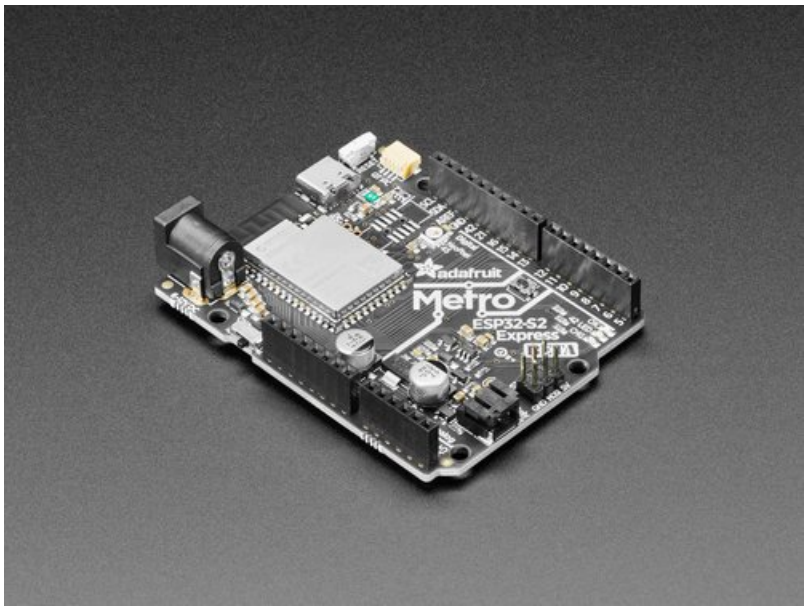
adafruit learning system

Adafruit Metro ESP32-S2

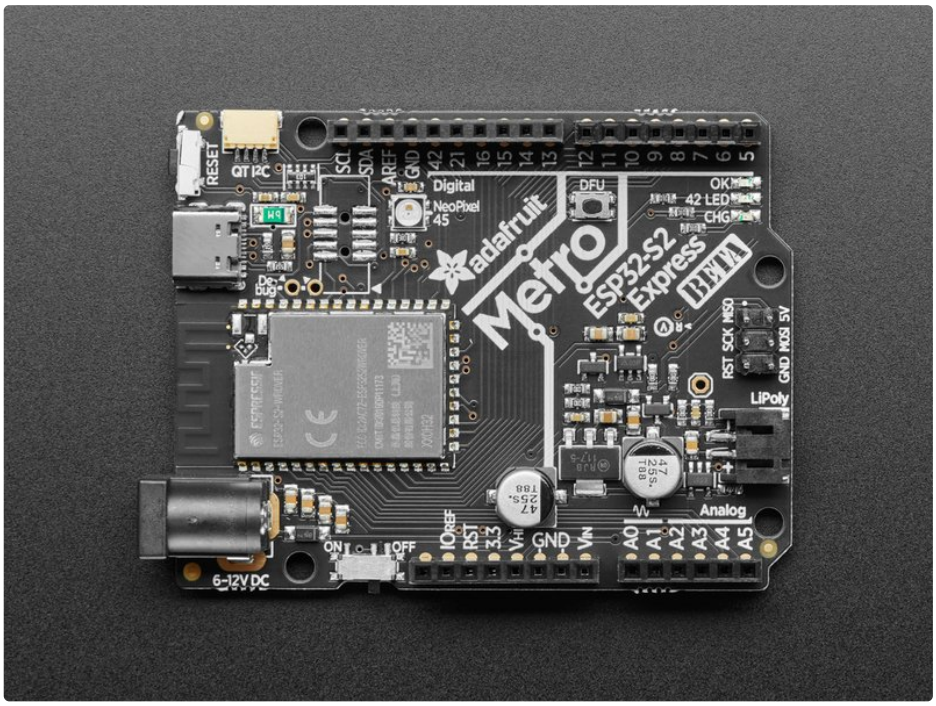
Created by Kattni Rembor



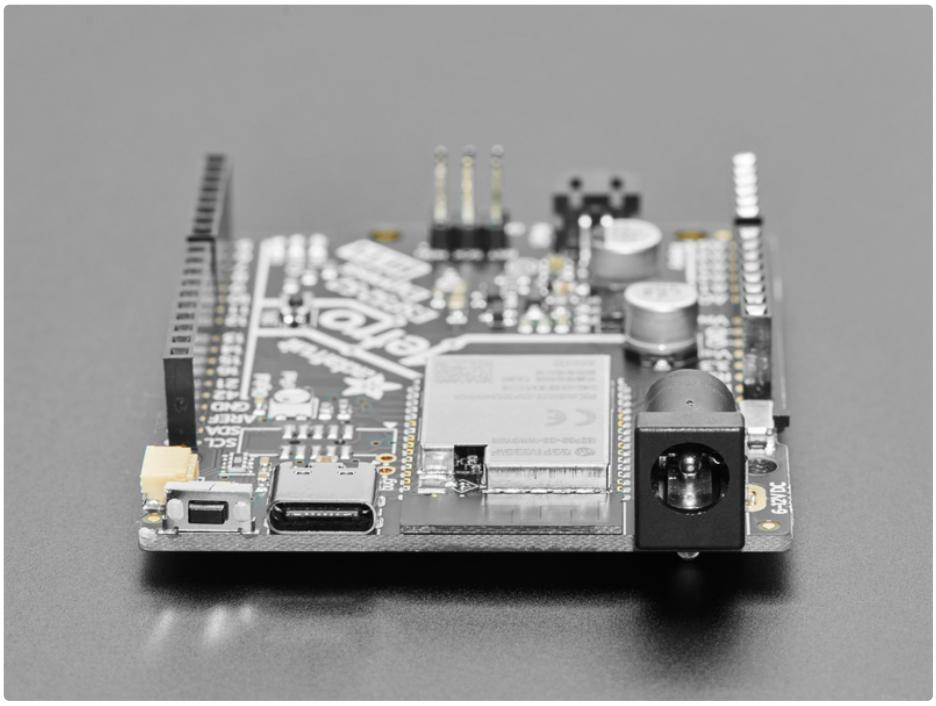
Last updated on 2021-10-22 11:44:52 AM EDT

Guide Contents

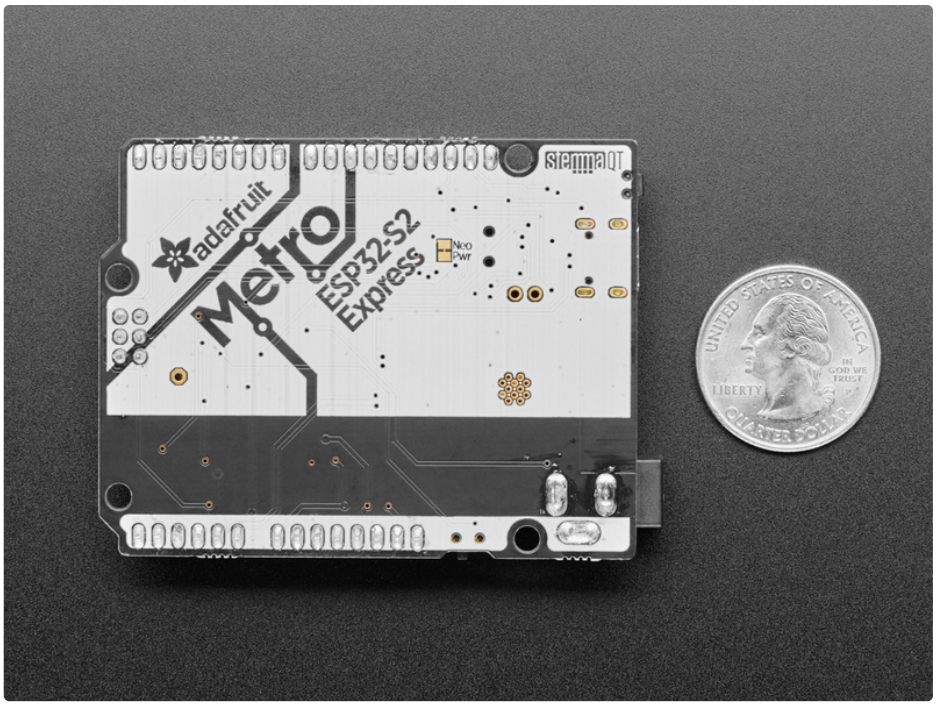
Overview



What's Metro shaped and has an ESP32-S2 WiFi module? What has a STEMMA QT connector for I2C devices, and a Lipoly charger circuit? What's finishing up testing and nearly ready for fabrication? That's right - its the new Adafruit Metro ESP32-S2!

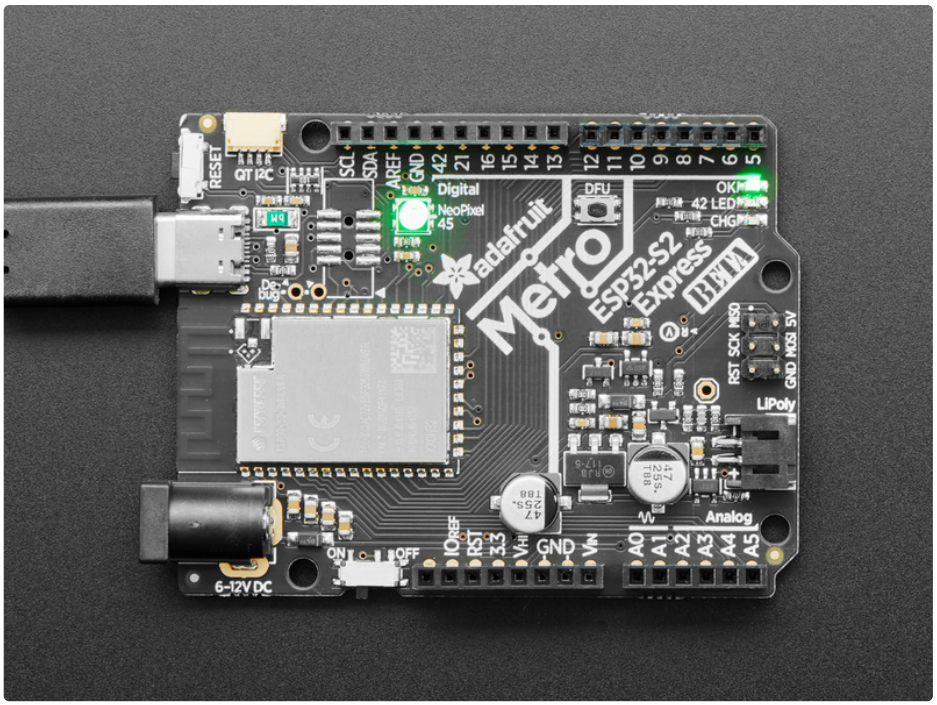


With native USB and a load of PSRAM this board is perfect for use with CircuitPython or Arduino, to add low-cost WiFi while keeping shield-compatibility.

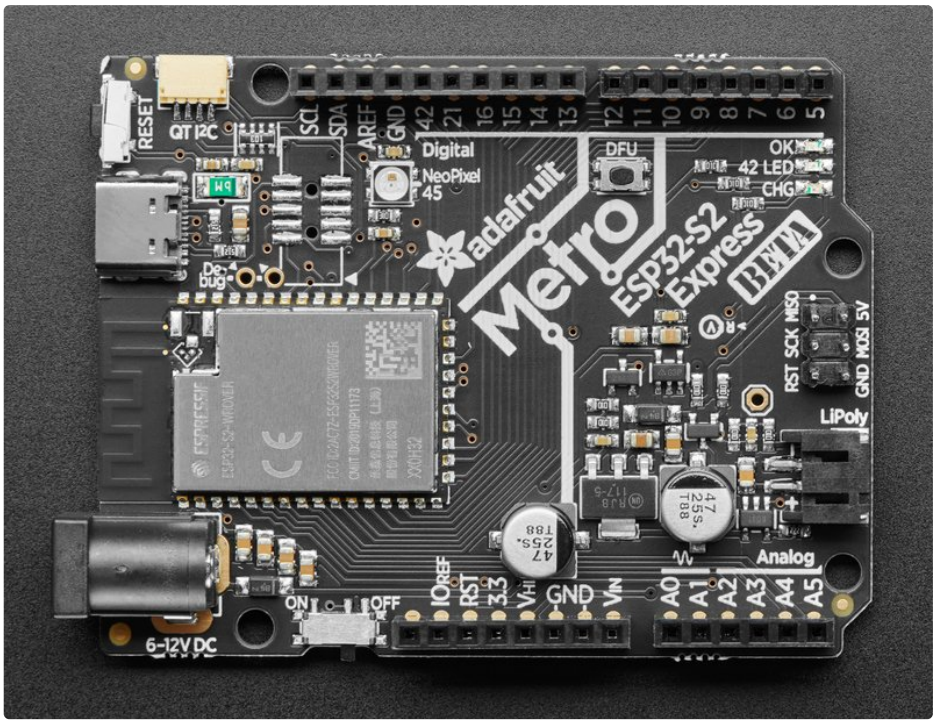


Features:

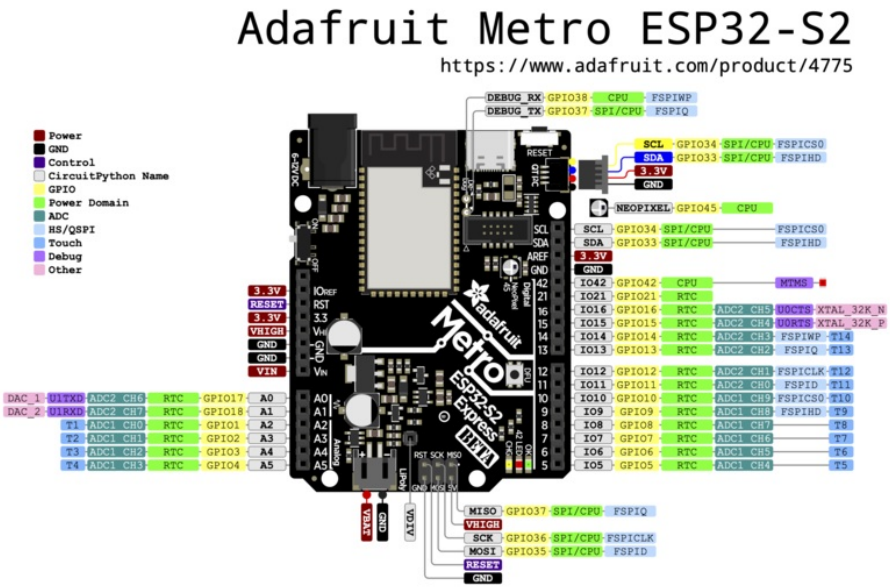
- **ESP32-S2 240MHz Tensilica processor** - the next generation of ESP32, now with native USB so it can act like a keyboard/mouse, MIDI device, disk drive, etc!
- **WROVER module** has FCC/CE certification and comes with 4 MByte of Flash and 2 MByte of PSRAM - you can have huge data buffers
- **Lotsa power options** - 6-12VDC barrel jack **or** USB type C **or** Lipoly battery
- **Built-in battery charging** when powered over DC or USB
- UNO-shape so shields can plug in
- **Reset and DFU** (BOOT0) buttons to get into the ROM bootloader (which is a USB serial port so you don't need a separate cable!)
- **Serial debug pins** (optional, for checking the hardware serial debug console)
- **JTAG pads** for advanced debugging access.
- **On/Off switch**
- **STEMMA QT** connector for I2C devices
- **On/Charge/User** LEDs + status **NeoPixel**
- **Works with Arduino or CircuitPython**
- **53.2mm x 72mm / 2" x 2.8"**
- **Height (w/ barrel jack): 14.8mm / 0.6"**
- **Weight: 22.5g**


RESET
QT I2C
SCL
SDA
AREF
GND
42
21
16
15
14
13
12
11
10
9
8
7
6
5
Digital
NeoPixel
45
DFU
OK
42 LED
CHG
adafruit
Metro
ESP32-S2
Express
BETA
RST
SCK
MISO
GND
MOSI
5V
LiPoly
Analog
A0
A1
A2
A3
A4
A5
IOREF
RST
3.3
VHI
GND
VIN
ON
OFF
6-12V DC

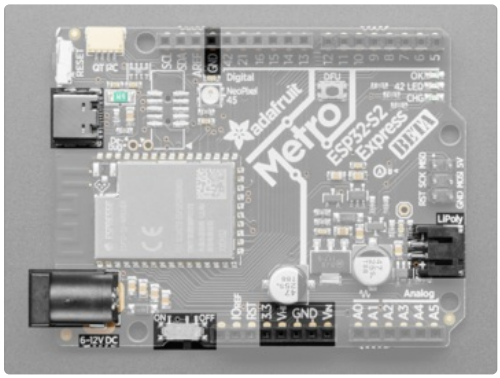
Pinouts



The Metro ESP32-S2 is loaded with all kinds of features. There's lots of pins and ports. Time to take a tour!



Power



There's a lot of power options available on the Metro ESP32-S2, and they're a little different than most Metro/Arduinos

Power Inputs

You have **three power input options:**

- **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!)
- **DC barrel jack** - The DC Jack is a 5.5mm/2.1mm center-positive DC connector, which is the most common available. Provide about 6V-12V here to power the Metro. Great for when you have a wall adapter power supply. Don't use a center-negative adapter, it won't work (the OK green LED will not light)
  When DC power 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!)
  If both DC and USB are plugged in, the metro will power itself & recharge the battery from whichever is highest
- **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 Metro and charge the battery. The battery will charge from the USB or DC power (whichever is plugged in and higher voltage), even if the board is powered off via the switch.
  If the battery is plugged in *and USB or DC is plugged in*, the Metro will power itself from USB or DC *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"!
  When switched off it will disable the 3.3V power which turns off the ESP32S2 and NeoPixel, but **it does *not* turn off the VHI or VIN pins** (see below) and will leave the battery charging.

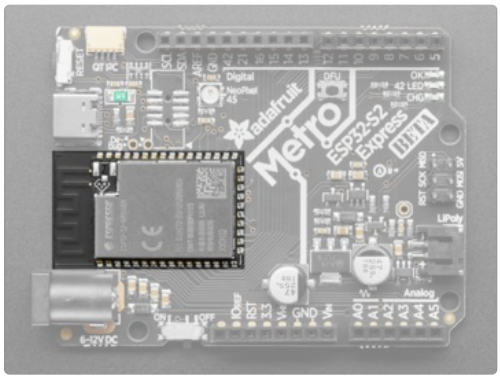
Power Outputs

- **3.3V** - This is the output pin from the 3.3V regulator, you can grab up to 400mA from this regulator for accessories, it's also used by the ESP32S2 which can have spiky current draw.
- **VHI** - This pin is usually marked 5V on Arduinos, and when USB or DC is plugged in, it will in fact provide 5V. However, *if you have the Metro on LiPo battery power, it will be powered from the battery* and thus *between 3.7V to 4.2V*

When powered from USB or DC it is regulated to 5V, when powered from battery only, it's not regulated, but it is high-current, great for driving servos and NeoPixels.

- **GND** - This is the common ground for all power and logic.
- **VIN** - This is the *higher* of the DC jack or USB voltage. So if the DC jack is plugged in and 9V, VIN is 9V. If only USB connected, this will be 5V.

ESP32-S2 WiFi Module



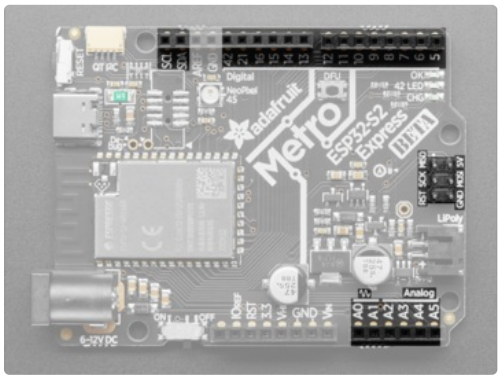
The **ESP32-S2 WROVER module**.

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.

Logic Pins



These are the logic pins that can be used to connect shields, sensors, servos, LEDs and more!

No pins are shared, and no pins are 'special' bootstrapping pins, so you can use any of them for input, or output, will pullups or pulldowns, without worry.

ESP32 chips allow for 'multiplexing' of almost all signals so it isn't like some pins can do PWM and others can. You can connect any of the available PWM channels, I2S channels, UART, I2C or SPI ports to *any* pin. There are some exceptions....

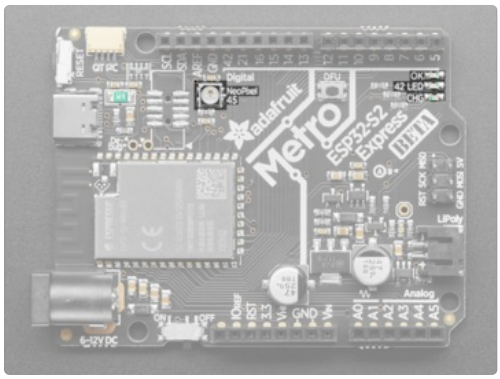
Pin numbers next to pins are the ESP32 IO pin number. E.g. pin 5 is IO5 and 21 is IO21. This is not true for pin names such as A0 thru A5 (these are IO17, IO18, IO1, IO2, IO3, and IO4 in that order), SPI pins (SCK is IO36, MOSI IS IO35 and MISO is IO37) or I2C pins (SDA is IO33 and SCL is IO34)

- **A0 and A1** are the only DAC output pins. These can be used as 8-bit true analog outputs. No other pins can do so.
- **A0 thru A5, IO5 to IO16** - can also be analog inputs. The labeled SPI port, I2C port and pins 21 and 42 cannot.

IO11 thru IO16 and A0 plus A1 are on ADC2
A2 thru A5 plus IO5 thru IO10 are on ADC1
Check the ESP32-S2 datasheet for the ADC channel names for each pin if you need em!

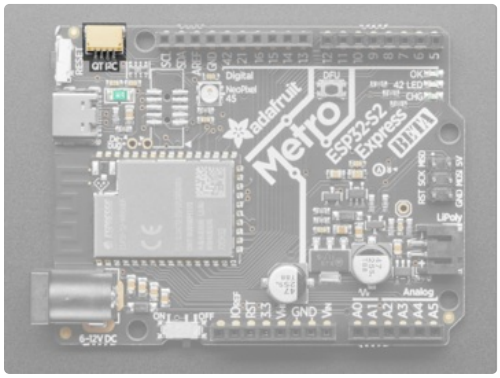
- **The 2x3 SPI pins** on the right side of the board is on the ESP32 high speed SPI peripheral - you can set any pins to be the low-speed peripheral but you won't get the speedy interface!
- **The SDA/SCL I2C pins** have 3.3V pullups on them, and are shared with the STEMMA QT port
- **Pin 42** is connected to a red LED *and* is also shared with the JTAG TMS pin. If you happen to be JTAG debugging, this pin will not be available to you.
- **Pin 45** is connected to the NeoPixel and is a special bootrstap pin but we only use it as an output so it doesn't matter that there's a pullup/down on it.

LEDs and NeoPixel



- **NeoPixel LED** - This addressable RGB NeoPixel LED is both a status LED and user controllable on **IO45**
- **LED** - This red LED on **IO42** is user controllable for blinky needs, it is shared with JTAG TMS and cannot be used if you happen to be JTAG decoding.
- **OK LED** - This green LED indicates the board is powered on, it is connected to the 3.3V power supply.
- **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.

STEMMA QT

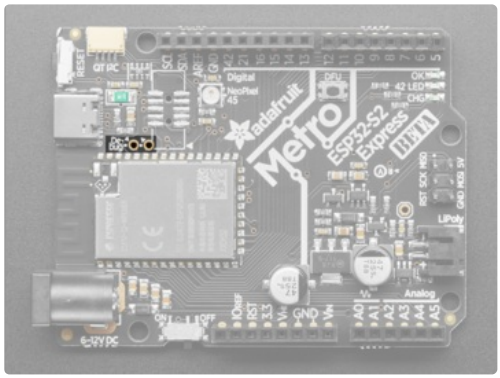


STEMMA QT **(https://adafru.it/Ft4)** - This JST SH 4-pin connector breaks out I2C (SCL, SDA, 3.3V, GND). It allows you to connect to various breakouts and sensors with **STEMMA QT** connectors (https://adafru.it/HMF) or to other things using assorted associated accessories (https://adafru.it/Ft6).

Works great with any STEMMA QT or Qwiic sensor/device

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

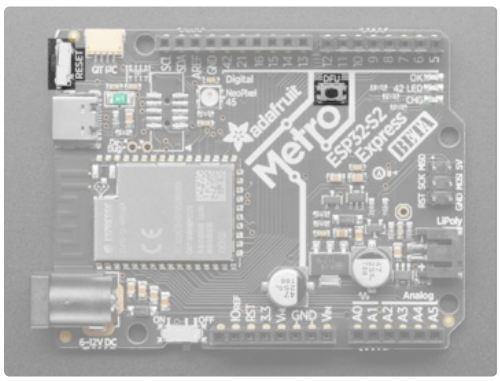
UART Debug



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 (https://adafru.it/dDd). This is useful if you are writing software and need to see the low level debug output without using JTAG debugging.

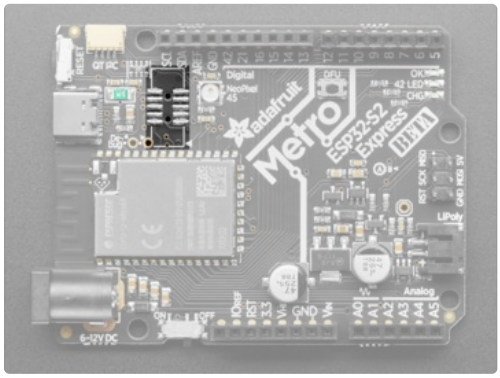
This is *not* where default `Serial.print()` or CircuitPython `print()` outputs go, because those will go through the USB port instead!

Reset and DFU



- **Reset button** - The reset button in the top left corner is used to reset the board.
- **DFU 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 DFU button while clicking reset button mentioned above.** When in the ROM bootloader, you can upload code and query the chip using `esptool`

JTAG Debug



If you'd like to do more advanced development, trace-debugging, or not use the bootloader, we have the JTAG interface exposed. You'll need to solder an 2x5 1.27mm pitch connector (https://adafru.it/w5e) or Mini 2x5 connector (https://adafru.it/Osc) to your board. A JLink or similar is needed to perform debugging.

SEGGER J-Link EDU Mini - JTAG/SWD Debugger

Doing some serious development on any ARM-based platform, and tired of 'printf' plus an LED to debug? A proper JTAG/SWD HW debugger can make debugging more of a pleasure and...

$19.95
In Stock
Add to Cart

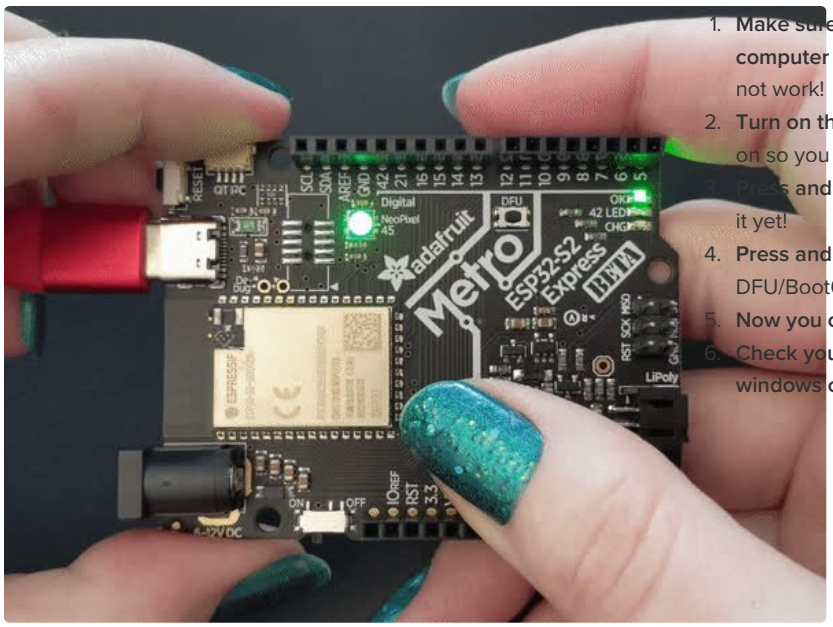
ROM Bootloader

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.

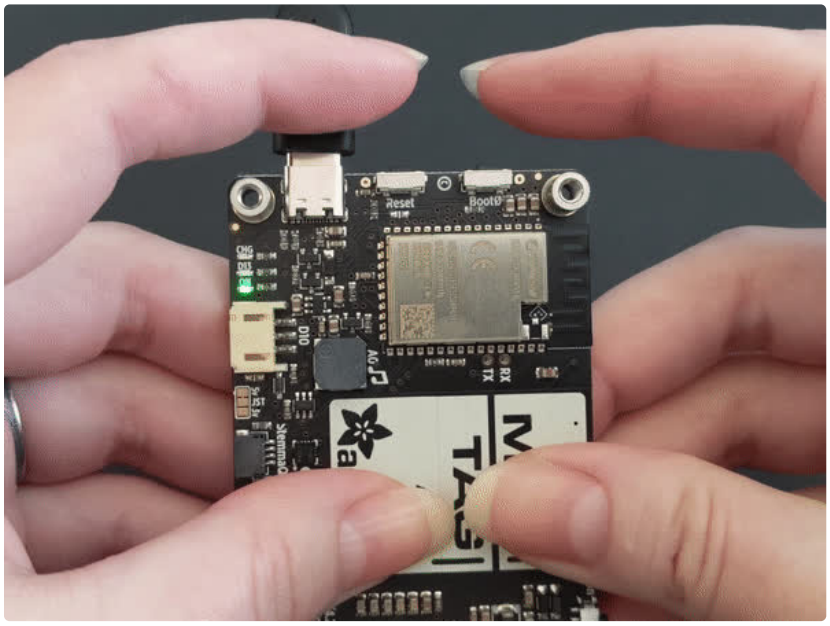
The ROM bootloader can never be disabled or erased, so its always there if you need it!

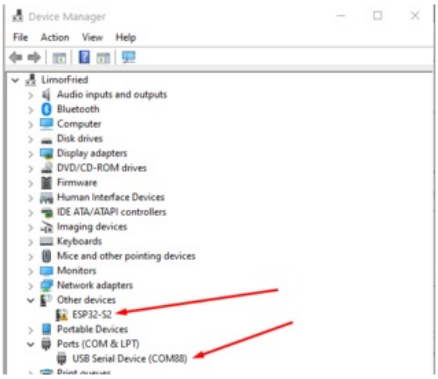
Enter ROM Bootloader Mode

Entering the bootloader is easy. Complete the following steps.



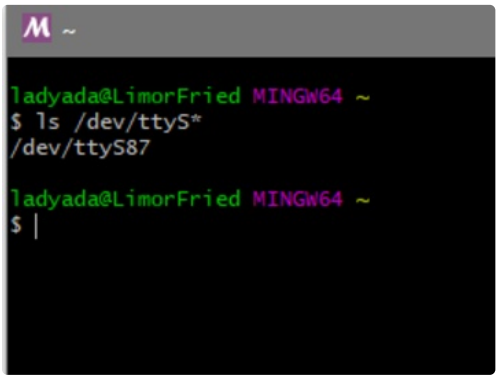
1. **Make sure your ESP32-S2 is plugged into USB port to your computer using a data/sync cable.** Charge-only cables will not work!
2. **Turn on the On/Off switch** - check that you see the OK light on so you know the board is powered, a prerequisite!
3. **Press and hold the DFU / Boot0 button down.** 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) (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:

```
C:\Users\ladyada>python3 C:\ESP32\esp-idf\components\esptool_py\esptool\esptool.py
esptool.py v3.0-dev
usage: esptool [-h] [--chip {auto,esp8266,esp32,esp32s2}] [--port PORT] [--baud BAUD]
               [--before {default_reset,no_reset,no_reset_no_sync}]
               [--after {hard_reset,soft_reset,no_reset}] [--no-stub] [--trace]
               [--override-vddsdio [{1.8V,1.9V,OFF}]] [--connect-attempts CONNECT_ATTEMPTS]
               {load_ram,dump_mem,read_mem,write_mem,write_flash,run,image_info,make_image,elf2image,read
_mac,chip_id,flash_id,read_flash_status,write_flash_status,read_flash,verify_flash,erase_flash,erase_regi
on,version,get_security_info}
               ...
```

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

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

```
C:\Users\ladyada>python3 C:\ESP32\esp-idf\components\esptool_py\esptool\esptool.py --port COM88  chip_id
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...
Warning: ESP32-S2 has no Chip ID. Reading MAC instead.
MAC: 7c:df:a1:00:3f:3e
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'.
```

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!

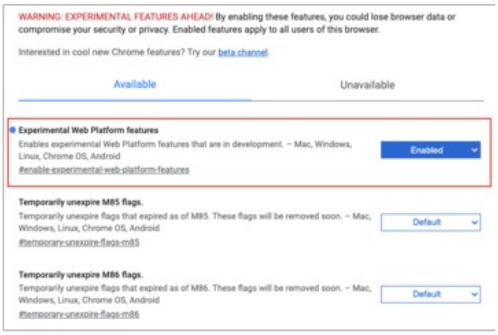
Web Serial ESPTool

The WebSerial ESPTool was designed to be a web-capable option for programming ESP32-S2 boards. It allows you to erase the contents of the microcontroller and program up to 4 files at different offsets.

This tool is a good alternative to 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 browser for this to work, Safari and Firefox, etc are *not* supported because we need Web Serial and only Chrome is supporting it to the level needed.



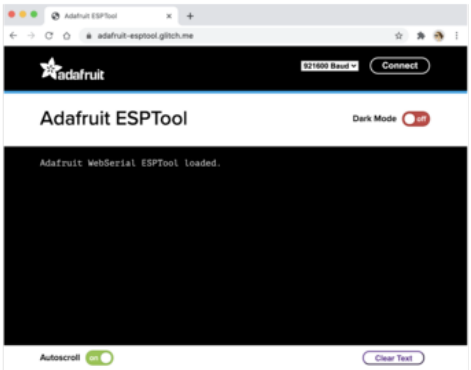
At the time of this tutorial, you'll need to enable the Serial API, which is really easy.

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

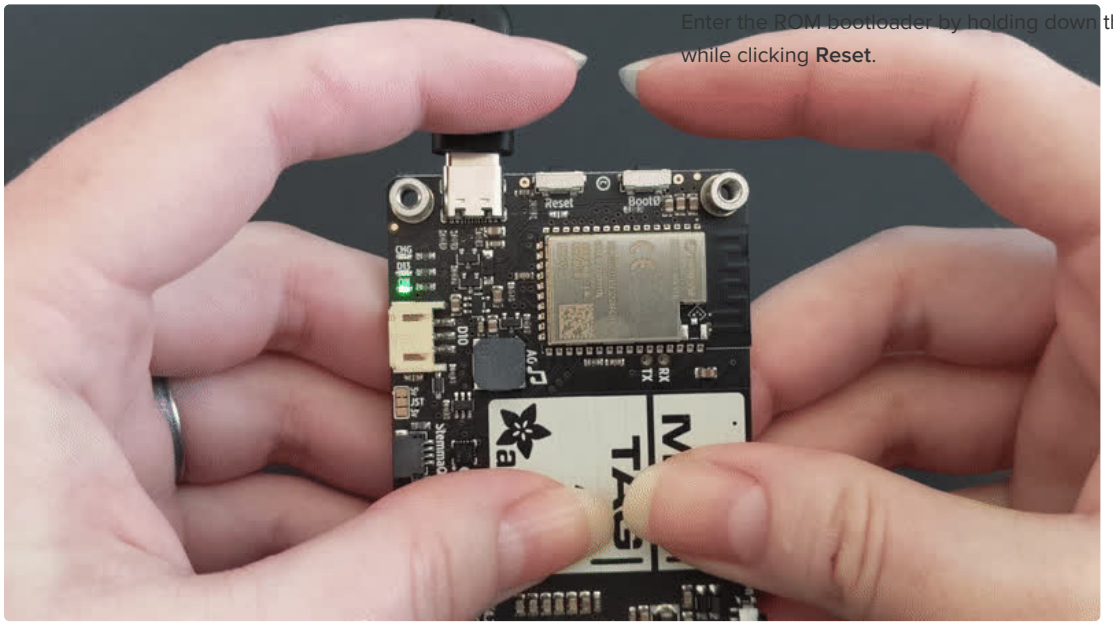
Restart Chrome

Connecting

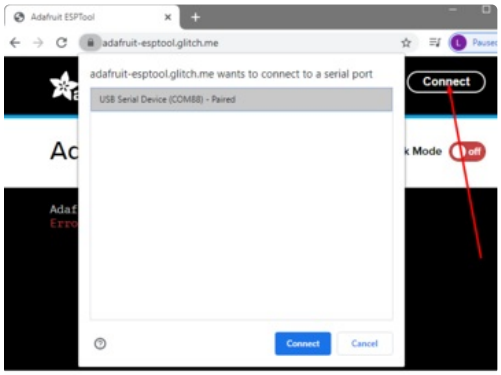
Before you can use the tool, you will need to put your board in bootloader mode and connect. Here are the steps:



In the Chrome browser visit https://adafruit.github.io/Adafruit_WebSerial_ESPTool/ (https://adafru should look like the image to the left



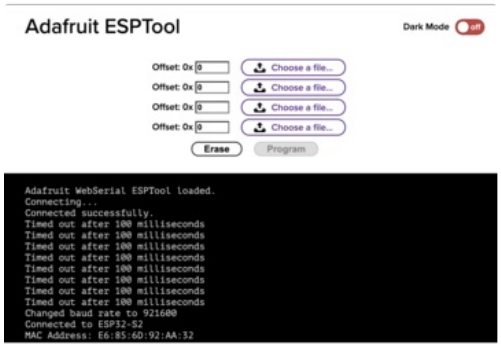
Enter the ROM bootloader by holding down the **BOOT0** button while clicking **Reset**.



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. You may want to remove all other USB devices so *only* the ESP32-S2 board is attached, that way there's no confusion over multiple ports!

```
Adafruit WebSerial ESPTool loaded.
Error: No port selected by the user.
Error: No port selected by the user.
Connecting...
Connected successfully.
Timed out after 100 milliseconds
Timed out after 100 milliseconds
Timed out after 100 milliseconds
Timed out after 100 milliseconds
Timed out after 100 milliseconds
Timed out after 100 milliseconds
Timed out after 100 milliseconds
Timed out after 100 milliseconds
Connected to ESP32-S2
MAC Address: 93:5B:55:BD:F1:E4
```

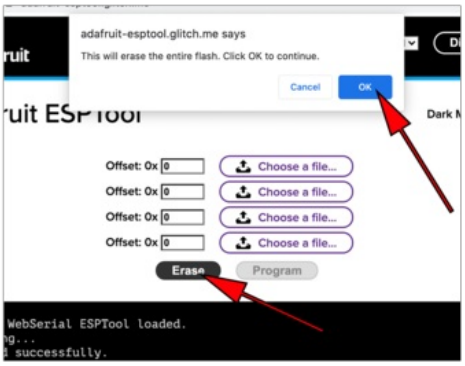
The Javascript code will now try to connect to the ROM bootloader. It may timeout for a bit until it succeeds. On success, you will see that it is **Connected** and will print out a unique **MAC address** identifying the board.



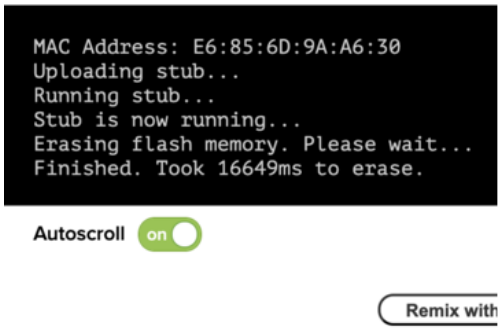
Once you have successfully connected, the command toolbar will appear.

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.



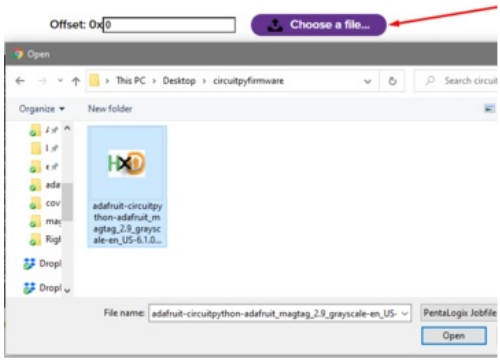
To erase the contents, click the Erase button. You will be prompted whether you want to continue. Click OK to continue or if you changed your mind, just click cancel.



Watch the log messages to see when it has completed. Please do not disconnect until it has finished.

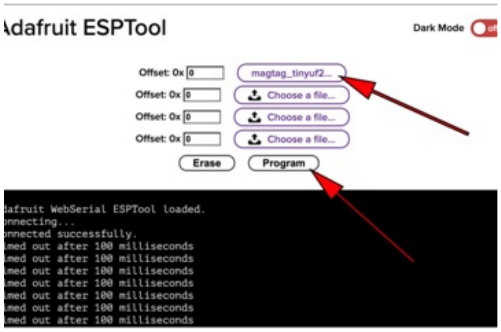
Programming the Microcontroller

Programming the microcontroller can be done with up to 4 files at different locations, but if we use the **tinyuf2combo BIN** file, which is available on the **Install UF2 Bootloader** page, we only need to use 1 file.

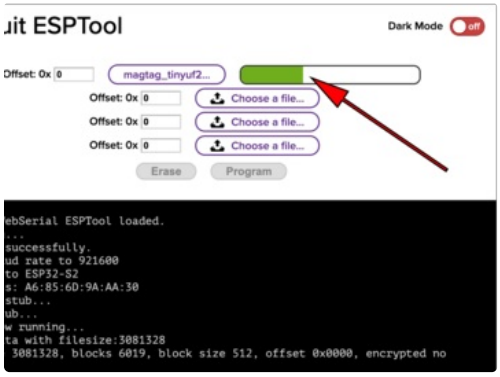


You can click on **Choose a file...** from any of the available buttons. It will only attempt to program buttons with a file and a unique location. Then select the Adafruit CircuitPython **BIN** files (not the UF2 file!)

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. Press the Reset button to get out of the ROM bootloader and you should see a **MAGTAGBOOT** drive (**METROS2BOOT** for the Metro) appear in your computer file explorer/finder.

You're now ready to copy the CircuitPython UF2 on to the drive which will set up CircuitPython!

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

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 MagTag! 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.

https://adafru.it/TSf

https://adafru.it/TSf

https://adafru.it/TSf

https://adafru.it/TSf

https://adafru.it/TSf

https://adafru.it/TSf

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

https://adafru.it/TSA

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

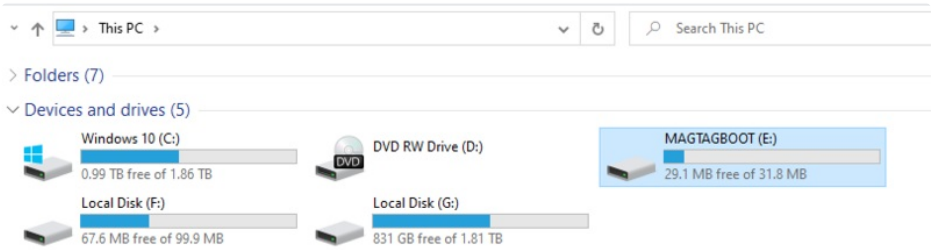
`esptool.py -p COM88 write_flash 0x0 combined.bin`

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'.
```

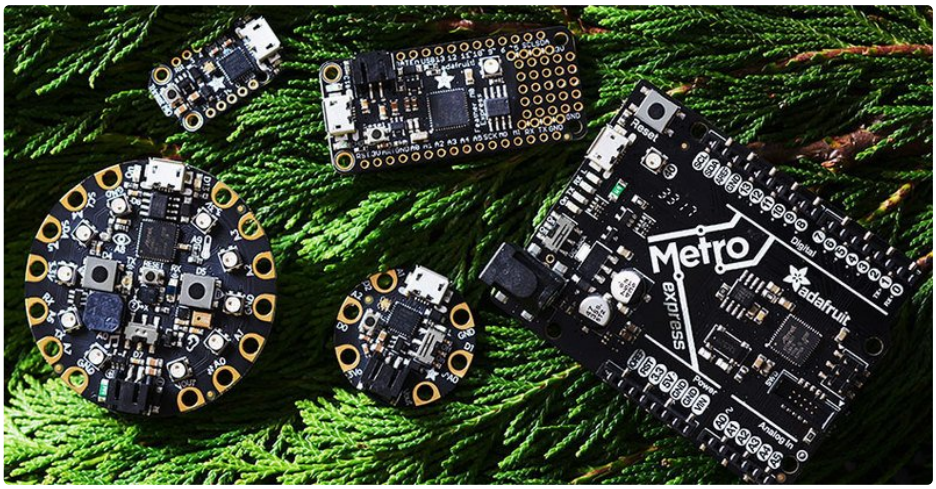
Click RESET button to launch the bootloader. You'll see a new disk drive on your computer with the name **MAGTAGBOOT** or similar



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) (https://adafru.it/Pdq) in this guide. This tool uses Web Serial to erase or upload firmware to your board.

Welcome To CircuitPython



So, you've got this 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 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!).

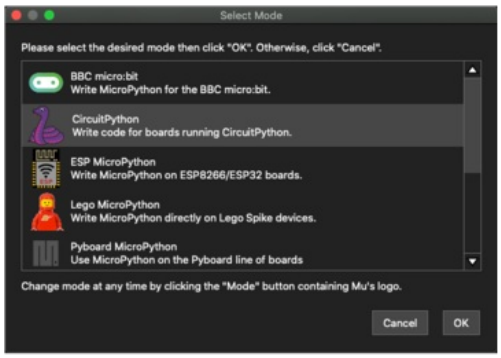
Download and Install Mu

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

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

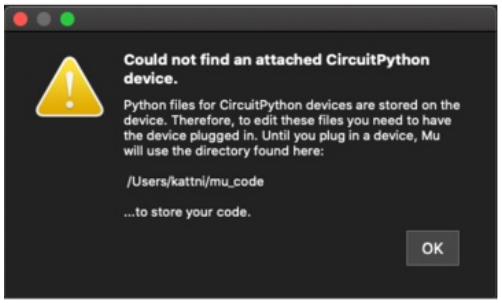
Click **Start Here** to find a wealth of other information, including extensive tutorials and and how-to's.

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...

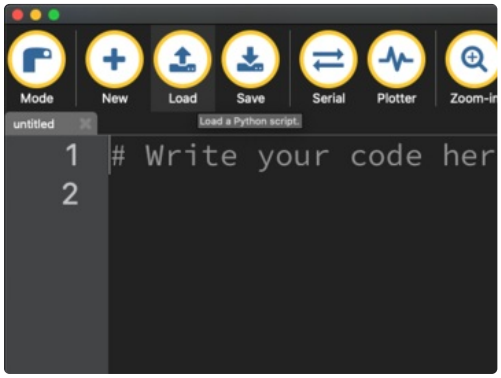
Creating and Editing Code

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 is not a problem on MacOS.)

Creating Code



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

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

Copy and paste the following code into your editor:

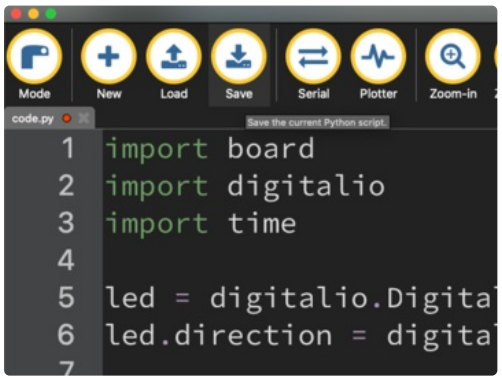
```
import board
import digitalio
import time

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

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

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

If you're using QT Py or a Trinkey, 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.



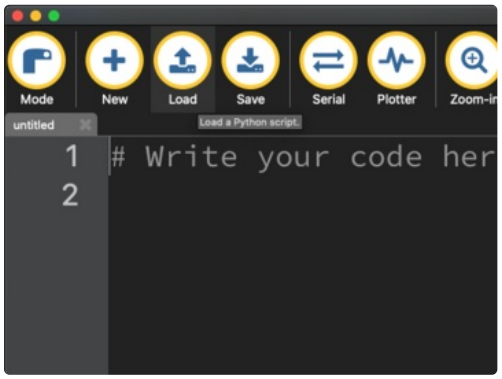
Save the **code.py** file on your **CIRCUITPY** drive.

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, 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.

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 looks like this:

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

This line would result in:

```
Hello, world!
```

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

The serial console receives output from your CircuitPython board sent over USB and displays it so you can see it. This is necessary when you've included a print statement in your code and you'd like to see what you printed. It is also helpful for troubleshooting errors, because your board will send errors and the serial console will print those too.

The serial console requires a terminal program. A terminal is a program that gives you a text-based interface to perform various tasks.

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

```
sudo apt purge modemmanager
```

Are you using Mu?

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

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

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

Setting Permissions on Linux

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

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

```
sudo adduser $USER dialout
```

After running the command above, reboot your machine to gain access to the group. On other Linux distributions, the group you need may be different. See Advanced Serial Console on Mac and Linux (https://adafru.it/AAI) 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 as a separate program.

Windows requires you to download a terminal program, check out this page for more details (https://adafru.it/AAH)

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

Interacting with the Serial Console

Once you've successfully connected to the serial console, it's time to start using it.

The code you wrote earlier has no output to the serial console. So, we're going to edit it to create some output.

Open your code.py file into your editor, and include a `print` statement. You can print anything you like! Just include your phrase between the quotation marks inside the parentheses. For example:

```
import board
import digitalio
import time

led = digitalio.DigitalInOut(board.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.

Keep your serial console window where you can see it. Save your file. You'll see what the serial console displays when the board reboots. Then you'll see your new change!

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

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

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

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

```
code.py
import board
import digitalio
import time

led = digitalio.DigitalInOut(board.D13)
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. We need to fix it!

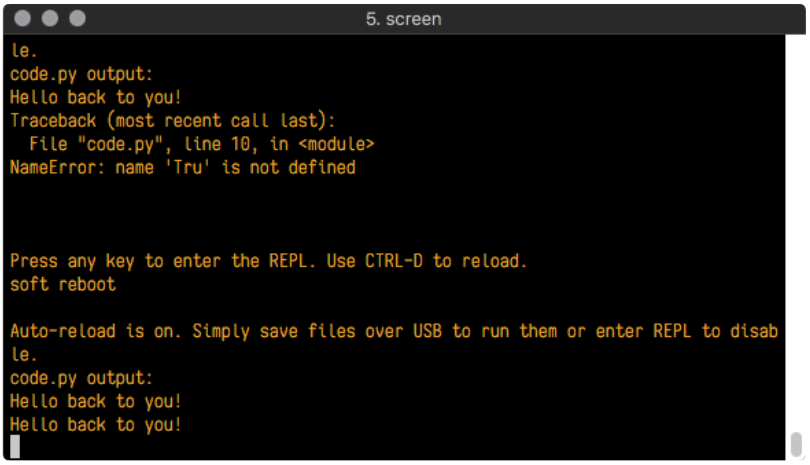
Usually when you run into errors, it's not because you introduced them on purpose. You may have 200 lines of code, and have no idea where your error could be hiding. This is where the serial console can help. Let's take a look!

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

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



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

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. 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.

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

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

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

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



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

If there is no code running, you will enter the REPL immediately after pressing Ctrl + C. There is no information about what your board was doing before you interrupted it because there is no code running.

```
2. screen
Auto-reload is on. Simply save files over USB to run them or enter REPL to disab
le.

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

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

```
2. screen

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

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

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

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

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

This is followed by the CircuitPython prompt.

```
>>>
```

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

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

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

Then press enter. You should then see a message.

```
2. screen
Auto-reload is on. Simply save files over USB to run them or enter REPL to disab
le.

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

Adafruit CircuitPython 2.1.0 on 2017-10-17; Adafruit CircuitPlayground Express w
ith samd21g18
>>> help()
Welcome to Adafruit CircuitPython 2.1.0!

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

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

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

```
3. screen
Adafruit CircuitPython 2.1.0 on 2017-10-17; Adafruit Feather M0 Express with sam
d21g18
>>> help()
Welcome to Adafruit CircuitPython 2.1.0!

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

To list built-in modules please do `help("modules")`.
>>> help("modules")
__main__          busio             neopixel_write    time
analogio          digitalio         nvm               touchio
array             framebuf          os                ucollections
audiobusio        gamepad           pulseio           ure
audioio           gc                random            usb_hid
bitbangio         math              samd              ustruct
board             microcontroller   storage
builtins          micropython       sys
Plus any modules on the filesystem
>>>
```

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

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

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

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

```
>>> dir(board)
['__class__', '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']
>>>
```

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." We're going to say hello to something else. Type into the REPL:

`print("Hello, CircuitPython!")`

Then press enter.

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

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

You can write single lines of code that run stand-alone. You can also write entire programs into the REPL to test them. As we said though, remember that nothing typed into the REPL is saved.

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

Try typing more into the REPL to see what happens!

Returning to the serial console

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

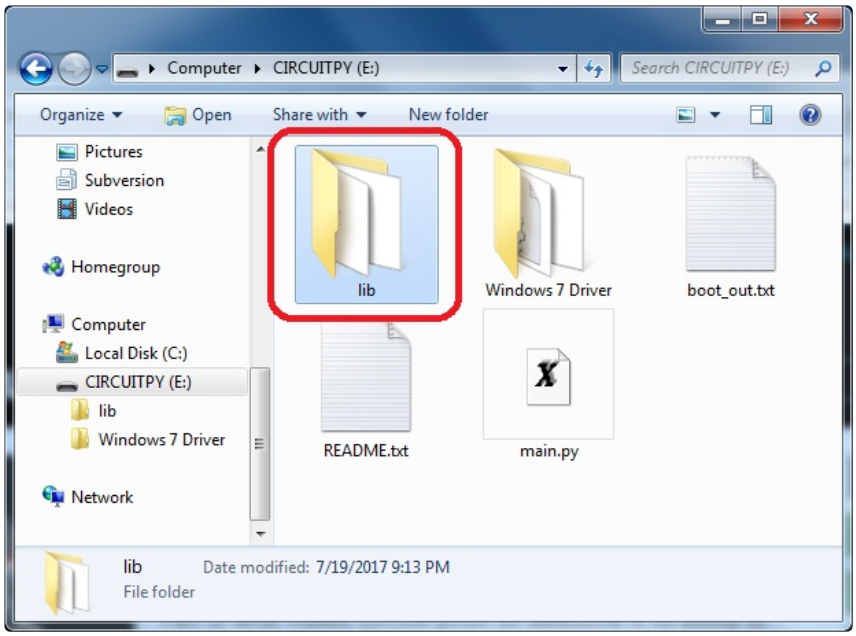
You can return to the REPL at any time!

CircuitPython Libraries

As 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 awesome is its ability to store code separately from the firmware itself. Storing code separately from the firmware makes it easier to update both the code you write and the libraries on which 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 a great reference for how it all should work. In Python terms, we 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, we provide a bundle full of our libraries.

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

Installing the CircuitPython Library Bundle

We're constantly updating and improving our libraries, so we don't (at this time) ship our CircuitPython boards with the full

library bundle. Instead, you can find example code in the guides for your board that depends on external libraries. Some of these libraries may be available from us at Adafruit, some may be written by community members!

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

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

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

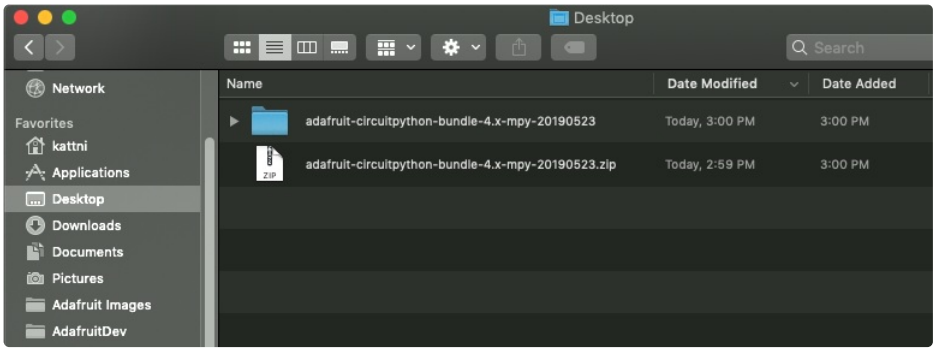
https://adafru.it/ENC

https://adafru.it/ENC

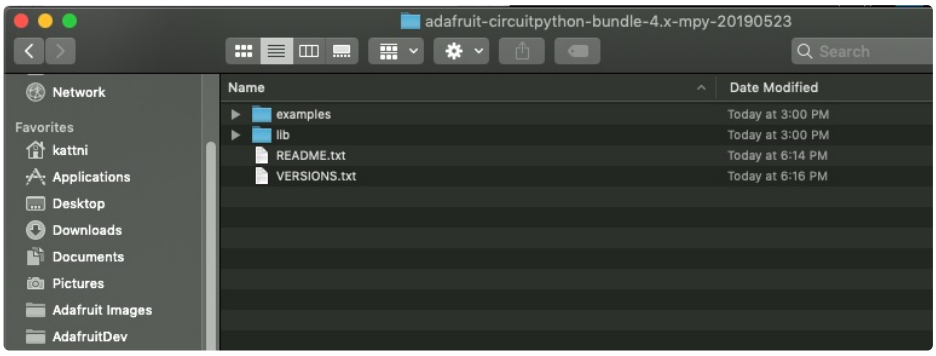
If you need another version, you can also visit the bundle release page (https://adafru.it/Ayy) which will let you select exactly what version you're looking for, as well as information about changes.

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

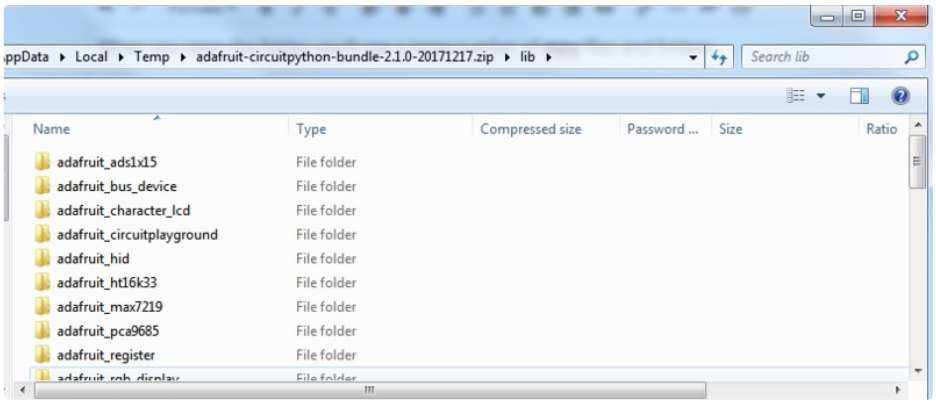
After downloading the zip, extract its contents. This is usually done by double clicking on the zip. On Mac OSX, it places the file in the same directory as the zip.



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



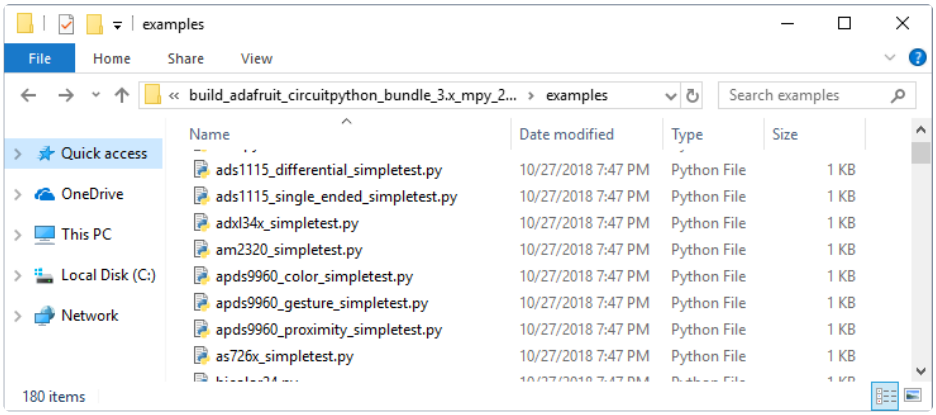
Now open the lib folder. When you open the folder, you'll see a large number of **mpy** files and folders



Example Files

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

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



Copying Libraries to Your Board

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

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

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

Example: `ImportError` Due to Missing Library

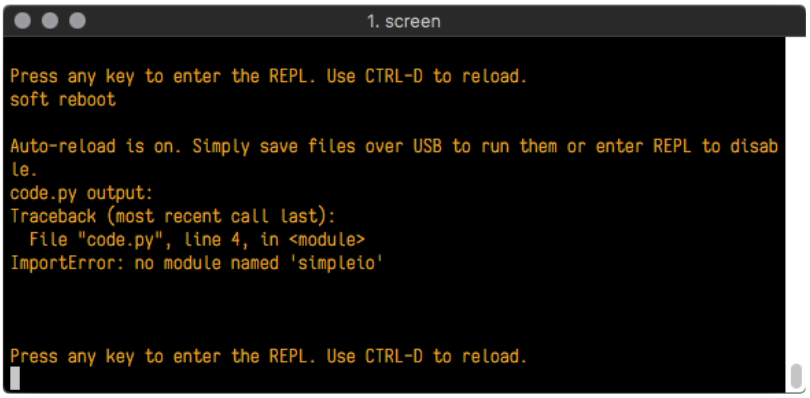
If you choose to load libraries as you need them, you may write up code that tries to use a library you haven't yet loaded. We're going to demonstrate what happens when you try to utilise a library that you don't have loaded on your board, and cover the steps required to resolve the issue.

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

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

```
import board
import time
import simpleio

led = simpleio.DigitalOut(board.LED)

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

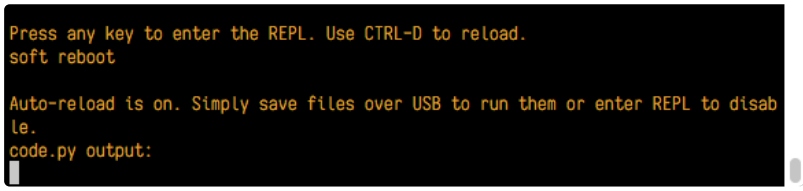
Save this file. Nothing happens to your board. Let's check the serial console to see what's going on.



We have an `ImportError`. It says there is `no module named 'simpleio'`. That's the one we just included in our code!

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

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



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

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

Library Install on Non-Express Boards

If you have a Trinket M0 or Gemma M0, you'll want to follow the same steps in the example above to install libraries as you need them. You don't always need to wait for an `ImportError` as you probably know what library you added to your code. Simply open the **lib** folder you downloaded, find the library you need, and drag it to the **lib** folder on your **CIRCUITPY** drive.

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

Updating CircuitPython Libraries/Examples

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

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

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

Advanced Serial Console on Windows

Windows 7 and 8.1

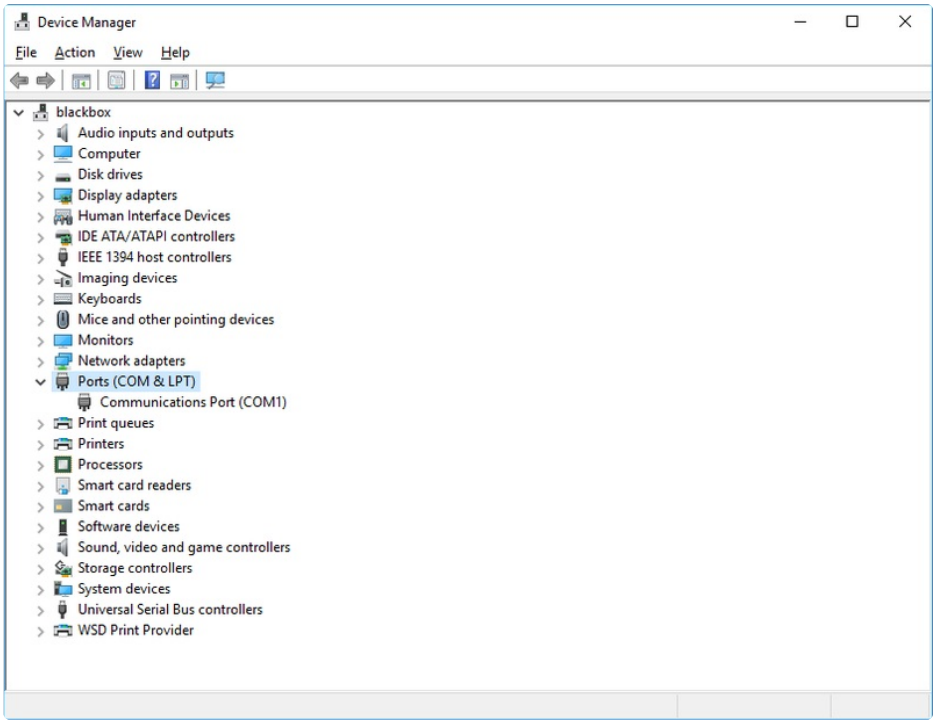
If you're using Windows 7 (or 8 or 8.1), you'll need to install drivers. See the [Windows 7 and 8.1 Drivers page](https://adafru.it/VuB) (https://adafru.it/VuB) for details. You will not need to install drivers on Mac, Linux or Windows 10.

We *strongly* encourage you 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) (https://adafru.it/RWc).

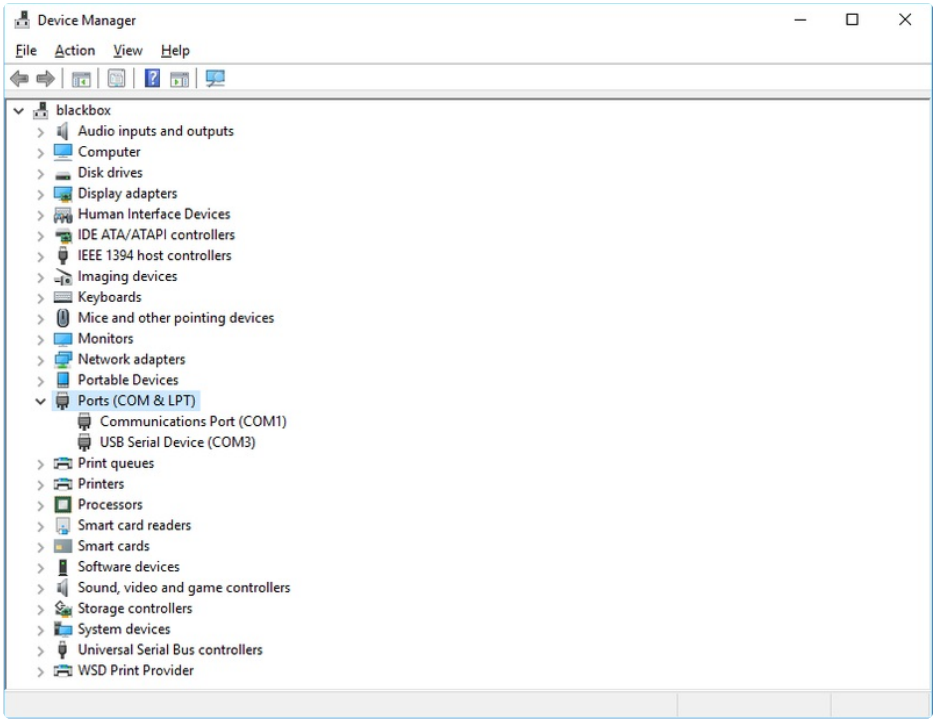
What's the COM?

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.

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



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



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

Install Putty

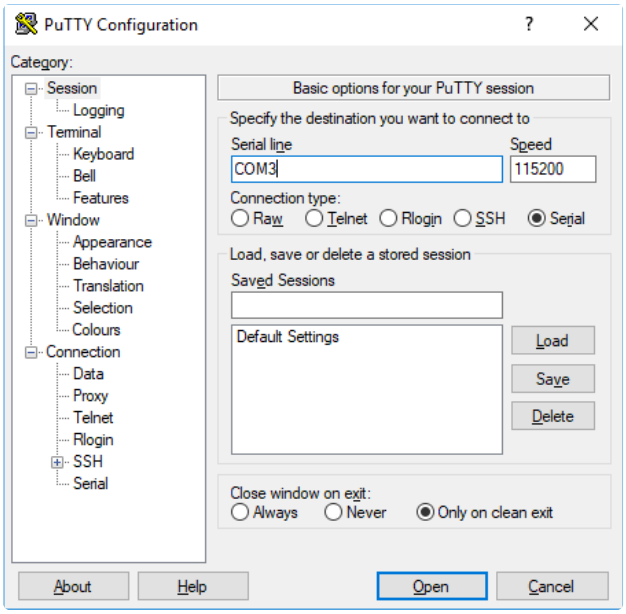
If you're using Windows, you'll need to download a terminal program. We're going to use PuTTY.

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

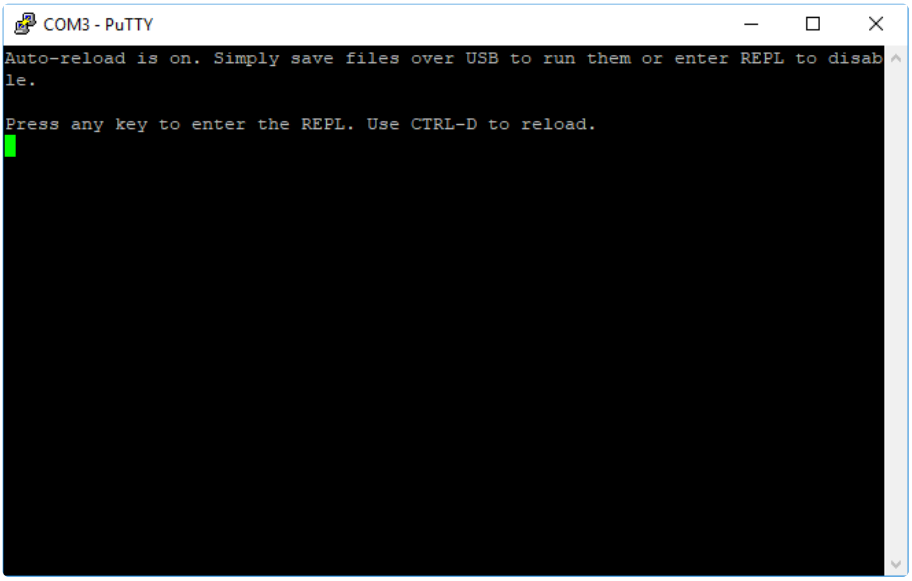
Now you need to open PuTTY.

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

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



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



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

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

Advanced Serial Console on Mac

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.

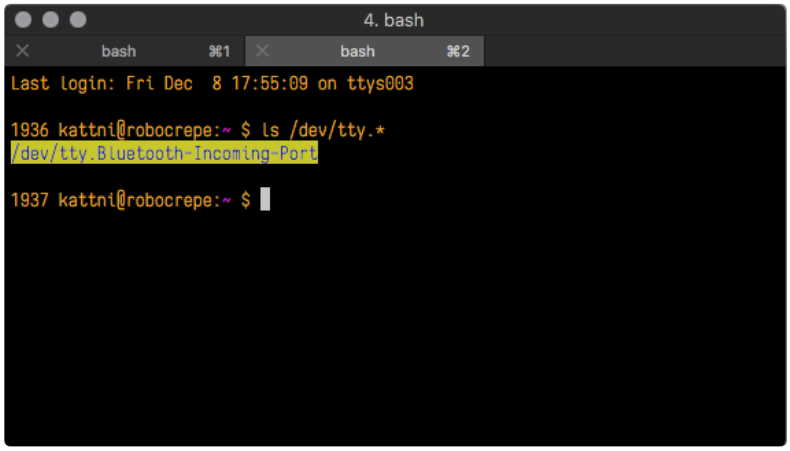
What's the Port?

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.*
```

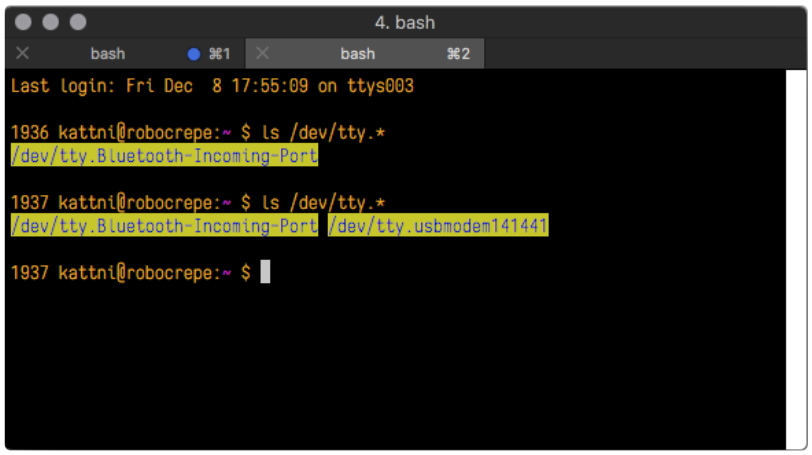
Each serial connection shows up in the `/dev/` directory. It has a name that starts with `tty.`. The command `ls` shows you a list of items in a directory. You can use `*` as a wildcard, to search for files that start with the same letters but end in something different. In this case, we're asking to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



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.

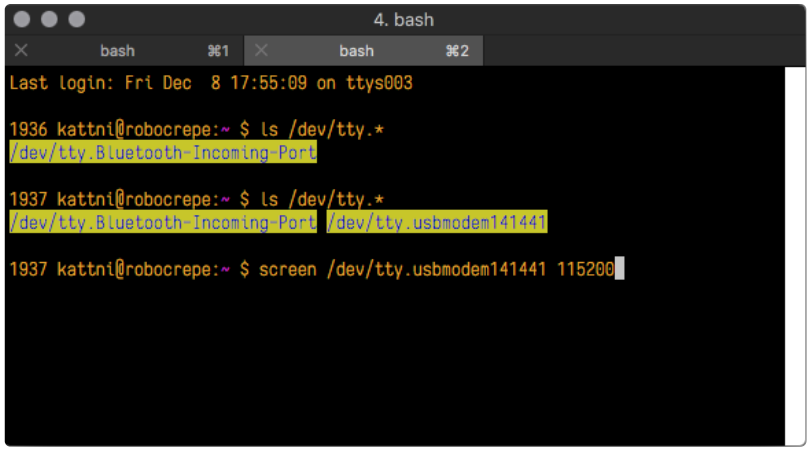
Using Linux, a new listing has appeared called `/dev/ttyACM0`. The `ttyACM0` part of this listing is the name the example board is using. Yours will be called something similar.

Connect with screen

Now that you know the name your board is using, you're ready connect to the serial console. We'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.

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

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

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

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

Is ESP8266 or ESP32 supported in CircuitPython? Why not?

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

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

We do not support ESP32 because it does not have native USB.

We do support ESP32-S2, which has native USB.

How do I connect to the Internet with CircuitPython?

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

Is there asyncio support in CircuitPython?

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

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

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

What is a `MemoryError`?

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

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

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

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

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

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

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

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

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

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

You can download `mpy-cross` for your operating system from https://adafruit-circuit-python.s3.amazonaws.com/index.html?prefix=bin/mpy-cross/ (https://adafru.it/QDK). 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.

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

How do I check how much memory I have free?

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

Will give you the number of bytes available for use.

Does CircuitPython support interrupts?

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

Does Feather M0 support WINC1500?

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

Can AVRs such as ATmega328 or ATmega2560 run CircuitPython?

No.

Commonly Used Acronyms

CP or CPy = CircuitPython (https://adafru.it/cpy-welcome)
CPC = Circuit Playground Classic (https://adafru.it/ncE)
CPX = Circuit Playground Express (https://adafru.it/wpF)

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/PEk).

Adafruit considers CircuitPython for the ESP32-S2 to be beta quality software.

Cannot reinitialize certain peripherals (especially busio.I2C)

If you create a `busio.I2C` object, call `.deinit()` on it, and then create another one, CircuitPython will lock up.

Workaround: Do not deinitialize I2C objects, except by soft reload or entering deep sleep.

No DAC-based audio output

Current versions of esp-idf 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 Adafruit I2S Stereo Decoder - UDA1334A Breakout (https://adafru.it/PEI).

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

Always Run the Latest Version of CircuitPython and Libraries

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

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

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

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

We are no longer building or supporting the CircuitPython 5.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)

CPLAYBOOT, TRINKETBOOT, FEATHERBOOT, or GEMMABOOT Drive Not Present

You may have a different board.

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

MakeCode

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

MacOS

DriveDx and its 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

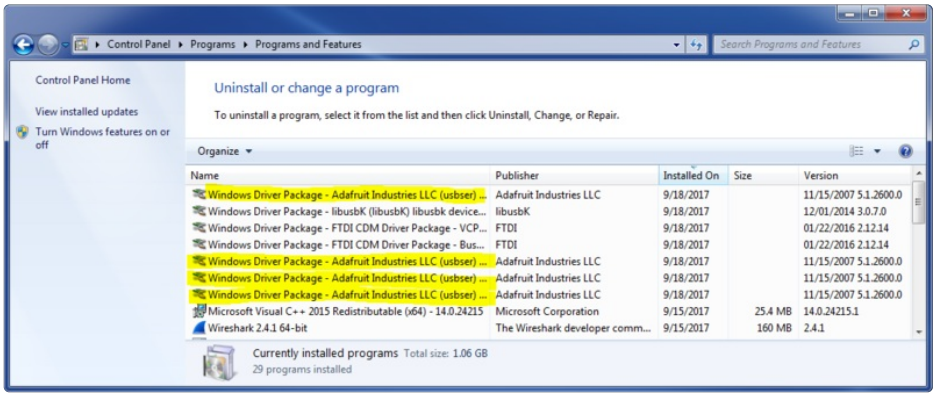
Did you install the Adafruit Windows Drivers package by mistake, or did you upgrade to Windows 10 with the driver package installed? You don't need to install this package on Windows 10 for most Adafruit boards. The old version (v1.5) can interfere with recognizing your device. Go to **Settings** ->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

Windows 7 or 8.1

Version 2.5.0.0 or later of the Adafruit Windows Drivers will fix the missing `boardnameBOOT` drive problem on Windows 7 and 8.1. To resolve this, first uninstall the old versions of the drivers:

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

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

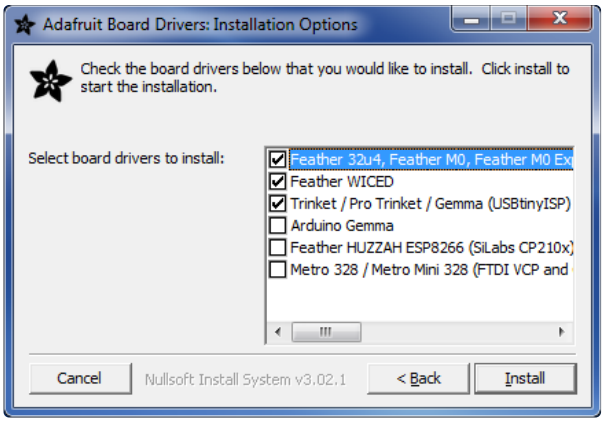


- Now install the new 2.5.0.0 (or higher) Adafruit Windows Drivers Package:

https://adafru.it/AB0

https://adafru.it/AB0

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



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

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

Windows Explorer Locks Up When Accessing `boardnameBOOT` Drive

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

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

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

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

CIRCUITPY Drive Does Not Appear

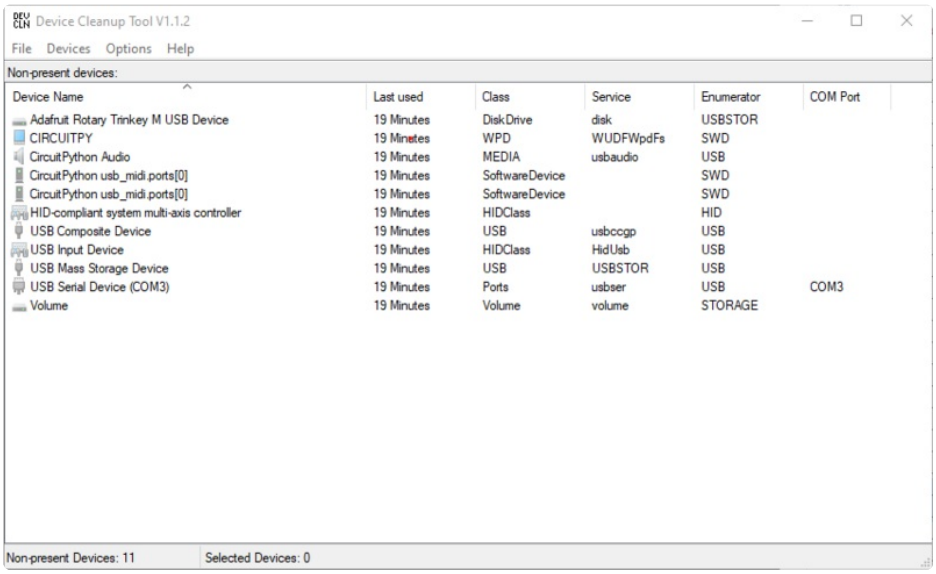
Kaspersky anti-virus can block the appearance of the `CIRCUITPY` drive. We haven't yet figured out a settings change that prevents this. Complete uninstallation of Kaspersky fixes the problem.

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

Device Errors or Problems on Windows

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

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



Device Name	Last used	Class	Service	Enumerator	COM Port
Adafruit Rotary Trinkey M USB Device	19 Minutes	DiskDrive	disk	USBSTOR	
CIRCUITPY	19 Minutes	WPD	WUDFWpdFs	SWD	
CircuitPython Audio	19 Minutes	MEDIA	usbaudio	USB	
CircuitPython usb_midi.ports[0]	19 Minutes	SoftwareDevice		SWD	
CircuitPython usb_midi.ports[0]	19 Minutes	SoftwareDevice		SWD	
HID-compliant system multi-axis controller	19 Minutes	HIDClass		HID	
USB Composite Device	19 Minutes	USB	usbccgp	USB	
USB Input Device	19 Minutes	HIDClass	HidUsb	USB	
USB Mass Storage Device	19 Minutes	USB	USBSTOR	USB	
USB Serial Device (COM3)	19 Minutes	Ports	usbser	USB	COM3
Volume	19 Minutes	Volume	volume	STORAGE	

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

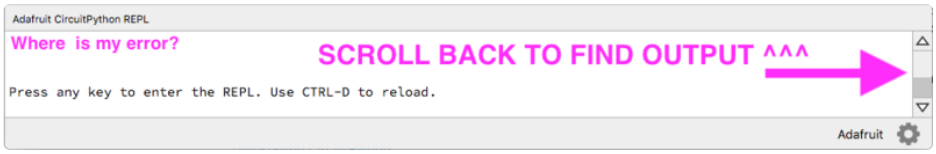
There are times when the serial console will accurately not display anything, such as, when no code is currently running, or when code with no serial output is already running before you open the console. However, if you find yourself in a situation where you feel it should be displaying something like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

CircuitPython RGB Status Light

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

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

CircuitPython 7.0.0 and Later

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

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

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

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

When entering the REPL, CircuitPython will set the status LED to **WHITE**. You can change the status 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.

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`. They are all 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.

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

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

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

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

Easiest Way: Use `storage.erase_filesystem()`

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

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

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

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

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

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

1. Download the correct erase file:

https://adafru.it/AdI

https://adafru.it/AdI

https://adafru.it/AdJ

https://adafru.it/AdJ

https://adafru.it/EVK

https://adafru.it/EVK

https://adafru.it/AdK

https://adafru.it/AdK

https://adafru.it/EoM

https://adafru.it/EoM

https://adafru.it/DjD

https://adafru.it/DjD

https://adafru.it/DBA

https://adafru.it/DBA

https://adafru.it/Eca

https://adafru.it/Eca

https://adafru.it/Gnc

https://adafru.it/Gnc

https://adafru.it/GAN

https://adafru.it/GAN

https://adafru.it/GAO

https://adafru.it/GAO

https://adafru.it/Jat

https://adafru.it/Jat

https://adafru.it/Q5B

https://adafru.it/Q5B

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

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

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

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

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

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

1. Download the erase file:

https://adafru.it/AdL

https://adafru.it/AdL

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

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

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

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

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

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

Running Out of File Space on Non-Express Boards

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

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

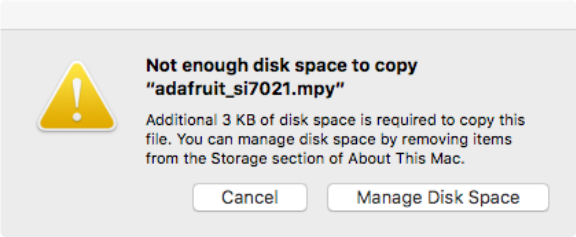
Delete something!

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

Use tabs

One unique feature of Python is that the indentation of code matters. Usually the recommendation is to indent code with four spaces for every indent. In general, we recommend that too. **However**, one trick to storing more human-readable code is to use a single tab character for indentation. This approach uses 1/4 of the space for indentation and can be significant when we're counting bytes.

MacOS loves to add extra files.



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

Prevent & Remove MacOS Hidden Files

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

```
ls -l /Volumes
```

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

Now follow the steps from this question (https://adafru.it/u1c) 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 `foo.mpy` file to the board use a command like:

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

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

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

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

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

Other MacOS Space-Saving Tips

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

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

Lets remove the `._` files first.

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

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

Device Locked Up or Boot Looping

In rare cases, it may happen that something in your **code.py** or **boot.py** files causes the device to get locked up, or even go into a boot loop. These are not your everyday Python exceptions, typically it's the result of a deeper problem within CircuitPython. In this situation, it can be difficult to recover your device if **CIRCUITPY** is not allowing you to modify the

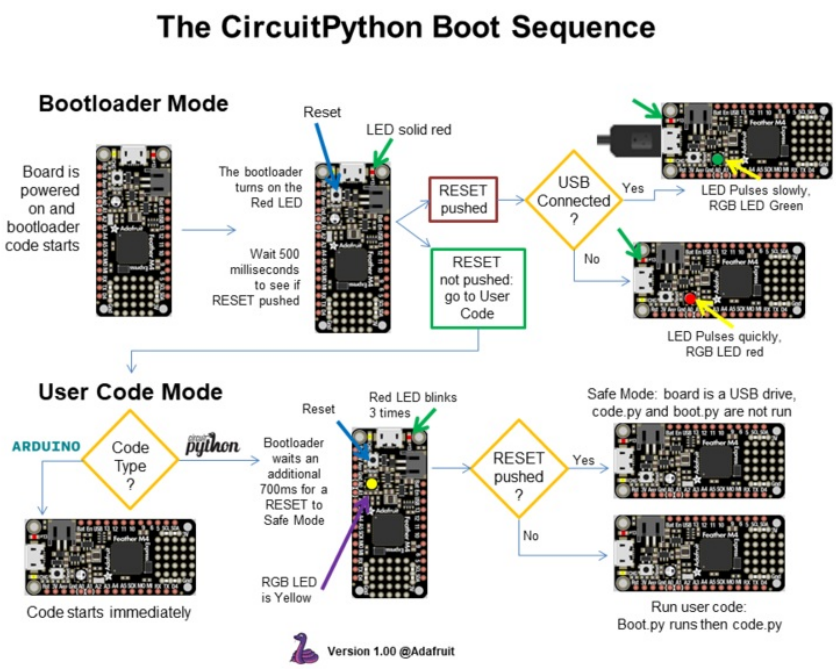
code.py or **boot.py** files. Safe mode is one recovery option. When the device boots up in safe mode it will not run the **code.py** or **boot.py** scripts, but will still connect the **CIRCUITPY** drive so that you can remove or modify those files as needed.

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

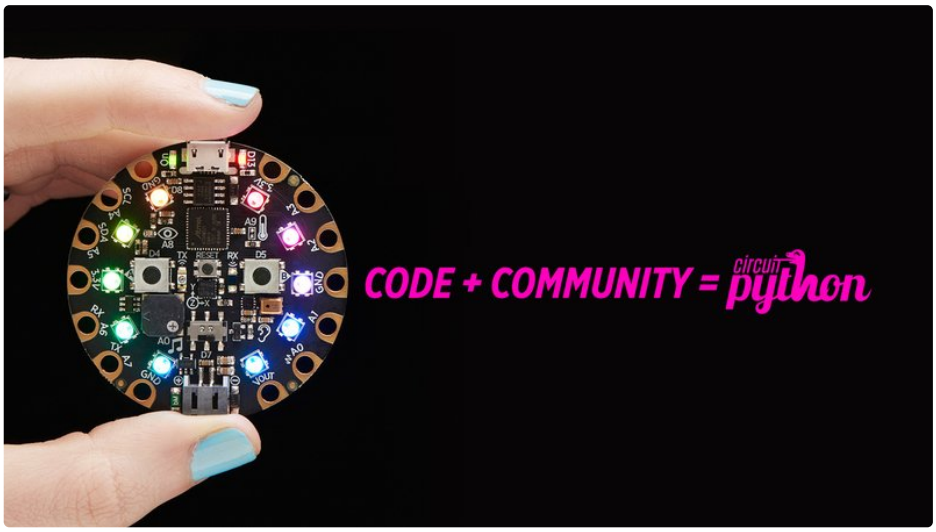
For most devices:
Press the reset button, and then when the RGB status LED blinks yellow, press the reset button again. Since your reaction time may not be that fast, try a "slow" double click, to catch the yellow LED on the second click.

For ESP32-S2 based devices:
Press and release the reset button, then press and release the boot button about 3/4 of a second later.

Refer to the following diagram for boot sequence details:



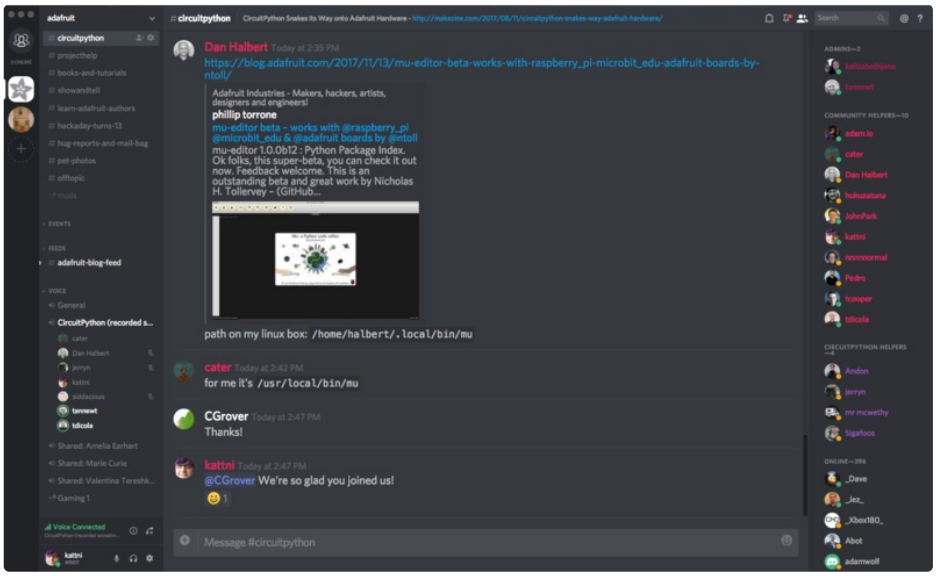
Welcome to the Community!



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

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

Adafruit Discord



The Adafruit Discord server is the best place to start. Discord is where the community comes together to volunteer and provide live support of all kinds. From general discussion to detailed problem solving, and everything in between, Discord is a digital maker space with makers from around the world.

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

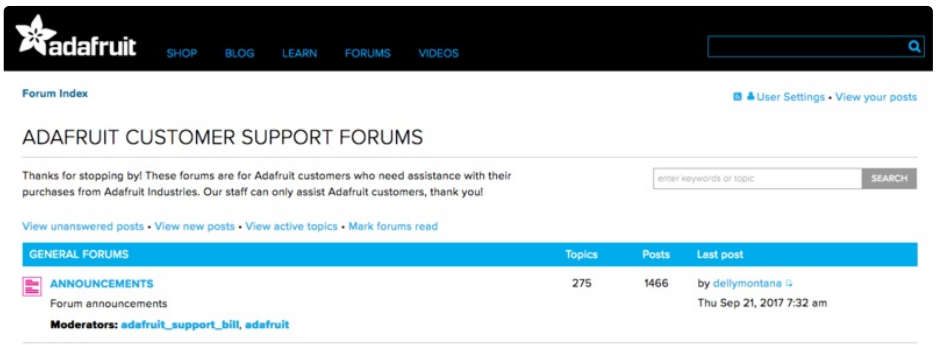
The help with CircuitPython channel is where to go with your CircuitPython questions. #help-with-circuitpython is there for new users and developers alike so feel free to ask a question or post a comment! Everyone of any experience level is welcome to join in on the conversation. We'd love to hear what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

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

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

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

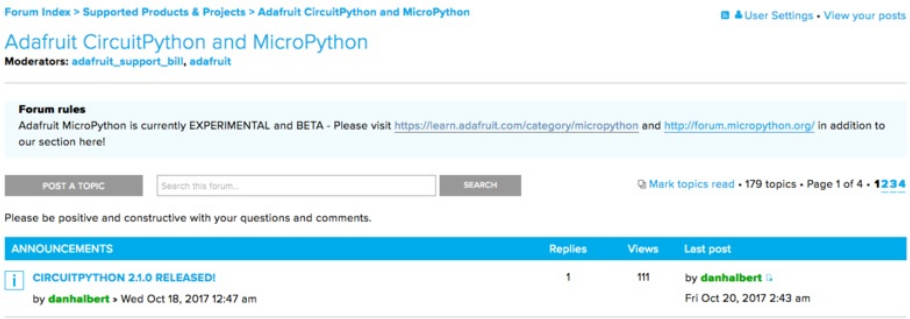
Adafruit Forums



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.

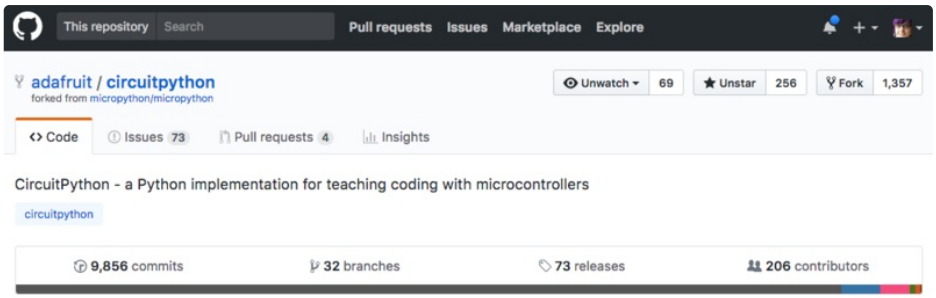
There are forum categories that cover all kinds of topics, including everything Adafruit. The Adafruit CircuitPython and MicroPython (https://adafru.it/xXA) category under "Supported Products & Projects" is the best place to post your CircuitPython questions.



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.

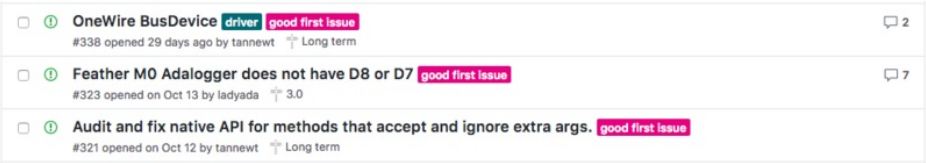
You might think you're just getting started, but you definitely know something that someone else doesn't. The great thing about the forums is that you can help others too! Everyone is welcome and encouraged to provide constructive feedback to any of the posted questions. This is an excellent way to contribute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

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 building CircuitPython. GitHub is the best source of ways to contribute to CircuitPython (https://adafru.it/tB7) itself. If you need an account, visit https://github.com/ (https://adafru.it/d6C)and sign up.

If you're new to GitHub or programming in general, there are great opportunities for you. Head over to adafruit/circuitpython (https://adafru.it/tB7) on GitHub, click on "Issues (https://adafru.it/Bee)", and you'll find a list that includes issues labeled "good first issue (https://adafru.it/Bef)". These are things we've identified as something that someone with any level of experience can help with. These issues include options like updating documentation, providing feedback, and fixing simple bugs.



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

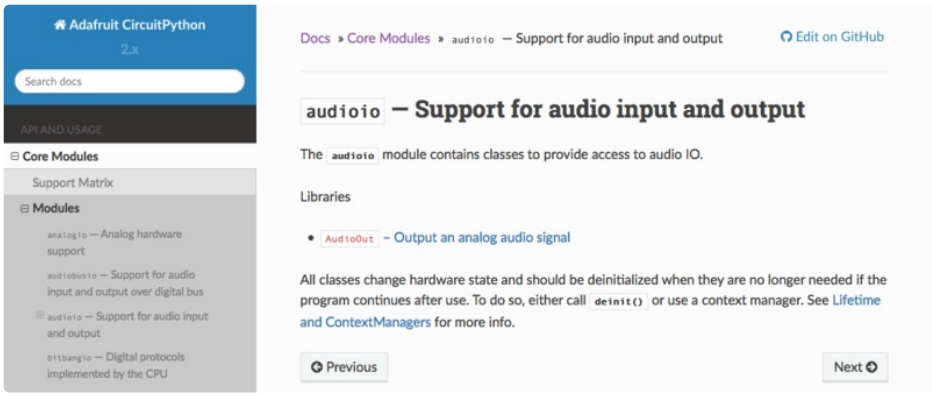
When working with CircuitPython, you may find problems. If you find a bug, that's great! We love bugs! Posting a detailed

issue to GitHub is an invaluable way to contribute to improving CircuitPython. Be sure to include the steps to replicate the issue as well as any other information you think is relevant. The more detail, the better!

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

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

ReadTheDocs



ReadTheDocs (https://adafru.it/Beg) is a an excellent resource for a more in depth look at CircuitPython. This is where you'll find things like API documentation and details about core modules. There is also a Design Guide that includes contribution guidelines for CircuitPython.

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

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

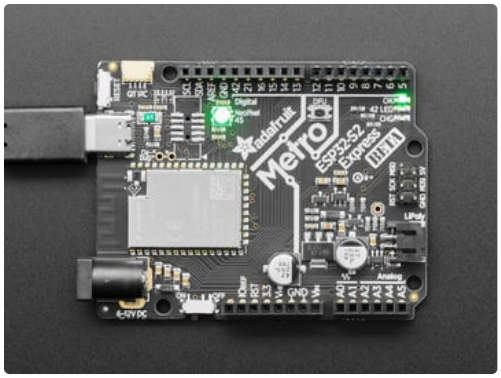
Follow the steps to get CircuitPython installed on your Metro ESP32-S2.

https://adafru.it/OsB

https://adafru.it/OsB

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

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



Plug your Metro ESP32-S2 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.

Follow the 6 steps listed in the Enter the ROM Bootloader section of the ROM Bootloader page (https://adafru.it/OsC) to enter the bootloader.

Follow the initial steps found in the Run esptool and check connection section of the ROM Bootloader page (https://adafru.it/OsC) 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!

CircuitPython Pin Names

CircuitPython for the Metro ESP32-S2 uses different pin names than you may be used to. Many CircuitPython boards use the D prefix for digital pin names, such as D1 or D12. The pin names for the Metro ESP32-S2 use the **IO** prefix, such as **IO1** or **IO12**.

The pin numbers on the Metro ESP32-S2 match the ESP32-S2 'low level chip pin numbers' that ESP32 users are most familiar with. The pins are not numbered like other typical Metro-shaped boards, so where you may expect pin 0 to be, its actually IO5.

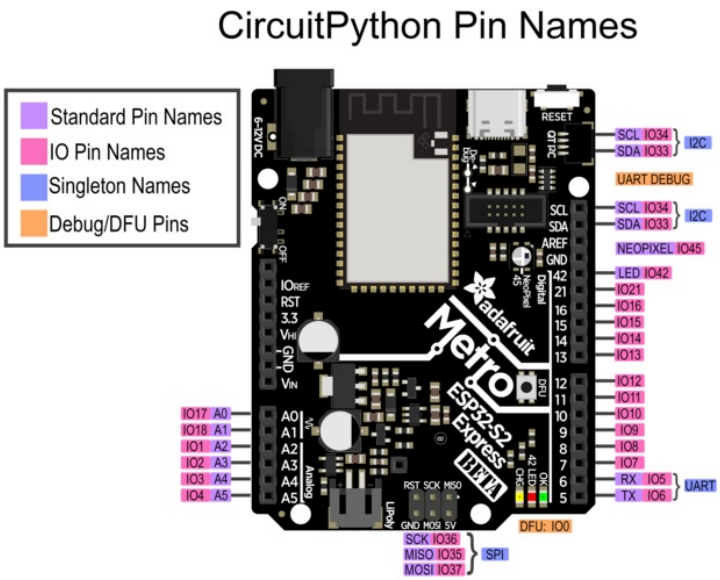
We're not yet using D prefix names to avoid the confusion of having D-prefix names not match the IO pins.

The following pins have **both** the standard CircuitPython pin name and the **IOx** pin name available:

- Analog pins A0-A5
- Default I2C port SCL & SDA
- Default SPI port SCK, MISO, MOSI
- Default hardware Serial port RX, TX
- LED (red LED)
- NEOPIXEL (built in RGB LED)

Pin Name Diagram

The following diagram shows the standard CircuitPython pin names, the IO pin names, the singleton names and the debug/DFU pins.



CircuitPython Internet Libraries

To use the internet-connectivity built into your ESP32-S2 with CircuitPython, you must first install a number of libraries. This page covers that process.

Adafruit CircuitPython Library Bundle

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

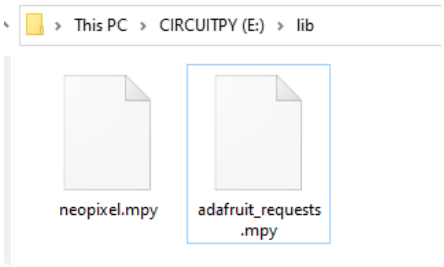
https://adafru.it/ENC

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. Instead, add each library as you need it, this will reduce the space usage but you'll need to put in a little more effort.

At a minimum we recommend the following libraries, in fact we more than recommend. They're basically required. So grab them and install them into **CIRCUITPY/lib** now!

- **adafruit_requests.mpy** - A requests-like library for HTTP commands.
- **neopixel.mpy** - Helper library to use NeoPixel LEDs, often built into the boards so they're great for quick feedback



Once you have added those files, please continue to the next page to set up and test Internet connectivity

CircuitPython Internet Test

Once you have CircuitPython installed and the minimum libraries installed we can get your board connected to the Internet.

To get connected, you will need to start by creating a **secrets.py** file.

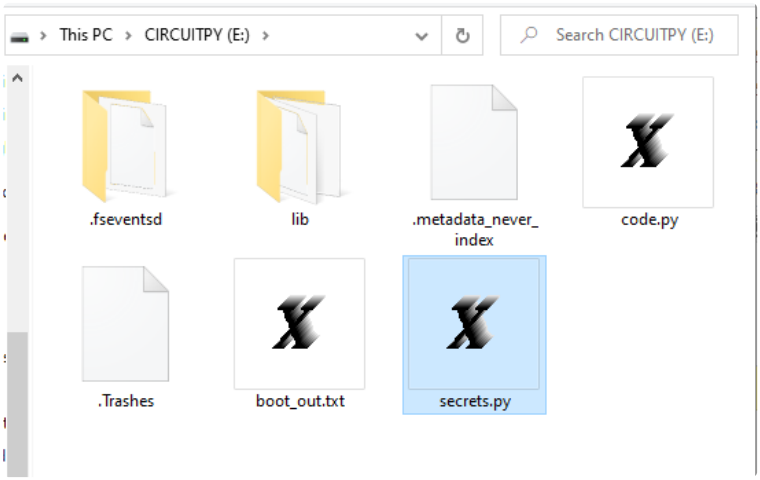
Secrets File

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 **secrets.py** file, that is in your **CIRCUITPY** drive, to hold secret/private/custom data. That way you can share your main project without worrying about accidentally sharing private stuff.

Your **secrets.py** file should look like this:

```
# 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

secrets = {
    'ssid' : 'home_wifi_network',
    'password' : 'wifi_password',
    'aio_username' : 'my_adafruit_io_username',
    'aio_key' : 'my_adafruit_io_key',
    'timezone' : "America/New_York", # http://worldtimeapi.org/timezones
    }
```

Copy and paste that text/code into a file called **secrets.py** and save it to your **CIRCUITPY** folder like so:



Inside is a python dictionary named secrets with a line for each entry. Each entry has an entry name (say `'ssid'`) and then a colon to separate it from the entry key `'home ssid'` and finally a comma ,

At a minimum you'll need to adjust the `ssid` and `password` for your local WiFi setup 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, just cause its called secrets doesn't mean you can't have general customization data in there!

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 **secrets.py** - keep that out of GitHub, Discord or other project-sharing sites.

Don't share your secrets.py file, it has your passwords and API keys in it!

Connect to WiFi

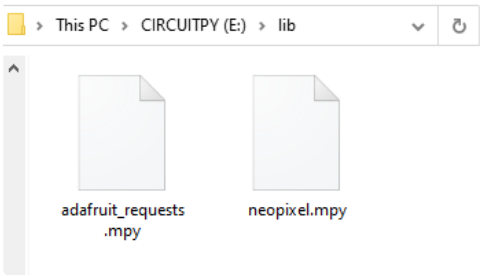
OK now you have your secrets setup - you can connect to the Internet using the Requests module.

First make sure you are running the latest version of Adafruit CircuitPython (https://adafru.it/Amd) for your board.

Next you'll need to install the necessary libraries to use the hardware--carefully follow the steps to find and install these libraries from Adafruit's CircuitPython library bundle (https://adafru.it/zdx). Our introduction guide has a great page on how to install the library bundle (https://adafru.it/ABU).

- **adafruit_requests**
- **neopixel**

Before continuing make sure your board's CIRCUITPY/lib folder or root filesystem has the above files copied over.



Once that's done, load up the following example using Mu or your favorite editor:

```
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"

# Get wifi details and more from a secrets.py file
try:
    from secrets import secrets
except ImportError:
    print("WiFi secrets are kept in secrets.py, please add them there!")
    raise

print("ESP32-S2 WebClient 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 %s"%secrets["ssid"])
wifi.radio.connect(secrets["ssid"], secrets["password"])
print("Connected to %s!"%secrets["ssid"])
print("My IP address is", wifi.radio.ipv4_address)

ipv4 = ipaddress.ip_address("8.8.4.4")
print("Ping google.com: %f ms" % (wifi.radio.ping(ipv4)*1000))

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

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

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

print()

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

print("done")
```

And save it to your board. Make sure the file is named **code.py**.

Open up your REPL, you should see something like the following:

```
1. screen /Users/brentrubell (screen)
Auto-reload is on. Simply save files over USB to run them or enter REPL to disable.
code.py output:
ESP32-S2 WebClient Test
My MAC addr: ['0x7c', '0xdf', '0xa1', '0x0', '0x52', '0xa0']
Avaliable WiFi networks:
        Brunelleschi            RSSI: -84       Channel: 6
        Transit         RSSI: -54       Channel: 1
        Fios-5dLNb              RSSI: -66       Channel: 1
        disconnecteder          RSSI: -86       Channel: 1
        SKJFios-ZV0O7           RSSI: -83       Channel: 11
        Fios-QIVUQ              RSSI: -83       Channel: 11
        Fios-ZV0O7              RSSI: -85       Channel: 11
        [redacted]              RSSI: -58       Channel: 2
        [redacted]              RSSI: -76       Channel: 8
        NETGEAR52               RSSI: -81       Channel: 10
Connecting to Transit
Connected to Transit!
None
My IP address is 192.168.1.182
Ping google.com: 0.065000 ms
Fetching text from http://wifitest.adafruit.com/testwifi/index.html
----------------------------------------
This is a test of Adafruit WiFi!
If you can read this, its working :)

----------------------------------------
Fetching json from https://www.adafruit.com/api/quotes.php
----------------------------------------
[{'text': 'Science, my lad, is made up of mistakes, but they are mistakes which it is u
seful to make, because they lead little by little to the truth', 'author': 'Jules Verne
'}]
----------------------------------------

Fetching and parsing json from https://api.github.com/repos/adafruit/circuitpython
----------------------------------------
CircuitPython GitHub Stars 1896
----------------------------------------
done
```

In order, the example code...

Checks the ESP32-S2's MAC address.

```
print("My MAC addr:", [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("Avaliable 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 **secrets.py** file, prints out its local IP address, and attempts to ping google.com to check its network connectivity.

```
print("Connecting to %s"%secrets["ssid"])
wifi.radio.connect(secrets["ssid"], secrets["password"])
print(print("Connected to %s!"%secrets["ssid"]))
print("My IP address is", wifi.radio.ipv4_address)

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

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 http **s** url for SSL connectivity.

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

OK you now have your ESP32-S2 board set up with a proper **secrets.py** file and can connect over the Internet. If not, check that your **secrets.py** file has the right ssid and password and retrace your steps until you get the Internet connectivity working!

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

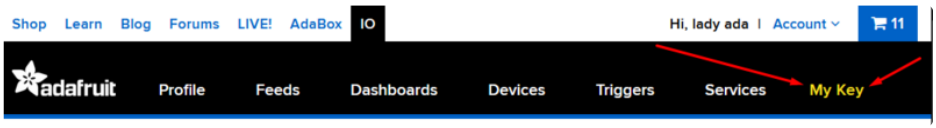
It's free! Visit https://accounts.adafruit.com/ (https://adafru.it/dyy) to register and make an account if you do not already have one

Step 2) Sign into Adafruit IO

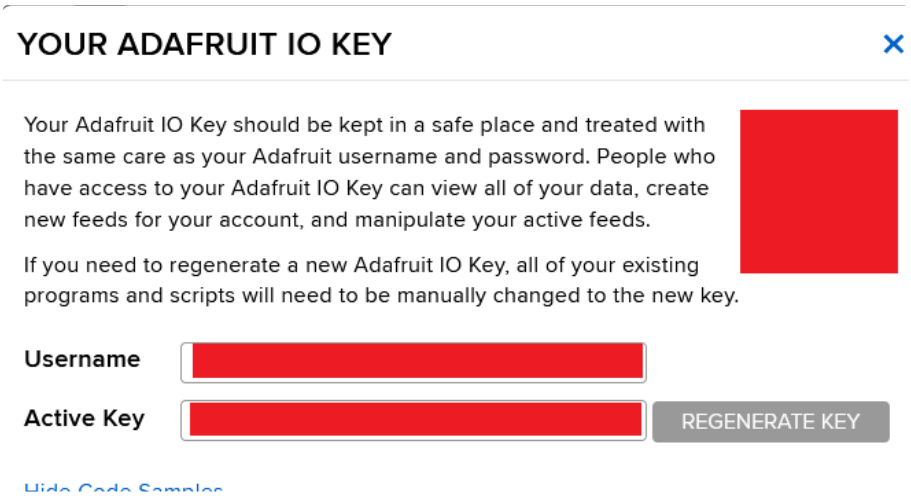
Head over to io.adafruit.com (https://adafru.it/fsU) and click **Sign In** to log into IO using your Adafruit account. It's free and fast to join.

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 your secrets.py file on your CIRCUITPY drive and add three lines for `aio_username`, `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

secrets = {
    'ssid' : 'home_wifi_network',
    'password' : 'wifi_password',
    'aio_username' : 'my_adafruit_io_username',
    'aio_key' : 'my_adafruit_io_key',
    'timezone' : "America/New_York", # http://worldtimeapi.org/timezones
    }
```

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 worldtimeapi 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 ssl
import wifi
import socketpool
import adafruit_requests
import secrets


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"

# Get wifi details and more from a secrets.py file
try:
    from secrets import secrets
except ImportError:
    print("WiFi secrets are kept in secrets.py, please add them there!")
    raise

# Get our username, key and desired timezone
aio_username = secrets["aio_username"]
aio_key = secrets["aio_key"]
location = secrets.get("timezone", None)
TIME_URL = "https://io.adafruit.com/api/v2/%s/integrations/time/strftime?x-aio-key=%s" % (aio_username, aio_key)
TIME_URL += "&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 %s"%secrets["ssid"])
wifi.radio.connect(secrets["ssid"], secrets["password"])
print("Connected to %s!"%secrets["ssid"])
print("My IP address is", wifi.radio.ipv4_address)

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

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!

```
Connecting to adafruit
Connected to adafruit!
My IP address is 10.0.1.148
Ping google.com: 0.008000 ms
Fetching text from https://io.adafruit.com/api/v2/[redacted]/integrations/time/strftime?x-aio-
key=[redacted]&fmt=%25Y-%25m-%25d+%25H%3A%25M%3A%25S.%25L+%25j+%25u+%25z+%25Z
----------------------------------------
2020-12-05 18:51:32.145 340 6 -0500 EST
----------------------------------------
```

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

Arduino IDE Setup

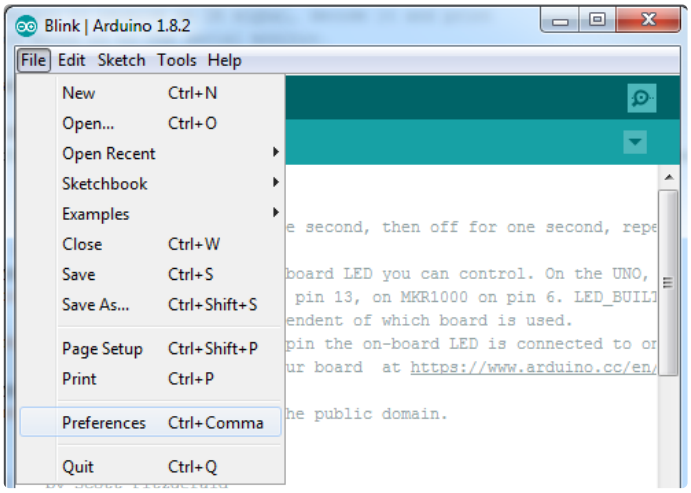
The first thing you will need to do is to download the latest release of the Arduino IDE. You will need to be using **version 1.8** or higher for this guide

https://adafru.it/f1P

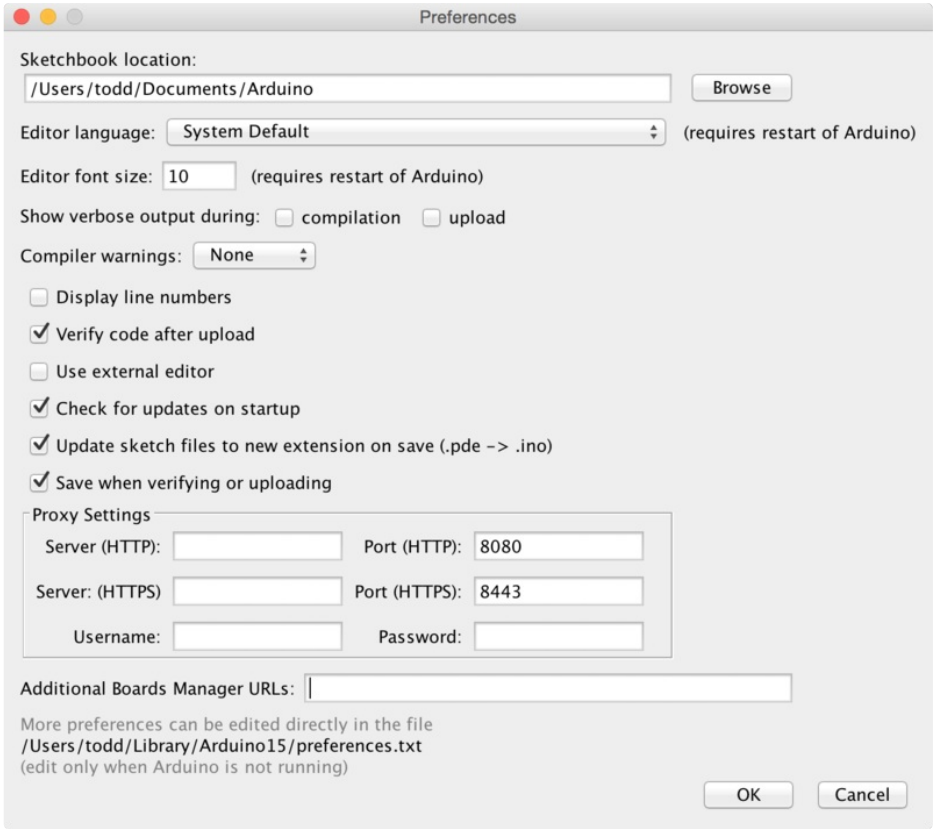
https://adafru.it/f1P

The ESP32-S2 Arduino board support package is currently part of the **2.0.0-alpha1** release. To use the ESP32-S2 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*.



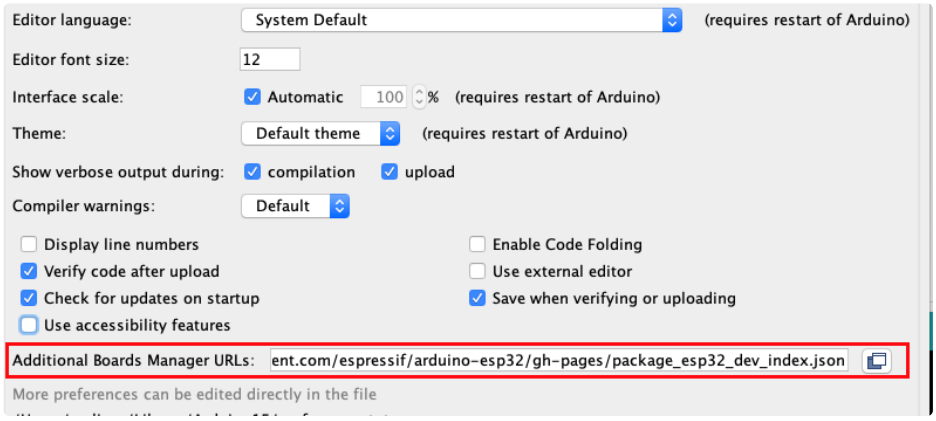
A dialog will pop up just like the one shown below.



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) (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_dev_index.json

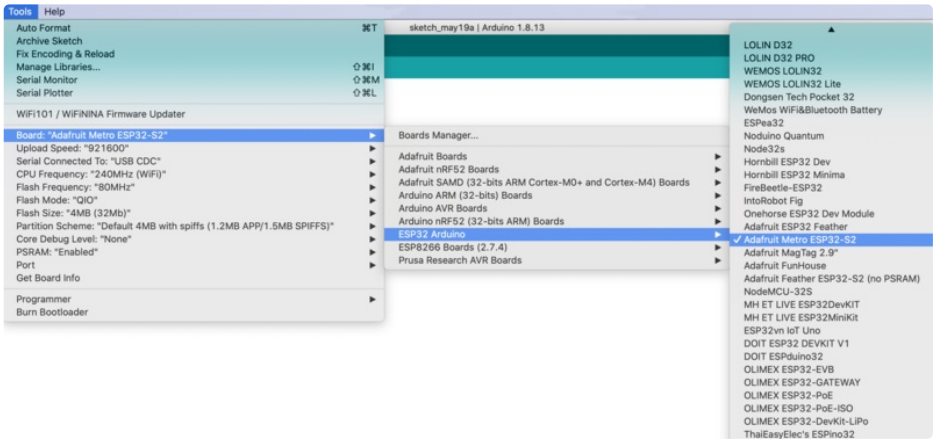


If you have multiple boards you want to support, say ESP8266 and Adafruit, have both URLs in the text box separated by a comma (,)

Once done click **OK** to save the new preference settings.

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 boards.

Look for the board called Adafruit Metro ESP32-S2.



Using with Arduino IDE

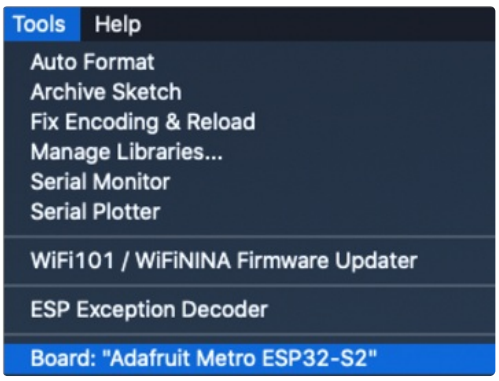
Arduino support for the ESP32-S2 is very challenging right now, we don't recommend it. Please use CircuitPython!

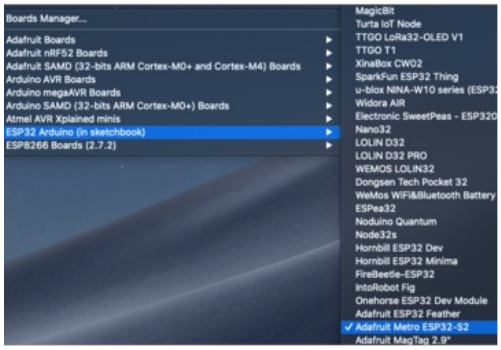
Blink

Now you can upload your first blink sketch!

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

Select ESP32-S2 Board in Arduino IDE





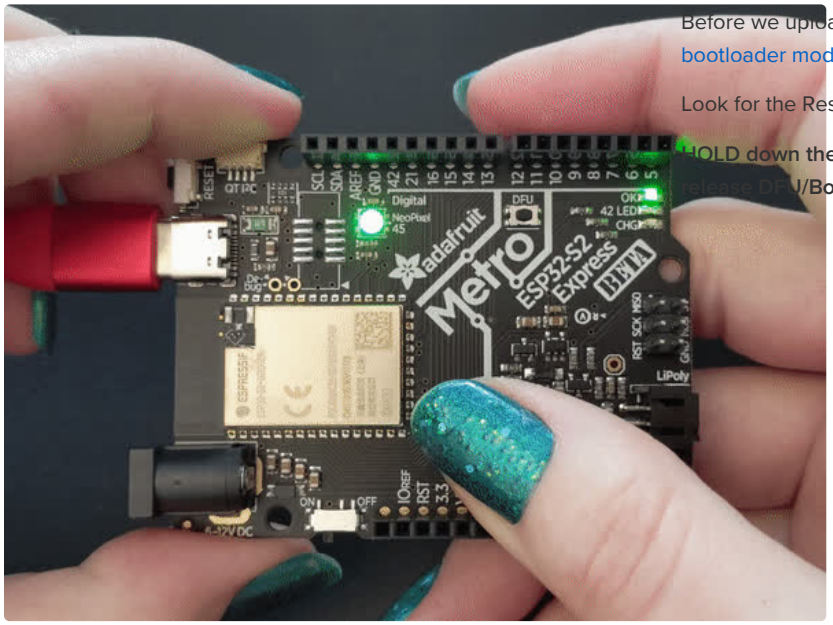
On the Arduino IDE, **click**:

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

If you don't see your board, make sure you have the latest version of the ESP32 board support package

Launch ESP32-S2 ROM Bootloader

ESP32-S2 support in Arduino does not yet auto-reset on upload, so you will have to put the board into ROM bootloader mode EVERY time you want to upload!

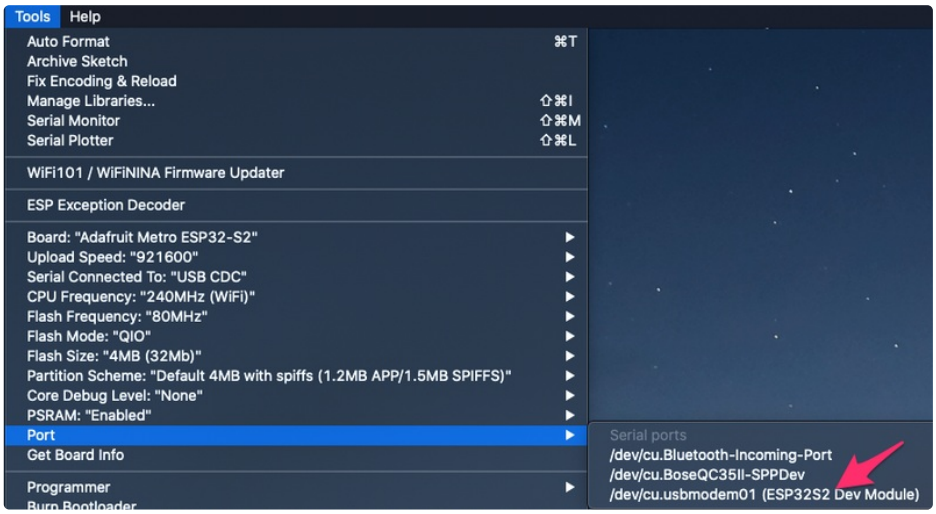


Before we upload a sketch, place your ESP32-S2 board into ROM bootloader mode (https://adafru.it/OsC).

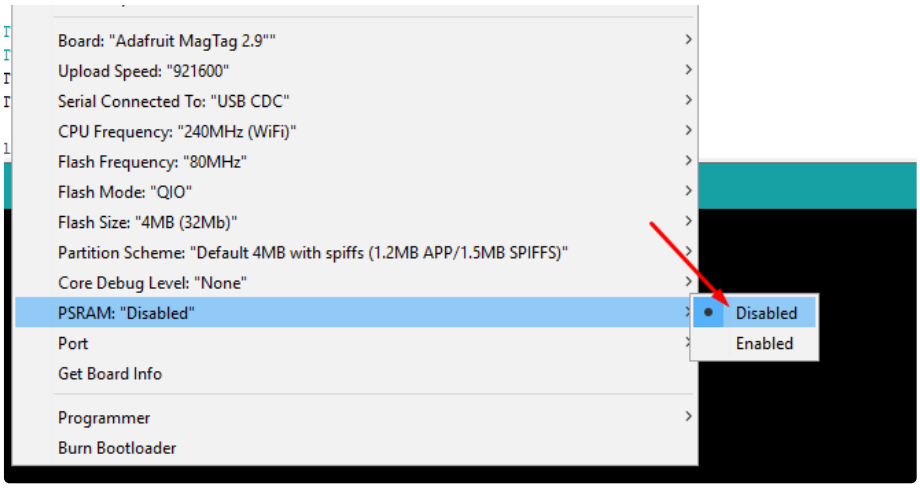
Look for the Reset button and a second DFU / BOOT0 button

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

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



Disable PSRAM (it is currently hard-faulting in Arduino (https://adafru.it/OCY))



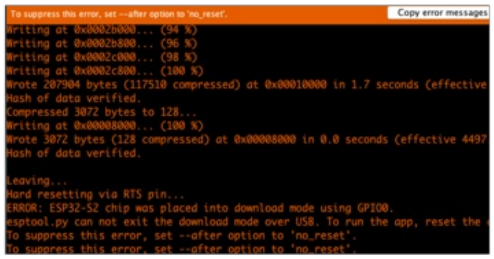
Load Blink Sketch

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:

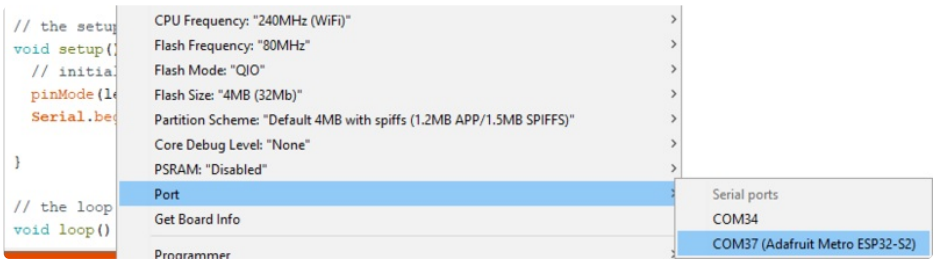


And click upload! After uploading, you may see something like this, warning you that we could not get out of reset.

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 must 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!



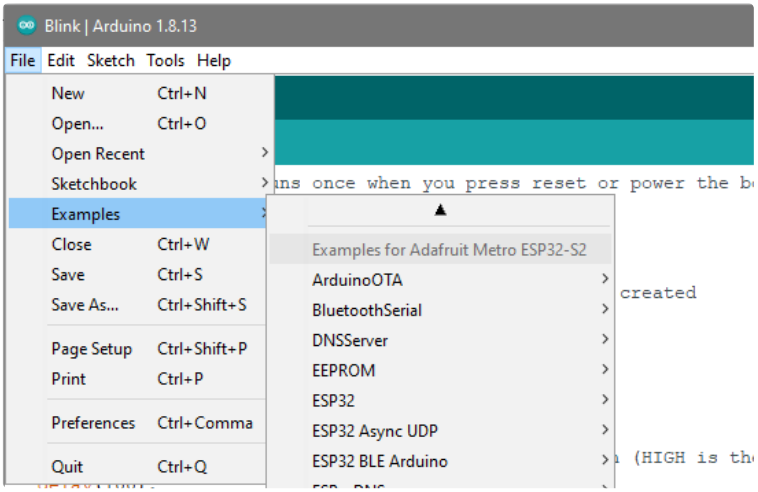
If you want to upload a new sketch, you will have to go through the same process:

- Reset board into ROM bootloader with DFU/BOOT0 + Reset buttons
- Select ESP ROM bootloader in Tools->Port menu
- Upload sketch
- Click reset button to launch code

[We have opened an issue with Espressif and hopefully they will fix auto-reset-into-bootloader soon!](https://adafru.it/OCg) (https://adafru.it/OCg)

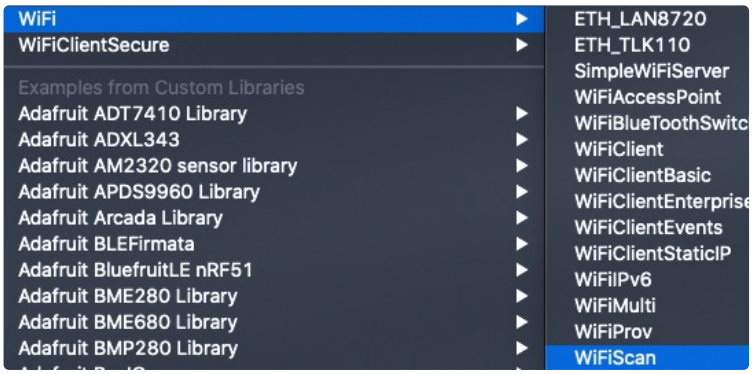
WiFi Test

Thanksfully if you have ESP32 sketches, they'll 'just work' with ESP32-S2. 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 MagTag"** etc)



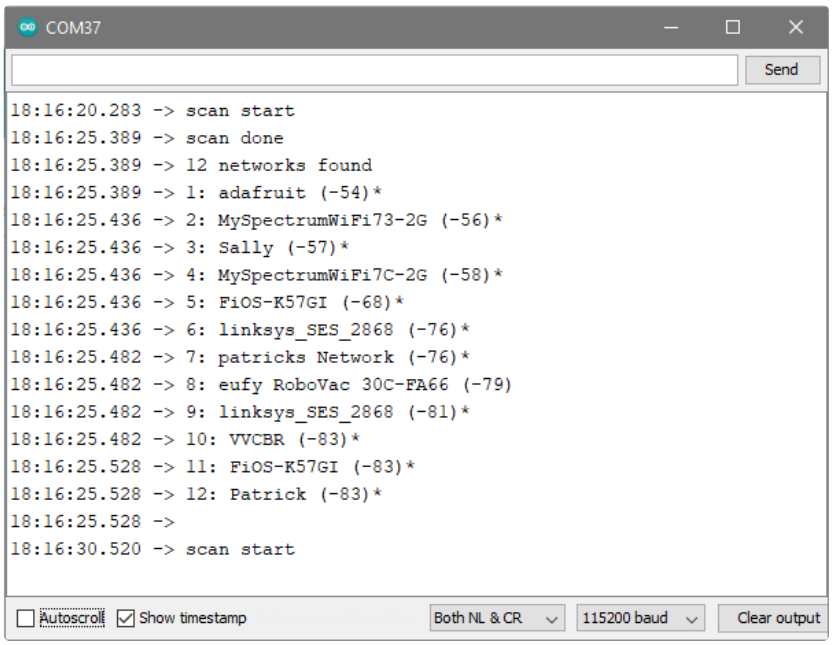
Let's start by scanning the local networks.

Load up the WiFiScan example under **Examples->Examples for Adafruit Metro ESP32-S2->WiFi->WiFiScan**



And **upload this example to your board**. The ESP32-S2 should scan and find WiFi networks around you.

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-S2. 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-S2 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:

```
/*
  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(9600);
  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);
  }
}


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-S2 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:

```
/*
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>

// 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(9600);
  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, 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! This example will connect to a twitter server to download a JSON snippet that says how many followers adafruit has

```
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
```

JSON Parsing Demo

This example is a little more advanced - many sites will have API's that give you JSON data. We will build on the previous SSL example to connect to twitter and get that JSON data chunk

Then we'll use ArduinoJSON (https://adafru.it/Evn) to convert that to a format we can use and then display that data on the serial port (which can then be re-directed to a display of some sort)

First up, use the Library manager to install ArduinoJSON (https://adafru.it/Evo).

Then load the example **JSONdemo** by copying the code below and pasting it into your Arduino IDE.

```
/*
This example creates a client object that connects and transfers
data using always SSL, then shows how to parse a JSON document in an HTTP response.

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

Written by Arturo Guadalupi + Copyright Benoit Blanchon 2014-2019
last revision November 2015

*/

#include <WiFiClientSecure.h>
#include <ArduinoJson.h>

// uncomment the next line if you have a 128x32 OLED on the I2C pins
//#define USE_OLED
```

```
#if defined(USE_OLED)
  #include <Adafruit_SSD1306.h>
  Adafruit_SSD1306 display = Adafruit_SSD1306(128, 32, &Wire);
#endif

// 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(9600);

  #if defined(USE_OLED)
    if(!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) { // Address 0x3C for 128x32
        Serial.println(F("SSD1306 allocation failed"));
        for(;;); // Don't proceed, loop forever
    }
    display.display();
    display.setTextSize(1);
    display.setTextColor(WHITE);
    display.clearDisplay();
    display.setCursor(0,0);
    #else
      // Don't wait for serial if we have an OLED
      while (!Serial) {
        delay(10); // wait for serial port to connect. Needed for native USB port only
      }
  #endif
  // attempt to connect to Wifi network:
  Serial.print("Attempting to connect to SSID: ");
  Serial.println(ssid);
  #if defined(USE_OLED)
    display.clearDisplay(); display.setCursor(0,0);
    display.print("Connecting to SSID\n"); display.println(ssid);
    display.display();
  #endif

  // Connect to WPA/WPA2 network. Change this line if using open or WEP network:
  WiFi.begin(ssid, pass);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }

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

  #if defined(USE_OLED)
    display.print("...OK!");
    display.display();
```

```
  #endif

  printWifiStatus();

}

uint32_t bytes = 0;

void loop() {
  Serial.println("\nStarting connection to server...");
  #if defined(USE_OLED)
    display.clearDisplay(); display.setCursor(0,0);
    display.print("Connecting to "); display.print(SERVER);
    display.display();
  #endif

  // 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();
  }

  // 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);
  #if defined(USE_OLED)
      display.print("Connection failed, code: "); display.println(status);
      display.display();
  #endif

  return;
  }

  // wait until we get a double blank line
  client.find("\r\n\r\n", 4);

  // 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());
    return;
  }

  // Extract values
  JsonObject root_0 = doc[0];
  Serial.println(F("Response:"));
  const char* root_0_screen_name = root_0["screen_name"];
  long root_0_followers_count = root_0["followers_count"];

  Serial.print("Twitter username: "); Serial.println(root_0_screen_name);
  Serial.print("Twitter followers: "); Serial.println(root_0_followers_count);
  #if defined(USE_OLED)
    display.clearDisplay(); display.setCursor(0,0);
```

```
    display.setTextSize(2);
    display.println(root_0_screen_name);
    display.println(root_0_followers_count);
    display.display();
    display.setTextSize(1);
  #endif


  // Disconnect
  client.stop();

  delay(10000);
}


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");
}
```

By default it will connect to to the Twitter banner image API, parse the username and followers, and display them.

```
Attempting to connect to SSID: Transit
........
Connected to WiFi
SSID: Transit
IP Address: 192.168.1.182
signal strength (RSSI):-54 dBm

Starting connection to server...
connected to server
Response:
Twitter username: adafruit
Twitter followers: 176400
```

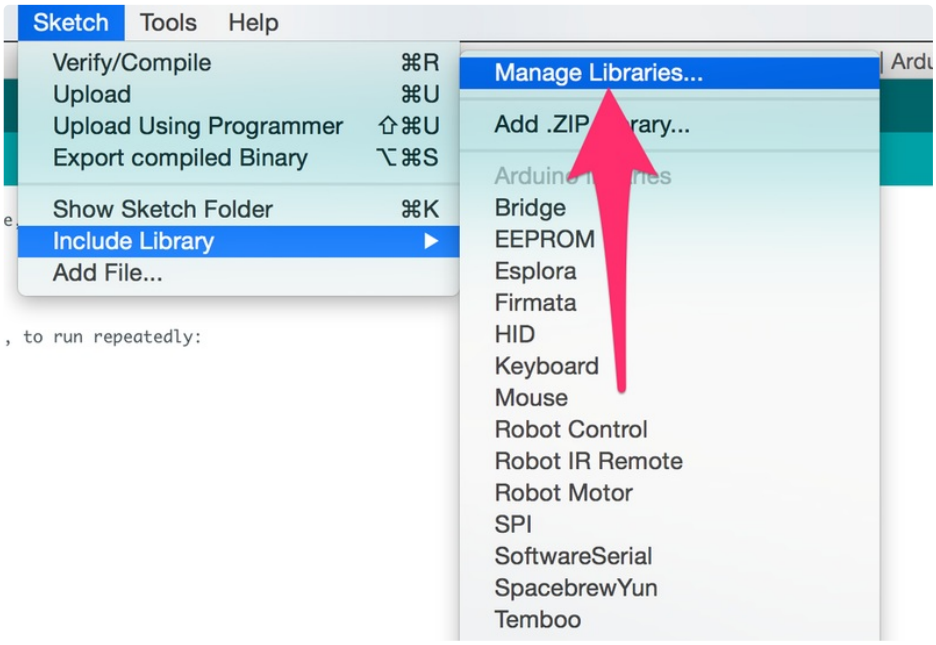
Usage with Adafruit IO

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

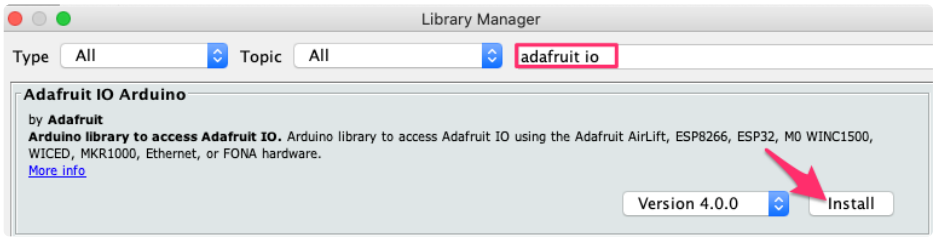
- For more information and guides about Adafruit IO, check out the Adafruit IO Basics Series. (https://adafru.it/iDX)

Install Libraries

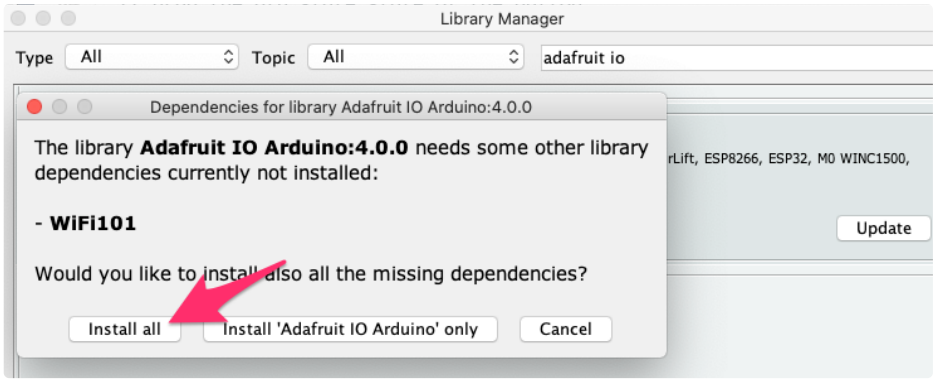
In the Arduino IDE, navigate to **Sketch -> Include Library->Manage Libraries...**



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.



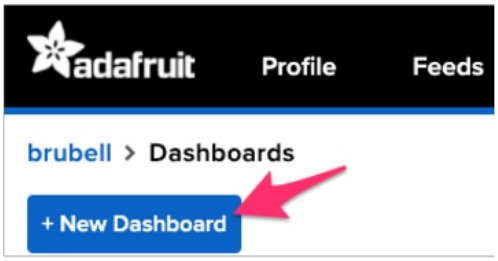
When asked to install dependencies, **click Install all**.

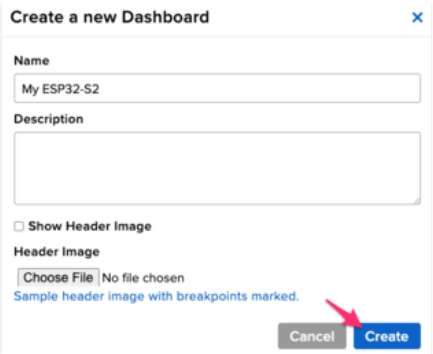


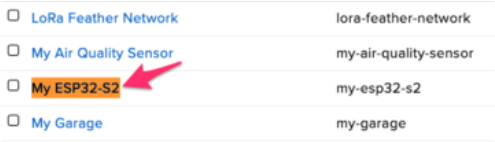
Adafruit IO Setup

If you do not already have an Adafruit IO account, [create one now](https://adafru.it/fH9) (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 board and Adafruit IO.

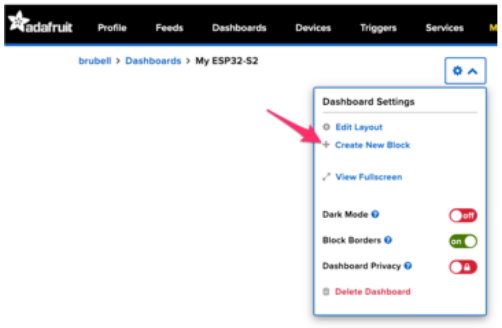


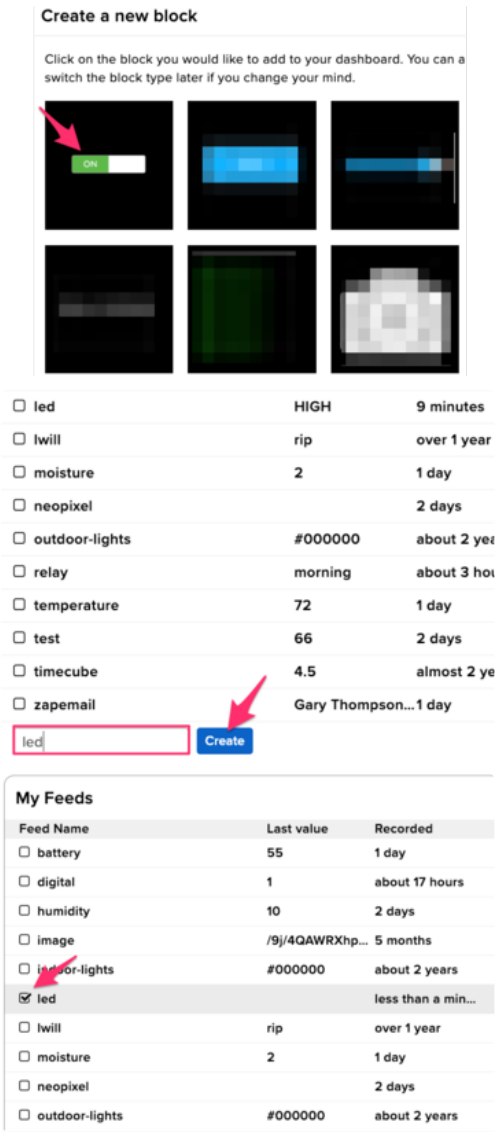




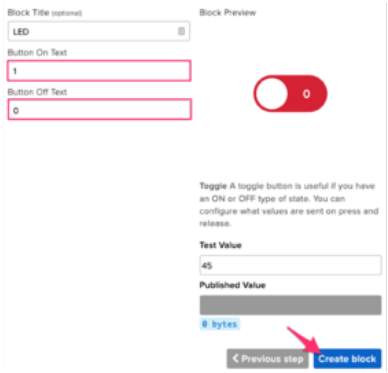
- **Click** the New Dashboard button.
- **Name your dashboard** *My ESP32-S2.*
- Your new dashboard should appear in the list.
- **Click** the link to be brought to your new dashboard.

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**.



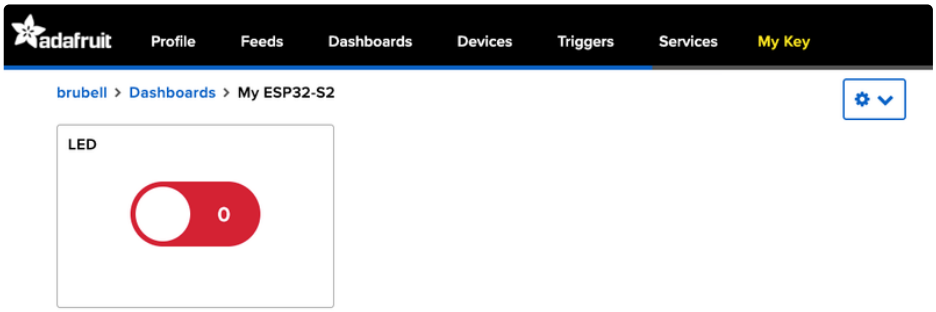


- **Click the cog** at the top right hand corner of your dashboard.
- In the dashboard settings dropdown, **click Create New Block.**
- **Select** the toggle block.
- Under My Feeds, **enter *led* as a feed name**. **Click Create**.
- **Choose the *led* feed** to connect it to the toggle block. **Click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

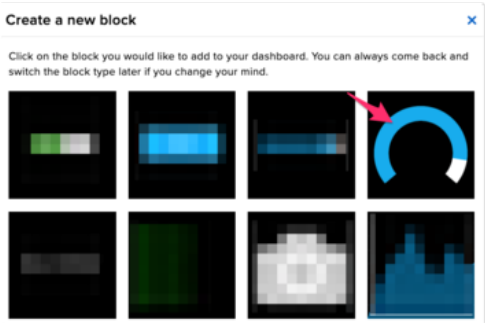


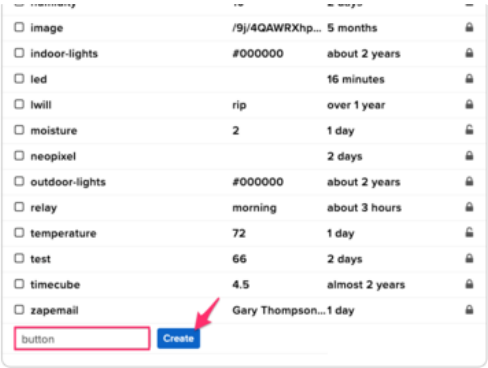
Under Block Settings,

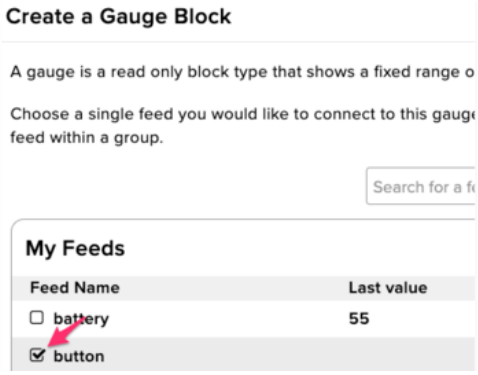
- **Change Button On Text** to **1**
- **Change Button Off Text** to **0**
- **Click Create block**



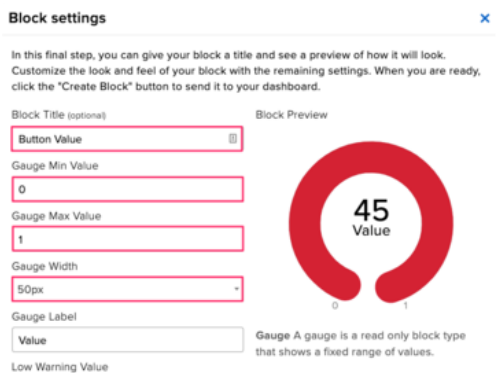
Next up, we'll want to display button press data from your board on Adafruit IO. To do this, **we'll add a gauge block to the Adafruit IO dashboard**. A gauge is a read only block type that shows a fixed range of values.







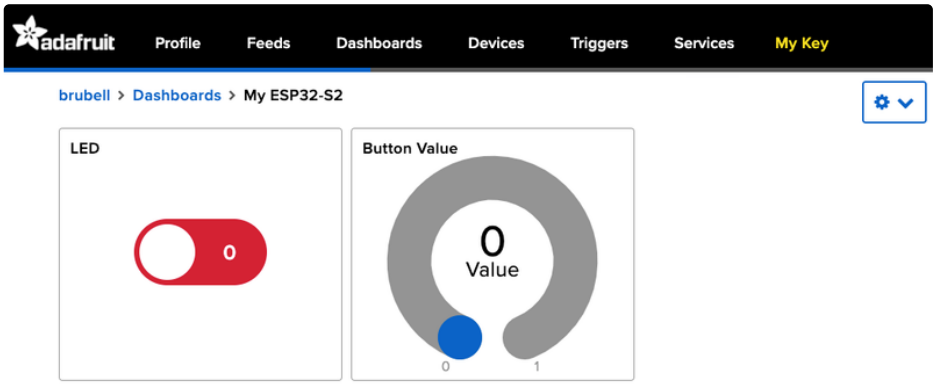
- **Click the cog** at the top right hand corner of your dashboard.
- In the dashboard settings dropdown, **click Create New Block.**
- **Select** the gauge block.
- Under My Feeds, **enter *button* as a feed name**.
  - **Click Create**.
- **Choose the *button* feed** to connect it to the toggle block.
  - **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, you'll need to add your network and Adafruit IO credentials. **Click on the config.h tab** in the sketch.

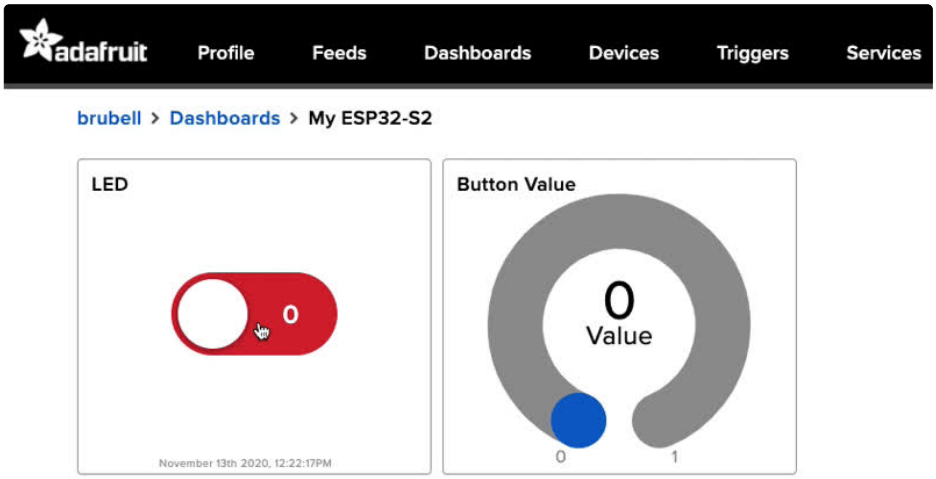
Obtain your Adafruit IO Credentials from navigating to io.adafruit.com and clicking My Key (https://adafru.it/fsU). Copy and paste these credentials next to `IO_USERNAME` and `IO_KEY`.

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

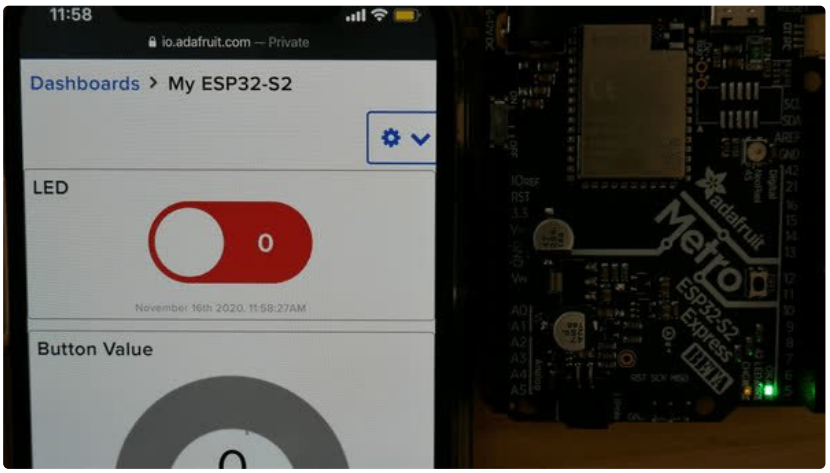
```
adafruitio_26_led_btn   config.h

#define WIFI_SSID "your_ssid"
#define WIFI_PASS "your_pass"

// uncomment the following line if you are using airlift
//#define USE_AIRLIFT

// uncomment the following line if you are using winc1500
// #define USE_WINC1500
```

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

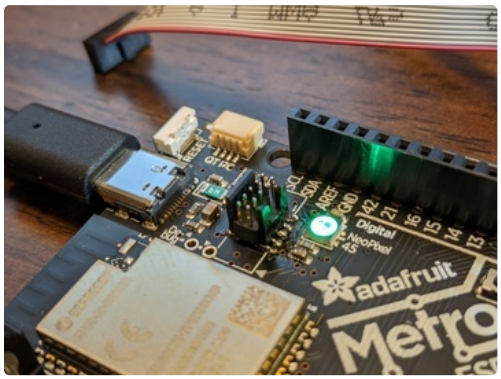
Open the Arduino Serial monitor and **navigate to the Adafruit IO dashboard you created**. You should see the gauge response to button press and the board's LED light up in response to the Toggle Switch block.



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



Debugging with OpenOCD



It is possible to use a true step-and-memory debugger using OpenOCD - you will need an external debugger like a JLink or FT2232 JTAG adapter

You can use an OpenOCD compatible debugging probe such as J-Link for source level debugging of C and C++ code on the Adafruit Metro ESP32-S2. (However, I had more luck with a J-link BASE than a J-Link EDU Mini and more consistent behavior with the built in debugger of the Kaluga devkit than either)

SEGGER J-Link EDU Mini - JTAG/SWD Debugger

Doing some serious development on any ARM-based platform, and tired of 'printf' plus an LED to debug? A proper JTAG/SWD HW debugger can make debugging more of a pleasure and...

$19.95

In Stock

Add to Cart

Metro ESP32S2

First you'll need to solder on an SWD connector at the indicated location on the Metro ESP32-S2 PCB. Note that the "key" of the box header must be on the same side as the "pin 1" arrow. These are very fine pitch headers and can be difficult to solder. Any solder bridges between the pins will prevent the connection from working.

SWD 0.05" Pitch Connector - 10 Pin SMT Box Header

This 1.27mm pitch, 2x5 male SMT Box Header is the same one used on our SWD Cable Breakout Board. The header...

$1.50

In Stock

Add to Cart

Mini SWD 0.05" Pitch Connector - 10 Pin SMT Box Header

We've carrying a new 1.27mm pitch 2x5 Mini SWD 0.05" Pitch Connector. It's a tinier, bite-sized version of the

$1.95

In Stock

Add to Cart

OpenOCD Setup

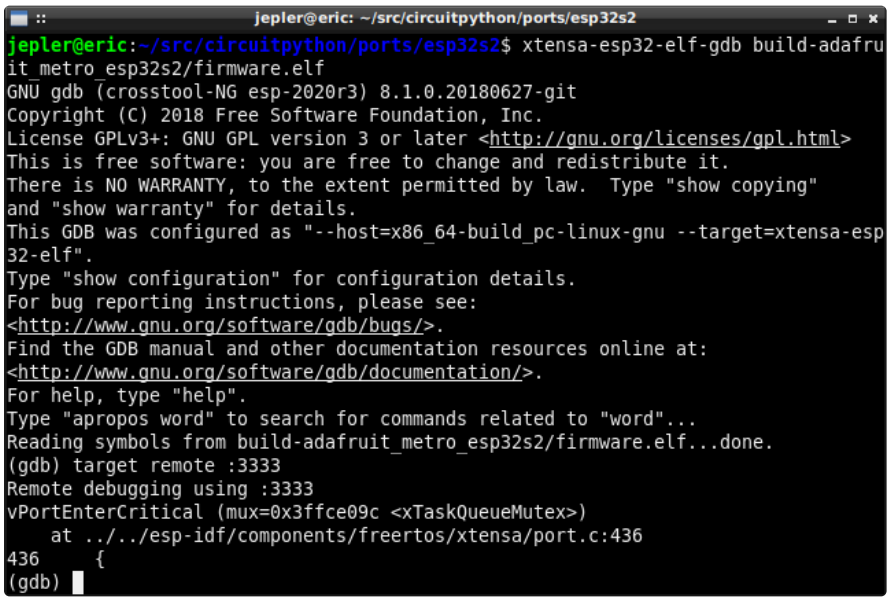
Next, you'll need the version of the Open On-Chip Debugger (OpenOCD) updated with ESP32-S2 support by Espressif. If you have a CircuitPython build environment available, just use the **export.sh** from your CircuitPython source directory. Otherwise, you'll want to follow the official installation instructions (https://adafru.it/OBa) from Espressif.

Verify that you have Espressif's version of OpenOCD--just look for "esp32" in the version number, and make sure the date is at least as new as this one:

```
$ openocd --version
Open On-Chip Debugger v0.10.0-esp32-20200709 (2020-07-09-08:54)
```

You also need the correct debugger program, `xtensa-esp32s2-elf-gdb`, this version or newer:

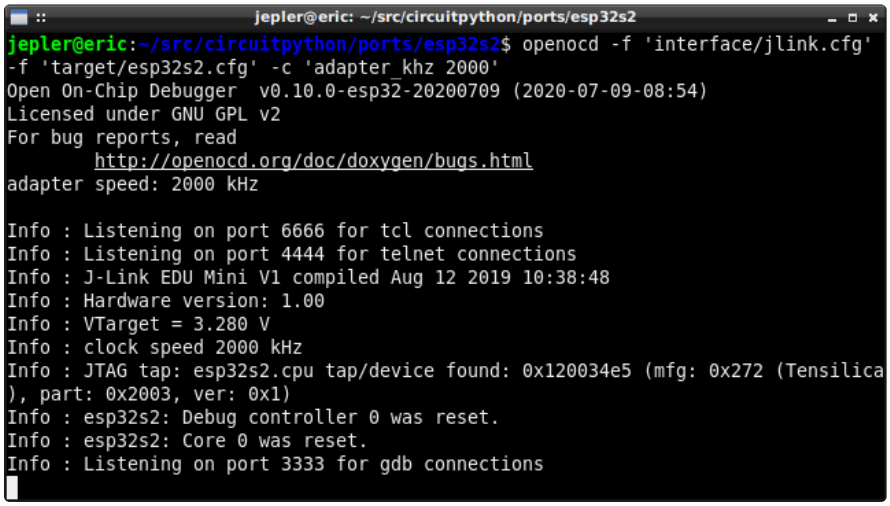
```
$ xtensa-esp32s2-elf-gdb --version
GNU gdb (crosstool-NG esp-2020r3) 8.1.0.20180627-git
```



In one terminal, start OpenOCD:
`openocd -f 'interface/jlink.cfg' -f 'target/esp32s2.cfg' -c 'adapter_khz 2000'`

In another terminal, start gdb and connect to OpenOCD:

```
$ xtensa-esp32s2-elf-gdb build-adafruit_metro_esp32s2/firmware.elf
(gdb) target remote :3333
Remote debugging using :3333
0x4001b800 in ?? ()
```



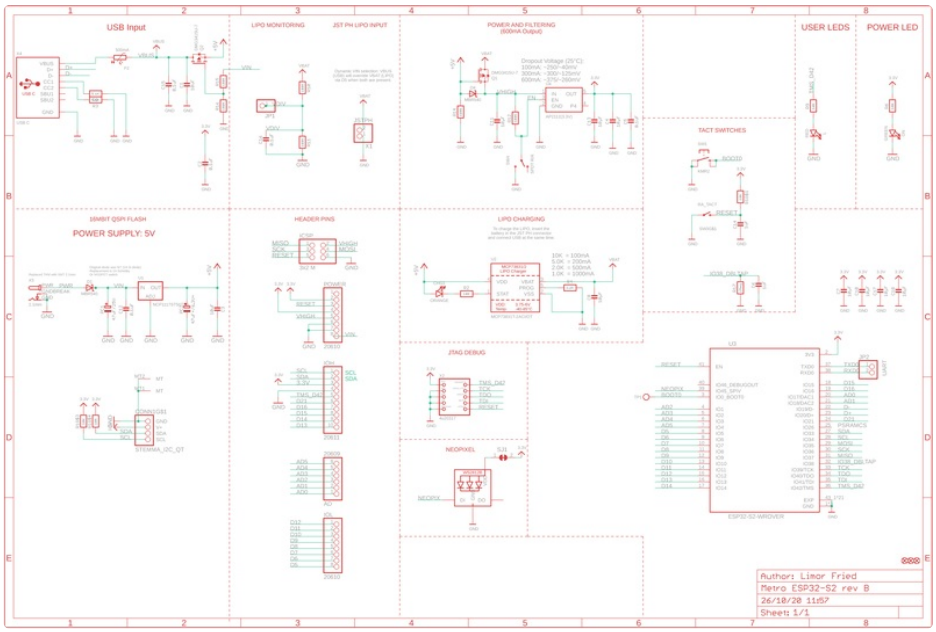
I have not had success programming the chip using gdb commands like `load` or OpenOCD commands like `program_esp`. It's inconvenient but I still use esptool when reprogramming the chip.

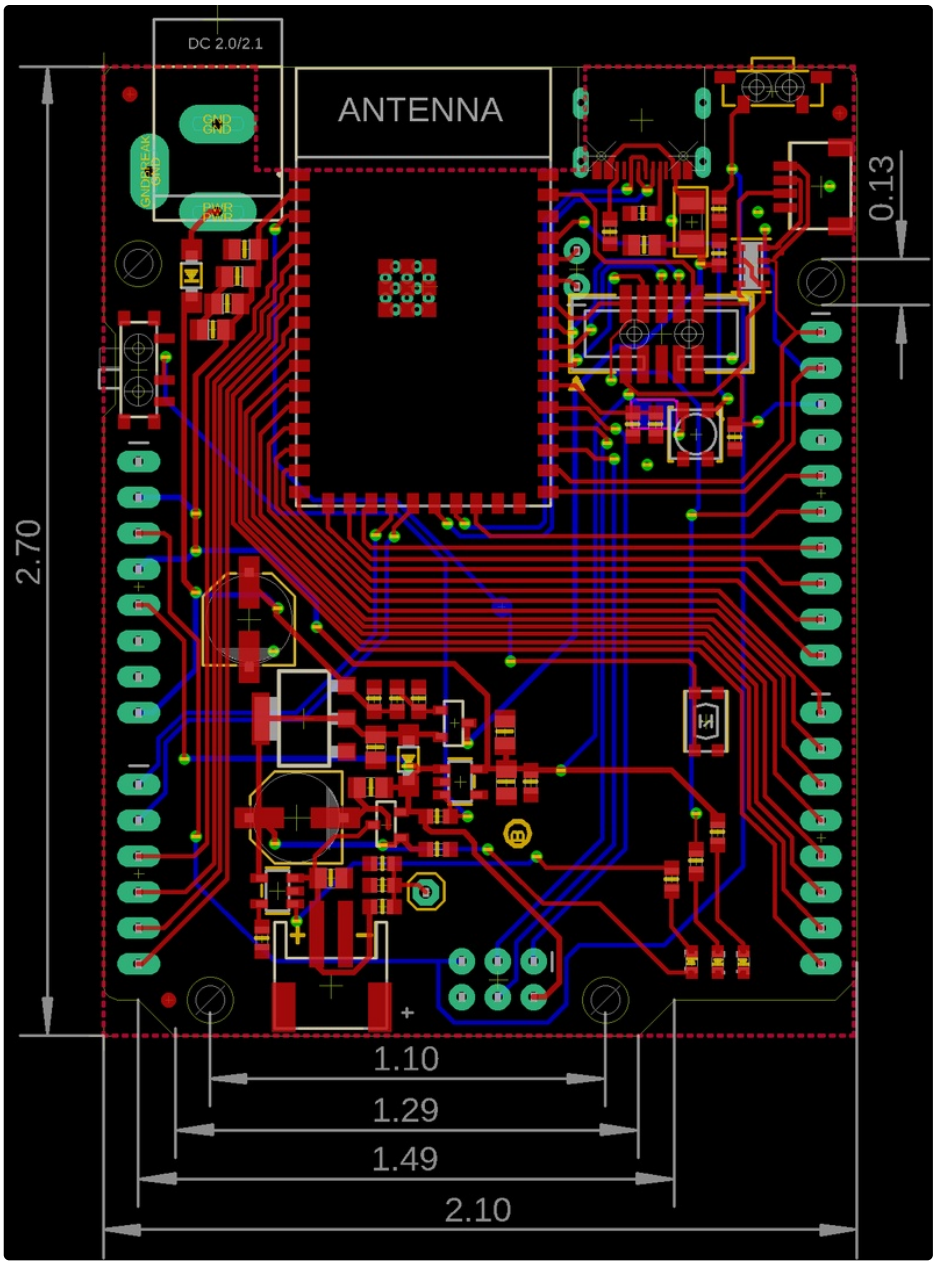
Downloads

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/Oqc)
- 3D Models on GitHub (https://adafru.it/OYA)
- Fritzing object in the Adafruit Fritzing Library (https://adafru.it/Oqd)

Schematic and Fab Print




DC 2.0/2.1
ANTENNA
2.70
0.13
1.10
1.29
1.49
2.10