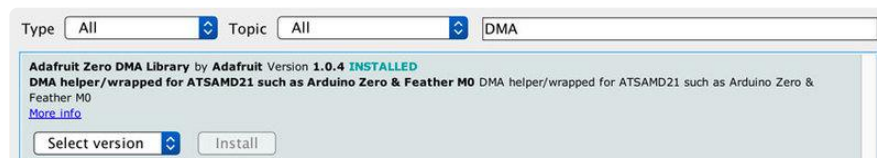
Adafruit SPIFlash

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



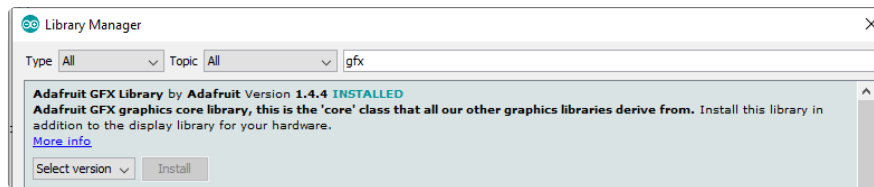
Adafruit Zero DMA

This is used by the Graphics Library if you choose to use DMA



Adafruit GFX

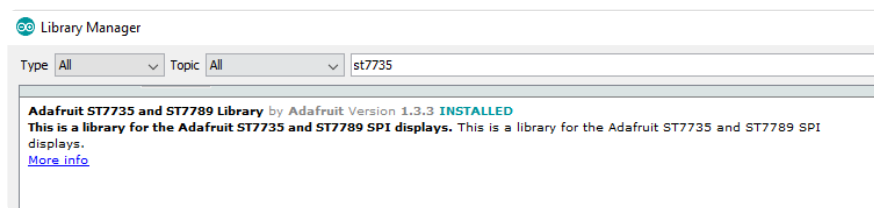
This is the graphics library used to draw to the screen



If using an older (pre-1.8.10) Arduino IDE, locate and install Adafruit_BusIO (newer versions do this one automatically).

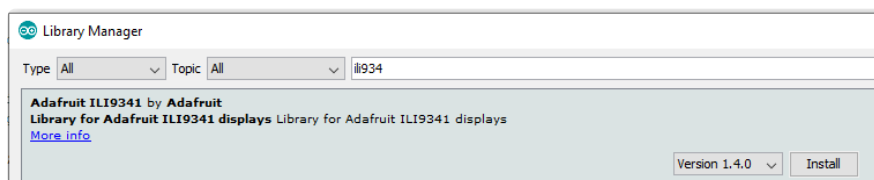
Adafruit ST7735

The display on the PyBadge/PyGamer & other Arcada boards



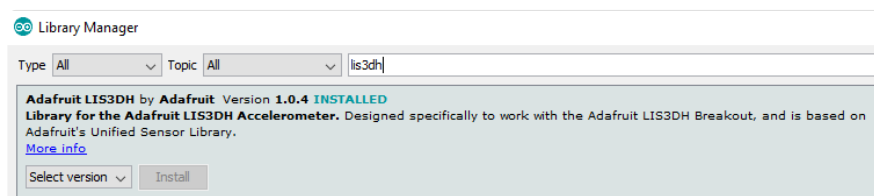
Adafruit ILI9341

The display on the PyPortal & other Arcada boards



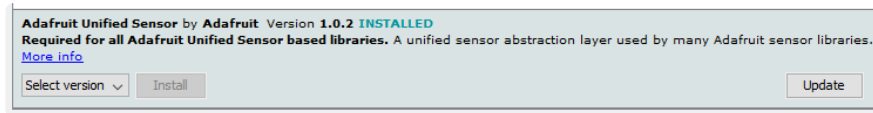
Adafruit LIS3DH

For reading the accelerometer data, required even if one is not on the board



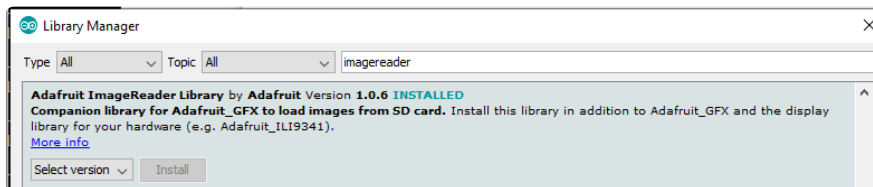
Adafruit Sensor

Needed by the LIS3DH Library, required even if one is not on the board



Adafruit ImageReader

For reading bitmaps from SPI Flash or SD and displaying



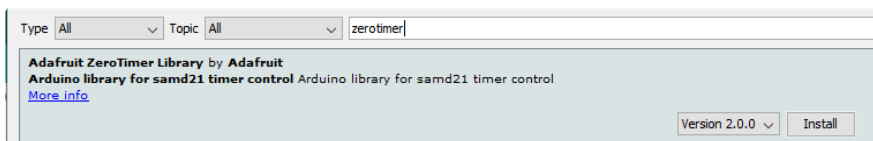
ArduinoJson

We use this library to read and write configuration files



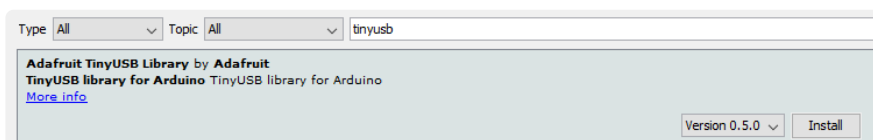
Adafruit ZeroTimer

We use this library to easily set timers and callbacks on the SAMD processors



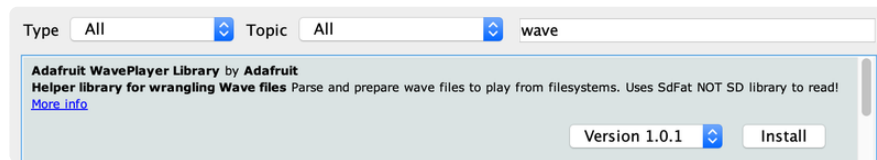
Adafruit TinyUSB

This lets us do cool stuff with USB like show up as a Keyboard or Disk Drive



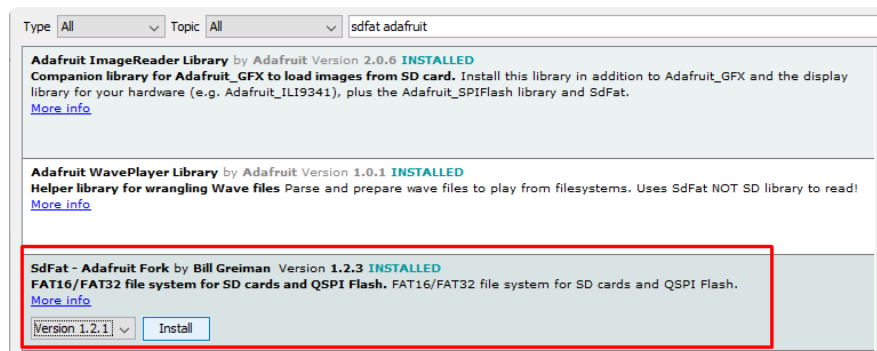
Adafruit WavePlayer

Helps us play .WAV sound files.



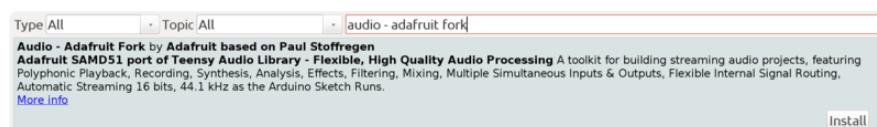
SdFat (Adafruit Fork)

The Adafruit fork of the really excellent SD card library that gives a lot more capability than the default SD library



Audio - Adafruit Fork

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



Using the TFT

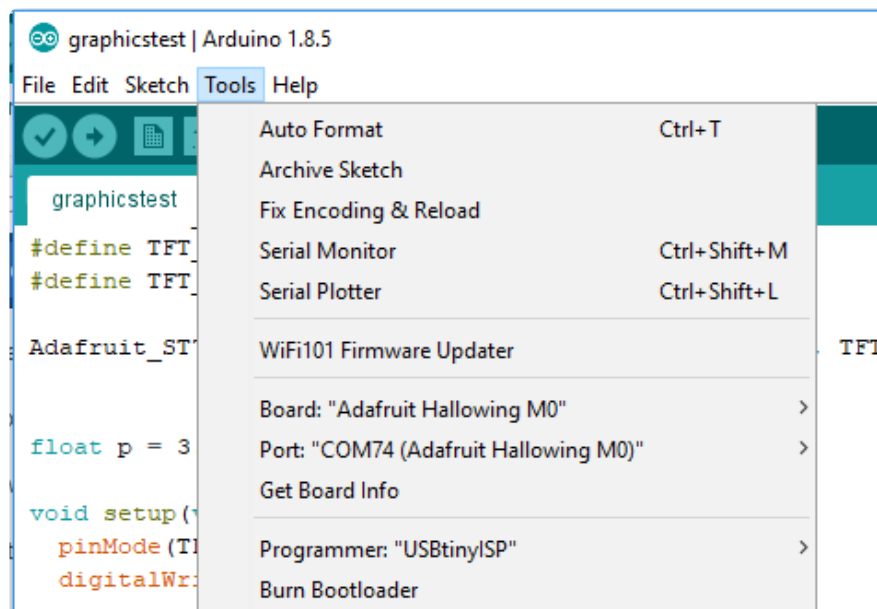
We've got libraries for using the TFT in Arduino, so its easy to get drawing!

Install Libraries

From the previous page, don't forget to install all those libraries!

Setup

Make sure you've got the Hallowing M0 selected as the board type, and the correct port as well



Now load up the example at the bottom of this page in a Arduino sketch. Its an adaptation of our standard graphics test, but you'll note we add some lines that are specific to the Hallowing.

First, these are the pins that are connected:

```
// These are 'hard wired'
#define TFT_CS      39
#define TFT_RST     37
#define TFT_DC      38
#define TFT_BACKLIGHT 7
```

In `setup()` don't forget to turn the backlight on!

```
void setup(void) {
  pinMode(TFT_BACKLIGHT, OUTPUT);
  digitalWrite(TFT_BACKLIGHT, HIGH);
}
```

You can dim the backlight with `analogWrite(TFT_BACKLIGHT, brightness)` where brightness is between 0 (off) and 255 (full on)

Also, rotate the display:


```
tft.setRotation(2);
```

Once you have it all working, you can visit the [Adafruit GFX library guide page \(https://adafruit.it/doL\)](https://adafruit.it/doL) to learn more about all the different shapes you can draw!

Graphics Test Code

```
/*
*****
This is a library for several Adafruit displays based on ST77* drivers.

Works with the HalloWing M0 Express
----> http://www.adafruit.com/products/3900

Check out the links above for our tutorials and wiring diagrams.

Adafruit invests time and resources providing this open source code,
please support Adafruit and open-source hardware by purchasing
products from Adafruit!

Written by Limor Fried/Ladyada for Adafruit Industries.
MIT license, all text above must be included in any redistribution
*****/

#include <Adafruit_GFX.h>    // Core graphics library
#include <Adafruit_ST7735.h> // Hardware-specific library for ST7735
#include <SPI.h>

#define TFT_CS      39 // Hallowing display control pins: chip select
#define TFT_RST     37 // Display reset
#define TFT_DC      38 // Display data/command select
#define TFT_BACKLIGHT 7 // Display backlight pin

// OPTION 1 (recommended) is to use the HARDWARE SPI pins, which are unique
// to each board and not reassignable.
Adafruit_ST7735 tft = Adafruit_ST7735(TFT_CS, TFT_DC, TFT_RST);

// OPTION 2 lets you interface the display using ANY TWO or THREE PINS,
// tradeoff being that performance is not as fast as hardware SPI above.
// #define TFT_MOSI 29 // Data out
// #define TFT_SCLK 30 // Clock out
// Adafruit_ST7735 tft = Adafruit_ST7735(TFT_CS, TFT_DC, TFT_MOSI, TFT_SCLK,
// TFT_RST);

float p = 3.1415926;

void setup(void) {
  Serial.begin(9600);
  Serial.print(F("Hello! ST77xx TFT Test"));

  tft.initR(INITR_HALLOWING); // Initialize HalloWing-oriented screen
  pinMode(TFT_BACKLIGHT, OUTPUT);
  digitalWrite(TFT_BACKLIGHT, HIGH); // Backlight on

  Serial.println(F("Initialized"));

  uint16_t time = millis();
  tft.fillScreen(ST77XX_BLACK);
  time = millis() - time;

  Serial.println(time, DEC);
  delay(500);
}
```

```

// large block of text
tft.fillScreen(ST77XX_BLACK);
testdrawtext("Lorem ipsum dolor sit amet, consectetur adipiscing elit. Curabitur
adipiscing ante sed nibh tincidunt feugiat. Maecenas enim massa, fringilla sed
malesuada et, malesuada sit amet turpis. Sed porttitor neque ut ante pretium vitae
malesuada nunc bibendum. Nullam aliquet ultrices massa eu hendrerit. Ut sed nisi
lorem. In vestibulum purus a tortor imperdiet posuere. ", ST77XX_WHITE);
delay(1000);

// tft print function!
tftPrintTest();
delay(4000);

// a single pixel
tft.drawPixel(tft.width()/2, tft.height()/2, ST77XX_GREEN);
delay(500);

// line draw test
testlines(ST77XX_YELLOW);
delay(500);

// optimized lines
testfastlines(ST77XX_RED, ST77XX_BLUE);
delay(500);

testdrawrects(ST77XX_GREEN);
delay(500);

testfillrects(ST77XX_YELLOW, ST77XX_MAGENTA);
delay(500);

tft.fillScreen(ST77XX_BLACK);
testfillcircles(10, ST77XX_BLUE);
testdrawcircles(10, ST77XX_WHITE);
delay(500);

testroundrects();
delay(500);

testtriangles();
delay(500);

mediabuttons();
delay(500);

Serial.println("done");
delay(1000);
}

void loop() {
  tft.invertDisplay(true);
  delay(500);
  tft.invertDisplay(false);
  delay(500);
}

void testlines(uint16_t color) {
  tft.fillScreen(ST77XX_BLACK);
  for (int16_t x=0; x < tft.width(); x+=6) {
    tft.drawLine(0, 0, x, tft.height()-1, color);
    delay(0);
  }
  for (int16_t y=0; y < tft.height(); y+=6) {
    tft.drawLine(0, 0, tft.width()-1, y, color);
    delay(0);
  }
}

tft.fillScreen(ST77XX_BLACK);
for (int16_t x=0; x < tft.width(); x+=6) {

```

```

    tft.drawLine(tft.width()-1, 0, x, tft.height()-1, color);
    delay(0);
}
for (int16_t y=0; y < tft.height(); y+=6) {
    tft.drawLine(tft.width()-1, 0, 0, y, color);
    delay(0);
}

tft.fillScreen(ST77XX_BLACK);
for (int16_t x=0; x < tft.width(); x+=6) {
    tft.drawLine(0, tft.height()-1, x, 0, color);
    delay(0);
}
for (int16_t y=0; y < tft.height(); y+=6) {
    tft.drawLine(0, tft.height()-1, tft.width()-1, y, color);
    delay(0);
}

tft.fillScreen(ST77XX_BLACK);
for (int16_t x=0; x < tft.width(); x+=6) {
    tft.drawLine(tft.width()-1, tft.height()-1, x, 0, color);
    delay(0);
}
for (int16_t y=0; y < tft.height(); y+=6) {
    tft.drawLine(tft.width()-1, tft.height()-1, 0, y, color);
    delay(0);
}
}

void testdrawtext(char *text, uint16_t color) {
    tft.setCursor(0, 0);
    tft.setTextColor(color);
    tft.setTextWrap(true);
    tft.print(text);
}

void testfastlines(uint16_t color1, uint16_t color2) {
    tft.fillScreen(ST77XX_BLACK);
    for (int16_t y=0; y < tft.height(); y+=5) {
        tft.drawFastHLine(0, y, tft.width(), color1);
    }
    for (int16_t x=0; x < tft.width(); x+=5) {
        tft.drawFastVLine(x, 0, tft.height(), color2);
    }
}

void testdrawrects(uint16_t color) {
    tft.fillScreen(ST77XX_BLACK);
    for (int16_t x=0; x < tft.width(); x+=6) {
        tft.drawRect(tft.width()/2 -x/2, tft.height()/2 -x/2, x, x, color);
    }
}

void testfillrects(uint16_t color1, uint16_t color2) {
    tft.fillScreen(ST77XX_BLACK);
    for (int16_t x=tft.width()-1; x > 6; x-=6) {
        tft.fillRect(tft.width()/2 -x/2, tft.height()/2 -x/2, x, x, color1);
        tft.drawRect(tft.width()/2 -x/2, tft.height()/2 -x/2, x, x, color2);
    }
}

void testfillcircles(uint8_t radius, uint16_t color) {
    for (int16_t x=radius; x < tft.width(); x+=radius*2) {
        for (int16_t y=radius; y < tft.height(); y+=radius*2) {
            tft.fillCircle(x, y, radius, color);
        }
    }
}
}

```

```

void testdrawcircles(uint8_t radius, uint16_t color) {
    for (int16_t x=0; x < tft.width()+radius; x+=radius*2) {
        for (int16_t y=0; y < tft.height()+radius; y+=radius*2) {
            tft.drawCircle(x, y, radius, color);
        }
    }
}

void testtriangles() {
    tft.fillScreen(ST77XX_BLACK);
    uint16_t color = 0xF800;
    int t;
    int w = tft.width()/2;
    int x = tft.height()-1;
    int y = 0;
    int z = tft.width();
    for(t = 0 ; t <= 15; t++) {
        tft.drawTriangle(w, y, y, x, z, x, color);
        x-=4;
        y+=4;
        z-=4;
        color+=100;
    }
}

void testroundrects() {
    tft.fillScreen(ST77XX_BLACK);
    uint16_t color = 100;
    int i;
    int t;
    for(t = 0 ; t <= 4; t+=1) {
        int x = 0;
        int y = 0;
        int w = tft.width()-2;
        int h = tft.height()-2;
        for(i = 0 ; i <= 16; i+=1) {
            tft.drawRoundRect(x, y, w, h, 5, color);
            x+=2;
            y+=3;
            w-=4;
            h-=6;
            color+=1100;
        }
        color+=100;
    }
}

void tftPrintTest() {
    tft.setTextWrap(false);
    tft.fillScreen(ST77XX_BLACK);
    tft.setCursor(0, 30);
    tft.setTextColor(ST77XX_RED);
    tft.setTextSize(1);
    tft.println("Hello World!");
    tft.setTextColor(ST77XX_YELLOW);
    tft.setTextSize(2);
    tft.println("Hello World!");
    tft.setTextColor(ST77XX_GREEN);
    tft.setTextSize(3);
    tft.println("Hello World!");
    tft.setTextColor(ST77XX_BLUE);
    tft.setTextSize(4);
    tft.print(1234.567);
    delay(1500);
    tft.setCursor(0, 0);
    tft.fillScreen(ST77XX_BLACK);
    tft.setTextColor(ST77XX_WHITE);
    tft.setTextSize(0);
    tft.println("Hello World!");
}

```

```

tft.setTextSize(1);
tft.setTextColor(ST77XX_GREEN);
tft.print(p, 6);
tft.println(" Want pi?");
tft.println(" ");
tft.print(8675309, HEX); // print 8,675,309 out in HEX!
tft.println(" Print HEX!");
tft.println(" ");
tft.setTextColor(ST77XX_WHITE);
tft.println("Sketch has been");
tft.println("running for: ");
tft.setTextColor(ST77XX_MAGENTA);
tft.print(millis() / 1000);
tft.setTextColor(ST77XX_WHITE);
tft.print(" seconds.");
}

void mediabuttons() {
  // play
  tft.fillScreen(ST77XX_BLACK);
  tft.fillRoundRect(25, 10, 78, 60, 8, ST77XX_WHITE);
  tft.fillTriangle(42, 20, 42, 60, 90, 40, ST77XX_RED);
  delay(500);
  // pause
  tft.fillRoundRect(25, 90, 78, 60, 8, ST77XX_WHITE);
  tft.fillRoundRect(39, 98, 20, 45, 5, ST77XX_GREEN);
  tft.fillRoundRect(69, 98, 20, 45, 5, ST77XX_GREEN);
  delay(500);
  // play color
  tft.fillTriangle(42, 20, 42, 60, 90, 40, ST77XX_BLUE);
  delay(50);
  // pause color
  tft.fillRoundRect(39, 98, 20, 45, 5, ST77XX_RED);
  tft.fillRoundRect(69, 98, 20, 45, 5, ST77XX_RED);
  // play color
  tft.fillTriangle(42, 20, 42, 60, 90, 40, ST77XX_GREEN);
}

```

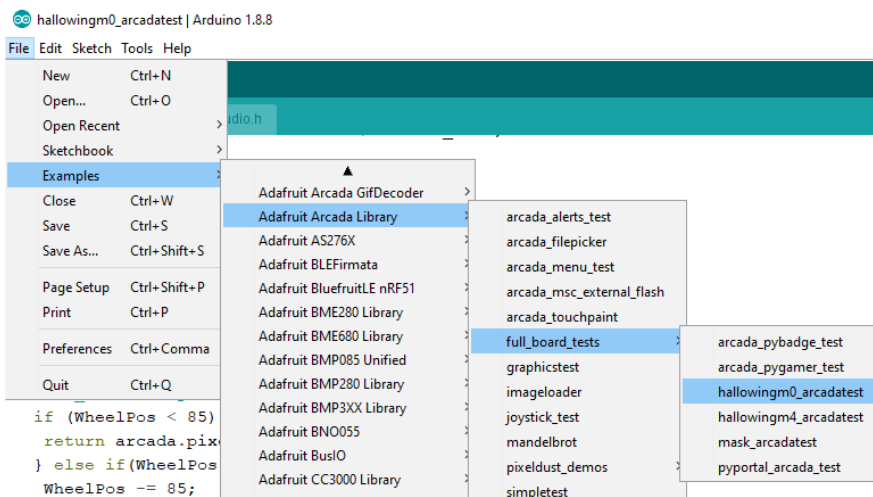
Full Test Sketch

Once you have the TFT working, you can now continue onto a full test sketch, which will show how to read the light sensor, accelerometer, capacitive touch, and even the battery voltage! We do this all thru the Arcada library, which allows us to abstract a wide range of boards, from the PyBadge, to the Hallowing, so you don't have to worry about pin numbers or exact display parts.

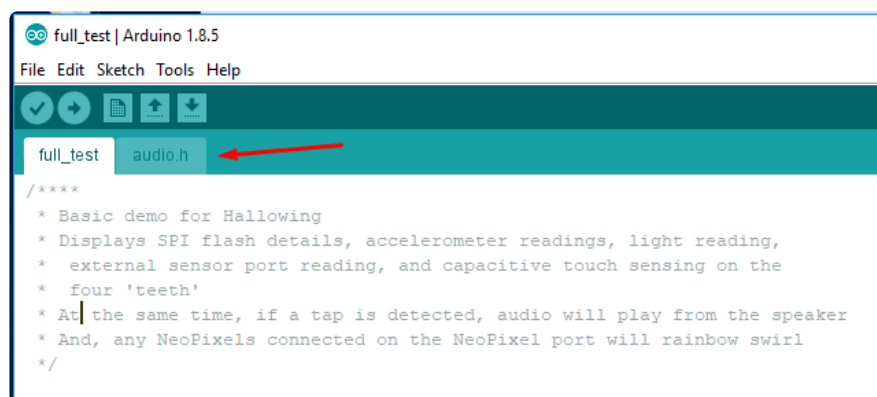
Install Libraries

[Don't forget to install all of the Arcada libraries! \(https://adafru.it/FV4\)](https://adafru.it/FV4)

Whew, once all are installed. You can load up the example code, which is in the Arcada Library -> Full Board Tests -> hallowingm0_arcadatest



Make sure once you've loaded it in, you have both the full_test code and the additional audio.h code



Upload and try it out! If you need to quickly test a board, here's the UF2 you can load on

hallowingtest.UF2

<https://adafru.it/FV5>

Arcada Library

This is a quickstart explanation of what Adafruit Arcada library provides, see the detailed Doxygen documents for arguments & return values

Initialization

- `arcadaBegin()` must be called first, it will set pin directions, turn off NeoPixels, and check for connected hardware

- `filesystemBeginMSD()` will initialize the storage method (SD or SPI flash) and check if a proper filesystem exists. On SD cards that's a FAT filesystem (so make sure its formatted). On SPI Flash we use CircuitPython's FAT filesystem, the best way to format is to load CircuitPython on once. If you're using TinyUSB as your USB stack, this will also make the disk drive appear on a computer
- `displayBegin()` initializes the display, you will need to turn on the backlight after this is done - we don't do it for you!

Joystick & Buttons

- `readJoystickX` and `readJoystickY` read the analog joystick (if there is one) and returns -512 to 511 with 0 being 'center' (approximately)
- `readButtons` returns a 32 bit mask for each button pressed at the moment of the function call - right now only the bottom 8 bits are used. Check `Adafruit_Arcada_Def.h` for the button mask names. Analog joysticks are checked against a threshold and 'emulate' a button press

Some boards, like the MONSTER M4SK and HalloWings, do not have a proper joystick - instead we will return the capacitive touch pads or buttons as if there was a joystick. For example, the M4SK's three buttons will return 'up', 'A' and 'down' respectively.

- After `readButtons` is called, `justPressedButtons` will tell you buttons were pressed as of the `readButtons` call
- Ditto for `justReleasedButtons`

Backlight, Speaker and Sensors

- Enable/disable speaker amplifier (if there is one) with `enableSpeaker` - this doesn't affect headphones if there are any
- `readBatterySensor` returns the battery voltage detected. You cannot detect whether a battery is being charged, only the voltage.
- `readLightSensor` will return 0 for dark, 1023 for bright surrounding light.
- `setBacklight` can set the backlight from 0 (off) to 255 (all the way on)

Alert Boxes

These info boxes and alert display on the screen to let the user know something they need to do, get ready for, or went wrong. You can have the alert wait for a button

press or have it return immediately (then you can delay or wait for something else to occur)

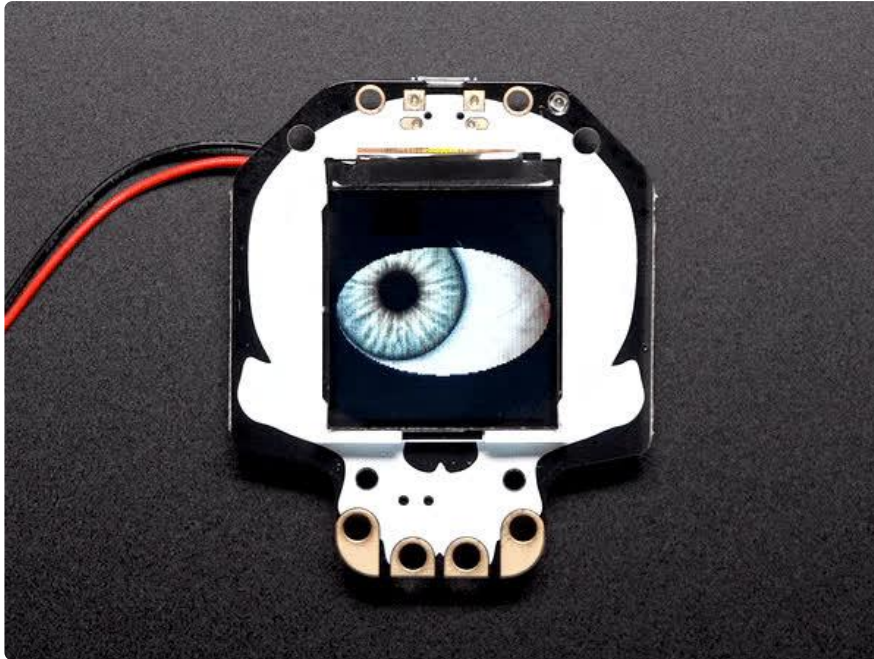
- `alertBox` is the generic, you can set the message, box and text color, as well as button press
- `infoBox` is an `alertBox` where the default button is A and the box color is white, text color is black
- `warnBox` is an `alertBox` where the default button is A and the box color is yellow, text color is black
- `errorBox` is an `alertBox` where the default button is A and the box color is red, text color is white
- `haltBox` is an `alertBox` where the box color is red, text color is white. It will sit in a busy loop and never return



Arcada Library Docs

[Arcada Library Docs \(https://adafru.it/FVvm\)](https://adafru.it/FVvm)

Spooky Eyes



Hallowing ships with a pre-loaded example of a human eye that looks around, blinks and reacts to light.

If you want to put the original spooky eye demo back on your Hallowing, [enter bootloader mode by double-clicking the Reset button \(https://adafru.it/C8r\)](https://adafru.it/C8r) and then drag this UF2 file over onto HALLOWBOOT:

Spooky_Eye_Human.UF2

<https://adafru.it/CmU>

There's a few customized variants as well, such as a fiery dragon eye:

Spooky_Eye_Dragon.UF2

<https://adafru.it/CmV>

A brown, animal-ish eye with no visible sclera:

Spooky_Eye_NoSclera.UF2

<https://adafru.it/CmW>

A psychedelic eye from our [Eye of Newt guide \(https://adafru.it/Cmd\)](https://adafru.it/Cmd):

Spooky_Eye_Newt.UF2

<https://adafru.it/CmX>

And a Terminator-inspired robotic eye:

Spooky_Eye_Terminator.UF2

<https://adafru.it/CFf>

Customizing the Spooky Eye Demo

The software controlling Hallowing's eye is extensively customizable. This requires some familiarity with the Arduino IDE and, depending on the extent of customizations you have in mind, perhaps some image editing and using Python scripts on the command line.

This is all explained in the “[Electronic Animated Eyes using Teensy 3.1/3.2 \(https://adafru.it/j6B\)](https://adafru.it/j6B)” guide (despite the name, it also works on various Adafruit “M0” and “M4” boards as well, including the Hallowing).

Most of the code there will automatically work on the Hallowing hardware, such as the display and light sensor. Other Hallowing-specific features, such as the capacitive touch pads, are not handled by the code...but could be added if you've done some Arduino programming before.

Synchronized Eyes

This page has been made into its own self-contained guide now:

[Synchronized Eyes with Two HalloWings \(https://adafru.it/CTz\)](https://adafru.it/CTz)

What is CircuitPython?

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



CircuitPython is based on Python

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

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



Why would I use CircuitPython?

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

Other reasons to use CircuitPython include:

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

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

CircuitPython

[CircuitPython \(https://adafru.it/tB7\)](https://adafru.it/tB7) is 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 flash 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!

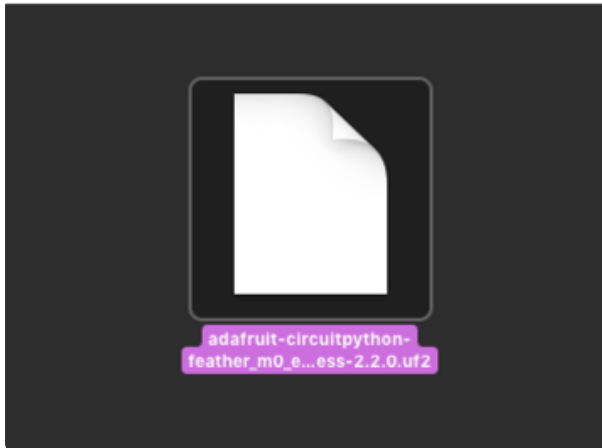
Set up CircuitPython Quick Start!

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

If you are doing a HalloWing CircuitPython that uses the TFT display, you'll want to download a different, newer version of CircuitPython that supports the display. Follow the directions in the Learn Guide you're using.

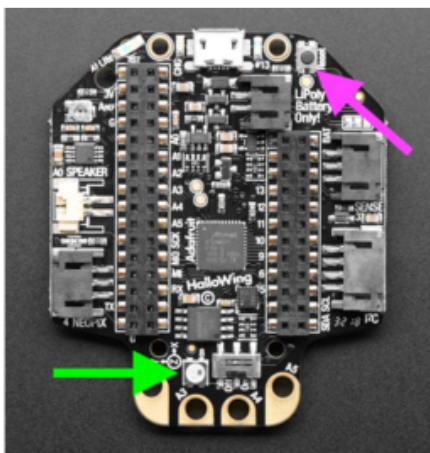
Download the latest version of
CircuitPython for this board via
CircuitPython.org

<https://adafru.it/Em7>



Click the green box above to download the latest version of CircuitPython for the HalloWing. It should be 3.0 or higher, preferably the latest version. Versions before 3.0 do not have HalloWing support.

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

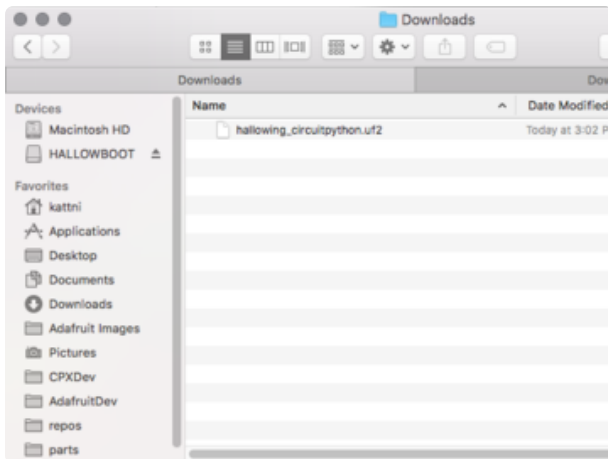


Plug your Feather M0 into your computer using a known-good USB cable.

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.

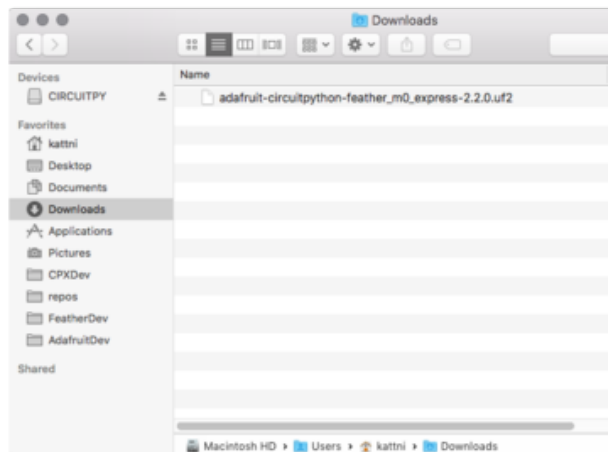
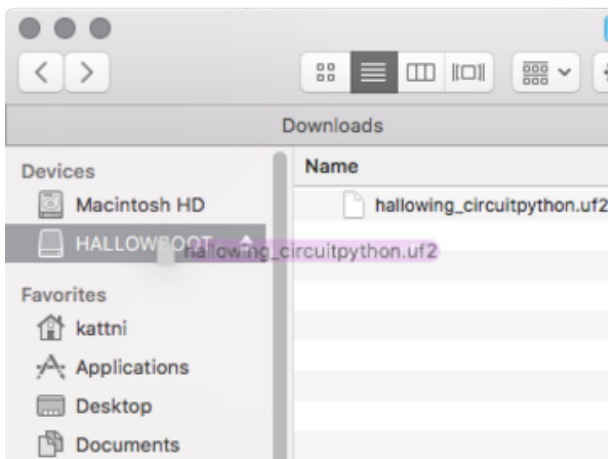
Double-click the Reset button next to the USB connector (magenta arrow) on your board, and you will see the NeoPixel RGB LED (green arrow) turn green. If it turns red, check the USB cable, try another USB port, etc. Note: The little red LED next to the USB connector will be dim red, and the little yellow LED on the opposite side will flash yellow. 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 HALLOWBOOT.

Drag the hallowing_circuitpython.uf2 file to HALLOWBOOT.



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

If you haven't added any code to your board, the only file that will be present is boot_out.txt. This is absolutely normal! It's time for you to add your code.py and get started!

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

Installing the 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



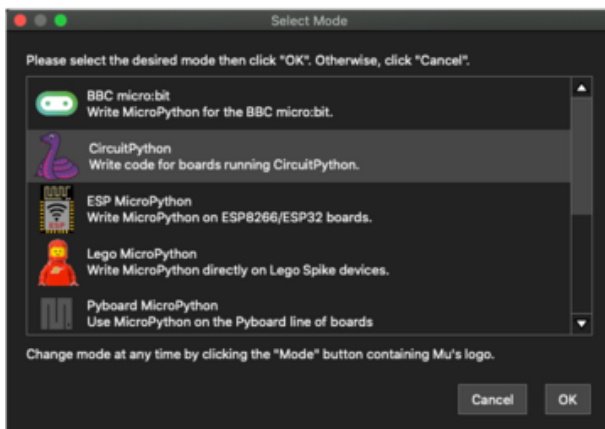
Download Mu from <https://codewith.mu> (<https://adafru.it/Be6>).

Click the Download link for downloads and installation instructions.

Click Start Here to find a wealth of other information, including extensive tutorials and how-to's.

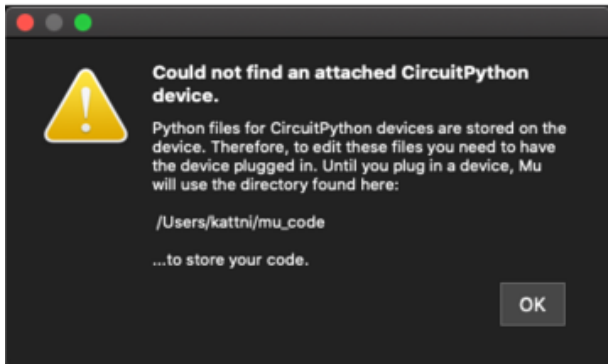
Windows users: due to the nature of MSI installers, please remove old versions of Mu before installing the latest version.

Starting Up 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 on startup, so if you do not have a CircuitPython board plugged in with a CIRCUITPY drive available, Mu will inform you where it will store any code you save until you plug in a board.

To avoid this warning, plug in a board and ensure that the CIRCUITPY drive is mounted before starting Mu.

Using 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...

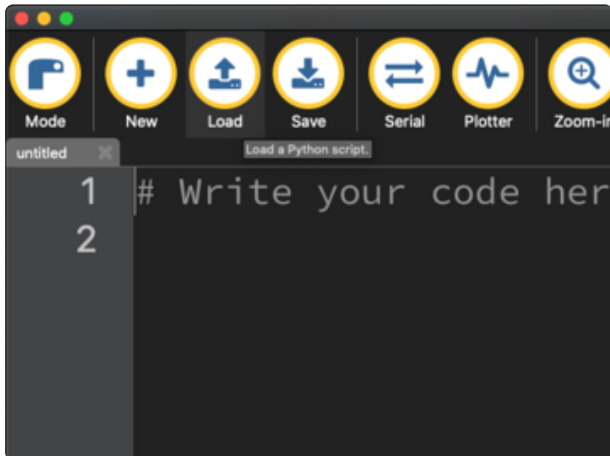
Creating and Editing Code

One of the best things about CircuitPython is how simple it is to get code up and running. This section covers 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. Adafruit strongly recommends 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 a number of other editors that work quite well. The [Recommended Editors page \(https://adafru.it/Vue\)](https://adafru.it/Vue) has more details. Otherwise, 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



Installing CircuitPython generates a code.py file on your CIRCUITPY drive. To begin your own program, open your editor, and load the code.py file from the CIRCUITPY drive.

If you are using Mu, click the Load button in the button bar, navigate to the CIRCUITPY drive, and choose code.py.

Copy and paste the following code into your editor:

```
import board
import digitalio
import time

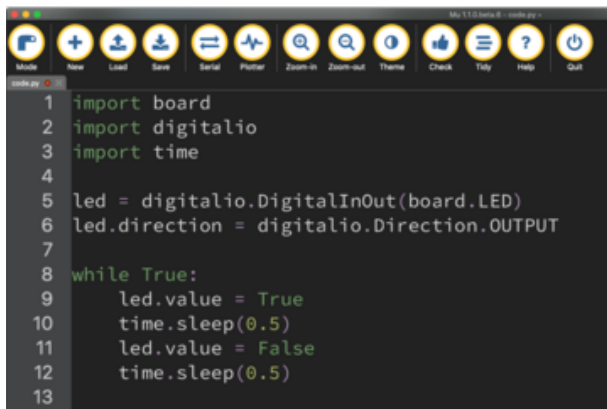
led = digitalio.DigitalInOut(board.LED)
led.direction = digitalio.Direction.OUTPUT

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

The KB2040, QT Py and the Trinkeys do not have a built-in little red LED! There is an addressable RGB NeoPixel LED. The above example will NOT work on the KB2040, QT Py or the Trinkeys!

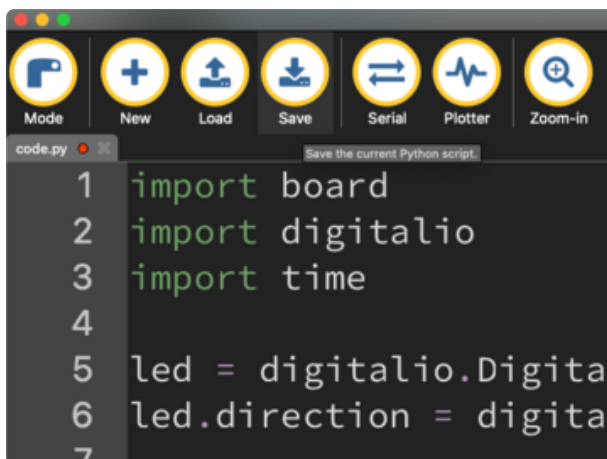
If you're using a KB2040, QT Py or a Trinkey, please download the [NeoPixel blink example \(https://adafru.it/UDU\)](https://adafru.it/UDU).

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.

A screenshot of the CircuitPython IDE interface. The top toolbar includes icons for Mode, New, Load, Save, Serial, Plotter, Zoom-in, Zoom-out, Theme, Check, Tidy, Help, and Quit. The code editor shows a Python script with the following lines:

```
1 import board
2 import digitalio
3 import time
4
5 led = digitalio.DigitalInOut(board.LED)
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)
13
```

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

A second screenshot of the CircuitPython IDE, showing the same Python script as the first image. The code is:

```
1 import board
2 import digitalio
3 import time
4
5 led = digitalio.DigitalIn
6 led.direction = digital
7
```

Save the code.py file on your CIRCUITPY drive.

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

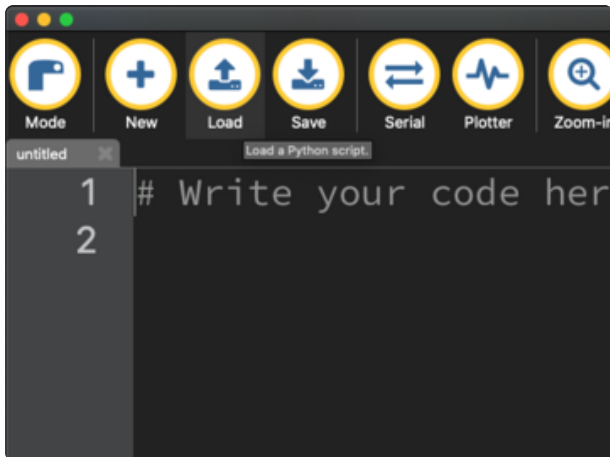
Congratulations, you've just run your first CircuitPython program!

On most boards you'll find a tiny red LED.

On the ItsyBitsy nRF52840, you'll find a tiny blue LED.

On QT Py M0, QT Py RP2040, and the Trinkey series, you will find only an RGB NeoPixel LED.

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 one warning before you continue...

Don't click reset or unplug your board!

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. 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 couple of ways to avoid filesystem corruption.

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

Check out the [Recommended Editors page \(https://adafru.it/Vue\)](https://adafru.it/Vue) for details on different editing options.

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 \(https://adafru.it/Den\)](https://adafru.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. You'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?

You 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 `code.py` is the recommended name for 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 (and Python) looks like this:

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

This line in your `code.py` would result in:

```
Hello, world!
```

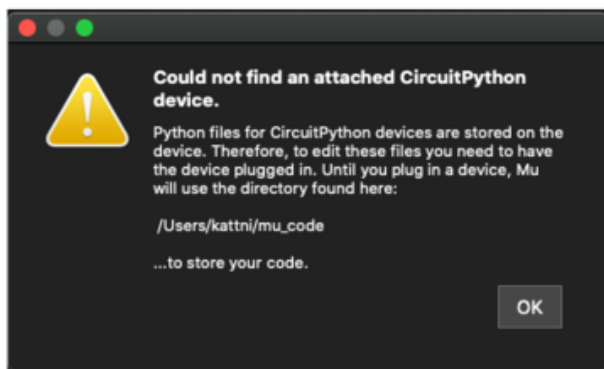
However, these print statements need somewhere to display. That's where the serial console comes in!

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 display those too.

The serial console requires an editor that has a built in terminal, or a separate terminal program. A terminal is a program that gives you a text-based interface to perform various tasks.

Are you using Mu?

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

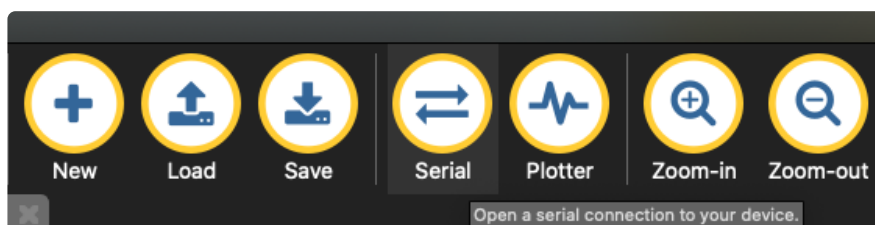


First, make sure your CircuitPython board is plugged in.

If you open Mu without a board plugged in, you may encounter the error seen here, letting you know no CircuitPython board was found and indicating where your code will be stored until you plug in a board.

If you are using Windows 7, make sure you installed the drivers (<https://adafru.it/VuB>).

Once you've opened Mu with your board plugged in, look for the Serial button in the button bar and click it.



The Mu window will split in two, horizontally, and display the serial console at the bottom.

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

Code done running.

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

If nothing appears in the serial console, it may mean your code is done running or has no print statements in it. Click into the serial console part of Mu, and press CTRL+D to reload.

Serial Console Issues or Delays on Linux

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 **modemmanager**, type the following command at a shell:

```
sudo apt purge modemmanager
```

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 the [Advanced](#)

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

Using Something Else?

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

Windows requires you to download a terminal program. [Check out the Advanced Serial Console on Windows page for more details. \(https://adafru.it/AAH\)](https://adafru.it/AAH)

MacOS has Terminal built in, though there are other options available for download. [Check the Advanced Serial Console on Mac page for more details. \(https://adafru.it/AAI\)](https://adafru.it/AAI)

Linux has a terminal program built in, though other options are available for download. [Check the Advanced Serial Console on Linux page for more details. \(https://adafru.it/VAO\)](https://adafru.it/VAO)

Once connected, you'll see something like the following.



```
Auto-reload is on. Simply save files over USB to run them or enter REPL to disable.  
code.py output:  
Hello, world!  
  
Code done running.  
  
Press any key to enter the REPL. Use CTRL-D to reload.
```

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, you'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
```



```

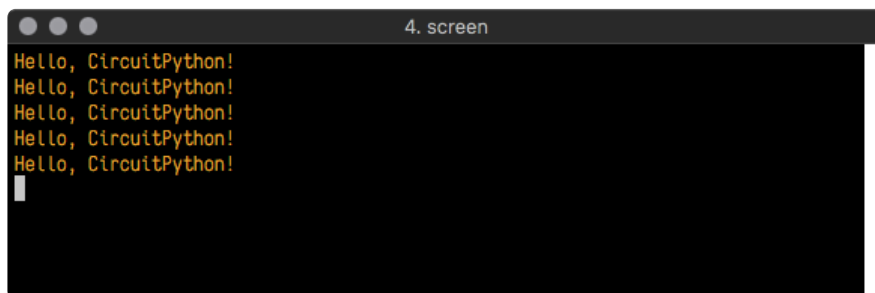
led = digitalio.DigitalInOut(board.LED)
led.direction = digitalio.Direction.OUTPUT

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.



```

4. screen
Hello, CircuitPython!
Hello, CircuitPython!
Hello, CircuitPython!
Hello, CircuitPython!
Hello, CircuitPython!

```

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

```

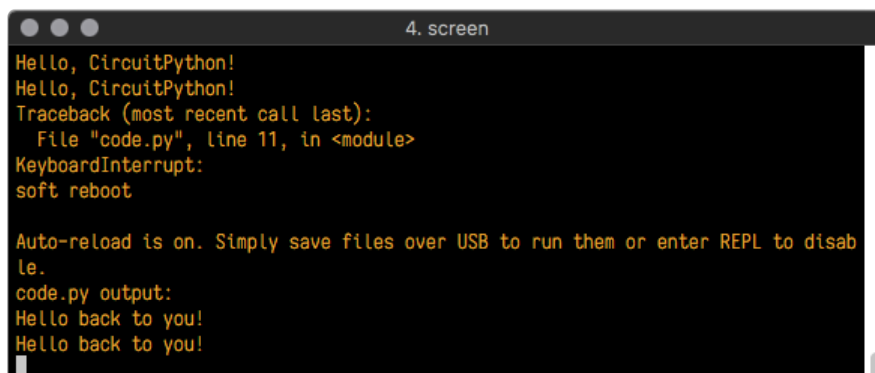
import board
import digitalio
import time

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

while True:
    print("Hello back to you!")
    led.value = True
    time.sleep(1)
    led.value = False
    time.sleep(1)

```

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!



```

4. screen
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 you 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**

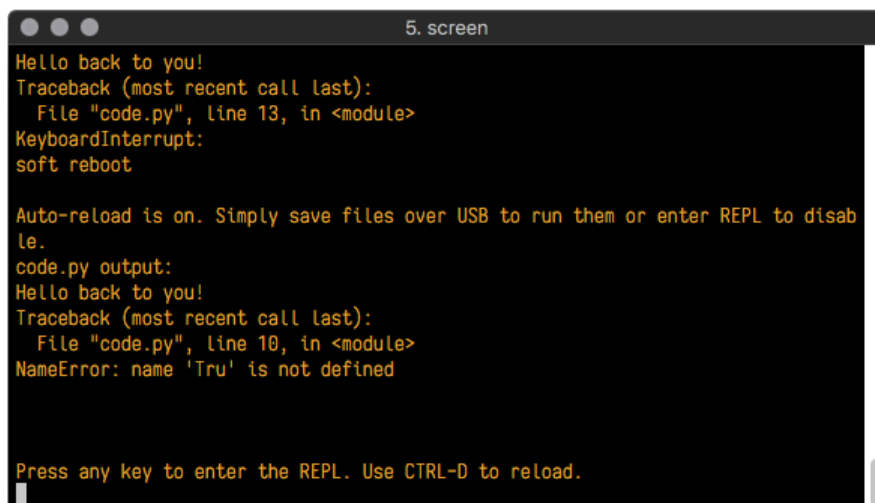
```
import board
import digitalio
import time

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

while True:
    print("Hello back to you!")
    led.value = Tru
    time.sleep(1)
    led.value = False
    time.sleep(1)
```

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. You 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!



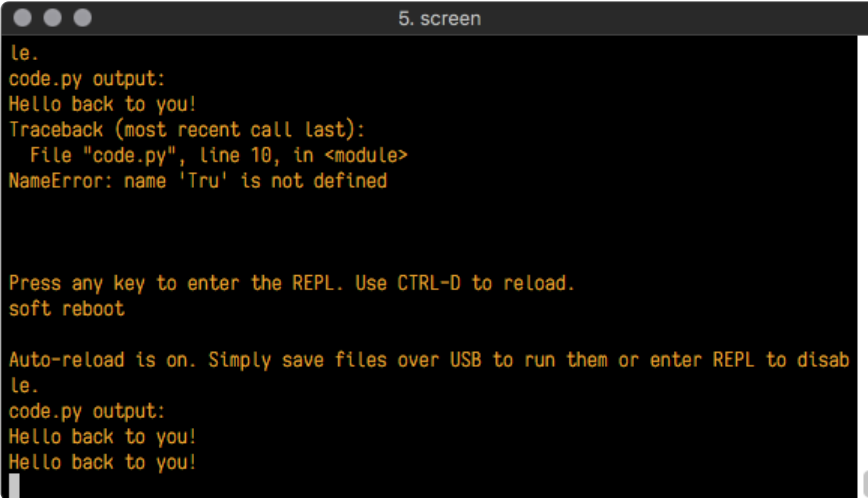
```
5. screen
Hello back to you!
Traceback (most recent call last):
  File "code.py", line 13, in <module>
KeyboardInterrupt:
soft reboot

Auto-reload is on. Simply save files over USB to run them or enter REPL to disable.
code.py output:
Hello back to you!
Traceback (most recent call last):
  File "code.py", line 10, in <module>
NameError: name 'Tru' is not defined

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

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

Go back to your code, and take a look at line 10. Obviously, you know what the problem is already. But if you didn't, you'd want to look at line 10 and see if you could figure it out. If you're still unsure, try googling the error to get some help. In this case, you know what to look for. You spelled True wrong. Fix the typo and save your file.



```
le.  
code.py output:  
Hello back to you!  
Traceback (most recent call last):  
  File "code.py", line 10, in <module>  
NameError: name 'Tru' is not defined  
  
Press any key to enter the REPL. Use CTRL-D to reload.  
soft reboot  
  
Auto-reload is on. Simply save files over USB to run them or enter REPL to disable.  
code.py output:  
Hello back to you!  
Hello back to you!
```

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

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

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

The REPL

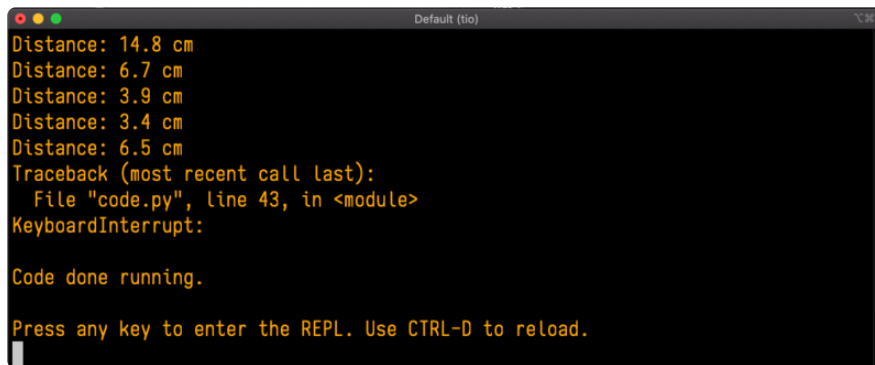
The other feature of the serial connection is the Read-Evaluate-Print-Loop, or REPL. The REPL allows you to enter individual lines of code and have them run immediately. It's really handy if you're running into trouble with a particular program and can't figure out why. It's interactive so it's great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

Entering the REPL

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

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

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

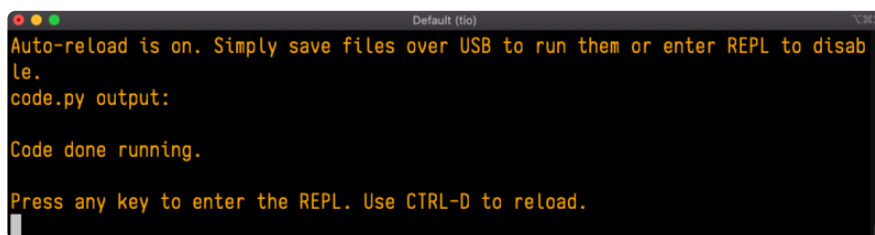


```
Distance: 14.8 cm
Distance: 6.7 cm
Distance: 3.9 cm
Distance: 3.4 cm
Distance: 6.5 cm
Traceback (most recent call last):
  File "code.py", line 43, in <module>
KeyboardInterrupt:

Code done running.

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

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



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

Code done running.

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

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

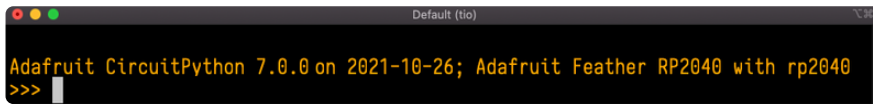


```
Auto-reload is on. Simply save files over USB to run them or enter REPL to disable.

Code done running.

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

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



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

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

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



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

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

This is followed by the CircuitPython prompt.

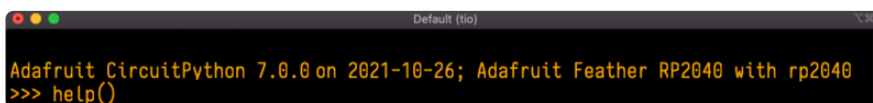


```
>>>
```

Interacting with the REPL

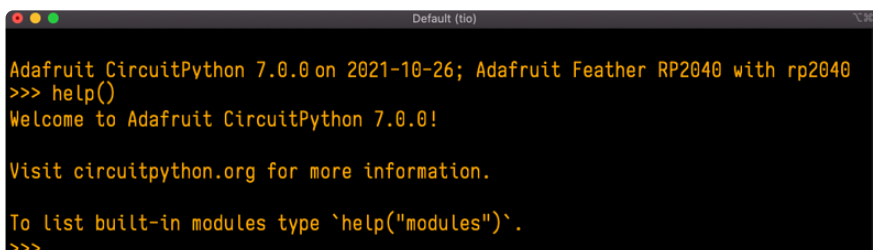
From this prompt you can run all sorts of commands and code. The first thing you'll do is run `help()`. This will tell you where to start exploring the REPL. To run code in the REPL, type it in next to the REPL prompt.

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



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

Then press enter. You should then see a message.



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

Visit circuitpython.org for more information.

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

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

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

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

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

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

```
>>> import board
>>>
```

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

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

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

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

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

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

Then press enter.

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

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

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

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

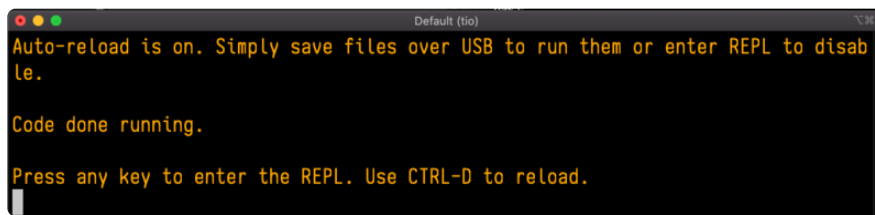
Try typing more into the REPL to see what happens!

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

Returning to the Serial Console

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

You can return to the REPL at any time!

A terminal window titled 'Default (tio)' with a black background and yellow text. The text reads: 'Auto-reload is on. Simply save files over USB to run them or enter REPL to disable.', 'Code done running.', and 'Press any key to enter the REPL. Use CTRL-D to reload.'

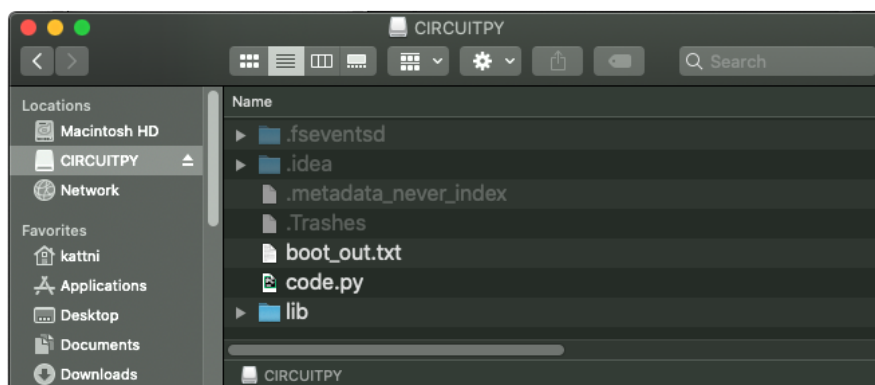
```
Default (tio)
Auto-reload is on. Simply save files over USB to run them or enter REPL to disable.
Code done running.
Press any key to enter the REPL. Use CTRL-D to reload.
```

CircuitPython Libraries

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

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

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



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

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

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

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

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

The Adafruit CircuitPython Library Bundle

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

Downloading the Adafruit CircuitPython Library Bundle

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

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

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

<https://adafru.it/ENC>

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

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

The CircuitPython Community Library Bundle

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

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

Downloading the CircuitPython Community Library Bundle

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

Click for the latest CircuitPython Community Library Bundle release

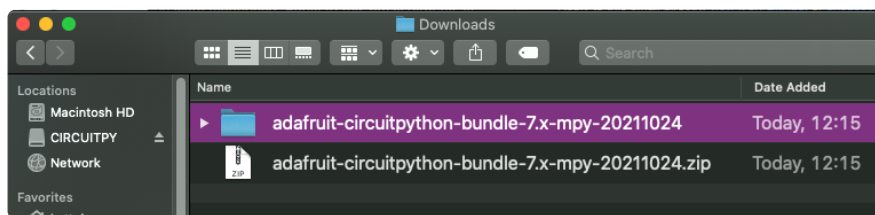
<https://adafru.it/VcN>

The link takes you to the latest release of the CircuitPython Community Library Bundle on GitHub. There are multiple versions of the bundle available. Download the

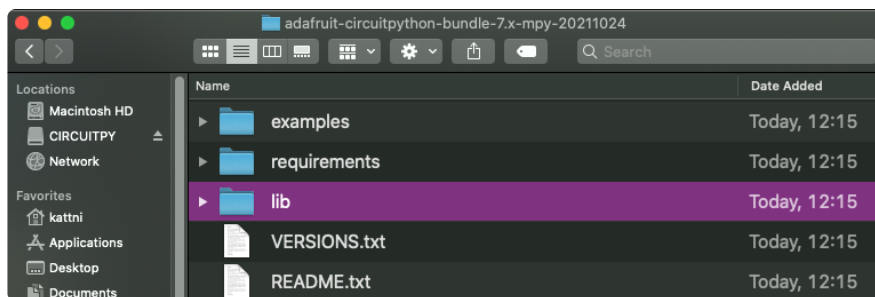
bundle version that matches your CircuitPython firmware version. If you don't know the version, check the version info in `boot_out.txt` file on the CIRCUITPY drive, or the initial prompt in the CircuitPython REPL. For example, if you're running v7.0.0, download the 7.x library bundle.

Understanding the Bundle

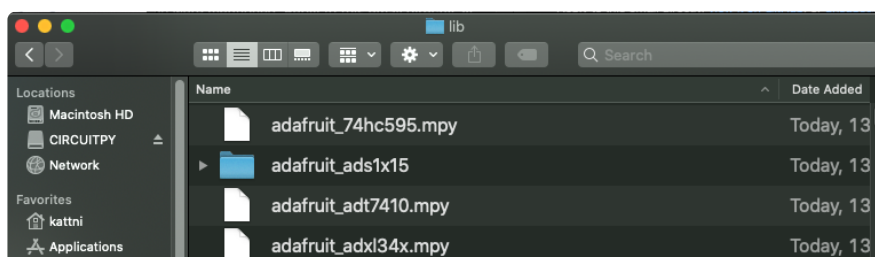
After downloading the zip, extract its contents. This is usually done by double clicking on the zip. On Mac OSX, it places the file in the same directory as the zip.



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



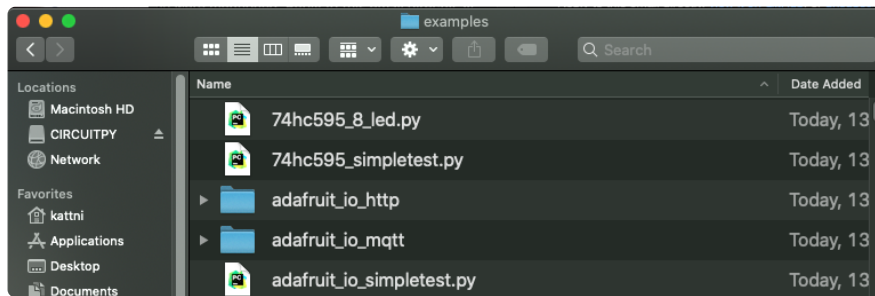
Now open the lib folder. When you open the folder, you'll see a large number of .mpy files, and folders.



Example Files

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

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



Copying Libraries to Your Board

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

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

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

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

Understanding Which Libraries to Install

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

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

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

- `import library_or_module`

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

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

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

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

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

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

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

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

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

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

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

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

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

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

The final two imports are not as clear. Remember, when `import` statements are formatted like this, the first thing after the `from` is the library name. In this case, the library name is `adafruit_hid`. A search of the bundle will find an `adafruit_hid` folder.

When a library is a folder, you must copy the entire folder and its contents as it is in the bundle to the lib folder on your CIRCUITPY drive. In this case, you would copy the entire adafruit_hid folder to your CIRCUITPY/lib folder.

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

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

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

Example: `ImportError` Due to Missing Library

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

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

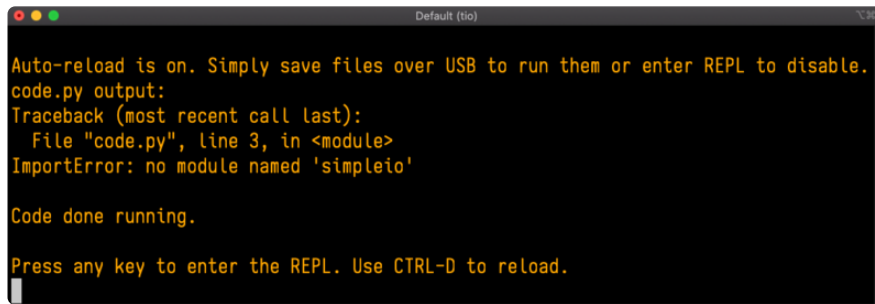
Let's use a modified version of the Blink example.

```
import board
import time
import simpleio

led = simpleio.DigitalOut(board.LED)

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

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

A screenshot of a serial console window titled "Default (tio)". The text displayed is: "Auto-reload is on. Simply save files over USB to run them or enter REPL to disable. code.py output: Traceback (most recent call last): File \"code.py\", line 3, in <module> ImportError: no module named 'simpleio' Code done running. Press any key to enter the REPL. Use CTRL-D to reload."

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

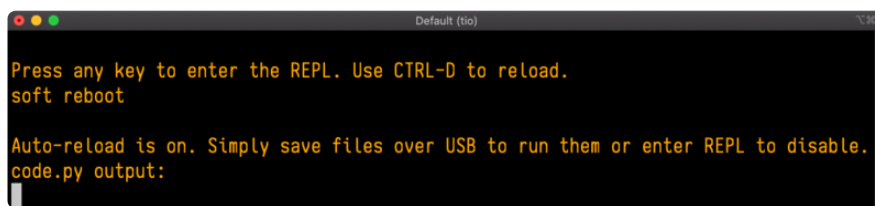
Code done running.

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

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

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

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

A screenshot of a serial console window titled "Default (tio)". The text displayed is: "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:"

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

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

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

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

Library Install on Non-Express Boards

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

You can still end up running out of space on your M0 non-Express board even if you only load libraries as you need them. There are a number of steps you can use to try

to resolve this issue. You'll find suggestions on the [Troubleshooting page \(https://adafru.it/Den\)](https://adafru.it/Den).

Updating CircuitPython Libraries and Examples

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

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

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

CircuitPython Pins and Modules

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

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

CircuitPython Pins

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

import board

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

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

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

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

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

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

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

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

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

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

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

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

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

I2C, SPI, and UART

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

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

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

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

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

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

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

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

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

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

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

What Are All the Available Names?

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

On a microcontroller board running CircuitPython, connect to the serial console. Then, save the following as `code.py` on your CIRCUITPY drive.

```
"""CircuitPython Essentials Pin Map Script"""
import microcontroller
import board

board_pins = []
for pin in dir(microcontroller.pin):
    if isinstance(getattr(microcontroller.pin, pin), microcontroller.Pin):
        pins = []
        for alias in dir(board):
            if getattr(board, alias) is getattr(microcontroller.pin, pin):
                pins.append("board.{}".format(alias))
        if len(pins) > 0:
            board_pins.append(" ".join(pins))
for pins in sorted(board_pins):
    print(pins)
```

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

```
board.A0 board.D0
board.A1 board.D1
board.A10 board.D10 board.MOSI
board.A2 board.D2
board.A3 board.D3
board.A6 board.D6 board.TX
board.A7 board.D7 board.RX
board.A8 board.D8 board.SCK
board.A9 board.D9 board.MISO
board.D4 board.SDA
board.D5 board.SCL
board.NEOPIXEL
board.NEOPIXEL_POWER
```

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

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

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

Microcontroller Pin Names

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

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

CircuitPython Built-In Modules

There is a set of modules used in most CircuitPython programs. One or more of these modules is always used in projects involving hardware. Often hardware requires installing a separate library from the Adafruit CircuitPython Bundle. But, if you try to find `board` or `digitalio` in the same bundle, you'll come up lacking. So, where do these modules come from? They're built into CircuitPython! You can find an comprehensive list of built-in CircuitPython modules and the technical details of their functionality from CircuitPython [here](https://adafru.it/QkB) (<https://adafru.it/QkB>) and the Python-like modules included [here](https://adafru.it/QkC) (<https://adafru.it/QkC>). However, not every module is available for every board due to size constraints or hardware limitations. How do you find out what modules are available for your board?

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

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

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

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

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

Troubleshooting

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

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

the CircuitPython Library Bundle that matches your version of CircuitPython. Please update CircuitPython and then visit <https://circuitpython.org/libraries> to download the latest Library Bundle.

Always Run the Latest Version of CircuitPython and Libraries

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

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

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

I have to continue using CircuitPython 5.x or earlier. Where can I find compatible libraries?

Adafruit is no longer building or supporting the CircuitPython 5.x or earlier library bundles. You are highly encouraged to [update CircuitPython to the latest version](https://adafru.it/Em8) (<https://adafru.it/Em8>) and use [the current version of the libraries](https://adafru.it/ENC) (<https://adafru.it/ENC>). However, if for some reason you cannot update, links to the previous bundles are available in the [FAQ](https://adafru.it/FwY) (<https://adafru.it/FwY>).

Bootloader (boardnameBOOT) Drive Not Present

You may have a different board.

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

MakeCode

If you are running a [MakeCode](https://adafru.it/zbY) (<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) (<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

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

It is [recommended](https://adafru.it/Amd) (<https://adafru.it/Amd>) that you upgrade to Windows 10 if possible; an upgrade is probably still free for you. Check [here](https://adafru.it/Amd) (<https://adafru.it/Amd>).

The Windows Drivers installer was last updated in November 2020 (v2.5.0.0) . Windows 7 drivers for CircuitPython boards released since then, including RP2040 boards, are not yet available. The boards work fine on Windows 10. A new release of the drivers is in process.

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/jlff) (<https://adafru.it/jlff>) or on the [Adafruit Discord](#) () if this does not work for you!

Windows Explorer Locks Up When Accessing boardnameBOOT Drive

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

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

Copying UF2 to boardnameBOOT Drive Hangs at 0% Copied

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

CIRCUITPY Drive Does Not Appear

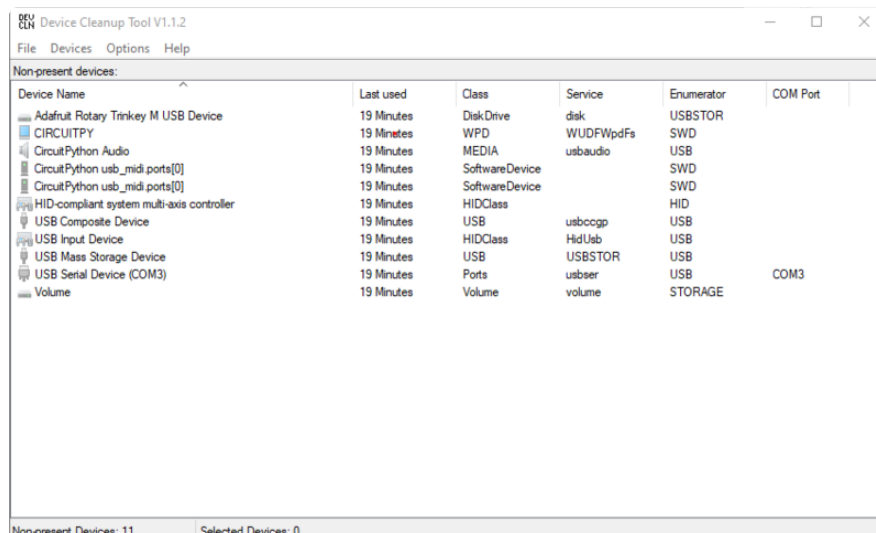
Kaspersky anti-virus can block the appearance of the CIRCUITPY drive. There has not yet been settings change discovered that prevents this. Complete uninstallation of Kaspersky fixes the problem.

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

Device Errors or Problems on Windows

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

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



Select all the devices you want to remove, and then press Delete. It is usually safe just to select everything. Any device that is removed will get a fresh install when you plug it in. Using the Device Cleanup Tool also discards all the COM port assignments for the unplugged boards. If you have used many Arduino and CircuitPython boards,

you have probably seen higher and higher COM port numbers used, seemingly without end. This will fix that problem.

Serial Console in Mu Not Displaying Anything

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

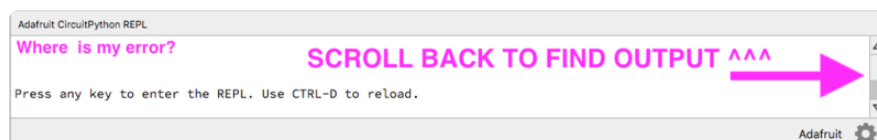
Depending on the size of your screen or Mu window, when you open the serial console, the serial console panel may be very small. This can be a problem. A basic CircuitPython error takes 10 lines to display!

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

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

More complex errors take even more lines!

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



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

code.py Restarts Constantly

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

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

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

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

```
import supervisor
supervisor.disable_autoreload()
```

CircuitPython RGB Status Light

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

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

CircuitPython 7.0.0 and Later

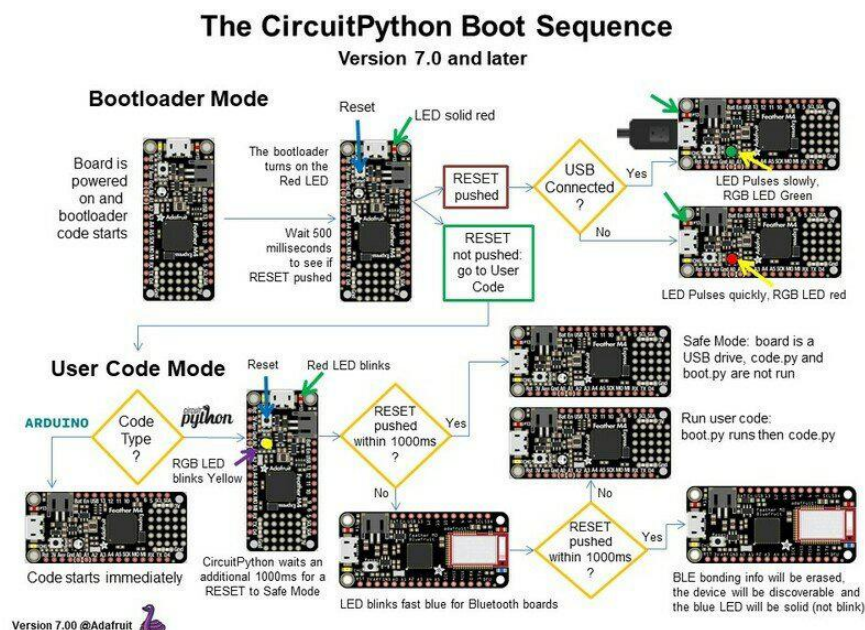
The status LED blinks were changed in CircuitPython 7.0.0 in order to save battery power and simplify the blinks. These blink patterns will occur on single color LEDs when the board does not have any RGB LEDs. Speed and blink count also vary for this reason.

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

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

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

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



CircuitPython 6.3.0 and earlier

Here's what the colors and blinking mean:

- steady GREEN: code.py (or code.txt, main.py, or main.txt) is running
- pulsing GREEN: code.py (etc.) has finished or does not exist

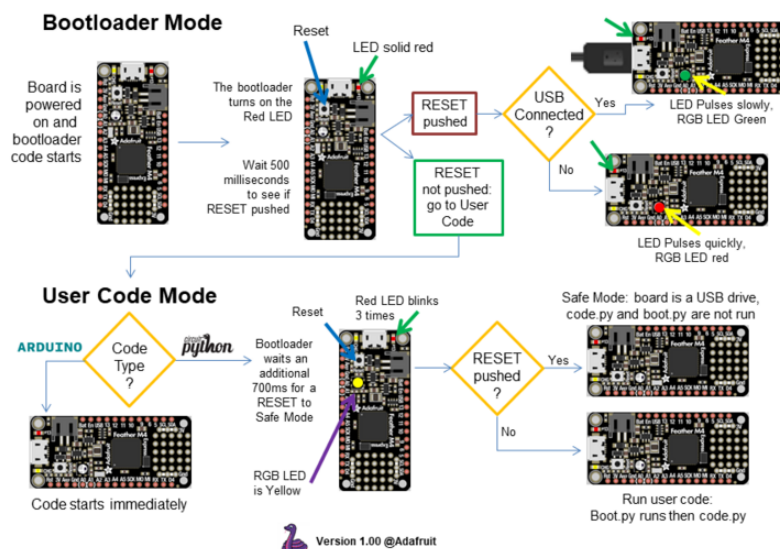
- steady YELLOW at start up: (4.0.0-alpha.5 and newer) CircuitPython is waiting for a reset to indicate that it should start in safe mode
- pulsing YELLOW: Circuit Python is in safe mode: it crashed and restarted
- steady WHITE: REPL is running
- steady BLUE: boot.py is running

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

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

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

The CircuitPython Boot Sequence



Serial console showing `ValueError: Incompatible .mpy file`

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

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

CIRCUITPY Drive Issues

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

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

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

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

Safe Mode

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

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

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

Entering Safe Mode in CircuitPython 7.x

To enter safe mode when using CircuitPython 7.x, plug in your board or hit reset (highlighted in red above). Immediately after the board starts up or resets, it waits 1000ms. On some boards, the onboard status LED will blink yellow during that time. If you press reset during that 1000ms, the board will start up in safe mode. It can be difficult to react to the yellow LED, so you may want to think of it simply as a "slow" double click of the reset button. (Remember, a fast double click of reset enters the bootloader.)

Entering Safe Mode in CircuitPython 6.x

To enter safe mode when using CircuitPython 6.x, plug in your board or hit reset (highlighted in red above). Immediately after the board starts up or resets, it waits 700ms. On some boards, the onboard status LED (highlighted in green above) will turn solid yellow during this time. If you press reset during that 700ms, the board will start up in safe mode. It can be difficult to react to the yellow LED, so you may want to think of it simply as a slow double click of the reset button. (Remember, a fast double click of reset enters the bootloader.)

In Safe Mode

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

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

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


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

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

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

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

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

To erase CIRCUITPY: `storage.erase_filesystem()`

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

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

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

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

Erase CIRCUITPY Without Access to the REPL

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

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

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

For the specific boards listed below:

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

1. Download the correct erase file:

Circuit Playground Express

<https://adafru.it/AdI>

Feather M0 Express

<https://adafru.it/AdJ>

Feather M4 Express

<https://adafru.it/EVK>

Metro M0 Express

<https://adafru.it/AdK>

Metro M4 Express QSPI Eraser

<https://adafru.it/EoM>

Trellis M4 Express (QSPI)

<https://adafru.it/DjD>

Grand Central M4 Express (QSPI)

<https://adafru.it/DBA>

PyPortal M4 Express (QSPI)

<https://adafru.it/Eca>

Circuit Playground Bluefruit (QSPI)

<https://adafru.it/Gnc>

Monster M4SK (QSPI)

<https://adafru.it/GAN>

PyBadge/PyGamer QSPI Eraser.UF2

<https://adafru.it/GAO>

CLUE_Flash_Erase.UF2

<https://adafru.it/Jat>

Matrix_Portal_M4_(QSPI).UF2

<https://adafru.it/Q5B>

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

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

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

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

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

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

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

1. Download the erase file:

SAMD21 non-Express Boards

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

2. Double-click the reset button on the board to bring up the boardnameBOOT drive.
3. Drag the erase .uf2 file to the boardnameBOOT drive.
4. The boot LED will start flashing again, and the boardnameBOOT drive will reappear.
5. [Drag the appropriate latest release CircuitPython \(https://adafru.it/Em8\)](https://adafru.it/Em8) .uf2 file to the boardnameBOOT drive.

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

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

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

Any SAMD21-based microcontroller that does not have external flash available is considered a SAMD21 non-Express board. Non-Express boards that do not have a

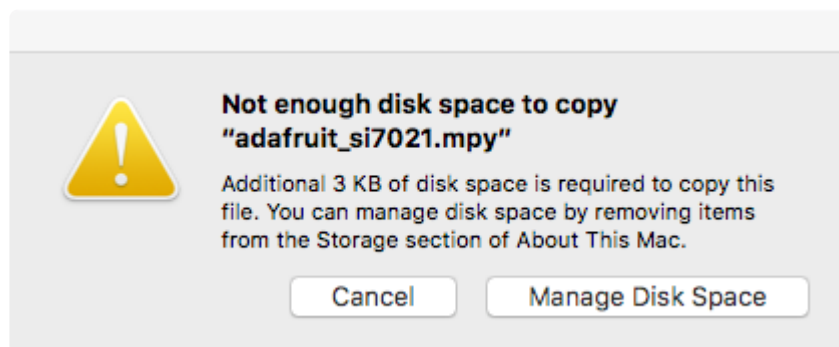
UF2 bootloader include the Feather M0 Basic Proto, Feather Adalogger, or the Arduino Zero.

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

Running Out of File Space on SAMD21 Non-Express Boards

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

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



Delete something!

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

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

Use tabs

One unique feature of Python is that the indentation of code matters. Usually the recommendation is to indent code with four spaces for every indent. In general, that

is recommended too. However, one trick to storing more human-readable code is to use a single tab character for indentation. This approach uses 1/4 of the space for indentation and can be significant when you're counting bytes.

On MacOS?

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

Prevent & Remove MacOS Hidden Files

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

```
ls -l /Volumes
```

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

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

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

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

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

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

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

Copy Files on MacOS Without Creating Hidden Files

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

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

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

(Replace `file_name.mpy` with the name of the file you want to copy.)

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

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

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

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

Other MacOS Space-Saving Tips

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

```

Last login: Thu Oct 28 17:19:15 on ttys008

7039 kattni@robocrepe:~ $ cd /Volumes/

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

7041 kattni@robocrepe:Volumes $

```

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

```

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

7042 kattni@robocrepe:Volumes $

```

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

```

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

7043 kattni@robocrepe:Volumes $

```

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

```

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

7044 kattni@robocrepe:Volumes $

```

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

Device Locked Up or Boot Looping

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

option. When the device boots up in safe mode it will not run the `code.py` or `boot.py` scripts, but will still connect the CIRCUITPY drive so that you can remove or modify those files as needed.

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

For most devices:

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

For ESP32-S2 based devices:

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

Refer to the diagrams above for boot sequence details.

"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 or reinstall CircuitPython whenever you want! Heck, you can change your mind every day!

There is nothing to uninstall. CircuitPython is "just another program" that is loaded onto your board. You simply load another program (Arduino or MakeCode) and it will overwrite CircuitPython.

Backup Your Code

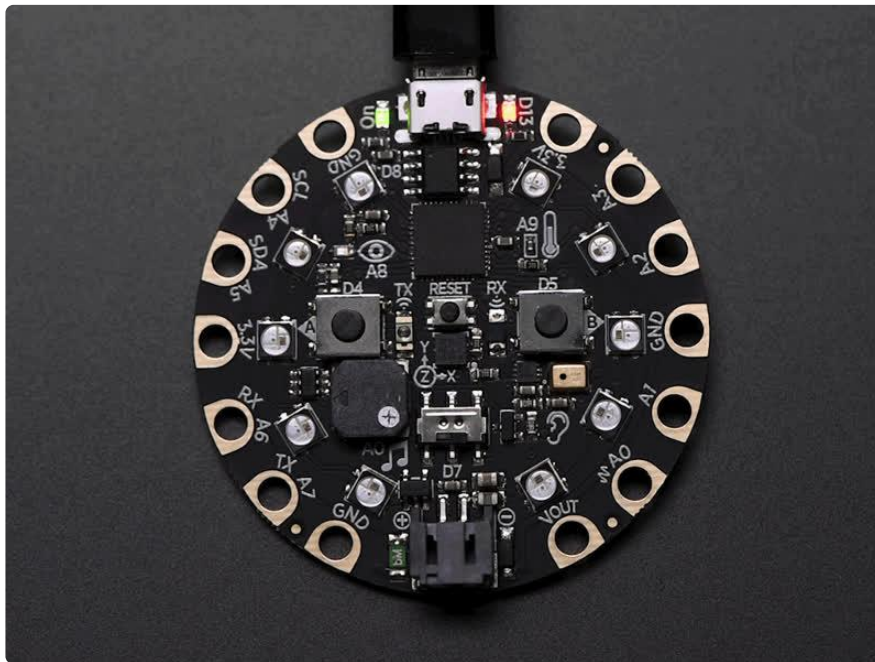
Before replacing CircuitPython, don't forget to make a backup of the code you have on the CIRCUITPY drive. That means your `code.py` any other files, the `lib` folder etc. You may lose these files when you remove CircuitPython, so backups are key! Just

drag the files to a folder on your laptop or desktop computer like you would with any USB drive.

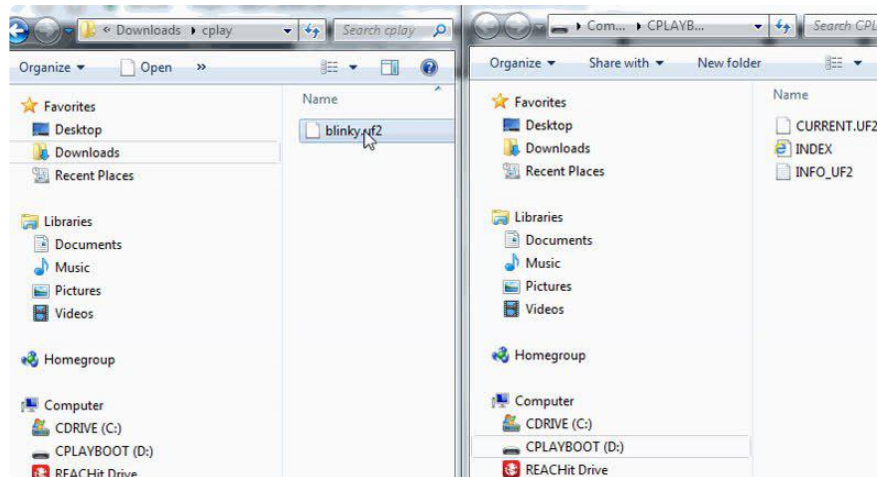
Moving Circuit Playground Express to MakeCode

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) (<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 CPLAYBOOT drive.



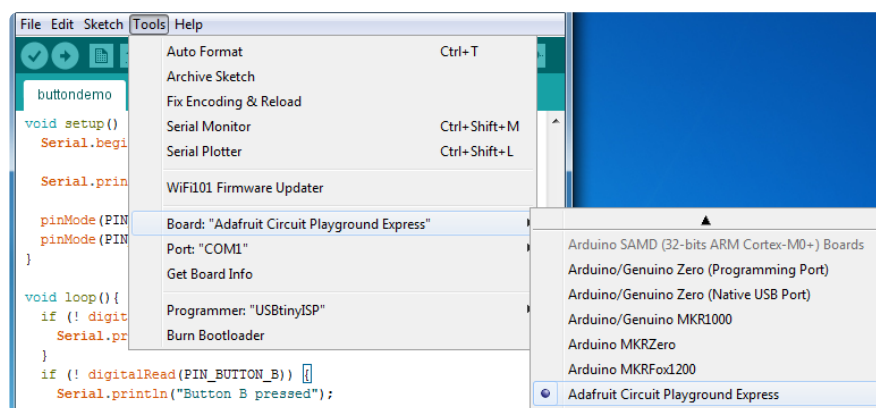
Your MakeCode is now running and CircuitPython has been removed. Going forward you only have to single click the reset button to get to CPLAYBOOT. This is an idiosyncrasy of MakeCode.

Moving to Arduino

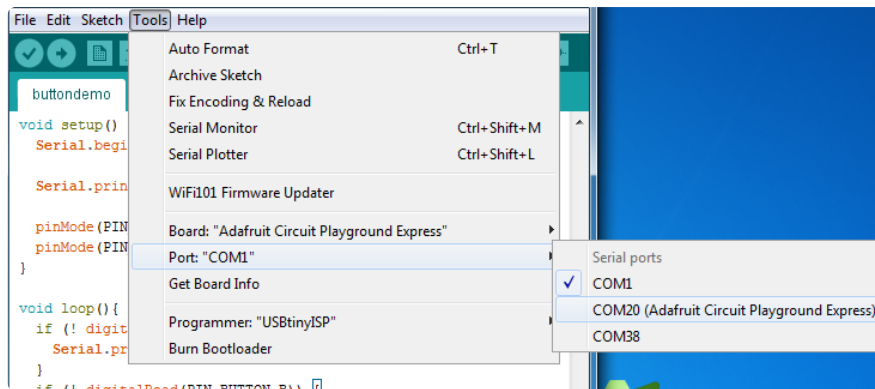
If you want to use Arduino instead, you just use the Arduino IDE to load an Arduino program. Here's an example of uploading a simple "Blink" Arduino program, but you don't have to use this particular program.

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

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



Select the correct matching Port:



Create a new simple Blink sketch example:

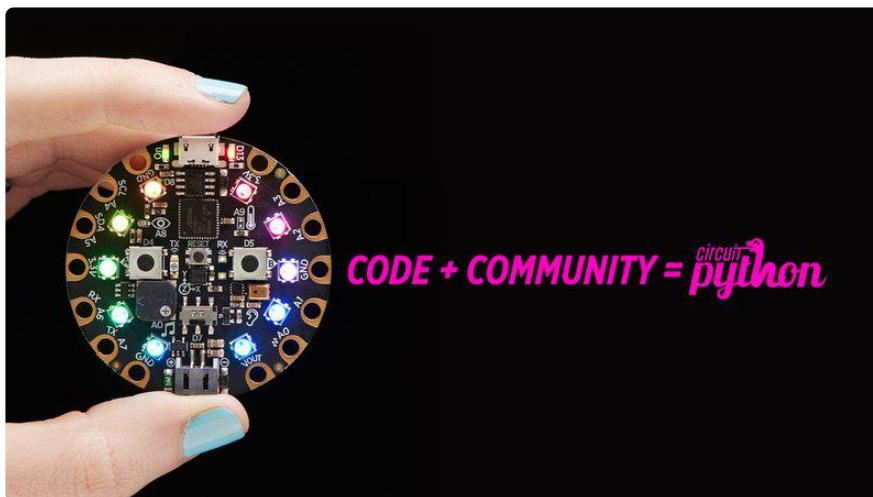
```
// the setup function runs once when you press reset or power the board
void setup() {
  // initialize digital pin 13 as an output.
  pinMode(13, OUTPUT);
}

// 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.

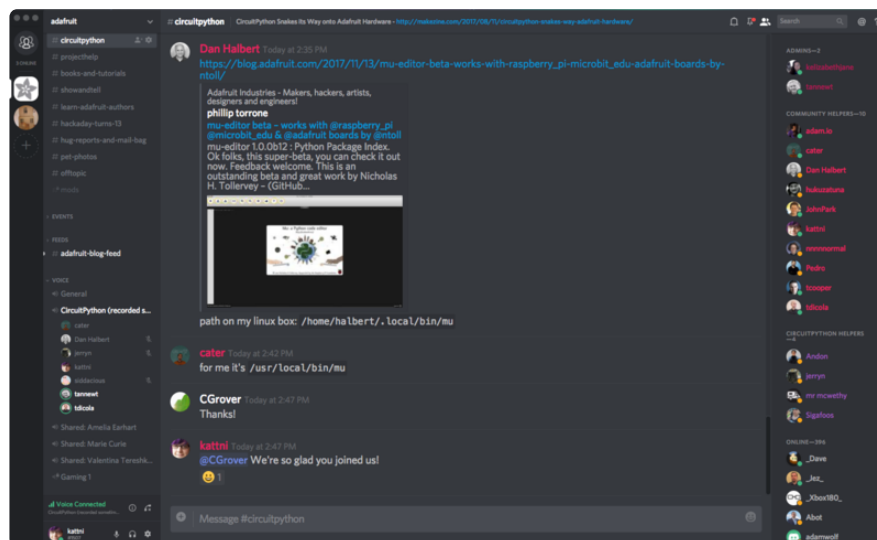
Welcome to the Community!



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. Whether this is your first microcontroller board or you're a seasoned software engineer, you have something important to offer the Adafruit CircuitPython community. This page highlights 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 #show-and-tell 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. Your contributions are important! The #circuitpython-dev 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. Everyone is looking forward to meeting you!

CircuitPython.org



Beyond the Adafruit Learn System, which you are viewing right now, the best place to find information about CircuitPython is [circuitpython.org](https://adafru.it/KJD) (<https://adafru.it/KJD>). Everything you need to get started with your new microcontroller and beyond is available. You can do things like [download CircuitPython for your microcontroller](https://adafru.it/Em8) (<https://adafru.it/Em8>) or [download the latest CircuitPython Library bundle](https://adafru.it/ENC) (<https://adafru.it/ENC>), or check out [which single board computers support Blinka](https://adafru.it/EA8) (<https://adafru.it/EA8>). You can also get to various other CircuitPython related things like Awesome CircuitPython or the Python for Microcontrollers newsletter. This is all incredibly useful, but it isn't necessarily community related. So why is it included here? The [Contributing page](https://adafru.it/VD7) (<https://adafru.it/VD7>).

Contributing

If you'd like to contribute to the CircuitPython project, the CircuitPython libraries are a great way to begin. This page is updated with daily status information from the CircuitPython libraries, including open pull requests, open issues and library infrastructure issues.

Do you write a language other than English? Another great way to contribute to the project is to contribute new localizations (translations) of CircuitPython, or update current localizations, using [Weblate](#).

If this is your first time contributing, or you'd like to see our recommended contribution workflow, we have a guide on [Contributing to CircuitPython with Git and Github](#). You can also find us in the #circuitpython channel on the [Adafruit Discord](#).

Have an idea for a new driver or library? [File an issue on the CircuitPython repo!](#)

CircuitPython itself is written in C. However, all of the Adafruit CircuitPython libraries are written in Python. If you're interested in contributing to CircuitPython on the Python side of things, check out [circuitpython.org/contributing](https://adafru.it/VD7) (<https://adafru.it/VD7>). You'll find information pertaining to every Adafruit CircuitPython library GitHub repository, giving you the opportunity to join the community by finding a contributing option that works for you.

Note the date on the page next to Current Status for:

Current Status for Tue, Nov 02, 2021

If you submit any contributions to the libraries, and do not see them reflected on the Contributing page, it could be that the job that checks for new updates hasn't yet run for today. Simply check back tomorrow!

Now, a look at the different options.

Pull Requests

The first tab you'll find is a list of open pull requests.



Pull Requests Open Issues Library Infrastructure Issues CircuitPython Localization

This is the current status of open pull requests and issues across all of the library repos.

Open Pull Requests

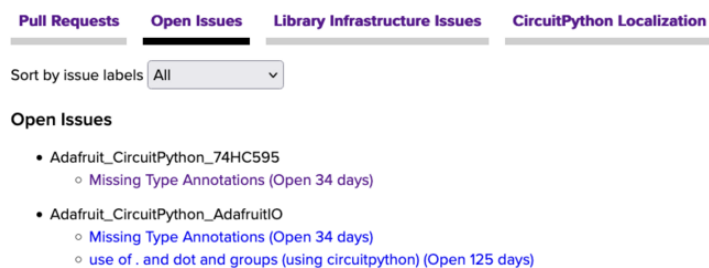
- Adafruit_CircuitPython_AdafruitIO
 - [Call wifi.connect\(\) after wifi.reset\(\)](#) (Open 113 days)
- Adafruit_CircuitPython_ADS1x15
 - [Supress f-string recommendation in .pylintrc](#) (Open 1 days)
- Adafruit_CircuitPython_ADT7410
 - [Adding critical temp features](#) (Open 168 days)

GitHub pull requests, or PRs, are opened when folks have added something to an Adafruit CircuitPython library GitHub repo, and are asking for Adafruit to add, or merge, their changes into the main library code. For PRs to be merged, they must first be reviewed. Reviewing is a great way to contribute! Take a look at the list of open

pull requests, and pick one that interests you. If you have the hardware, you can test code changes. If you don't, you can still check the code updates for syntax. In the case of documentation updates, you can verify the information, or check it for spelling and grammar. Once you've checked out the update, you can leave a comment letting us know that you took a look. Once you've done that for a while, and you're more comfortable with it, you can consider joining the CircuitPythonLibrarians review team. The more reviewers we have, the more authors we can support. Reviewing is a crucial part of an open source ecosystem, CircuitPython included.

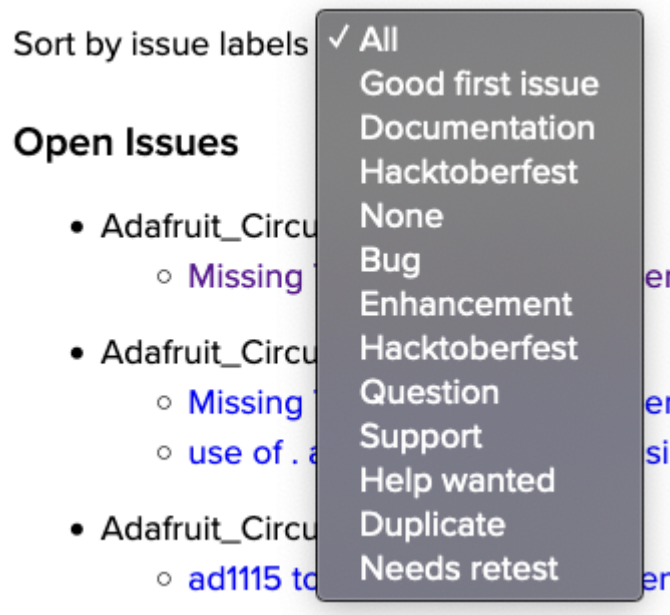
Open Issues

The second tab you'll find is a list of open issues.



GitHub issues are filed for a number of reasons, including when there is a bug in the library or example code, or when someone wants to make a feature request. Issues are a great way to find an opportunity to contribute directly to the libraries by updating code or documentation. If you're interested in contributing code or documentation, take a look at the open issues and find one that interests you.

If you're not sure where to start, you can search the issues by label. Labels are applied to issues to make the goal easier to identify at a first glance, or to indicate the difficulty level of the issue. Click on the dropdown next to "Sort by issue labels" to see the list of available labels, and click on one to choose it.



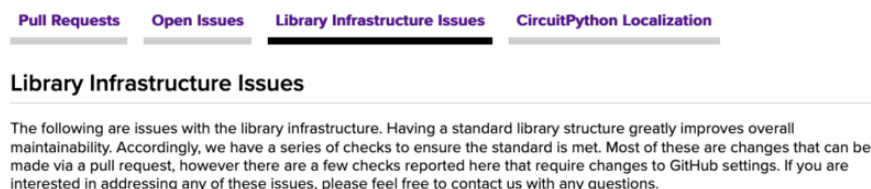
If you're new to everything, new to contributing to open source, or new to contributing to the CircuitPython project, you can choose "Good first issue". Issues with that label are well defined, with a finite scope, and are intended to be easy for someone new to figure out.

If you're looking for something a little more complicated, consider "Bug" or "Enhancement". The Bug label is applied to issues that pertain to problems or failures found in the library. The Enhancement label is applied to feature requests.

Don't let the process intimidate you. If you're new to Git and GitHub, there is [a guide](https://adafru.it/Dkh) (<https://adafru.it/Dkh>) to walk you through the entire process. As well, there are always folks available on [Discord](#) () to answer questions.

Library Infrastructure Issues

The third tab you'll find is a list of library infrastructure issues.



This section is generated by a script that runs checks on the libraries, and then reports back where there may be issues. It is made up of a list of subsections each containing links to the repositories that are experiencing that particular issue. This page is available mostly for internal use, but you may find some opportunities to

contribute on this page. If there's an issue listed that sounds like something you could help with, mention it on Discord, or file an issue on GitHub indicating you're working to resolve that issue. Others can reply either way to let you know what the scope of it might be, and help you resolve it if necessary.

CircuitPython Localization

The fourth tab you'll find is the CircuitPython Localization tab.

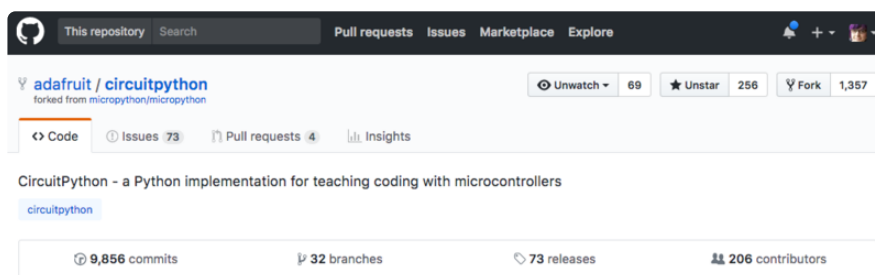


If you speak another language, you can help translate CircuitPython! The translations apply to informational and error messages that are within the CircuitPython core. It means that folks who do not speak English have the opportunity to have these messages shown to them in their own language when using CircuitPython. This is incredibly important to provide the best experience possible for all users.

CircuitPython uses Weblate to translate, which makes it much simpler to contribute translations. You will still need to know some CircuitPython-specific practices and a few basics about coding strings, but as with any CircuitPython contributions, folks are there to help.

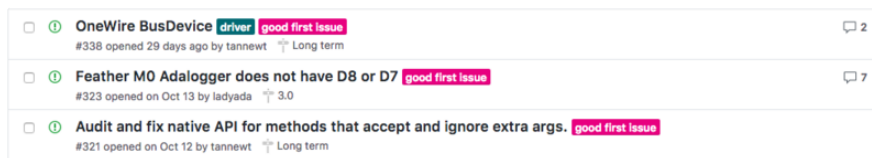
Regardless of your skill level, or how you want to contribute to the CircuitPython project, there is an opportunity available. The [Contributing page \(https://adafru.it/VD7\)](https://adafru.it/VD7) is an excellent place to start!

Adafruit GitHub



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 the CircuitPython project. The CircuitPython core is written in C. The libraries are written in Python. GitHub is the best source of ways to contribute to the [CircuitPython core](https://adafru.it/tB7) (<https://adafru.it/tB7>), and the [CircuitPython libraries](https://adafru.it/VFv) (<https://adafru.it/VFv>). 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. For the CircuitPython core, head over to the CircuitPython repository on GitHub, click on "[Issues](https://adafru.it/tBb)" (<https://adafru.it/tBb>), and you'll find a list that includes issues labeled "[good first issue](https://adafru.it/Bef)" (<https://adafru.it/Bef>"). For the libraries, head over to the [Contributing page Issues list](https://adafru.it/VFv) (<https://adafru.it/VFv>), and use the drop down menu to search for "[good first issue](https://adafru.it/VFw)" (<https://adafru.it/VFw>"). These issues are things that have been 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. If you need help getting started with GitHub, there is an excellent guide on [Contributing to CircuitPython with Git and GitHub](https://adafru.it/Dkh) (<https://adafru.it/Dkh>).



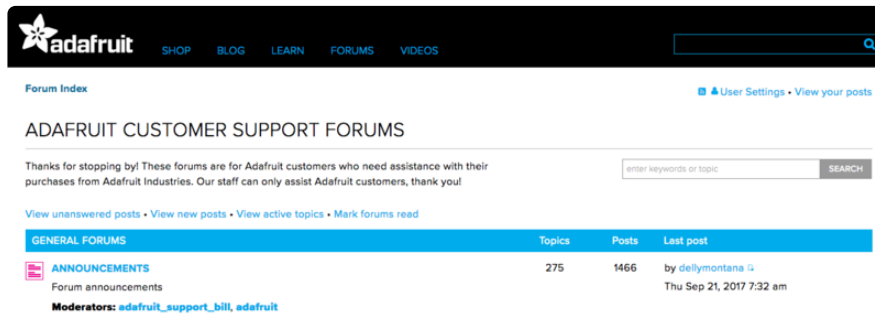
Already experienced and looking for a challenge? Checkout the rest of either issues list and you'll find plenty of ways to contribute. You'll find all sorts of things, from new driver requests, to library bugs, to core module updates. There's plenty of opportunities for everyone at any level!

When working with or using CircuitPython or the CircuitPython libraries, you may find problems. If you find a bug, that's great! The team loves bugs! Posting a detailed issue to GitHub is an invaluable way to contribute to improving CircuitPython. For CircuitPython itself, file an issue [here](https://adafru.it/tBb) (<https://adafru.it/tBb>). For the libraries, file an issue on the specific library repository on GitHub. 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 stable and unstable releases is a very important part of contributing CircuitPython. The developers can't possibly find all the problems themselves! They 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!

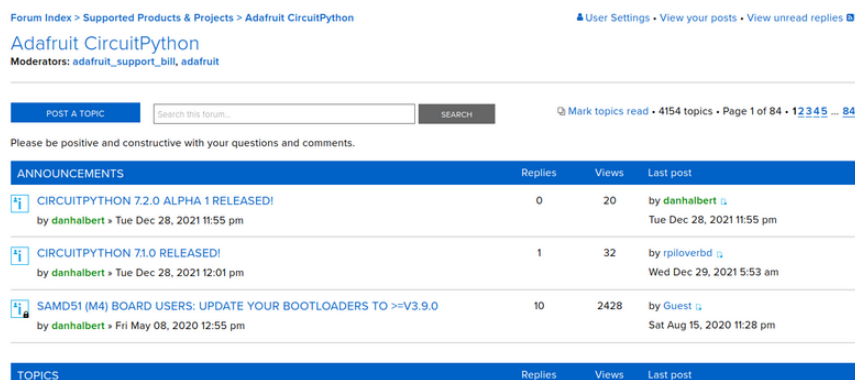
Adafruit Forums



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.

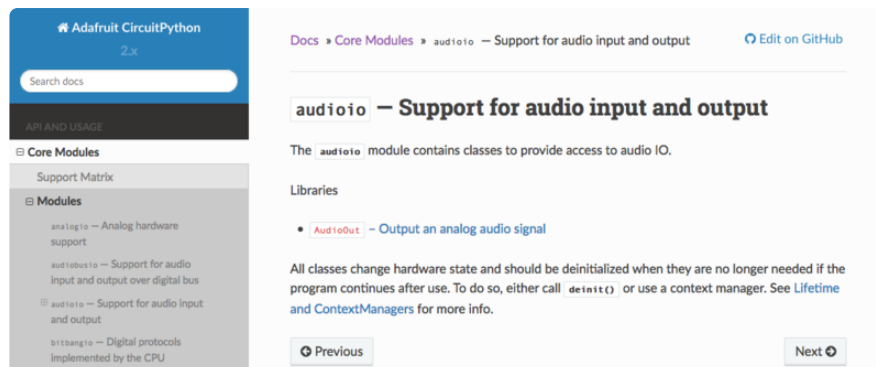
There are forum categories that cover all kinds of topics, including everything Adafruit. The [Adafruit CircuitPython \(https://adafru.it/xXA\)](https://adafru.it/xXA) category under "Supported Products & Projects" is the best place to post your CircuitPython questions.



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!

Read the Docs



[Read the Docs \(https://adafru.it/Beg\)](https://adafru.it/Beg) is an excellent resource for a more detailed look at the CircuitPython core and the CircuitPython libraries. This is where you'll find things like API documentation and example code. For an in depth look at viewing and understanding Read the Docs, check out the [CircuitPython Documentation \(https://adafru.it/VFx\)](https://adafru.it/VFx) page!

Here is blinky:

```
import time
import digitalio
import board

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

UF2 Bootloader Details

This is an information page for advanced users who are curious how we get code from your computer into your Express board!

Adafruit SAMD21 (M0) and SAMD51 (M4) boards feature an improved bootloader that makes it easier than ever to flash different code onto the microcontroller. This bootloader makes it easy to switch between Microsoft MakeCode, CircuitPython and

Arduino.

Instead of needing drivers or a separate program for flashing (say, `bossac`, `jlink` or `avrdude`), one can simply drag a file onto a removable drive.

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

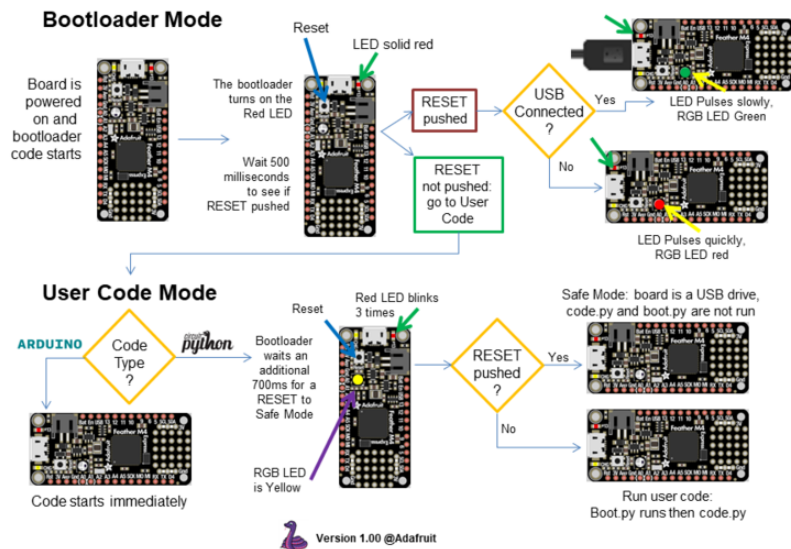
The bootloader is also BOSSA compatible, so it can be used with the Arduino IDE which expects a BOSSA bootloader on ATSAMD-based boards

For more information about UF2, [you can read a bunch more at the MakeCode blog \(https://adafru.it/w5A\)](https://adafru.it/w5A), then [check out the UF2 file format specification. \(https://adafru.it/vPE\)](https://adafru.it/vPE)

Visit [Adafruit's fork of the Microsoft UF2-samd bootloader GitHub repository \(https://adafru.it/Beu\)](https://adafru.it/Beu) for source code and releases of pre-built bootloaders on [CircuitPython.org \(https://adafru.it/Em8\)](https://adafru.it/Em8).

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

The CircuitPython Boot Sequence

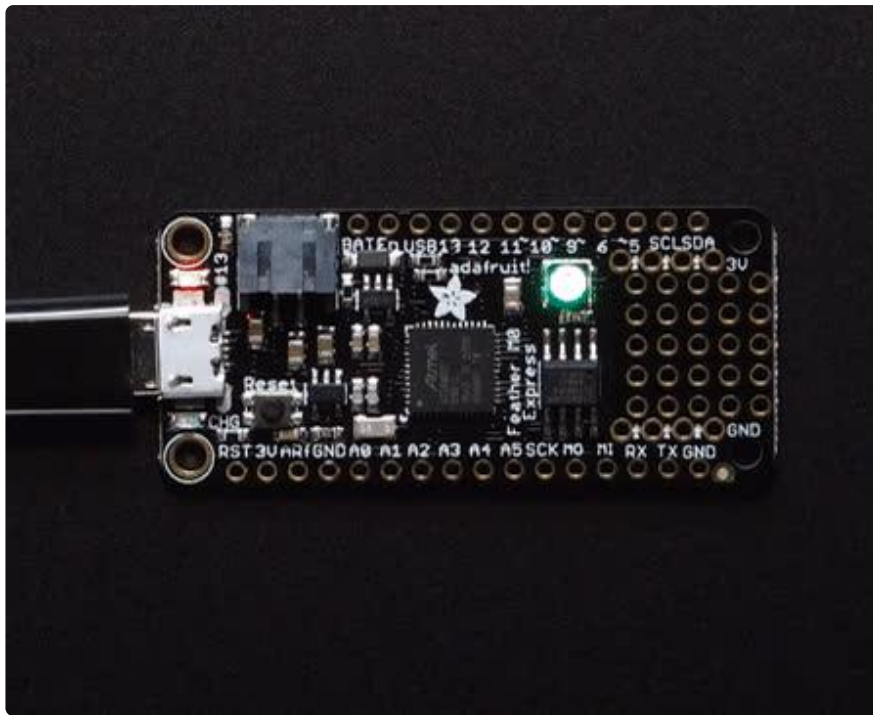


Entering Bootloader Mode

The first step to loading new code onto your board is triggering the bootloader. It is easily done by double tapping the reset button. Once the bootloader is active you will see the small red LED fade in and out and a new drive will appear on your computer with a name ending in BOOT. For example, feathers show up as FEATHERBOOT, while the new CircuitPlayground shows up as CPLAYBOOT, Trinket M0 will show up as TRINKETBOOT, and Gemma M0 will show up as GEMMABOOT

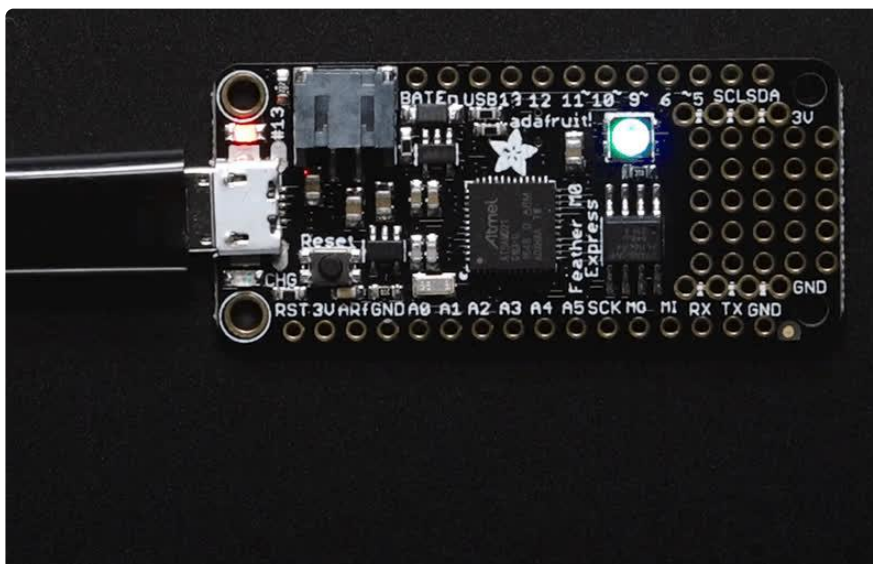
Furthermore, when the bootloader is active, it will change the color of one or more onboard neopixels to indicate the connection status, red for disconnected and green for connected. If the board is plugged in but still showing that its disconnected, try a different USB cable. Some cables only provide power with no communication.

For example, here is a Feather M0 Express running a colorful Neopixel swirl. When the reset button is double clicked (about half second between each click) the NeoPixel will stay green to let you know the bootloader is active. When the reset button is clicked once, the 'user program' (NeoPixel color swirl) restarts.

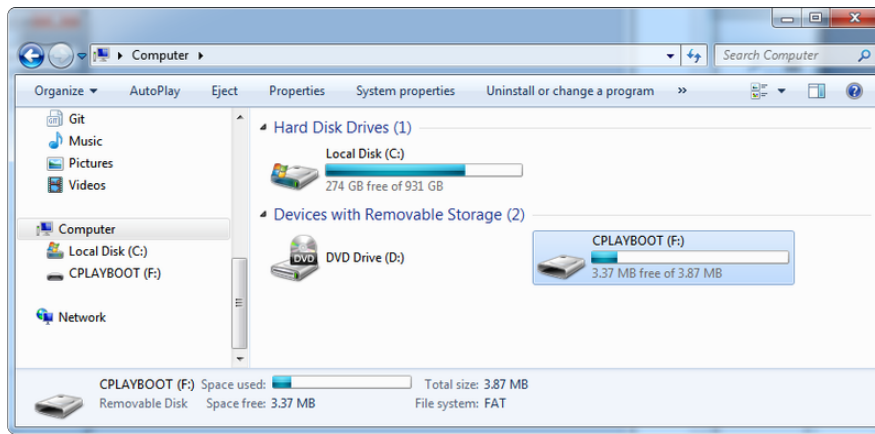


If the bootloader couldn't start, you will get a red NeoPixel LED.

That could mean that your USB cable is no good, it isn't connected to a computer, or maybe the drivers could not enumerate. Try a new USB cable first. Then try another port on your computer!

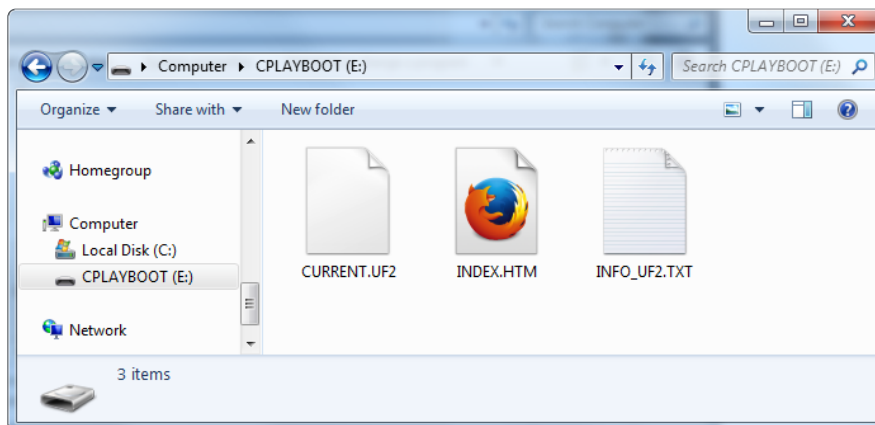


Once the bootloader is running, check your computer. You should see a USB Disk drive...



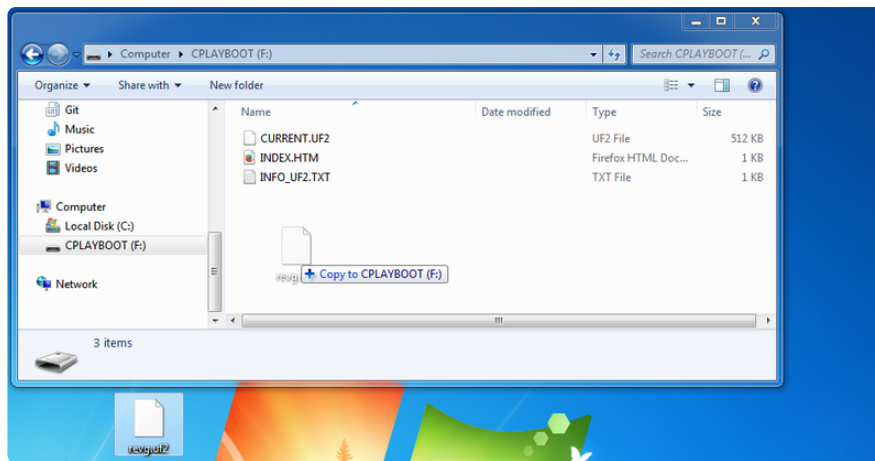
Once the bootloader is successfully connected you can open the drive and browse the virtual filesystem. This isn't the same filesystem as you use with CircuitPython or Arduino. It should have three files:

- CURRENT.UF2 - The current contents of the microcontroller flash.
- INDEX.HTM - Links to Microsoft MakeCode.
- INFO_UF2.TXT - Includes bootloader version info. Please include it on bug reports.

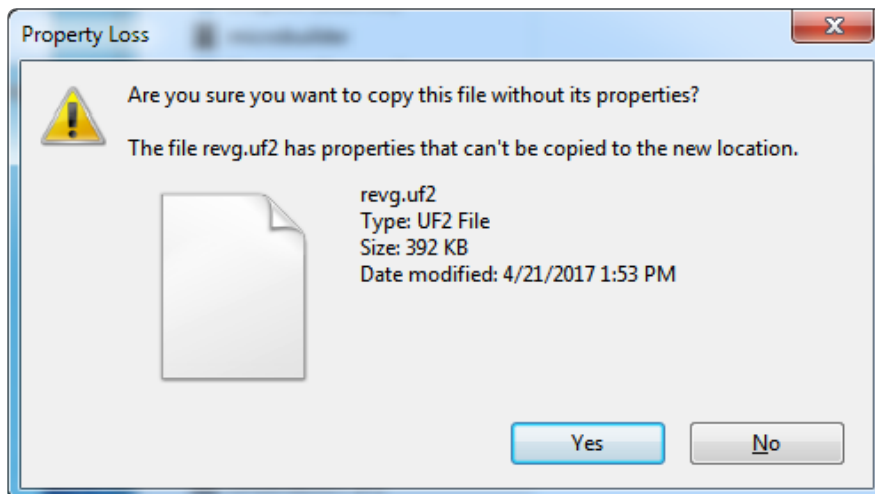


Using the Mass Storage Bootloader

To flash something new, simply drag any UF2 onto the drive. After the file is finished copying, the bootloader will automatically restart. This usually causes a warning about an unsafe eject of the drive. However, its not a problem. The bootloader knows when everything is copied successfully.



You may get an alert from the OS that the file is being copied without its properties. You can just click Yes



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

Using the BOSSA Bootloader

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

Windows 7 Drivers

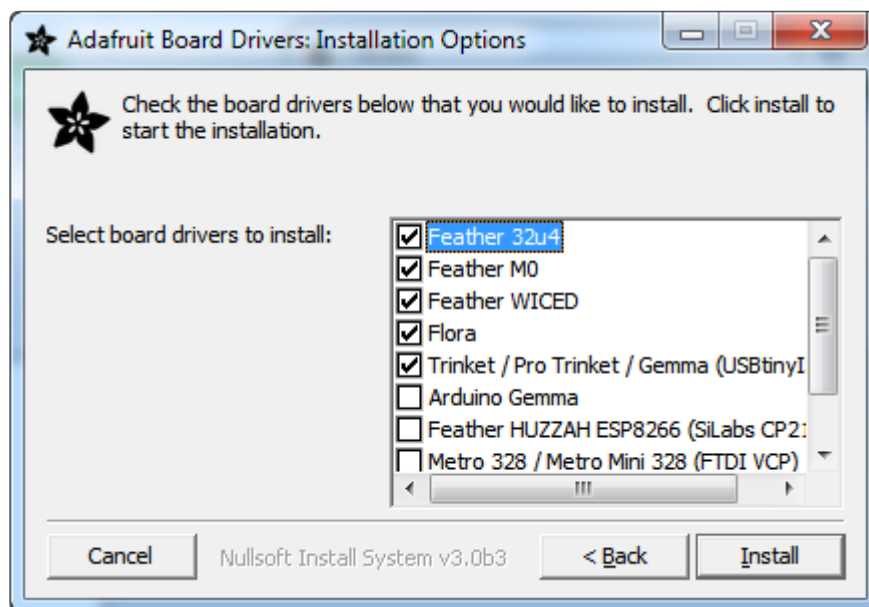
If you are running Windows 7 (or, goodness, something earlier?) You will need a Serial Port driver file. Windows 10 users do not need this so skip this step.

You can download our full driver package here:

Download Latest Adafruit Driver
Installer

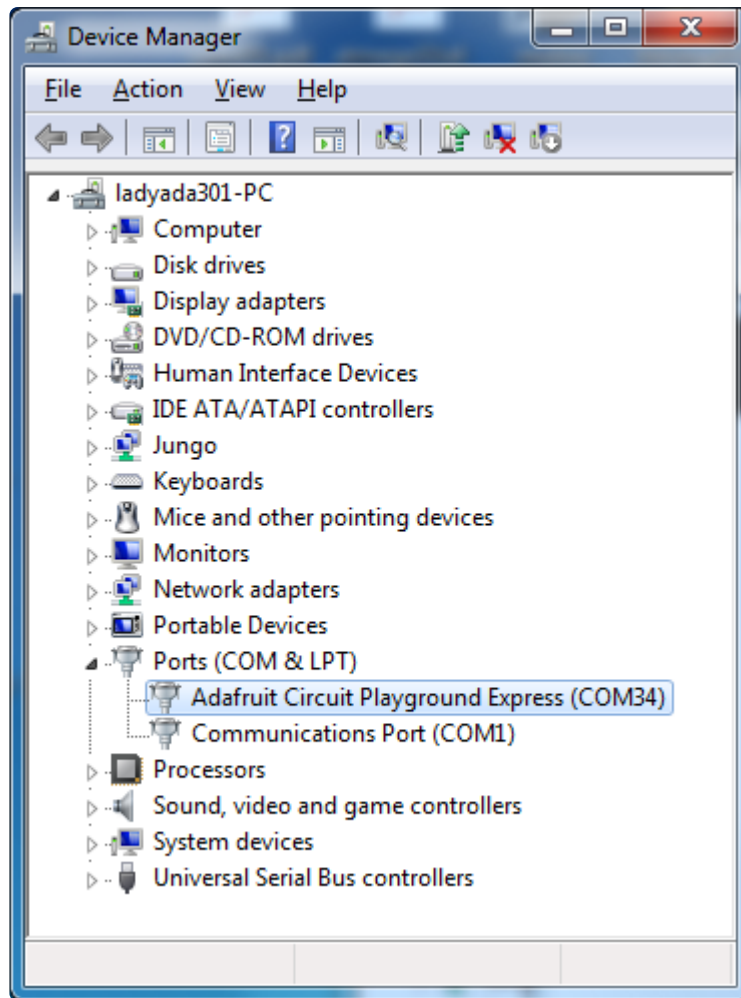
<https://adafru.it/AB0>

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

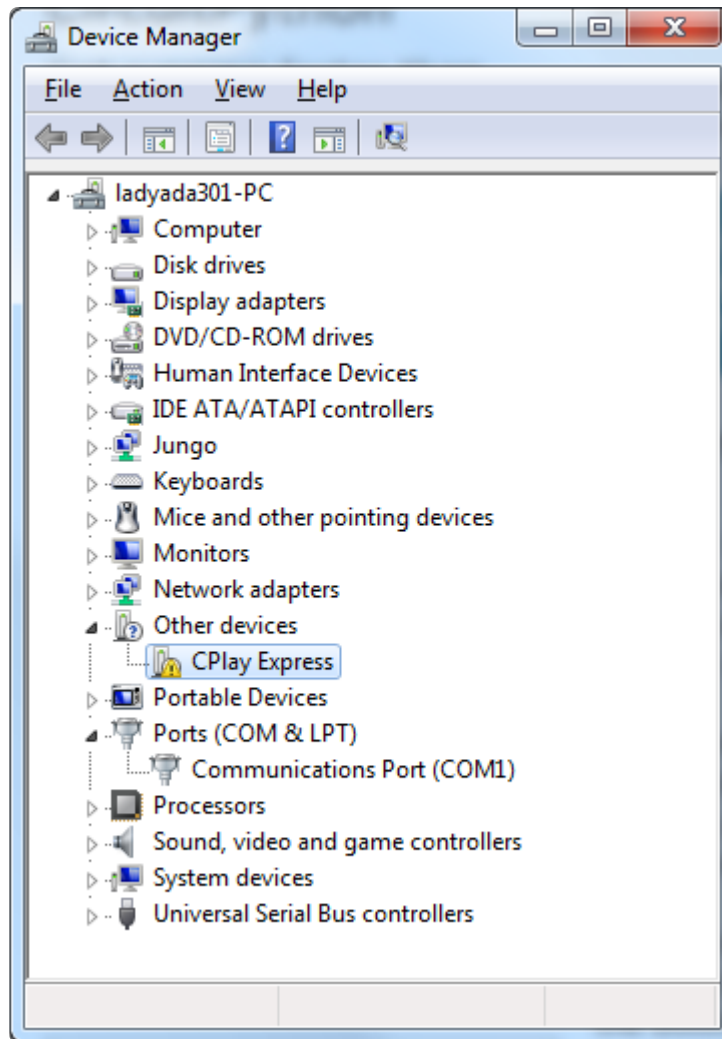


Verifying Serial Port in Device Manager

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



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



Running bossac on the command line

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

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

With bossac versions 1.9 or later, you must use the `--offset` parameter on the command line, and it must have the correct value for your board.

With bossac version 1.9 or later, you must give an `--offset` parameter on the command line to specify where to start writing the firmware in flash memory. This

parameter was added in bossac 1.8.0 with a default of `0x2000`, but starting in 1.9, the default offset was changed to `0x0000`, which is not what you want in most cases. If you omit the argument for bossac 1.9 or later, you will probably see a "Verify Failed" error from bossac. Remember to change the option for `-p` or `--port` to match the port on your Mac.

Replace the filename below with the name of your downloaded `.bin`: it will vary based on your board!

Using bossac Versions 1.7.0, 1.8

There is no `--offset` parameter available. Use a command line like this:

```
bossac -p=/dev/cu.usbmodem14301 -e -w -v -R adafruit-circuitpython-boardname-version.bin
```

For example,

```
bossac -p=/dev/cu.usbmodem14301 -e -w -v -R adafruit-circuitpython-feather_m0_express-3.0.0.bin
```

Using bossac Versions 1.9 or Later

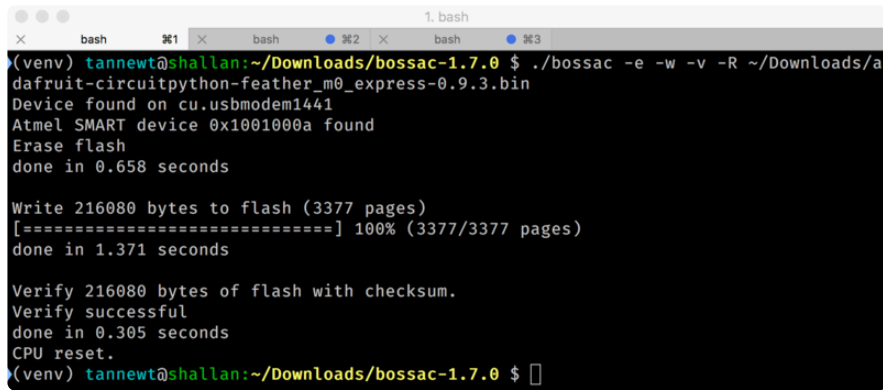
For M0 boards, which have an 8kB bootloader, you must specify `-offset=0x2000`, for example:

```
bossac -p=/dev/cu.usbmodem14301 -e -w -v -R --offset=0x2000 adafruit-circuitpython-feather_m0_express-3.0.0.bin
```

For M4 boards, which have a 16kB bootloader, you must specify `-offset=0x4000`, for example:

```
bossac -p=/dev/cu.usbmodem14301 -e -w -v -R --offset=0x4000 adafruit-circuitpython-feather_m4_express-3.0.0.bin
```

This will **e**rase the chip, **w**rite the given file, **v**erify the write and **R**eset the board. On Linux or MacOS you may need to run this command with `sudo ./bossac ...`, or add yourself to the dialout group first.



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

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

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

Updating the bootloader

The UF2 bootloader is relatively new and while we've done a ton of testing, it may contain bugs. Usually these bugs effect reliability rather than fully preventing the bootloader from working. If the bootloader is flaky then you can try updating the bootloader itself to potentially improve reliability.

If you're using MakeCode on a Mac, you need to make sure to upload the bootloader to avoid a serious problem with newer versions of MacOS. See instructions and more details [here](https://adafru.it/ECU) (<https://adafru.it/ECU>).

In general, you shouldn't have to update the bootloader! If you do think you're having bootloader related issues, please post in the forums or discord.

Updating the bootloader is as easy as flashing CircuitPython, Arduino or MakeCode. Simply enter the bootloader as above and then drag the update bootloader uf2 file below. This uf2 contains a program which will unlock the bootloader section, update the bootloader, and re-lock it. It will overwrite your existing code such as CircuitPython or Arduino so make sure everything is backed up!

After the file is copied over, the bootloader will be updated and appear again. The INFO_UF2.TXT file should show the newer version number inside.

For example:

```
UF2 Bootloader v2.0.0-adafruit.5 SFHWR0
Model: Metro M0
Board-ID: SAMD21G18A-Metro-v0
```

Lastly, reload your code from Arduino or MakeCode or flash the [latest CircuitPython core](https://adafru.it/Em8) (<https://adafru.it/Em8>).

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

Circuit Playground Express V3.7.0
update-bootloader.uf2

<https://adafru.it/JcN>

Feather M0 Express v3.7.0 update-
bootloader.uf2

<https://adafru.it/JcO>

Metro M0 Express v3.7.0 update-
bootloader.uf2

<https://adafru.it/JcR>

Gemma M0 v3.7.0 update-
bootloader.uf2

<https://adafru.it/JcU>

Trinket M0 v3.7.0 update-
bootloader.uf2

<https://adafru.it/JcX>

Itsy Bitsy M0 v3.7.0 update-
bootloader.uf2

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

Grand Central M4 v3.7.0 update-
bootloader.uf2

<https://adafru.it/Jd2>

Latest version of update-
bootloader.uf2 for other boards.
Make sure you pick the right one.

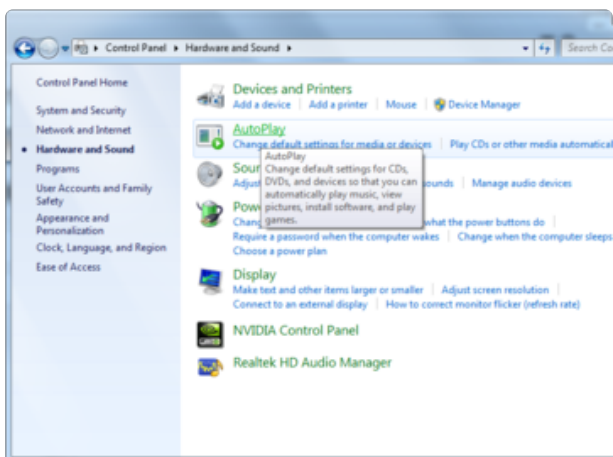
<https://adafru.it/Bmg>

Getting Rid of Windows Pop-ups

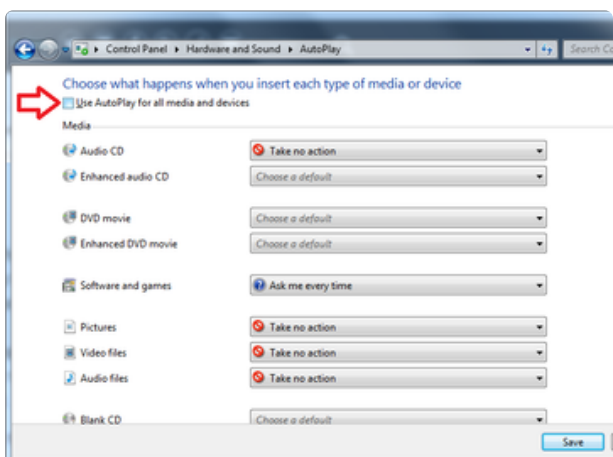
If you do a lot of development on Windows with the UF2 bootloader, you may get annoyed by the constant "Hey you inserted a drive what do you want to do" pop-ups.



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



Click on the AutoPlay header



Uncheck the box at the top, labeled Use Autoplay for all devices

Making your own UF2

Making your own UF2 is easy! All you need is a .bin file of a program you wish to flash and [the Python conversion script \(https://adafru.it/vZb\)](https://adafru.it/vZb). Make sure that your program was compiled to start at 0x2000 (8k) for M0 boards or 0x4000 (16kB) for M4 boards. The bootloader takes up the first 8kB (M0) or 16kB (M4). CircuitPython's [linker script \(https://adafru.it/CXh\)](https://adafru.it/CXh) is an example on how to do that.

Once you have a .bin file, you simply need to run the Python conversion script over it. Here is an example from the directory with uf2conv.py. This command will produce a firmware.uf2 file in the same directory as the source firmware.bin. The uf2 can then be flashed in the same way as above.

```
# For programs with 0x2000 offset (default)
uf2conv.py -c -o build-circuitplayground_express/firmware.uf2 build-
circuitplayground_express/firmware.bin

# For programs needing 0x4000 offset (M4 boards)
uf2conv.py -c -b 0x4000 -o build-metro_m4_express/firmware.uf2 build-
metro_M4_express/firmware.bin
```

Installing the bootloader on a fresh/bricked board

If you somehow damaged your bootloader or maybe you have a new board, you can use a JLink to re-install it.

[Here's a Learn Guide explaining how to fix the bootloader on a variety of boards using Atmel Studio \(https://adafru.it/F5f\)](https://adafru.it/F5f)

[Here's a short writeup by turbinenreiter on how to do it for the Feather M4 \(but adaptable to other boards\) \(https://adafru.it/ven\)](https://adafru.it/ven)

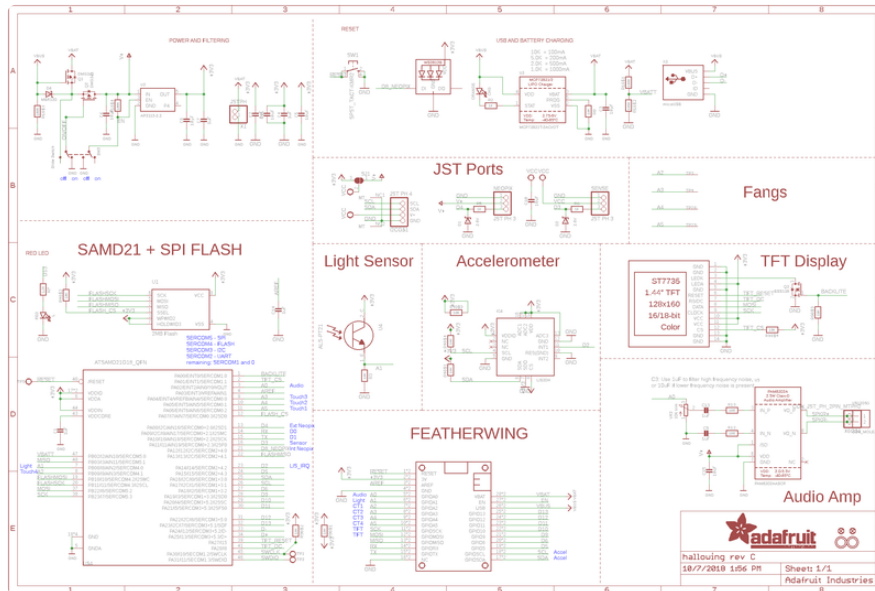
Downloads

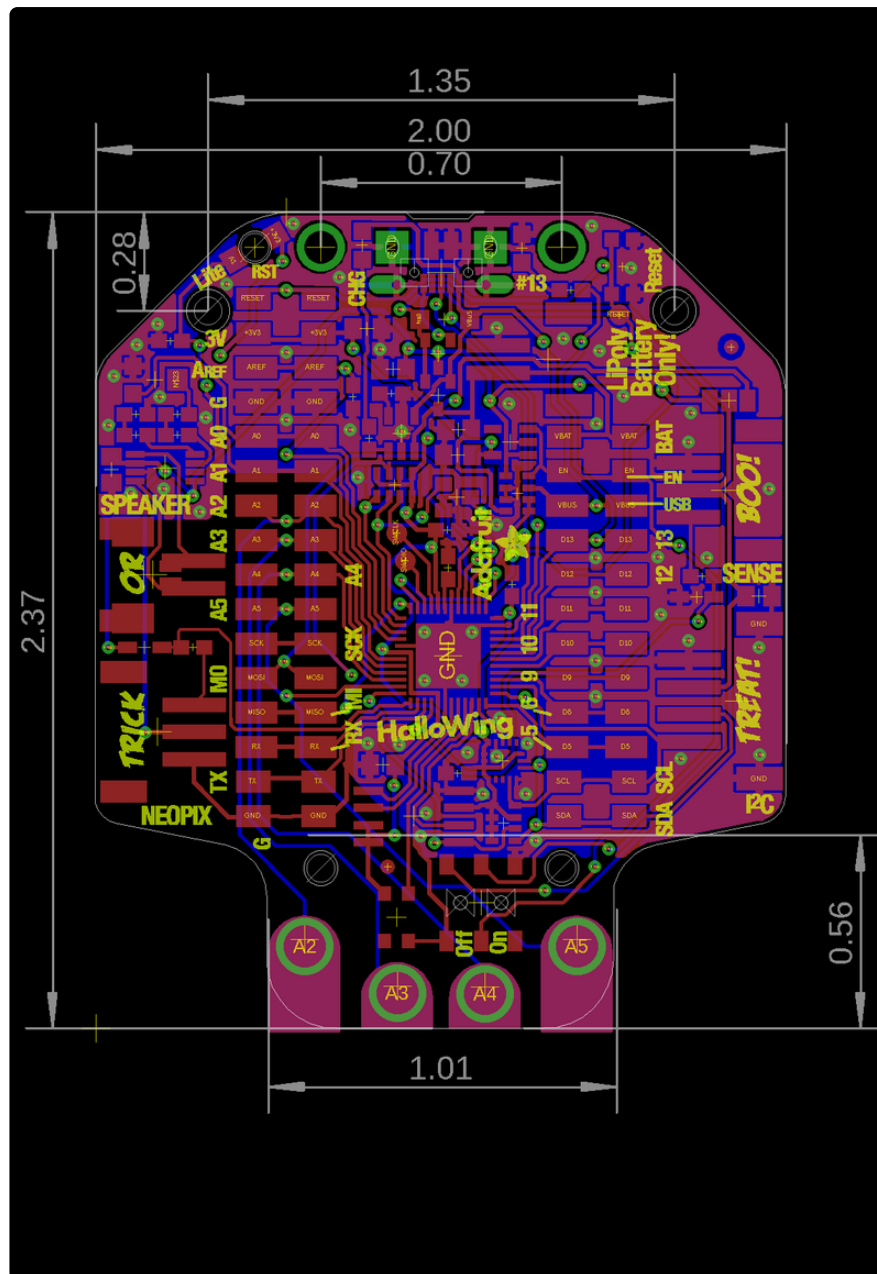
Files

- [LIS3DH datasheet \(https://adafru.it/jwe\)](https://adafru.it/jwe)
- [LIS3DH app note \(https://adafru.it/jwf\)](https://adafru.it/jwf)
- [ST7735R display driver datasheet \(https://adafru.it/aP9\)](https://adafru.it/aP9)

- [Raw 1.44" TFT Datasheet \(https://adafru.it/dYA\)](https://adafru.it/dYA)
- [EagleCAD PCB files on GitHub \(https://adafru.it/C8t\)](https://adafru.it/C8t)
- [Fritzing object in Adafruit Fritzing Library \(https://adafru.it/aP3\)](https://adafru.it/aP3)

Schematic & Fabrication Print





Troubleshooting

TFT Screen Adhesive

The TFT screen on the HalloWing can become un-adhered if it is bumped or wiggled too much (or left in a very hot Jeep near the windshield in the hot southern California sun, for a totally hypothetical example that didn't necessarily happen to the author). Fixing this is pretty easy -- you can use double-stick tape, E6000 glue, or Sugru to fix it back in place. Here are some action photos of these fixes.





Double Stick Tape

Cut three thin slices of double stick tape, then place them around the edges of the backlight.

Press the screen down and hold for a few seconds to adhere.



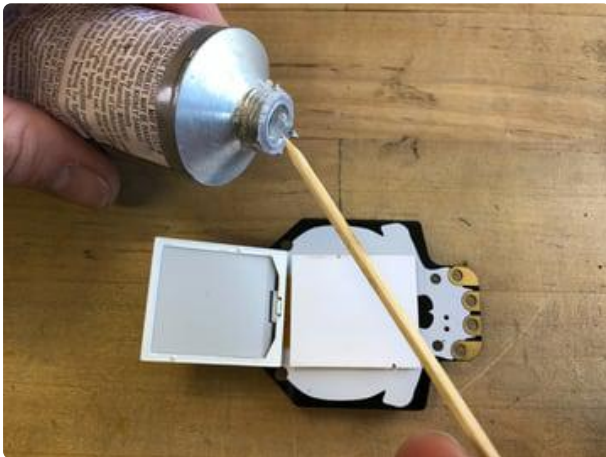
E6000 Glue

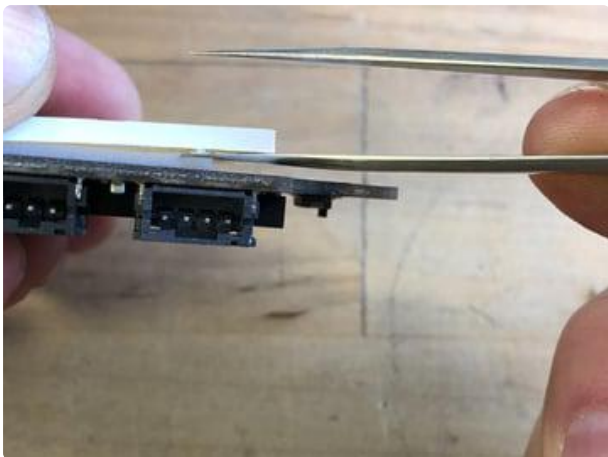
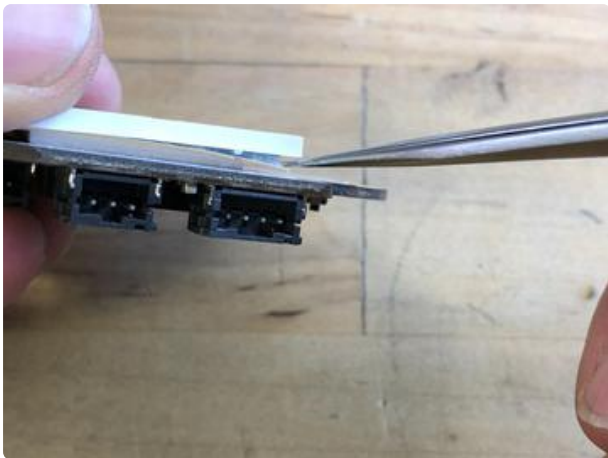
This is strong glue and is even better than tape if you plan to leave the HalloWing screen exposed on a costume without the lens and cover.



Use a toothpick or skewer to place small dabs of the glue around the perimeter of the backlight.

Press the screen down and hold for a few seconds, then allow to cure overnight.





Sugru

You can go all out and create a protective frame using [Sugru](https://adafruit.it/ekR) (<https://adafruit.it/ekR>) moldable silicone rubber.



Open a sachet of Sugru and tear off a small ball of it.

Roll the ball into a small cylinder and press up against one side of the board and screen.



Repeat for the other side.

Allow Sugru to cure overnight.





Diagnostics

Want to see some stats on your HalloWing? Double-click the board's reset button to get to bootloader mode, then drag the HallowWingM4_Diagnostics.UF2 onto the HALLOWBOOT drive!

This UF2 is for the Hallowing M4

HALLOWING M4 DIAGNOSTIC.UF2

<https://adafru.it/G6F>

If you have the Hallowing M0 use this UF2

hallowing m0 test.UF2

<https://adafru.it/QDT>

Mouser Electronics

Authorized Distributor

Click to View Pricing, Inventory, Delivery & Lifecycle Information:

[Adafruit:](#)

[3900](#)