Adafruit MagTag

Created by Kattni Rembor

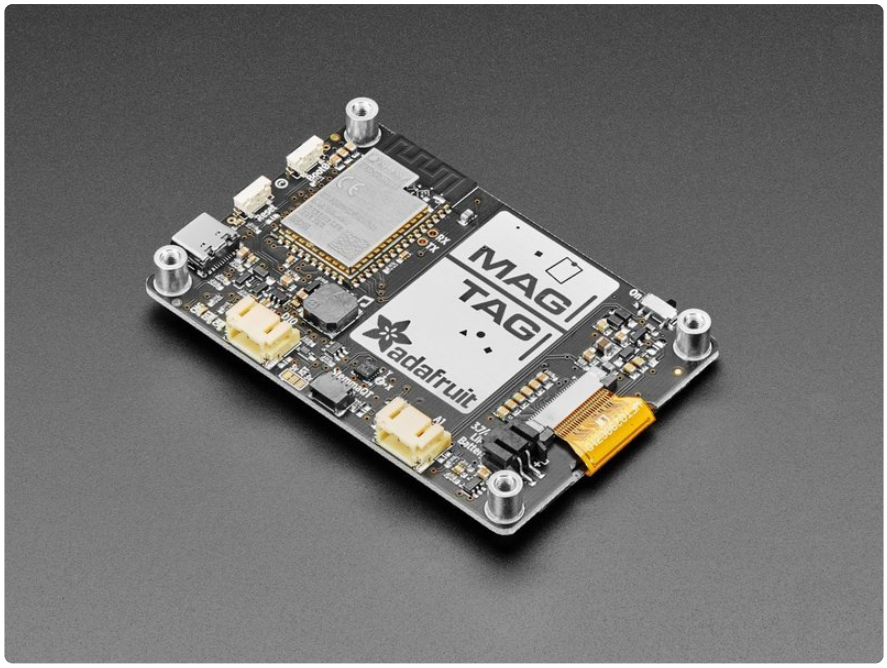
https://learn.adafruit.com/adafruit-magtag

Last updated on 2025-01-13 11:15:05 PM EST

Table of Contents

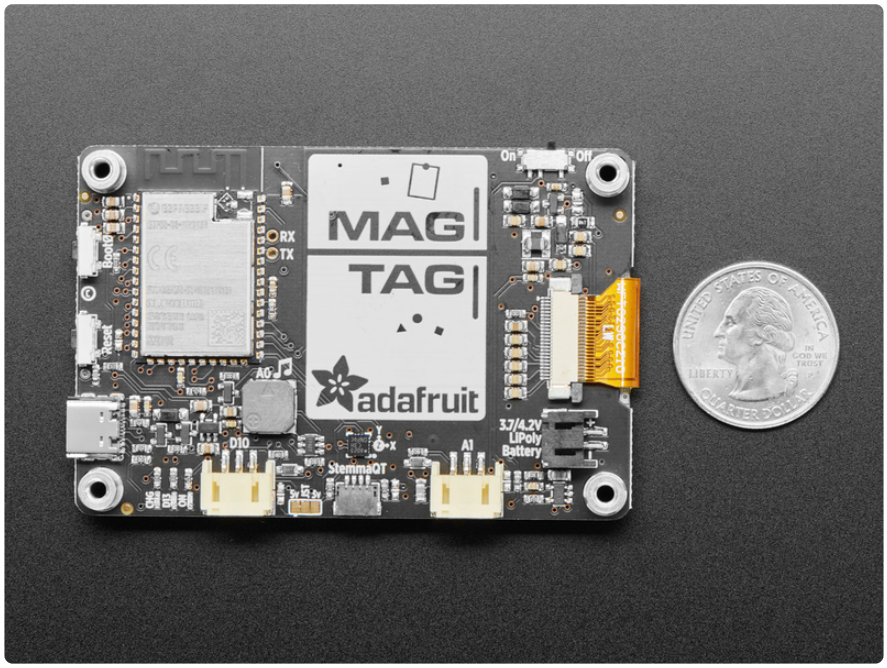
Overview

The Adafruit MagTag combines the new ESP32-S2 wireless module and a 2.9" grayscale E-Ink display to make a low-power IoT display that can show data on its screen even when power is removed! The ESP32-S2 is great because it builds on the years of code and support for the ESP32 and also adds native USB support so you can use this board with Arduino or CircuitPython!

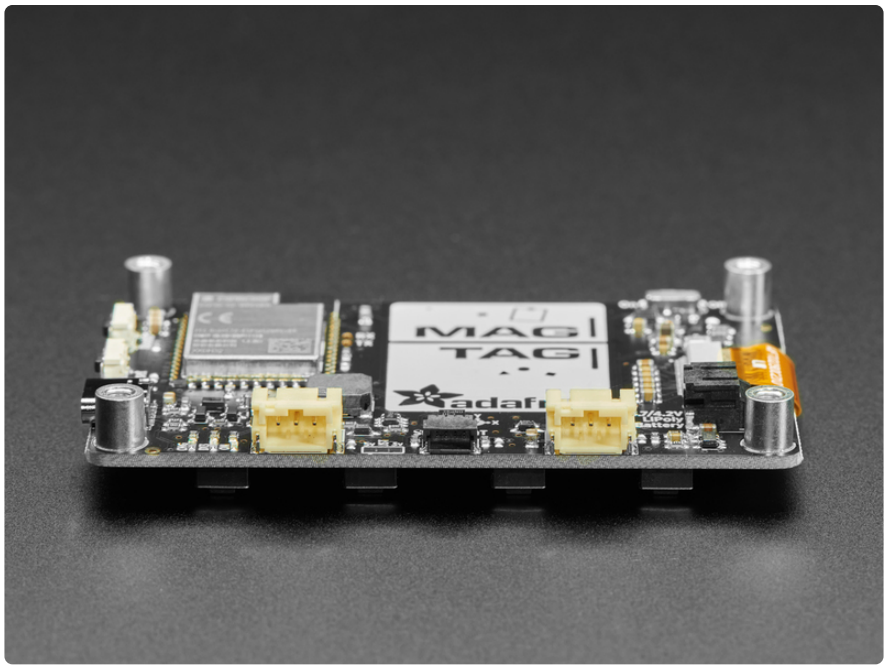


We designed this board to be low-power friendly - with a spot for a 350 or 420 mAh battery and built in battery charging over USB C. During deep sleep, with the

NeoPixels and speaker amplifier disabled, we measured 250uA power draw so you can run for a few weeks between charges.



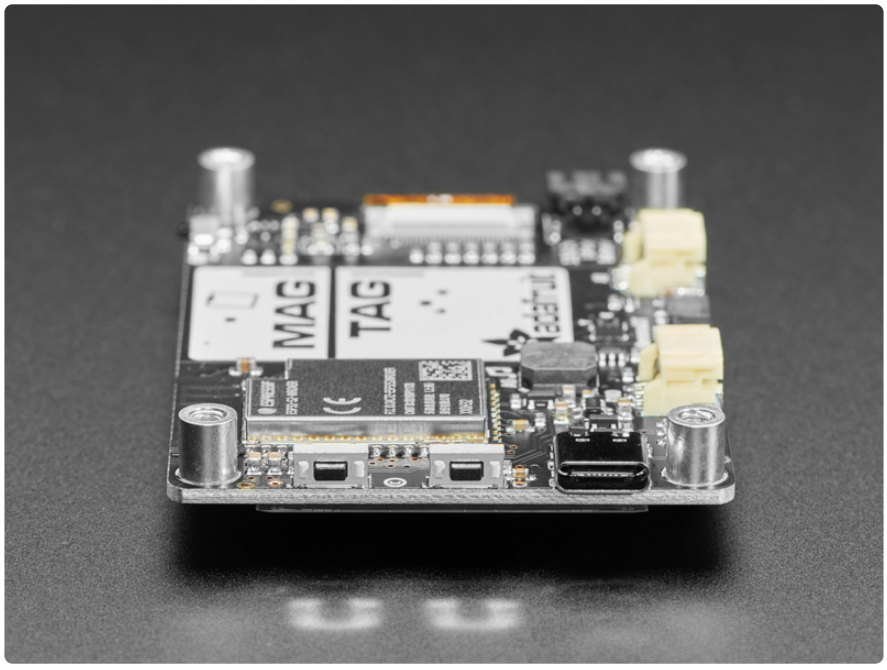
And of course, the Mag in MagTag stands for magnetic. We have four M3 standoffs that will work perfectly with these mini magnet feet. (http://adafru.it/4631) (Originally they're designed for RGB Matrices but they'll do an excellent job here as well). Screw on the feet and you can attach this display to a metallic shelf, fridge, or bench.



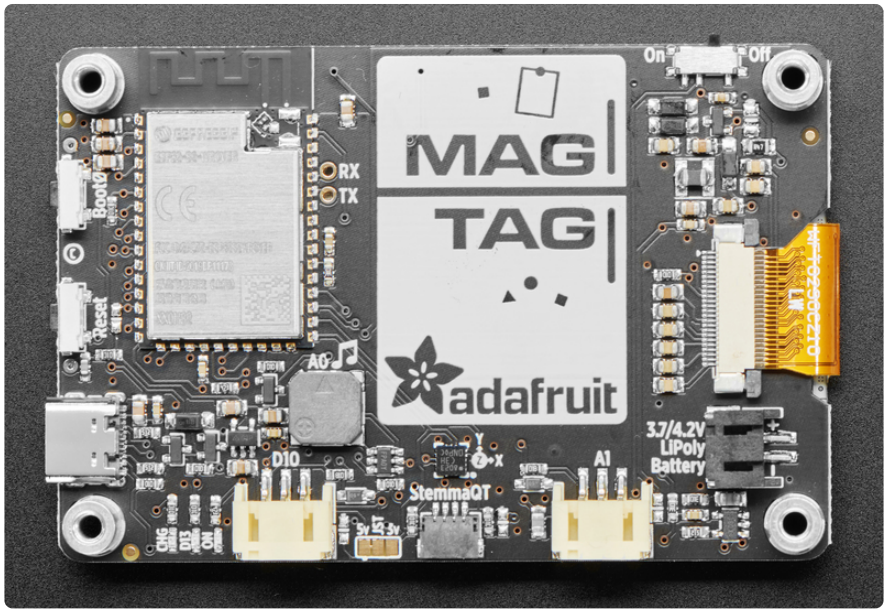
Here's the cool hardware we put together:

- **ESP32-S2 240MHz Tensilica processor** - the next generation of ESP32, now with native USB so it can act like a keyboard/mouse, MIDI device, disk drive, etc!

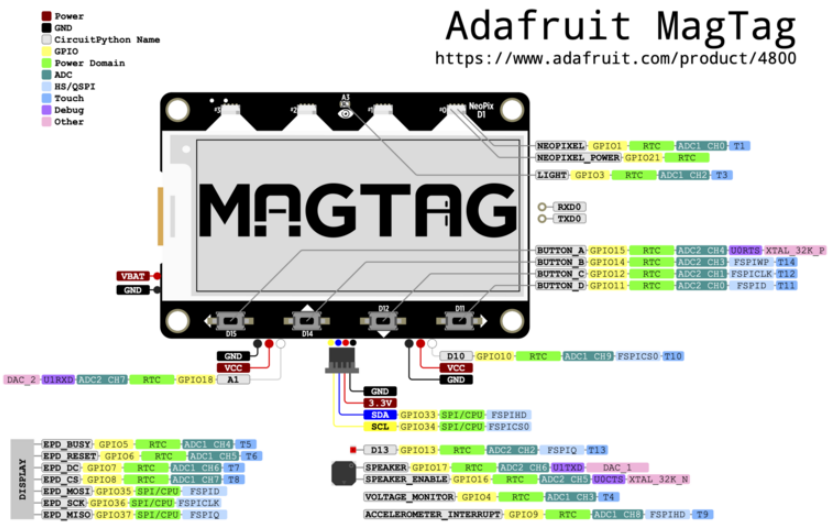
- **WROVER module** has FCC/CE certification and comes with 4 MByte of Flash and 2 MByte of PSRAM - you can have huge data buffers
- **2.9" grayscale display with 296x128 pixels**. Each pixel can be white, light gray, dark gray or black. Compared to 'tri-color' displays with a red pigment, this display takes a lot less time to update, only about a second instead of 15 seconds!
- **USB C** power and data connector
- **Four RGB side-emitting NeoPixels** so you can light up the display with any color or pattern
- **Four buttons** can be used to wake up the ESP32 from deep-sleep, or select different modes
- **Triple-axis accelerometer** (LIS3DH) can be used to detect orientation of the display
- **Speaker/Buzzer** with mini class D amplifier on DAC output A0 can play tones or lo-fi audio clips.
- Front facing **light sensor**
- **STEMMA QT** port for attaching all sorts of I2C devices (https://adafru.it/HMF)
- Two **STEMMA 3 pin JST** connectors for attaching NeoPixels (http://adafru.it/3919), speakers (http://adafru.it/3885), servos (http://adafru.it/4326) or relays (http://adafru.it/4409).
- **On/Off switch**
- **Boot** and **Reset buttons** for re-programming



Pinouts



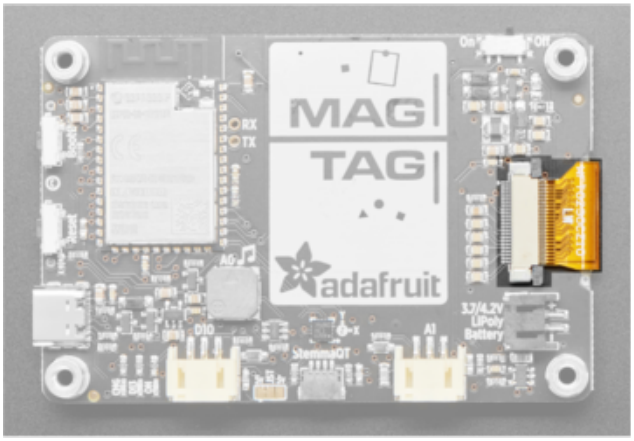
The MagTag has a great eInk display. It's also packed with buttons, connectors and sensors. Time to take a tour!



Click here to view a PDF version of the pinout diagram (https://adafru.it/ZqF)

Note that some elements of the MagTag, like the light sensor, NeoPixels, and speaker, require enabling before usage. This is because we optimized for low-power usage. If the light sensor, NeoPixels or speaker aren't working, see below for the enabling pins!

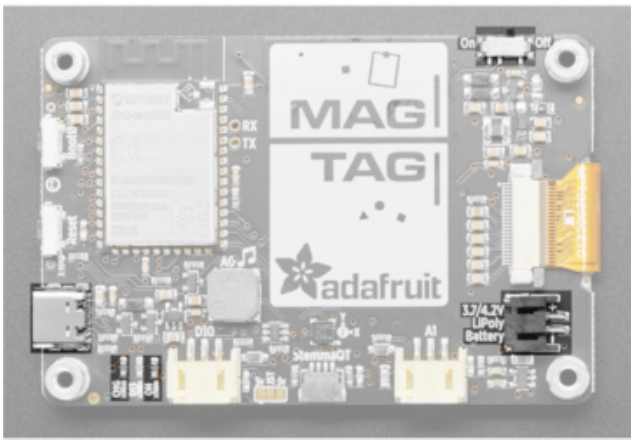
eInk Display and Display Connector



Front and center is a **2.9" grayscale eInk display with 296x128 pixels**. Each pixel can be white, light gray, dark gray or black. For use with general Arduino libraries, you can try GxEPD2_290_T5 - the Arduino library Adafruit uses is ThinkInk_290_Grayscale4_T5.

On the back, along the right side, the display cable wraps around the board to the **display connector** on the back.

Power



There are two ways to power the MagTag board: the USB type C connector or a 3.7/4.2V Lipoly battery.

Power Inputs

- **USB C port** - This is used for both powering and programming the board. You can power it with any USB C cable and will request 5V from a USB C PD.

When USB is plugged in it will charge the Lipoly battery. If there is no battery attached, the yellow LED will flicker (it's looking for a battery!)

- **LiPoly connector/charger** - You can plug in any 250mAh or larger 3.7/4.2V Lipoly battery into this **JST 2-PH port** to both power your MagTag and charge the battery. The battery will charge from the USB, even if the board is powered off via the switch.
  If the battery is plugged in and USB is plugged in, the MagTag will power itself from USB and it will charge the battery up.
  When the battery is charging, the yellow CHG LED will be lit. When charging is complete, the LED will turn off.

Power Control

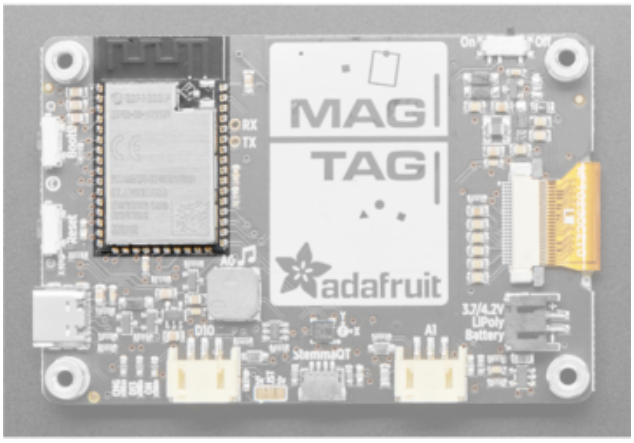
- **On/Off switch** - This switch controls power to the board. If you plug in your board and nothing happens, make sure the switch is flipped to "ON"!

The on-off switch is not very rugged - when switching, push at the 'bottom' of the actuator, not the tip so you don't 'twist' the switch, and don't flick it back and forth. It's not necessary for most uses!

Power LEDs

- **OK LED** - This green LED indicates the board is powered on, it is connected to the 3.3V power supply. This LED draws 40uA. If you need lower power you can remove this LED with a soldering iron.
- **CHG LED** - This yellow LED lets you know when the plugged in battery is charging and when it's fully charged. It's normal for this LED to flicker when no battery is in place, that's the charge circuitry trying to detect whether a battery is there or not.

ESP32-S2 WiFi Module



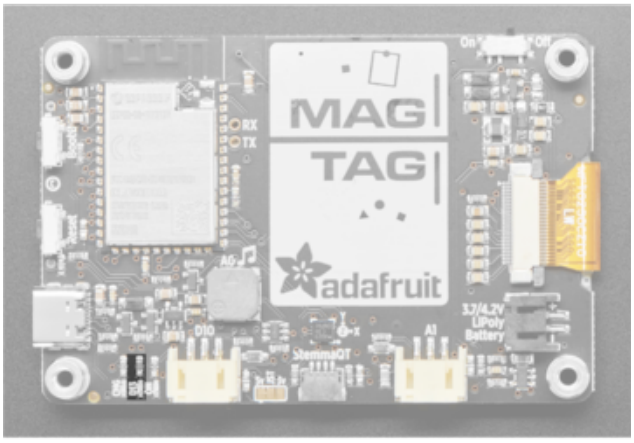
The **ESP32-S2 WROVER module**.

The ESP32-S2 is a highly-integrated, low-power, 2.4 GHz Wi-Fi System-on-Chip (SoC) solution that now has **built-in native USB** as well as some other interesting new technologies like Time of Flight distance measurements. With its state-of-the-art power and RF performance, this SoC is an ideal choice for a wide variety of application scenarios relating to the Internet of Things (IoT) (https://adafru.it/Bwq), wearable electronics (https://adafru.it/Osb), and smart homes.

Please note, this is a single-core 240 MHz chip so it won't be as fast as ESP32's with dual-core. Also, there is no Bluetooth support. However, we are super excited about the ESP32-S2's native USB which unlocks a lot of capabilities for advanced interfacing! **This WROVER module comes with 4 MB flash and 2 MB PSRAM.**

The 4 MB of flash is inside the module and is used for **both** program firmware and filesystem storage. For example, in CircuitPython, we have 3 MB set aside for program firmware (this includes two OTA option spots as well) and a 1MB section for CircuitPython scripts and files.

NeoPixels and Red LED

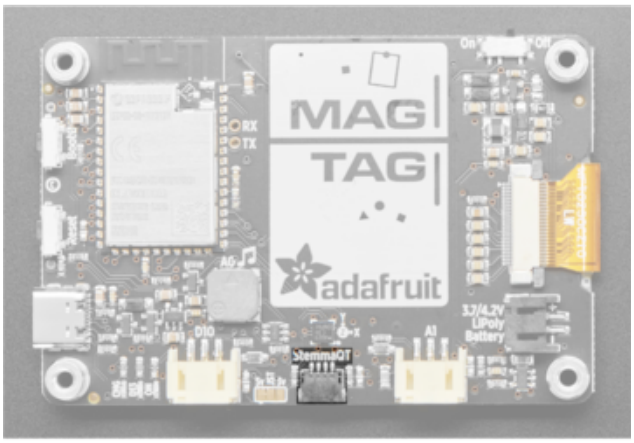


NeoPixel LEDs and red LED.

On the front of the board, along the top, are **four addressable RGB side-emitting NeoPixel LEDs** labeled together as **NeoPix D1**, and individually labeled **#0**, **#1**, **#2**, and **#3**, so you can light up the display with any color or pattern. You can use GPIO 1 to control the NeoPixels.
On the back, on the bottom left, is a **red LED** labeled **D13**. It is user-controllable for blinky needs. You can blink this at any time.

To use the NeoPixel LEDs you must also set pin 21 to be an output and LOW - this is the NeoPixel power pin. If not using the NeoPixels, keep pin 21 as an input or HIGH output - that will remove the quiescent power usage of the NeoPixels so you can have lower power in sleep mode.

STEMMA QT



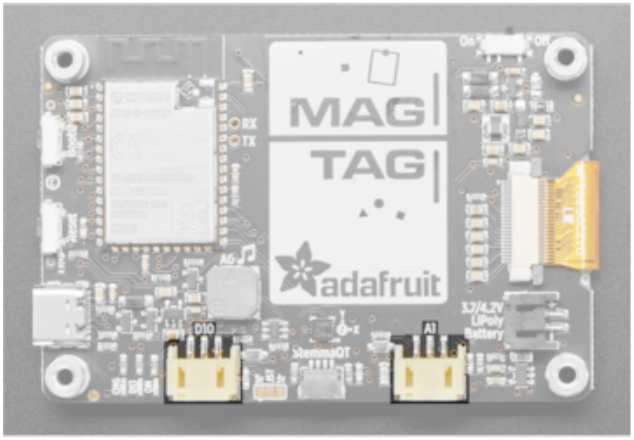
STEMMA QT **(https://adafru.it/Ft4)** - This JST SH 4-pin connector breaks out I2C (SCL, SDA, 3.3V, GND). It allows you to connect to various breakouts and sensors with **STEMMA QT** connectors (https://adafru.it/HMF) or to other things using assorted associated accessories (https://adafru.it/Ft6).

In CircuitPython, you can use the STEMMA connector with `board.SCL` and `board.SDA`, or `board.STEMMA_I2C()`.

Works great with any STEMMA QT or Qwiic sensor/device

You can also use it with Grove I2C devices thanks to this handy cable (http://adafru.it/4528)

Digital/Analog Connectors



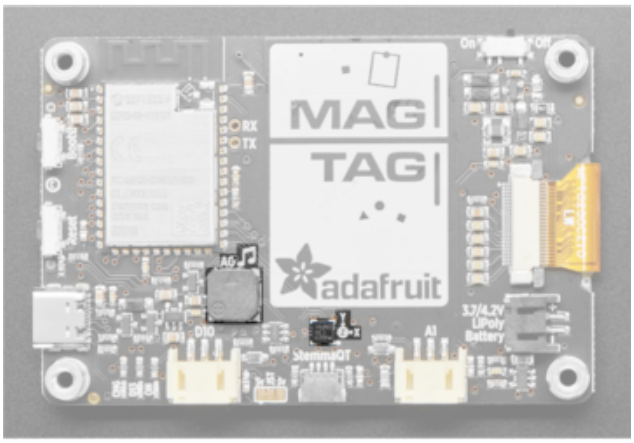
On the bottom are two connectors labeled D10 and A1. These are **STEMMA 3 pin JST digital or analog connectors** for attaching NeoPixels (http://adafru.it/3919), speakers (http://adafru.it/3885), servos (http://adafru.it/4326) or relays (http://adafru.it/4409). These pins can be analog inputs or digital I/O.

Both connectors have protection 1K resistors + 3.6V zener diodes so you can drive an LED directly from the output. The maximum current from these connectors is 200mA.

A1 is a 'true' analog output. Both can be used for PWM as well as analog inputs. The maximum input voltage is 2.6V, after which the zener diode will kick in to drain excess voltage.

The power output is 5V by default, but a jumper can be cut/soldered to change it to 3.3V.

Speaker and Sensors



Towards the middle of the board, at the bottom right corner of the ESP32-S2 module, is a **speaker/buzzer** labeled with **A0** and a musical note. This includes a mini class D amplifier on DAC output A0 and can play tones or lo-fi audio clips.

In the center of the board, towards the bottom, is an **LIS3DH accelerometer** labeled with X, Y and Z, that can be used to detect the orientation of the display. It is connected to the I2C port and available on I2C address 0x19. The IRQ line is connected to GPIO 9.
On the front of the board, in the center of the top is a front-facing **light sensor** labeled with **A3** and an eye.

Note that **to use the speaker you must also set the speaker shutdown pin IO 16 to be an output and HIGH**. If not playing audio, keep this pin as an input or LOW, to reduce the power usage of the amplifier circuit.

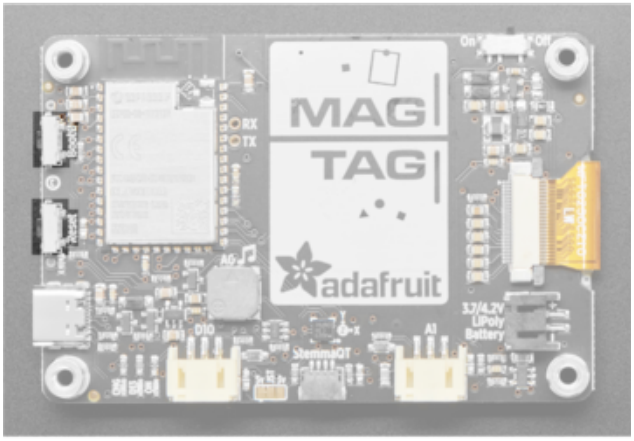
To use the light sensor you must also set pin 21 to be an output and LOW to turn on the power supply to it and the NeoPixels. If not using the light sensor or NeoPixels, keep pin 21 as an input or HIGH output - that will remove the quiescent power usage of the NeoPixels and light sensor so you can have lower power in sleep mode.

Buttons

On the front of the board, along the bottom, there are **four user-controllable buttons** labeled both with pin names and a different arrow for each button. The buttons are on **D11**, **D12**, **D14** and **D15**. Can be used to wake up the ESP32-S2 from deep-sleep, or select different modes.

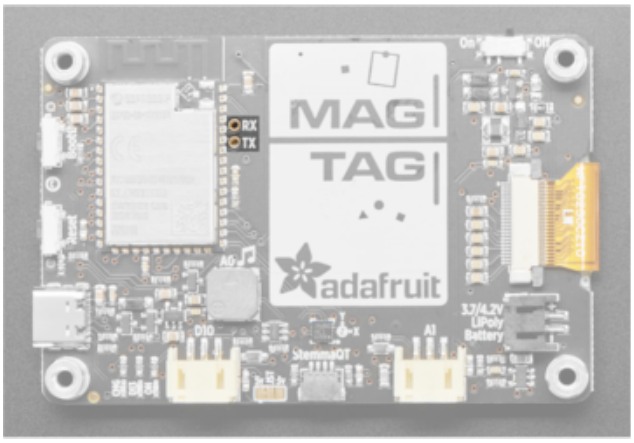
There are no pull-ups on board, use internal pullups for this pins - when the buttons are pressed the IO pin labeled is set to LOW

Reset and Boot0



Reset button - The reset button in the top left corner is used to reset the board. **Boot0 button** - This is connected to **BOOT0** and can be used to put the board into ROM bootloader mode. To enter ROM bootloader mode, **hold down the BOOT0 button while clicking reset button mentioned above.** When in the ROM bootloader, you can upload code and query the chip using `esptool`.

UART Debug



The hardware UART debug port has two broken out pins. You can connect these to a USB console cable in order to read the debug output from the ESP32 IDF (http://adafru.it/954). This is useful if you are writing software and need to see the low level debug output.

This is not where default `Serial.print()` or CircuitPython `print()` outputs go, because those will go through the USB port instead!

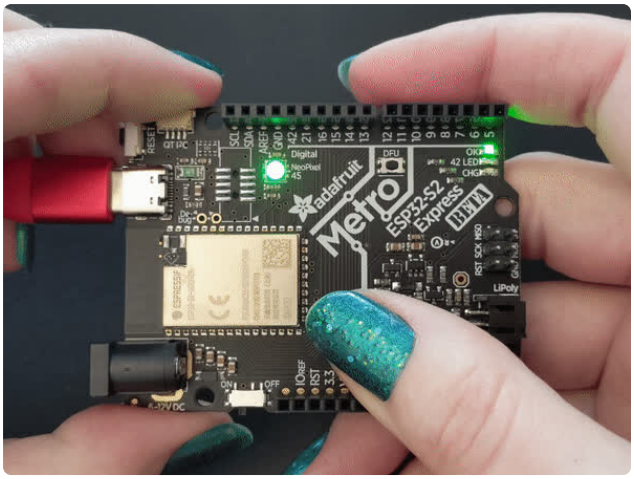
ROM Bootloader

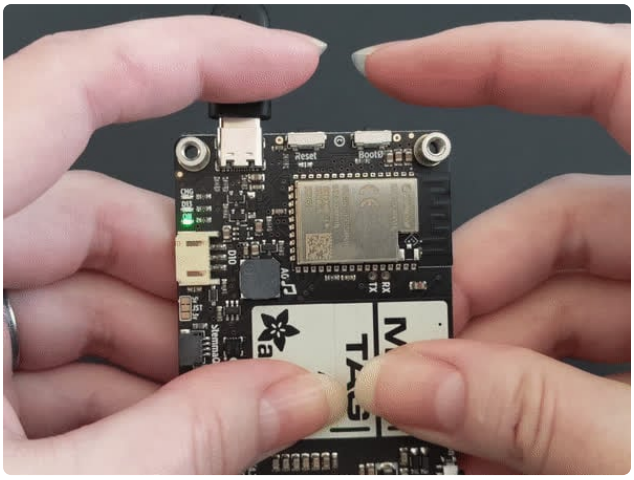
The ESP32-S2 has a built in bootloader, which means you never have to worry about 'bricking' your board. You can use it to load code directly, say CircuitPython or the binary output of an Arduino compilation or you can use it to load a second bootloader on, like UF2 which has a drag-n-drop interface.

The ROM bootloader can never be disabled or erased, so its always there if you need it!

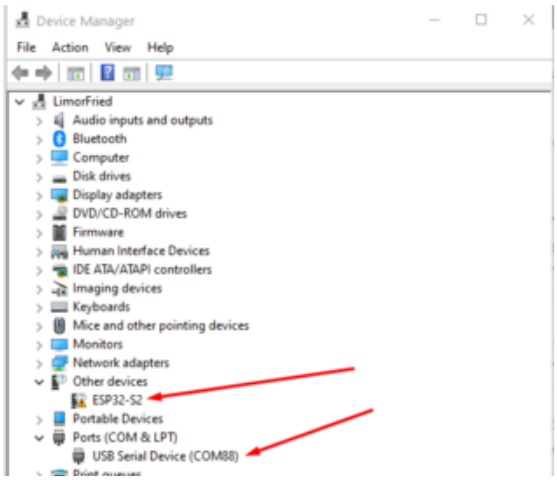
Enter ROM Bootloader Mode

Entering the bootloader is easy. Complete the following steps.



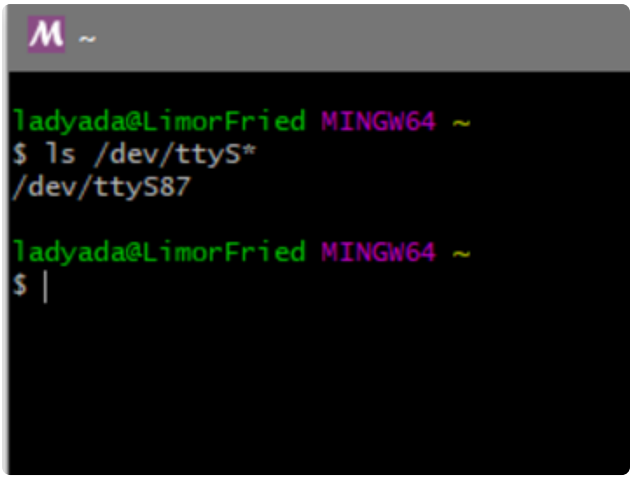


1. **Make sure your ESP32-S2 is plugged into USB port to your computer using a data/sync cable.** Charge-only cables will not work!
2. **Turn on the On/Off switch** - check that you see the OK light on so you know the board is powered, a prerequisite!
3. **Press and hold the DFU / Boot0 button down.** Don't let go of it yet!
4. **Press and release the Reset button.** You should have the DFU/Boot0 button pressed while you do this.
5. **Now you can release the DFU / Boot0 button**
6. **Check your computer for a new serial / COM port. On windows check the Device manager**



On Windows check the Device manager - you will see a COM port, for example here its COM88. You may also see another "Other device" called ESP32-S2

It's best to do this with no other dev boards plugged in so you don't get confused about which COM port is the ESP32-S2



On Mac/Linux you will need to find the tty name which lives under /dev

On Linux, try **ls /dev/ttyS*** for example, to find the matching serial port name. In this case it shows up as **/dev/ttyS87**. If you don't see it listed try **ls /dev/ttyA*** on some Linux systems it might show up like **/dev/ttyACM0**

On Mac, try **ls /dev/cu.usbmodem*** for example, to find the matching serial port name. In this case, it shows up as **/dev/cu.usbmodem01**

It's best to do this with no other dev boards plugged in so you don't get confused about which serial port is the ESP32-S2

Run esptool and check connection

Once you have entered ROM bootloader mode, you can then use Espressif's esptool program (https://adafru.it/E9p) to communicate with the chip! `esptool` is the 'official' programming tool and is the most common/complete way to program an ESP chip.

You will need to use the command line / Terminal to install and run `esptool`.

You will also need to have pip and Python installed (any version!)

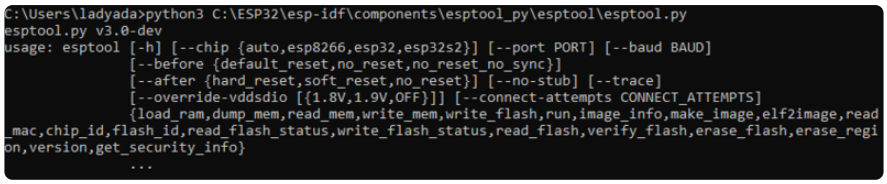
Install the latest version using pip (you may be able to run `pip` without the `3` depending on your setup):

```
pip3 install --upgrade esptool
```

Then, you can run:

```
esptool.py
```

Run `esptool.py` in a new terminal/command line and verify you get something like the below:



Make sure you are running esptool v 3.0 or higher, which adds ESP32-S2 support

esptool v3.2 has a bug which prevents automatic chip detection. Add the argument `--chip esp32-s2` to your commands if you are getting errors.

Run the following command, replacing the identifier after `--port` with the `COMxx`, `/dev/cu.usbmodemxx` or `/dev/ttySxx` you found above.

```
esptool.py --port COM88 chip_id
```

You should get a notice that it connected over that port and found an ESP32-S2

You can now upload a binary file with the following command

```
esptool.py --port COM88 --after=no_reset write_flash 0x0 firmware.bin
```

don't forget to change the `--port` name to match, and the file name from firmware.bin to whatever the firmware file name is.

For example, I downloaded CircuitPython .bin and programmed it thus:

```
C:\Users\ladyada\Desktop>python3 C:\ESP32\esp-idf\components\esptool_py\esptool\esptool.py --port COM88 -
-after=no_reset write_flash  0x0 adafruit-circuitpython-adafruit_esp32s2_eink_portal-en_US-20201102-58ce4
e1.bin
esptool.py v3.0-dev
Serial port COM88
Connecting...
Detecting chip type... ESP32-S2
Chip is ESP32-S2
Features: WiFi, ADC and temperature sensor calibration in BLK2 of efuse
Crystal is 40MHz
MAC: 7c:df:a1:00:3f:3e
Uploading stub...
Running stub...
Stub running...
Configuring flash size...
Auto-detected Flash size: 4MB
Compressed 1304816 bytes to 843841...
Wrote 1304816 bytes (843841 compressed) at 0x00000000 in 25.1 seconds (effective 416.6 kbit/s)...
Hash of data verified.

Leaving...
Staying in bootloader.
```

Once the data is verified, press the **Reset** button once more to launch the code you just programmed in!

Web Serial ESPTool

The WebSerial ESPTool was designed to be a web-capable option for programming Espressif ESP family microcontroller boards that have a serial based ROM bootloader. It allows you to erase the contents of the microcontroller and program up to 4 files at different offsets.

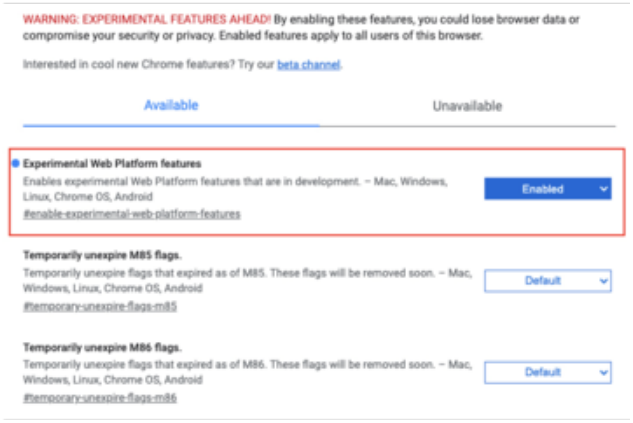
For boards that lack native USB, like the ESP32 or ESP32-C3 microcontroller, this is how any firmware like CircuitPython **.bin** files can be loaded. **There is no drag-and-drop to a folder option for these boards.**

For boards with native USB, like ESP32-S2, -S3, etc. this is how the UF2 bootloader **.bin** file can be loaded. Once the UF2 bootloader is on, firmware like CircuitPython **.uf2** files can be drag-and-dropped to a **BOOT** folder.

This tool is a good alternative for folks who cannot run Python `esptool.py` on their computer or are having difficulty installing or using `esptool.py`

Enabling Web Serial

You will have to use the Chrome or Chromium-based browser for this to work. For example, Edge and Opera are Chromium (https://adafru.it/10BL). Safari and Firefox, etc are not supported - they have not implemented Web Serial (https://adafru.it/10BM)!



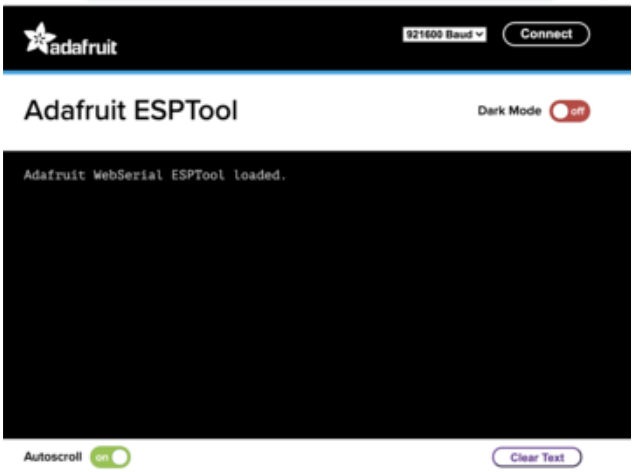
If you have a very old version of Chrome, you'll need to enable the Serial API, which is really easy.

Visit **chrome://flags** from within Chrome. Find and enable the **Experimental Web Platform features**

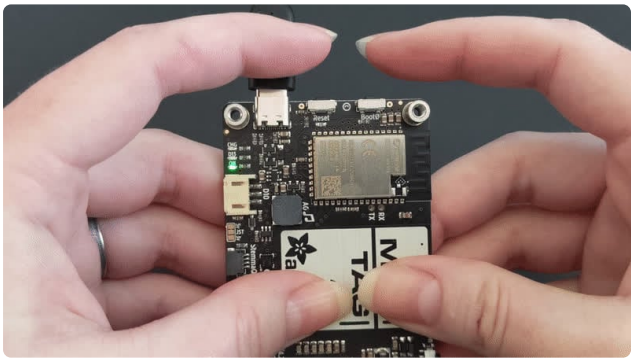
Restart Chrome

Connecting

Before you can use the tool, you will need to put your board in bootloader mode and connect. Here are the steps:

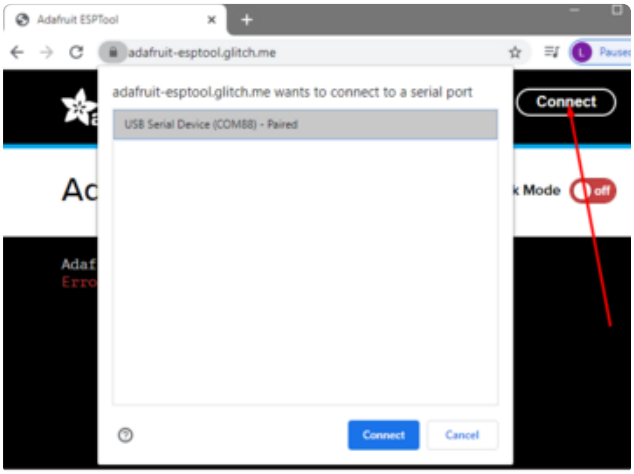


In the **Chrome browser** visit https://adafruit.github.io/Adafruit_WebSerial_ESPTool/ (https://adafru.it/PMB). You should see something like the image shown.



For all ESP32-family boards, you can enter the ROM bootloader by holding down the **BOOT** button while clicking **RST**.

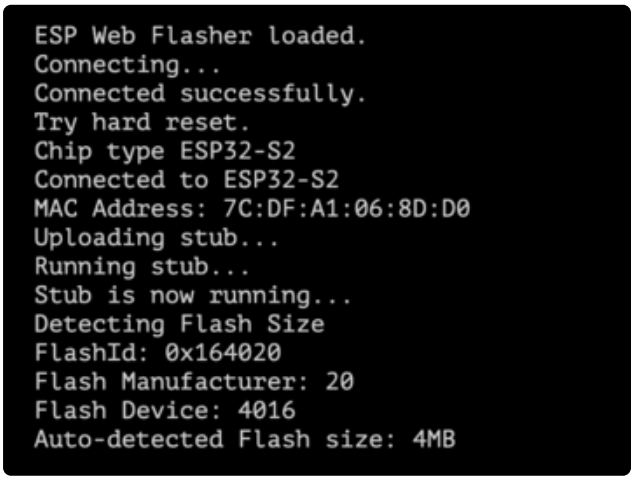
If you have an ESP32 board without BOOT (say because its an original ESP32, not -S2 or -S3, etc) you can skip this step because there's an auto-boot circuit.



Press the **Connect** button in the top right of the web browser. You will get a pop up asking you to select the COM or Serial port. Look for something with **ESP32**, **JTAG Loader**, **SLAB**, or **FTDI** in the name.

Remember, you should remove all other USB devices so only the target board is attached, that way there's no confusion over multiple ports!

On some systems, such as MacOS, there may be additional system ports that appear in the list.



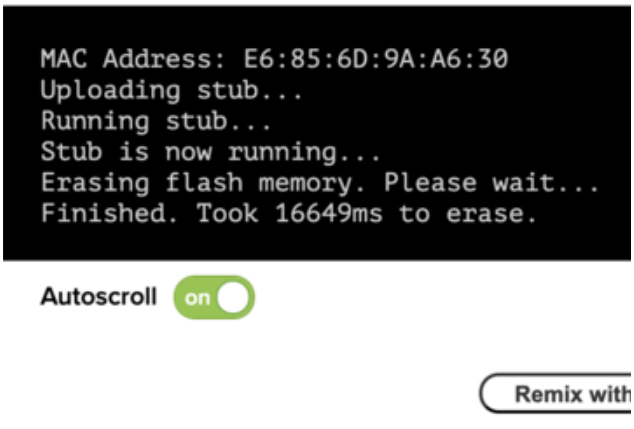
The Javascript code will now try to connect to the ROM bootloader. It may timeout for a bit until it succeeds. On success, you will see that it is **Connected** and will print out a unique **MAC address** identifying the board along with other information that was detected.

Once you have successfully connected, the command toolbar will appear.

Erasing the Contents

If you would like to erase the entire flash area so that you can start with a clean slate, you can use the erase feature. We recommend doing this if you are having issues.

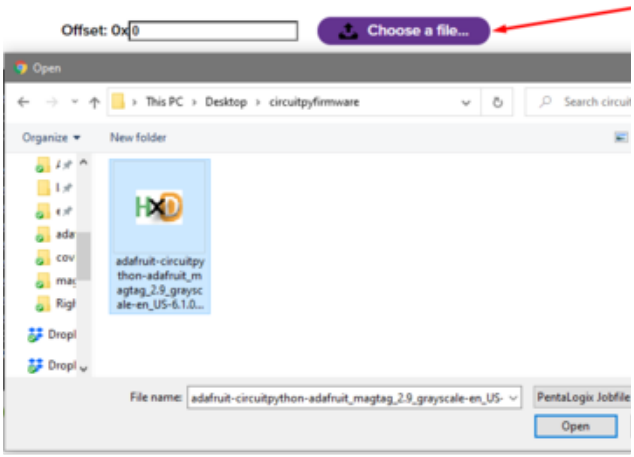
To erase the contents, click the Erase button. You will be prompted whether you want to continue. Click OK to continue or if you changed your mind, just click cancel.



You'll see "Erasing flash memory. Please wait..." This will eventually be followed by "Finished." and the amount of time it took to erase.

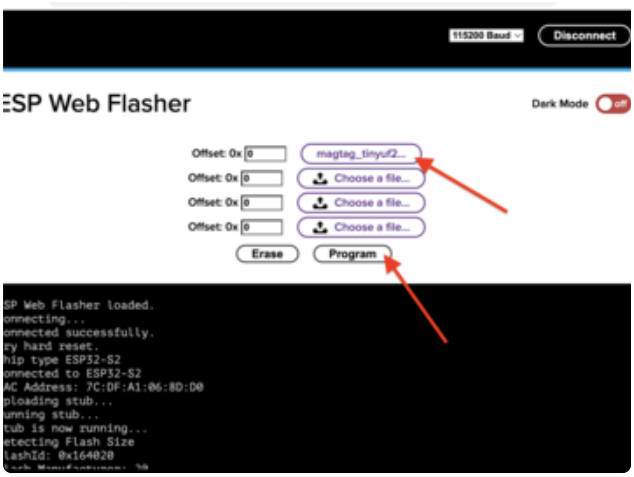
Do not disconnect! Immediately continue on to Programming the ESP Microcontroller.

Programming the ESP Microcontroller



You can click on **Choose a file...** from any of the available buttons. It will only attempt to program buttons with a file and a unique location. Then select the **.bin** file(s) - **not the UF2 file**!

Verify that the **Offset** box next to the file location you used is 0x0.



Once you choose a file, the button text will change to match your filename. You can then select the **Program** button to start flashing.

A progress bar will appear and after a minute or two, you will have written the firmware.

After using the tool, press the reset button to get out of bootloader mode and launch the new firmware!

For boards lacking native USB, like the ESP32 and ESP32-C3, **no folder will show up after pressing reset**. If CircuitPython firmware was loaded, the REPL can be accessed over a serial COM port.

For other boards, like ESP32-S2, -S3, etc., a **BOOT** folder should show up. A CircuitPython UF2 (https://adafru.it/Em8) file can now be copied over to the **BOOT** folder, after which the **CIRCUITPY** folder should then show up.

Install UF2 Bootloader

If you're familiar with our other products and chipsets you may be famliar with our drag-n-drop bootloader, a.k.a UF2. We have a UF2 bootloader for the ESP32-S2, that will let you drag firmware on/off a USB disk drive

Unlike the M0 (SAMD21) and M4 (SAMD51) boards, there is no bootloader protection for the UF2 bootloader. That means it is possible to erase or damage the bootloader, especially if you upload Arduino sketches to ESP32S2 boards that doesn't "know" there's a bootloader it should not overwrite!

However, thanks to the ROM bootloader, you don't have to worry about it if the UF2 bootloader is damaged. You can simply re-load the UF2 bootloader (USB-disk-style) with the ROM bootloader (non-USB-drive)

Installing a new bootloader will erase your board! Be sure to back up your data first.

Step 1. Get into the ROM bootloader and install esptool.py

See the previous page on how to do that! (https://adafru.it/OBc)

Step 2. Download the TinyUF2 release for your board

Choose the right release file from the list below. If your board is not explicitly mentioned, find it in the "all boards" link. These links are to **.zip** files.

Latest MagTag TinyUF2 release

https://adafru.it/TSf

Latest Metro ESP32-S2 TinyUF2 release

https://adafru.it/TSf

Latest Adafruit FunHouse TinyUF2 release

https://adafru.it/TSf

Look here if your board is not one of the ones above:

Latest TinyUF2 release for all boards

https://adafru.it/TSA

Step 3. Extract the combined.bin file from TinyUF2 release

The file you downloaded in Step 2 is a **.zip** file. Unzip it and find the **combined.bin** file. Note that this file is 3MB but that's because the bootloader is near the end of the available flash. It's not actually 3MB of program: mostof the file is empty but it's easier to program if we give you one combined 'swiss cheese' file. Save this file to your desktop or wherever you plan to run `esptool` from.

Step 4. Option A) Use esptool.py to upload

- Put the board into bootloader mode (hold down BOOT0/DFU and click reset)
- Determine the serial or COM port of the board

Run this command and replace the serial port name with your matching port and the file you just downloaded

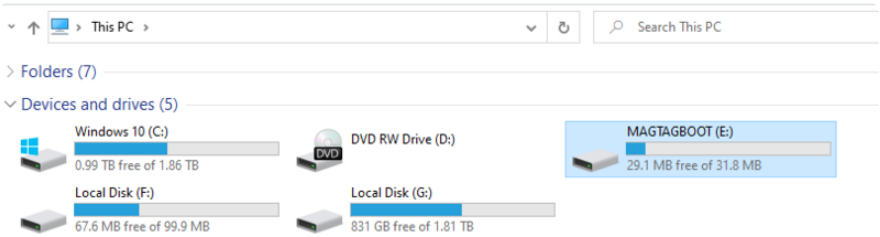
```
esptool.py -p COM88 write_flash 0x0 combined.bin
```

There might be a bit of a 'wait' when programming, where it doesn't seem like its working. Give it a minute, it has to erase the old flash code which can cause it to seem like its not running.

You'll finally get an output like this:

```
esptool.py v3.0-dev
Serial port COM88
Connecting...
Detecting chip type... ESP32-S2
Chip is ESP32-S2
Features: WiFi, ADC and temperature sensor calibration in BLK2 of efuse
Crystal is 40MHz
MAC: 7c:df:a1:05:f8:9a
Uploading stub...
Running stub...
Stub running...
Configuring flash size...
Auto-detected Flash size: 4MB
Compressed 3081264 bytes to 98937...
Wrote 3081264 bytes (98937 compressed) at 0x00000000 in 22.8 seconds (effective 1080.0 kbit/s)...
Hash of data verified.

Leaving...
Hard resetting via RTS pin...
ERROR: ESP32-S2 chip was placed into download mode using GPIO0.
esptool.py can not exit the download mode over USB. To run the app, reset the chip manually.
To suppress this error, set --after option to 'no_reset'.
```

Click RESET button to launch the bootloader. You'll see a new disk drive on your computer with the name **MAGTAGBOOT** or similar



Step 4 Option B) Use the Web Serial ESPTool to upload

Another option if you are having trouble getting ESPTool running, is to use the Web Serial ESPTool (https://adafru.it/Pdq) in this guide. This tool uses Web Serial to erase or upload firmware to your board.

Install CircuitPython

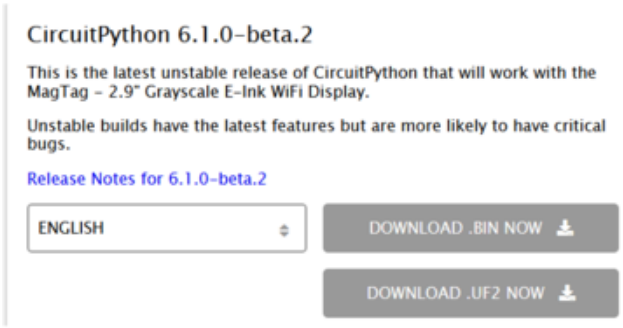
CircuitPython (https://adafru.it/tB7) is a derivative of MicroPython (https://adafru.it/BeZ) designed to simplify experimentation and education on low-cost microcontrollers. It makes it easier than ever to get prototyping by requiring no upfront desktop software downloads. Simply copy and edit files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

Set Up CircuitPython

Follow the steps to get CircuitPython installed on your MagTag.

Download the latest CircuitPython for your board from circuitpython.org

https://adafru.it/OBd



Click the link above and download the latest .BIN and .UF2 file

(depending on how you program the ESP32S2 board you may need one or the other, might as well get both)

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

Plug your MagTag 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.

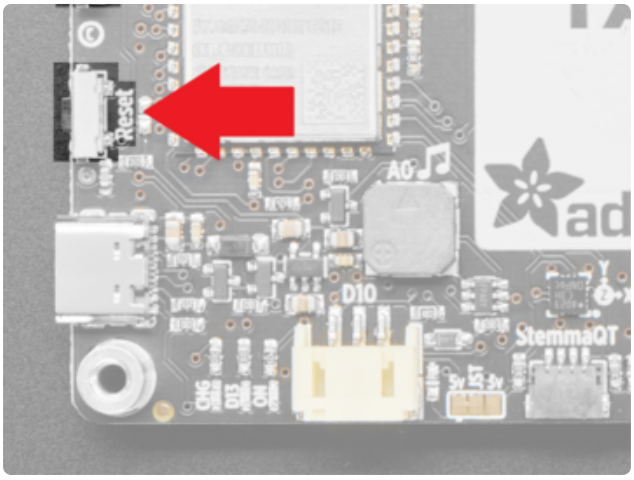
Option 1 - Load with UF2 Bootloader

This is by far the easiest way to load CircuitPython. **However it requires your board has the UF2 bootloader installed. Some early boards do not (we hadn't written UF2 yet!) - in which case you can load using the built in ROM bootloader.**

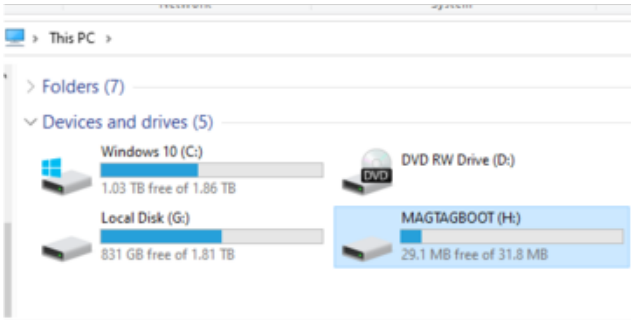
Still, try this first!

Try Launching UF2 Bootloader

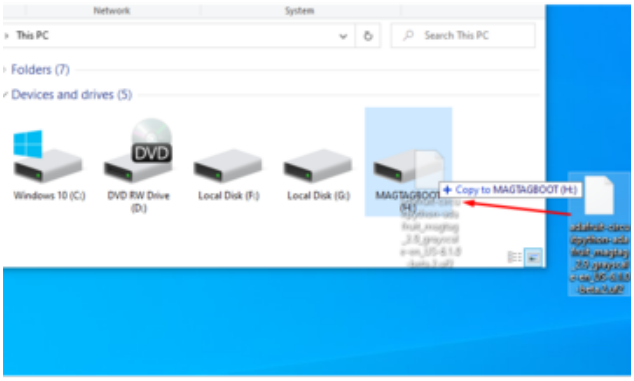
Loading CircuitPython by drag-n-drop UF2 bootloader is the easier way and we recommend it. If you have a MagTag where the front of the board is black, your MagTag came with UF2 already on it.



Launch UF2 by **double-clicking** the Reset button (the one next to the USB C port). You may have to try a few times to get the timing right.

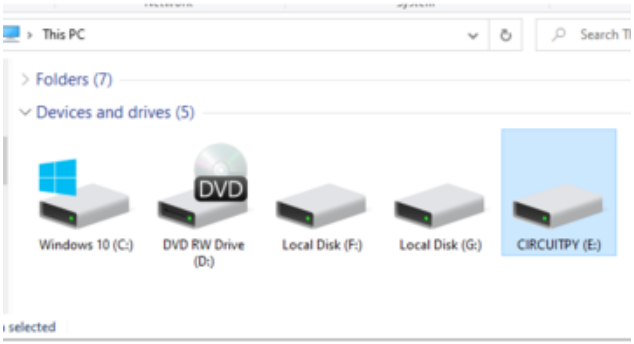


If the UF2 bootloader is installed, you will see a new disk drive appear called **MAGTAGBOOT**



Copy the **UF2** file you downloaded at the first step of this tutorial onto the **MAGTAGBOOT** drive

If you're using Windows and you get an error at the end of the file copy that says **Error from the file copy, Error 0x800701B1: A device which does not exist was specified.** You can ignore this error, the bootloader sometimes disconnects without telling Windows, the install completed just fine and you can continue. If its really annoying, you can also upgrade the bootloader (the latest version of the UF2 bootloader fixes this warning) (https://adafru.it/Pfk)

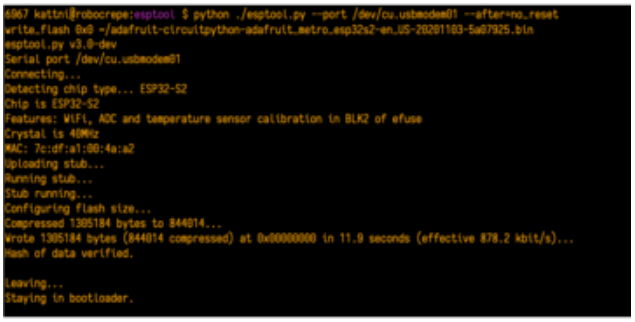


Your board should auto-reset into CircuitPython, or you may need to press reset. A **CIRCUITPY** drive will appear. You're done! Go to the next pages.

Option 2 - Use esptool to load BIN file

If you have an original MagTag with while soldermask on the front, we didn't have UF2 written for the ESP32S2 yet so it will not come with the UF2 bootloader.

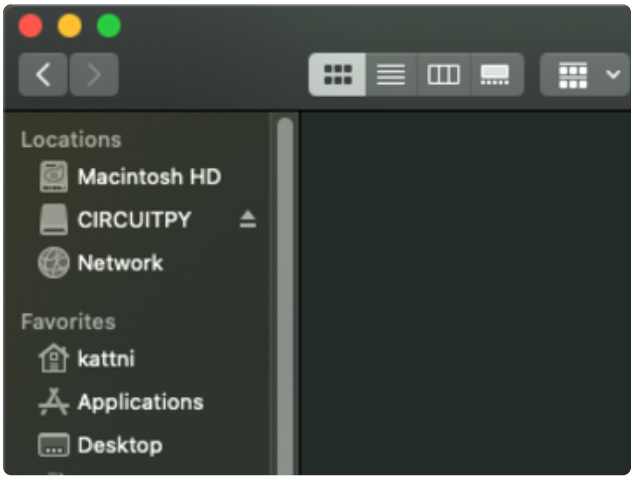
You can upload with **esptool** to the ROM (hardware) bootloader instead!



Follow the initial steps found in the Run esptool and check connection section of the ROM Bootloader page (https://adafru.it/OBc) to verify your environment is set up, your board is successfully connected, and which port it's using.

In the final command to write a binary file to the board, replace the port with your port, and replace "firmware.bin" with the the file you downloaded above.

The output should look something like the output in the image.



Press reset to exit the bootloader.

Your **CIRCUITPY** drive should appear!

You're all set! Go to the next pages.

Option 3 - Use Chrome Browser To Upload BIN file

If for some reason you cannot get esptool to run, you can always try using the Chrome-browser version of esptool we have written. This is handy if you don't have Python on your computer, or something is really weird with your setup that makes esptool not run (which happens sometimes and isn't worth debugging!) You can follow along on the Web Serial ESPTool (https://adafru.it/Pdq) page and either load the UF2

bootloader and then come back to Option 1 on this page, or you can download the CircuitPython BIN file directly using the tool in the same manner as the bootloader.

CircuitPython Internet Test

One of the great things about the ESP32 is the built-in WiFi capabilities. This page covers the basics of getting connected using CircuitPython.

The first thing you need to do is update your **code.py** to the following. Click the **Download Project Bundle** button below to download the necessary libraries and the **code.py** file in a zip file. Extract the contents of the zip file, and copy the **entire lib folder** and the **code.py** file to your **CIRCUITPY** drive.

```
# SPDX-FileCopyrightText: 2020 Brent Rubell for Adafruit Industries
#
# SPDX-License-Identifier: MIT

import os
import ipaddress
import ssl
import wifi
import socketpool
import adafruit_requests

# URLs to fetch from
TEXT_URL = "http://wifitest.adafruit.com/testwifi/index.html"
JSON_QUOTES_URL = "https://www.adafruit.com/api/quotes.php"
JSON_STARS_URL = "https://api.github.com/repos/adafruit/circuitpython"

print("ESP32-S2 WebClient Test")

print(f"My MAC address: {[hex(i) for i in wifi.radio.mac_address]}")

print("Available WiFi networks:")
for network in wifi.radio.start_scanning_networks():
    print("\t%s\t\tRSSI: %d\tChannel: %d" % (str(network.ssid, "utf-8"),
                                             network.rssi, network.channel))
wifi.radio.stop_scanning_networks()

print(f"Connecting to {os.getenv('CIRCUITPY_WIFI_SSID')}")
wifi.radio.connect(os.getenv("CIRCUITPY_WIFI_SSID"),
os.getenv("CIRCUITPY_WIFI_PASSWORD"))
print(f"Connected to {os.getenv('CIRCUITPY_WIFI_SSID')}")
print(f"My IP address: {wifi.radio.ipv4_address}")

ping_ip = ipaddress.IPv4Address("8.8.8.8")
ping = wifi.radio.ping(ip=ping_ip)

# retry once if timed out
if ping is None:
    ping = wifi.radio.ping(ip=ping_ip)

if ping is None:
    print("Couldn't ping 'google.com' successfully")
else:
    # convert s to ms
    print(f"Pinging 'google.com' took: {ping * 1000} ms")

pool = socketpool.SocketPool(wifi.radio)
requests = adafruit_requests.Session(pool, ssl.create_default_context())
```

```
print(f"Fetching text from {TEXT_URL}")
response = requests.get(TEXT_URL)
print("-" * 40)
print(response.text)
print("-" * 40)

print(f"Fetching json from {JSON_QUOTES_URL}")
response = requests.get(JSON_QUOTES_URL)
print("-" * 40)
print(response.json())
print("-" * 40)

print()

print(f"Fetching and parsing json from {JSON_STARS_URL}")
response = requests.get(JSON_STARS_URL)
print("-" * 40)
print(f"CircuitPython GitHub Stars: {response.json()['stargazers_count']}")
print("-" * 40)

print("Done")
```

Your **CIRCUITPY** drive should resemble the following.

To get connected, the next thing you need to do is update the **settings.toml** file.

The settings.toml File

We expect people to share tons of projects as they build CircuitPython WiFi widgets. What we want to avoid is people accidentally sharing their passwords or secret tokens and API keys. So, we designed all our examples to use a **settings.toml** file, that is on your **CIRCUITPY** drive, to hold secret/private/custom data. That way you can share your main project without worrying about accidentally sharing private stuff.

If you have a fresh install of CircuitPython on your board, the initial **settings.toml** file on your **CIRCUITPY** drive is empty.

To get started, you can update the **settings.toml** on your **CIRCUITPY** drive to contain the following code.

```
# SPDX-FileCopyrightText: 2023 Adafruit Industries
#
# SPDX-License-Identifier: MIT
```

```
# This is where you store the credentials necessary for your code.
# The associated demo only requires WiFi, but you can include any
# credentials here, such as Adafruit IO username and key, etc.
CIRCUITPY_WIFI_SSID = "your-wifi-ssid"
CIRCUITPY_WIFI_PASSWORD = "your-wifi-password"
```

This file should contain a series of Python variables, each assigned to a string. Each variable should describe what it represents (say `wifi_ssid`), followed by an = (equals sign), followed by the data in the form of a Python string (such as `"my-wifi-password"` including the quote marks).

At a minimum you'll need to add/update your WiFi SSID and WiFi password, so do that now!

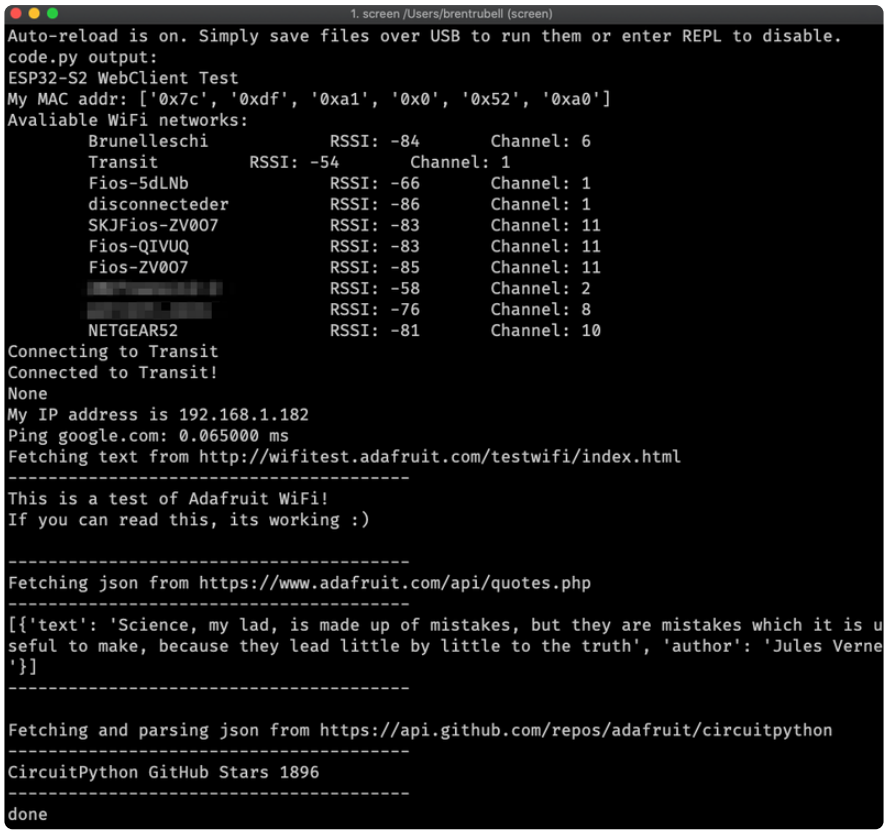
As you make projects you may need more tokens and keys, just add them one line at a time. See for example other tokens such as one for accessing GitHub or the Hackaday API. Other non-secret data like your timezone can also go here.

For the correct time zone string, look at http://worldtimeapi.org/timezones (https://adafru.it/EcP) and remember that if your city is not listed, look for a city in the same time zone, for example Boston, New York, Philadelphia, Washington DC, and Miami are all on the same time as New York.

Of course, don't share your **settings.toml** - keep that out of GitHub, Discord or other project-sharing sites.

Don't share your settings.toml file! It has your passwords and API keys in it!

If you connect to the serial console, you should see something like the following:



In order, the example code...

Checks the ESP32's MAC address.

```
print(f"My MAC address: {[hex(i) for i in wifi.radio.mac_address]}")
```

Performs a scan of all access points and prints out the access point's name (SSID), signal strength (RSSI), and channel.

```
print("Available WiFi networks:")
for network in wifi.radio.start_scanning_networks():
    print("\t%s\t\tRSSI: %d\tChannel: %d" % (str(network.ssid, "utf-8"),
                                              network.rssi, network.channel))
wifi.radio.stop_scanning_networks()
```

Connects to the access point you defined in the **settings.toml** file, and prints out its local IP address.

```
print(f"Connecting to {os.getenv('WIFI_SSID')}")
wifi.radio.connect(os.getenv("WIFI_SSID"), os.getenv("WIFI_PASSWORD"))
print(f"Connected to {os.getenv('WIFI_SSID')}")
print(f"My IP address: {wifi.radio.ipv4_address}")
```

Attempts to ping a Google DNS server to test connectivity. If a ping fails, it returns `None`. Initial pings can sometimes fail for various reasons. So, if the initial ping is successful (`is not None`), it will print the echo speed in ms. If the initial ping fails, it

will try one more time to ping, and then print the returned value. If the second ping fails, it will result in `"Ping google.com: None ms"` being printed to the serial console. Failure to ping does not always indicate a lack of connectivity, so the code will continue to run.

```
ping_ip = ipaddress.IPv4Address("8.8.8.8")
ping = wifi.radio.ping(ip=ping_ip) * 1000
if ping is not None:
    print(f"Ping google.com: {ping} ms")
else:
    ping = wifi.radio.ping(ip=ping_ip)
    print(f"Ping google.com: {ping} ms")
```

The code creates a socketpool using the wifi radio's available sockets. This is performed so we don't need to re-use sockets. Then, it initializes a a new instance of the requests (https://adafru.it/E9o) interface - which makes getting data from the internet really really easy.

```
pool = socketpool.SocketPool(wifi.radio)
requests = adafruit_requests.Session(pool, ssl.create_default_context())
```

To read in plain-text from a web URL, call `requests.get` - you may pass in either a http, or a https url for SSL connectivity.

```
print(f"Fetching text from {TEXT_URL}")
response = requests.get(TEXT_URL)
print("-" * 40)
print(response.text)
print("-" * 40)
```

Requests can also display a JSON-formatted response from a web URL using a call to `requests.get`.

```
print(f"Fetching json from {JSON_QUOTES_URL}")
response = requests.get(JSON_QUOTES_URL)
print("-" * 40)
print(response.json())
print("-" * 40)
```

Finally, you can fetch and parse a JSON URL using `requests.get`. This code snippet obtains the `stargazers_count` field from a call to the GitHub API.

```
print(f"Fetching and parsing json from {JSON_STARS_URL}")
response = requests.get(JSON_STARS_URL)
print("-" * 40)
print(f"CircuitPython GitHub Stars: {response.json()['stargazers_count']}")
print("-" * 40)
```

OK you now have your ESP32 board set up with a proper **settings.toml** file and can connect over the Internet. If not, check that your **settings.toml** file has the right SSID and password and retrace your steps until you get the Internet connectivity working!

IPv6 Networking

Starting in CircuitPython 9.2, IPv6 networking is available on most Espressif wifi boards. Socket-using libraries like **adafruit_requests** and **adafruit_ntp** will need to be updated to use the new APIs and for now can only connect to services on IPv4.

IPv6 connectivity & privacy

IPv6 addresses are divided into many special kinds, and many of those kinds (like those starting with **FC**, **FD**, **FE**) are private or local; Addresses starting with other prefixes like **2002:** and **2001:** are globally routable. In 2024, far from all ISPs and home networks support IPv6 internet connectivity. For more info consult resources like Wikipedia (https://adafru.it/1a4z). If you're interested in global IPv6 connectivity you can use services like Hurricane Electric (https://adafru.it/1a4A) to create an "IPv6 tunnel" (free as of 2024, but requires expertise and a compatible router or host computer to set up)

It's also important to be aware that, as currently implemented by Espressif, there are privacy concerns especially when these devices operate on the global IPv6 network: The device's unique identifier (its EUI-64 or MAC address) is used by default as part of its IPv6 address. This means that the device identity can be tracked across multiple networks by any service it connects to.

Enable IPv6 networking

Due to the privacy consideration, IPv6 networking is not automatically enabled. Instead, it must be explicitly enabled by a call to `start_dhcp_client` with the `ipv6=True` argument specified:

```
wifi.start_dhcp_client(ipv6=True)
```

Check IP addresses

The read-only `addresses` property of the `wifi.radio` object holds all addresses, including IPv4 and IPv6 addresses:

```
&gt;&gt;&gt; wifi.radio.addresses
('FE80::7EDF:A1FF:FE00:518C', 'FD5F:3F5C:FE50:0:7EDF:A1FF:FE00:518C', '10.0.3.96')
```

The `wifi.radio.dns` servers can be IPv4 or IPv6:

```
&gt;&gt;&gt; wifi.radio.dns
('FD5F:3F5C:FE50::1',)
```

```
&gt;&gt;&gt; wifi.radio.dns = ("1.1.1.1",)
&gt;&gt;&gt; wifi.radio.dns
('1.1.1.1',)
```

Ping v6 networks

`wifi.radio.ping` accepts v6 addresses and names:

```
&gt;&gt;&gt; wifi.radio.ping("google.com")
0.043
&gt;&gt;&gt; wifi.radio.ping("ipv6.google.com")
0.048
```

Create & use IPv6 sockets

Use the address family `socket.AF_INET6`. After the socket is created, use methods like `connect`, `send`, `recfrom_into`, etc just like for IPv4 sockets. This code snippet shows communicating with a private-network NTP server; this IPv6 address will not work on your network:

```
&gt;&gt;&gt; ntp_addr = ("fd5f:3f5c:fe50::20e", 123)
&gt;&gt;&gt; PACKET_SIZE = 48
&gt;&gt;&gt;
&gt;&gt;&gt; buf = bytearray(PACKET_SIZE)
&gt;&gt;&gt; with socket.socket(socket.AF_INET6, socket.SOCK_DGRAM) as s:
...     s.settimeout(1)
...     buf[0] = 0b0010_0011
...     s.sendto(buf, ntp_addr)
...     print(s.recvfrom_into(buf))
...     print(buf)
...
48
(48, ('fd5f:3f5c:fe50::20e', 123))
bytearray(b'$\x01\x03\xeb\x00\x00\x00\x00\x00\x00\x00GGPS\x00\xeaA0h\x07s;
\xc0\x00\x00\x00\x00\x00\x00\x00\xeaA0n\xeb4\x82-\xeaA0n\xebAU\xb1')
```

Getting The Date & Time

A very common need for projects is to know the current date and time. Especially when you want to deep sleep until an event, or you want to change your display based on what day, time, date, etc. it is

Determining the correct local time is really really hard. There are various time zones, Daylight Savings dates, leap seconds, etc. Trying to get NTP time and then back-calculating what the local time is, is extraordinarily hard on a microcontroller just isn't worth the effort and it will get out of sync as laws change anyways.

For that reason, we have the free adafruit.io time service. **Free for anyone with a free adafruit.io account.** You do need an account because we have to keep accidentally mis-programmed-board from overwhelming adafruit.io and lock them out temporarily. Again, it's free!

There are other services like WorldTimeAPI, but we don't use those for our guides because they are nice people and we don't want to accidentally overload their site. Also, there's a chance it may eventually go down or also require an account.

Step 1) Make an Adafruit account

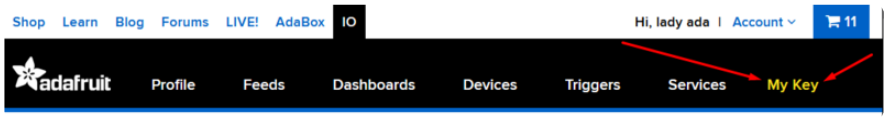
It's free! Visit https://accounts.adafruit.com/ (https://adafru.it/dyy) to register and make an account if you do not already have one

Step 2) Sign into Adafruit IO

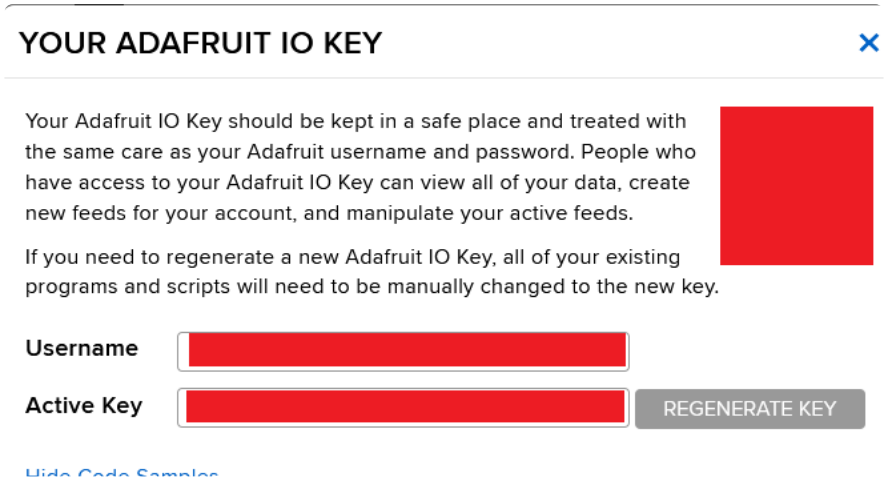
Head over to io.adafruit.com (https://adafru.it/fsU) and click **Sign In** to log into IO using your Adafruit account. It's free and fast to join.

Step 3) Get your Adafruit IO Key

Click on **My Key** in the top bar



You will get a popup with your **Username** and **Key** (In this screenshot, we've covered it with red blocks)



Go to the **settings.toml** file on your CIRCUITPY drive and add three lines for `AIO_USERNAME`, `ADAFRUIT_AIO_KEY` and `TIMEZONE` so you get something like the following:

```
# This file is where you keep secret settings, passwords, and tokens!
# If you put them in the code you risk committing that info or sharing it

CIRCUITPY_WIFI_SSID = "your-wifi-ssid"
```

```
CIRCUITPY_WIFI_PASSWORD = "your-wifi-password"
ADAFRUIT_AIO_USERNAME = "your-adafruit-io-username"
ADAFRUIT_AIO_KEY = "your-adafruit-io-key"
# Timezone names from http://worldtimeapi.org/timezones
TIMEZONE="America/New_York"
```

The timezone is optional, if you don't have that entry, adafruit.io will guess your timezone based on geographic IP address lookup. You can visit http://worldtimeapi.org/timezones (https://adafru.it/EcP) to see all the time zones available (even though we do not use Worldtime for time-keeping, we do use the same time zone table).

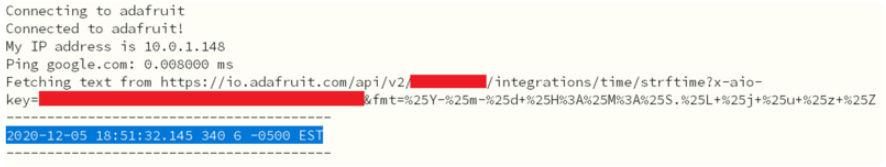
Step 4) Upload Test Python Code

This code is like the Internet Test code from before, but this time it will connect to adafruit.io and get the local time

```
import ipaddress
import os
import ssl
import wifi
import socketpool
import adafruit_requests
import secrets


# Get our username, key and desired timezone
ssid = os.getenv("CIRCUITPY_WIFI_SSID")
password = os.getenv("CIRCUITPY_WIFI_PASSWORD")
aio_username = os.getenv("ADAFRUIT_AIO_USERNAME")
aio_key = os.getenv("ADAFRUIT_AIO_KEY")
timezone = os.getenv("TIMEZONE")
TIME_URL = f"https://io.adafruit.com/api/v2/{aio_username}/integrations/time/
strftime?x-aio-key={aio_key}&amp;tz={timezone}"
TIME_URL += "&amp;fmt=%25Y-%25m-%25d+%25H%3A%25M%3A%25S.%25L+%25j+%25u+%25z+%25Z"

print("ESP32-S2 Adafruit IO Time test")

print("My MAC addr:", [hex(i) for i in wifi.radio.mac_address])

print("Available WiFi networks:")
for network in wifi.radio.start_scanning_networks():
    print("\t%s\t\tRSSI: %d\tChannel: %d" % (str(network.ssid, "utf-8"),
            network.rssi, network.channel))
wifi.radio.stop_scanning_networks()

print("Connecting to", ssid)
wifi.radio.connect(ssid, password)
print(f"Connected to {ssid}!")
print("My IP address is", wifi.radio.ipv4_address)

ipv4 = ipaddress.ip_address("8.8.4.4")
print("Ping google.com:", wifi.radio.ping(ipv4), "ms")

pool = socketpool.SocketPool(wifi.radio)
requests = adafruit_requests.Session(pool, ssl.create_default_context())

print("Fetching text from", TIME_URL)
response = requests.get(TIME_URL)
print("-" * 40)
```

```
print(response.text)
print("-" * 40)
```

After running this, you will see something like the below text. We have blocked out the part with the secret username and key data!



Note at the end you will get the date, time, and your timezone! If so, you have correctly configured your **settings.toml** and can continue to the next steps!

MagTag-Specific CircuitPython Libraries

To use all the amazing features of your MagTag with CircuitPython, you must first install a number of libraries. This page covers that process.

Get Latest Adafruit CircuitPython Bundle

Download the Adafruit CircuitPython Library Bundle. You can find the latest release here:

Download the latest Library Bundle from circuitpython.org

https://adafru.it/ENC

Download the **adafruit-circuitpython-bundle-version-mpy-*.zip** bundle zip file, and unzip a folder of the same name. Inside you'll find a **lib** folder. The entire collection of libraries is too large to fit on the **CIRCUITPY** drive. Therefore, you'll need to copy the necessary libraries to your board individually.

At a minimum, the following libraries are required. Copy the following folders or .mpy files to the **lib** folder on your **CIRCUITPY** drive. **If the library is a folder, copy the entire folder** to the **lib** folder on your board.

Library folders (copy the whole folder over to lib):

- **adafruit_magtag** - This is a helper library designed for using all of the features of the MagTag, including networking, buttons, NeoPixels, etc.
- **adafruit_portalbase** - This library is the base library that adafruit_magtag is built on top of.

- **adafruit_bitmap_font** - There is fancy font support, and it's easy to make new fonts. This library reads and parses font files.
- **adafruit_display_text** - This library displays text on the screen.
- **adafruit_io** - This library helps connect the MagTag to our free data logging and viewing service
- **adafruit_minimqtt** - This library provides MQTT service for Adafruit IO.

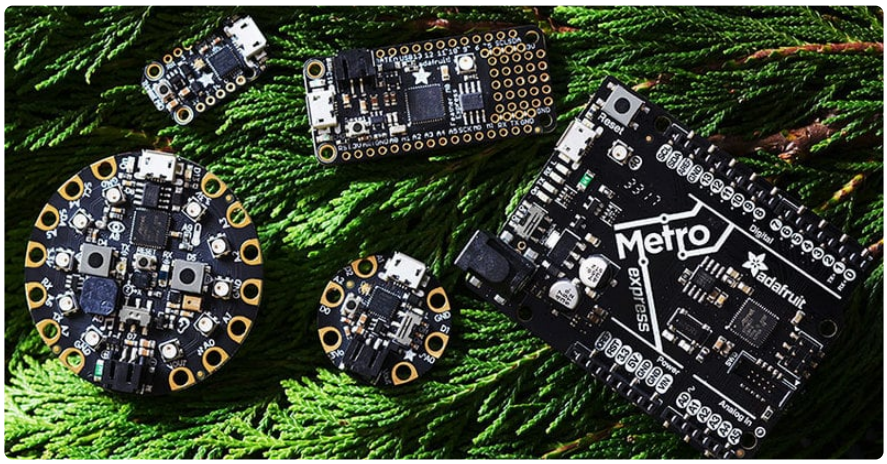
Library files:

- **adafruit_requests.mpy** - This library allows us to perform HTTP requests and get responses back from servers. GET/POST/PUT/PATCH - they're all in here!
- **adafruit_fakerequests.mpy** - This library allows you to create fake HTTP requests by using local files.
- **adafruit_miniqr.mpy** - QR creation library lets us add easy-to-scan 2D barcodes to the E-Ink display
- **neopixel.mpy** - This library is used to control the onboard NeoPixels.
- **simpleio.mpy** - This library is used for tone generation.

Secrets

Even if you aren't planning to go online with your MagTag, you'll need to have a **secrets.py** file in the root directory (top level) of your **CIRCUITPY** drive. If you do not intend to connect to wireless, it does not need to have valid data in it. Here's more info on the secrets.py file (https://adafru.it/P3b).

Welcome To CircuitPython



So, you've got a new **CircuitPython compatible board**. You plugged it in. Maybe it showed up as a disk drive called **CIRCUITPY**. Maybe it didn't! Either way, you need to know where to go from here. Well, this guide has you covered!

This guide will get you started with CircuitPython!

There are many amazing things about your new board. One of them is the ability to run CircuitPython. You may have seen that name on the Adafruit site (https://adafru.it/dAR) somewhere. Not sure what it is? This guide can help!

"But I've never coded in my life. There's no way I do it!" You absolutely can! CircuitPython is designed to help you learn from the ground up. If you're new to everything, this is the place to start!

This guide will walk you through how to get started with CircuitPython. You'll learn how to install CircuitPython, get updated to the newest version of CircuitPython, setup a serial connection, and edit your code. You'll learn some basics of how CircuitPython works, and about the CircuitPython libraries. You'll also find a list of frequently asked questions, and a series of troubleshooting steps if you run into any issues.

Welcome to CircuitPython!

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

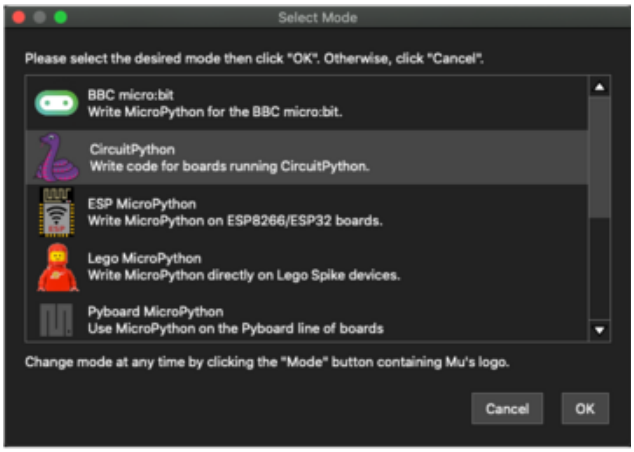
Download Mu from https://codewith.mu (https://adafru.it/Be6).

Click the **Download** link for downloads and installation instructions.

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

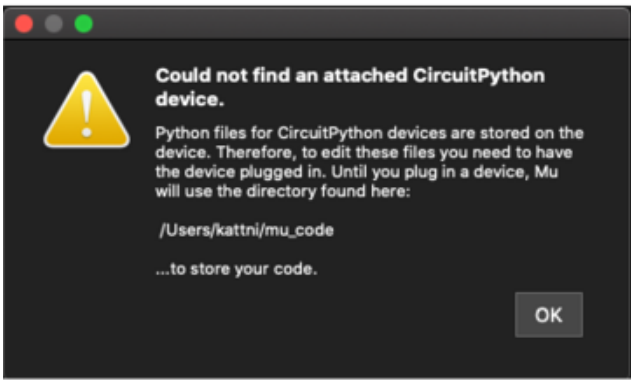
Windows users: due to the nature of MSI installers, please remove old versions of Mu before installing the latest version.

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.

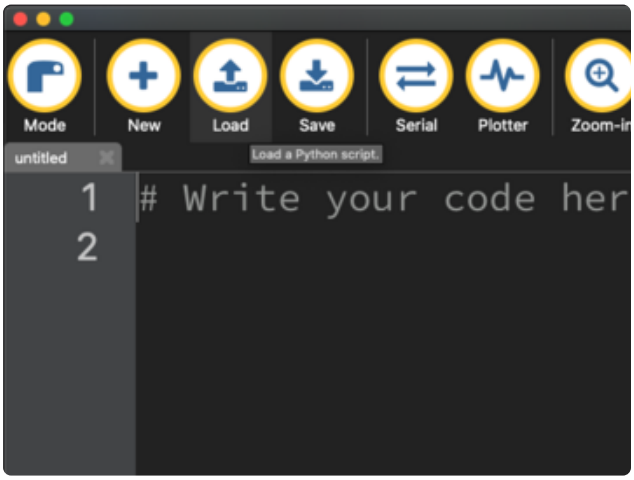
To create and edit code, all you'll need is an editor. There are many options. **Adafruit strongly recommends using Mu! It's designed for CircuitPython, and it's really simple and easy to use, with a built in serial console!**

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) 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 was formerly not a problem on macOS, but see the warning below.)

> macOS Sonoma 14.1 introduced a bug that delays writes to small drives such as CIRCUITPY drives. This caused errors when saving files to CIRCUITPY. There is a workaround. The bug was fixed in Sonoma 14.4, but at the cost of greatly slowed writes to drives 1GB or smaller.

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:

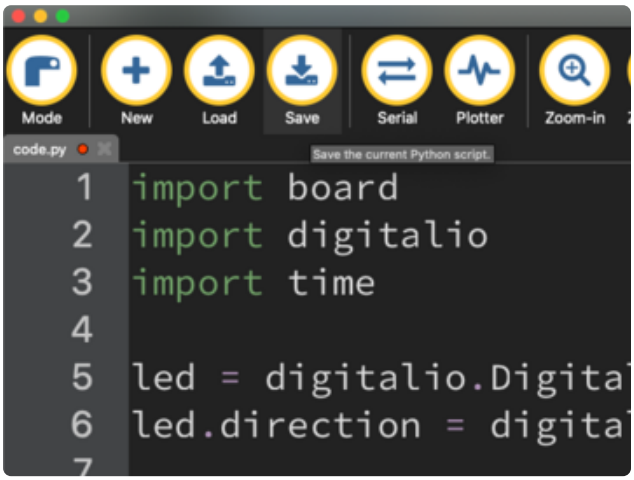
```
import board
import digitalio
import time

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 , Qualia, 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, Qualia, or the Trinkeys!

If you're using a KB2040, QT Py, Quaila, or a Trinkey, or any other board without a single-color LED that can blink, please download the NeoPixel blink example (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.

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.



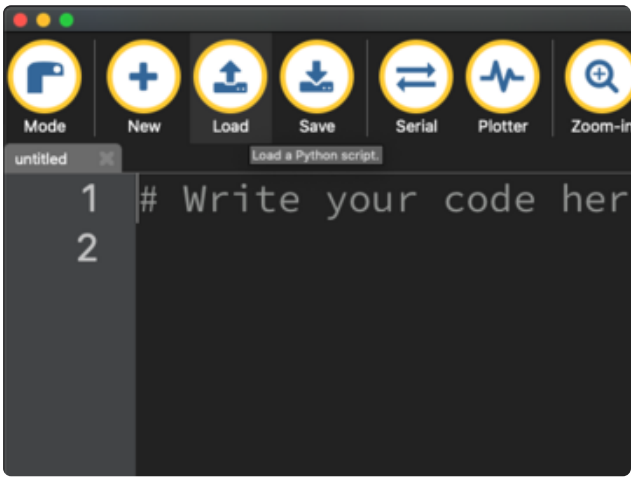
Save the **code.py** file on your **CIRCUITPY** drive.

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, Qualia, 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) 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) page of every board guide to get your board up and running again.

If you are having trouble saving code on Windows 10, try including this code snippet at the top of code.py:

```
import supervisor
supervisor.runtime.autoreload = False
```

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!`

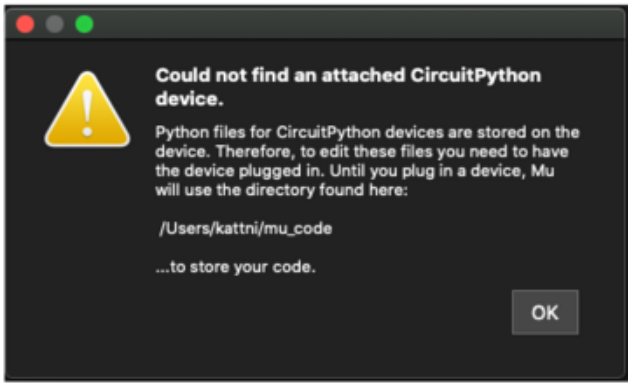
However, these print statements need somewhere to display. That's where the serial console comes in!

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.

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) for details on how to add yourself to the right group.

Using Something Else?

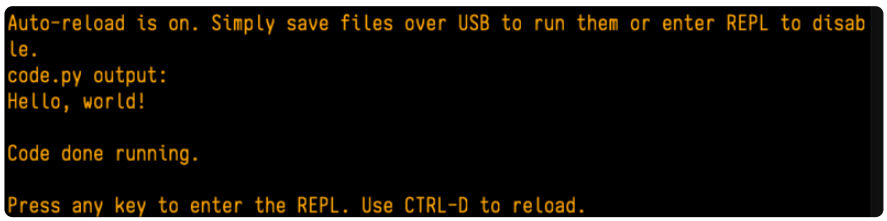
If you're not using Mu to edit, are using or if for some reason you are not a fan of its built in serial console, you can run the serial console from a separate program.

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

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)

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)

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



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)
```

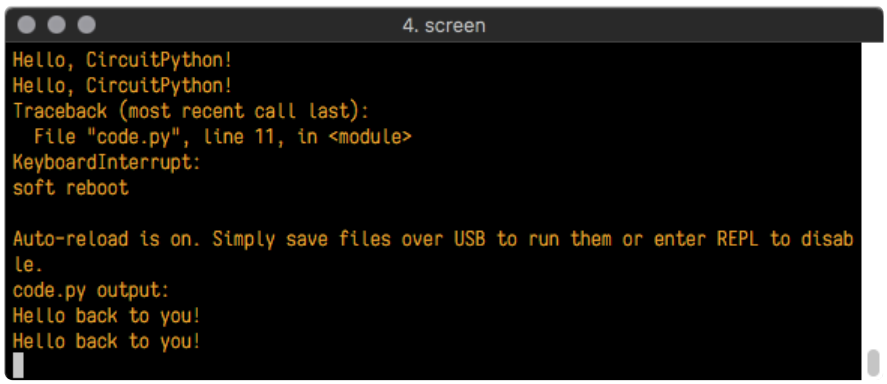
Save your file.

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

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!



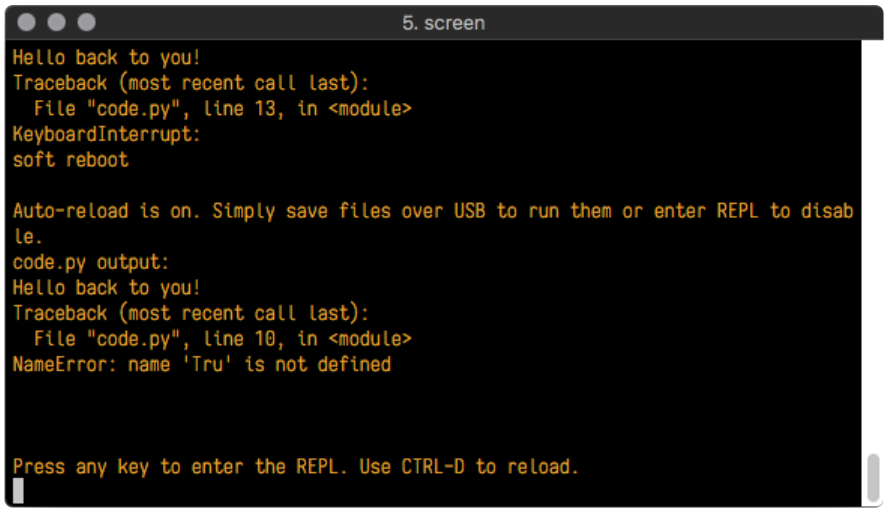
The `Traceback (most recent call last):` is telling you the last thing your board was doing before you saved your file. This is normal behavior and will happen every time the board resets. This is really handy for troubleshooting. Let's introduce an error so you can see how it is used.

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)
```

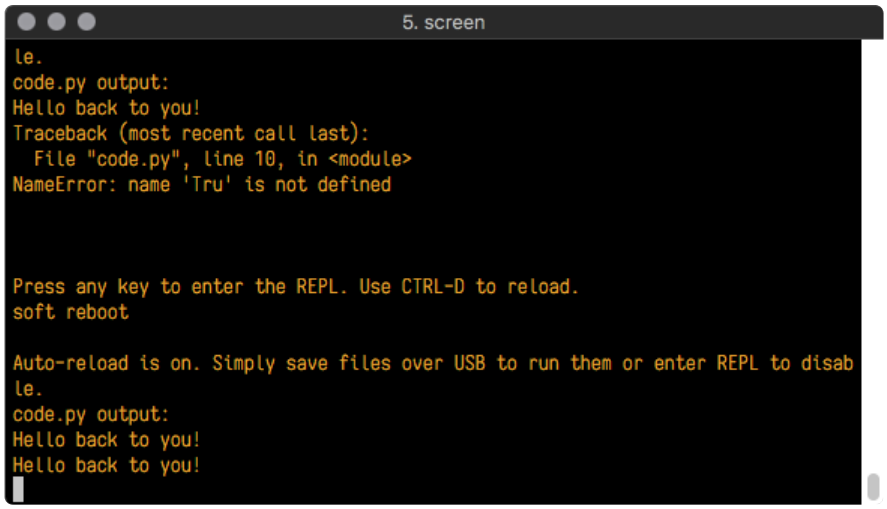
Save your file. You will notice that your red LED will stop blinking, and you may have a colored status LED blinking at you. This is because the code is no longer correct and can no longer run properly. You need to fix it!

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!



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.



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 **R**ead-**E**valuate-**P**rint-**L**oop, or REPL. The REPL allows you to enter individual lines of code and have them run immediately. It's really handy if you're running into trouble with a particular program and can't figure out why. It's interactive so it's great for testing new ideas.

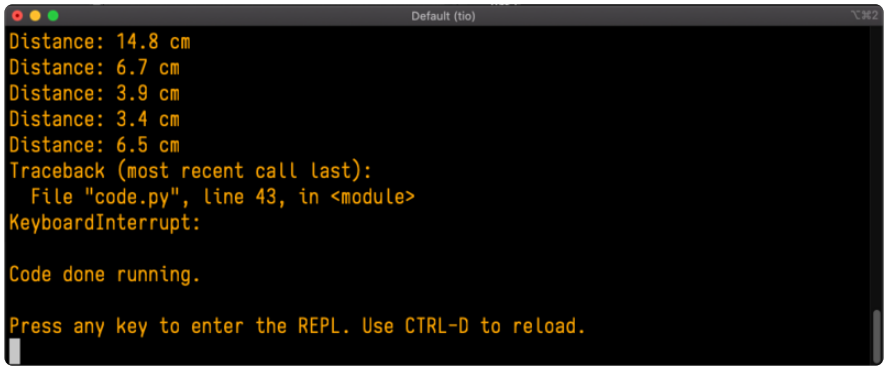
Entering the REPL

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**.

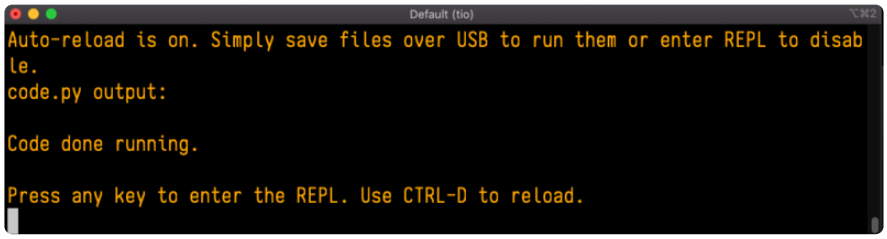
If there is code running, in this case code measuring distance, it will stop and you'll see `Press any key to enter the REPL. Use CTRL-D to reload.` Follow those instructions, and press any key on your keyboard.

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`

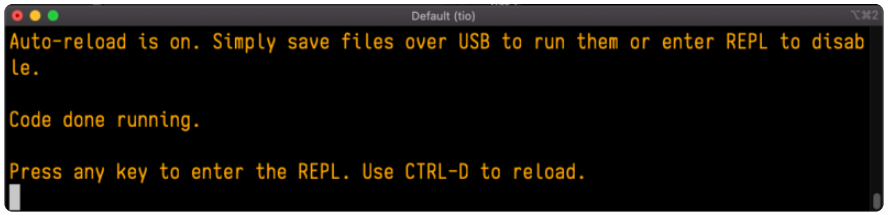
is you pressing CTRL+C. This information can be handy when troubleshooting, but for now, don't worry about it. Just note that it is expected behavior.



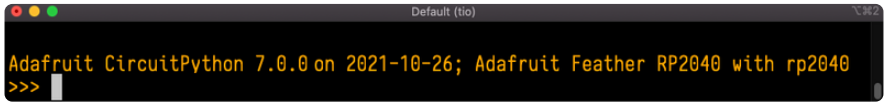
If your **code.py** file is empty or does not contain a loop, it will show an empty output and `Code done running.`. There is no information about what your board was doing before you interrupted it because there is no code running.



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.



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



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

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

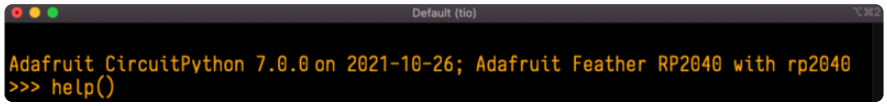
This line tells you the version of CircuitPython you're using and when it was released. Next, it gives you the type of board you're using and the type of microcontroller the board uses. Each part of this may be different for your board depending on the versions you're working with.

This is followed by the CircuitPython prompt.

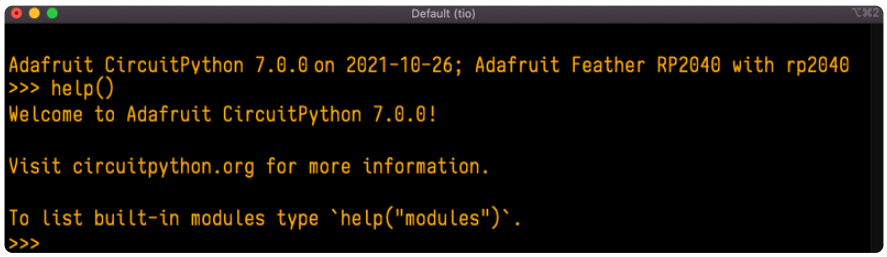
Interacting with the REPL

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.



Then press enter. You should then see a message.



First part of the message is another reference to the version of CircuitPython you're using. Second, a URL for the CircuitPython related project guides. Then... wait. What's this? To list built-in modules 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
_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
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.

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.

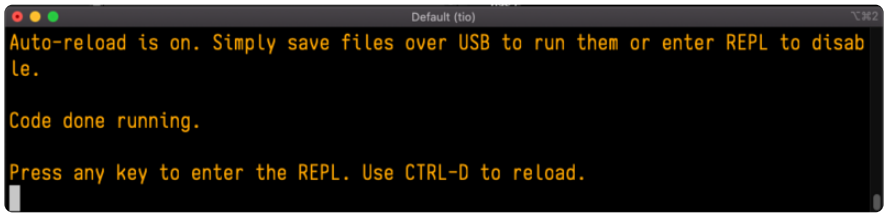
Try typing more into the REPL to see what happens!

> 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 **CTRL+D**. This will reload your board and reenter the serial console. You will restart the program you had running before entering the REPL. In the console window, you'll see any output from the program you had running. And if your program was affecting anything visual on the board, you'll see that start up again as well.

You can return to the REPL at any time!



Advanced Serial Console on Windows

Windows 7 and 8.1

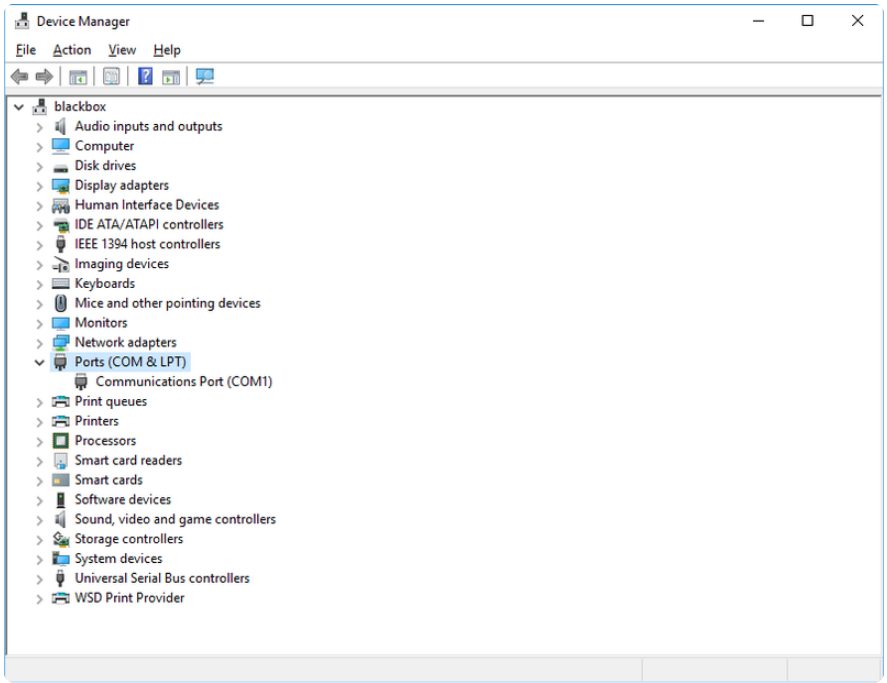
If you're using Windows 7 (or 8 or 8.1), you'll need to install drivers. See the Windows 7 and 8.1 Drivers page (https://adafru.it/VuB) for details. You will not need to install drivers on Mac, Linux or Windows 10.

You are strongly encouraged to upgrade to Windows 10 if you are still using Windows 7 or Windows 8 or 8.1. Windows 7 has reached end-of-life and no longer receives security updates. A free upgrade to Windows 10 is still available (https://adafru.it/RWc).

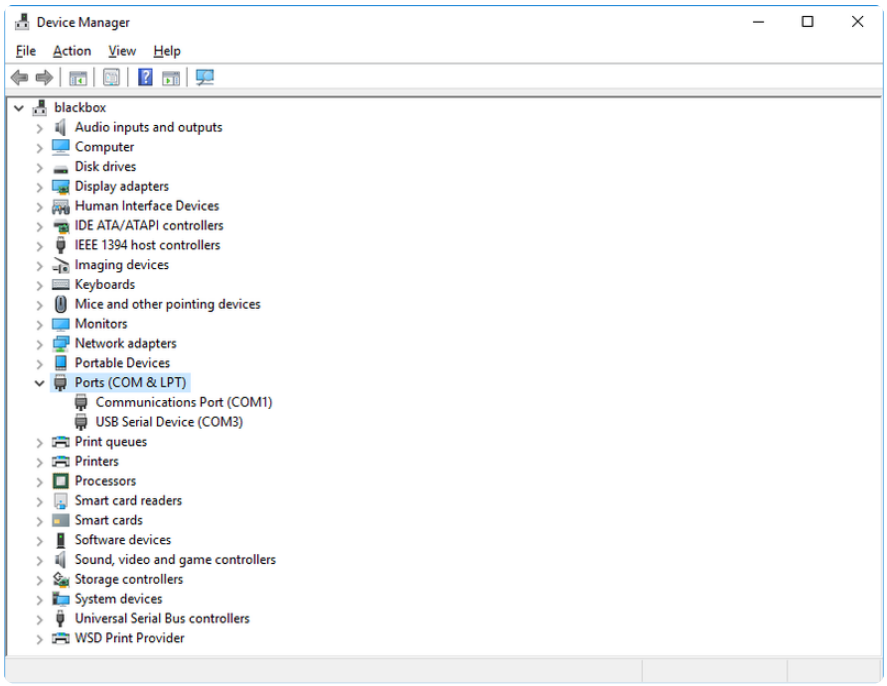
What's the COM?

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

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



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



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

Install Putty

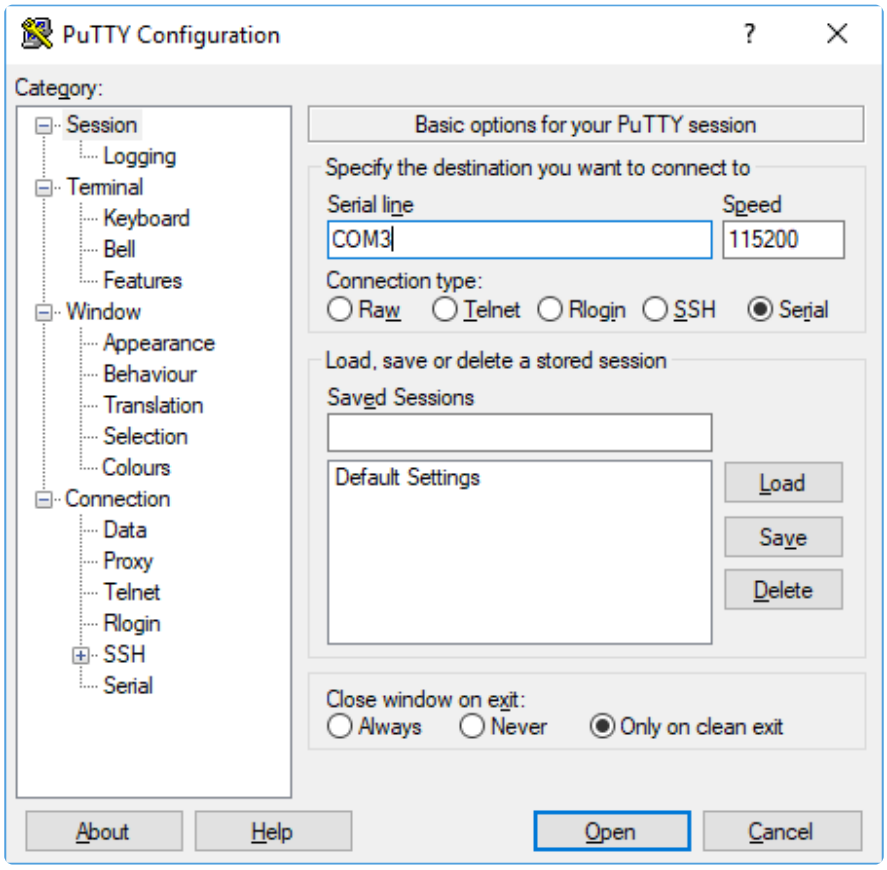
If you're using Windows, you'll need to download a terminal program. You're going to use PuTTY.

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

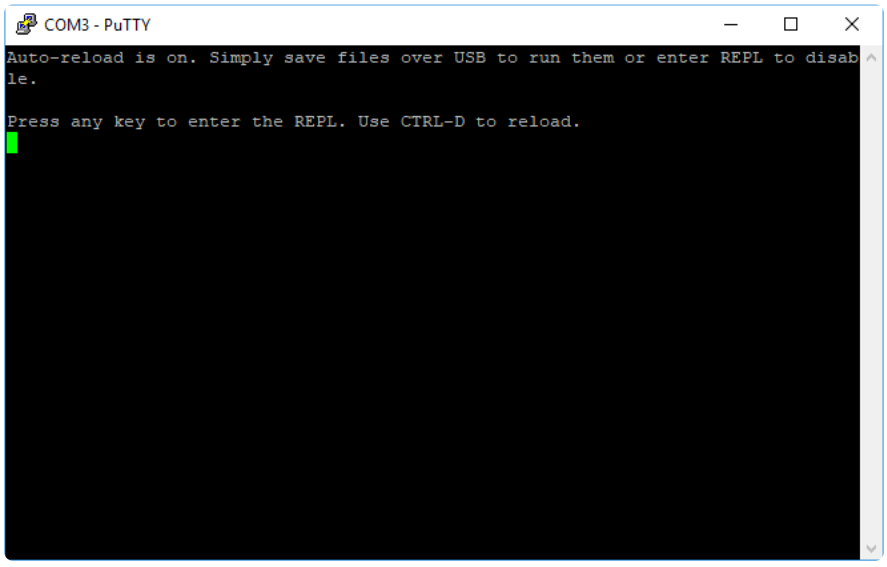
Now you need to open PuTTY.

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

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



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



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

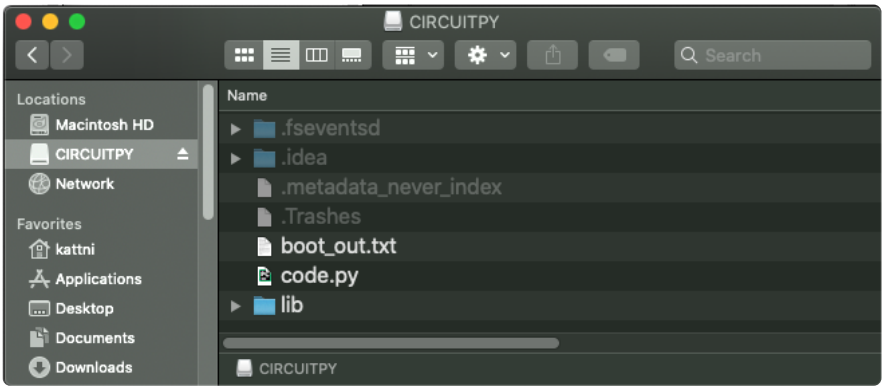
Great job! You've connected to the serial console!

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) 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.

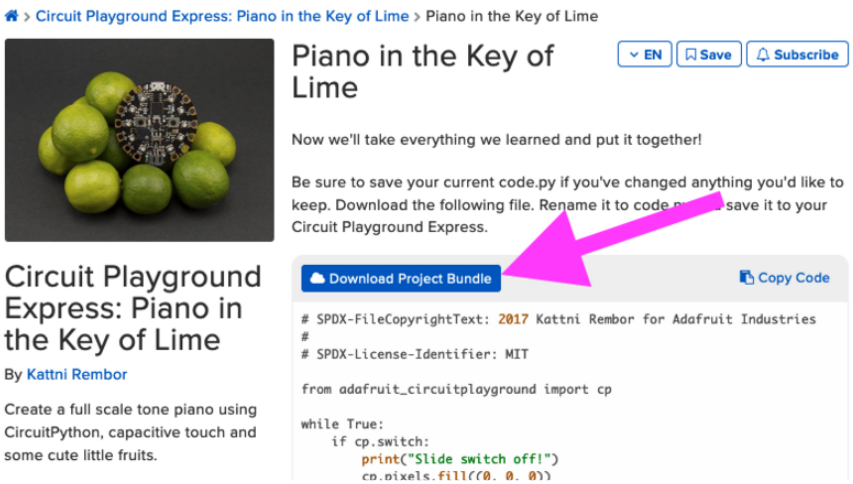
Either way, as you start to explore CircuitPython, you'll want to know how to get libraries on board.

The Adafruit Learn Guide Project Bundle

The quickest and easiest way to get going with a project from the Adafruit Learn System is by utilising the Project Bundle. Most guides now have a **Download Project Bundle** button available at the top of the full code example embed. This button downloads all the necessary files, including images, etc., to get the guide project up and running. Simply click, open the resulting zip, copy over the right files, and you're good to go!

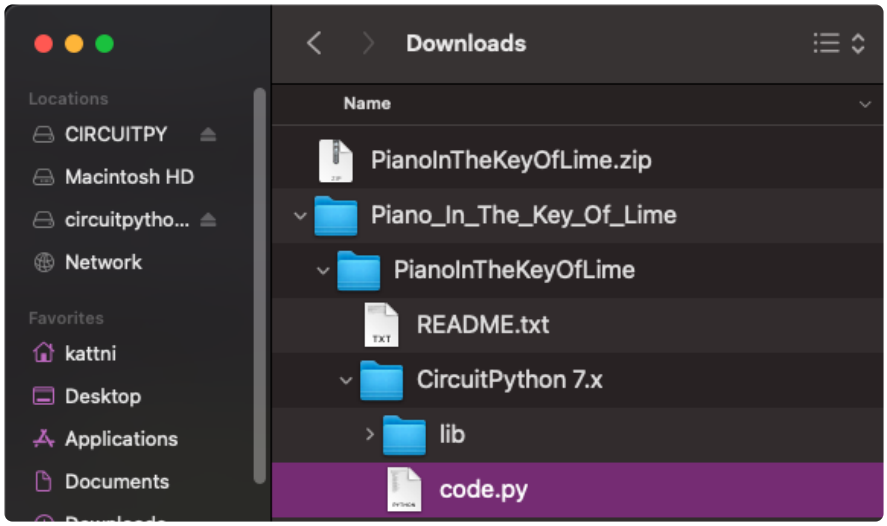
The first step is to find the Download Project Bundle button in the guide you're working on.

> The Download Project Bundle button is only available on full demo code embedded from GitHub in a Learn guide. Code snippets will NOT have the button available.



When you copy the contents of the Project Bundle to your CIRCUITPY drive, it will replace all the existing content! If you don't want to lose anything, ensure you copy your current code to your computer before you copy over the new Project Bundle content!

The Download Project Bundle button downloads a zip file. This zip contains a series of directories, nested within which is the **code.py**, any applicable assets like images or audio, and the **lib/** folder containing all the necessary libraries. The following zip was downloaded from the Piano in the Key of Lime guide.



The Piano in the Key of Lime guide was chosen as an example. That guide is specific to Circuit Playground Express, and cannot be used on all boards. Do not expect to download that exact bundle and have it work on your non-CPX microcontroller.

When you open the zip, you'll find some nested directories. Navigate through them until you find what you need. You'll eventually find a directory for your CircuitPython version (in this case, 7.x). In the version directory, you'll find the file and directory you need: **code.py** and **lib/**. Once you find the content you need, you can copy it all over to your **CIRCUITPY** drive, replacing any files already on the drive with the files from the freshly downloaded zip.

In some cases, there will be other files such as audio or images in the same directory as code.py and lib/. Make sure you include all the files when you copy things over!

Once you copy over all the relevant files, the project should begin running! If you find that the project is not running as expected, make sure you've copied ALL of the project files onto your microcontroller board.

That's all there is to using the Project Bundle!

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 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.

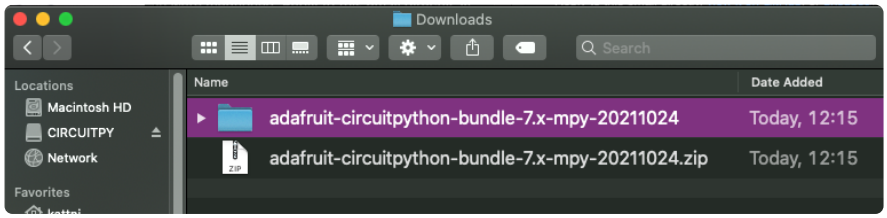
Click for the latest CircuitPython Community Library Bundle release

https://adafru.it/VCn

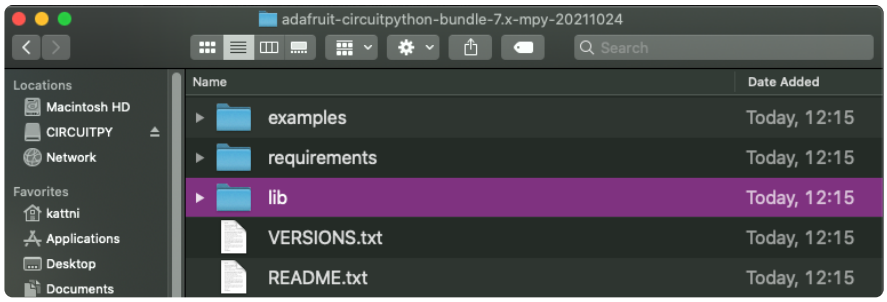
The link takes you to the latest release of the CircuitPython Community Library Bundle on GitHub. There are multiple versions of the bundle available. **Download the bundle version that matches your CircuitPython firmware version.** If you don't know the version, check the version info in **boot_out.txt** 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

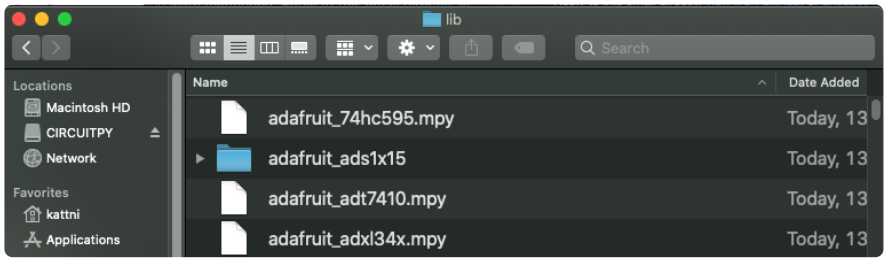
After downloading the zip, extract its contents. This is usually done by double clicking on the zip. On Mac OSX, it places the file in the same directory as the zip.



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.



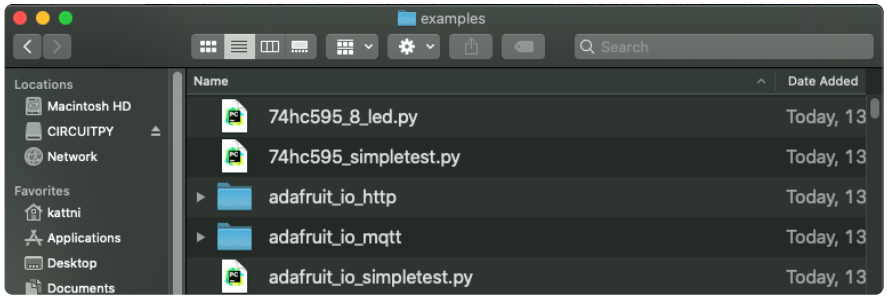
Now open the lib folder. When you open the folder, you'll see a large number of **.mpy** files, and folders.



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) on The REPL page (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.

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 **CIRCUITPY** 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.

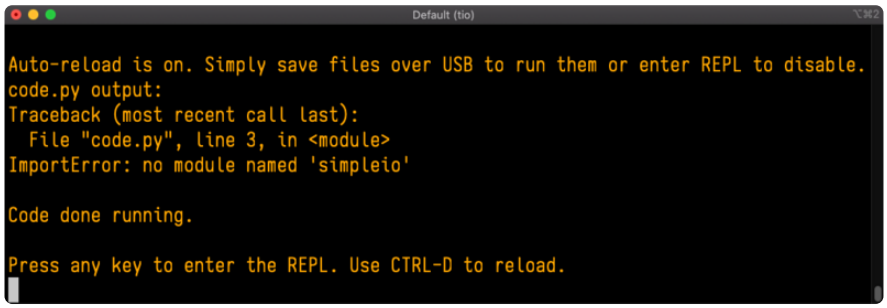
This demonstration will only return an error if you do not have the required library loaded into the **lib** folder on your **CIRCUITPY** drive.

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)
```

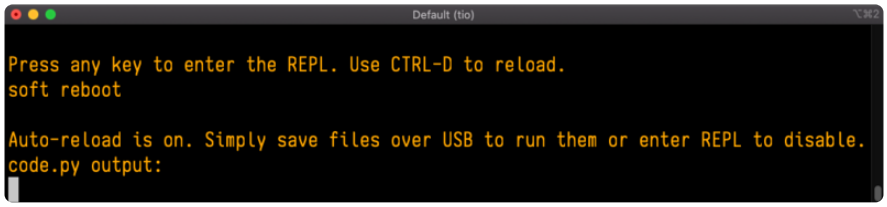
Save this file. Nothing happens to your board. Let's check the serial console to see what's going on.



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.



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).

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.

CircUp CLI Tool

There is a command line interface (CLI) utility called CircUp (https://adafru.it/Tfi) that can be used to easily install and update libraries on your device. Follow the directions on the install page within the CircUp learn guide (https://adafru.it/-Ad). Once you've got it installed you run the command `circup update` in a terminal to interactively update all libraries on the connected CircuitPython device. See the usage page in the CircUp guide (https://adafru.it/-Ah) for a full list of functionality

---

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 SAMD21. **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 SAMD21 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 SAMD21, 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) section below.

The results of `dir(board)` for CircuitPython compatible boards will look similar to the results for the QT Py SAMD21 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',
 'IO1', 'IO10', 'IO11', 'IO12', 'IO13', 'IO14', 'IO15', 'IO16', 'IO17', 'IO18',
'IO2', 'IO21', 'IO3', 'IO33', 'IO34', 'IO35', 'IO36', 'IO37', 'IO4', 'IO42', 'IO
45', 'IO5', 'IO6', 'IO7', 'IO8', 'IO9', '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, first, connect to the serial console.

In the example below, click the **Download Project Bundle** button below to download the necessary libraries and the **code.py** file in a zip file. Extract the contents of the zip file, open the directory **CircuitPython_Essentials/Pin_Map_Script/** and then click on the directory that matches the version of CircuitPython you're using and copy the contents of that directory to your **CIRCUITPY** drive.

Your **CIRCUITPY** drive should now look similar to the following image:


```
# SPDX-FileCopyrightText: 2020 anecdata for Adafruit Industries
# SPDX-FileCopyrightText: 2021 Neradoc for Adafruit Industries
# SPDX-FileCopyrightText: 2021-2023 Kattni Rembor for Adafruit Industries
# SPDX-FileCopyrightText: 2023 Dan Halbert for Adafruit Industries
#
# SPDX-License-Identifier: MIT

"""CircuitPython Essentials Pin Map Script"""
import microcontroller
import board
try:
    import cyw43  # raspberrypi
except ImportError:
    cyw43 = None

board_pins = []
for pin in dir(microcontroller.pin):
    if (isinstance(getattr(microcontroller.pin, pin), microcontroller.Pin) or
        (cyw43 and isinstance(getattr(microcontroller.pin, pin), cyw43.CywPin))):
        pins = []
        for alias in dir(board):
            if getattr(board, alias) is getattr(microcontroller.pin, pin):
                pins.append(f"board.{alias}")
        # Add the original GPIO name, in parentheses.
        if pins:
            # Only include pins that are in board.
            pins.append(f"({str(pin)})")
            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 SAMD21:

```
code.py output:
board.A0 board.D0 (PA02)
board.A1 board.D1 (PA03)
board.A10 board.D10 board.MOSI (PA10)
board.A2 board.D2 (PA04)
board.A3 board.D3 (PA05)
board.A6 board.D6 board.TX (PA06)
board.A7 board.D7 board.RX (PA07)
board.A8 board.D8 board.SCK (PA11)
board.A9 board.D9 board.MISO (PA09)
board.D4 board.SDA (PA16)
board.D5 board.SCL (PA17)
board.NEOPIXEL (PA18)
board.NEOPIXEL_POWER (PA15)
```

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 (PA02)`. This means that you can access pin **A0** in CircuitPython using both `board.A0` and `board.D0`.

The pins in parentheses are the microcontroller pin names. See the next section for more info on those.

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 SAMD21 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']
```

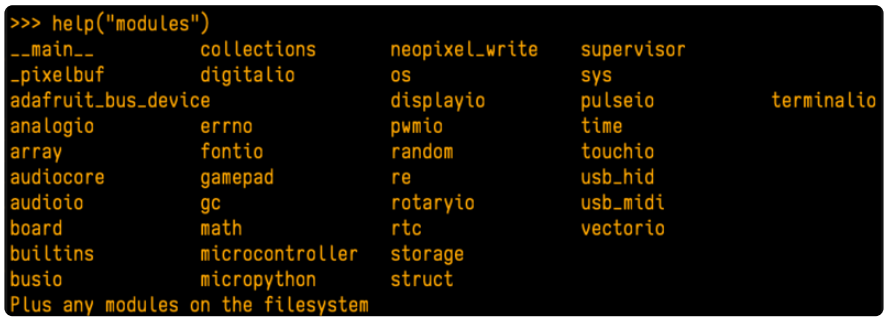
CircuitPython Built-In Modules

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) and the Python-like modules included here (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), 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")
```



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

Advanced Serial Console on Mac

Connecting to the serial console on Mac does not require installing any drivers or extra software. You'll use a terminal program to find your board, and `screen` to connect to it. Terminal and `screen` both come installed by default.

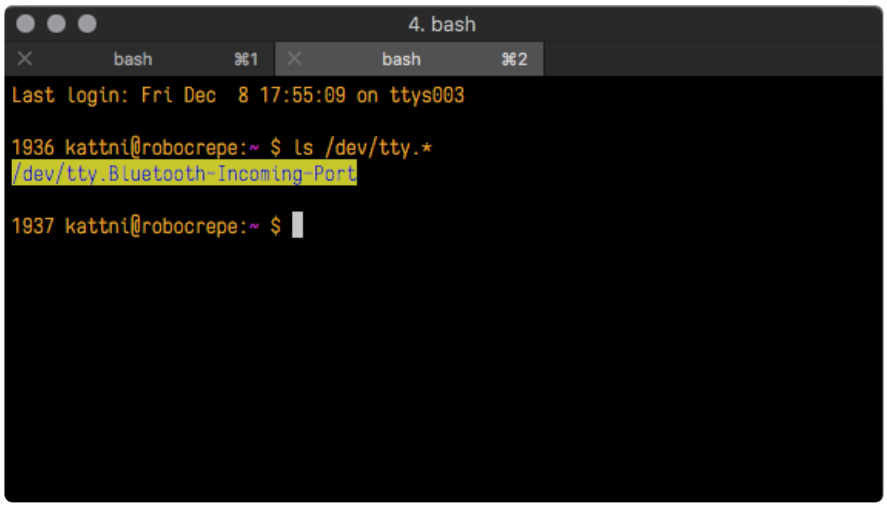
What's the Port?

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

The easiest way to determine which port the board is using is to first check **without** the board plugged in. Open Terminal and type the following:

`ls /dev/tty.*`

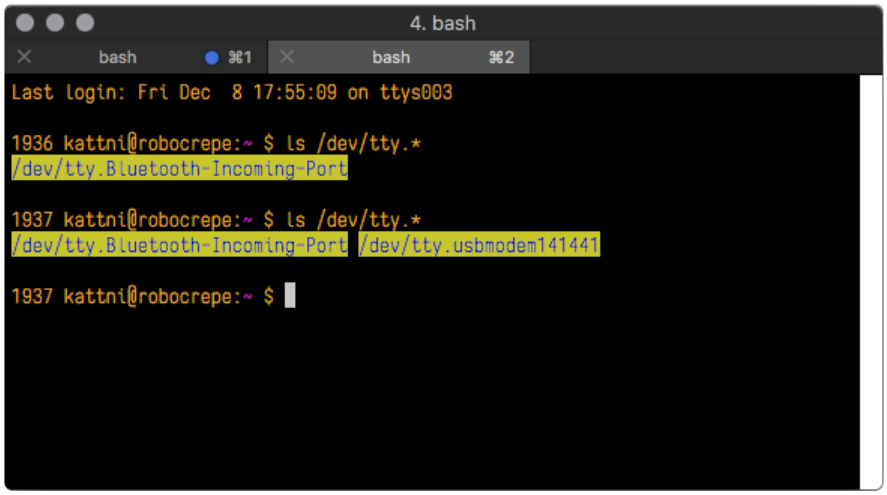
Each serial connection shows up in the `/dev/` directory. It has a name that starts with `tty.`. The command `ls` shows you a list of items in a directory. You can use `*` as a wildcard, to search for files that start with the same letters but end in something different. In this case, you're asking to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



Now, plug your board. In Terminal, type:

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

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



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

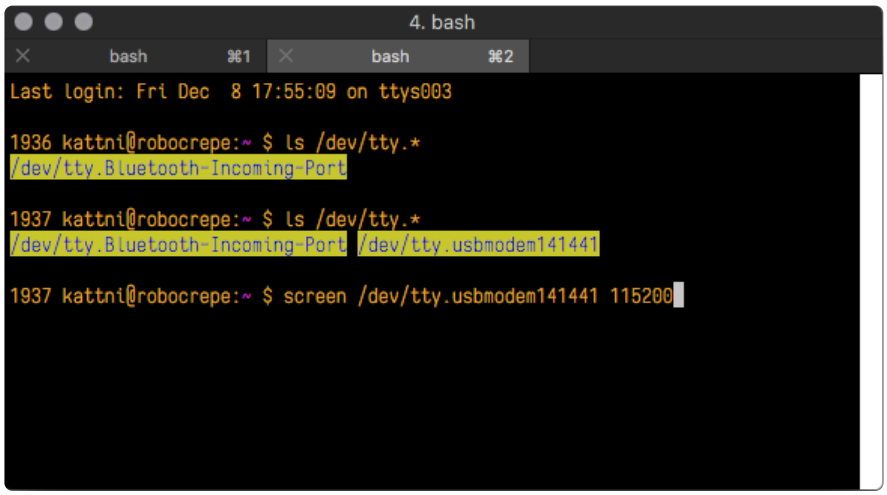
Using Linux, a new listing has appeared called `/dev/ttyACM0`. The `ttyACM0` part of this listing is the name the example board is using. Yours will be called something similar.

Connect with screen

Now that you know the name your board is using, you're ready connect to the serial console. You're going to use a command called `screen`. The `screen` command is included with MacOS. To connect to the serial console, use Terminal. Type the following command, replacing `board_name` with the name you found your board is using:

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

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



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

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

Frequently Asked Questions

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

What are some common acronyms to know?

CP or CPy = CircuitPython (https://adafru.it/KJD)
CPC = Circuit Playground Classic (http://adafru.it/3000) (does not run CircuitPython)
CPX = Circuit Playground Express (http://adafru.it/3333)
CPB = Circuit Playground Bluefruit (http://adafru.it/4333)

Using Older Versions

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.

I have to continue using CircuitPython 7.x or earlier. Where can I find compatible libraries?

We are no longer building or supporting the CircuitPython 7.x or earlier library bundles. We highly encourage you to update CircuitPython to the latest version **(https://adafru.it/Em8) and use** the current version of the libraries **(https://adafru.it/ENC).** However, if for some reason you cannot update, here are the last available library bundles for older versions:

- 2.x bundle (https://adafru.it/FJA)
- 3.x bundle (https://adafru.it/FJB)
- 4.x bundle (https://adafru.it/QDL)
- 5.x bundle (https://adafru.it/QDJ)
- 6.x bundle (https://adafru.it/Xmf)
- 7.x bundle (https://adafru.it/18e9)

Python Arithmetic

Does CircuitPython support floating-point numbers?

All CircuitPython boards support floating point arithmetic, even if the microcontroller chip does not support floating point in hardware. Floating point numbers are stored in 30 bits, with an 8-bit exponent and a 22-bit mantissa. Note that this is two bits less than standard 32-bit single-precision floats. You will get about 5-1/2 digits of decimal precision.

(The **broadcom** port may provide 64-bit floats in some cases.)

Does CircuitPython support long integers, like regular Python?

Python long integers (integers of arbitrary size) are available on most builds, except those on boards with the smallest available firmware size. On these boards, integers are stored in 31 bits.

Boards without long integer support are mostly SAMD21 ("M0") boards without an external flash chip, such as the Adafruit Gemma M0, Trinket M0, QT Py M0, and the Trinkey series. There are also a number of third-party boards in this category. There are also a few small STM third-party boards without long integer support.

`time.localtime()`, `time.mktime()`, `time.time()`, and `time.monotonic_ns()` are available only on builds with long integers.

Wireless Connectivity

How do I connect to the Internet with CircuitPython?

If you'd like to include WiFi in your project, your best bet is to use a board that is running natively on ESP32 chipsets - those have WiFi built in!

If your development board has an SPI port and at least 4 additional pins, you can check out this guide (https://adafru.it/F5X) on using AirLift with CircuitPython - extra wiring is required and some boards like the MacroPad or NeoTrellis do not have enough available pins to add the hardware support.

For further project examples, and guides about using AirLift with specific hardware, check out the Adafruit Learn System (https://adafru.it/VBr).

How do I do BLE (Bluetooth Low Energy) with CircuitPython?

nRF52840, nRF52833, and **as of CircuitPython 9.1.0**, ESP32, ESP32-C3, and ESP32-S3 boards (with 8MB) have the most complete BLE implementation. Your program can act as both a BLE central and

peripheral. As a central, you can scan for advertisements, and connect to an advertising board. As a peripheral, you can advertise, and you can create services available to a central. Pairing and bonding are supported.

Most Espressif boards with only 4MB of flash do not have enough room to include BLE in CircuitPython 9. Check the Module Support Matrix (https://adafru.it/-Cy) to see if your board has support for `_bleio`. CircuitPython 10 is planned to support _bleio on Espressif boards with 4MB flash.

Note that the ESP32-S2 does not have Bluetooth capability.

On most other boards with adequate firmware space, BLE is available for use with AirLift (https://adafru.it/11Av) or other NINA-FW-based co-processors. Some boards have this coprocessor on board, such as the PyPortal (https://adafru.it/11Aw). Currently, this implementation only supports acting as a BLE peripheral. Scanning and connecting as a central are not yet implemented. Bonding and pairing are not supported.

Are there other ways to communicate by radio with CircuitPython?

Check out Adafruit's RFM boards (https://adafru.it/11Ay)for simple radio communication supported by CircuitPython, which can be used over distances of 100m to over a km, depending on the version. The RFM SAMD21 M0 boards can be used, but they were not designed for CircuitPython, and have limited RAM and flash space; using the RFM breakouts or FeatherWings with more capable boards will be easier.

Asyncio and Interrupts

Is there asyncio support in CircuitPython?

There is support for asyncio starting with CircuitPython 7.1.0, on all boards except the smallest SAMD21 builds. Read about using it in the Cooperative Multitasking in CircuitPython (https://adafru.it/XnA) Guide.

Does CircuitPython support interrupts?

No. CircuitPython does not currently support interrupts - please use asyncio for multitasking / 'threaded' control of your code

Status RGB LED

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

The status LED can tell you what's going on with your CircuitPython board. Read more here for what the colors mean! (https://adafru.it/Den)

Memory Issues

What is a MemoryError?

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

What do I do when I encounter a MemoryError?

Try resetting your board. Each time you reset the board, it reallocates the memory. While this is unlikely to resolve your issue, it's a simple step and is worth trying.

Make sure you are using **.mpy** versions of libraries. All of the CircuitPython libraries are available in the bundle in a **.mpy** format which takes up less memory than **.py** format. Be sure that you're using the

latest library bundle (https://adafru.it/uap) for your version of CircuitPython.

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

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

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

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

How can I create my own .mpy files?

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

You can download **mpy-cross** for your operating system from here (https://adafru.it/QDK). Builds are available for Windows, macOS, x64 Linux, and Raspberry Pi Linux. Choose the latest **mpy-cross** whose version matches the version of CircuitPython you are using.

On macOS and Linux, after you download **mpy-cross**, you must make the the file executable by doing `chmod +x name-of-the-mpy-cross-executable`.

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

How do I check how much memory I have free?

Run the following to see the number of bytes available for use:

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

Unsupported Hardware

Is ESP8266 or ESP32 supported in CircuitPython? Why not?

We dropped ESP8266 support as of 4.x - For more information please read about it here (https://adafru.it/CiG)!

As of CircuitPython 8.x we have started to support ESP32 and ESP32-C3 and have added a WiFi workflow for wireless coding! (https://adafru.it/10JF)

We also support ESP32-S2 & ESP32-S3, which have native USB.

Does Feather M0 support WINC1500?

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

Can AVRs such as ATmega328 or ATmega2560 run CircuitPython?

No.

ESP32-S2 Bugs & Limitations

Nobody likes bugs, but all nontrivial software and hardware has some. The master list of problems is the Issues list on github (https://adafru.it/188d).

I2C at 100 kHz bus frequency runs slowly

The default I2C bus clock speed is 100 kHz (100000) . At that rate, the ESP32-S2 will leave 10ms (https://adafru.it/ZCc) gaps between I2C transactions. This can slow down your I2C interactions considerably, such as when you are controlling a stepper motor with a PCA9685 controller.

Raising the I2C bus frequency to 125 kHz (125000) or higher fixes this problem. If your I2C peripheral can handle higher frequencies, you can use 400 kHz (400000) or even in some cases 1 MHz (1000000).

Note that `board.I2C()` creates an I2C bus that runs at 100 kHz. The bus frequency cannot be changed.. To create an I2C bus on the default I2C pins that runs at a different frequency, you must use `busio.I2C(board.SCL, board.SDA, frequency=)`.

No DAC-based audio output

Current versions of the ESP-IDF SDK do not have the required APIs for DAC-based audio output. Once a future version of ESP-IDF that adds it, it will be possible to implement DAC-based AudioOut in CircuitPython.

Workaround: PWMOut can create tones and buzzes.

Workaround: I2SOut audio is currently being developed and will work with boards such as the I2S 3W Class D Amplifier Breakout - MAX98357A (http://adafru.it/3006).

Deep Sleep & Wake-up sources

ESP32-S2 has hardware limitations on what kind of "pin alarms" can wake it. The following combinations are possible:

- EITHER one or two pins that wake from deep sleep when they are pulled LOW
- OR an arbitrary number of pins that wake from deep sleep when they are pulled HIGH, and optionally one pin that wakes from deep sleep when pulled LOW

This means that "wake" buttons should be wired so that pressing them pulls HIGH and a pull DOWN resistor is used with the pin. However, in some hardware designs including the original MagTag, the integrated buttons are pulled LOW when pressed and so only 1 or 2 buttons can be selected to wake the MagTag.

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).**

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

As 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 7.x or earlier. Where can I find compatible libraries?

Adafruit is no longer building or supporting the CircuitPython 7.x or earlier library bundles. You are highly encourged to update CircuitPython to the latest version (https://adafru.it/Em8) and use the current version of the libraries (https://adafru.it/ENC). However, if for some reason you cannot update, links to the previous bundles are available in the FAQ (https://adafru.it/FwY).

macOS Sonoma before 14.4: Errors Writing to CIRCUITPY
macOS 14.4 - 15.1: Slow Writes to CIRCUITPY

macOS Sonoma before 14.4 took many seconds to complete writes to small FAT drives, 8MB or smaller. This causes errors when writing to CIRCUITPY. The best solution was to remount the CIRCUITPY drive after it is automatically mounted. Or consider downgrading back to Ventura if that works for you. This problem was tracked in CircuitPython GitHub issue 8449 (https://adafru.it/18ea).

Below is a shell script to do this remount conveniently (courtesy @czei in GitHub (https://adafru.it/18ea)). Copy the code here into a file named, say, **remount-CIRCUITPY.sh**. Place the file in a directory on your PATH, or in some other convenient place.

macOS Sonoma 14.4 and versions of macOS before Sequoia 15.2 did not have the problem above, but did take an inordinately long time to write to FAT drives of size 1GB or less (40 times longer than 2GB drives). As of macOS 15.2, writes are no longer very slow. This problem was tracked in CircuitPython GitHub issue 8918 (https://adafru.it/19iD).

```
#!/bin/sh
#
# This works around bug where, by default,
# macOS 14.x before 14.4 writes part of a file immediately,
# and then doesn't update the directory for 20-60 seconds, causing
# the file system to be corrupted.
#

disky=`df | grep CIRCUITPY | cut -d" " -f1`
sudo umount /Volumes/CIRCUITPY
sudo mkdir /Volumes/CIRCUITPY
sleep 2
sudo mount -v -o noasync -t msdos $disky /Volumes/CIRCUITPY
```

Then in a Terminal window, do this to make this script executable:

```
chmod +x remount-CIRCUITPY.sh
```

Place the file in a directory on your `PATH`, or in some other convenient place.

Now, each time you plug in or reset your CIRCUITPY board, run the file **remount-CIRCUITPY.sh**. You can run it in a Terminal window or you may be able to place it on the desktop or in your dock to run it just by double-clicking.

This will be something of a nuisance but it is the safest solution.

This problem is being tracked in this CircuitPython issue (https://adafru.it/18ea).

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)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) 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 accompanything **SAT SMART Driver** can interfere with seeing the BOOT drive. See this forum post (https://adafru.it/sTc) for how to fix the problem.

Windows 10 or later

Did you install the Adafruit Windows Drivers package by mistake, or did you upgrade to Windows 10 or later with the driver package installed? You don't need to install this package on Windows 10 or 11for 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

Windows 7 and 8.1 have reached end of life. It is recommended (https://adafru.it/Amd) that you upgrade to Windows 10 or 11 if possible. Drivers are available for some older CircuitPython boards, but there are no plans to release drivers for newer boards.

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 available. There are no plans to release drivers for newer boards. The boards work fine on Windows 10 and later.

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

Let us know in the Adafruit support forums (https://adafru.it/jlf) 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 or Disappears Quickly

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.

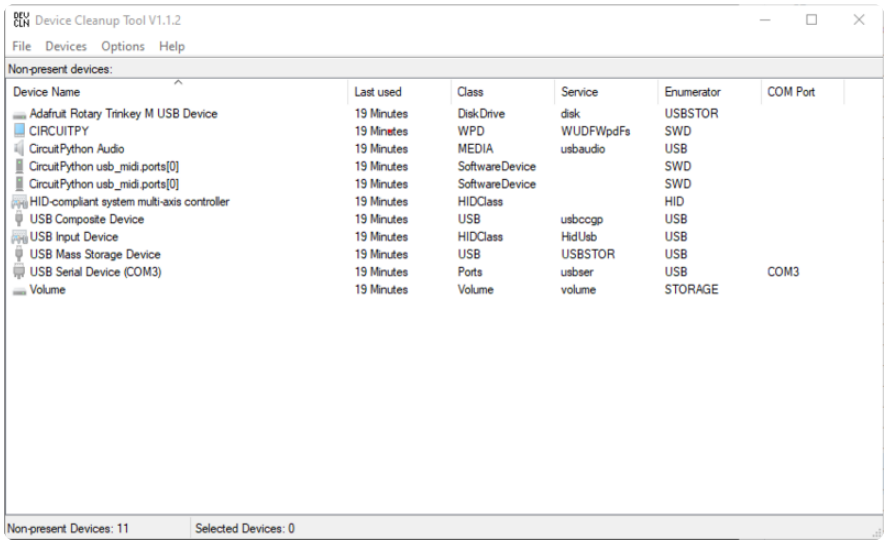
Norton anti-virus can interfere with **CIRCUITPY**. A user has reported this problem on Windows 7. The user turned off both Smart Firewall and Auto Protect, and **CIRCUITPY** then appeared.

Sophos Endpoint security software can cause CIRCUITPY to disappear (https://adafru.it/ELr) and the BOOT drive to reappear. It is not clear what causes this behavior.

Samsung Magician can cause CIRCUITPY to disappear (reported here (https://adafru.it/18eb) and here (https://adafru.it/18ec)).

Device Errors or Problems on Windows

Windows can become confused about USB device installations. Try cleaning up your USB devices. Use Uwe Sieber's Device Cleanup Tool (https://adafru.it/RWd) (on that page, scroll down to "Device Cleanup Tool"). 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

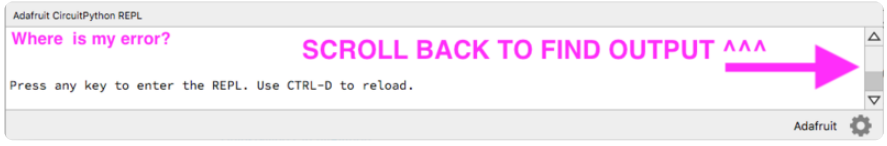
There are times when the serial console will accurately not display anything, such as, when no code is currently running, or when code with no serial output is already running before you open the console. However, if you find yourself in a situation where you feel it should be displaying something like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

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.runtime.autoreload = False
```

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

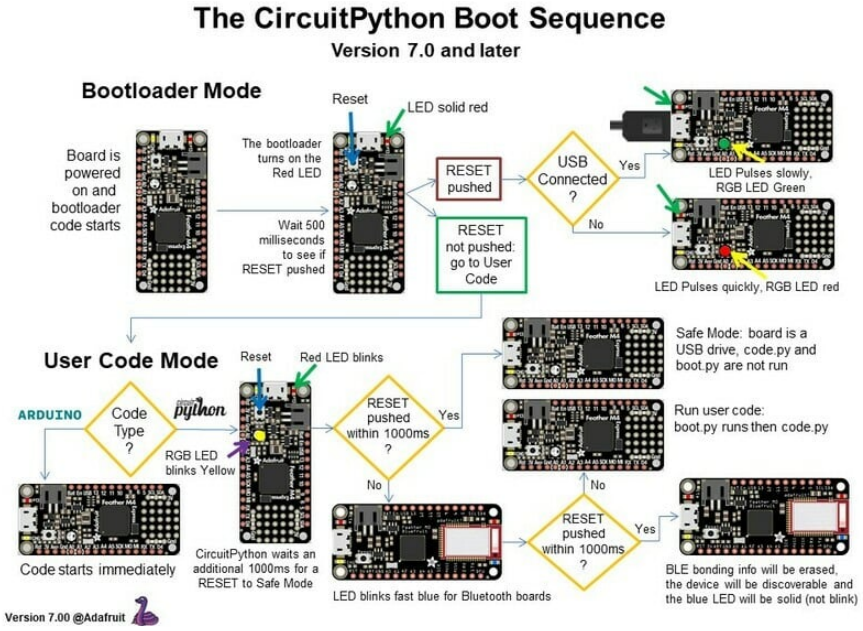
The status LED blinks were changed in CircuitPython 7.0.0 in order to save battery power and simplify the blinks. These blink patterns will occur on single color LEDs when the board does not have any RGB LEDs. Speed and blink count also vary for this reason.

On start up, the LED will blink **YELLOW** multiple times for 1 second. Pressing the RESET button (or on Espressif, the BOOT button) 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

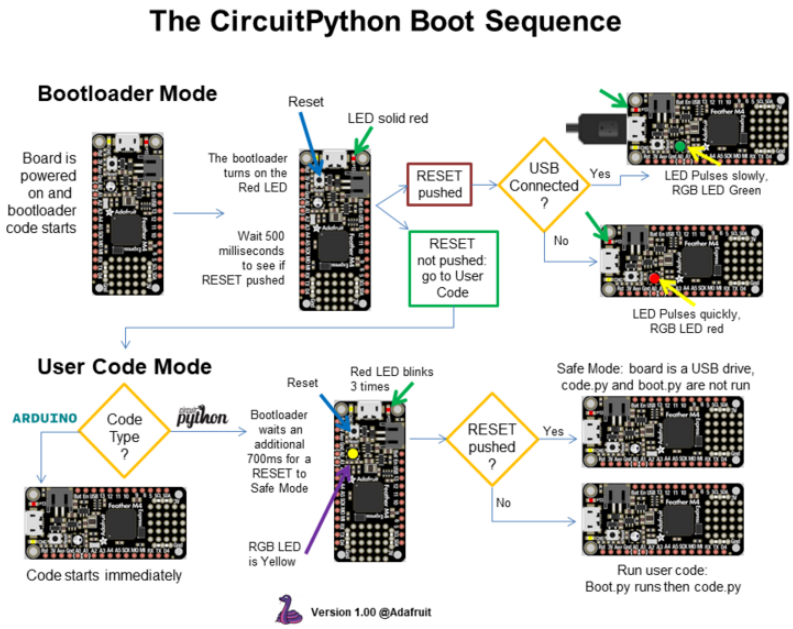
Here's what the colors and blinking mean:

- 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.



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).

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 and Later

You can enter safe by pressing reset during the right time when the board boots. Immediately after the board starts up or resets, it waits one second. On some boards, the onboard status LED will blink yellow during that time. If you press reset during that one second period, 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

You can enter safe by pressing reset during the right time when the board boots.. Immediately after the board starts up or resets, it waits 0.7 seconds. On some boards, the onboard status LED (highlighted in green above) will turn solid yellow during this time. If you press reset during that 0.7 seconds, 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) to do this.

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

```
&gt;&gt;&gt; import storage
&gt;&gt;&gt; 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

RP2040 boards (flash_nuke.uf2)

https://adafru.it/18ed

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) **.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). 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) **.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) YYou'll also need to load your code and reinstall your libraries!

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

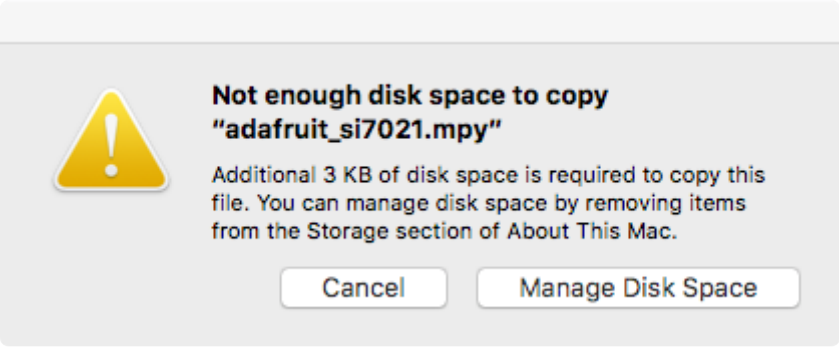
Any SAMD21-based microcontroller that does not have external flash available is considered a SAMD21 non-Express board. Non-Express boards that do **not** have a UF2 bootloader include the Feather M0 Basic Proto, Feather Adalogger, or the Arduino Zero.

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), 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) 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:

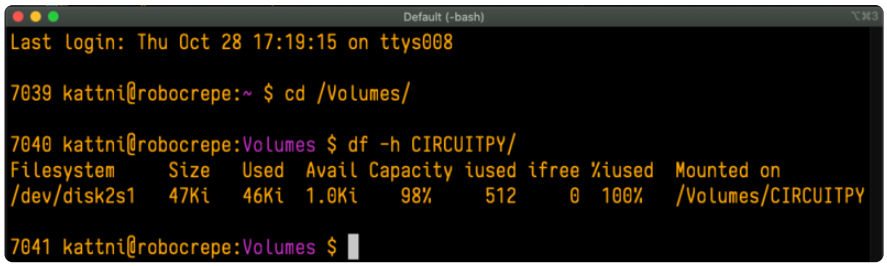
```
cp -rX folder_to_copy /Volumes/CIRCUITPY
```

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

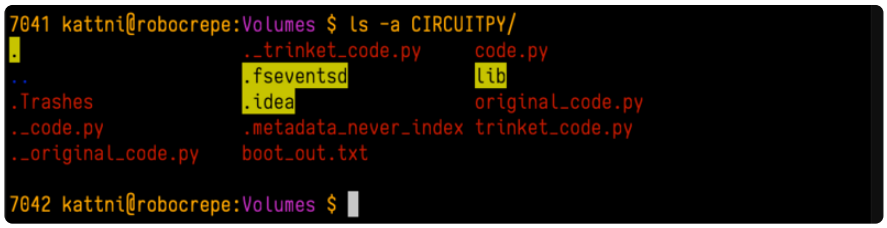
```
# if lib does not exist, you'll create a file named lib !
cp -X file_name.mpy /Volumes/CIRCUITPY/lib
# This is safer, and will complain if a lib folder does not exist.
cp -X file_name.mpy /Volumes/CIRCUITPY/lib/
```

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.

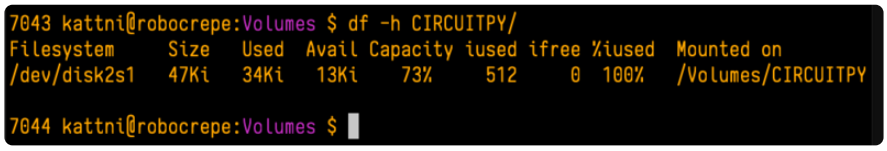


That's not very much space left! The next step is to show a list of the files currently on the **CIRCUITPY** drive, including the hidden files, using the `ls` command. You cannot use Finder to do this, you must do it via command line!



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.

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



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

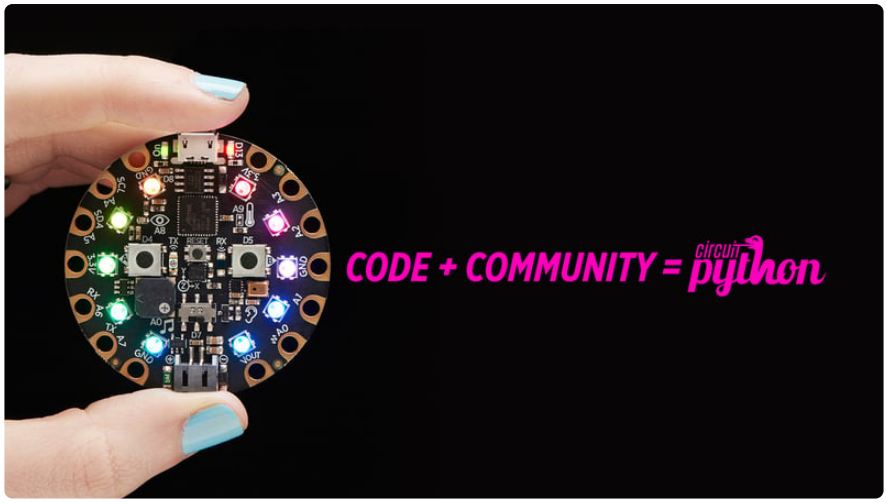
Device Locked Up or Boot Looping

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.

For more information on safe mode and how to enter safe mode, see the Safe Mode section on this page (https://adafru.it/Den).

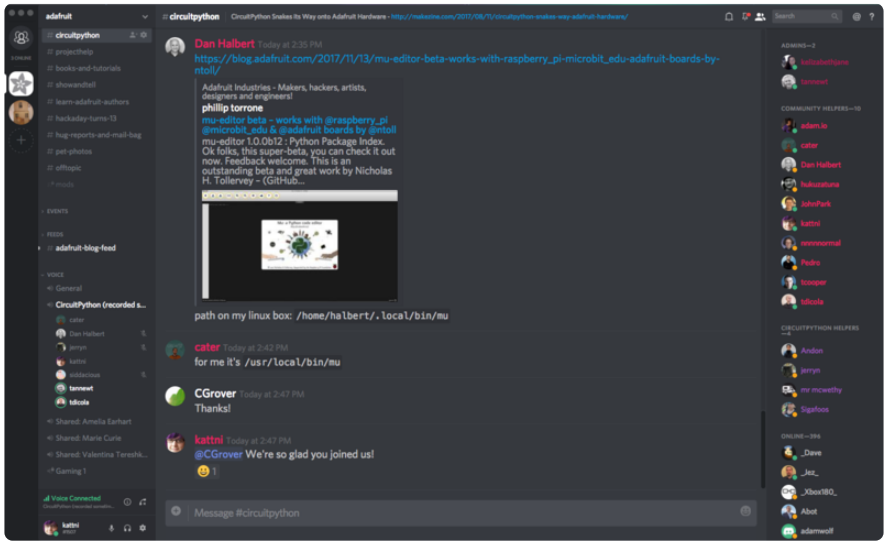
Welcome to the Community!



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.

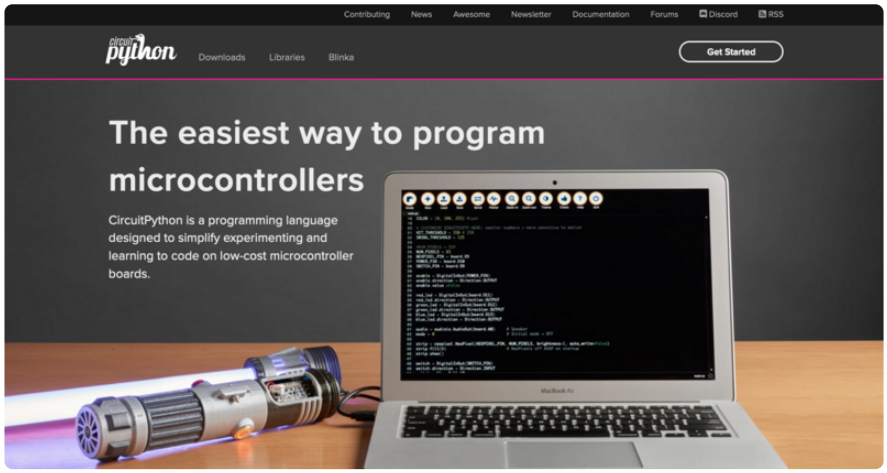
The help with CircuitPython channel is where to go with your CircuitPython questions. #help-with-circuitpython is there for new users and developers alike so feel free to ask a question or post a comment! Everyone of any experience level is welcome to join in on the conversation. Your contributions are 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.

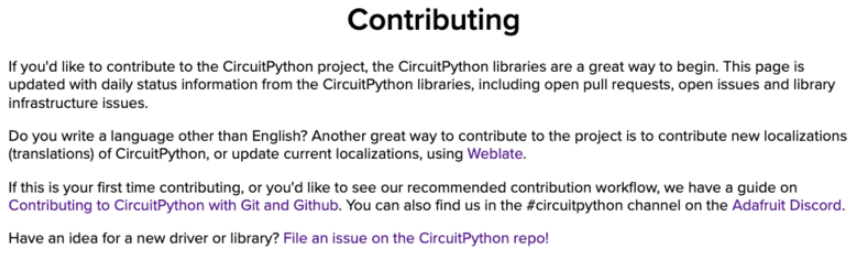
The Adafruit Discord is the 24x7x365 hackerspace that you can bring your granddaughter to.

Visit https://adafru.it/discord ()to sign up for Discord. Everyone is looking forward to meeting you!

CircuitPython.org

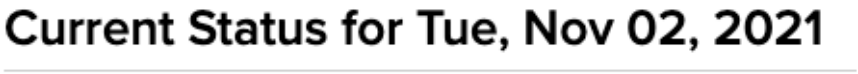


Beyond the Adafruit Learn System, which you are viewing right now, the best place to find information about CircuitPython is circuitpython.org (https://adafru.it/KJD). Everything you need to get started with your new microcontroller and beyond is available. You can do things like download CircuitPython for your microcontroller (https://adafru.it/Em8) or download the latest CircuitPython Library bundle (https://adafru.it/ENC), or check out which single board computers support Blinka (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).



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). 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**:

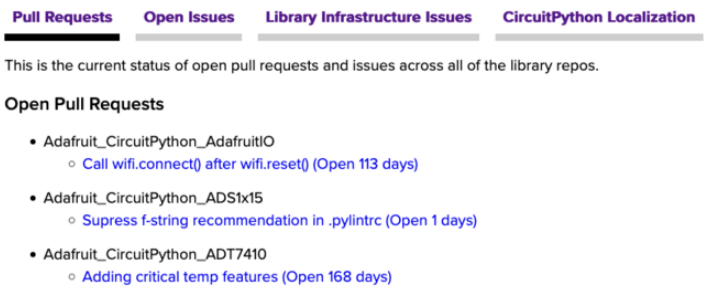


If you submit any contributions to the libraries, and do not see them reflected on the Contributing page, it could be that the job that checks for new updates hasn't yet run for today. Simply check back tomorrow!

Now, a look at the different options.

Pull Requests

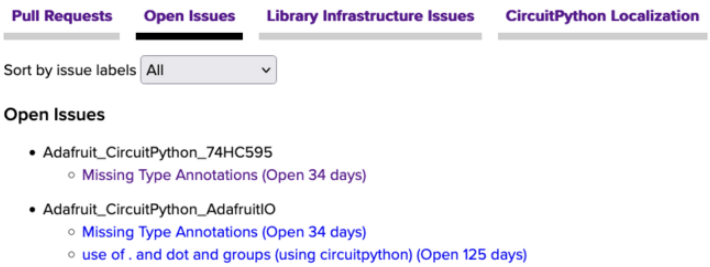
The first tab you'll find is a list of **open pull requests**.



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

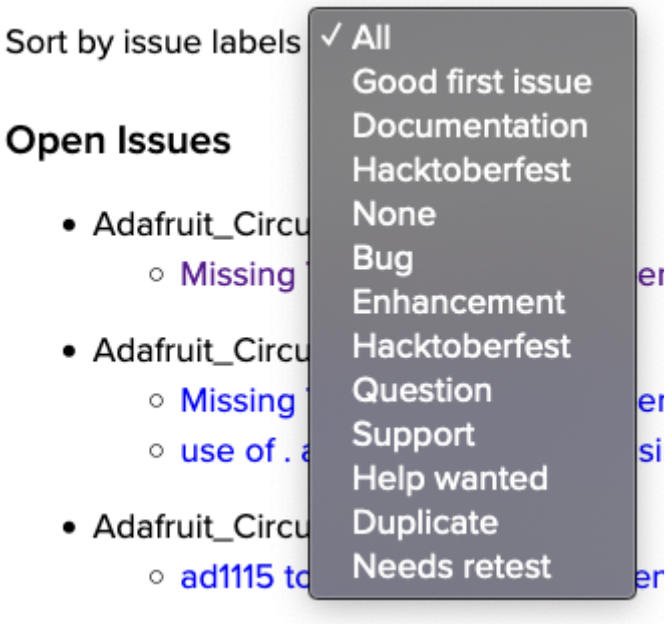
The second tab you'll find is a list of **open issues**.



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.



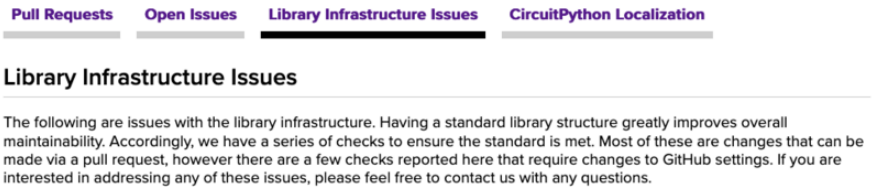
If you're new to everything, new to contributing to open source, or new to contributing to the CircuitPython project, you can choose "Good first issue". Issues with that label are well defined, with a finite scope, and are intended to be easy for someone new to figure out.

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) to walk you through the entire process. As well, there are always folks available on Discord () to answer questions.

Library Infrastructure Issues

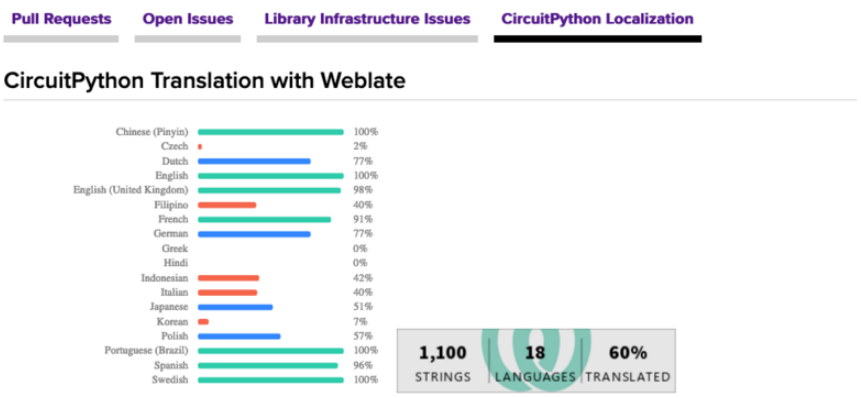
The third tab you'll find is a list of **library infrastructure issues**.



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

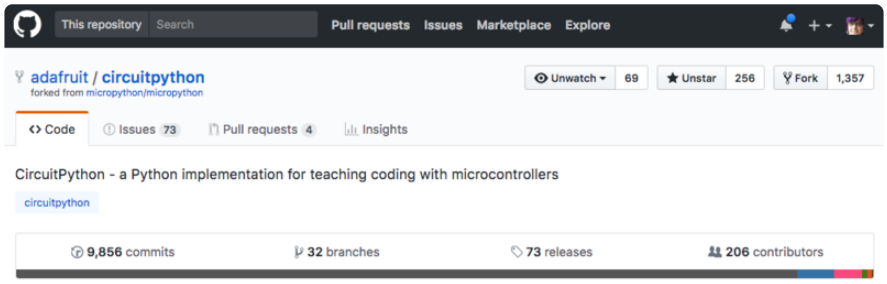
The fourth tab you'll find is the **CircuitPython Localization** tab.



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.

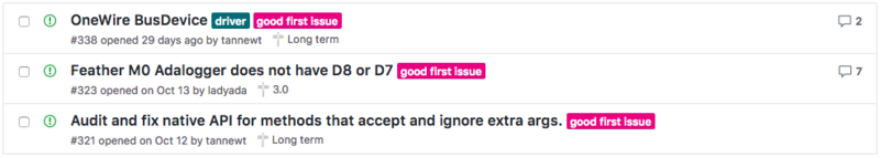
Regardless of your skill level, or how you want to contribute to the CircuitPython project, there is an opportunity available. The Contributing page (https://adafru.it/VD7) is an excellent place to start!

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), and the CircuitPython libraries (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)", and you'll find a list that includes issues labeled "good first issue (https://adafru.it/188e)". For the libraries, head over to the Contributing page Issues list (https://adafru.it/VFv), and use the drop down menu to search for "good first issue (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).



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!

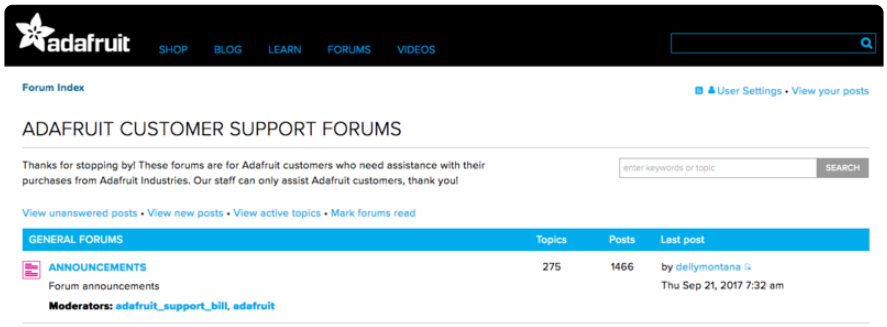
When working with or using CircuitPython or the CircuitPython libraries, you may find problems. If you find a bug, that's great! The team loves bugs! Posting a detailed issue to GitHub is an invaluable way to contribute to improving CircuitPython. For CircuitPython itself, file an issue here (https://adafru.it/tBb). 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!

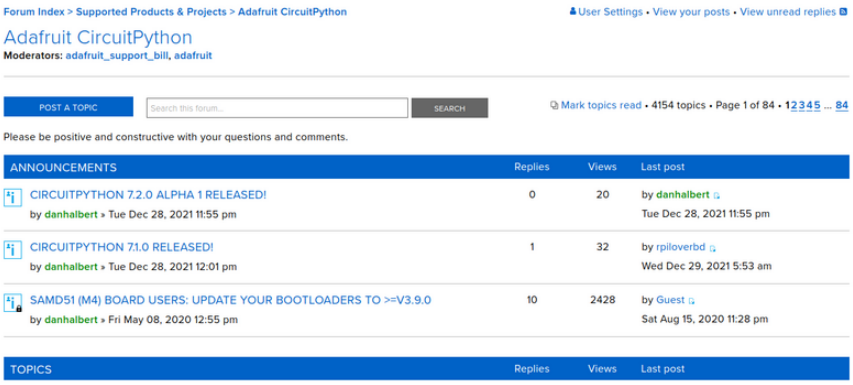
Adafruit Forums



The Adafruit Forums (https://adafru.it/jlf) 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.

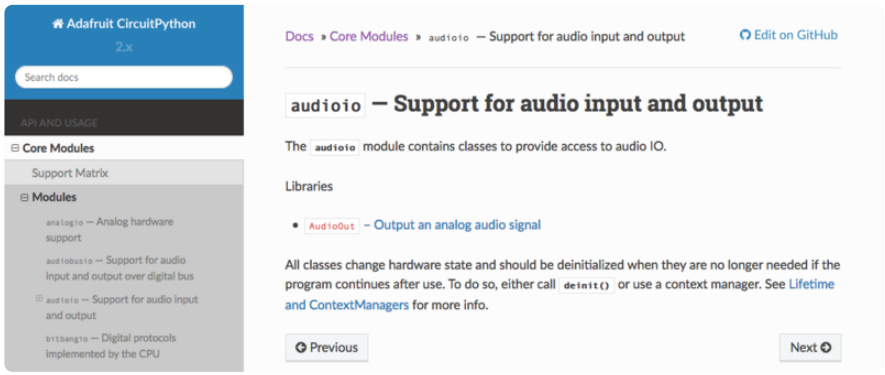
There are forum categories that cover all kinds of topics, including everything Adafruit. The Adafruit CircuitPython (https://adafru.it/xXA) category under "Supported Products & Projects" is the best place to post your CircuitPython questions.



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) is a 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) 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)
```

Arduino IDE Setup

The ESP32-S2/S3 bootloader does not have USB serial support for Windows 7 or 8. (See https://github.com/espressif/arduino-esp32/issues/5994) please update to version 10 which is supported by espressif! Alternatively you can try this community-crafted Windows 7 driver (https://github.com/kutukvpavel/Esp32-Win7-VCP-drivers)

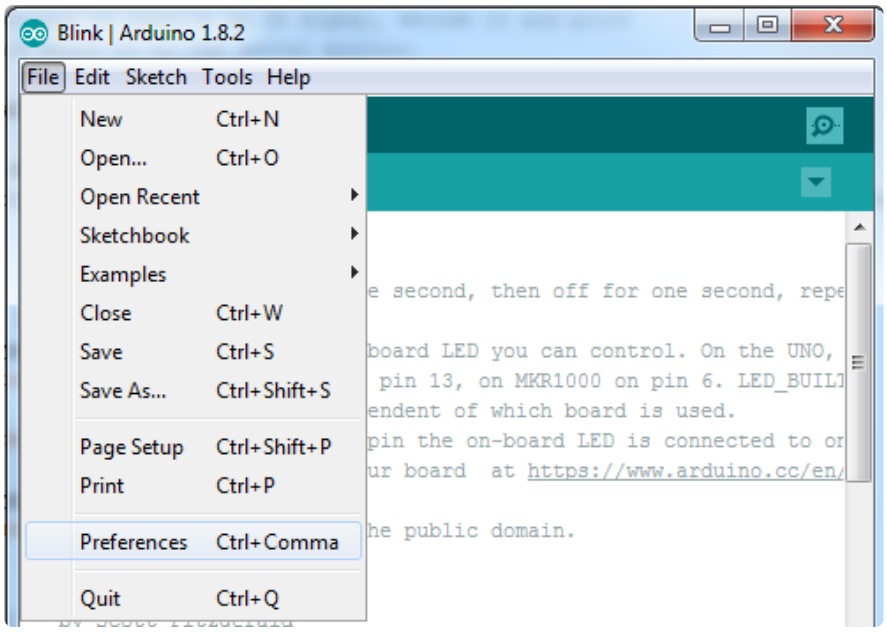
The first thing you will need to do is to download the latest release of the Arduino IDE. You will need to be using **version 1.8** or higher for this guide

Arduino IDE Download

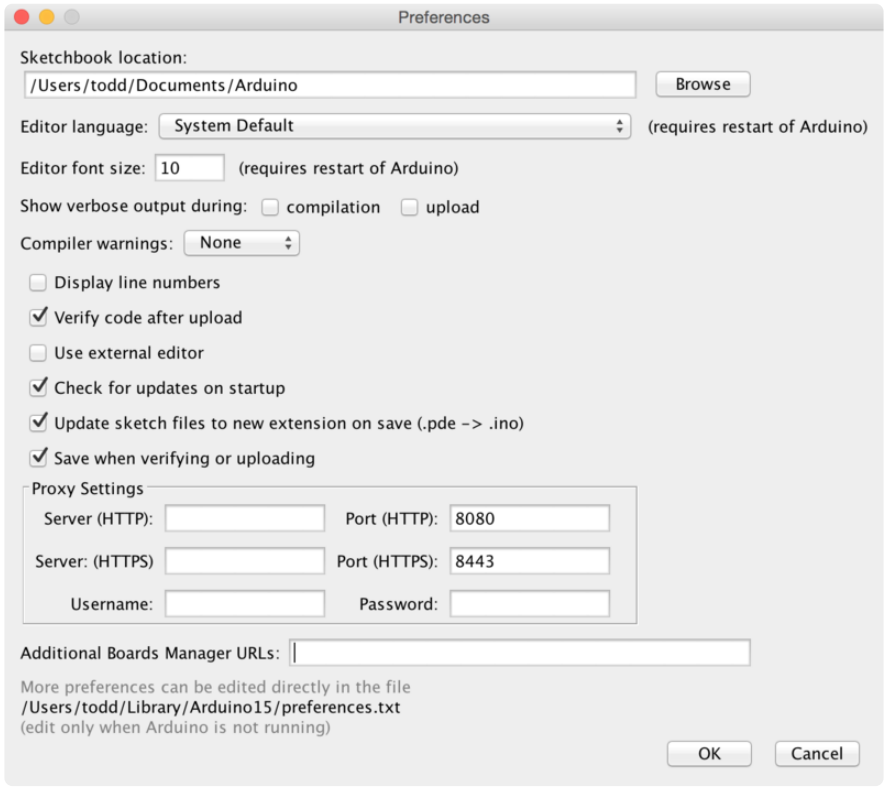
https://adafru.it/f1P

To use the ESP32-S2/S3 with Arduino, you'll need to follow the steps below for your operating system. You can also check out the Espressif Arduino repository for the most up to date details on how to install it (https://adafru.it/weF).

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



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

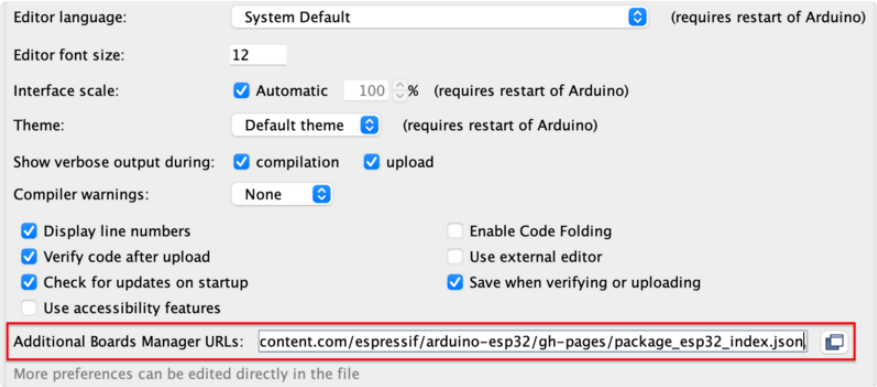


We will be adding a URL to the new **Additional Boards Manager URLs** option. The list of URLs is comma separated, and you will only have to add each URL once. New Adafruit boards and updates to existing boards will automatically be picked up by the Board Manager each time it is opened. The URLs point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

To find the most up to date list of URLs you can add, you can visit the list of third party board URLs on the Arduino IDE wiki (https://adafru.it/f7U). We will only need to add

one URL to the IDE in this example, but **you can add multiple URLS by separating them with commas**. Copy and paste the link below into the **Additional Boards Manager URLs** option in the Arduino IDE preferences.

```
https://raw.githubusercontent.com/espressif/arduino-esp32/gh-pages/package_esp32_index.json
```



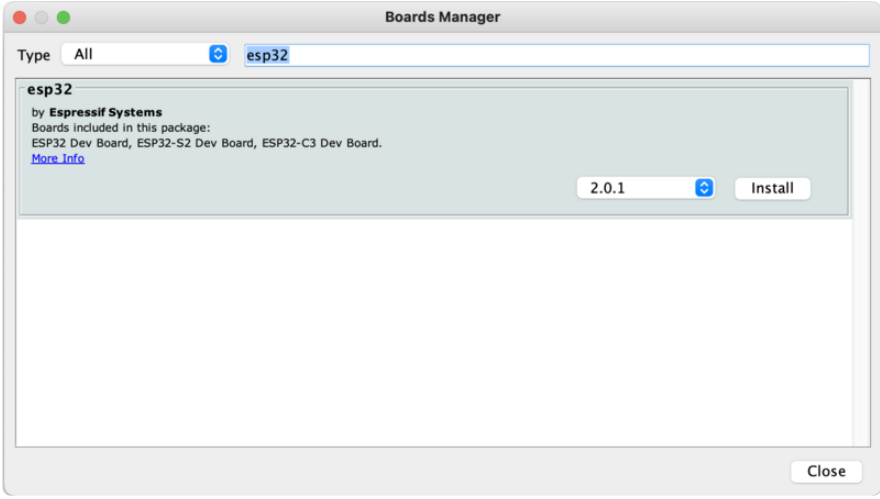
If you're an advanced hacker and want the 'bleeding edge' release that may have fixes (or bugs!) you can check out the dev url instead:

```
https://raw.githubusercontent.com/espressif/arduino-esp32/gh-pages/package_esp32_dev_index.json
```

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.

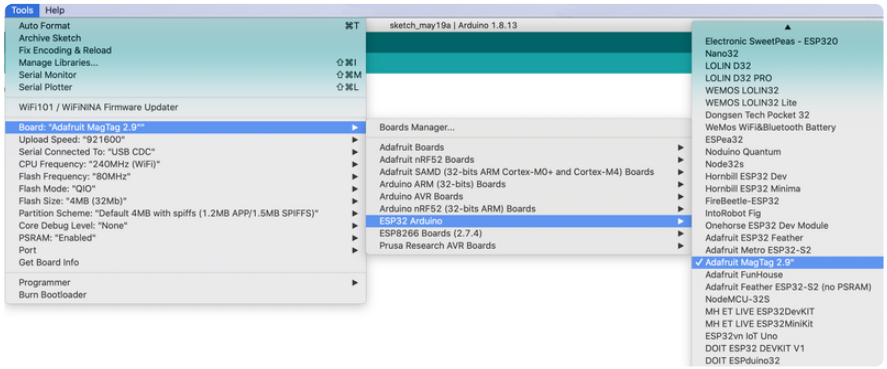
The next step is to actually install the Board Support Package (BSP). Go to the **Tools → Board → Board Manager** submenu. A dialog should come up with various BSPs. Search for **esp32**. Choose the latest version, which may be later than the version shown in the screenshot below.



Click the **Install** button and wait for it to finish. Once it is finished, you can close the dialog.

In the **Tools → Board** submenu you should see **ESP32 Arduino** and in that dropdown it should contain the ESP32 boards along with all the latest ESP32-S2/S3 boards.

Look for the board called **Adafruit MagTag 2.9"**.



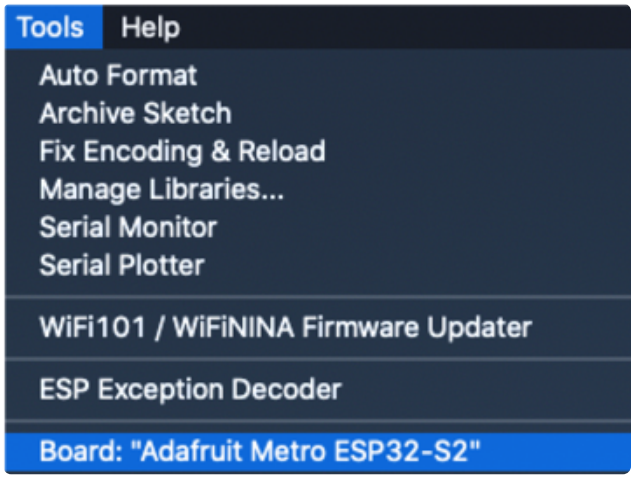
Using with Arduino IDE

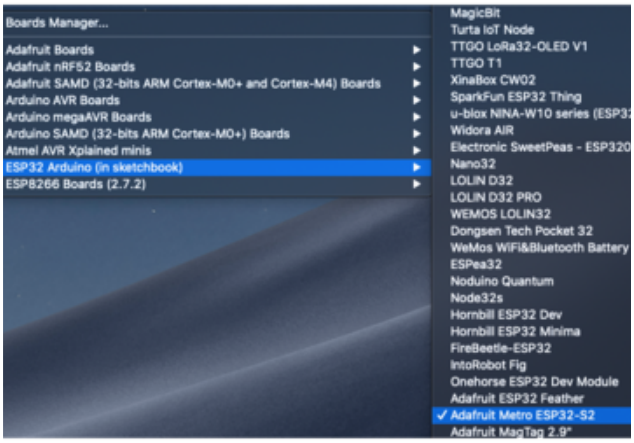
Blink

Now you can upload your first blink sketch!

Plug in the ESP32-S2/S3 board and wait for it to be recognized by the OS (just takes a few seconds).

Select ESP32-S2/S3 Board in Arduino IDE





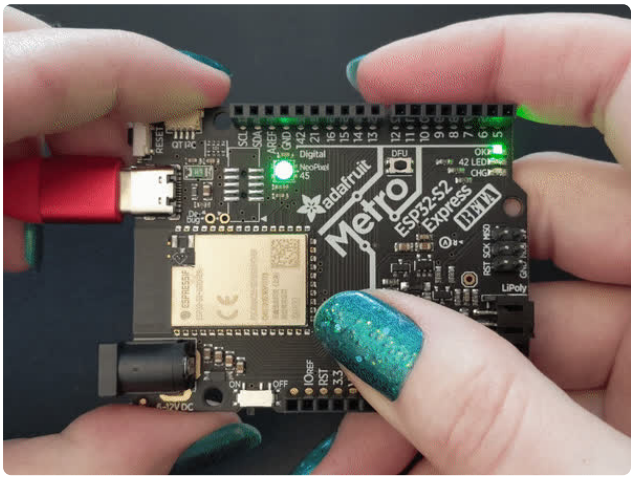
On the Arduino IDE, **click**:

Tools -> Board -> ESP32 Arduino -> Your Adafruit ESP32-S2/S3 board

The screenshot shows Metro S2 but you may have a different board. Make sure the name matches the exact product you purchased. If you don't see your board, make sure you have the latest version of the ESP32 board support package

Launch ESP32-S2/S3 ROM Bootloader

ESP32-S2/S3 support in Arduino uses native USB which can crash. If you ever DON'T see a serial/COM port, you can always manually enter bootloading mode. This bootloader is in ROM, it is 'un-brickable' so you can always use this technique to get into the bootloader. However, after uploading your Arduino code you MUST press reset to start the sketch



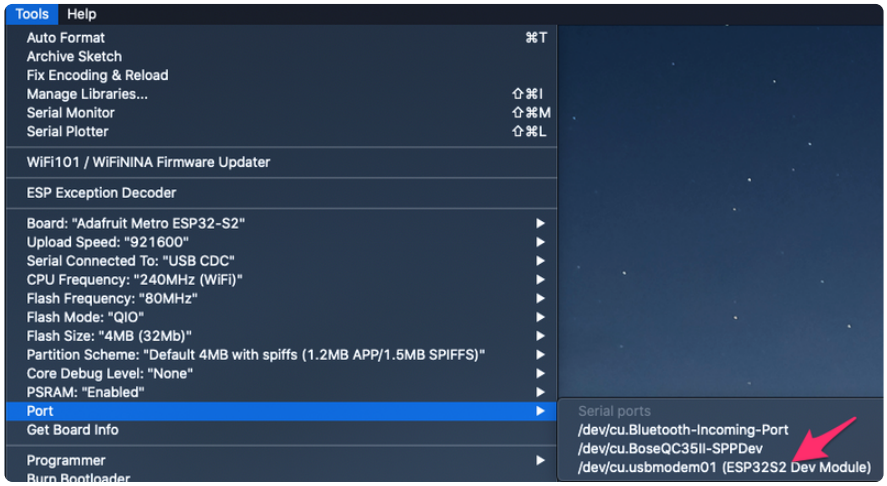
Before we upload a sketch, place your ESP32-S2/S3 board into ROM bootloader mode (https://adafru.it/OsC).

Look for the Reset button and a second DFU / BOOT0 button

HOLD down the DFU/Boot0 button while you click Reset. Then release DFU/Boot0 button

The GIF shows a Metro S2 but your board may look different. It will still have BOOT and Reset buttons somewhere

It should appear under **Tools -> Port** as **ESP32-S2/S3 Dev Module**.



Do not select any other port than the one that is called "ESP32S2 Dev Module" or "ESP32S3 Dev Module"

Load Blink Sketch

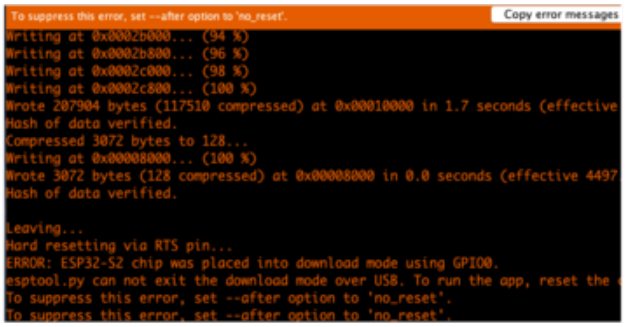
Now open up this Blink example in a new sketch window

```
// the setup function runs once when you press reset or power the board
void setup() {
  // initialize built in LED pin as an output.
  pinMode(LED_BUILTIN, OUTPUT);
  // initialize USB serial converter so we have a port created
  Serial.begin();
}
```

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

> Note that we use LED_BUILTIN not pin 13 for the LED pin. That's because we don't always use pin 13 for the LED on boards. For example, on the Metro ESP32-S2 the LED is on pin 42!

And click upload! After uploading, you may see something like this:

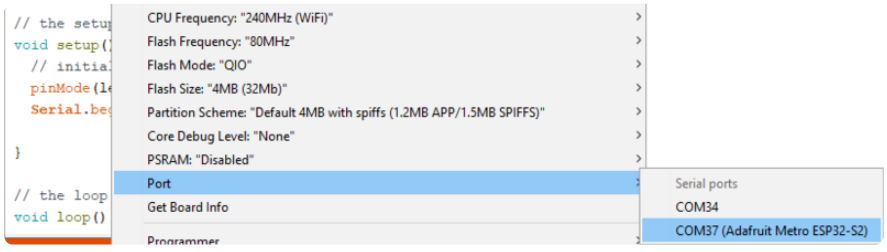


And click upload! After uploading, you may see something like this, warning you that we could not get out of reset.

This is normal! **Press the RESET button on your board to launch the sketch**

That's it, you will be able to see the red LED blink. You will also see a new serial port created.

You may call `Serial.begin();` in your sketch to create the serial port so don't forget it, it is not required for other Arduinos or previous ESP boards!



You can now select the **new serial port** name which will be different than the bootloader serial port. Arduino IDE will try to use auto-reset to automatically put the board into bootloader mode when you ask it to upload new code

If you ever DON'T see a serial port, or something is not working out with upload you can always manually enter bootloader mode:

- Reset board into ROM bootloader with DFU/BOOT0 + Reset buttons
- Select the ESP32S2/S3 Dev Board ROM bootloader serial port in Tools->Port menu
- Upload sketch
- Click reset button to launch code

Arduino Basics

Arduino support for the ESP32-S2 is very challenging right now, we don't recommend it. Please use CircuitPython!

Once you have Arduino installed and set up and you can upload simple blink sketches, you can move on to using each element of the MagTag board.

Using the Red LED

It's always good to blink the LED when you want to verify if something is happening on your board. The LED is on IO #13, but we recommend you use the `LED_BUILTIN` macro and you can use this simple sketch example to blink the LED:

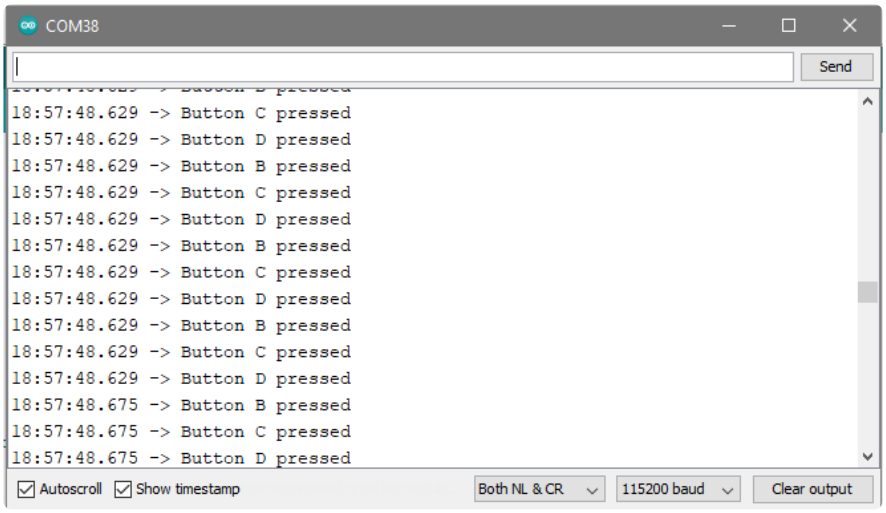
```
void setup() {
  // initialize built in LED pin as an output.
  pinMode(LED_BUILTIN, OUTPUT);
  // initialize USB serial converter so we have a port created
  Serial.begin();
}

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

Reading the Buttons

There are four buttons on the front of the MagTag - they're connected to digital pins IO **11**, **12**, **14**, and **15**. (Note we skip 13 since thats the built in Red LED). However, we recommend you use the constants `BUTTON_A`, `BUTTON_B`, `BUTTON_C`, `BUTTON_D`.

```
// SPDX-FileCopyrightText: 2020 Limor Fried for Adafruit Industries
//
// SPDX-License-Identifier: MIT

void setup() {
  Serial.begin(115200);

  pinMode(BUTTON_A, INPUT_PULLUP);
  pinMode(BUTTON_B, INPUT_PULLUP);
  pinMode(BUTTON_C, INPUT_PULLUP);
  pinMode(BUTTON_D, INPUT_PULLUP);
}

void loop() {
  if (! digitalRead(BUTTON_A)) {
    Serial.println("Button A pressed");
  }
  if (! digitalRead(BUTTON_B)) {
    Serial.println("Button B pressed");
  }
  if (! digitalRead(BUTTON_C)) {
    Serial.println("Button C pressed");
  }
  if (! digitalRead(BUTTON_D)) {
    Serial.println("Button D pressed");
  }

  // small debugging delay
  delay(10);
}
```

Open the serial console and press buttons to see the serial output printed!



Using On-board Speaker

A small buzzer on the back of the MagTag can be used to create tones!

Note that, by default, the speaker amplifer is disabled so you have to enable it like so:

```
// set speaker enable pin to output
  pinMode(SPEAKER_SHUTDOWN, OUTPUT);
  // and immediately disable it to save power
  digitalWrite(SPEAKER_SHUTDOWN, LOW);
```

You can then turn it back on with

`digitalWrite(SPEAKER_SHUTDOWN, HIGH);`

when you're ready to play tones or short audio clips

Using On-Board NeoPixels

There are 4 NeoPixels on pin IO #1 (we recommend using the macro `PIN_NEOPIXEL`), they also have a power enable pin on `NEOPIXEL_POWER` you will need to set it to be an OUTPUT and LOW before writing data to the NeoPixels. If you turn the power pin off (by setting it HIGH) you will need to re-send your NeoPixel data after re-enabling power because the pixels will forget their setting when they lose power.

Start by installing NeoPixel library support for Arduino (https://adafru.it/nBF).

Here's an example sketch that builds on the button press example to turn the LEDs different colors when you press the buttons.

Note we fill the NeoPixels after we turn on the power and then turn them off when no buttons are pressed.

```
// SPDX-FileCopyrightText: 2020 Limor Fried for Adafruit Industries
//
// SPDX-License-Identifier: MIT

#include <Adafruit_NeoPixel.h>

Adafruit_NeoPixel pixels = Adafruit_NeoPixel(4, PIN_NEOPIXEL, NEO_GRB + NEO_KHZ800);

void setup() {
  //Initialize serial
  Serial.begin(115200);

  pinMode(BUTTON_A, INPUT_PULLUP);
  pinMode(BUTTON_B, INPUT_PULLUP);
  pinMode(BUTTON_C, INPUT_PULLUP);
  pinMode(BUTTON_D, INPUT_PULLUP);

  // Neopixel power
  pinMode(NEOPIXEL_POWER, OUTPUT);
  digitalWrite(NEOPIXEL_POWER, LOW); // on

  pixels.begin();
  pixels.setBrightness(50);
  pixels.fill(0xFF00FF);
  pixels.show(); // Initialize all pixels to 'off'
}

void loop() {
  if (! digitalRead(BUTTON_A)) {
    Serial.println("Button A pressed");
    digitalWrite(NEOPIXEL_POWER, LOW); // on
    pixels.fill(0xFF0000);
    pixels.show();
  }
  else if (! digitalRead(BUTTON_B)) {
    Serial.println("Button B pressed");
    digitalWrite(NEOPIXEL_POWER, LOW); // on
```

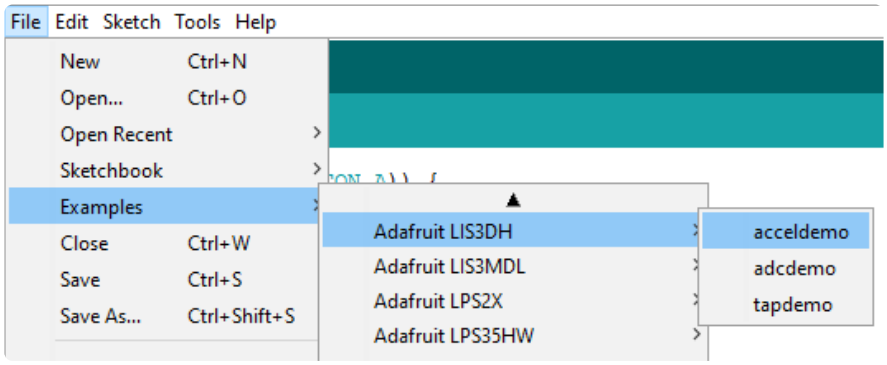
```
    pixels.fill(0x00FF00);
    pixels.show();
  }
  else if (! digitalRead(BUTTON_C)) {
    Serial.println("Button C pressed");
    digitalWrite(NEOPIXEL_POWER, LOW); // on
    pixels.fill(0x0000FF);
    pixels.show();
  }
  else if (! digitalRead(BUTTON_D)) {
    Serial.println("Button D pressed");
    digitalWrite(NEOPIXEL_POWER, LOW); // on
    pixels.fill(0xFF00FF);
    pixels.show();
  }
  else {
    // No buttons pressed! turn em off
    digitalWrite(NEOPIXEL_POWER, HIGH);
  }
}
```

Using On-board Accelerometer

There's a pre-soldered accelerometer that you can use to detect orientation or motion.

Start by installing the Arduino LIS3DH library (https://adafru.it/OkE)

You can test the LIS3DH by loading the included **acceldemo** in the Arduino library

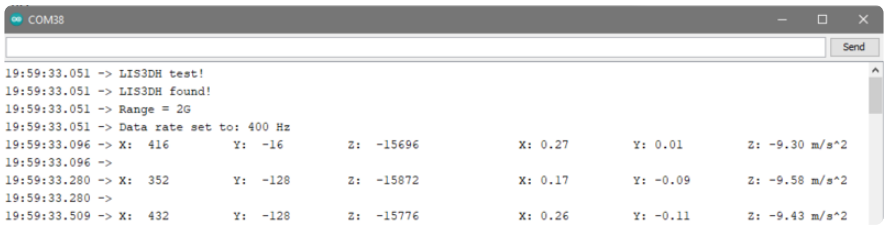


Before you upload, go down to find this section:

```
if (! lis.begin(0x18)) {   // change this to 0x19 for alternative i2c address
    Serial.println("Couldnt start");
    while (1) yield();
  }
```

And change `lis.begin(0x18)` to `lis.begin(0x19)`

Now you can upload, reset, and check the serial port for acceleration data!

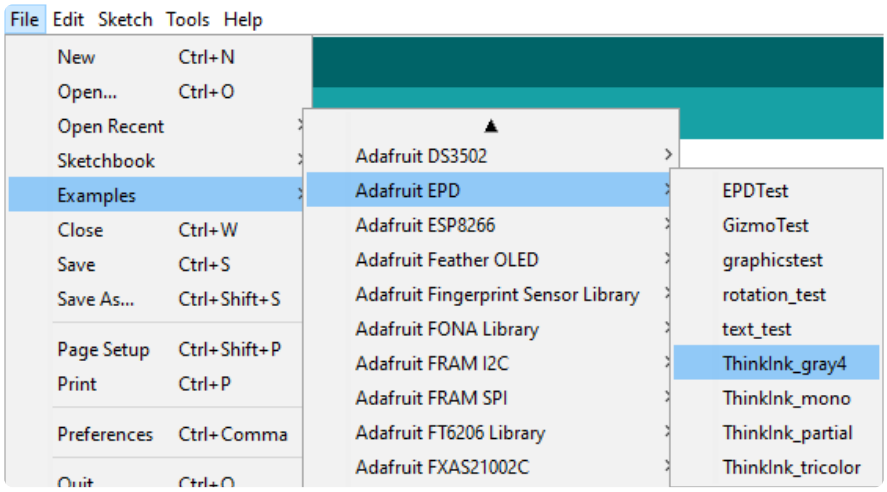


Using the E-Ink Display

You've been so patient, it's time to draw to the display!

Install the Adafruit EPD / ThinkInk library and helper libraries as shown here (https://adafru.it/OCn), and then return once installed.

Open the **ThinkInk_gray4** example



Change the pin definitions near the top to:

```
#define EPD_DC      7
#define EPD_CS      8
#define EPD_BUSY    -1
#define SRAM_CS     -1
#define EPD_RESET   6
```

Make sure this line near the top is uncommented, for the `ThinkInk_290_Grayscale_T5` type of display (there are a lot of different displays!)

```
// 2.9" Grayscale Featherwing, MagTag or Breakout:
ThinkInk_290_Grayscale4_T5 display(EPD_DC, EPD_RESET, EPD_CS, SRAM_CS, EPD_BUSY);
```

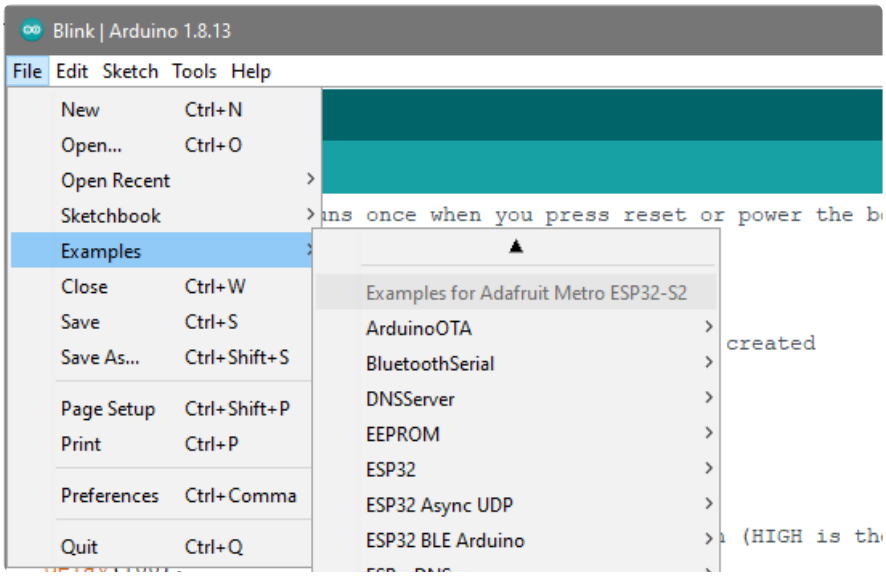
You can now upload the example to your MagTag to see it display various graphics and text in monochrome and grayscale.

For more information on E-Ink displays, check out our detailed guide (https://adafru.it/GOc)

For more information on how to display graphics, and text, check out the Adafruit GFX guide (https://adafru.it/doL)

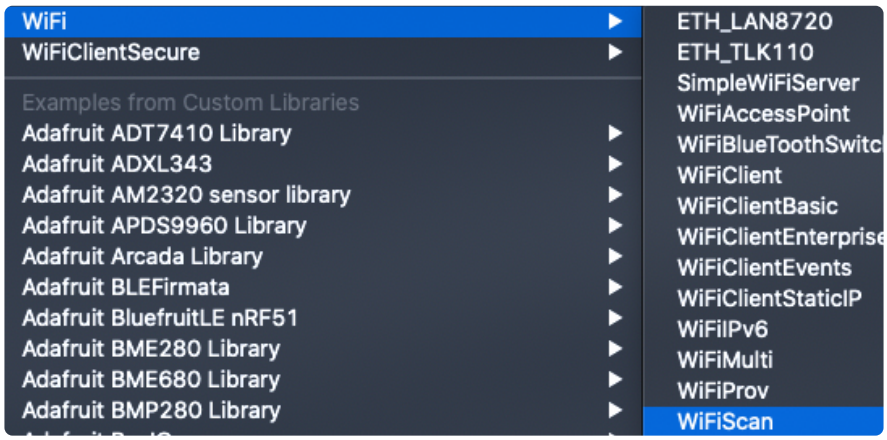
WiFi Test

Thanksfully if you have ESP32 sketches, they'll 'just work' with variations of ESP32. You can find a wide range of examples in the **File->Examples->Examples for Adafruit Metro ESP32-S2** subheading (the name of the board may vary so it could be **"Examples for Adafruit Feather ESP32 V2"** etc)



Let's start by scanning the local networks.

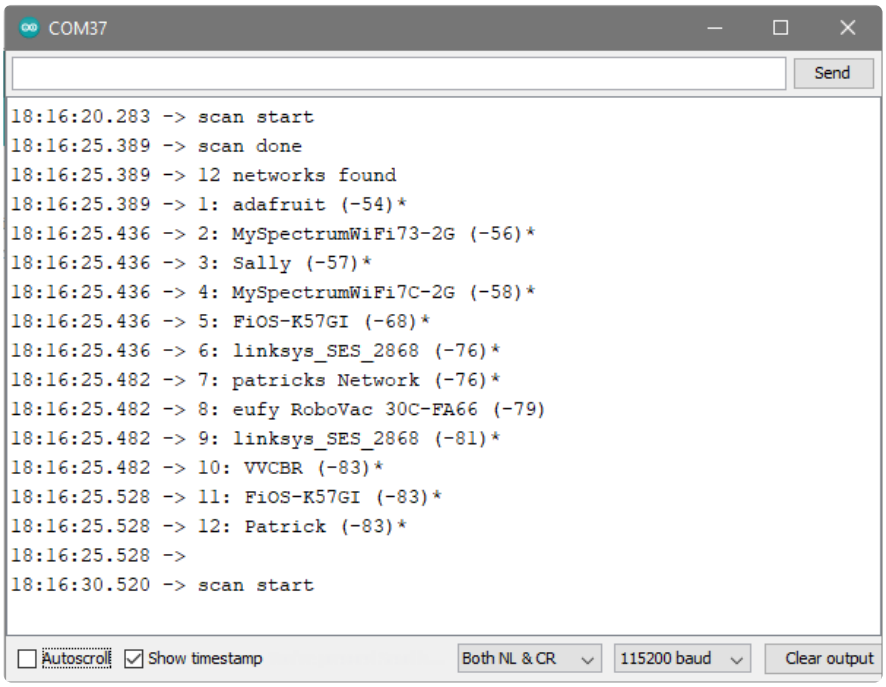
Load up the WiFiScan example under **Examples->Examples for YOUR BOARDNAME->WiFi->WiFiScan**



And **upload this example to your board**. The ESP32 should scan and find WiFi networks around you.

For ESP32, open the serial monitor, to see the scan begin.

For ESP32-S2, -S3 and -C3, don't forget you have to click Reset after uploading through the ROM bootloader. Then select the new USB Serial port created by the ESP32. It will take a few seconds for the board to complete the scan.



If you can not scan any networks, check your power supply. You need a solid power supply in order for the ESP32 to not brown out. A skinny USB cable or drained battery can cause issues.

WiFi Connection Test

Now that you can scan networks around you, its time to connect to the Internet!

Copy the example below and paste it into the Arduino IDE:

```
// SPDX-FileCopyrightText: 2020 Brent Rubell for Adafruit Industries
//
// SPDX-License-Identifier: MIT


/*
  Web client

 This sketch connects to a website (wifitest.adafruit.com/testwifi/index.html)
 using the WiFi module.

 This example is written for a network using WPA encryption. For
 WEP or WPA, change the Wifi.begin() call accordingly.

 This example is written for a network using WPA encryption. For
 WEP or WPA, change the Wifi.begin() call accordingly.

 created 13 July 2010
 by dlf (Metodo2 srl)
 modified 31 May 2012
 by Tom Igoe
 */
```

```
#include <WiFi.h>

// Enter your WiFi SSID and password
char ssid[] = "YOUR_SSID";             // your network SSID (name)
char pass[] = "YOUR_SSID_PASSWORD";    // your network password (use for WPA, or
use as key for WEP)
int keyIndex = 0;                      // your network key Index number (needed
only for WEP)

int status = WL_IDLE_STATUS;
// if you don't want to use DNS (and reduce your sketch size)
// use the numeric IP instead of the name for the server:
//IPAddress server(74,125,232,128);  // numeric IP for Google (no DNS)

char server[] = "wifitest.adafruit.com";    // name address for adafruit test
char path[]   = "/testwifi/index.html";

// Initialize the Ethernet client library
// with the IP address and port of the server
// that you want to connect to (port 80 is default for HTTP):
WiFiClient client;

void setup() {
  //Initialize serial and wait for port to open:
  Serial.begin(115200);
  while (!Serial) {
    ; // wait for serial port to connect. Needed for native USB port only
  }

  // attempt to connect to Wifi network:
  Serial.print("Attempting to connect to SSID: ");
  Serial.println(ssid);

  WiFi.begin(ssid, pass);
  while (WiFi.status() != WL_CONNECTED) {
      delay(500);
      Serial.print(".");
  }

  Serial.println("");
  Serial.println("Connected to WiFi");
  printWifiStatus();

  Serial.println("\nStarting connection to server...");
  // if you get a connection, report back via serial:
  if (client.connect(server, 80)) {
    Serial.println("connected to server");
    // Make a HTTP request:
    client.print("GET "); client.print(path); client.println(" HTTP/1.1");
    client.print("Host: "); client.println(server);
    client.println("Connection: close");
    client.println();
  }
}

void loop() {
  // if there are incoming bytes available
  // from the server, read them and print them:
  while (client.available()) {
    char c = client.read();
    Serial.write(c);
  }

  // if the server's disconnected, stop the client:
  if (!client.connected()) {
    Serial.println();
    Serial.println("disconnecting from server.");
    client.stop();
```

```
    // do nothing forevermore:
    while (true) {
      delay(100);
    }
  }
}


void printWifiStatus() {
  // print the SSID of the network you're attached to:
  Serial.print("SSID: ");
  Serial.println(WiFi.SSID());

  // print your board's IP address:
  IPAddress ip = WiFi.localIP();
  Serial.print("IP Address: ");
  Serial.println(ip);

  // print the received signal strength:
  long rssi = WiFi.RSSI();
  Serial.print("signal strength (RSSI):");
  Serial.print(rssi);
  Serial.println(" dBm");
}
```

NOTE: You must **change the `SECRET_SSID` and `SECRET_PASS` in the example code to your WiFi SSID and password** before uploading this to your board.

```
// Enter your WiFi SSID and password
char ssid[] = "YOUR_SSID";            // your network SSID (name)
char pass[] = "YOUR_SSID_PASSWORD";   // your network password (use for WPA, or use as key for WEP)
int keyIndex = 0;                     // your network key Index number (needed only for WEP)
```

After you've set it correctly, upload and check the serial monitor. You should see the following. If not, go back, check wiring, power and your SSID/password

```
Attempting to connect to SSID: Transit
........
Connected to WiFi
SSID: Transit
IP Address: 192.168.1.182
signal strength (RSSI):-57 dBm

Starting connection to server...
connected to server
HTTP/1.1 200 OK
Server: nginx/1.10.3 (Ubuntu)
Date: Wed, 11 Nov 2020 20:51:30 GMT
Content-Type: text/html
Content-Length: 70
Last-Modified: Thu, 16 May 2019 18:21:16 GMT
Connection: close
ETag: "5cddaa1c-46"
Accept-Ranges: bytes

This is a test of Adafruit WiFi!
If you can read this, its working :)

disconnecting from server.
```

Secure Connection Example

Many servers today do not allow non-SSL connectivity. Lucky for you the ESP32 has a great TLS/SSL stack so you can have that all taken care of for you. Here's an example of a using a secure WiFi connection to connect to the Twitter API.

Copy and paste it into the Arduino IDE:

```
// SPDX-FileCopyrightText: 2015 Arturo Guadalupi
// SPDX-FileCopyrightText: 2020 Brent Rubell for Adafruit Industries
//
// SPDX-License-Identifier: MIT

/*
This example creates a client object that connects and transfers
data using always SSL.

It is compatible with the methods normally related to plain
connections, like client.connect(host, port).

Written by Arturo Guadalupi
last revision November 2015

*/

#include <WiFiClientSecure.h>
#include <WiFi.h>

// Enter your WiFi SSID and password
char ssid[] = "YOUR_SSID";             // your network SSID (name)
```

```
char pass[] = "YOUR_SSID_PASSWORD";    // your network password (use for WPA, or
use as key for WEP)
int keyIndex = 0;                      // your network key Index number (needed
only for WEP)

int status = WL_IDLE_STATUS;
// if you don't want to use DNS (and reduce your sketch size)
// use the numeric IP instead of the name for the server:
//IPAddress server(74,125,232,128);  // numeric IP for Google (no DNS)

#define SERVER "cdn.syndication.twimg.com"
#define PATH   "/widgets/followbutton/info.json?screen_names=adafruit"

// Initialize the SSL client library
// with the IP address and port of the server
// that you want to connect to (port 443 is default for HTTPS):
WiFiClientSecure client;

void setup() {
  //Initialize serial and wait for port to open:
  Serial.begin(115200);
  while (!Serial) {
    ; // wait for serial port to connect. Needed for native USB port only
  }

  // attempt to connect to Wifi network:
  Serial.print("Attempting to connect to SSID: ");
  Serial.println(ssid);

  WiFi.begin(ssid, pass);
  while (WiFi.status() != WL_CONNECTED) {
      delay(500);
      Serial.print(".");
  }

  Serial.println("");
  Serial.println("Connected to WiFi");
  printWifiStatus();

  client.setInsecure(); // don't use a root cert

  Serial.println("\nStarting connection to server...");
  // if you get a connection, report back via serial:
  if (client.connect(SERVER, 443)) {
    Serial.println("connected to server");
    // Make a HTTP request:
    client.println("GET " PATH " HTTP/1.1");
    client.println("Host: " SERVER);
    client.println("Connection: close");
    client.println();
  }
}

uint32_t bytes = 0;

void loop() {
  // if there are incoming bytes available
  // from the server, read them and print them:
  while (client.available()) {
    char c = client.read();
    Serial.write(c);
    bytes++;
  }

  // if the server's disconnected, stop the client:
  if (!client.connected()) {
    Serial.println();
    Serial.println("disconnecting from server.");
    client.stop();
```

```
    Serial.print("Read "); Serial.print(bytes); Serial.println(" bytes");

    // do nothing forevermore:
    while (true);
  }
}


void printWifiStatus() {
  // print the SSID of the network you're attached to:
  Serial.print("SSID: ");
  Serial.println(WiFi.SSID());

  // print your board's IP address:
  IPAddress ip = WiFi.localIP();
  Serial.print("IP Address: ");
  Serial.println(ip);

  // print the received signal strength:
  long rssi = WiFi.RSSI();
  Serial.print("signal strength (RSSI):");
  Serial.print(rssi);
  Serial.println(" dBm");
}
```

As before, **update the ssid and password first**, then upload the example to your board.

Note we use `WiFiClientSecure client` instead of `WiFiClient client;` to require a SSL connection!

```
Attempting to connect to SSID: Transit
......
Connected to WiFi
SSID: Transit
IP Address: 192.168.1.182
signal strength (RSSI):-52 dBm

Starting connection to server...
connected to server
HTTP/1.1 200 OK
Accept-Ranges: bytes
Access-Contol-Allow-Origin: platform.twitter.com
Access-Control-Allow-Methods: GET
Age: 12
cache-control: must-revalidate, max-age=600
content-disposition: attachment; filename=json.json
Content-Type: application/json;charset=utf-8
Date: Wed, 11 Nov 2020 20:58:39 GMT
expires: Wed, 11 Nov 2020 21:08:39 GMT
Last-Modified: Wed, 11 Nov 2020 20:58:27 GMT
Server: ECS (agb/52BA)
strict-transport-security: max-age=631138519
timing-allow-origin: *
X-Cache: HIT
x-connection-hash: a50988a9020759ec70520caef6c38bcf
x-content-type-options: nosniff
x-frame-options: SAMEORIGIN
x-response-time: 12
x-transaction: 003d88570028acec
x-tw-cdn: VZ
x-tw-cdn: VZ
x-xss-protection: 0
Content-Length: 197
Connection: close

[{"following":false,"id":"20731304","screen_name":"adafruit","name":"adafruit industries","
disconnecting from server.
Read 966 bytes
```

Arduino Sleep

Once you have hardware tested and working, and you've also got WiFi connected - its time to explore deep sleep modes. When in deep sleep, the ESP32-S2 is basically completely off, it uses a tiny amount of power just to keep track of time so you can wake up after some amount of time.

When the ESP32-S2 comes back from deep-sleep it essentially does a hard-reset and begins over from scratch, so you don't get to keep your WiFi connection active. However, its pretty fast to re-connect so as long as you are only waking up about once a minute, your battery will last a lot longer!

Good Quality Sleep

Before you go into deep sleep, make sure you shut down the E-Ink display, NeoPixels, light sensor, and speaker with the following instructions:

```
display.powerDown();
digitalWrite(EPD_RESET, LOW); // hardware power down mode
digitalWrite(SPEAKER_SHUTDOWN, LOW); // off
digitalWrite(NEOPIXEL_POWER, HIGH); // off
```

Then enter deep sleep mode with

```
esp_sleep_enable_timer_wakeup(1000000);
esp_deep_sleep_start();
```

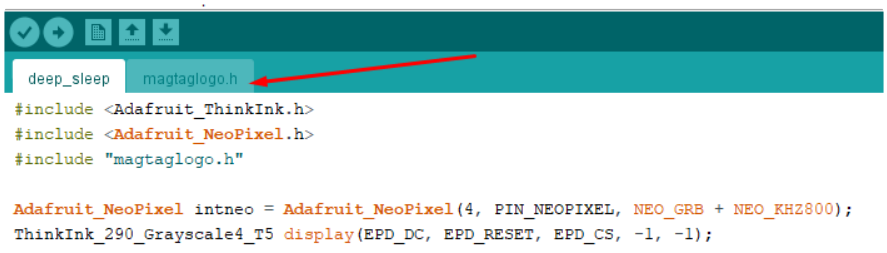
the `esp_sleep_enable_timer_wakeup(10000000);` line tells the ESP32-S2 to restart in 10000000 microseconds, a.k.a 10000 milliseconds or 10 seconds

Here's an example sketch that wakes up, turns on the speaker and NeoPixels, draws a bitmap onto the EPD and then goes to sleep for one second. We use this demo to do power monitoring tests.

```
// SPDX-FileCopyrightText: 2020 Limor Fried for Adafruit Industries
//
// SPDX-License-Identifier: MIT

#include <Adafruit_ThinkInk.h>
#include <Adafruit_NeoPixel.h>
#include "magtaglogo.h"

Adafruit_NeoPixel intneo = Adafruit_NeoPixel(4, PIN_NEOPIXEL, NEO_GRB + NEO_KHZ800);
ThinkInk_290_Grayscale4_T5 display(EPD_DC, EPD_RESET, EPD_CS, -1, -1);

void setup() {
  //Initialize serial and wait for port to open:
  Serial.begin(115200);

  pinMode(BUTTON_A, INPUT_PULLUP);
  pinMode(BUTTON_B, INPUT_PULLUP);
```

```
  pinMode(BUTTON_C, INPUT_PULLUP);
  pinMode(BUTTON_D, INPUT_PULLUP);
  pinMode(13, OUTPUT);
  digitalWrite(13, LOW);

  // Neopixel power
  pinMode(NEOPIXEL_POWER, OUTPUT);
  pinMode(SPEAKER_SHUTDOWN, OUTPUT);
  digitalWrite(NEOPIXEL_POWER, LOW); // on
  digitalWrite(SPEAKER_SHUTDOWN, HIGH); // on

  intneo.begin();
  intneo.setBrightness(50);
  intneo.fill(100, 0, 100);
  intneo.show();

  display.begin(THINKINK_MONO);
  display.clearBuffer();
  display.drawBitmap(0, 38, magtaglogo_mono, MAGTAGLOGO_WIDTH, MAGTAGLOGO_HEIGHT,
EPD_BLACK);
  display.display();
  delay(1500);
  display.powerDown();
  digitalWrite(EPD_RESET, LOW); // hardware power down mode

  digitalWrite(SPEAKER_SHUTDOWN, LOW); // off
  digitalWrite(NEOPIXEL_POWER, HIGH); // off
  esp_sleep_enable_timer_wakeup(1000000);
  esp_deep_sleep_start();
}

void loop() {
}
```

Be sure to download the sketch including the logo header file by clicking **Download Project Zip** and verify that in the Arduino IDE you have both tabs:



This example will go to deep sleep for only one second, you can change that by adjusting the number in this line:

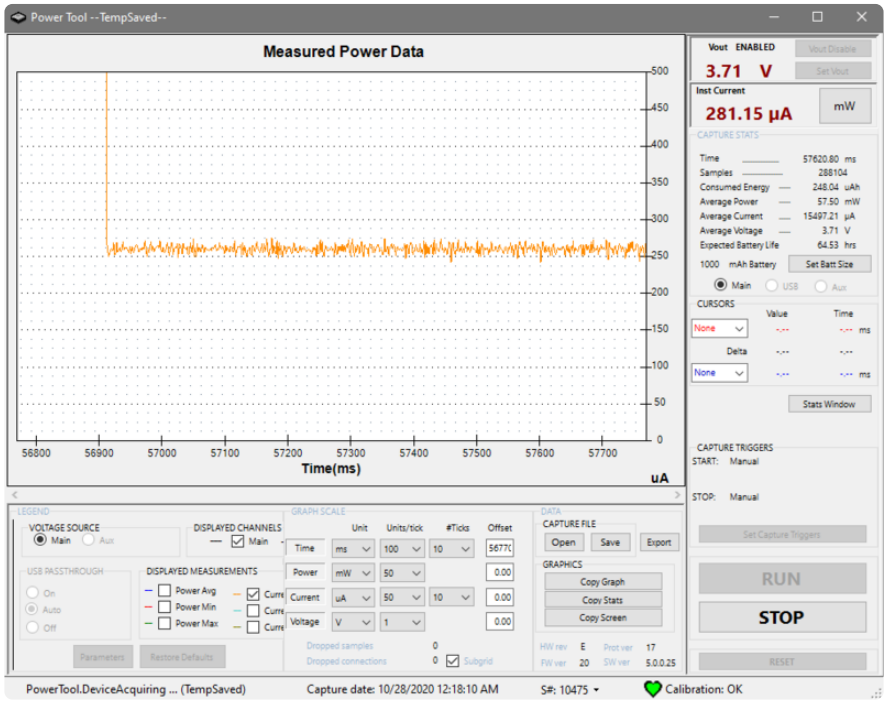
`esp_sleep_enable_timer_wakeup(1000000);`

When using deep sleep, the power draw drops to about 250uA. These are the approximate current uses:

- 40uA are used by the onboard green OK LED (you can remove the LED with a soldering iron)
- 120uA are used by the EPD in deepsleep/reset mode (we couldnt figure out how to reduce this any more but tested suggestions are welcome!)

- 50uA are used by the 3.3V regulator
- 10uA are used by the VBat resistor divider
- 30uA are used by the ESP32-S2 in deep sleep

USB power will be higher than this number because under USB power, we also power the battery charger. These current draws are only for battery usage and assume you have disabled NeoPixels, speaker, and EPD.



Shipping Demo

The board ships with a demo that tests out the buttons, LEDs, EPD, accelerometer, and speaker. It may be handy if you ever need to verify your hardware is fully working!

- The NeoPixels will light up in a rainbow design
- When button A, B, C are pressed, the LEDs will turn to a solid color red, green or blue
- When button D is pressed, a chime sounds
- Red LED blinks on and off
- Display shows bitmap MagTag logo. When the accelerometer detects that the board is 'upside down' the display is rotated
- In serial console, light sensor reading and accelerometer XYZ is output

- In serial console, I2C port on STEMMA QT is scanned and visible I2C device addresses are printed

You can put your board into UF2 Bootloader mode and drag-n-drop this file on to load it:

mag_tag_shipping_demo.UF2

https://adafru.it/PfK

You can download the full example code for compilation by clicking **Download Project Zip** on this sketch to get all the files (there are two header files, `coin.h` with 8-bit audio and `magtaglogo.h` with the bitmap logo)

Before uploading to your magtag make sure you downloaded the full project and got the two header files!

> If you are looking for the ThinkInk library to use with this example, click the link below to get a zip including the ThinkInk library.

To get the full set of libraries that go with the Adafruit EPD library, click this link and download the latest zip file (https://adafru.it/-5b).


```
// SPDX-FileCopyrightText: 2020 Limor Fried for Adafruit Industries
//
// SPDX-License-Identifier: MIT

#include <Adafruit_NeoPixel.h>
#include <Wire.h>
#include <Adafruit_ThinkInk.h>
#include <Adafruit_LIS3DH.h>
#include "coin.h"
#include "magtaglogo.h"

Adafruit_NeoPixel intneo = Adafruit_NeoPixel(4, PIN_NEOPIXEL, NEO_GRB + NEO_KHZ800);
ThinkInk_290_Grayscale4_T5 display(EPD_DC, EPD_RESET, EPD_CS, -1, -1);
Adafruit_LIS3DH lis = Adafruit_LIS3DH();

uint8_t j = 0;
void setup() {
  Serial.begin(115200);
```

```
  //while (!Serial) { delay(10); }
  delay(100);
  Serial.println("Adafruit EPD Portal demo");

  intneo.begin();
  intneo.setBrightness(50);
  intneo.show(); // Initialize all pixels to 'off'

  pinMode(BUTTON_A, INPUT_PULLUP);
  pinMode(BUTTON_B, INPUT_PULLUP);
  pinMode(BUTTON_C, INPUT_PULLUP);
  pinMode(BUTTON_D, INPUT_PULLUP);

  pinMode(SPEAKER_SHUTDOWN, OUTPUT);
  digitalWrite(SPEAKER_SHUTDOWN, LOW);

  // Red LED
  pinMode(13, OUTPUT);

  // Neopixel power
  pinMode(NEOPIXEL_POWER, OUTPUT);
  digitalWrite(NEOPIXEL_POWER, LOW); // on

  display.begin(THINKINK_MONO);

  if (! lis.begin(0x19)) {
    Serial.println("Couldnt start LIS3DH");
    display.clearBuffer();
    display.setTextSize(3);
    display.setTextColor(EPD_BLACK);
    display.setCursor(20, 40);
    display.print("No LIS3DH?");
    display.display();
    while (1) delay(100);
  }

  analogReadResolution(12); //12 bits
  analogSetAttenuation(ADC_11db);  //For all pins
  display.clearBuffer();
  display.drawBitmap(0, 38, magtaglogo_mono, MAGTAGLOGO_WIDTH, MAGTAGLOGO_HEIGHT,
EPD_BLACK);
  display.display();
}



uint8_t rotation = 0;

void loop() {
  j++;
  if (j == 0) {
    Serial.print("Rotation: "); Serial.println(rotation);
    if (rotation == 0 || rotation == 2) {
      display.setRotation(rotation);
      display.clearBuffer();
      display.drawBitmap(0, 38, magtaglogo_mono, MAGTAGLOGO_WIDTH,
MAGTAGLOGO_HEIGHT, EPD_BLACK);
      display.display();
    }
  }
  // Red LED On
  digitalWrite(13, HIGH);

  //Serial.print(".");
  if (j % 10 == 0) {
      sensors_event_t event;
      lis.getEvent(&event);

      /* Display the results (acceleration is measured in m/s^2) */
```

```
      Serial.print("X: "); Serial.print(event.acceleration.x);
      Serial.print(" \tY: "); Serial.print(event.acceleration.y);
      Serial.print(" \tZ: "); Serial.print(event.acceleration.z);
      Serial.println(" m/s^2 ");
      if ((event.acceleration.x < -5) && (abs(event.acceleration.y) < 5)) {
        rotation = 1;
      }
      if ((abs(event.acceleration.x) < 5) && (event.acceleration.y > 5)) {
        rotation = 0;
      }
      if ((event.acceleration.x > 5) && (abs(event.acceleration.y) < 5)) {
        rotation = 3;
      }
      if ((abs(event.acceleration.x) < 5) && (event.acceleration.y < -5)) {
        rotation = 2;
      }

      int light = analogRead(LIGHT_SENSOR);
      Serial.print("Light sensor: ");
      Serial.println(light);

      Serial.print("I2C scanner: ");
      for (int i = 0x07; i <= 0x77; i++) {
        Wire.beginTransmission(i);
        bool found (Wire.endTransmission() == 0);
        if (found) {
          Serial.printf("0x%02x, ", i);
        }
      }
      Serial.println();
  }

  if (! digitalRead(BUTTON_A)) {
    Serial.println("Button A pressed");
    intneo.fill(0xFF0000);
    intneo.show();
  }
  else if (! digitalRead(BUTTON_B)) {
    Serial.println("Button B pressed");
    intneo.fill(0x00FF00);
    intneo.show();
  }
  else if (! digitalRead(BUTTON_C)) {
    Serial.println("Button C pressed");
    intneo.fill(0x0000FF);
    intneo.show();
  }
  else if (! digitalRead(BUTTON_D)) {
    intneo.fill(0x0);
    intneo.show();
    Serial.println("Button D pressed");
    digitalWrite(SPEAKER_SHUTDOWN, HIGH);
    play_tune(audio, sizeof(audio));
    digitalWrite(SPEAKER_SHUTDOWN, LOW);
  } else {
    // neopixelate
    for (int i = 0; i < intneo.numPixels(); i++) {
      intneo.setPixelColor(i, Wheel(((i * 256 / intneo.numPixels()) + j) & 255));
    }
    intneo.show();
  }

  // Red LED off
  digitalWrite(13, LOW);


  delay(10);
}
```

```
void play_tune(const uint8_t *audio, uint32_t audio_length) {
  uint32_t t;
  uint32_t prior, usec = 1000000L / SAMPLE_RATE;

  prior = micros();
  for (uint32_t i=0; i<audio_length; i++) {
    while((t = micros()) - prior < usec);
    dacWrite(A0, audio[i]);
    prior = t;
  }
}

// Input a value 0 to 255 to get a color value.
// The colours are a transition r - g - b - back to r.
uint32_t Wheel(byte WheelPos) {
  WheelPos = 255 - WheelPos;
  if (WheelPos < 85) {
    return intneo.Color(255 - WheelPos * 3, 0, WheelPos * 3);
  }
  if (WheelPos < 170) {
    WheelPos -= 85;
    return intneo.Color(0, WheelPos * 3, 255 - WheelPos * 3);
  }
  WheelPos -= 170;
  return intneo.Color(WheelPos * 3, 255 - WheelPos * 3, 0);
}
```

Quotes Example

OK now that we have the onboard elements, WiFi and deep sleep all worked out, we can now create a full demo!

This example connects to the online Adafruit Quotes service, and displays an inspiring quote once a minute with deep sleep in between.

Don't forget to update the SSID and password!

> If you are looking for the ThinkInk library to use with this example, click the link below to get a zip including the ThinkInk library.

To get the full set of libraries that go with the Adafruit EPD library, click this link and download the latest zip file (https://adafru.it/-5b).


```
// SPDX-FileCopyrightText: 2017 Evandro Copercini
//
// SPDX-License-Identifier: Apache-2.0

/*
  Wifi secure connection example for ESP32
  Running on TLS 1.2 using mbedTLS
  2017 - Evandro Copercini - Apache 2.0 License.
*/

#include <WiFiClientSecure.h>
#include <WiFi.h>
#include <ArduinoJson.h>
#include "Adafruit_ThinkInk.h"
#include "Adafruit_NeoPixel.h"
#include <Fonts/FreeSans9pt7b.h>

const char* ssid     = "adafruit";     // your network SSID (name of wifi network)
const char* password = "ffffffff"; // your network password

const char*  server = "www.adafruit.com";
const char*  path   = "/api/quotes.php";

WiFiClientSecure client;
ThinkInk_290_Grayscale4_T5 display(EPD_DC, EPD_RESET, EPD_CS, -1, -1);
Adafruit_NeoPixel intneo = Adafruit_NeoPixel(4, PIN_NEOPIXEL, NEO_GRB + NEO_KHZ800);


void deepSleep() {
  pinMode(NEOPIXEL_POWER, OUTPUT);
  pinMode(SPEAKER_SHUTDOWN, OUTPUT);
  digitalWrite(SPEAKER_SHUTDOWN, LOW); // off
  digitalWrite(NEOPIXEL_POWER, HIGH); // off
  digitalWrite(EPD_RESET, LOW); // off (yes required to save a few mA)
  pinMode(13, OUTPUT);
  digitalWrite(13, LOW);

  esp_sleep_enable_timer_wakeup(60 * 1000000); // 60 seconds
  esp_deep_sleep_start();
}

void setup() {
  //Initialize serial and wait for port to open:
```

```
  Serial.begin(115200);
  //while (!Serial) delay(10);

  pinMode(BUTTON_A, INPUT_PULLUP);
  pinMode(BUTTON_B, INPUT_PULLUP);
  pinMode(BUTTON_C, INPUT_PULLUP);
  pinMode(BUTTON_D, INPUT_PULLUP);
  pinMode(EPD_BUSY, INPUT);
  pinMode(13, OUTPUT);
  digitalWrite(13, HIGH);


  display.begin(THINKINK_GRAYSCALE4);

  Serial.print("Attempting to connect to SSID: ");
  Serial.println(ssid);

  display.clearBuffer();
  display.setFont(&FreeSans9pt7b);
  display.setTextSize(1);
  display.setTextColor(EPD_BLACK);
  display.setCursor(10, 30);
  display.print("Connecting to SSID ");
  display.println(ssid);
  display.display();

  WiFi.begin(ssid, password);

  // attempt to connect to Wifi network:
  while (WiFi.status() != WL_CONNECTED) {
    Serial.print(".");
    delay(100);
  }

  Serial.print("Connected to ");
  Serial.println(ssid);

  //client.setCACert(test_root_ca);
  //client.setCertificate(test_client_key); // for client verification
  //client.setPrivateKey(test_client_cert);   // for client verification

// Neopixel power
  pinMode(NEOPIXEL_POWER, OUTPUT);
  digitalWrite(NEOPIXEL_POWER, LOW); // on
  intneo.fill(25, 0, 0);
  intneo.show();

  Serial.println("\nStarting connection to server...");
  client.setInsecure();
  if (!client.connect(server, 443)) {
    Serial.println("Connection failed!");
    deepSleep();
  }

  intneo.fill(25, 25, 0);
  intneo.show();

  Serial.println("Connected to server!");
  // Make a HTTP request:
  client.print("GET "); client.print(path); client.println(" HTTP/1.1");
  client.print("Host: "); client.println(server);
  client.println("Connection: close");
  client.println();

  // Check HTTP status
  char status[32] = {0};
  client.readBytesUntil('\r', status, sizeof(status));
  if (strcmp(status, "HTTP/1.1 200 OK") != 0) {
    Serial.print(F("Unexpected response: "));
```

```
  Serial.println(status);
  deepSleep();
}

intneo.fill(0, 25, 0);
intneo.show();
while (client.connected()) {
  String line = client.readStringUntil('\n');
  if (line == "\r") {
    Serial.println("headers received");
    break;
  }
}

intneo.fill(0, 25, 25);
intneo.show();
while (client.peek() != '[') {
  client.read();
}

intneo.fill(0, 0, 25);
intneo.show();

// Allocate the JSON document
// Use arduinojson.org/v6/assistant to compute the capacity.
const size_t capacity = JSON_ARRAY_SIZE(1) + JSON_OBJECT_SIZE(8) + 200;
DynamicJsonDocument doc(capacity);

// Parse JSON object
DeserializationError error = deserializeJson(doc, client);
if (error) {
  Serial.print(F("deserializeJson() failed: "));
  Serial.println(error.c_str());
  deepSleep();
}

intneo.fill(0, 25, 25);
intneo.show();

// Extract values
JsonObject root_0 = doc[0];
Serial.println(F("Response:"));
const char* root_0_text = root_0["text"];
const char* root_0_author = root_0["author"];

Serial.print("Quote: "); Serial.println(root_0_text);
Serial.print("Author: "); Serial.println(root_0_author);

display.clearBuffer();
display.setFont(&FreeSans9pt7b);
display.setTextSize(1);
display.setTextWrap(true);
display.setTextColor(EPD_BLACK);
display.setCursor(10, 30);
display.println(root_0_text);
display.setTextColor(EPD_DARK);
display.setCursor(40, 120);
display.println(root_0_author);
display.display();
while (!digitalRead(EPD_BUSY)) {
  delay(10);
}

while (client.available() > 0)
{
  //read back one line from the server
  String line = client.readStringUntil('\r');
  Serial.println(line);
}
```

```
  // disconnect
  client.stop();
  deepSleep();
}

void loop() {

}
```

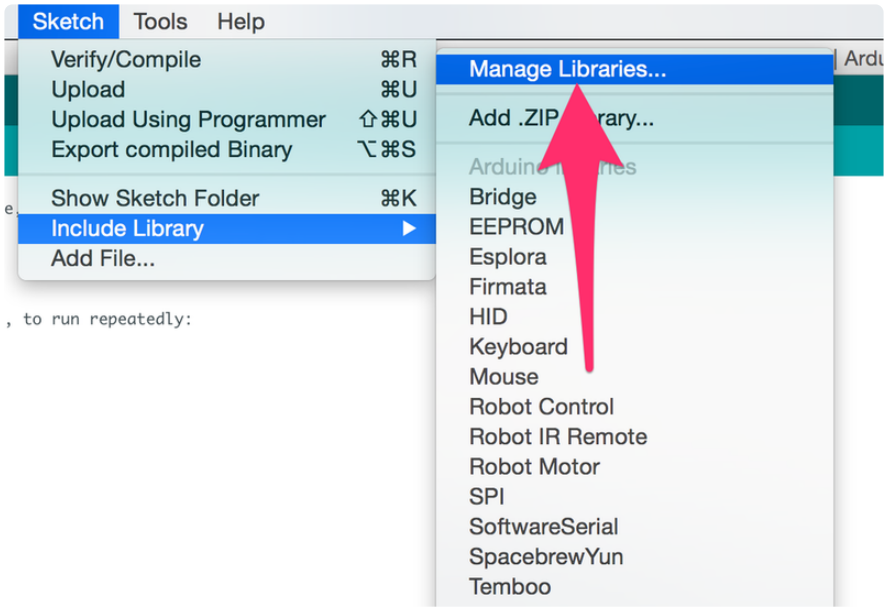
Usage with Adafruit IO

The ESP32-S2/S3 is an affordable, all-in-one, option for connecting your projects to the internet using our IoT platform, Adafruit IO (https://adafru.it/Eg2).

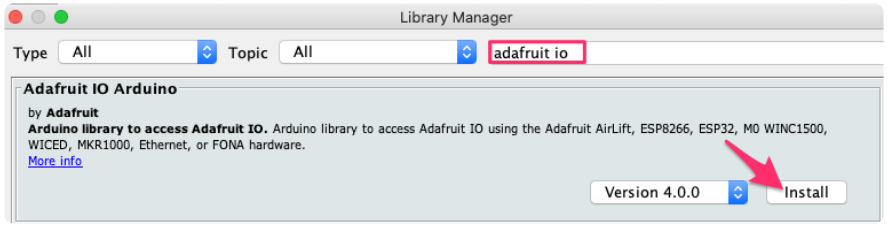
- For more information and guides about Adafruit IO, check out the Adafruit IO Basics Series. (https://adafru.it/iDX)

Install Libraries

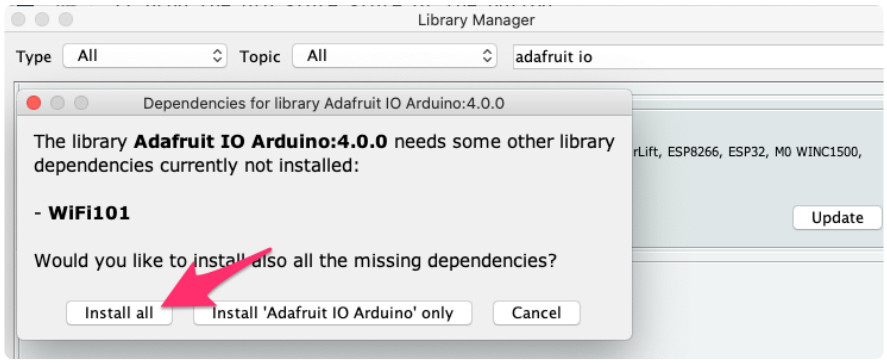
In the Arduino IDE, navigate to **Sketch -> Include Library->Manage Libraries...**



Enter **Adafruit IO Arduino** into the search box, and click **Install** on the **Adafruit IO Arduino** library option to install version 4.0.0 or higher.



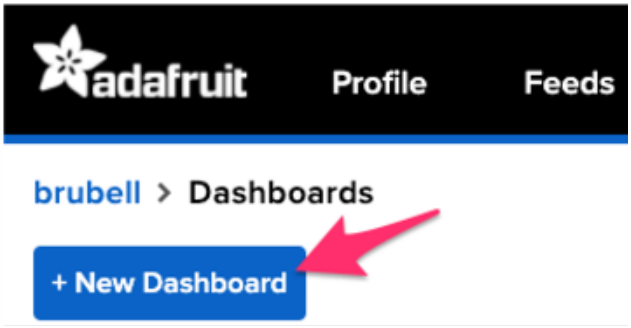
When asked to install dependencies, **click Install all.**

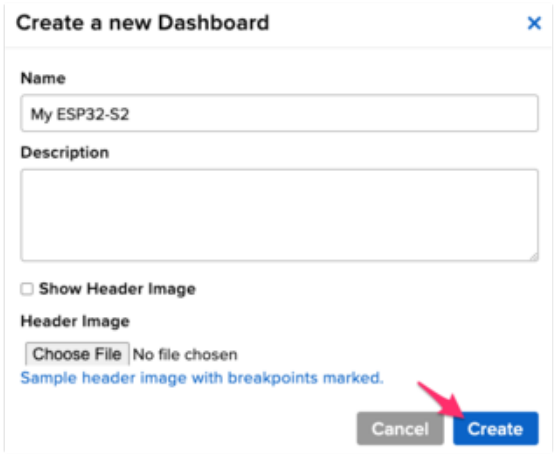


Adafruit IO Setup

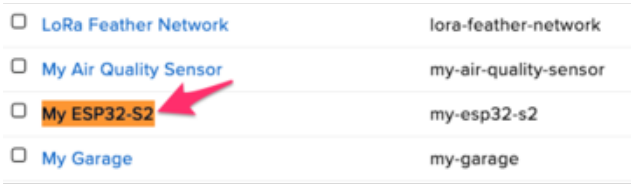
If you do not already have an Adafruit IO account, create one now (https://adafru.it/fH9). Next, **navigate to the Adafruit IO Dashboards page**.

We'll create a dashboard to visualize and interact with the data being sent between your ESP32-S2/S3 board and Adafruit IO.

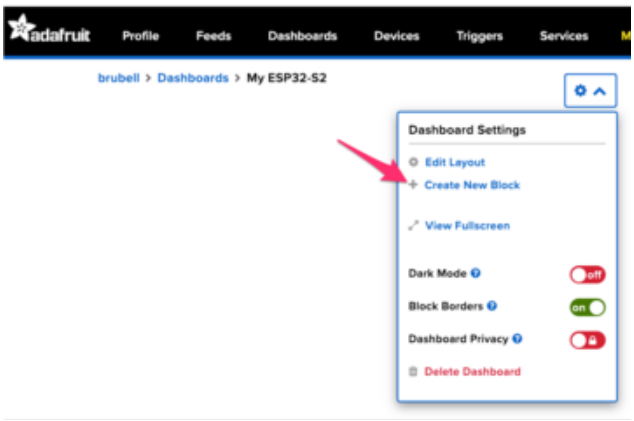


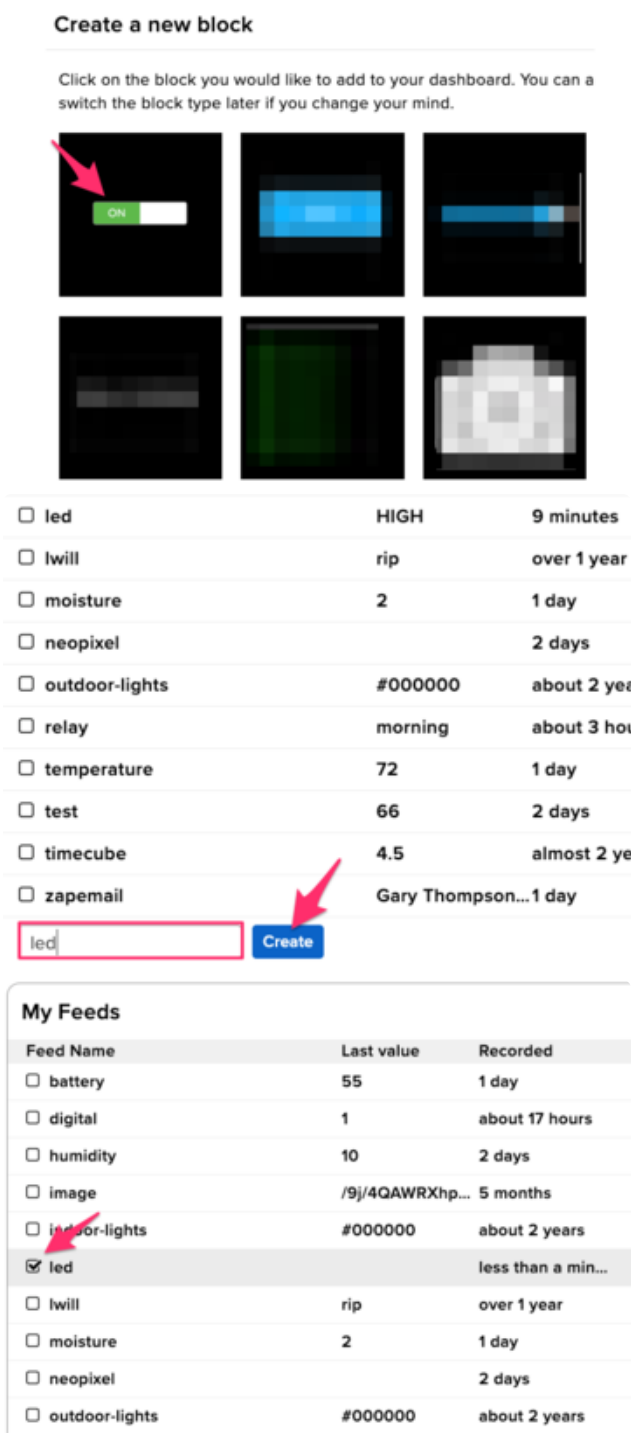


Click the New Dashboard button.
Name your dashboard My ESP32-S2 or My ESP32-S3 depending on your board.
Your new dashboard should appear in the list.
Click the link to be brought to your new dashboard.



We'll want to turn the board's LED on or off from Adafruit IO. To do this, **we'll need to add a toggle button to our dashboard**.

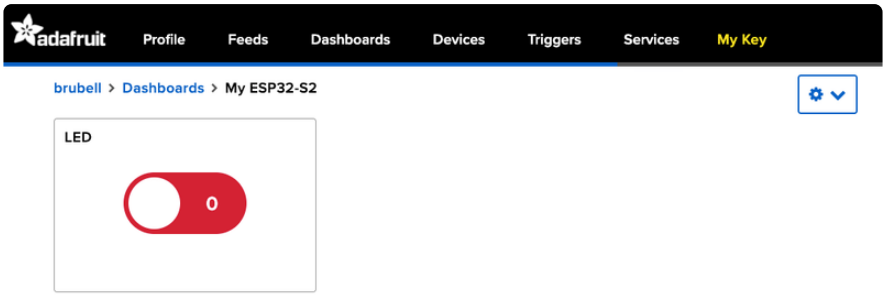




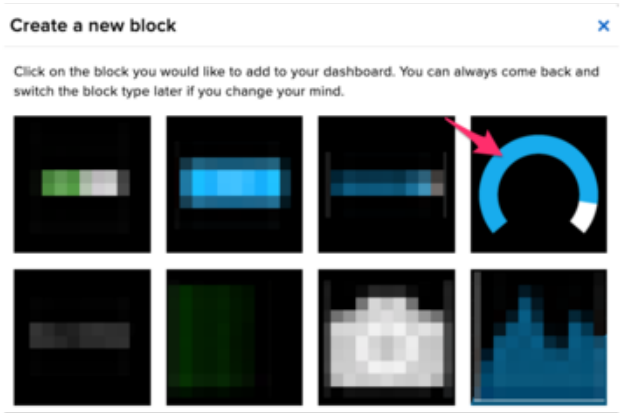
Click the cog at the top right hand corner of your dashboard.
In the dashboard settings dropdown, **click Create New Block.**
Select the toggle block.
Under My Feeds, **enter led as a feed name**. **Click Create**.
Choose the led feed to connect it to the toggle block. **Click Next step**.

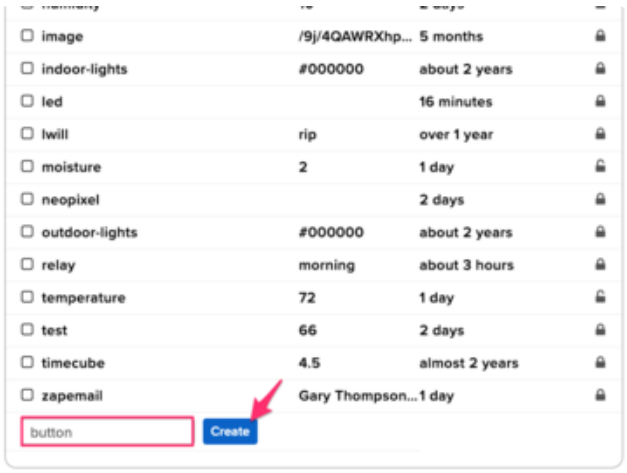
Under Block Settings,

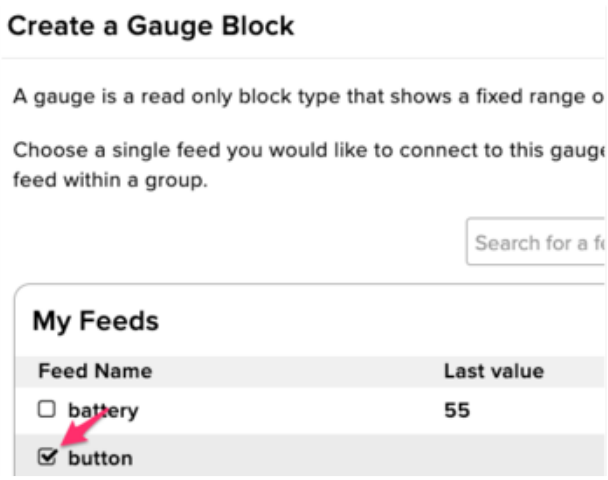
Change Button On Text to **1**
Change Button Off Text to **0**
Click Create block



Next up, we'll want to display button press data from your board on Adafruit IO. To do this, **we'll add a gauge block to the Adafruit IO dashboard**. A gauge is a read only block type that shows a fixed range of values.





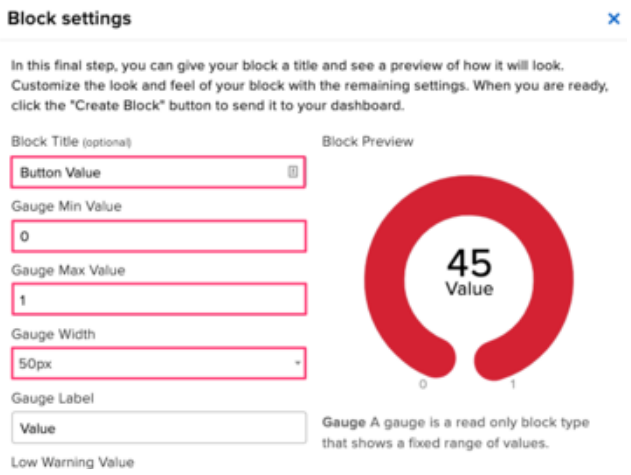


Click the cog at the top right hand corner of your dashboard.
In the dashboard settings dropdown, **click Create New Block.**
Select the gauge block.
Under My Feeds, **enter button as a feed name**.

Click Create.

Choose the button feed to connect it to the toggle block.

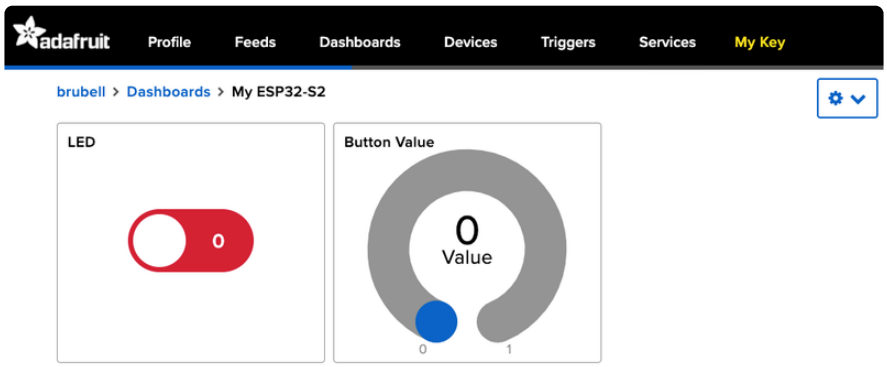
Click Next step.



Under block settings,

Change Block Title to **Button Value**
Change Gauge Min Value to **0**, the button's state when it's off
Change Gauge Max Value to **1**, the button's state when it's on
Click Create block

Your dashboard should look like the following:



Code Usage

For this example, you will need to open the **adafruitio_26_led_btn** example included with the **Adafruit IO Arduino** library. In the Arduino IDE, navigate to **File -> Examples -> Adafruit IO Arduino -> adafruitio_26_led_btn**.

Before uploading this code to the ESP32-S2/S3, you'll need to add your network and Adafruit IO credentials. **Click on the config.h tab** in the sketch.

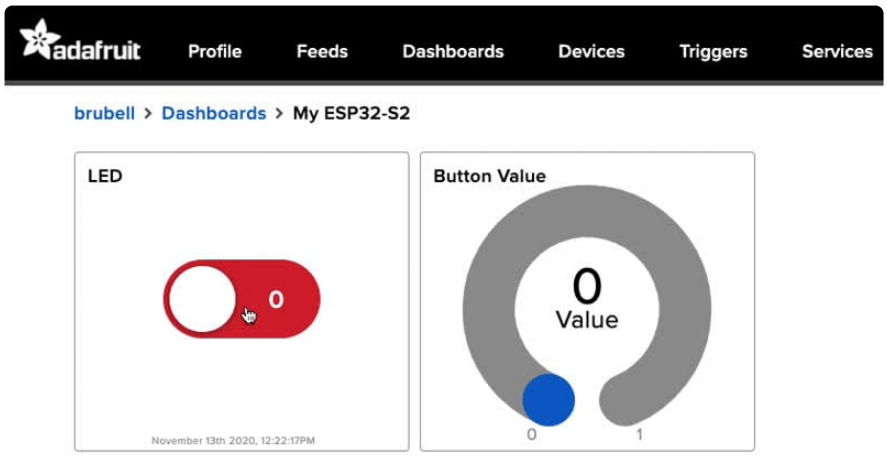
Obtain your Adafruit IO Credentials from navigating to io.adafruit.com and clicking My Key (https://adafru.it/fsU). Copy and paste these credentials next to `IO_USERNAME` and `IO_KEY`.

Enter your network credentials next to `WIFI_SSID` and `WIFI_PASS`.

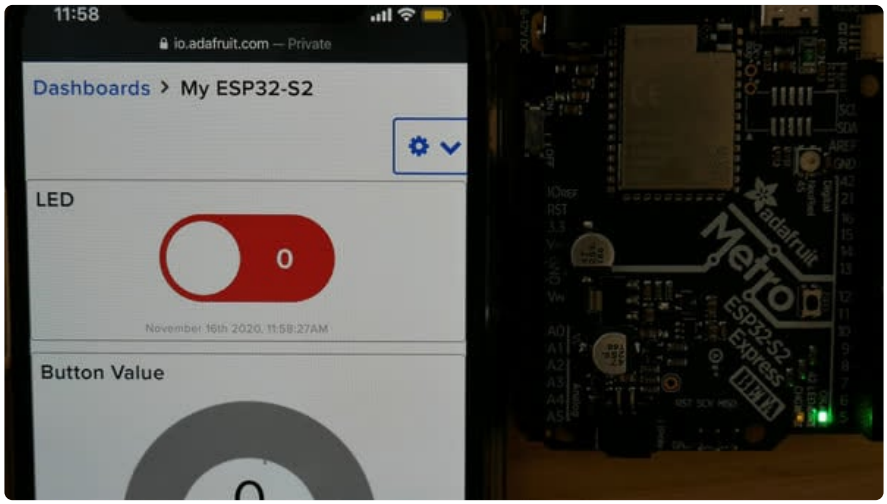
```
adafruitio_26_led_btn | config.h

20
21 #define WIFI_SSID "your_ssid"
22 #define WIFI_PASS "your_pass"
23
24 // uncomment the following line if you are using airlift
25 //#define USE_AIRLIFT
26
27 // uncomment the following line if you are using winc1500
28 // #define USE_WINC1500
29
```

Click the Upload button to upload your sketch to the ESP32-S2/S3. After uploading, **press the RESET button on your board to launch the sketch.**

Open the Arduino Serial monitor and **navigate to the Adafruit IO dashboard you created**. You should see the gauge response to button press and the board's LED light up in response to the Toggle Switch block.



You should also see the ESP32-S2/S3's LED turning on and off when the LED is toggled:



WipperSnapper Setup

The WipperSnapper firmware and ecosystem are in BETA and are actively being developed to add functionality, more boards, more sensors, and fix bugs. We encourage you to try out WipperSnapper with the understanding that it is not final release software and is still in development. If you encounter any bugs, glitches, or difficulties during the beta period, or with this guide, please contact us via http://io.adafruit.com/support

What is WipperSnapper

WipperSnapper is a firmware designed to turn any WiFi-capable board into an Internet-of-Things device without programming a single line of code. WipperSnapper connects to Adafruit IO (https://adafru.it/fsU), a web platform designed (by Adafruit! (https://adafru.it/Bo5)) to display, respond, and interact with your project's data.

Simply load the WipperSnapper firmware onto your board, add credentials, and plug it into power. Your board will automatically register itself with your Adafruit IO account.

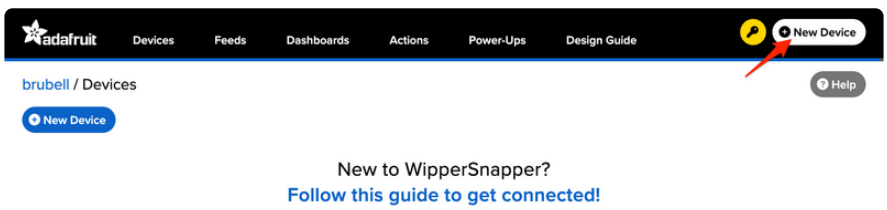
From there, you can add components to your board such as buttons, switches, potentiometers, sensors, and more! Components are dynamically added to hardware, so you can immediately start interacting, logging, and streaming the data your projects produce without writing code.

Sign up for Adafruit.io

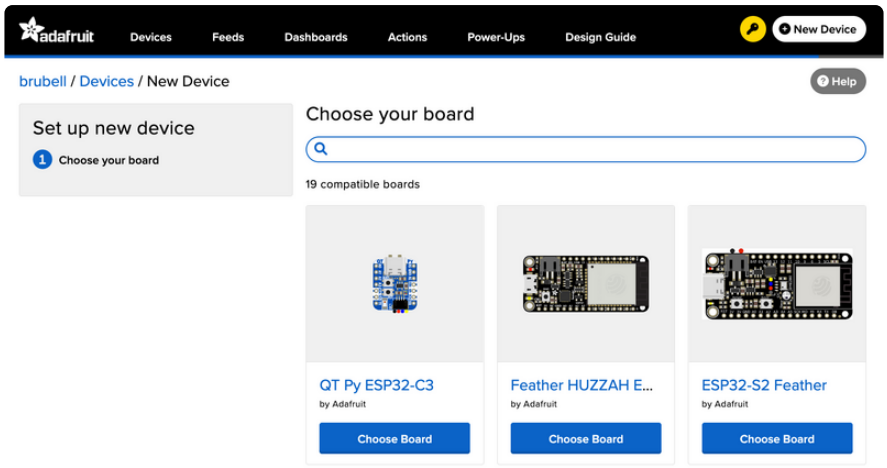
You will need an Adafruit IO account to use WipperSnapper on your board. If you do not already have one, head over to io.adafruit.com (https://adafru.it/fsU) to create a free account.

Add a New Device to Adafruit IO

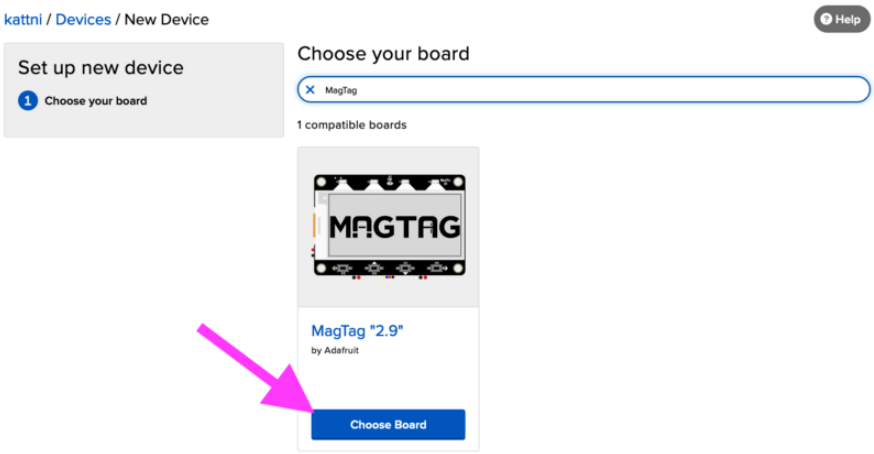
Log into your Adafruit IO (https://adafru.it/fsU) account. Click the New Device button at the top of the page.



After clicking New Device, you should be on the board selector page. This page displays every board that is compatible with the WipperSnapper firmware.



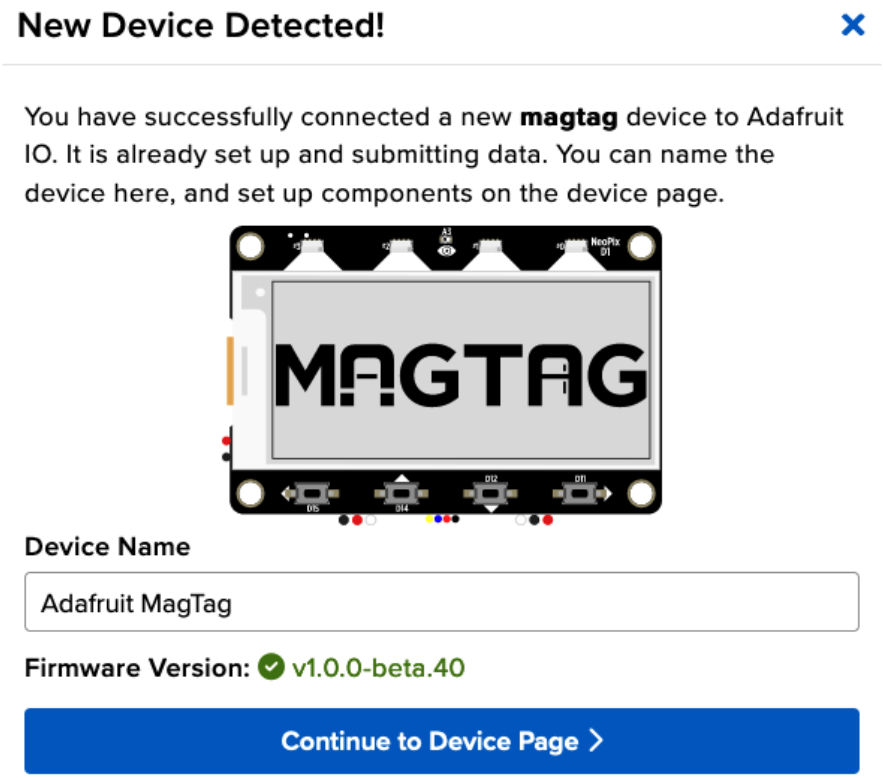
In the board selector page's search bar, search for the MagTag. Once you've located the board you'd like to install WipperSnapper on, click the **Choose Board** button to bring you to the self-guided installation wizard.



Follow the step-by-step instructions on the page to install Wippersnapper on your device and connect it to Adafruit IO.

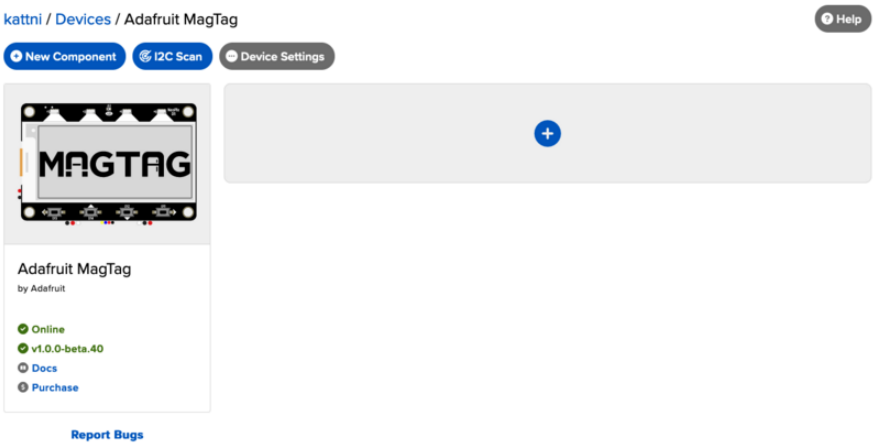
If the installation was successful, a popover should appear displaying that your board has successfully been detected by Adafruit IO.

Give your board a name and click "Continue to Device Page".



You should be brought to your board's device page.

Next, Visit this guide's **WipperSnapper Essentials** pages to learn how to interact with your board using Adafruit IO.



Feedback

Adafruit.io WipperSnapper is in **beta** and you can help improve it!

If you have suggestions or general feedback about the installation process - visit https://io.adafruit.com/support (https://adafru.it/Sgb), click "Contact Adafruit IO Support" and select "I have feedback or suggestions for the WipperSnapper Beta".

Troubleshooting

If you encountered an issue during installation, please try the steps below first.

If you're still unable to resolve the issue, or if your issue is not listed below, get in touch with us directly at https://io.adafruit.com/support (https://adafru.it/Sgb). Make sure to click "Contact Adafruit IO Support" and select "There is an issue with WipperSnapper. Something is broken!"

I don't see my board on Adafruit IO, it is stuck connecting to WiFi

First, make sure that you selected the correct board on the board selector.

Next, please make sure that you entered your WiFi credentials properly, there are no spaces/special characters in either your network name (SSID) or password, and that you are connected to a 2.4GHz wireless network.

If you're still unable to connect your board to WiFi, please make a new post on the WipperSnapper technical support forum with the error you're experiencing, the LED colors which are blinking, and the board you're using. (https://adafru.it/V6a)

I don't see my board on Adafruit IO, it is stuck "Registering with Adafruit IO"

Try hard-resetting your board by unplugging it from USB power and plugging it back in.

If the error is still occurring, please make a new post on the WipperSnapper technical support forum with information about what you're experiencing, the LED colors which are blinking (if applicable), and the board you're using. (https://adafru.it/V6a)

"Uninstalling" WipperSnapper

WipperSnapper firmware is an application that is loaded onto your board. There is nothing to "uninstall". However, you may want to "move" your board from running WipperSnapper to running Arduino or CircuitPython. You also may need to restore your board to the state it was shipped to you from the Adafruit factory.

Moving from WipperSnapper to CircuitPython

Follow the steps on the Installing CircuitPython page (https://adafru.it/Amd) to install CircuitPython on your board running WipperSnapper.

- If you are unable to double-tap the RST button to enter the UF2 bootloader, follow the "Factory Resetting a WipperSnapper Board" instructions below.

Uploading this sketch will overwrite WipperSnapper. If you want to re-install WipperSnapper, follow the instructions at the top of this page.

Moving from WipperSnapper to Arduino

If you want to use your board with Arduino, you will use the Arduino IDE to load any sketch onto your board.

First, follow the page below to set up your Arduino IDE environment for use with your board.

Setup Arduino IDE

https://adafru.it/10eE

Then, follow the page below to upload the "Arduino Blink" sketch to your board.

Upload Arduino Blink Sketch

https://adafru.it/10eF

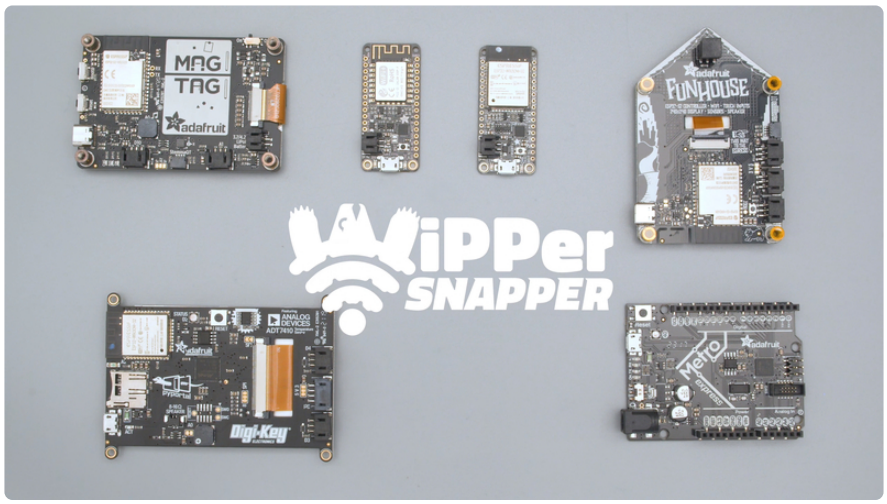
Uploading this sketch will overwrite WipperSnapper. If you want to re-install WipperSnapper, follow the instructions at the top of this page.

Factory Resetting a WipperSnapper Board

Sometimes, hardware gets into a state that requires it to be "restored" to the original state it shipped in. If you'd like to get your board back to its original factory state, follow the guide below.

This board does not have a Factory Reset firmware. You can re-install the UF2 bootloader (https://adafru.it/Pfk) and then install CircuitPython/Arduino by following the instructions above.

WipperSnapper Essentials



You've installed WipperSnapper firmware on your board and connected it to Adafruit IO. Next, to learn how to use Adafruit IO!

The Adafruit IO supports a large number of components. Components are physical parts such as buttons, switches, sensors, servos, LEDs, RGB LEDs, and more.

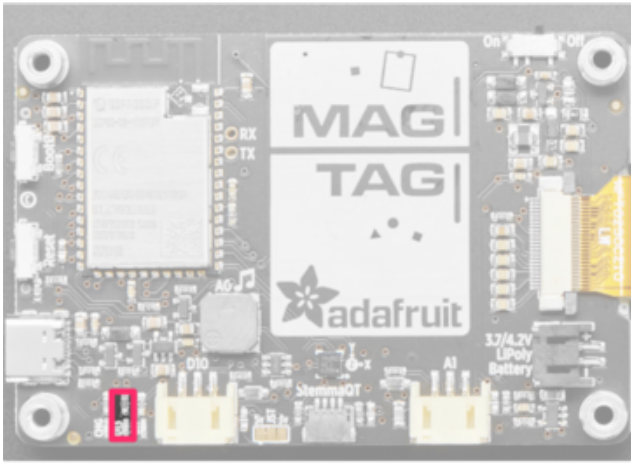
The following pages will get you up and running with WipperSnapper as you interact with your board's built-in components, such as read the value of a push button, send the value of an I2C sensor to the internet, and wirelessly control colorful LEDs.

LED Blink

This demo shows controlling an LED from Adafruit IO. The same kind of control can be used for relays, lights, motors, or solenoids.

One of the first programs you typically write to get used to embedded programming is a sketch that repeatably blinks an LED. IoT projects are wireless, so after completing this section, you'll be able to turn on (or off) the LED built into your board from anywhere in the world.

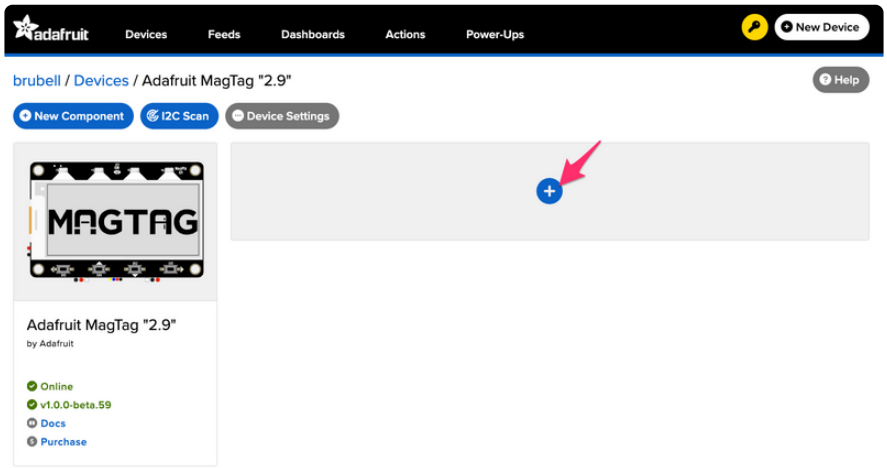
Where is the LED on my board?

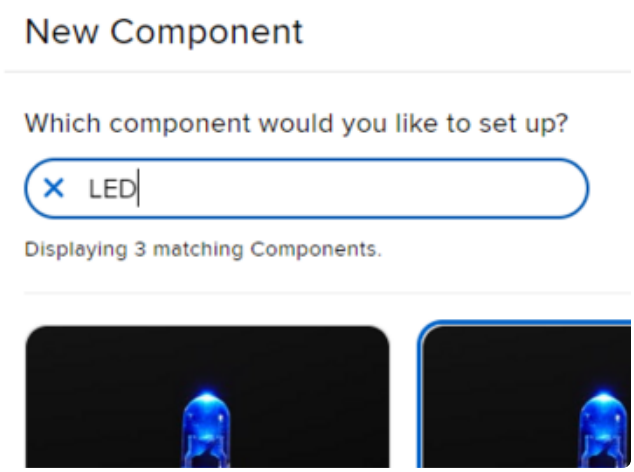


On the back, on the bottom left, is a **red LED** labeled **D13**.

Create a LED Component on Adafruit IO

On the device page, click the New Component (or "+") button to open the component picker.





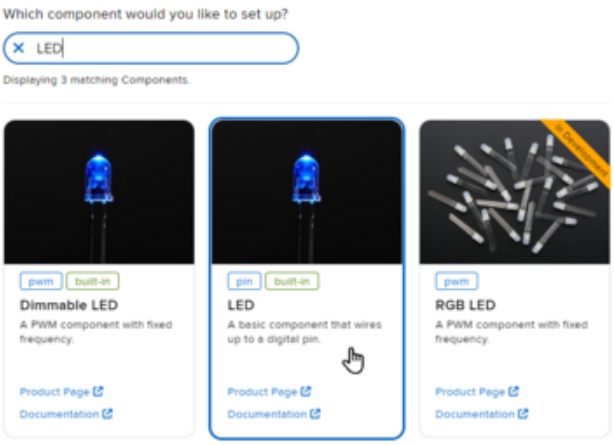
Search for the component name by entering `LED` into the text box on the component picker, the list of components should update as soon as you stop typing.

Filtering and searching for components

Since WipperSnapper supports such a large number of components, there is keyword filtering. Try searching for various keywords, like:

- component names: `aht20`, `servo`, `buzzer`, `button`, `led`, etc
- sensor types: `light`, `temperature`, `pressure`, `humidity`, etc
- interface: `i2c`, `uart`, `ds18x20`, `pin`, etc (also I2C addresses e.g. `0x44`)
- vendor: `Adafruit`, `ASAIR`, `Infineon`, `Bosch`, `Honeywell`, `Sensirion`, etc

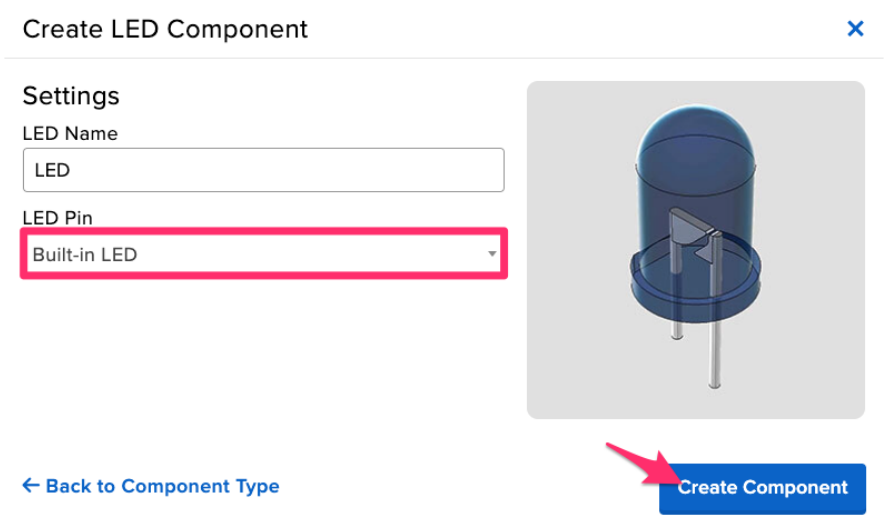
There are also product and documentation links to every component. Follow the links beneath the component descriptions to be taken to the appropriate product page or Learn Guide



Select the **LED** from the list of components.

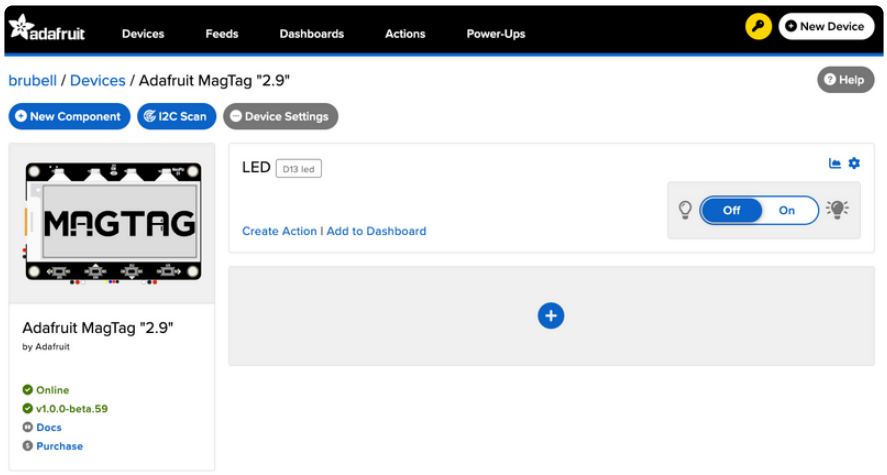
On the Create LED Component form, the board's LED pin is pre-selected.

Click Create Component.



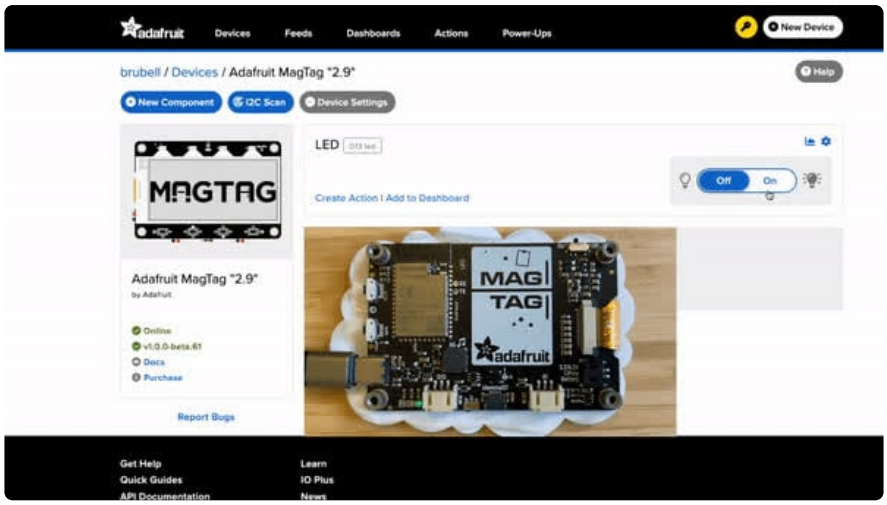
Behind the scenes, Adafruit IO sends send a command to your board running WipperSnapper telling it to configure "LED Pin" as a digital output.

Your board's page on Adafruit IO shows a new LED component.



Usage

On the board page, toggle the LED component by clicking the toggle switch. This should turn your board's built-in LED on or off.

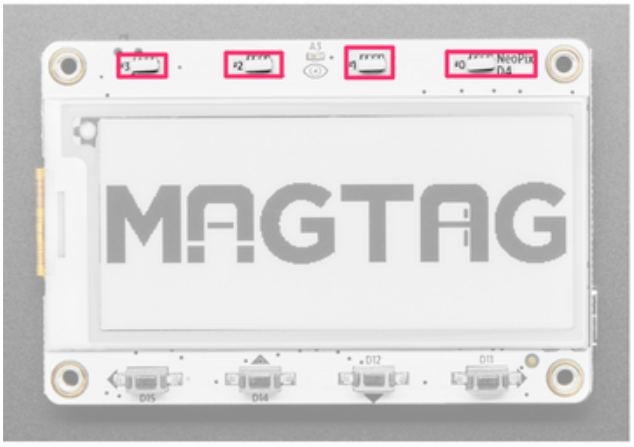


NeoPixel LEDs

Your board has a WS281x RGB LED (NeoPixel, in Adafruit jargon) built in. Boards running the WipperSnapper firmware can be wirelessly controlled by Adafruit IO to interact with NeoPixels.

On this page, you'll learn how to change the color and brightness of the NeoPixel built into your board from Adafruit IO.

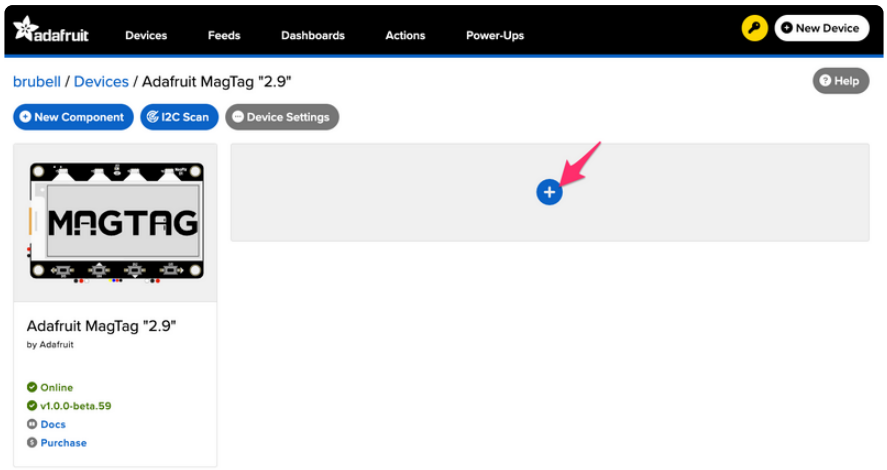
Where is the NeoPixel on my board?

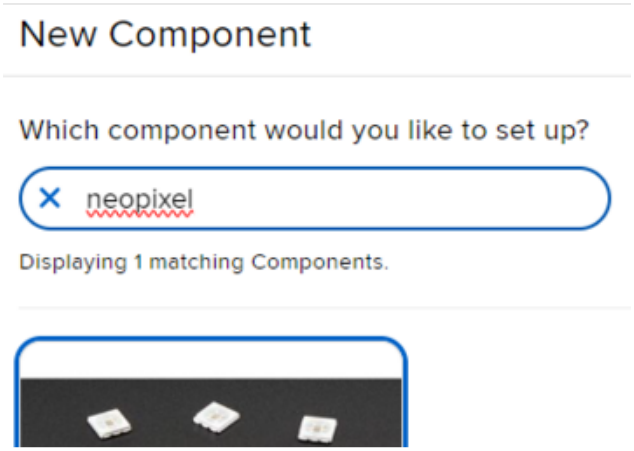


On the front of the board, along the top, are **four addressable RGB side-emitting NeoPixel LEDs** labeled together as **NeoPix**.

Create the NeoPixel Component

On the device page, click the New Component (or "+") button to open the component picker.





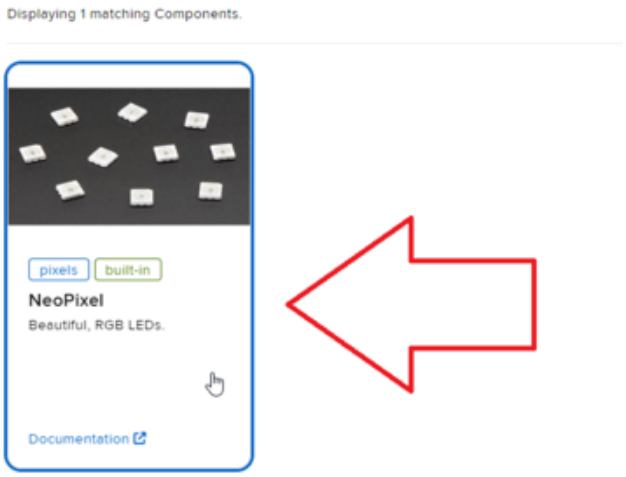
Search for the component name by entering `neopixel` into the text box on the component picker, the list of components should update as soon as you stop typing

Filtering and searching for components

Since WipperSnapper supports such a large number of components, you can use keyword filtering. Try searching for various keywords, like:

- component names: `aht20`, `servo`, `buzzer`, `button`, `neopixel`, etc
- sensor types: `light`, `temperature`, `pressure`, `humidity`, etc
- interface: `i2c`, `uart`, `ds18x20`, `pin`, etc (also I2C addresses e.g. `0x44`)
- vendor: `Adafruit`, `ASAIR`, `Infineon`, `Bosch`, `Honeywell`, `Sensirion`, etc

There is also added product and documentation links for every component, follow the links beneath the component descriptions to be taken to the appropriate product page or Learn Guide



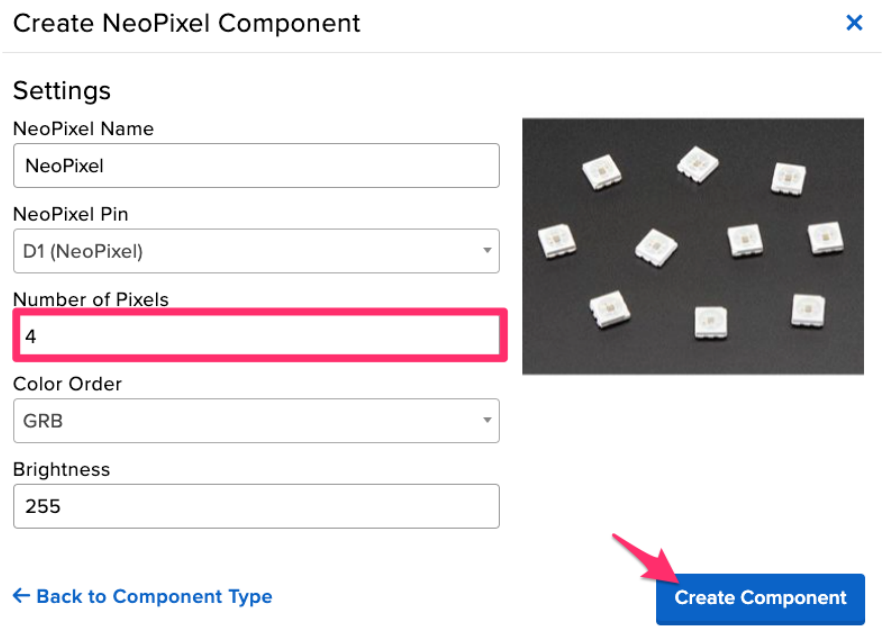
Select the **NeoPixel** from the list of results to go to the component configuration page.

There will be a back button if you select the wrong component, and you can use the Edit component icon (⚙) on the device page to update the component configuration in the future.

The board NeoPixel pin is automatically found and selected.

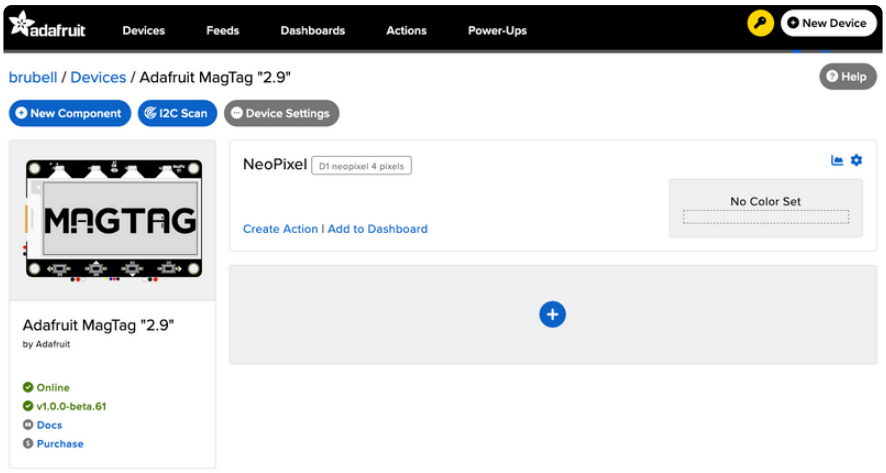
Set the **Number of Pixels** to 4, reflecting the 4 NeoPixels on the MagTag.

Click **Create Component**



Behind the scenes, Adafruit IO sends a command to your board running WipperSnapper firmware telling it to configure the pin as a NeoPixel component with the settings from the form.

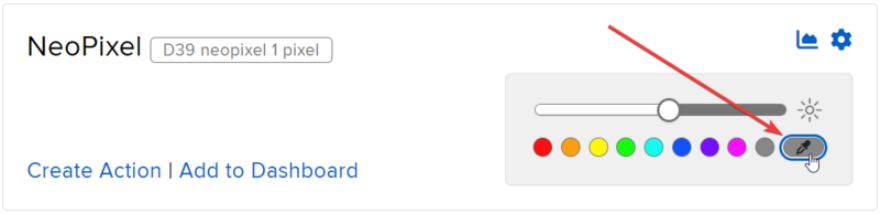
The Device page shows the NeoPixel component.



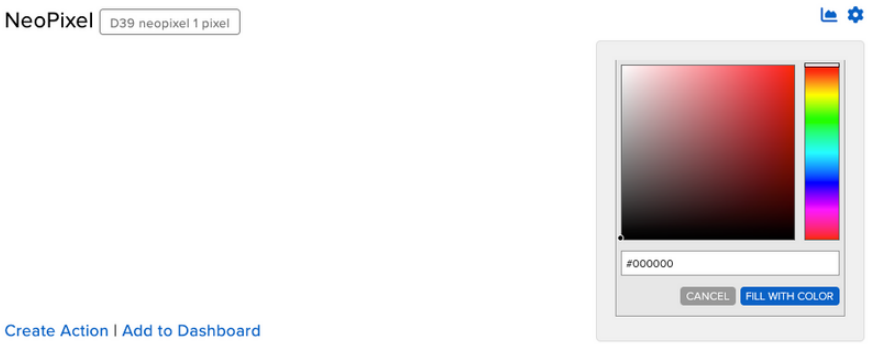
Set the NeoPixel's RGB Color

Since no colors have been set yet, the color picker's default value is `#000000` (black in hex color code) and appears "off". You can change that to make the NeoPixel shine brightly!

On the NeoPixel component, click the color dropper at the end of the color swatch list.



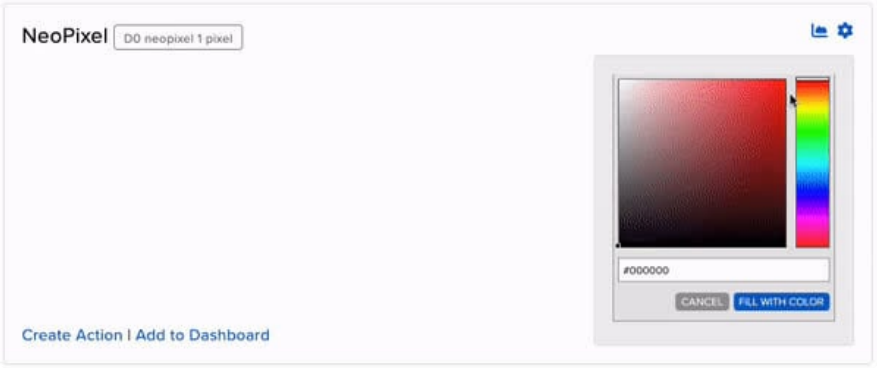
A color picker pops open! Next: learning how Adafruit IO uses hex color codes to represent the colors on your NeoPixel.



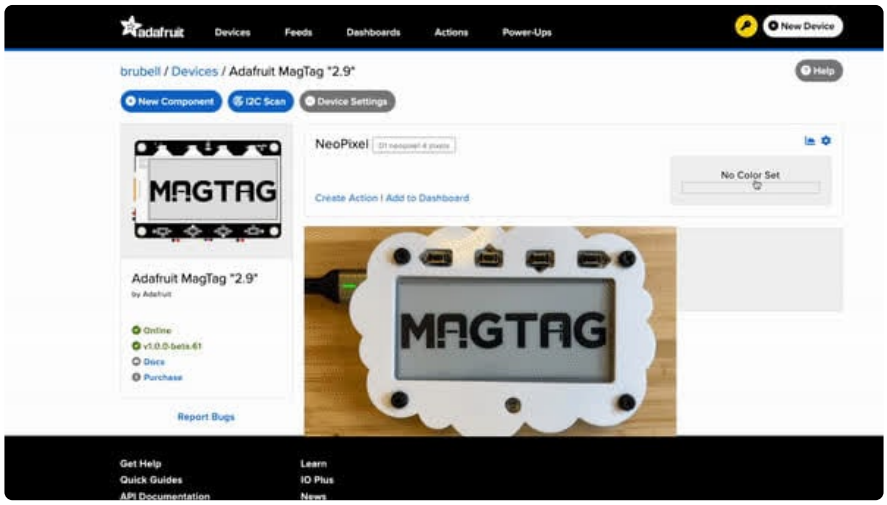
Hex Colors 101

The color picker on Adafruit IO uses hex color codes to represent Red, Green, and Blue values. For example, `#FF0000` is the hex color code for the color red. The colors (`#FF0000`) red component is `FF` (255 translated to decimal), the green component is `00` and the blue component is `00`. Translated to RGB format, the color is `RGB (255, 0, 0)`.

Using the color picker, or by manually entering a hex color code, select a color.

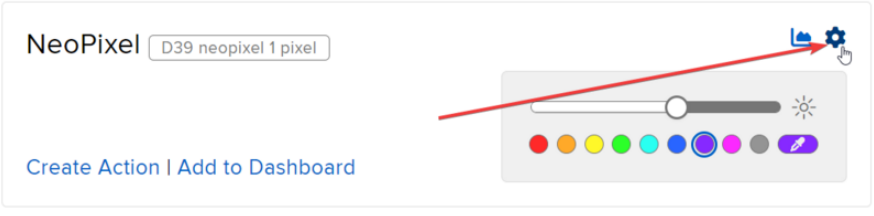


When you're ready to set the color of your device's NeoPixel, click **FILL WITH COLOR**. The NeoPixel will immediately glow!



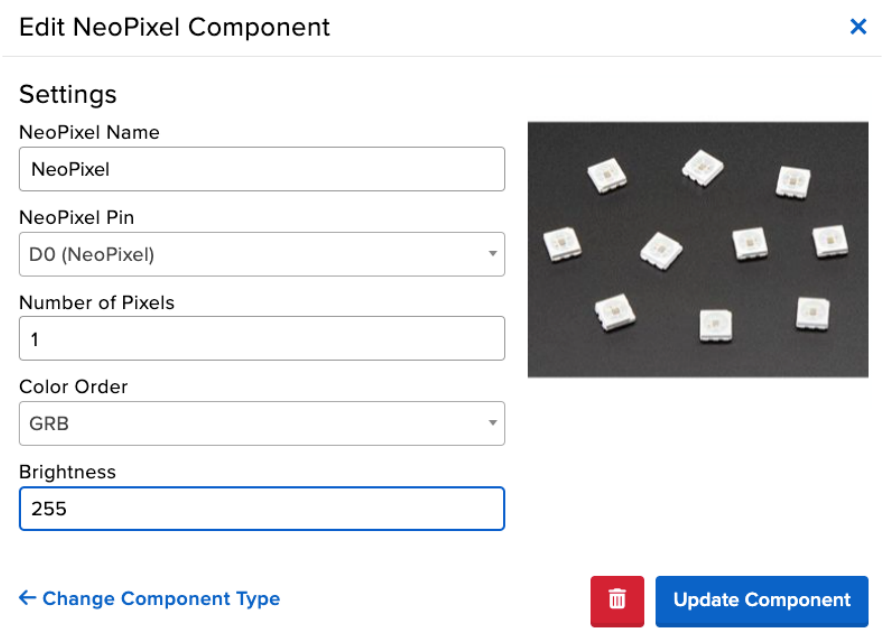
Set NeoPixel Brightness

If the NeoPixel is too bright (or too dim), you can change the overall brightness. Click the gear/cog icon on the NeoPixel component to open its settings.



On the NeoPixel component form, set Brightness to a value between 0 (fully off) and 255 (full brightness).

Click the **Update Component** button to send the updated configuration to your device.



Read a Push-button

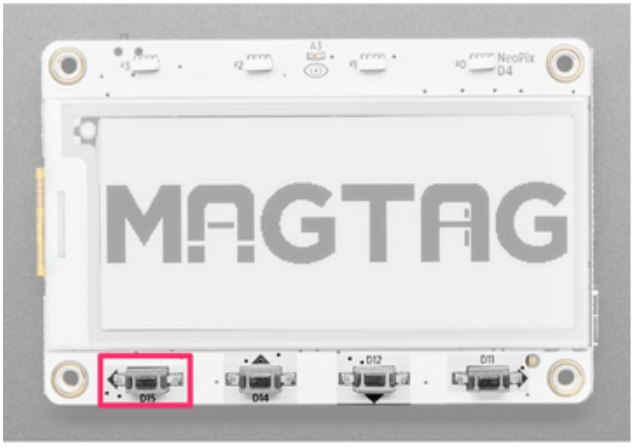
This demo shows reading the state of a push-button from WipperSnapper. But the same kind of control can be used for reading switches, break beam sensors, and other digital sensors.

You can configure a board running WipperSnapper to read data from standard input buttons, switches, or digital sensors, and send the value to Adafruit IO.

From Adafruit IO, you will configure one of the pushbuttons on your board as a push button component. Then, when the button is pressed (or released), a value will be published to Adafruit IO.

Button Location

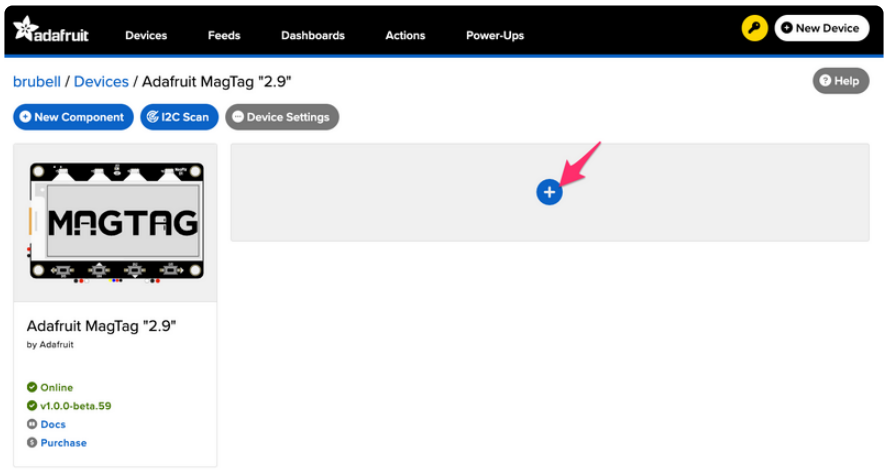
This example uses the board's built-in push-button and internal pull-up resistor instead of wiring a push-button up.

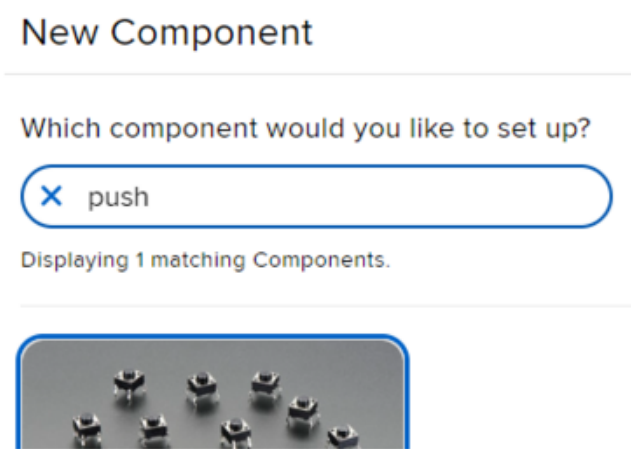


On the front of the board, along the bottom, there are four user-controllable buttons. In this example, we are going to use the first button on the left-hand side of the MagTag display, labeled **D15** on the silkscreen.

Create a Push-button Component on Adafruit IO

On the device page, click the New Component (or "+") button to open the component picker.





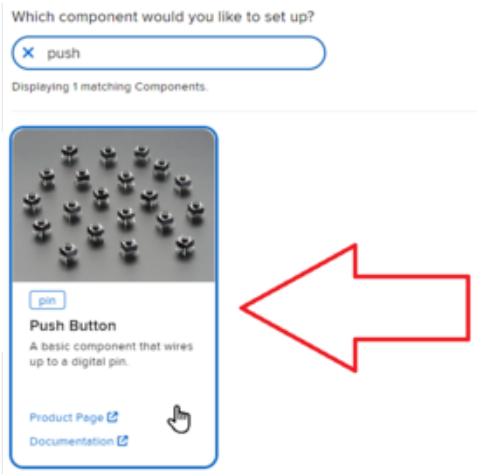
Search for the component name by entering `push` into the text box on the component picker, the list of components should update as soon as you stop typing.

Filtering and searching for components

Since WipperSnapper supports such a large number of components, you can use filtering. Try searching for various keywords, like:

- component names: `aht20`, `servo`, `buzzer`, `button`, `potentiometer`, etc
- sensor types: `light`, `temperature`, `pressure`, `humidity`, etc
- interface: `i2c`, `uart`, `ds18x20`, `pin`, etc (also I2C addresses e.g. `0x44`)
- vendor: `Adafruit`, `ASAIR`, `Infineon`, `Bosch`, `Honeywell`, `Sensirion`, etc

There are also added product and documentation links for every component. Follow the links beneath the component descriptions to be taken to the appropriate product page or Learn Guide.

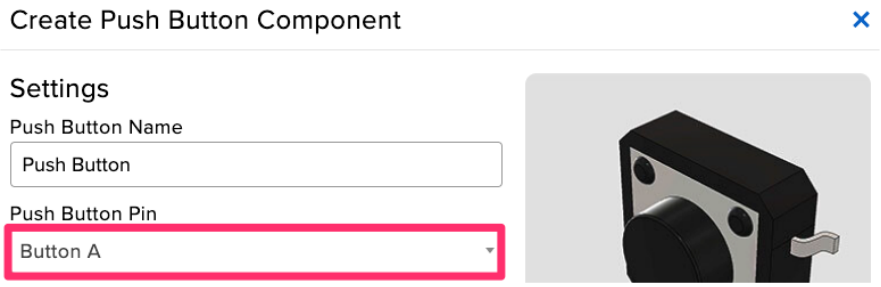


Select the **Push Button** from the list of results to go to the component configuration page.

There will be a back button if you select the wrong component, and you can use the Edit component icon (⚙) on the device page to update the component configuration in the future.

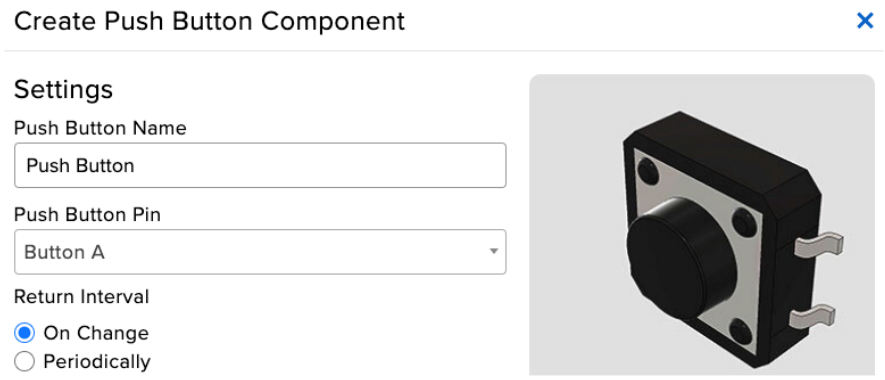
The "Create Push Button Component" form presents you with options for configuring the push button.

Start by selecting the board's pin connected to the push button.

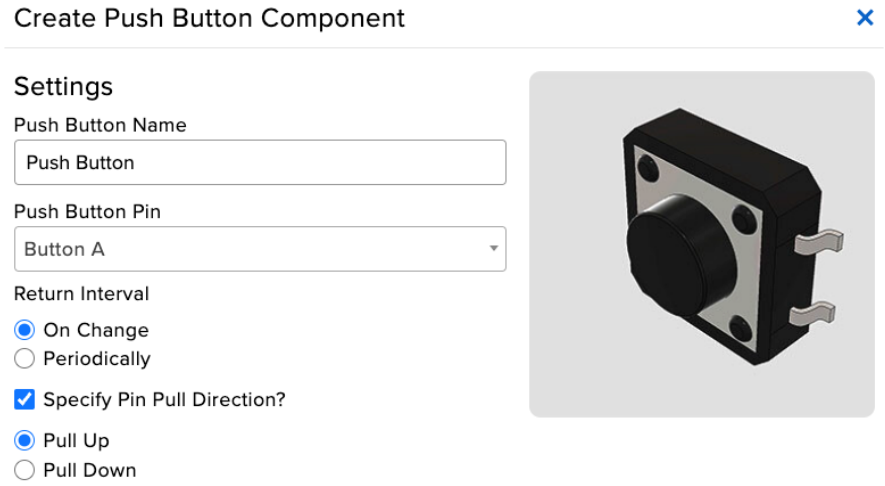


The Return Interval dictates how frequently the value of the push-button will be sent from the board to Adafruit IO.

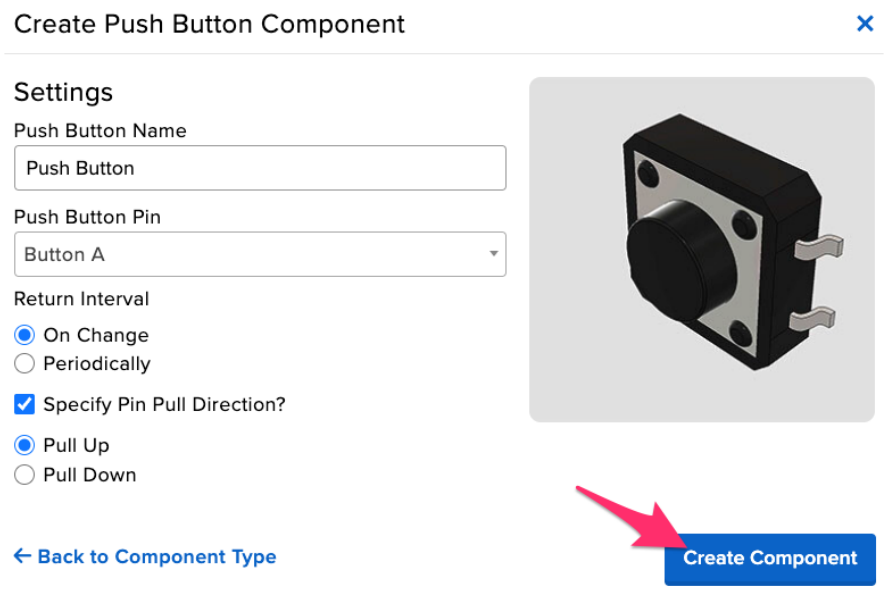
For this example, you will configure the push button's value to be only sent when the value changes (i.e. when it's either pressed or depressed).



Finally, check the **Specify Pin Pull Direction** checkbox and select the pull direction.

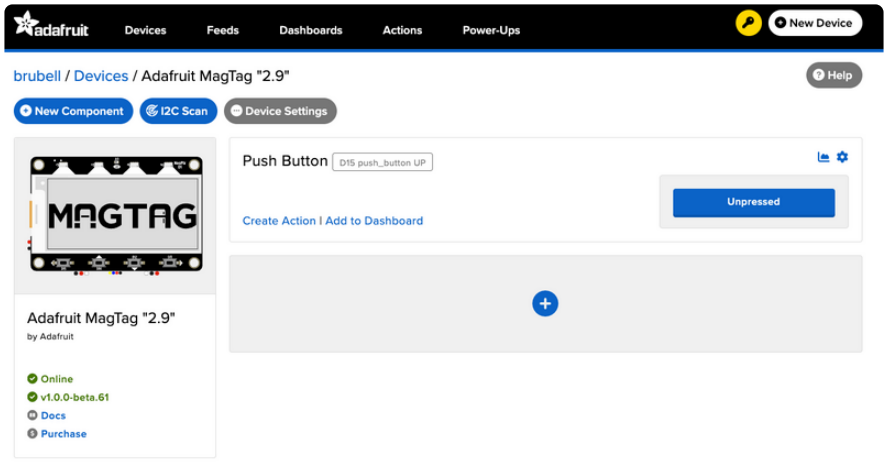


Make sure the form's settings look like the following screenshot. Then, click **Create Component**.

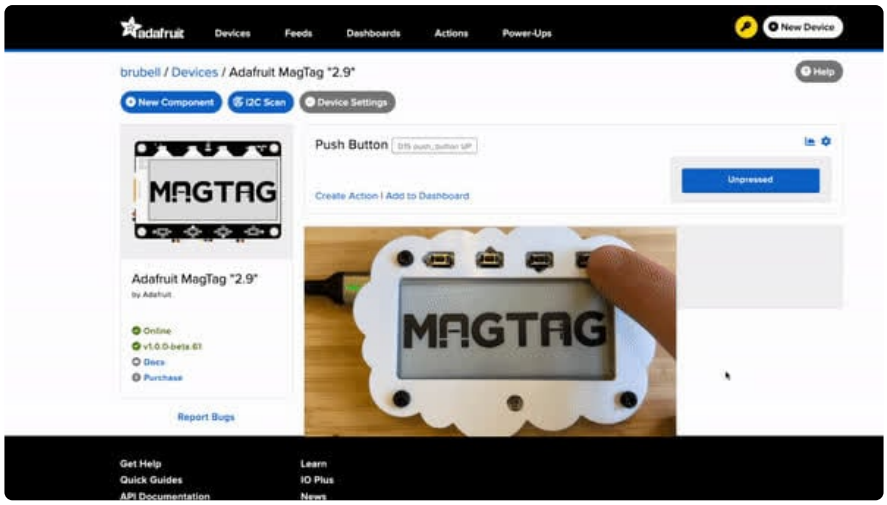


Adafruit IO sends a command to your WipperSnapper board, telling it to configure the GPIO pin you selected to behave as a digital input pin and to enable it to pull up the internal resistor.

Your board's page should also show the new push-button component.



Push the button on your board to change the value of the push-button component on Adafruit IO.



Analog Input: Light Sensor

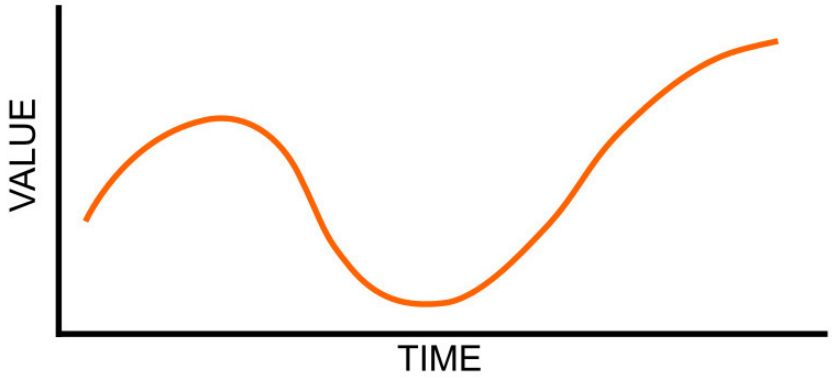
This page demonstrates using a light sensor as an analog input. However, the same process can be used for other analog input components on Adafruit IO such as the potentiometer.

Your microcontroller board has both digital and analog signal capabilities. Some pins are analog, some are digital, and some are capable of both. Check the **Pinouts** page in this guide for details about your board.

Analog signals are different from digital signals in that they can be any voltage and can vary continuously and smoothly between voltages. An analog signal is like a dimmer switch on a light, whereas a digital signal is like a simple on/off switch.

Digital signals only can ever have two states, they are either are **on** (high logic level voltage like 3.3V) or **off** (low logic level voltage like 0V / ground).

By contrast, analog signals can be any voltage in-between on and off, such as 1.8V or 0.001V or 2.98V and so on.



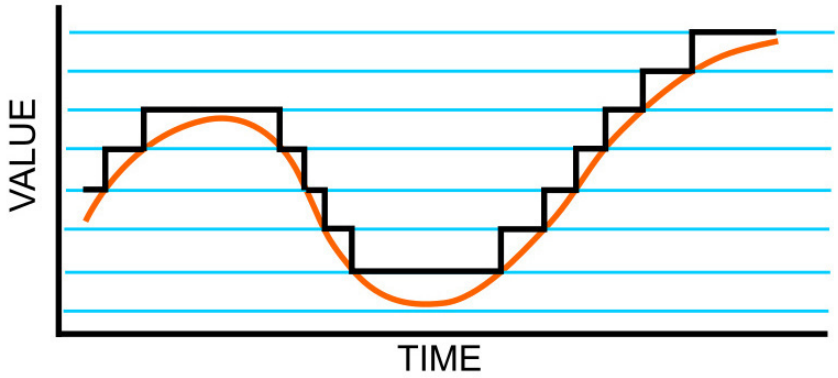
Analog signals are continuous values which means they can be an infinite number of different voltages. Think of analog signals like a floating point or fractional number, they can smoothly transiting to any in-between value like 1.8V, 1.81V, 1.801V, 1.8001V, 1.80001V and so forth to infinity.

Many devices use analog signals, in particular sensors typically output an analog signal or voltage that varies based on something being sensed like light, heat, humidity, etc.

Analog to Digital Converter (ADC)

An analog-to-digital-converter, or ADC, is the key to reading analog signals and voltages with a microcontroller. An ADC is a device that reads the voltage of an analog signal and converts it into a digital, or numeric, value. The microcontroller can't read analog signals directly, so the analog signal is first converted into a numeric value by the ADC.

The black line below shows a digital signal over time, and the red line shows the converted analog signal over the same amount of time.



Once that analog signal has been converted by the ADC, the microcontroller can use those digital values any way you like!

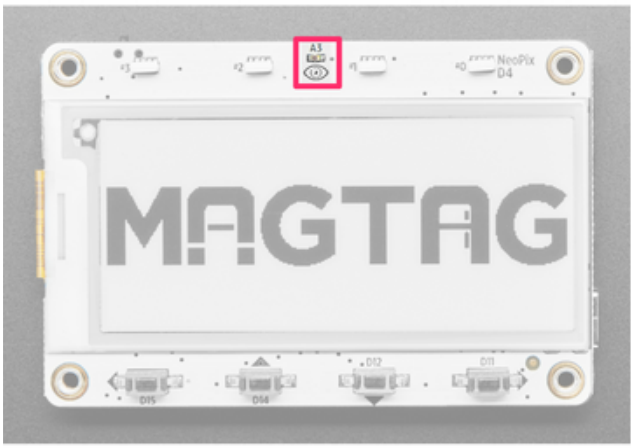
Light Sensor

A light sensor (also known as a CdS cell, light-dependent resistor, or photoresistor) detects light. They change their resistive value (in ohms, Ω) depending on how much light shines into the photocell.

When a light sensor is exposed to more light, the resistance decreases. When it is exposed to less light, the resistance increases.

By using a light sensor wired in a specific way (as a voltage divider), we can turn resistance into voltage. That change is then read by your board's Analog-to-Digital converter and sent to Adafruit IO.

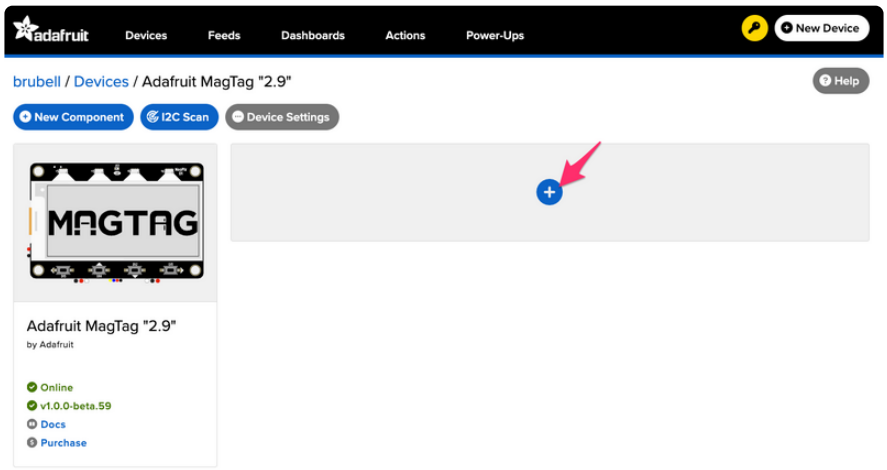
Where is the Light Sensor on my board?

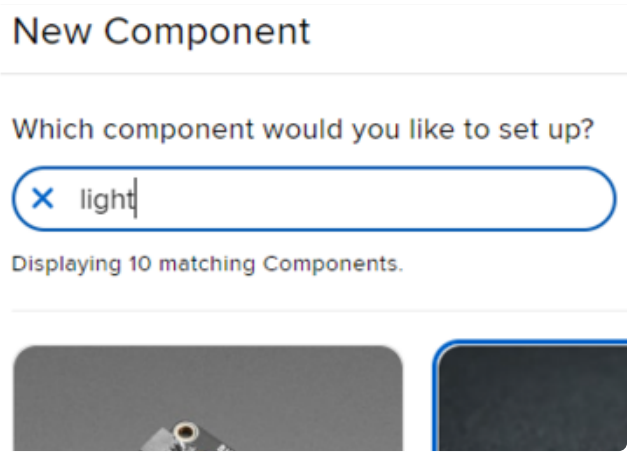


On the front of the board, in the center of the top is a front-facing **light sensor** labeled with **A3** and an eye.

Create a Light Sensor Component

On the device page, click the New Component (or "+") button to open the component picker.





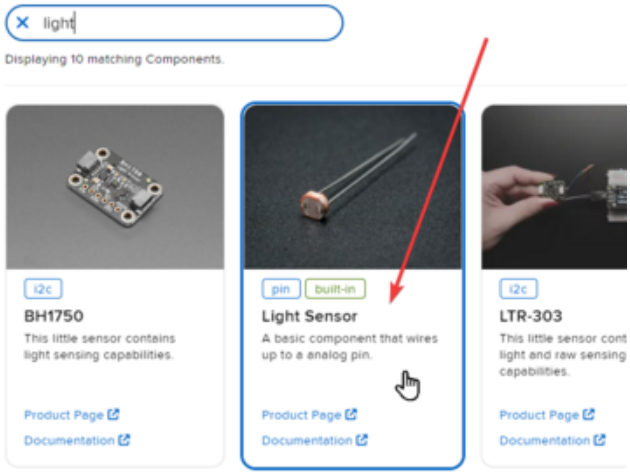
Search for the component name by entering `light` into the text box on the component picker, the list of components should update as soon as you stop typing.

Filtering and searching for components

WipperSnapper supports such a large number of components we added filtering!
Try searching for various keywords, like:

- component names: `aht20`, `servo`, `buzzer`, `button`, `relay`, etc
- sensor types: `light`, `temperature`, `pressure`, `humidity`, etc
- interface: `i2c`, `uart`, `ds18x20`, `pin`, etc (also I2C addresses e.g. `0x44`)
- vendor: `Adafruit`, `ASAIR`, `Infineon`, `Bosch`, `Honeywell`, `Sensirion`, etc

We've also added product and documentation links to every component, follow the links beneath the component descriptions to be taken to the appropriate product page or Learn-Guide.

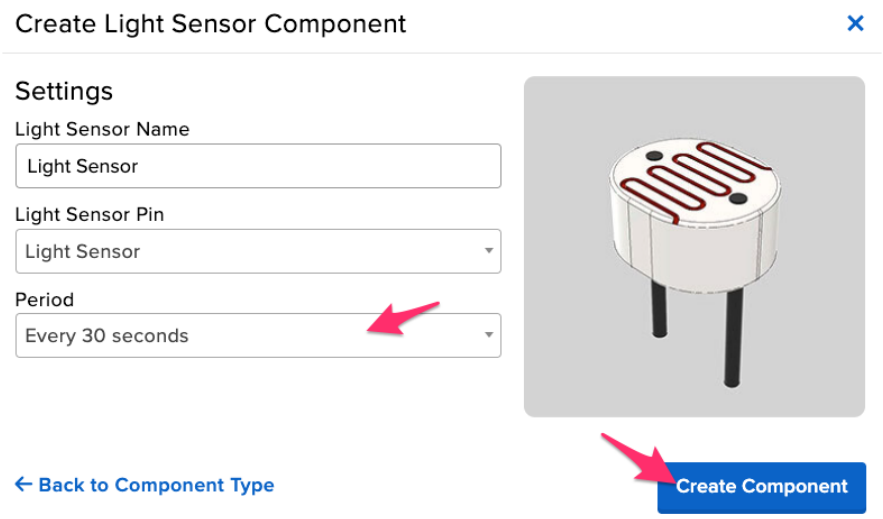


Select the **Light Sensor** from the list of results to go to the component configuration page.

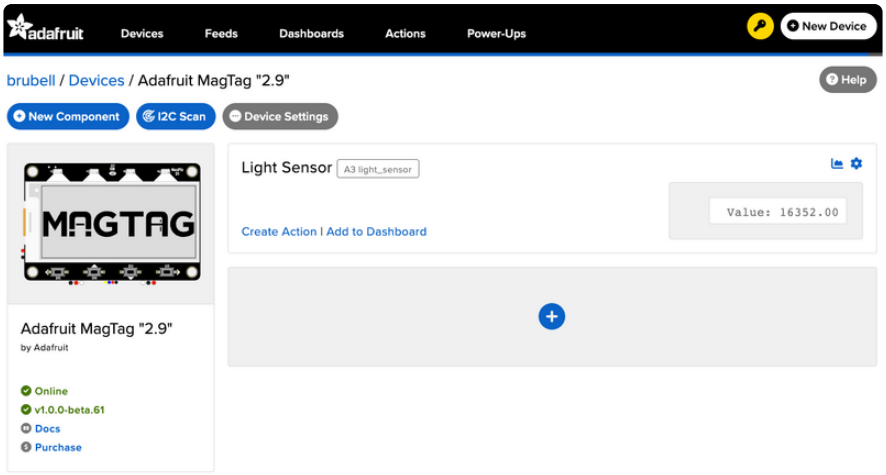
There will be a back button if you select the wrong component, and you can use the Edit component icon (⚙) on the device page to update the component configuration in the future.

The name and pin for the light sensor on your board are automatically selected. The **Period** determines how frequently the light sensor's value will be checked and sent to Adafruit IO. Set it to check the light sensor value every 30 seconds.

Click Create Component.

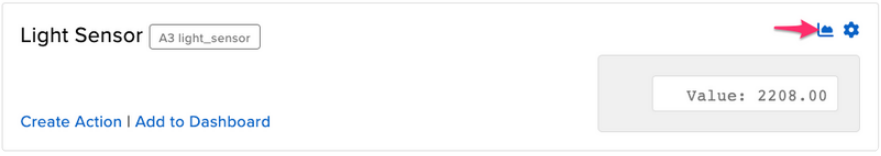


The device page shows a new light sensor component. The value of this component will change every 30 seconds.

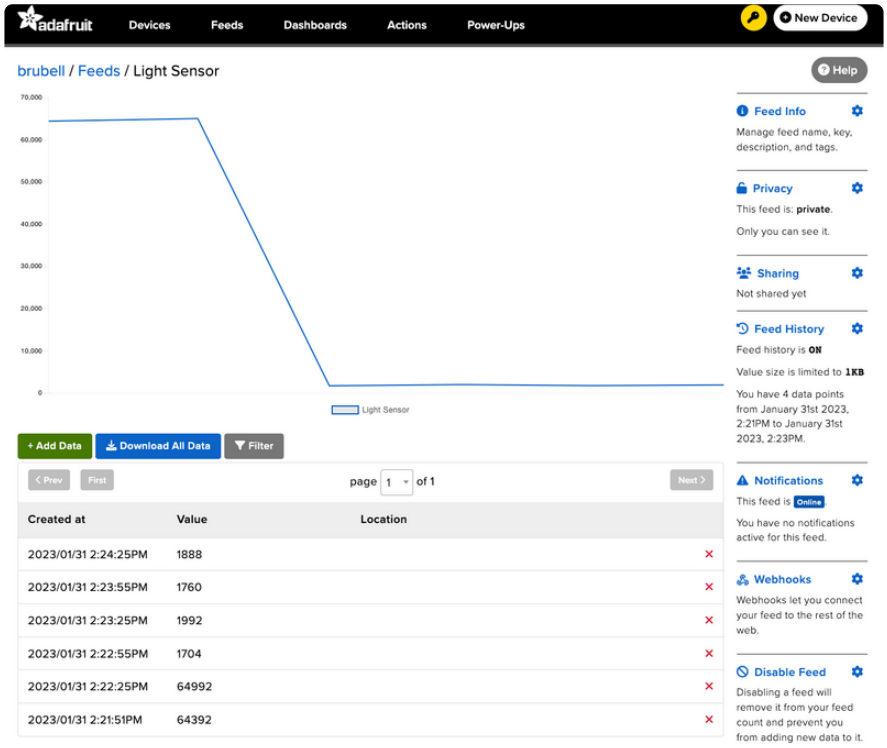


Light Sensor Usage

To test the light sensor, try covering the light sensor with a piece of paper. Navigate to the feed page by clicking the graph icon on the top right corner of the light sensor component.



On the light sensor feed page, you'll be able to observe a graph of the light sensor values as they change over time.



I2C Sensors

While this page uses the "MCP9808 High Accuracy I2C Temperature Sensor Breakout", the process for adding an I2C sensor to your board running WipperSnapper is similar for all I2C sensors.

Inter-Integrated Circuit, aka **I2C**, is a two-wire protocol for connecting sensors and "devices" to a microcontroller. A large number of sensors, including the ones sold by Adafruit, use I2C to communicate.

Typically, using I2C with a microcontroller involves programming. Adafruit IO and WipperSnapper let you configure a microcontroller to read data from an I2C sensor and publish that data to the internet without writing code.

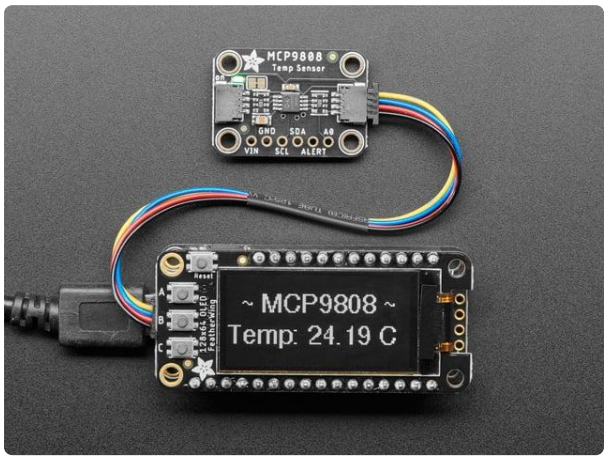
The WipperSnapper firmware supports a number of I2C sensors, viewable in list format here (https://adafru.it/Zbq).

- If you do not see the I2C sensor you're attempting to use with WipperSnapper, Adafruit has a guide on adding a component to Adafruit IO WipperSnapper here (https://adafru.it/Zbr).

On this page, you'll learn how to wire up an I2C sensor to your board. Then, you'll create a new component on Adafruit IO for your I2C sensor and send the sensor values to Adafruit IO. Finally, you'll learn how to locate, interpret, and download the data produced by your sensors.

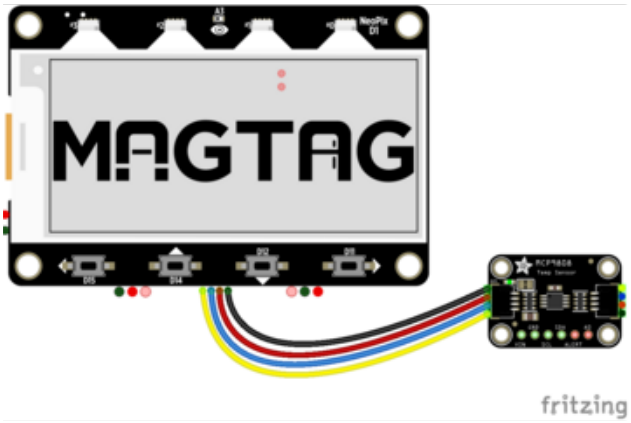
Parts

You will need the following parts to complete this page:



Adafruit MCP9808 High Accuracy I2C Temperature Sensor Breakout
The MCP9808 digital temperature sensor is one of the more accurate/precise we've ever seen, with a typical accuracy of ±0.25°C over the sensor's -40°C to...
https://www.adafruit.com/product/5027

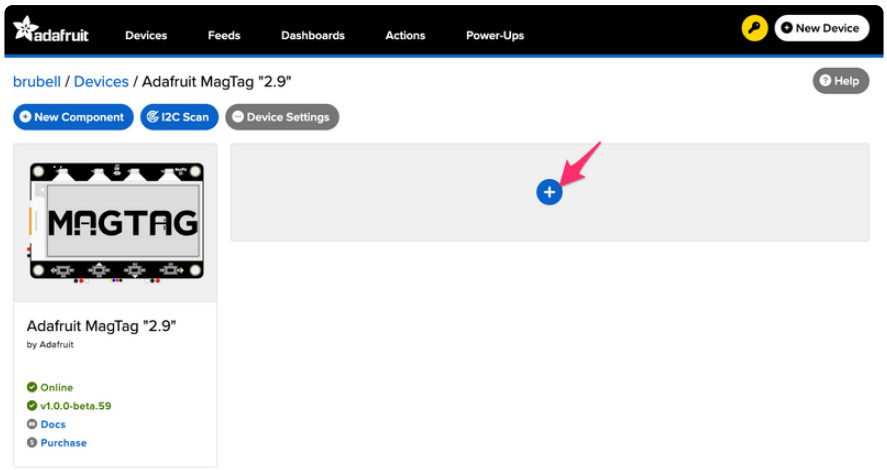
Wiring

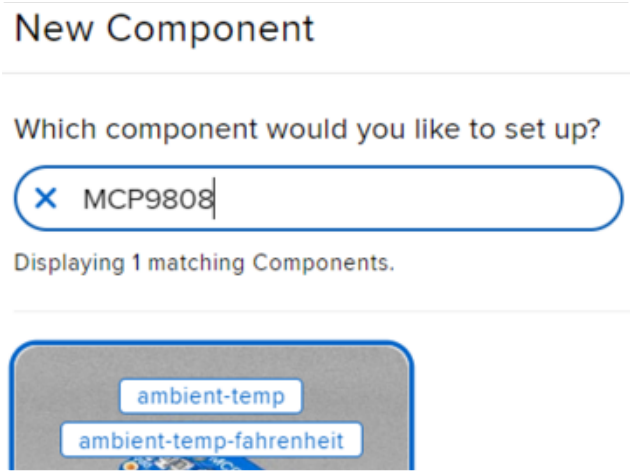


With a STEMMA Male to Male Cable, connect the board STEMMA QT Port to the MCP9808 STEMMA QT Port.

Add an MCP9808 Component

On the device page, click the New Component (or "+") button to open the component picker.





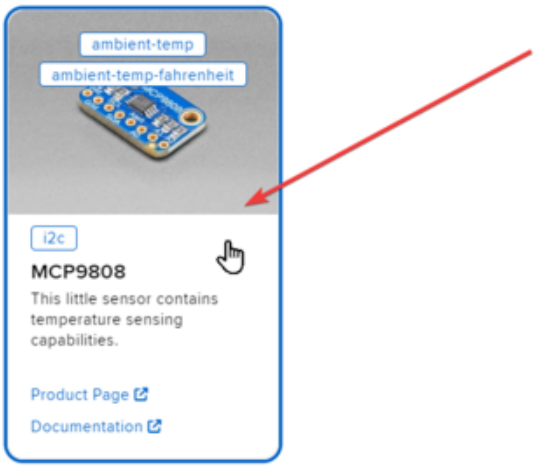
Search for the component name by entering `MCP9808` into the text box on the component picker, the list of components should update as soon as you stop typing.

Filtering and searching for components

Since WipperSnapper supports such a large number of components, there is keyword filtering. Try searching for various keywords, like:

- component names: `aht20`, `servo`, `buzzer`, `button`, `potentiometer`, etc
- sensor types: `light`, `temperature`, `pressure`, `humidity`, etc
- interface: `i2c`, `uart`, `ds18x20`, `pin`, etc (also I2C addresses e.g. `0x44`)
- vendor: `Adafruit`, `ASAIR`, `Infineon`, `Bosch`, `Honeywell`, `Sensirion`, etc

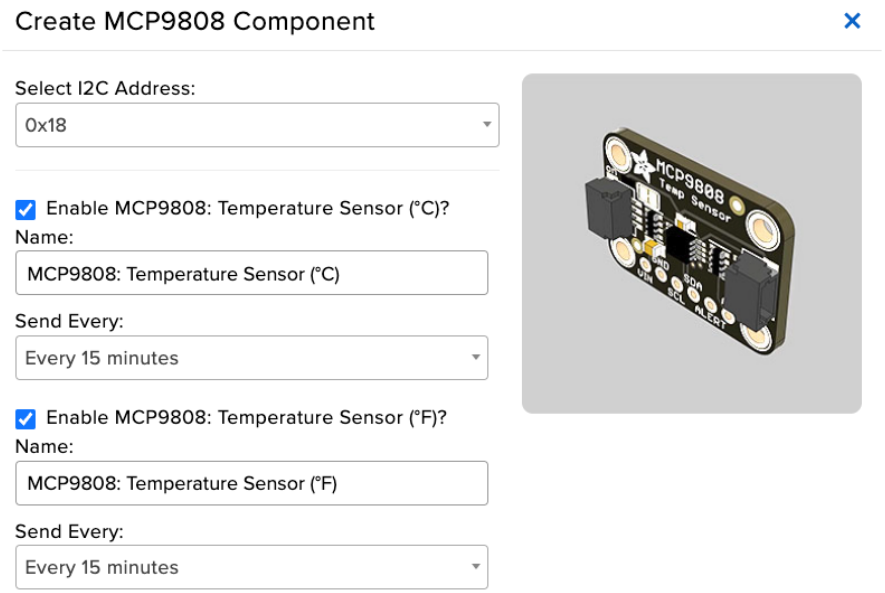
There are added product and documentation links for every component, follow the links beneath the component descriptions to be taken to the appropriate product page or Learn Guide.



Select the **MCP9808** from the list of results to go to the component configuration page.

There will be a back button if you select the wrong component, and you can use the Edit component icon (⚙) on the device page to update the component configuration in the future.

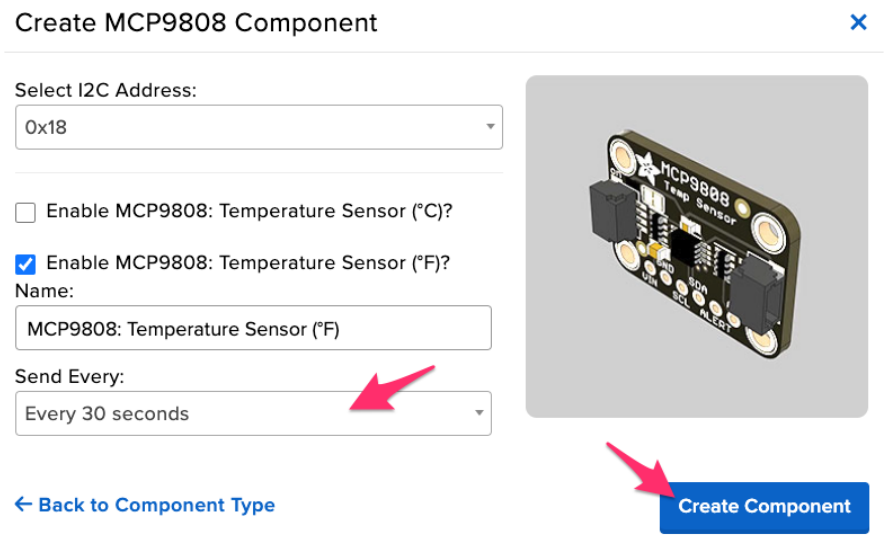
On the component configuration page, the MCP9808's I2C sensor address should be listed along with the sensor's settings.



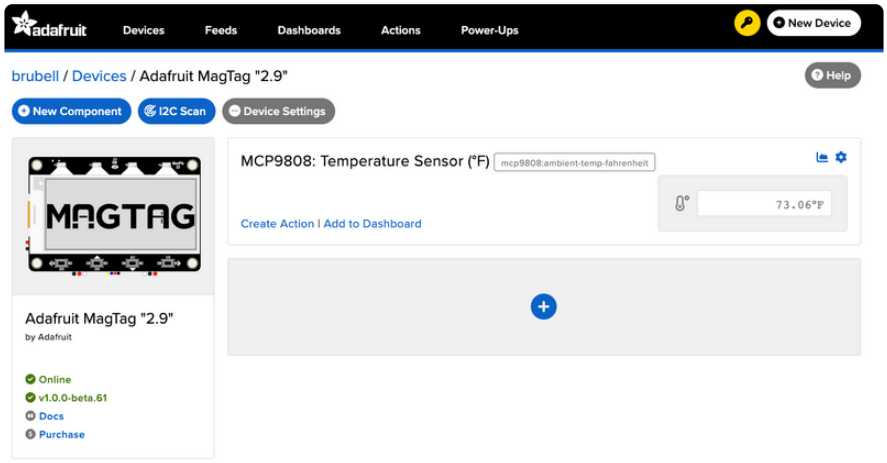
The MCP9808 sensor can measure ambient temperature. This page has individual options for reading the ambient temperature, in either Celsius or Fahrenheit. You may select the readings which are appropriate to your application and region.

The **Send Every** option is specific to each sensor measurement. This option will tell the board how often it should read from the sensor and send the data to Adafruit IO. Measurements can range from every 30 seconds to every 24 hours.

For this example, set the **Send Every** interval for both seconds to **Every 30 seconds.** Click **Create Component**.



The board page should now show the MCP9808 component you created. After the interval you configured elapses, the WipperSnapper firmware running on your board automatically reads values from the sensor and sends them to Adafruit IO.



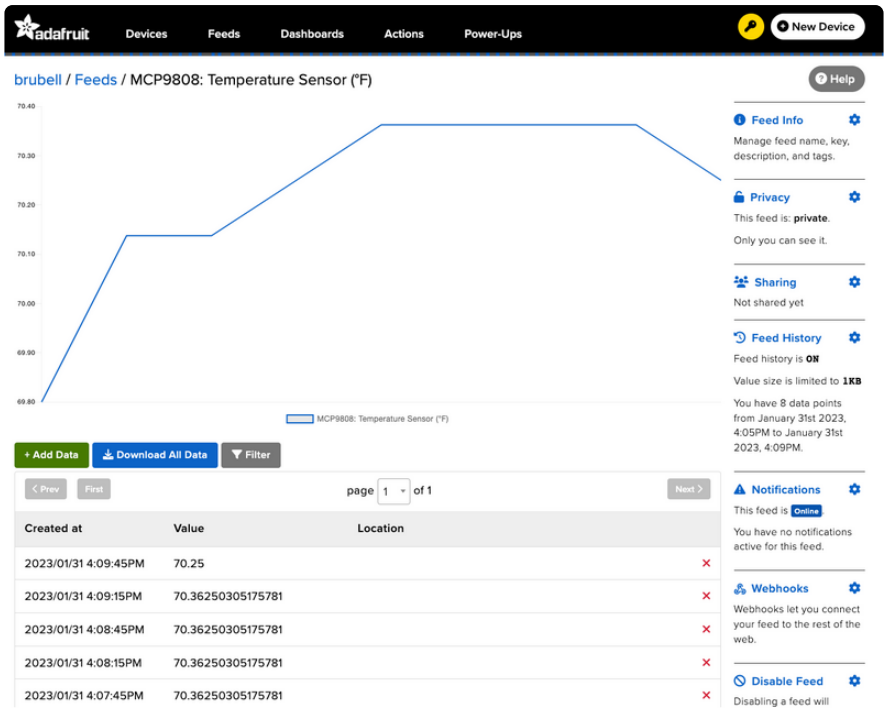
Read I2C Sensor Values

Now to look behind the scenes at a powerful element of using Adafruit IO and WipperSnapper. When a new component is created on Adafruit IO, an Adafruit IO Feed (https://adafru.it/ioA) is also created. This Feed holds your sensor component values for long-term storage (30 days of storage for Adafruit IO Free and 60 days for Adafruit IO Plus plans).

Aside from holding the **values** read by a sensor, the component's feed also holds **metadata** about the data pushed to Adafruit IO. This includes settings for whether the data is public or private, what license the stored sensor data falls under, and a general description of the data.

Next, to look at the sensor temperature feed. To navigate to a component's feed, click on the chart icon in the upper-right-hand corner of the component.

On the component's feed page, you'll each data point read by your sensor and when they were reported to Adafruit IO.



Doing more with your sensor's Adafruit IO Feed

This only scratches the surface of what Adafruit IO Feeds can accomplish for your IoT projects. For a complete overview of Adafruit IO Feeds, including tasks like downloading feed data, sharing a feed, removing erroneous data points from a feed, and more, head over to the "Adafruit IO Basics: Feed" learning guide (https://adafru.it/ioA).

Downloads

Files

- ESP32-S2 product page with resources (https://adafru.it/OpE)
- ESP32-S2 datasheet (https://adafru.it/OpF)
- ESP32-S2 WROVER datasheet (https://adafru.it/Oqa)
- ESP32-S2 Technical Reference (https://adafru.it/Oqb)
- EagleCAD files on GitHub (https://adafru.it/OB2)
- 3D Models on GitHub (https://adafru.it/OCB)
- Fritzing object in the Adafruit Fritzing Library (https://adafru.it/OB3)
- PDF for MagTag PrettyPins Pinout Diagram (https://adafru.it/ZqF)

SVG for MagTag Prettypins Pinout Diagram

https://adafru.it/Zra

All In One Shipping Demo

This file can be burned to your MagTag using the ROM bootloader (address 0x0) to install the TinyUF2 bootloader plus shipping demo

Download the file to your computer, enter ROM bootloader mode, then run after changing the COM/Serial port:

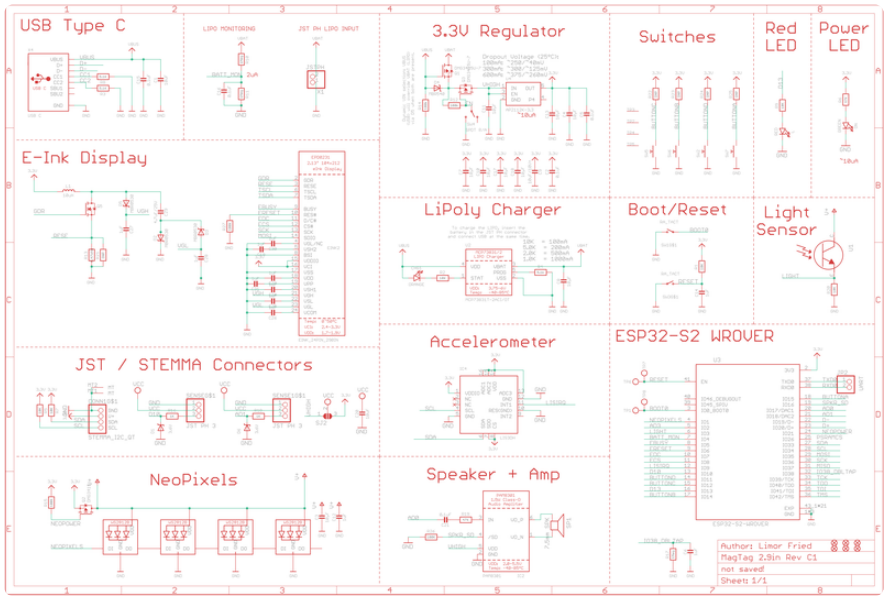
```
esptool.py -p COM88 write_flash 0x0 magtag_demoboot.bin
```

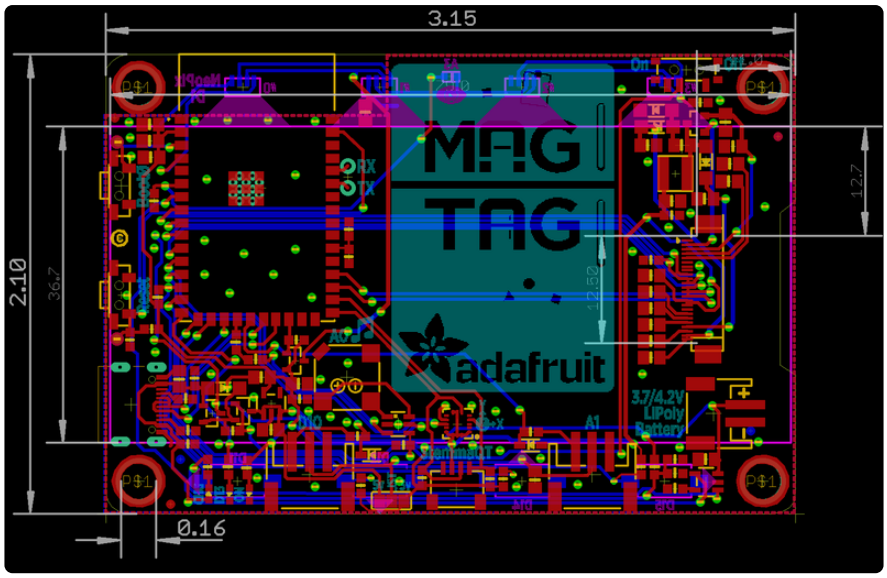
note that esptool will seem to 'hang' for a bit...that's normal, a lot of the file is empty and it doesn't tell you its skipping ahead.

magtag_demoboot.bin

https://adafru.it/OFW

Schematic and Fab Print





Acrylic Front and Back Plates

Suitable for laser cutting. There are two front designs (cloud and arrow) and a back piece (for using MagTag as a wearable badge rather than the magnetic feet). These can be secured with 8mm M3 screws.

Adobe Illustrator or SVG file formats:

MagTag_Acrylic.ai

https://adafru.it/U2a

MagTag_Acrylic.svg

https://adafru.it/U2b