



Adafruit Qualia ESP32-S3 for RGB-666 Displays

Created by Melissa LeBlanc-Williams



<https://learn.adafruit.com/adafruit-qualia-esp32-s3-for-rgb666-displays>

Last updated on 2023-10-20 10:53:17 AM EDT

Table of Contents

Overview	7
Pinouts	11
<ul style="list-style-type: none">• Microcontroller and WiFi• 40-Pin Display Connector• IO Expander• Stemma QT Connector• Reset and Boot0 Pins• Debug Pin• SPI Pins• Analog Connector/Pins• Buttons• Backlight Jumpers• IO Expander Address Jumpers• Parallel Interface Jumpers	
CircuitPython	16
<ul style="list-style-type: none">• CircuitPython Quickstart	
The CIRCUITPY Drive	19
<ul style="list-style-type: none">• Boards Without CIRCUITPY	
CircuitPython Pins and Modules	20
<ul style="list-style-type: none">• CircuitPython Pins• import board• I2C, SPI, and UART• What Are All the Available Names?• Microcontroller Pin Names• CircuitPython Built-In Modules	
Installing the Mu Editor	26
<ul style="list-style-type: none">• Download and Install Mu• Starting Up Mu• Using Mu	
Creating and Editing Code	28
<ul style="list-style-type: none">• Creating Code• Editing Code• Back to Editing Code...• Naming Your Program File	
Exploring Your First CircuitPython Program	33
<ul style="list-style-type: none">• Imports & Libraries• Setting Up The LED• Loop-de-loops• What Happens When My Code Finishes Running?• What if I Don't Have the Loop?	
Connecting to the Serial Console	36
<ul style="list-style-type: none">• Are you using Mu?• Serial Console Issues or Delays on Linux	

- [Setting Permissions on Linux](#)
- [Using Something Else?](#)

[Interacting with the Serial Console](#) 39

[The REPL](#) 42

- [Entering the REPL](#)
- [Interacting with the REPL](#)
- [Returning to the Serial Console](#)

[CircuitPython Libraries](#) 46

- [The Adafruit Learn Guide Project Bundle](#)
- [The Adafruit CircuitPython Library Bundle](#)
- [Downloading the Adafruit CircuitPython Library Bundle](#)
- [The CircuitPython Community Library Bundle](#)
- [Downloading the CircuitPython Community Library Bundle](#)
- [Understanding the Bundle](#)
- [Example Files](#)
- [Copying Libraries to Your Board](#)
- [Understanding Which Libraries to Install](#)
- [Example: ImportError Due to Missing Library](#)
- [Library Install on Non-Express Boards](#)
- [Updating CircuitPython Libraries and Examples](#)
- [CircUp CLI Tool](#)

[CircuitPython Documentation](#) 57

- [CircuitPython Core Documentation](#)
- [CircuitPython Library Documentation](#)

[Recommended Editors](#) 64

- [Recommended editors](#)
- [Recommended only with particular settings or add-ons](#)
- [Editors that are NOT recommended](#)

[Advanced Serial Console on Windows](#) 65

- [Windows 7 and 8.1](#)
- [What's the COM?](#)
- [Install Putty](#)

[Advanced Serial Console on Mac](#) 69

- [What's the Port?](#)
- [Connect with screen](#)

[Advanced Serial Console on Linux](#) 71

- [What's the Port?](#)
- [Connect with screen](#)
- [Permissions on Linux](#)

[Frequently Asked Questions](#) 75

- [Using Older Versions](#)
- [Python Arithmetic](#)
- [Wireless Connectivity](#)
- [Asyncio and Interrupts](#)
- [Status RGB LED](#)
- [Memory Issues](#)

- [Unsupported Hardware](#)

Troubleshooting

80

- [Always Run the Latest Version of CircuitPython and Libraries](#)
- [I have to continue using CircuitPython 7.x or earlier. Where can I find compatible libraries?](#)
- [Bootloader \(boardnameBOOT\) Drive Not Present](#)
- [Windows Explorer Locks Up When Accessing boardnameBOOT Drive](#)
- [Copying UF2 to boardnameBOOT Drive Hangs at 0% Copied](#)
- [CIRCUITPY Drive Does Not Appear or Disappears Quickly](#)
- [Device Errors or Problems on Windows](#)
- [Serial Console in Mu Not Displaying Anything](#)
- [code.py Restarts Constantly](#)
- [CircuitPython RGB Status Light](#)
- [CircuitPython 7.0.0 and Later](#)
- [CircuitPython 6.3.0 and earlier](#)
- [Serial console showing ValueError: Incompatible .mpy file](#)
- [CIRCUITPY Drive Issues](#)
- [Safe Mode](#)
- [To erase CIRCUITPY: storage.erase_filesystem\(\)](#)
- [Erase CIRCUITPY Without Access to the REPL](#)
- [For the specific boards listed below:](#)
- [For SAMD21 non-Express boards that have a UF2 bootloader:](#)
- [For SAMD21 non-Express boards that do not have a UF2 bootloader:](#)
- [Running Out of File Space on SAMD21 Non-Express Boards](#)
- [Delete something!](#)
- [Use tabs](#)
- [On MacOS?](#)
- [Prevent & Remove MacOS Hidden Files](#)
- [Copy Files on MacOS Without Creating Hidden Files](#)
- [Other MacOS Space-Saving Tips](#)
- [Device Locked Up or Boot Looping](#)

Welcome to the Community!

98

- [Adafruit Discord](#)
- [CircuitPython.org](#)
- [Adafruit GitHub](#)
- [Adafruit Forums](#)
- [Read the Docs](#)

Create Your settings.toml File

107

- [CircuitPython settings.toml File](#)
- [settings.toml File Tips](#)
- [Accessing Your settings.toml Information in code.py](#)

CircuitPython Internet Test

110

- [The settings.toml File](#)

Converting Arduino_GFX init strings to CircuitPython

115

- [Using Arduino_GFX Init Codes](#)
- [Using Init Code Files](#)
- [Script Output](#)

Determining Timings

119

- [Using a Data Sheet](#)
- [Fill in the Settings](#)
- [Experimenting with Settings](#)

- Testing your Settings with CircuitPython

CircuitPython Display Setup 123

- Example TFT_PINS
- Example TFT_TIMINGS
- I/O Expander
- Display Initialization Code
- Sending Initialization Code via I2C IO Expander
- I2C Bus Speed
- Constructing the framebuffer and the display
- Dot clocks

CircuitPython Touch Display Usage 135

- Determining the I2C Address
- Initializing the Touch Controller
- Reading from the Touch Controller
- Example

Qualia S3 RGB-666 with TL021WVC02 2.1" 480x480 Round Display 139

- Initialization Codes
- Timings
- Example

Qualia S3 RGB-666 with TL034WVS05 3.4" 480x480 Square Display 142

- Initialization Codes
- Timings
- Example

Qualia S3 RGB-666 with TL040HDS20 4.0" 720x720 Square Display 146

- Initialization Codes
- Timings
- Example

Qualia S3 RGB-666 with TL032FWV01 3.2" 320x820 Bar Display 149

- Initialization Codes
- Timings
- Example

Arduino IDE Setup 153

Using with Arduino IDE 156

- Blink
- Select ESP32-S2/S3 Board in Arduino IDE
- Launch ESP32-S2/S3 ROM Bootloader
- Load Blink Sketch

WiFi Test 160

- WiFi Connection Test
- Secure Connection Example
- JSON Parsing Demo

Usage with Adafruit IO 171

- Install Libraries
- Adafruit IO Setup
- Code Usage

Arduino Rainbow Demo	179
--------------------------------------	-----

Arduino Touch Display Usage	182
---	-----

- [Determining the I2C Address](#)
- [Initializing the Touch Controller](#)
- [Reading from the Touch Controller](#)
- [Example](#)

Install UF2 Bootloader	184
--	-----

Factory Reset	185
-------------------------------	-----

- [Factory Reset Firmware UF2](#)
- [Factory Reset and Bootloader Repair](#)
- [Download .bin and Enter Bootloader](#)
- [Step 1. Download the factory-reset-and-bootloader.bin file](#)
- [Step 2. Enter ROM bootloader mode](#)
- [The WebSerial ESPTool Method](#)
- [Connect](#)
- [Erase the Contents](#)
- [Program the ESP32-S2/S3](#)
- [The esptool Method \(for advanced users\)](#)
- [Install ESPTool.py](#)
- [Test the Installation](#)
- [Connect](#)
- [Erase the Flash](#)
- [Installing the Bootloader](#)
- [Reset the board](#)
- [Older Versions of Chrome](#)
- [The Flash an Arduino Sketch Method](#)
- [Arduino IDE Setup](#)
- [Load the Blink Sketch](#)

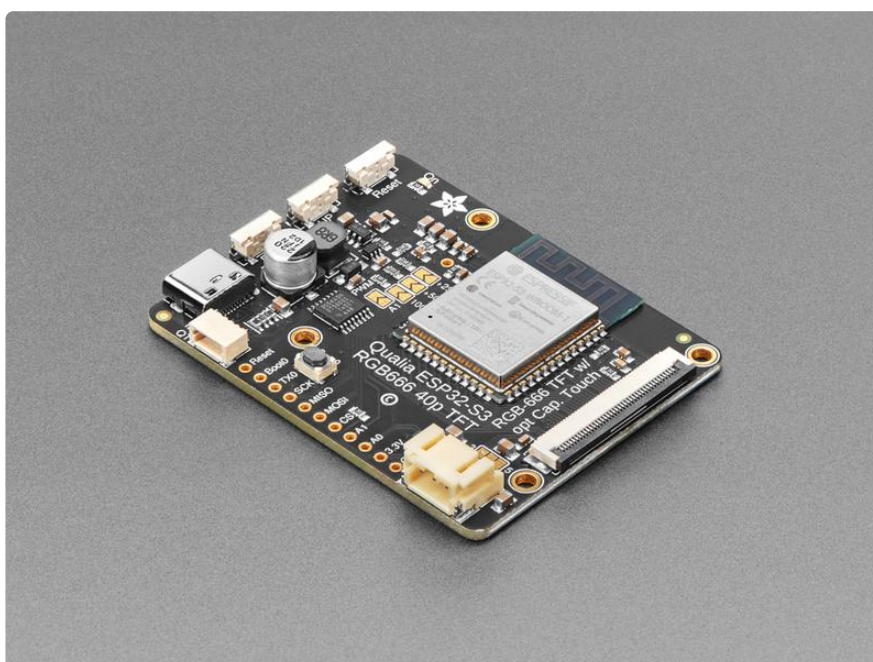
Downloads	198
---------------------------	-----

- [Schematic](#)
- [Fab Print](#)

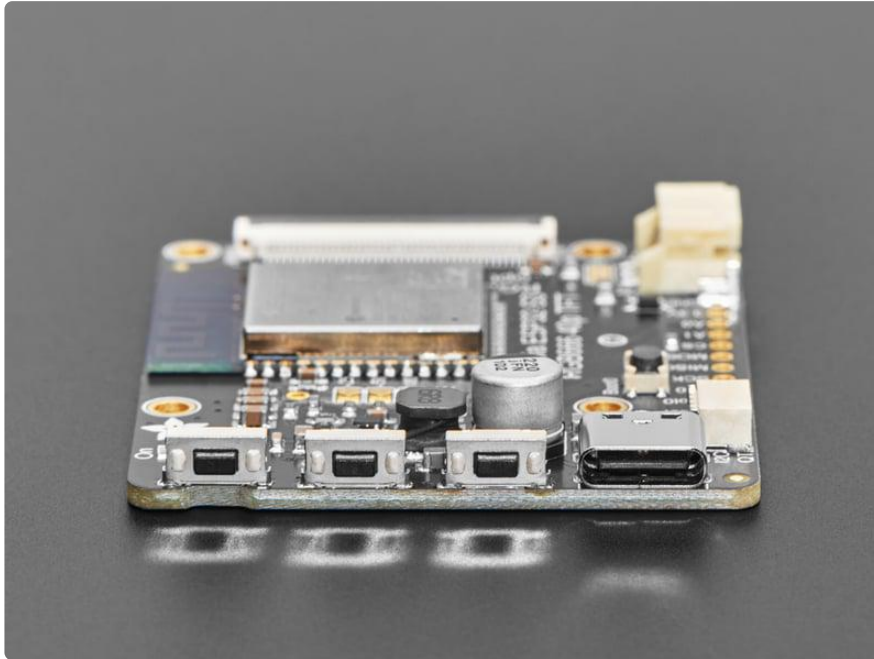
Overview



There are things everyone loves: ice cream, kittens, and honkin' large TFT LCD screens. We're no strangers to small TFT's - [from our itsy 1.14" color display \(\)](#) that graces many-a-TFT-Feather to [our fancy 3.5" 320x480 \(\)](#) breakout screen. But most people who dabble or engineer with microcontrollers know that you sort of 'top out' at 320x480 - that's the largest resolution you can use with every day SPI or 8-bit 8080 interfaces. After that, you're in TTL-interface TFT land, where displays no longer have an internal memory buffer and instead the controller has to continuously write scanline data over a 16, 18 or 24 pin interface.

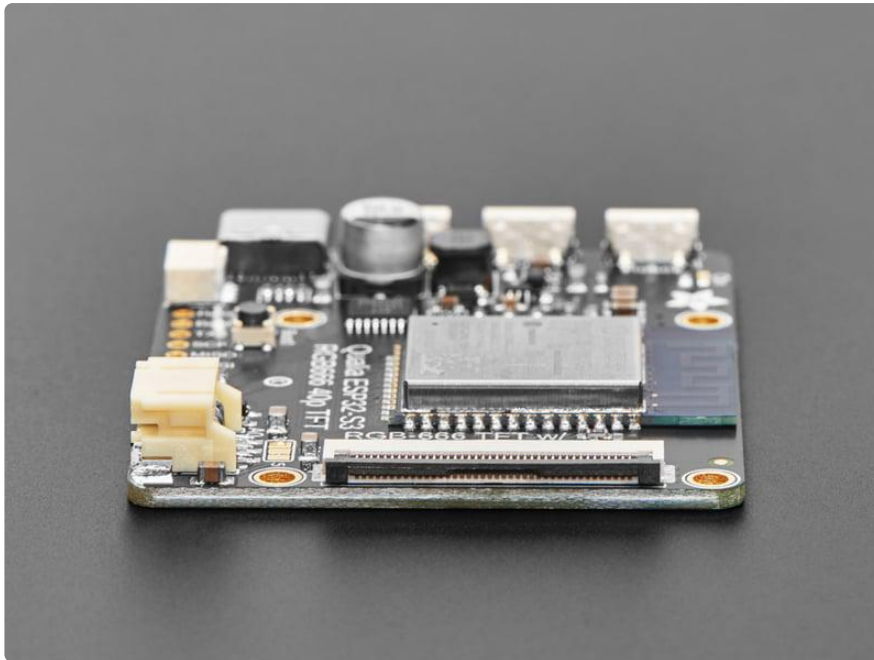


RGB TTL interface TFT displays can get big: they start out at around 4.3" diagonal 480x272, and can get to 800x480, 800x600 or even 720x720. For displays that big, you need a lot of video RAM (800x480 at 24 bit color is just over 1MB), plenty of spare GPIO to dedicate, and a peripheral that will DMA the video RAM out to the display continuously. This is a setup familiar to people working with hefty microcontrollers or microcomputers, the sort of device that run cell phones, or your car's GPS navigation screen. But until now, nearly impossible to use on low cost microcontrollers.



The ESP32-S3 is the first low-cost microcontroller that has a built in peripheral that can drive TTL displays, and it can come with enough PSRAM to buffer those large images. For example, on the Adafruit Qualia ESP32-S3 for TTL RGB-666 Displays, we use a S3 module with 16 MB of Flash and 8 MB of octal PSRAM. Using the built in RGB display peripheral you can display graphics, images, animations or even video (cinepak, natch!) with near-instantaneous updates since the whole screen gets updated about 30 frames per second (FPS).

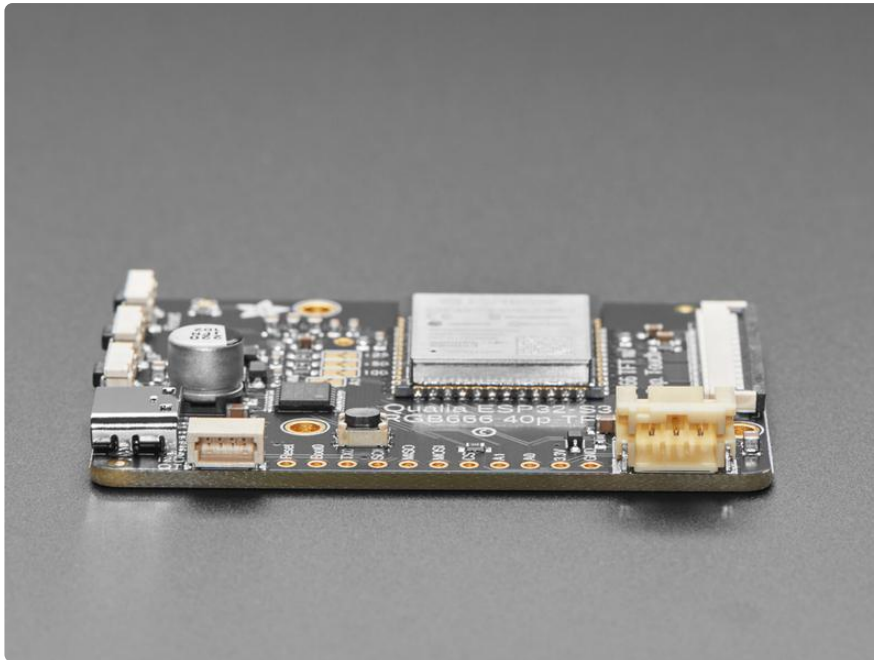
This dev board is designed to make it easy for you to explore displays that use the 'secondary standard' 40-pin RGB-666 connector. This pin order is most commonly seen on square, round and bar displays. [You'll want to compare the display you're using to this datasheet \(\)](#), and if it matches, you'll probably be good! One nice thing about this connector ordering is that it also includes pins for capacitive touch overlay, and we wire those up to the ESP32-S3's I2C port so you can also have touch control with your display.



Don't forget! This is just the development board, a display is not included. Use any RGB-666 pinout display with or without a touch overlay. Note that you will need to program in the driver initialization code, dimensions, and pulse widths in your programming language. Here are some known-working displays that you can use in Arduino and CircuitPython:

- [2.1" 480x480 Round with Capacitive Touch \(\)](#)
- [2.1" 480x480 Round without Touch \(\)](#)
- [4" 720x720 Square with Capacitive Touch \(\)](#)
- [4" 720x720 Round without Touch \(\)](#)
- [4.6" 960x320 Rectangular Bar \(\)](#)

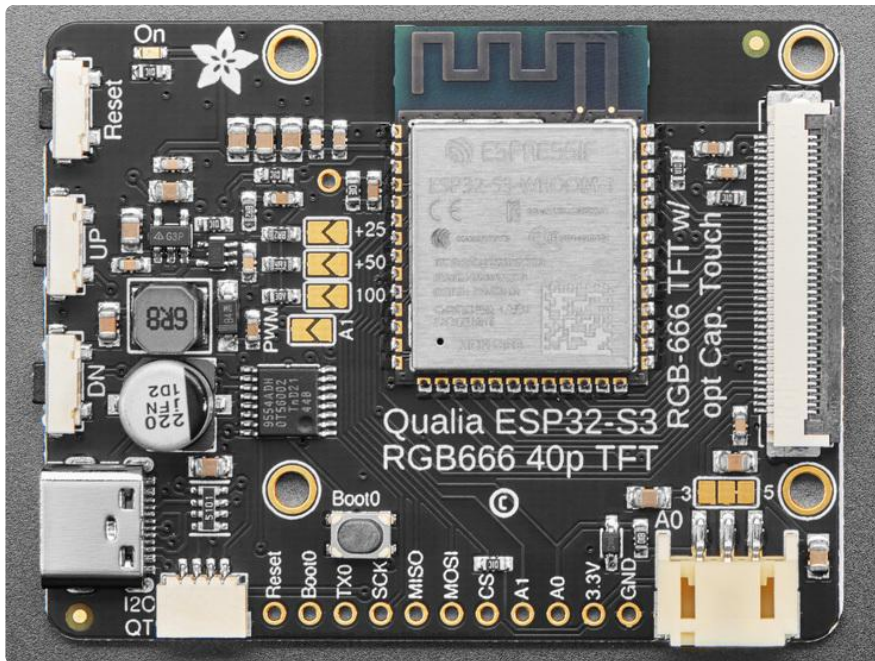
On the Qualia board we have the S3 modules, with 16 pins connected to the TFT for 5-6-5 RGB color, plus HSync, VSync, Data Enable and Pixel Clock. There's a constant current backlight control circuit using the [TPS61169 \(\)](#) which can get up to 30V forward voltage and can be configured for 25mA-200mA in 25mA increments (default is 25mA). Power and programming is provided over a USB C connector, wired to the S3's native USB port. For debugging, the hardware UART TX pin is available as well.



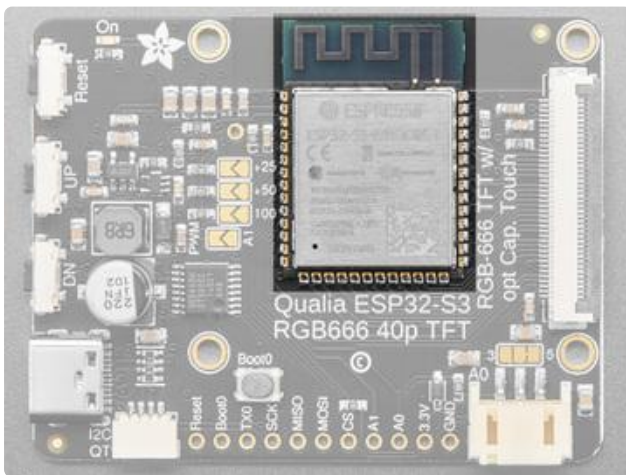
Since almost every GPIO is used, and almost all RGB-666 displays need to be initialized over SPI, we put a [PCA9554](#) () I/O expander on the shared I2C bus. Arduino or CircuitPython can be instructed on how to use the expander to reset and init the display you have if necessary. The remaining expander pins are connected to two right-angle buttons, and the display backlight.

The expander is what lets us have a full 4-pin SPI port and two more analog GPIO pins - [enough to wire up an MMC in 1-wire SDIO mode along with an I2S amplifier to make an A/V playback demo](#) (). Maybe we can even eat ice cream while watching kitten vids! There is also the shared I2C port, we provide a Stemma QT / Qwiic port for easy addition of any sensor or device you like.

Pinouts



Microcontroller and WiFi

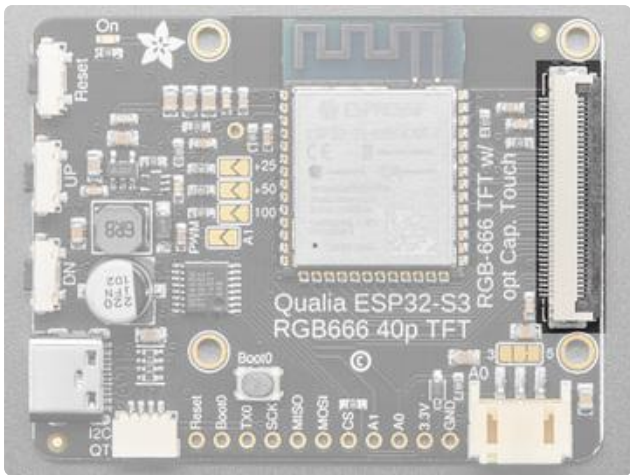


The main processor chip is the Espressif ESP32-S3 with 3.3v logic/power. It has 16MB of Flash and 8MB of RAM.

The ESP32-S3 comes with WiFi and Bluetooth LE baked right in, though CircuitPython only supports WiFi at this time, not BLE on the S3 chip

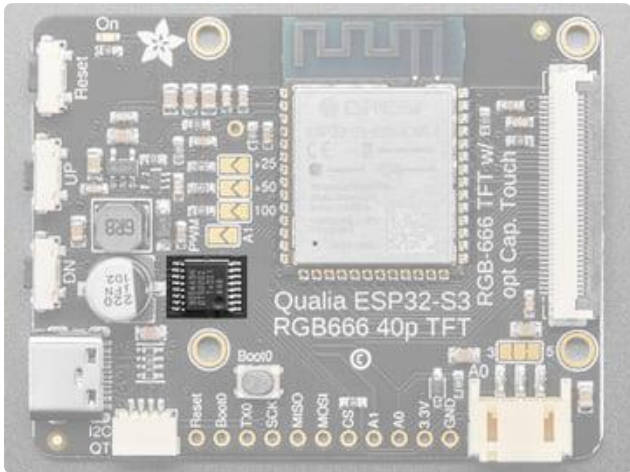
40-Pin Display Connector

Not all 40-pin displays have the power pins in the same place. Hooking up a non RGB666 display with the Qualia S3 risks damaging the display.



There is a 40-pin display connector to connect your display. Displays should be connected with the pins of the cable down towards the board and the colored side facing you.

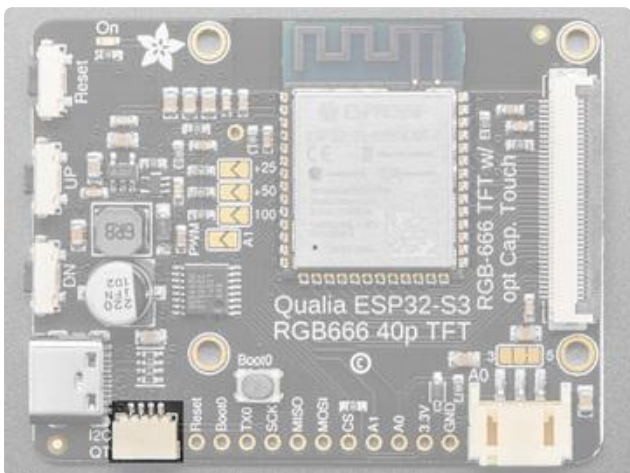
IO Expander



The Qualia S3 includes a PCA9554 IO Expander. The IO Expander is connected via the I2C bus. The main purpose of the expander is to add additional pins to communicate with the display.

The default address of the IO expander is 0x3F, but it can be changed by soldering jumpers on the reverse side in case it interferes with another I2C device.

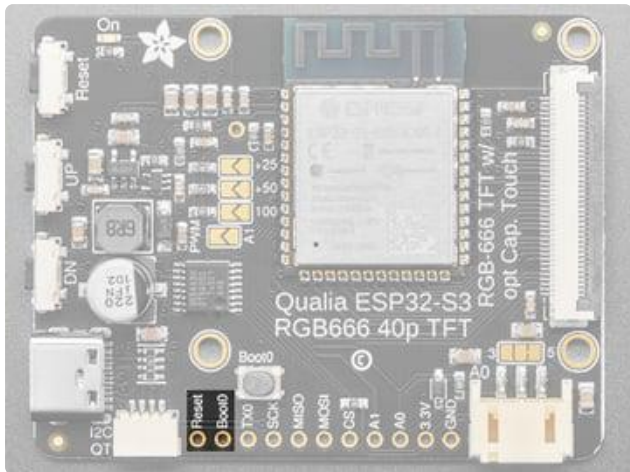
Stemma QT Connector



There is a 4-pin Stemma QT connector on the left. The I2C has pullups to 3.3V power.

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

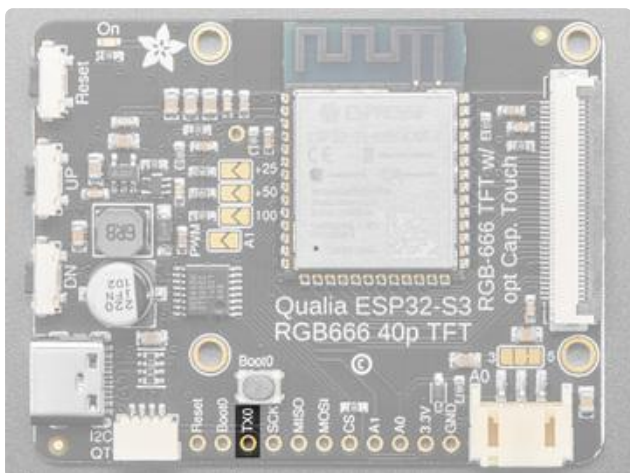
Reset and Boot0 Pins



Reset is the Reset pin. Tie to ground to manually reset the ESP32-S3.

Tying Boot0 to ground while resetting will place the ESP32-S3 in ROM bootloader mode.

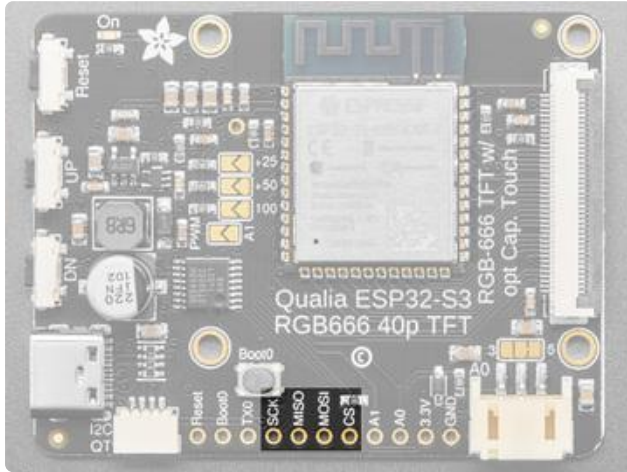
Debug Pin



If you'd like to do lower level debugging, we have the ESP32-S3's TXD0 debug pin exposed as TX0 to view messages.

To read, you would connect a Serial UART cable Receive connection here and the cable ground connection to the GND pin.

SPI Pins



The SPI pins of the ESP32-S3 are exposed for communication with other SPI hardware.

Each of these pins can alternatively be used for digital I/O:

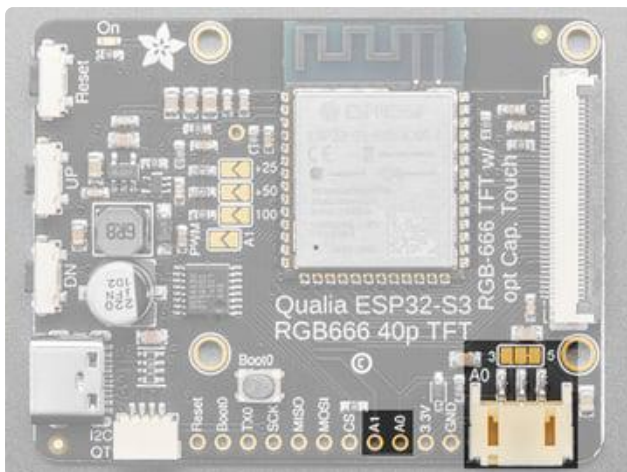
SCK is connected to `board.TFT_SCK` Arduino [5](#).

MISO is connected to `board.TFT_MISO` Arduino [6](#).

MOSI is connected to `board.TFT_MOSI` Arduino [7](#).

CS is connected to `board.TFT_CS` Arduino [15](#) and includes a 10K Pull-up resistor.

Analog Connector/Pins



On the bottom side towards the right, there is a connector labeled A0. This is a 3-pin JST analog connector for sensors, NeoPixels, or analog output or input.

For the JST connected, there is a jumper above that can be cut and soldered to use 3V instead of 5V.

Along the bottom there are also pins labeled `A0` and `A1`.

Each of these pins can be used for analog inputs or digital I/O.

Buttons

There are three buttons along the left side of the Qualia S3.

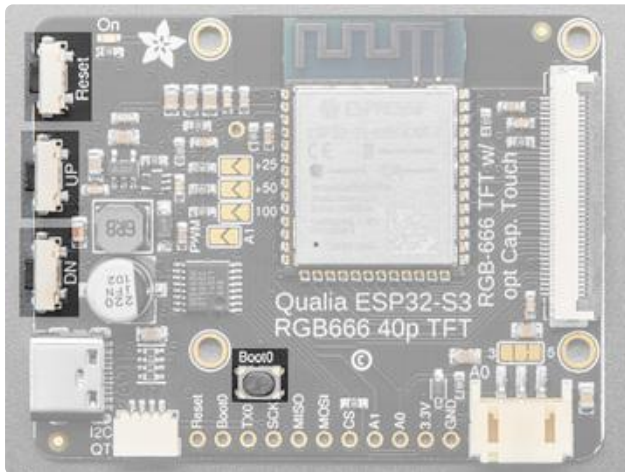
The Reset button is located in the top position. Click it once to re-start your firmware. Click it again after about a half second to enter bootloader mode.

The UP button is located in the middle and is connected to the IO expander

The DN button, or Down button, is located on the bottom and is connected to the IO expander.

The expander implements a light pullup for each of the buttons and pressing either of them pulls the input low.

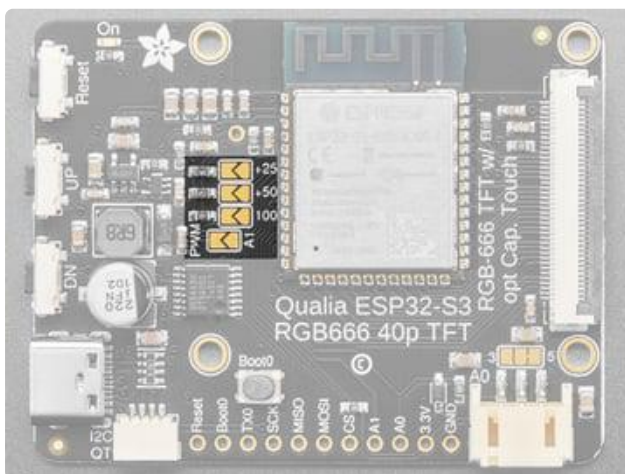
The Boot0 button is located between the up button and the Microcontroller. Hold it while pressing reset to enter ROM Bootloader mode.



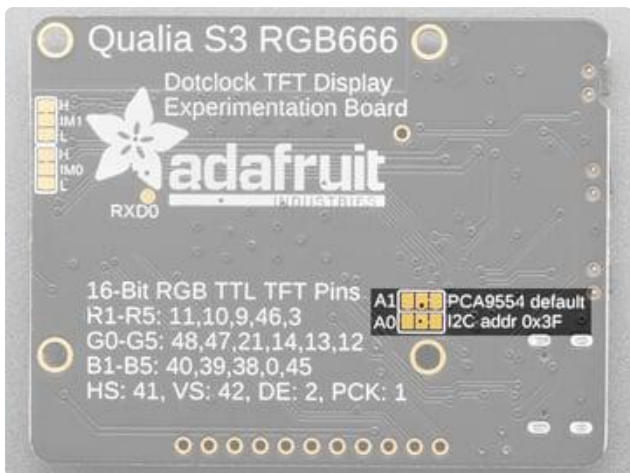
Backlight Jumpers

Soldering the bottom PWM jumper allows using Pin **A1** to control the backlight of the display.

By default, 25mA is provided to the backlight, but additional amperage can be set by soldering the top jumpers to provide up to 200mA if needed.

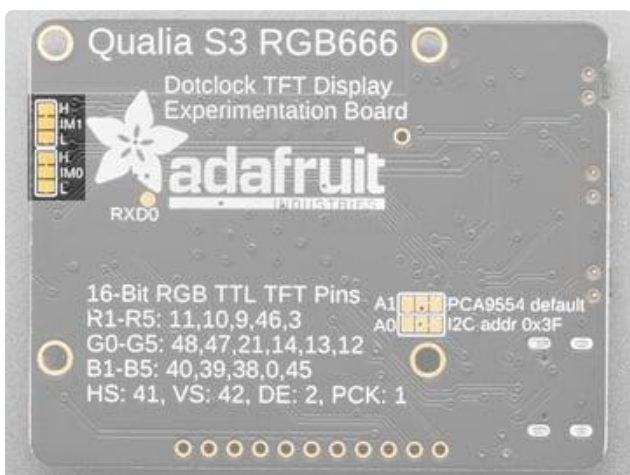


IO Expander Address Jumpers



On the reverse, are a couple of solderable jumpers to change the I2C address of the IO Expander. By default, both jumpers are set to high, providing a default address of 0x3F. However, it can be set between 0x3B-0x3F.

Parallel Interface Jumpers



The IM0 and IM1 jumpers are for selecting the mode of the parallel interface for the display. The default selection should work for most displays.

CircuitPython

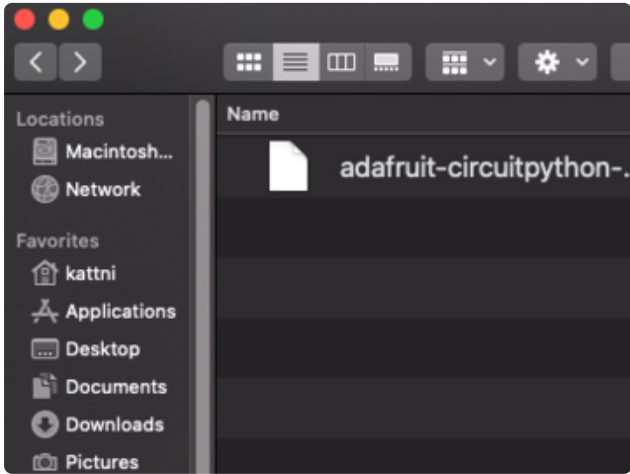
[CircuitPython \(\)](#) is a derivative of [MicroPython \(\)](#) designed to simplify experimentation and education on low-cost microcontrollers. It makes it easier than ever to get prototyping by requiring no upfront desktop software downloads. Simply copy and edit files on the CIRCUITPY drive to iterate.

CircuitPython Quickstart

Follow this step-by-step to quickly get CircuitPython running on your board.

This microcontroller requires the latest unstable (development) release of CircuitPython. Click below to visit the downloads page on circuitpython.org for your board. Then, Browse S3 under Absolute Newest.

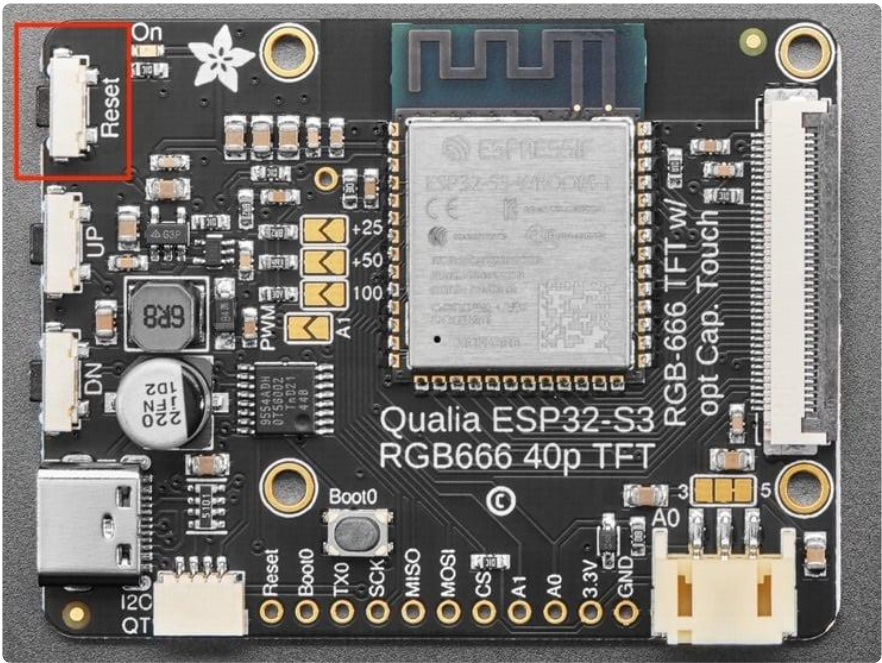
Download the latest version of CircuitPython for this board via circuitpython.org



Click the link above to download the latest CircuitPython UF2 file.

Save it wherever is convenient for you.

The Qualia S3 does not have a RGB status LED



Plug your board into your computer, using a known-good data-sync cable, directly, or via an adapter if needed.

Double-click the reset button (highlighted in red above), and you will see the RGB status LED(s) turn green (highlighted in green above). If you see red, try another port, or if you're using an adapter or hub, try without the hub, or different adapter or hub.

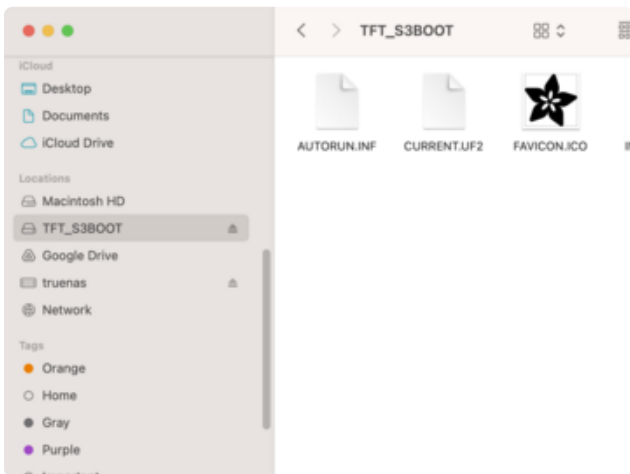
This board does not have a Neopixel, so you will need to just double tap the reset button.

For this board, tap reset and wait about a half a second and then tap reset again.

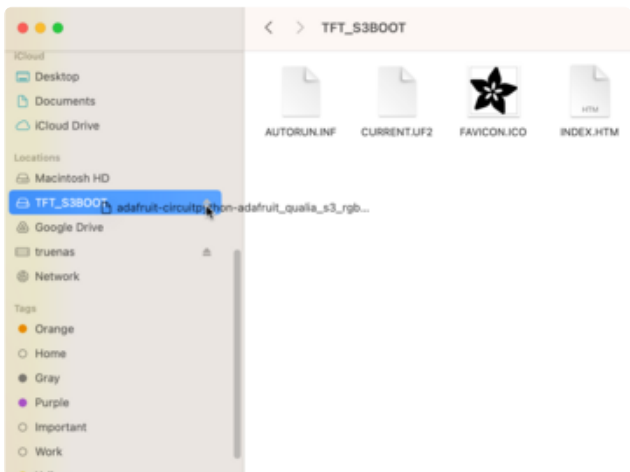
Some boards may not have a UF2 bootloader installed. If double-clicking does not work, follow the instructions on the "Install UF2 Bootloader" page in this guide.

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

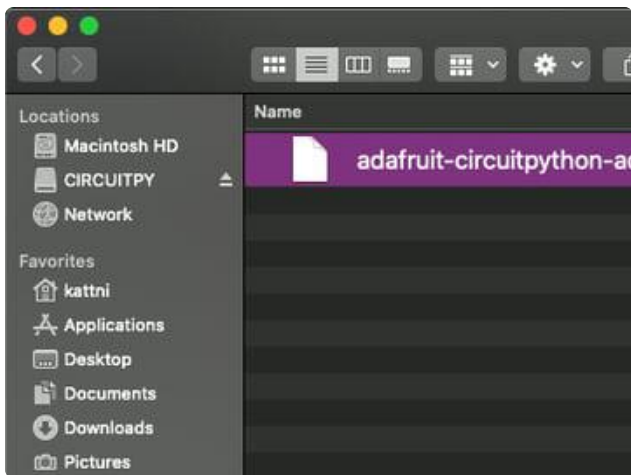
A lot of people end up using charge-only USB cables and it is very frustrating! Make sure you have a USB cable you know is good for data sync.



You will see a new disk drive appear called TFT_S3BOOT.



Drag the adafruit_circuitpython_etc.uf2 file to TFT_S3BOOT.



The BOOT drive will disappear and a new disk drive called CIRCUIPTY will appear.

That's it!

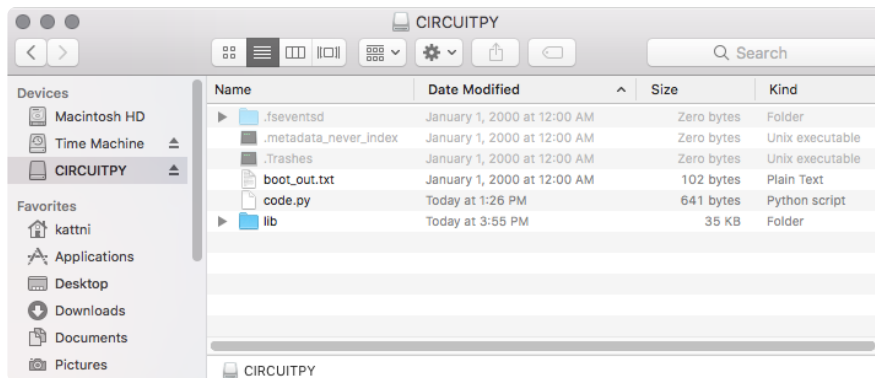
The CIRCUIPTY Drive

When CircuitPython finishes installing, or you plug a CircuitPython board into your computer with CircuitPython already installed, the board shows up on your computer as a USB drive called CIRCUIPTY.

The CIRCUIPTY drive is where your code and the necessary libraries and files will live. You can edit your code directly on this drive and when you save, it will run automatically. When you create and edit code, you'll save your code in a `code.py` file located on the CIRCUIPTY drive. If you're following along with a Learn guide, you can paste the contents of the tutorial example into `code.py` on the CIRCUIPTY drive and save it to run the example.

With a fresh CircuitPython install, on your CIRCUIPTY drive, you'll find a `code.py` file containing `print("Hello World!")` and an empty `lib` folder. If your CIRCUIPTY drive does not contain a `code.py` file, you can easily create one and save it to the drive. CircuitPython looks for `code.py` and executes the code within the file automatically when the board starts up or resets. Following a change to the contents of CIRCUIPTY, such as making a change to the `code.py` file, the board will reset, and the code will be run. You do not need to manually run the code. This is what makes it so easy to get started with your project and update your code!

Note that all changes to the contents of CIRCUIPTY, such as saving a new file, renaming a current file, or deleting an existing file will trigger a reset of the board.



Boards Without CIRCUITPY

CircuitPython is available for some microcontrollers that do not support native USB. Those boards cannot present a CIRCUITPY drive. This includes boards using ESP32 or ESP32-C3 microcontrollers.

On these boards, there are alternative ways to transfer and edit files. You can use the [Thonny editor](#) (), which uses hidden commands sent to the REPL to read and write files. Or you can use the CircuitPython web workflow, introduced in Circuitpython 8. The web workflow provides browser-based WiFi access to the CircuitPython filesystem. These guides will help you with the web workflow:

- [CircuitPython on ESP32 Quick Start](#) ()
- [CircuitPython Web Workflow Code Editor Quick Start](#) ()

CircuitPython Pins and Modules

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

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

CircuitPython Pins

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

`import board`

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

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

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

Here is the output for the QT Py SAMD21. You may have a different board, and this list will vary, based on the board.

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

The following pins have labels on the physical QT Py SAMD21 board: A0, A1, A2, A3, SDA, SCL, TX, RX, SCK, MISO, and MOSI. You see that there are many more entries available in `board` than the labels on the QT Py.

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

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

On the flip side, there may be multiple names for one pin. For example, on the QT Py SAMD21, pin A0 is labeled on the physical board silkscreen, but it is available in CircuitPython as both `A0` and `D0`. For more information on finding all the names for a given pin, see the [What Are All the Available Pin Names? \(\)](#) section below.

The results of `dir(board)` for CircuitPython compatible boards will look similar to the results for the QT Py SAMD21 in terms of the pin names, e.g. A0, D0, etc. However, some boards, for example, the Metro ESP32-S2, have different styled pin names. Here is the output for the Metro ESP32-S2.

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

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

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

I2C, SPI, and UART

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

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

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

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

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

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

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

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

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

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

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

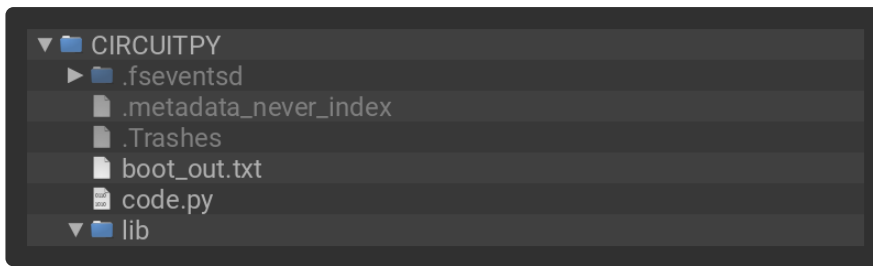
What Are All the Available Names?

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

On a microcontroller board running CircuitPython, first, connect to the serial console.

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

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



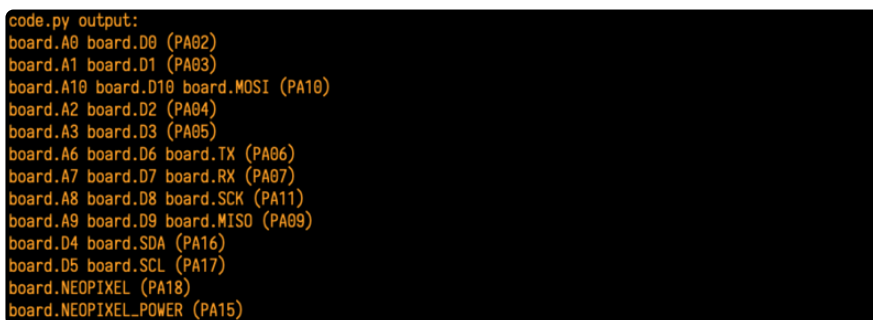
```
# SPDX-FileCopyrightText: 2020 anecddata for Adafruit Industries
# SPDX-FileCopyrightText: 2021 Neradoc for Adafruit Industries
# SPDX-FileCopyrightText: 2021-2023 Kattni Rembor for Adafruit Industries
# SPDX-FileCopyrightText: 2023 Dan Halbert for Adafruit Industries
#
# SPDX-License-Identifier: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

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

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

for pins in sorted(board_pins):
    print(pins)
```

Here is the result when this script is run on QT Py SAMD21:



Each line represents a single pin. Find the line containing the pin name that's labeled on the physical board, and you'll find the other names available for that pin. For example, the first pin on the board is labeled A0. The first line in the output is `board.A0 board.D0 (PA02)`. This means that you can access pin A0 in CircuitPython using both `board.A0` and `board.D0`.

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

You'll notice there are two "pins" that aren't labeled on the board but appear in the list: `board.NEOPIXEL` and `board.NEOPIXEL_POWER`. Many boards have several of these special pins that give you access to built-in board hardware, such as an LED or an on-board sensor. The QT Py SAMD21 only has one on-board extra piece of hardware, a NeoPixel LED, so there's only the one available in the list. But you can also control whether or not power is applied to the NeoPixel, so there's a separate pin for that.

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

Microcontroller Pin Names

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

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

CircuitPython Built-In Modules

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

There are two options for this. You can check the [support matrix \(\)](#), and search for your board by name. Or, you can use the REPL.

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

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

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

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

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



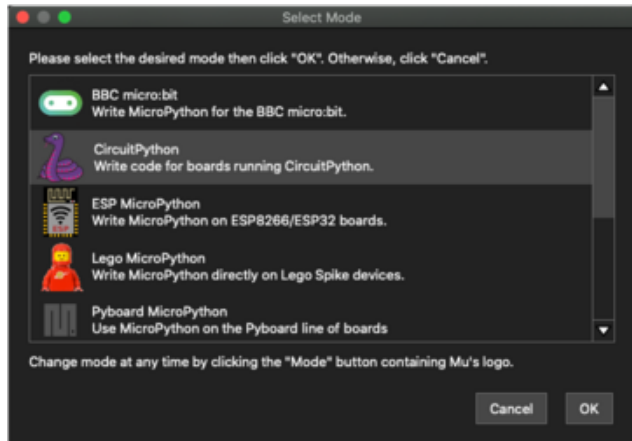
Download Mu from <https://codewith.mu> ().

Click the Download link for downloads and installation instructions.

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

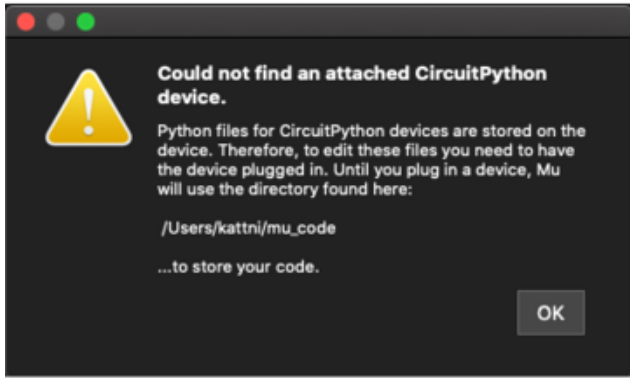
Windows users: due to the nature of MSI installers, please remove old versions of Mu before installing the latest version.

Starting Up Mu



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

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

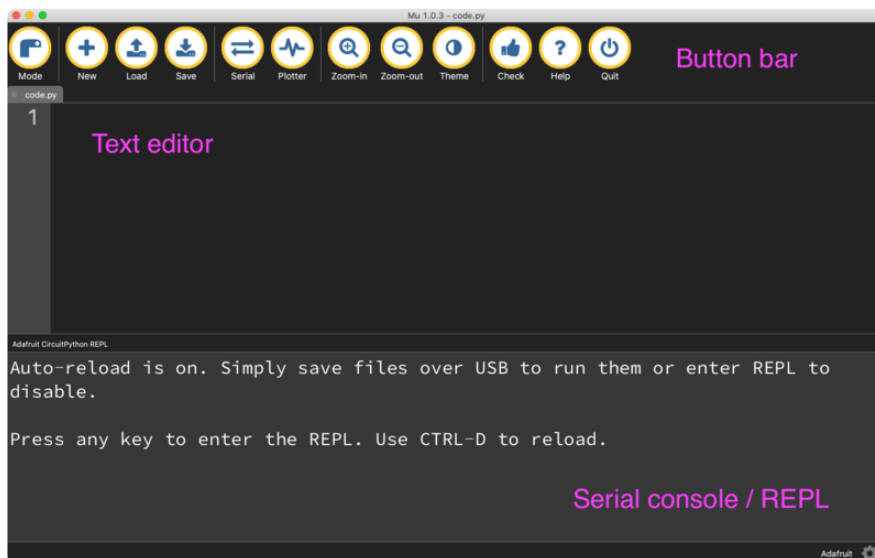


Mu attempts to auto-detect your board on startup, so if you do not have a CircuitPython board plugged in with a CIRCUITPY drive available, Mu will inform you where it will store any code you save until you plug in a board.

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

Using Mu

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



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

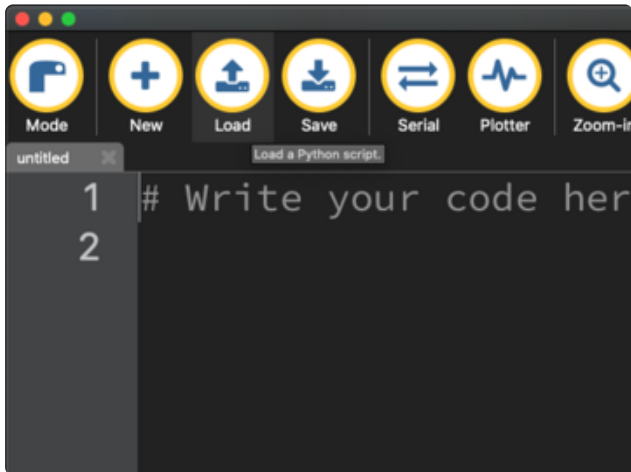
Creating and Editing Code

One of the best things about CircuitPython is how simple it is to get code up and running. This section covers how to create and edit your first CircuitPython program.

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

If you don't or can't use Mu, there are a number of other editors that work quite well. The [Recommended Editors page \(\)](#) has more details. Otherwise, make sure you do "Eject" or "Safe Remove" on Windows or "sync" on Linux after writing a file if you aren't using Mu. (This is not a problem on MacOS.)

Creating Code



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

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

Copy and paste the following code into your editor:

```
import board
import digitalio
import time

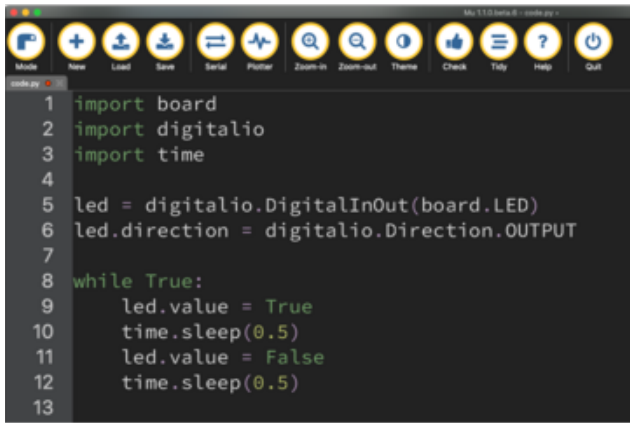
led = digitalio.DigitalInOut(board.LED)
led.direction = digitalio.Direction.OUTPUT

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

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

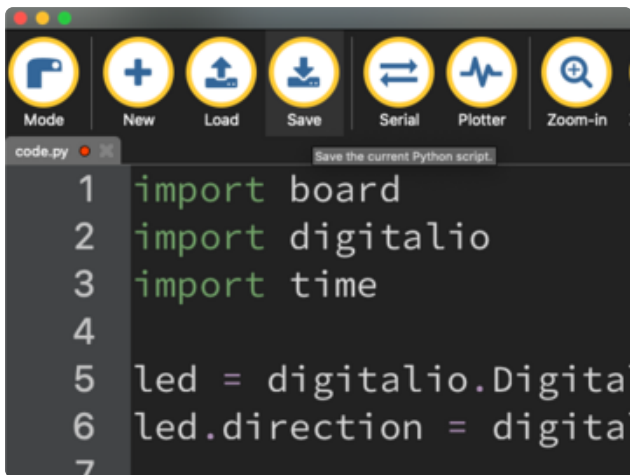
If you're using a KB2040, QT Py or a Trinkey, please download the [NeoPixel blink example \(\)](#).

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.



```
1 import board
2 import digitalio
3 import time
4
5 led = digitalio.DigitalInOut(board.LED)
6 led.direction = digitalio.Direction.OUTPUT
7
8 while True:
9     led.value = True
10    time.sleep(0.5)
11    led.value = False
12    time.sleep(0.5)
13
```

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



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

Save the code.py file on your CIRCUITPY drive.

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

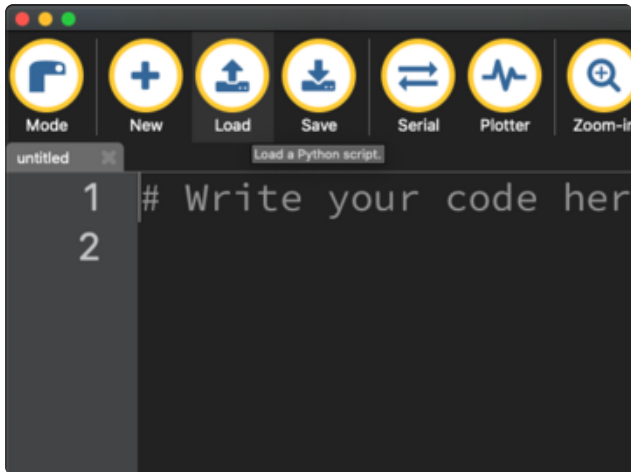
Congratulations, you've just run your first CircuitPython program!

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

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

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

Editing Code



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

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

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

There's one warning before you continue...

Don't click reset or unplug your board!

The CircuitPython code on your board detects when the files are changed or written and will automatically re-start your code. This makes coding very fast because you save, and it re-runs. If you unplug or reset the board before your computer finishes writing the file to your board, you can corrupt the drive. If this happens, you may lose the code you've written, so it's important to backup your code to your computer regularly.

There are a couple of ways to avoid filesystem corruption.

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

Check out the [Recommended Editors page \(\)](#) 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 \(\)](#) 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.

Exploring Your First CircuitPython Program

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

Here is the original code again:

```
import board
import digitalio
import time

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

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

Imports & Libraries

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

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

The `import` statements tells the board that you're going to use a particular library or module in your code. In this example, you imported three modules: `board`, `digitalio`, and `time`. All three of these modules are built into CircuitPython, so no separate library files are needed. That's one of the things that makes this an excellent first example. You don't need anything extra to make it work!

These three modules each have a purpose. The first one, `board`, gives you access to the hardware on your board. The second, `digitalio`, lets you access that hardware as inputs/outputs. The third, `time`, lets you control the flow of your code in multiple ways, including passing time by 'sleeping'.

Setting Up The LED

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

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

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

Loop-de-loops

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

Inside our loop, you have four items:

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

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

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

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

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

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

What Happens When My Code Finishes Running?

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

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

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

What if I Don't Have the Loop?

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

you are writing a simple program that doesn't seem to work, you may need to add a loop to the end so the program doesn't exit.

The simplest loop would be:

```
while True:  
    pass
```

And remember - you can press CTRL+C to exit the loop.

See also the [Behavior section in the docs \(\)](#).

Connecting to the Serial Console

One of the staples of CircuitPython (and programming in general!) is something called a "print statement". This is a line you include in your code that causes your code to output text. A print statement in CircuitPython (and Python) looks like this:

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

This line in your code.py would result in:

```
Hello, world!
```

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

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

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

Are you using Mu?

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

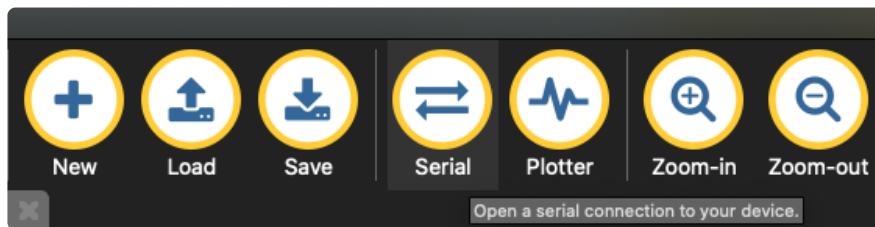


First, make sure your CircuitPython board is plugged in.

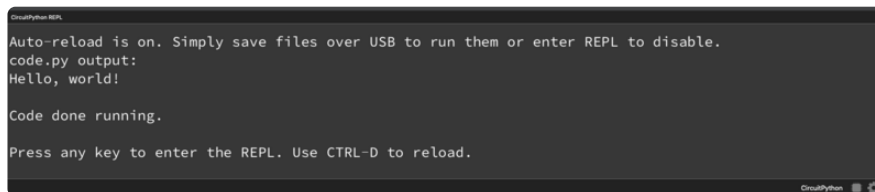
If you open Mu without a board plugged in, you may encounter the error seen here, letting you know no CircuitPython board was found and indicating where your code will be stored until you plug in a board.

If you are using Windows 7, make sure you installed the drivers ().

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



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



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

Serial Console Issues or Delays on Linux

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

To remove `modemmanager`, type the following command at a shell:

```
sudo apt purge modemmanager
```

Setting Permissions on Linux

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



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

```
sudo adduser $USER dialout
```

After running the command above, reboot your machine to gain access to the group. On other Linux distributions, the group you need may be different. See the [Advanced Serial Console on Linux \(\)](#) for details on how to add yourself to the right group.

Using Something Else?

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

Windows requires you to download a terminal program. [Check out the Advanced Serial Console on Windows page for more details. \(\)](#)

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

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

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

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

Code done running.

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

Interacting with the Serial Console

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

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

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

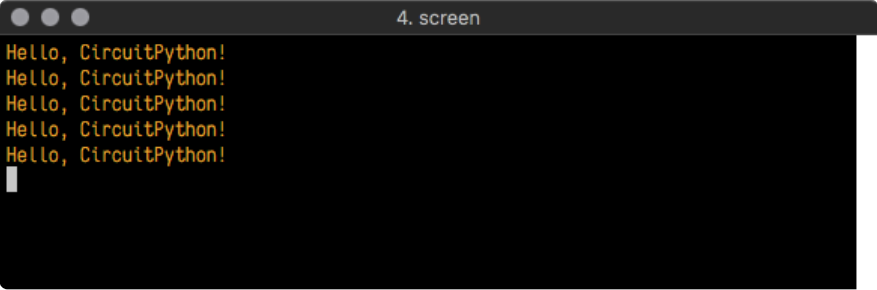
```
import board
import digitalio
import time

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

while True:
    print("Hello, CircuitPython!")
    led.value = True
    time.sleep(1)
    led.value = False
    time.sleep(1)
```

Save your file.

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



```
4. screen
Hello, CircuitPython!
Hello, CircuitPython!
Hello, CircuitPython!
Hello, CircuitPython!
Hello, CircuitPython!
|
```

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

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

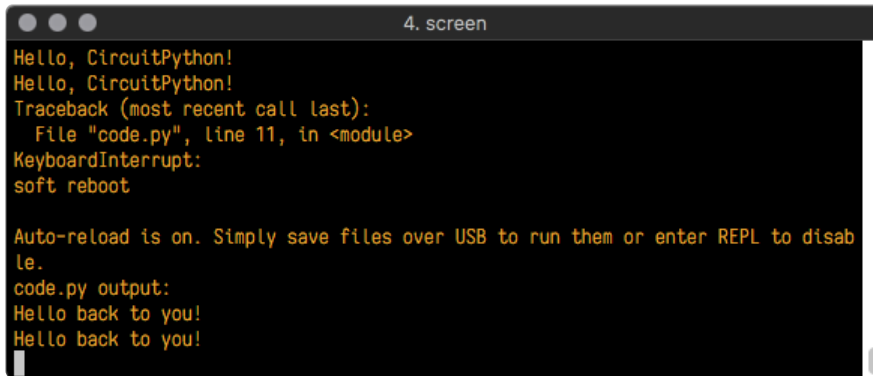
```

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

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

```

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



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

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

```

import board
import digitalio
import time

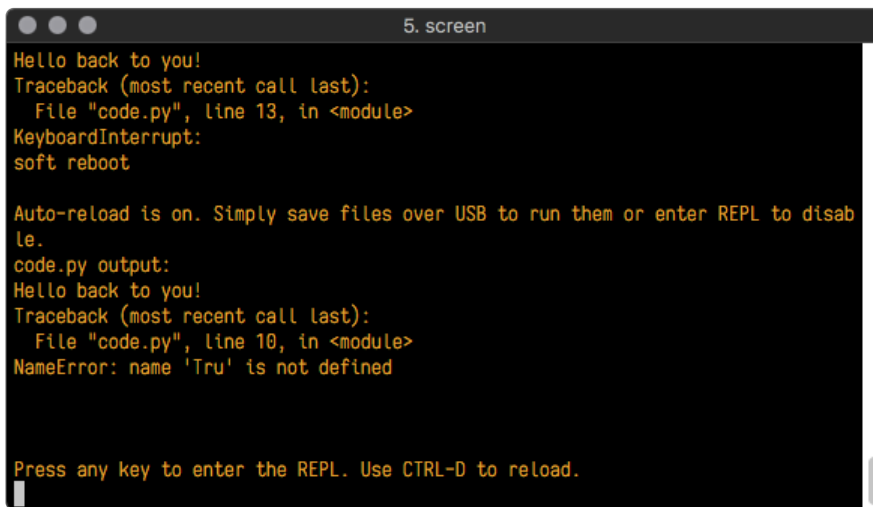
led = digitalio.DigitalInOut(board.LED)
led.direction = digitalio.Direction.OUTPUT

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

```

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

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



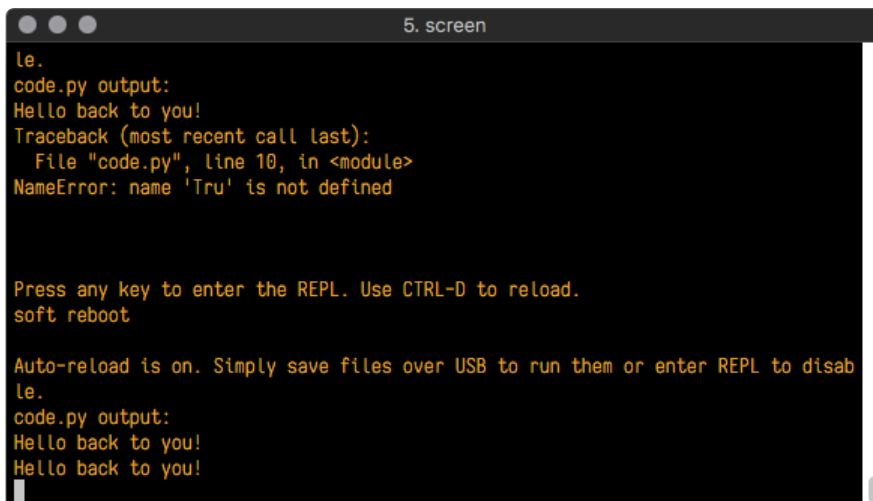
```
5. screen
Hello back to you!
Traceback (most recent call last):
  File "code.py", line 13, in <module>
KeyboardInterrupt:
soft reboot

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

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

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

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



```
5. screen
le.
code.py output:
Hello back to you!
Traceback (most recent call last):
  File "code.py", line 10, in <module>
NameError: name 'Tru' is not defined

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

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

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

The serial console will display any output generated by your code. Some sensors, such as a humidity sensor or a thermistor, receive data and you can use print

statements to display that information. You can also use print statements for troubleshooting, which is called "print debugging". Essentially, if your code isn't working, and you want to know where it's failing, you can put print statements in various places to see where it stops printing.

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

The REPL

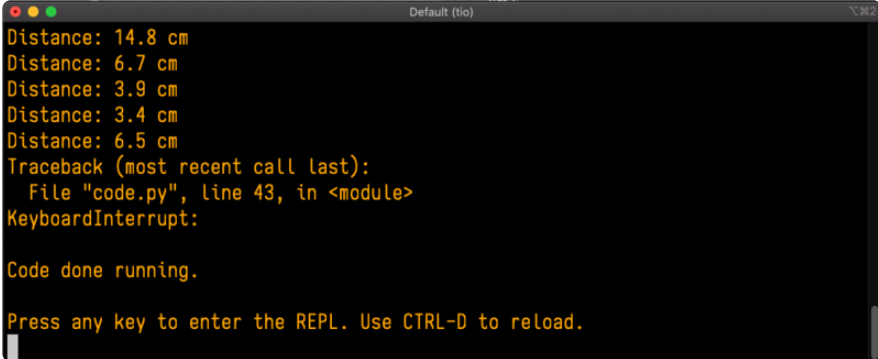
The other feature of the serial connection is the Read-Evaluate-Print-Loop, or REPL. The REPL allows you to enter individual lines of code and have them run immediately. It's really handy if you're running into trouble with a particular program and can't figure out why. It's interactive so it's great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

Entering the REPL

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

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

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

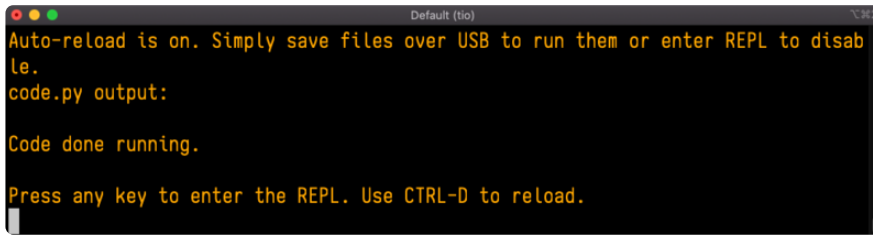


```
Distance: 14.8 cm
Distance: 6.7 cm
Distance: 3.9 cm
Distance: 3.4 cm
Distance: 6.5 cm
Traceback (most recent call last):
  File "code.py", line 43, in <module>
KeyboardInterrupt:

Code done running.

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

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

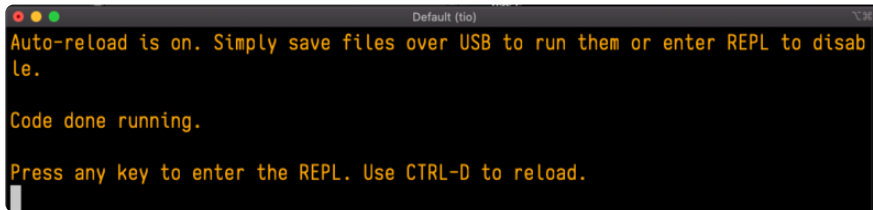


```
Default (tio)
Auto-reload is on. Simply save files over USB to run them or enter REPL to disable.
code.py output:

Code done running.

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

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

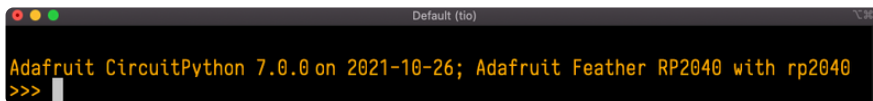


```
Default (tio)
Auto-reload is on. Simply save files over USB to run them or enter REPL to disable.

Code done running.

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

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



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

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

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



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

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

This is followed by the CircuitPython prompt.



```
>>>
```

Interacting with the REPL

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

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

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

Then press enter. You should then see a message.

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

Visit circuitpython.org for more information.

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

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

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

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

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

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

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

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

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

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

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

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

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

Then press enter.

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

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

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

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

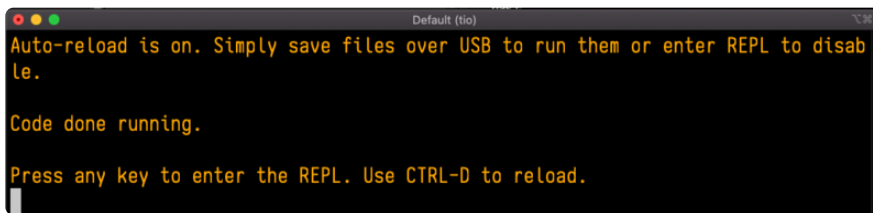
Try typing more into the REPL to see what happens!

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

Returning to the Serial Console

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

You can return to the REPL at any time!



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

Code done running.

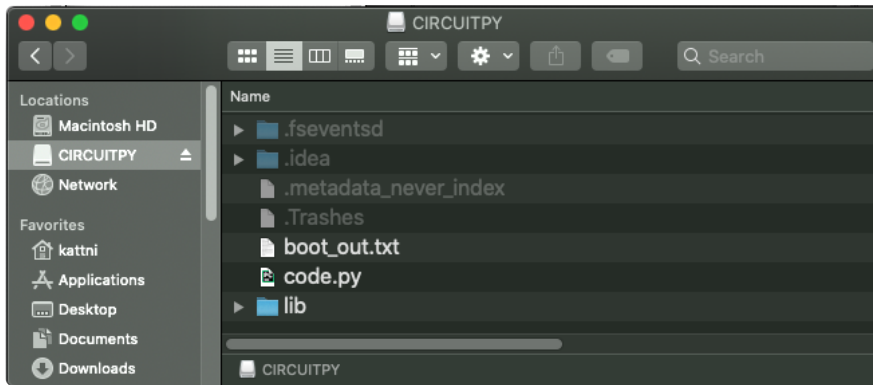
Press any key to enter the REPL. Use CTRL-D to reload.
```

CircuitPython Libraries

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

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

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



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

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

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

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

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

The Adafruit Learn Guide Project Bundle

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

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

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

Circuit Playground Express: Piano in the Key of Lime > Piano in the Key of Lime

Piano in the Key of Lime

Now we'll take everything we learned and put it together!

Be sure to save your current code.py if you've changed anything you'd like to keep. Download the following file. Rename it to code.py and save it to your Circuit Playground Express.

[Download Project Bundle](#) [Copy Code](#)

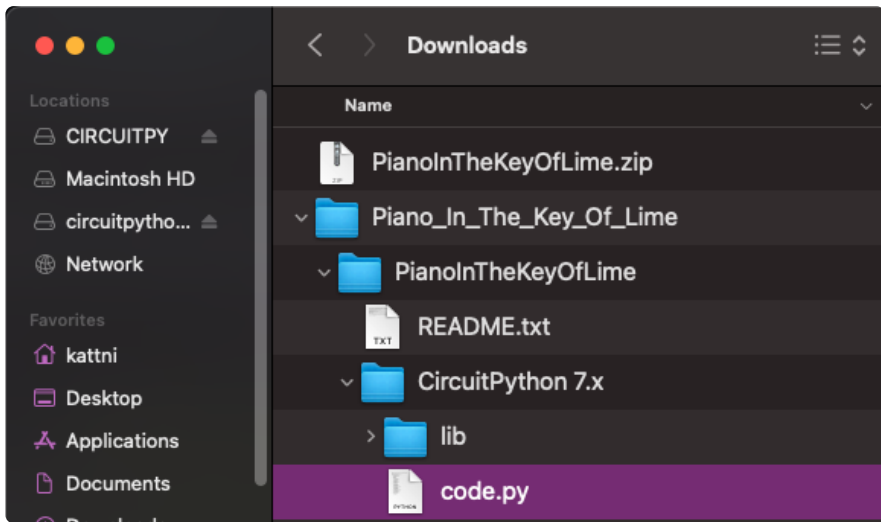
```
# SPDX-FileCopyrightText: 2017 Kattni Rembor for Adafruit Industries
#
# SPDX-License-Identifier: MIT

from adafruit_circuitplayground import cp

while True:
    if cp.switch:
        print("Slide switch off!")
        cp.pixels.fill((0, 0, 0))
```

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

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



The Piano in the Key of Lime guide was chosen as an example. That guide is specific to Circuit Playground Express, and cannot be used on all boards. Do not

expect to download that exact bundle and have it work on your non-CPX microcontroller.

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

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

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

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

The Adafruit CircuitPython Library Bundle

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

Downloading the Adafruit CircuitPython Library Bundle

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

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

Click to visit circuitpython.org for the latest Adafruit CircuitPython Library Bundle

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

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

The CircuitPython Community Library Bundle

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

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

Downloading the CircuitPython Community Library Bundle

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

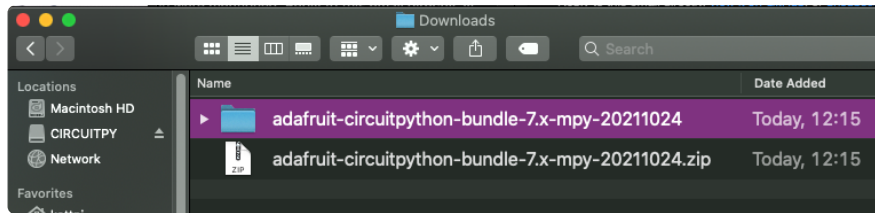
Click for the latest CircuitPython Community Library Bundle release

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

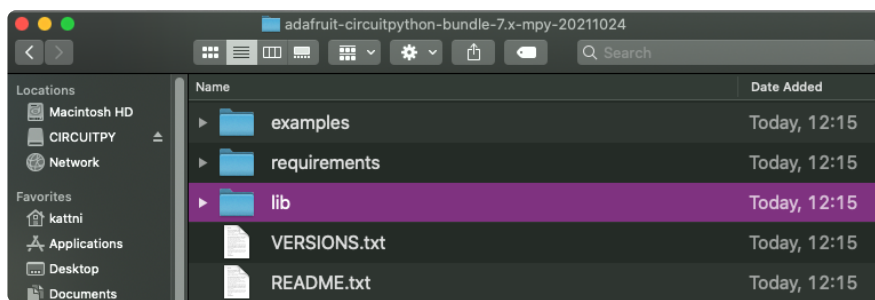
the version, check the version info in boot_out.txt file on the CIRCUITPY drive, or the initial prompt in the CircuitPython REPL. For example, if you're running v7.0.0, download the 7.x library bundle.

Understanding the Bundle

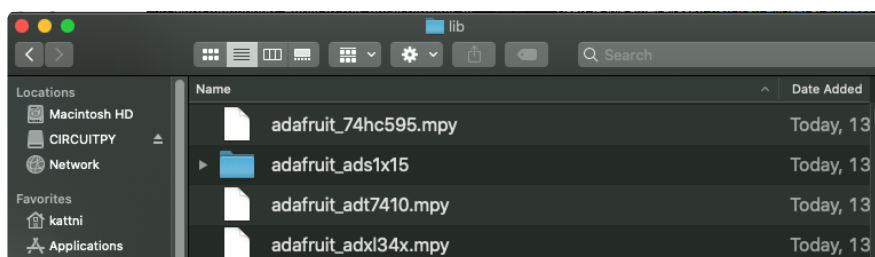
After downloading the zip, extract its contents. This is usually done by double clicking on the zip. On Mac OSX, it places the file in the same directory as the zip.



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



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

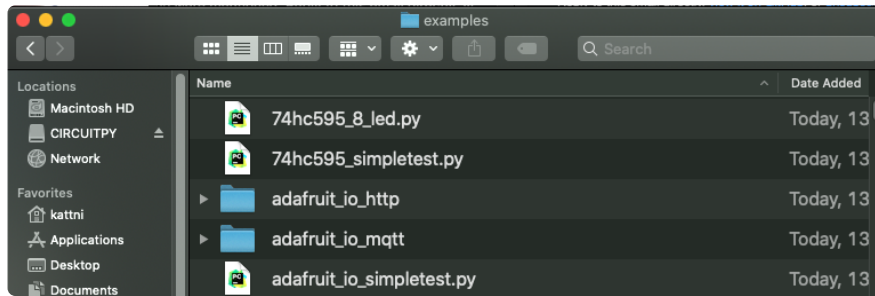


Example Files

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

- Allow for quick testing of devices.

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



Copying Libraries to Your Board

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

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

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

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

Understanding Which Libraries to Install

You now know how to load libraries on to your CircuitPython-compatible microcontroller board. You may now be wondering, how do you know which libraries you need to install? Unfortunately, it's not always straightforward. Fortunately, there is an obvious place to start, and a relatively simple way to figure out the rest. First up: the best place to start.

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

- `import library_or_module`

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

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

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

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

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

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

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

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

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

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

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

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

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

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

The final two imports are not as clear. Remember, when `import` statements are formatted like this, the first thing after the `from` is the library name. In this case, the library name is `adafruit_hid`. A search of the bundle will find an `adafruit_hid` folder. When a library is a folder, you must copy the entire folder and its contents as it is in the bundle to the lib folder on your CIRCUITPY drive. In this case, you would copy the entire `adafruit_hid` folder to your CIRCUITPY/lib folder.

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

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

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

Example: `ImportError` Due to Missing Library

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

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

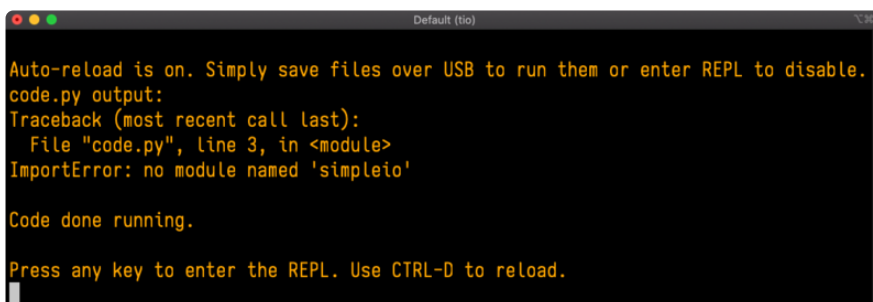
Let's use a modified version of the Blink example.

```
import board
import time
import simpleio

led = simpleio.DigitalOut(board.LED)

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

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



The screenshot shows a terminal window titled "Default (tio)". The output text is as follows:

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

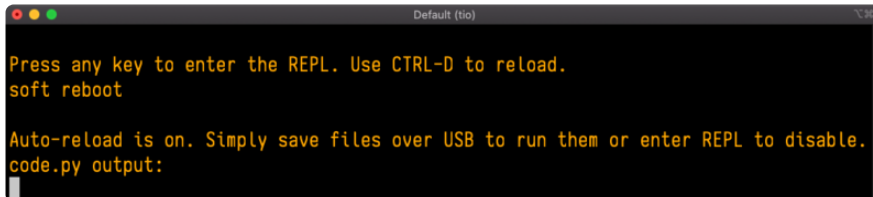
Code done running.

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

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

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

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

A screenshot of a terminal window titled "Default (tio)". The text inside the terminal is yellow on a black background. It reads: "Press any key to enter the REPL. Use CTRL-D to reload." followed by "soft reboot" on a new line. The next line says "Auto-reload is on. Simply save files over USB to run them or enter REPL to disable." and the final line is "code.py output:".

```
Default (tio)
Press any key to enter the REPL. Use CTRL-D to reload.
soft reboot
Auto-reload is on. Simply save files over USB to run them or enter REPL to disable.
code.py output:
```

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

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

Library Install on Non-Express Boards

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

You can still end up running out of space on your M0 non-Express board even if you only load libraries as you need them. There are a number of steps you can use to try to resolve this issue. You'll find suggestions on the [Troubleshooting page \(\)](#).

Updating CircuitPython Libraries and Examples

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

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

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

CircUp CLI Tool

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

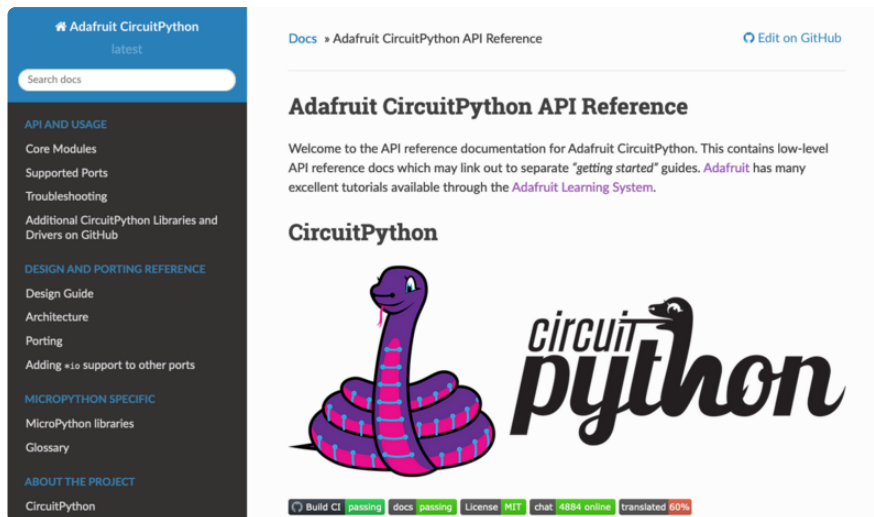
CircuitPython Documentation

You've learned about the CircuitPython built-in modules and external libraries. You know that you can find the modules in CircuitPython, and the libraries in the Library Bundles. There are guides available that explain the basics of many of the modules and libraries. However, there's sometimes more capabilities than are necessarily showcased in the guides, and often more to learn about a module or library. So, where can you find more detailed information? That's when you want to look at the API documentation.

The entire CircuitPython project comes with extensive documentation available on Read the Docs. This includes both the [CircuitPython core \(\)](#) and the [Adafruit CircuitPython libraries \(\)](#).

CircuitPython Core Documentation

The [CircuitPython core documentation \(\)](#) covers many of the details you might want to know about the CircuitPython core and related topics. It includes API and usage info, a design guide and information about porting CircuitPython to new boards, MicroPython info with relation to CircuitPython, and general information about the project.



The main page covers the basics including where to download CircuitPython, how to contribute, differences from MicroPython, information about the project structure, and a full table of contents for the rest of the documentation.

The list along the left side leads to more information about specific topics.

The first section is API and Usage. This is where you can find information about how to use individual built-in core modules, such as `time` and `digitalio`, details about the supported ports, suggestions for troubleshooting, and basic info and links to the library bundles. The Core Modules section also includes the Support Matrix, which is a table of which core modules are available on which boards.

The second section is Design and Porting Reference. It includes a design guide, architecture information, details on porting, and adding module support to other ports.

The third section is MicroPython Specific. It includes information on MicroPython and related libraries, and a glossary of terms.

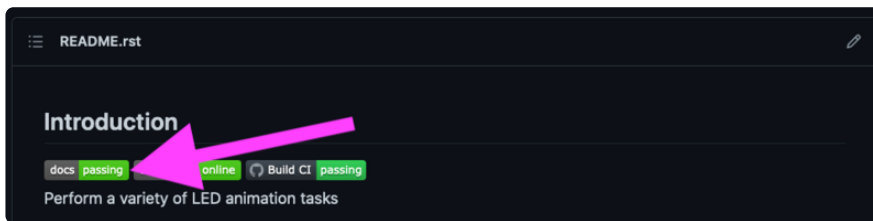
The fourth and final section is About the Project. It includes further information including details on building, testing, and debugging CircuitPython, along with various other useful links including the Adafruit Community Code of Conduct.

Whether you're a seasoned pro or new to electronics and programming, you'll find a wealth of information to help you along your CircuitPython journey in the documentation!

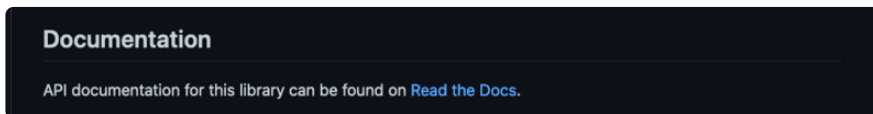
CircuitPython Library Documentation

The Adafruit CircuitPython libraries are documented in a very similar fashion. Each library has its own page on Read the Docs. There is a comprehensive list available [here](#) (). Otherwise, to view the documentation for a specific library, you can visit the GitHub repository for the library, and find the link in the README.

For the purposes of this page, the [LED Animation library](#) () documentation will be featured. There are two links to the documentation in each library GitHub repo. The first one is the docs badge near the top of the README.



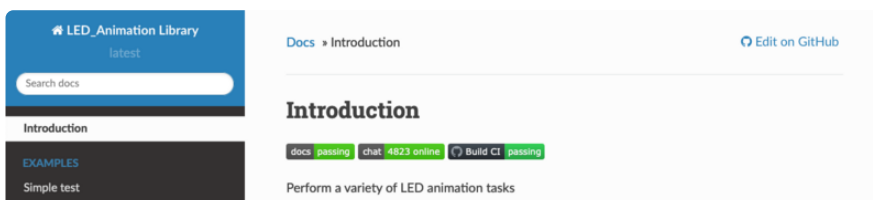
The second place is the Documentation section of the README. Scroll down to find it, and click on Read the Docs to get to the documentation.



Now that you know how to find it, it's time to take a look at what to expect.

Not all library documentation will look exactly the same, but this will give you some idea of what to expect from library docs.

The Introduction page is generated from the README, so it includes all the same info, such as PyPI installation instructions, a quick demo, and some build details. It also includes a full table of contents for the rest of the documentation (which is not part of the GitHub README). The page should look something like the following.



The left side contains links to the rest of the documentation, divided into three separate sections: Examples, API Reference, and Other Links.

Examples

The [Examples section](#) () is a list of library examples. This list contains anywhere from a small selection to the full list of the examples available for the library.

This section will always contain at least one example - the simple test example.



The simple test example is usually a basic example designed to show your setup is working. It may require other libraries to run. Keep in mind, it's simple - it won't showcase a comprehensive use of all the library features.

The LED Animation simple test demonstrates the Blink animation.



In some cases, you'll find a longer list, that may include examples that explore other features in the library. The LED Animation documentation includes a series of examples, all of which are available in the library. These examples include demonstrations of both basic and more complex features. Simply click on the example that interests you to view the associated code.



When there are multiple links in the Examples section, all of the example content is, in actuality, on the same page. Each link after the first is an anchor link to the specified section of the page. Therefore, you can also view all the available examples by scrolling down the page.

You can view the rest of the examples by clicking through the list or scrolling down the page. These examples are fully working code. Which is to say, while they may rely on other libraries as well as the library for which you are viewing the documentation, they should not require modification to otherwise work.

API Reference

The [API Reference section \(\)](#) includes a list of the library functions and classes. The API (Application Programming Interface) of a library is the set of functions and classes the library provides. Essentially, the API defines how your program interfaces with the functions and classes that you call in your code to use the library.

There is always at least one list item included. Libraries for which the code is included in a single Python (.py) file, will only have one item. Libraries for which the code is multiple Python files in a directory (called subpackages) will have multiple items in this list. The LED Animation library has a series of subpackages, and therefore, multiple items in this list.

Click on the first item in the list to begin viewing the API Reference section.



As with the Examples section, all of the API Reference content is on a single page, and the links under API Reference are anchor links to the specified section of the page.

When you click on an item in the API Reference section, you'll find details about the classes and functions in the library. In the case of only one item in this section, all the

available functionality of the library will be contained within that first and only subsection. However, in the case of a library that has subpackages, each item will contain the features of the particular subpackage indicated by the link. The documentation will cover all of the available functions of the library, including more complex ones that may not interest you.

The first list item is the animation subpackage. If you scroll down, you'll begin to see the available features of animation. They are listed alphabetically. Each of these things can be called in your code. It includes the name and a description of the specific function you would call, and if any parameters are necessary, lists those with a description as well.

```
class adafruit_led_animation.animation.Animation(pixel_object, speed, color, peers=None, paused=False, name=None)
```

Base class for animations.

add_cycle_complete_receiver(callback)
Adds an additional callback when the cycle completes.

Parameters
callback – Additional callback to trigger when a cycle completes. The callback is passed the animation object instance.

after_draw()
Animation subclasses may implement after_draw() to do operations after the main draw() is called.

You can view the other subpackages by clicking the link on the left or scrolling down the page. You may be interested in something a little more practical. Here is an example. To use the LED Animation library Comet animation, you would run the following example.

```
# SPDX-FileCopyrightText: 2021 Kattni Rembor for Adafruit Industries
# SPDX-License-Identifier: MIT

"""
This example animates a jade comet that bounces from end to end of the strip.

For QT Py Haxpress and a NeoPixel strip. Update pixel_pin and pixel_num to match
your wiring if
using a different board or form of NeoPixels.

This example will run on SAMD21 (M0) Express boards (such as Circuit Playground
Express or QT Py
Haxpress), but not on SAMD21 non-Express boards (such as QT Py or Trinket).
"""
import board
import neopixel

from adafruit_led_animation.animation.comet import Comet
from adafruit_led_animation.color import JADE

# Update to match the pin connected to your NeoPixels
pixel_pin = board.A3
# Update to match the number of NeoPixels you have connected
pixel_num = 30
```

```
pixels = neopixel.NeoPixel(pixel_pin, pixel_num, brightness=0.5, auto_write=False)
comet = Comet(pixels, speed=0.02, color=JADE, tail_length=10, bounce=True)

while True:
    comet.animate()
```

Note the line where you create the `comet` object. There are a number of items inside the parentheses. In this case, you're provided with a fully working example. But what if you want to change how the comet works? The code alone does not explain what the options mean.

So, in the API Reference documentation list, click the [adafruit_led_animation.animation.comet](#) link and scroll down a bit until you see the following.

```
class adafruit_led_animation.animation.comet.Comet(pixel_object, speed, color, tail_length=0, reverse=False, bounce=False, name=None, ring=False)
```

A comet animation.

Parameters

- `pixel_object` – The initialised LED object.
- `speed` (*float*) – Animation speed in seconds, e.g. `0.1`.
- `color` – Animation color in `(r, g, b)` tuple, or `0x000000` hex format.
- `tail_length` (*int*) – The length of the comet. Defaults to 25% of the length of the `pixel_object`. Automatically compensates for a minimum of 2 and a maximum of the length of the `pixel_object`.
- `reverse` (*bool*) – Animates the comet in the reverse order. Defaults to `False`.
- `bounce` (*bool*) – Comet will bounce back and forth. Defaults to `True`.
- `ring` (*bool*) – Ring mode. Defaults to `False`.

Look familiar? It is! This is the documentation for setting up the comet object. It explains what each argument provided in the comet setup in the code meant, as well as the other available features. For example, the code includes `speed=0.02`. The documentation clarifies that this is the "Animation speed in seconds". The code doesn't include `ring`. The documentation indicates this is an available setting that enables "Ring mode".

This type of information is available for any function you would set up in your code. If you need clarification on something, wonder whether there's more options available, or are simply interested in the details involved in the code you're writing, check out the documentation for the CircuitPython libraries!

Other Links

This section is the same for every library. It includes a list of links to external sites, which you can visit for more information about the CircuitPython Project and Adafruit.

That covers the CircuitPython library documentation! When you are ready to go beyond the basic library features covered in a guide, or you're interested in understanding those features better, the library documentation on Read the Docs has you covered!

Recommended Editors

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

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

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

To avoid the likelihood of filesystem corruption, use an editor that writes out the file completely when you save it. Check out the list of recommended editors below.

Recommended editors

- [mu \(\)](#) is an editor that safely writes all changes (it's also our recommended editor!)
- [emacs \(\)](#) is also an editor that will [fully write files on save \(\)](#)
- [Sublime Text \(\)](#) safely writes all changes
- [Visual Studio Code \(\)](#) appears to safely write all changes
- gedit on Linux appears to safely write all changes
- [IDLE \(\)](#), in Python 3.8.1 or later, [was fixed \(\)](#) to write all changes immediately
- [Thonny \(\)](#) fully writes files on save

Recommended only with particular settings or add-ons

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

The editors listed below are specifically NOT recommended!

Editors that are NOT recommended

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

Advanced Serial Console on Windows

Windows 7 and 8.1

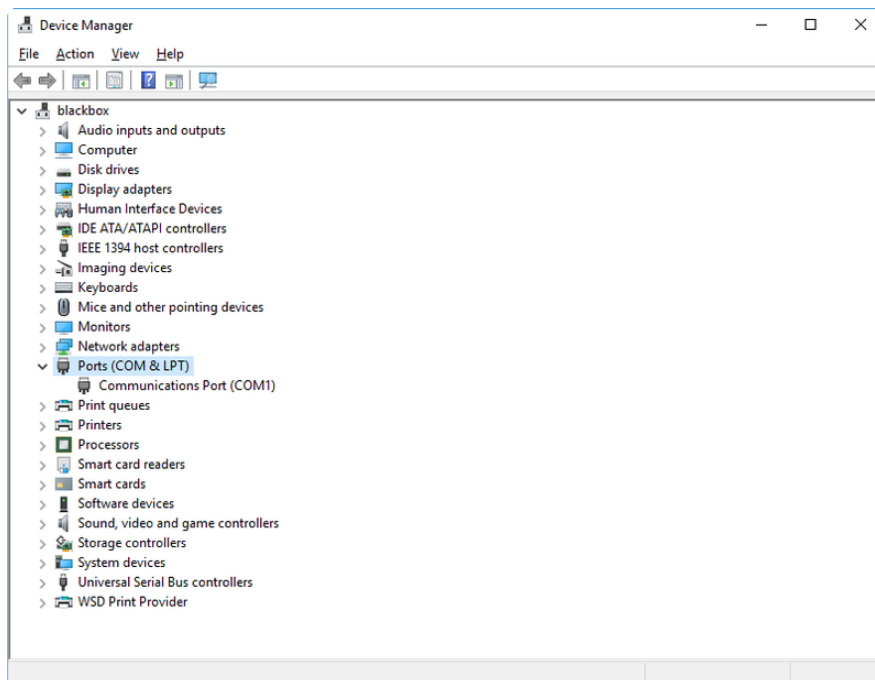
If you're using Windows 7 (or 8 or 8.1), you'll need to install drivers. See the [Windows 7 and 8.1 Drivers page \(\)](#) for details. You will not need to install drivers on Mac, Linux or Windows 10.

You are strongly encouraged to upgrade to Windows 10 if you are still using Windows 7 or Windows 8 or 8.1. Windows 7 has reached end-of-life and no longer receives security updates. A free upgrade to Windows 10 is [still available \(\)](#).

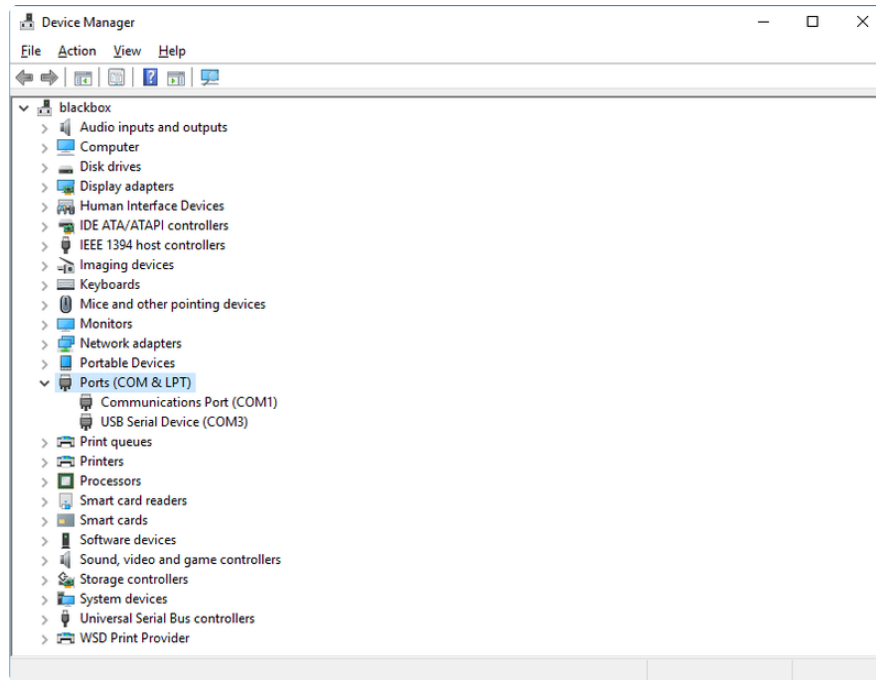
What's the COM?

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

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



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



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

Install Putty

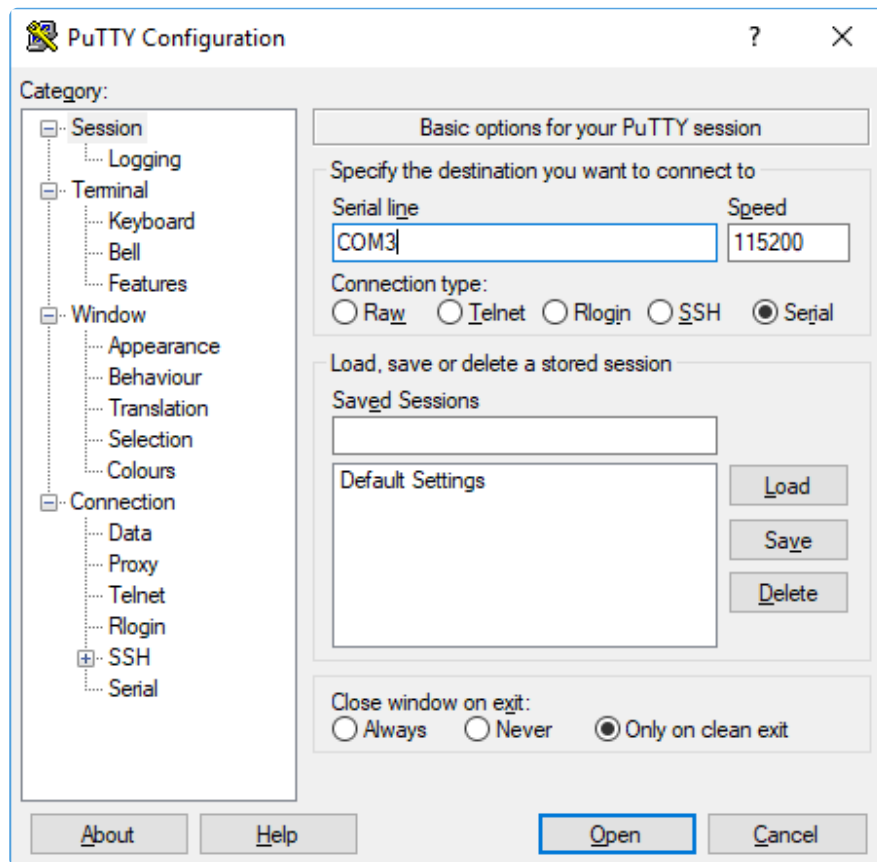
If you're using Windows, you'll need to download a terminal program. You're going to use PuTTY.

The first thing to do is download the [latest version of PuTTY \(\)](#). 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.

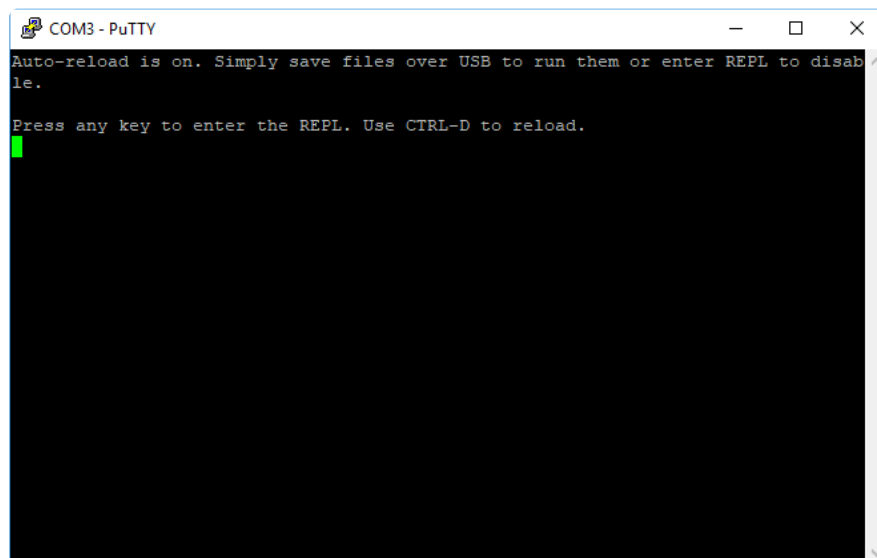
Now you need to open PuTTY.

- 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.

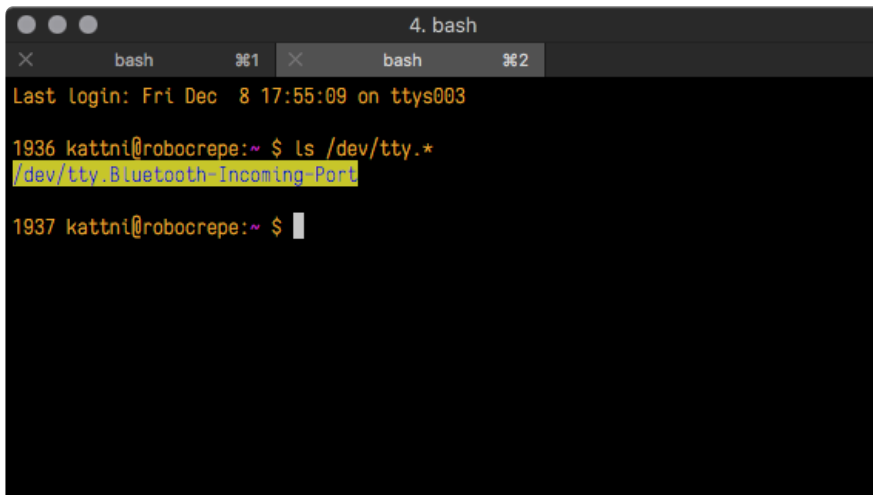
What's the Port?

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, you're asking to see all of the listings in `/dev/` that start with `t` and end in anything. This will show us the current serial connections.

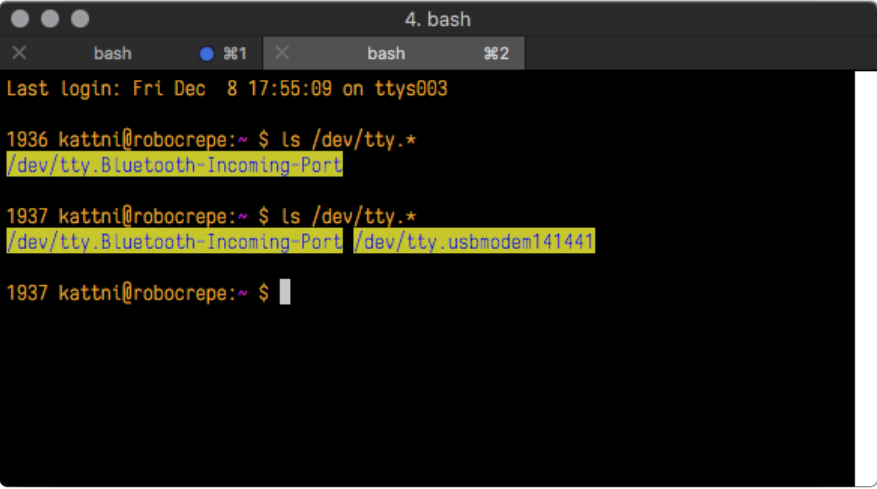


```
4. bash
bash %1  bash %2
Last login: Fri Dec  8 17:55:09 on ttys003
1936 kattni@robocrepe:~ $ ls /dev/tty.*
/dev/tty.Bluetooth-Incoming-Port
1937 kattni@robocrepe:~ $
```

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 terminal window titled "4. bash" with two tabs labeled "bash" and "bash". The terminal shows the following text: "Last login: Fri Dec 8 17:55:09 on ttys003", "1936 kattni@robocrepe:~ \$ ls /dev/tty.*", "/dev/tty.Bluetooth-Incoming-Port", "1937 kattni@robocrepe:~ \$ ls /dev/tty.*", "/dev/tty.Bluetooth-Incoming-Port /dev/tty.usbmodem141441", and "1937 kattni@robocrepe:~ \$". The paths are highlighted in yellow.

```
4. bash
bash
bash
Last login: Fri Dec 8 17:55:09 on ttys003
1936 kattni@robocrepe:~ $ ls /dev/tty.*
/dev/tty.Bluetooth-Incoming-Port
1937 kattni@robocrepe:~ $ ls /dev/tty.*
/dev/tty.Bluetooth-Incoming-Port /dev/tty.usbmodem141441
1937 kattni@robocrepe:~ $
```

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

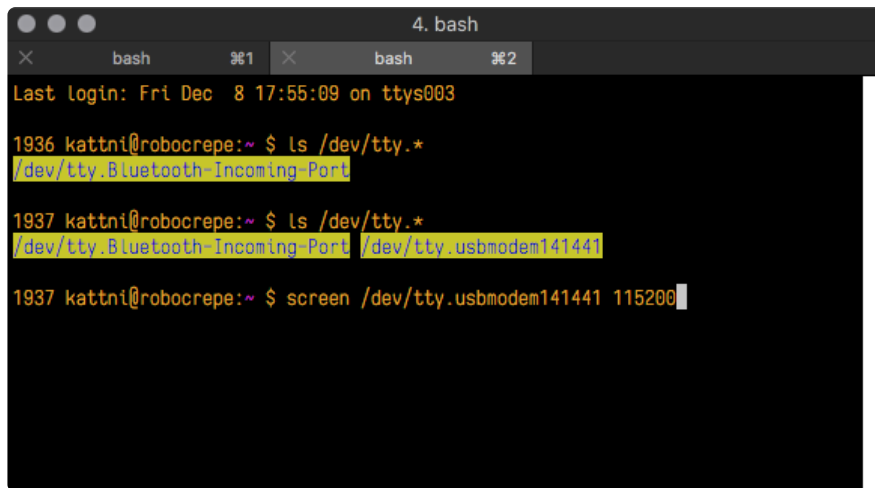
Using Linux, a new listing has appeared called `/dev/ttyACM0`. The `ttyACM0` part of this listing is the name the example board is using. Yours will be called something similar.

Connect with screen

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

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

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



```
4. bash
bash %1 bash %2
Last login: Fri Dec 8 17:55:09 on ttys003
1936 kattni@robocrepe:~ $ ls /dev/tty.*
/dev/tty.Bluetooth-Incoming-Port
1937 kattni@robocrepe:~ $ ls /dev/tty.*
/dev/tty.Bluetooth-Incoming-Port /dev/tty.usbmodem141441
1937 kattni@robocrepe:~ $ screen /dev/tty.usbmodem141441 115200
```

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!

Advanced Serial Console on Linux

Connecting to the serial console on Linux does not require installing any drivers, but you may need to install `screen` using your package manager. You'll use a terminal program to find your board, and `screen` to connect to it. There are a variety of terminal programs such as `gnome-terminal` (called Terminal) or `Konsole` on KDE.

The `tio` program works as well to connect to your board, and has the benefit of automatically reconnecting. You would need to install it using your package manager.

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.

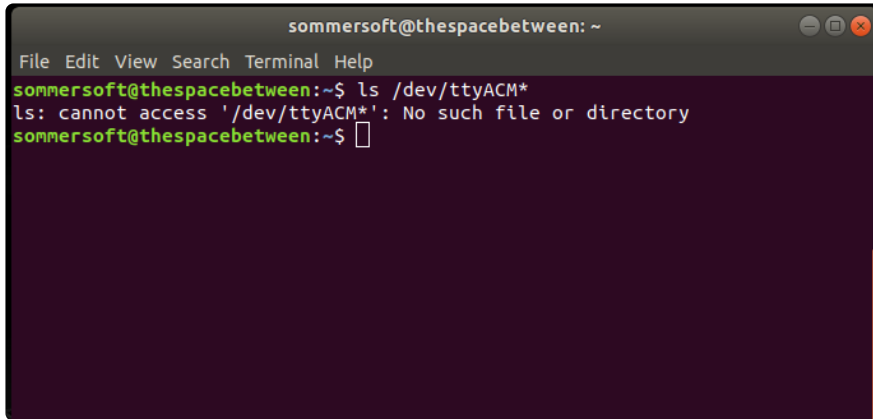
The easiest way to determine which port the board is using is to first check without the board plugged in. Open your terminal program and type the following:

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

Each serial connection shows up in the `/dev/` directory. It has a name that starts with `ttyACM`. The command `ls` shows you a list of items in a directory. You can use `*` as a wildcard, to search for files that start with the same letters but end in something

different. In this case, You're asking to see all of the listings in /dev/ that start with ttyA CM and end in anything. This will show us the current serial connections.

In the example below, the error is indicating that are no current serial connections starting with ttyACM.

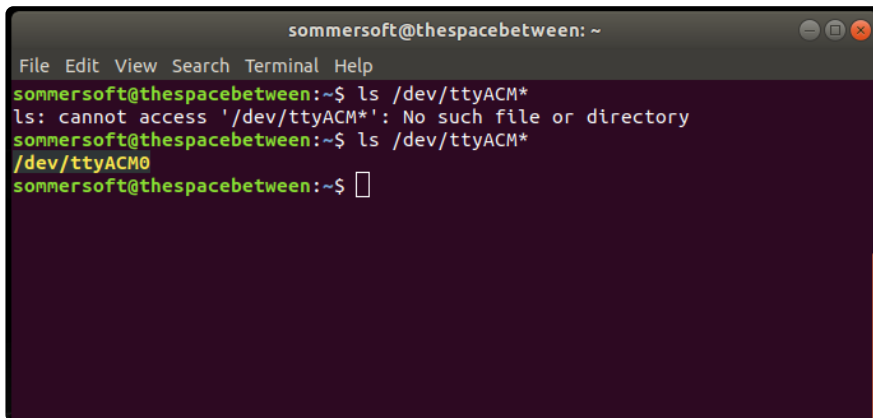


```
sommersoft@thespacebetween: ~  
File Edit View Search Terminal Help  
sommersoft@thespacebetween:~$ ls /dev/ttyACM*  
ls: cannot access '/dev/ttyACM*': No such file or directory  
sommersoft@thespacebetween:~$
```

Now plug in your board. In your terminal program, type:

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

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



```
sommersoft@thespacebetween: ~  
File Edit View Search Terminal Help  
sommersoft@thespacebetween:~$ ls /dev/ttyACM*  
ls: cannot access '/dev/ttyACM*': No such file or directory  
sommersoft@thespacebetween:~$ ls /dev/ttyACM*  
/dev/ttyACM0  
sommersoft@thespacebetween:~$
```

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

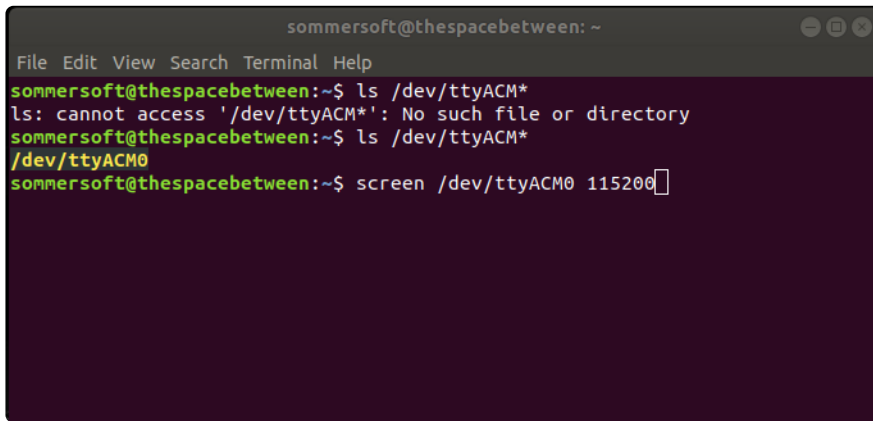
Connect with screen

Now that you know the name your board is using, you're ready connect to the serial console. You'll use a command called `screen`. You may need to install it using the package manager.

To connect to the serial console, use your terminal program. 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.



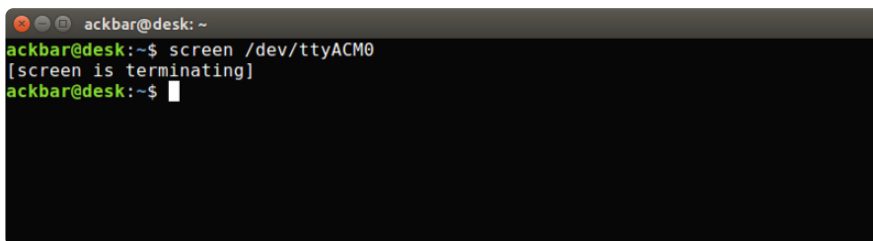
```
sommersoft@thespacebetween: ~  
File Edit View Search Terminal Help  
sommersoft@thespacebetween:~$ ls /dev/ttyACM*  
ls: cannot access '/dev/ttyACM*': No such file or directory  
sommersoft@thespacebetween:~$ ls /dev/ttyACM*  
/dev/ttyACM0  
sommersoft@thespacebetween:~$ screen /dev/ttyACM0 115200
```

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

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

Permissions on Linux

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



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

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

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

Once you enter your password, you should be in:

```
ackbar@desk: ~
Auto-reload is on. Simply save files over USB to run them or enter REPL to disable.

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

Adafruit CircuitPython 2.1.0 on 2017-10-17; Adafruit Trinket M0 with samd21e18
>>> █
```

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

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

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

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

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

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

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

And you're in:

```
ackbar@desk: ~
Auto-reload is on. Simply save files over USB to run them or enter REPL to disable.

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

Adafruit CircuitPython 2.1.0 on 2017-10-17; Adafruit Trinket M0 with samd21e18
>>>
```

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

Frequently Asked Questions

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

What are some common acronyms to know?

- CP or CPy = [CircuitPython \(\)](#)
 - CPC = [Circuit Playground Classic \(\)](#) (does not run CircuitPython)
 - CPX = [Circuit Playground Express \(\)](#)
 - CPB = [Circuit Playground Bluefruit \(\)](#)
-

Using Older Versions

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

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

We are no longer building or supporting the CircuitPython 7.x or earlier library bundles. We highly encourage you to [update CircuitPython to the latest version \(\)](#) and use [the current version of the libraries \(\)](#). However, if for some reason you cannot update, here are the last available library bundles for older versions:

- [2.x bundle \(\)](#)
 - [3.x bundle \(\)](#)
 - [4.x bundle \(\)](#)
 - [5.x bundle \(\)](#)
 - [6.x bundle \(\)](#)
 - [7.x bundle \(\)](#)
-

Python Arithmetic

Does CircuitPython support floating-point numbers?

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

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

Does CircuitPython support long integers, like regular Python?

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

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

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

Wireless Connectivity

How do I connect to the Internet with CircuitPython?

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

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

For further project examples, and guides about using AirLift with specific hardware, check out [the Adafruit Learn System](#) ().

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

The nRF52840 and nRF52833 boards have the most complete BLE implementation. Your program can act as both a BLE central and peripheral. As a central, you can scan for advertisements, and connect to an advertising board. As a peripheral, you can advertise, and you can create services available to a central. Pairing and bonding are supported.

ESP32-C3 and ESP32-S3 boards currently provide an [incomplete](#) () BLE implementation. Your program can act as a central, and connect to a peripheral. You can advertise, but you cannot create services. You cannot advertise anonymously. Pairing and bonding are not supported.

The ESP32 could provide a similar implementation, but it is not yet available. Note that the ESP32-S2 does not have Bluetooth capability.

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

Are there other ways to communicate by radio with CircuitPython?

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

Asyncio and Interrupts

Is there asyncio support in CircuitPython?

There is support for asyncio starting with CircuitPython 7.1.0, on all boards except the smallest SAMD21 builds. Read about using it in the [Cooperative Multitasking in CircuitPython](#) () Guide.

Does CircuitPython support interrupts?

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

Status RGB LED

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

The status LED can tell you what's going on with your CircuitPython board. [Read more here for what the colors mean!](#) ()

Memory Issues

What is a MemoryError?

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

What do I do when I encounter a MemoryError?

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

Make sure you are using .mpy versions of libraries. All of the CircuitPython libraries are available in the bundle in a .mpy format which takes up less memory than .py format. Be sure that you're using [the latest library bundle \(\)](#) for your version of CircuitPython.

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

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

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

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

How can I create my own .mpy files?

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

You can download `mpy-cross` for your operating system from [here \(\)](#). 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?

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

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

Unsupported Hardware

Is ESP8266 or ESP32 supported in CircuitPython? Why not?

We dropped ESP8266 support as of 4.x - For more information please read about it [here](#) ()!

As of CircuitPython 8.x we have started to support ESP32 and ESP32-C3 and have added a WiFi workflow for wireless coding! ()

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

Does Feather M0 support WINC1500?

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

Can AVR's such as ATmega328 or ATmega2560 run CircuitPython?

No.

Troubleshooting

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

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

Always Run the Latest Version of CircuitPython and Libraries

As CircuitPython development continues and there are new releases, Adafruit will stop supporting older releases. You need to [update to the latest CircuitPython. \(\)](#).

You need to download the CircuitPython Library Bundle that matches your version of CircuitPython. Please update CircuitPython and then [download the latest bundle \(\)](#).

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

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

Adafruit is no longer building or supporting the CircuitPython 7.x or earlier library bundles. You are highly encouraged to [update CircuitPython to the latest version \(\)](#) and use [the current version of the libraries \(\)](#). However, if for some reason you cannot update, links to the previous bundles are available in the [FAQ \(\)](#).

Bootloader (boardnameBOOT) Drive Not Present

You may have a different board.

Only Adafruit Express boards and the SAMD21 non-Express boards ship with the [UF2 bootloader \(\)](#) installed. The Feather M0 Basic, Feather M0 Adalogger, and similar boards use a regular Arduino-compatible bootloader, which does not show a boardnameBOOT drive.

MakeCode

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

macOS

DriveDx and its accompanying SAT SMART Driver can interfere with seeing the BOOT drive. [See this forum post \(\)](#) for how to fix the problem.

Windows 10

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

Windows 7 or 8.1

To use a CircuitPython-compatible board with Windows 7 or 8.1, you must install a driver. Installation instructions are available [here \(\)](#).

It is [recommended \(\)](#) that you upgrade to Windows 10 if possible; an upgrade is probably still free for you. Check [here \(\)](#).

The Windows Drivers installer was last updated in November 2020 (v2.5.0.0) . Windows 7 drivers for CircuitPython boards released since then, including RP2040 boards, are not available. There are no plans to release drivers for new boards. The boards work fine on Windows 10.

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

Let us know in the [Adafruit support forums \(\)](#) or on the [Adafruit Discord \(\)](#) if this does not work for you!

Windows Explorer Locks Up When Accessing boardnameBOOT Drive

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

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

Copying UF2 to boardnameBOOT Drive Hangs at 0% Copied

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

CIRCUITPY Drive Does Not Appear or Disappears Quickly

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

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

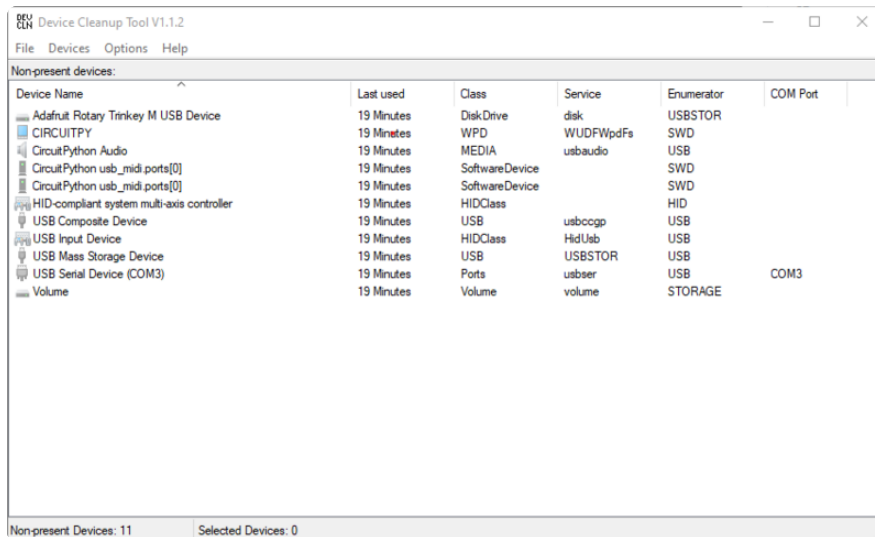
Sophos Endpoint security software [can cause CIRCUITPY to disappear \(\)](#) and the BOOT drive to reappear. It is not clear what causes this behavior.

Samsung Magician can cause CIRCUITPY to disappear (reported [here](#) () and [here](#) ()).

Device Errors or Problems on Windows

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

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



Select all the devices you want to remove, and then press Delete. It is usually safe just to select everything. Any device that is removed will get a fresh install when you plug it in. Using the Device Cleanup Tool also discards all the COM port assignments for the unplugged boards. If you have used many Arduino and CircuitPython boards, you have probably seen higher and higher COM port numbers used, seemingly without end. This will fix that problem.

Serial Console in Mu Not Displaying Anything

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

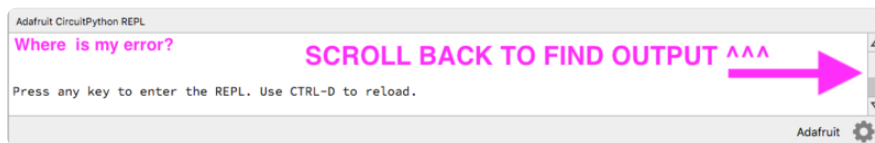
Depending on the size of your screen or Mu window, when you open the serial console, the serial console panel may be very small. This can be a problem. A basic CircuitPython error takes 10 lines to display!

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

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

More complex errors take even more lines!

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



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

code.py Restarts Constantly

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

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

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

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

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

CircuitPython RGB Status Light

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

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

CircuitPython 7.0.0 and Later

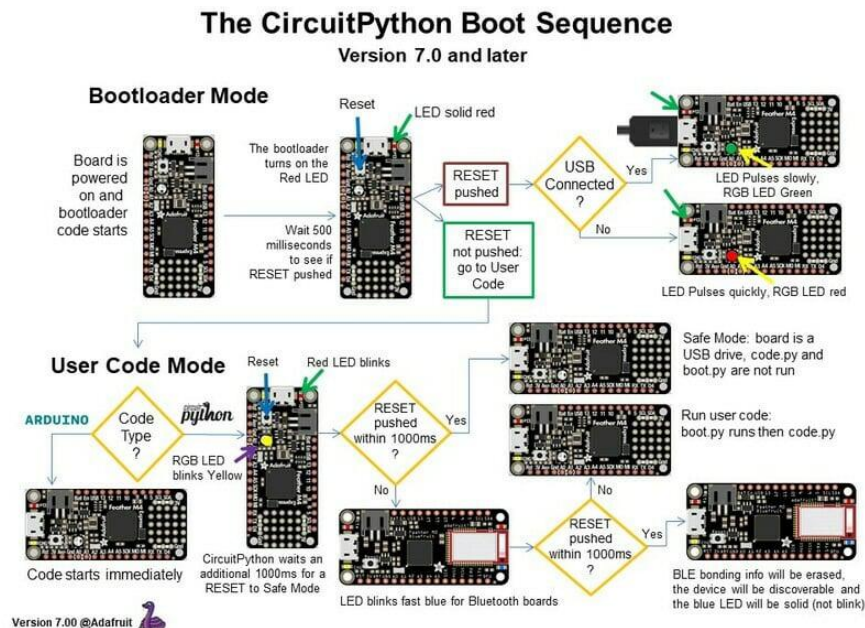
The status LED blinks were changed in CircuitPython 7.0.0 in order to save battery power and simplify the blinks. These blink patterns will occur on single color LEDs when the board does not have any RGB LEDs. Speed and blink count also vary for this reason.

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

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

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

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



CircuitPython 6.3.0 and earlier

Here's what the colors and blinking mean:

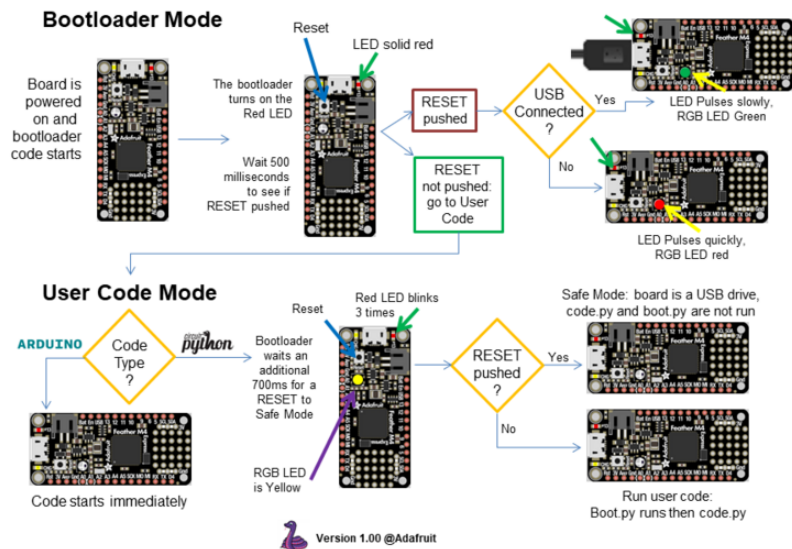
- steady GREEN: code.py (or code.txt, main.py, or main.txt) is running
- pulsing GREEN: code.py (etc.) has finished or does not exist
- steady YELLOW at start up: (4.0.0-alpha.5 and newer) CircuitPython is waiting for a reset to indicate that it should start in safe mode
- pulsing YELLOW: Circuit Python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

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

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

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

The CircuitPython Boot Sequence



Serial console showing **ValueError: Incompatible .mpy file**

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

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

CIRCUITPY Drive Issues

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

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

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

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

Safe Mode

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

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

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

Entering Safe Mode in CircuitPython 7.x and Later

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

Entering Safe Mode in CircuitPython 6.x

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

In Safe Mode

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

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

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

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

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

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

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

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

To erase CIRCUITPY: `storage.erase_filesystem()`

CircuitPython includes a built-in function to erase and reformat the filesystem. If you have a version of CircuitPython older than 2.3.0 on your board, you can [update to the newest version \(\)](#) to do this.

1. [Connect to the CircuitPython REPL \(\)](#) using Mu or a terminal program.
2. Type the following into the REPL:

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

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

Erase CIRCUITPY Without Access to the REPL

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

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

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

For the specific boards listed below:

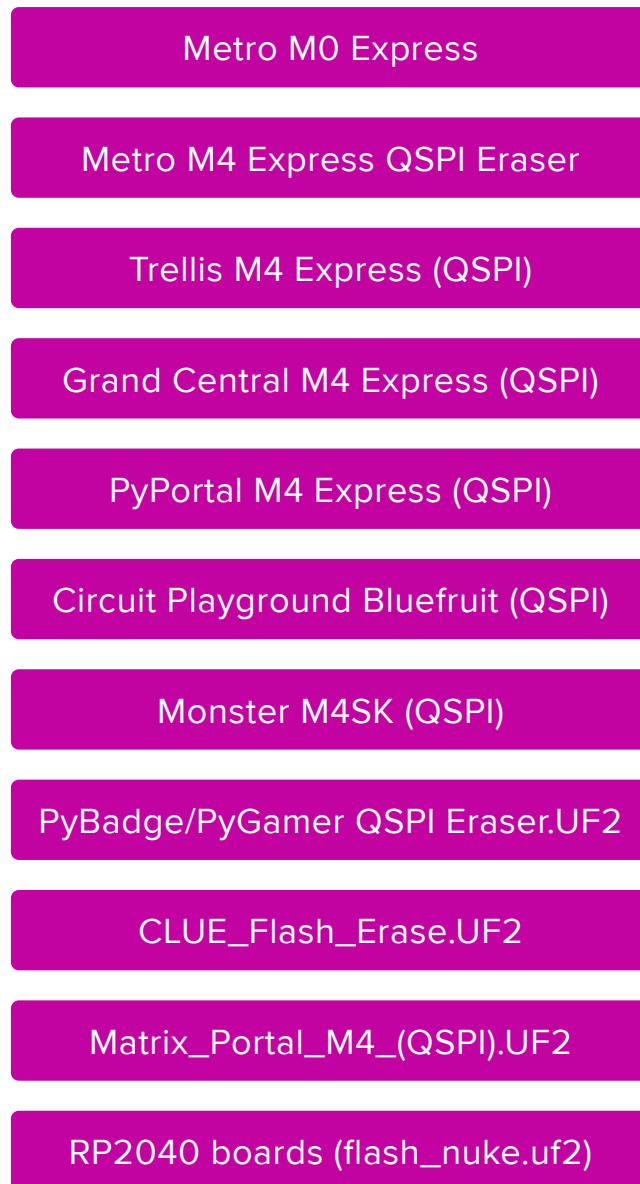
If the board you are trying to erase is listed below, follow the steps to use the file to erase your board.

1. Download the correct erase file:

Circuit Playground Express

Feather M0 Express

Feather M4 Express



2. Double-click the reset button on the board to bring up the boardnameBOOT drive.
3. Drag the erase .uf2 file to the boardnameBOOT drive.
4. The status LED will turn yellow or blue, indicating the erase has started.
5. After approximately 15 seconds, the status LED will light up green. On the NeoTrellis M4 this is the first NeoPixel on the grid
6. Double-click the reset button on the board to bring up the boardnameBOOT drive.
7. [Drag the appropriate latest release of CircuitPython \(\)](#) .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 \(\)](#). You'll also need to load your code and reinstall your libraries!

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

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

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

1. Download the erase file:

SAMD21 non-Express Boards

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 \(\)](#) .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 \(\)](#) You'll also need to load your code and reinstall your libraries!

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

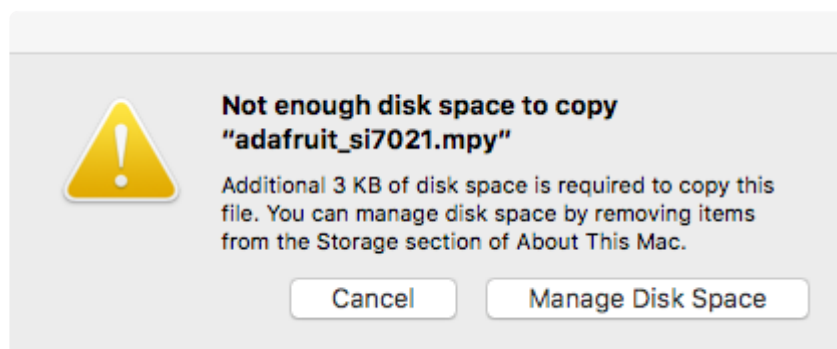
Any SAMD21-based microcontroller that does not have external flash available is considered a SAMD21 non-Express board. Non-Express boards that do not have a UF2 bootloader include the Feather M0 Basic Proto, Feather Adalogger, or the Arduino Zero.

If you are trying to erase a non-Express board that does not have a UF2 bootloader, [follow these directions to reload CircuitPython using `bossac` \(\)](#), which will erase and re-create CIRCUITPY.

Running Out of File Space on SAMD21 Non-Express Boards

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

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



Delete something!

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

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

Use tabs

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

On MacOS?

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

Prevent & Remove MacOS Hidden Files

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

```
ls -l /Volumes
```

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

Now follow the [steps from this question \(\)](#) to run these terminal commands that stop hidden files from being created on the board:

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

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

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

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

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

Copy Files on MacOS Without Creating Hidden Files

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

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

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

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

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

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

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

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

Other MacOS Space-Saving Tips

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


```
Default (-bash)
Last login: Thu Oct 28 17:19:15 on ttys008

7039 kattni@robocrepe:~ $ cd /Volumes/

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

7041 kattni@robocrepe:Volumes $
```

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

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

7042 kattni@robocrepe:Volumes $
```

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

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

7043 kattni@robocrepe:Volumes $
```

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

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

7044 kattni@robocrepe:Volumes $
```

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

Device Locked Up or Boot Looping

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

scripts, but will still connect the CIRCUITPY drive so that you can remove or modify those files as needed.

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

For most devices:

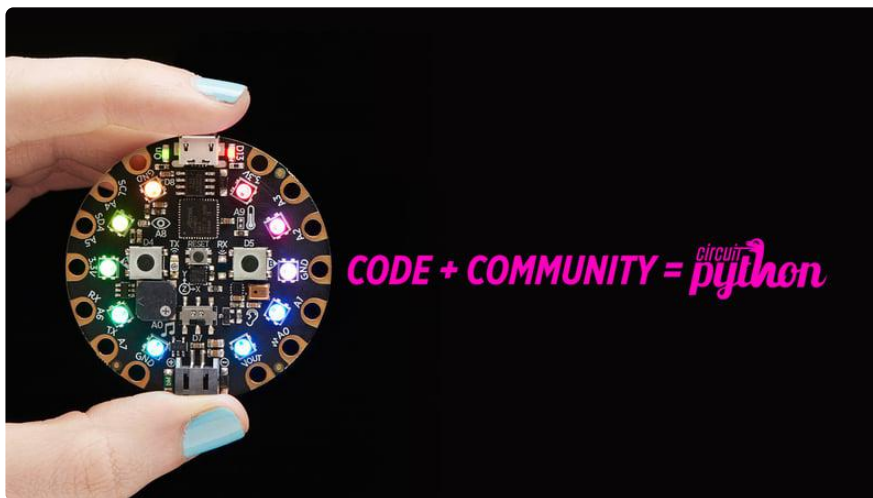
Press the reset button, and then when the RGB status LED blinks yellow, press the reset button again. Since your reaction time may not be that fast, try a "slow" double click, to catch the yellow LED on the second click.

For ESP32-S2 based devices:

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

Refer to the diagrams above for boot sequence details.

Welcome to the Community!

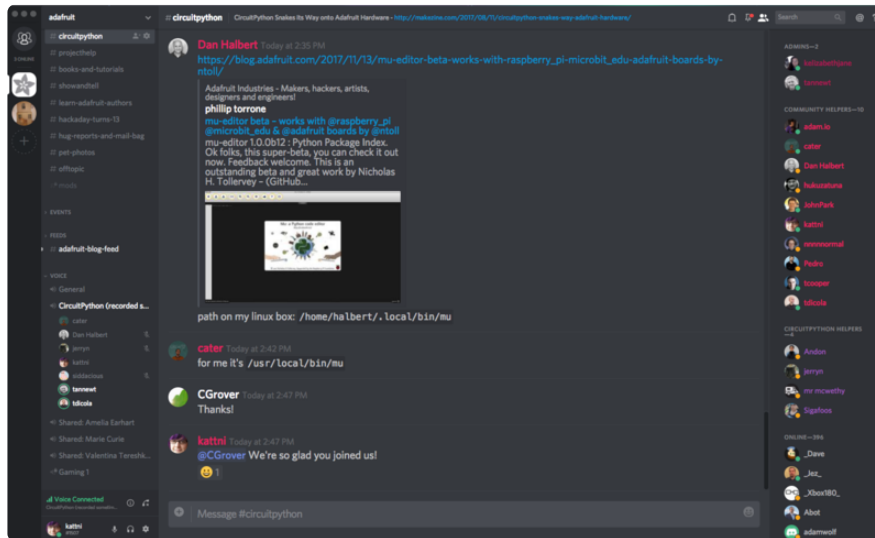


CircuitPython is a programming language that's super simple to get started with and great for learning. It runs on microcontrollers and works out of the box. You can plug it in and get started with any text editor. The best part? CircuitPython comes with an amazing, supportive community.

Everyone is welcome! CircuitPython is Open Source. This means it's available for anyone to use, edit, copy and improve upon. This also means CircuitPython becomes better because of you being a part of it. Whether this is your first microcontroller board or you're a seasoned software engineer, you have something important to offer

the Adafruit CircuitPython community. This page highlights some of the many ways you can be a part of it!

Adafruit Discord



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

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

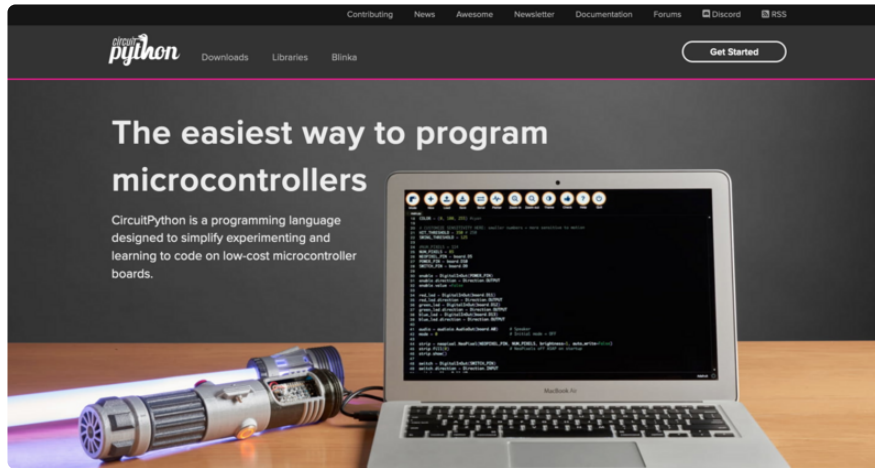
The help with CircuitPython channel is where to go with your CircuitPython questions. #help-with-circuitpython is there for new users and developers alike so feel free to ask a question or post a comment! Everyone of any experience level is welcome to join in on the conversation. Your contributions are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

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

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

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

CircuitPython.org



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

Contributing

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

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

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

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

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

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

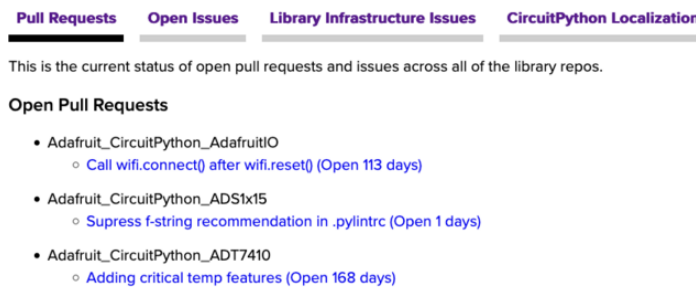
Current Status for Tue, Nov 02, 2021

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

Now, a look at the different options.

Pull Requests

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



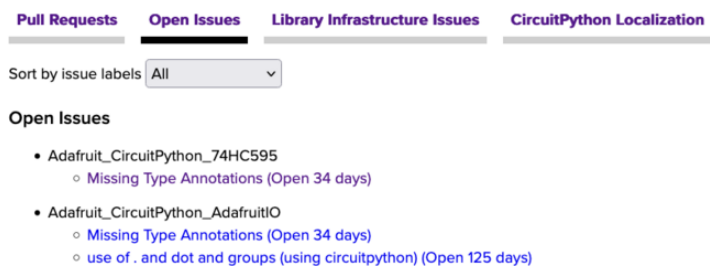
The screenshot shows a navigation bar with four tabs: 'Pull Requests' (selected), 'Open Issues', 'Library Infrastructure Issues', and 'CircuitPython Localization'. Below the tabs, a message states: 'This is the current status of open pull requests and issues across all of the library repos.' Underneath, the 'Open Pull Requests' section lists three items:

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

GitHub pull requests, or PRs, are opened when folks have added something to an Adafruit CircuitPython library GitHub repo, and are asking for Adafruit to add, or merge, their changes into the main library code. For PRs to be merged, they must first be reviewed. Reviewing is a great way to contribute! Take a look at the list of open pull requests, and pick one that interests you. If you have the hardware, you can test code changes. If you don't, you can still check the code updates for syntax. In the case of documentation updates, you can verify the information, or check it for spelling and grammar. Once you've checked out the update, you can leave a comment letting us know that you took a look. Once you've done that for a while, and you're more comfortable with it, you can consider joining the CircuitPythonLibrarians review team. The more reviewers we have, the more authors we can support. Reviewing is a crucial part of an open source ecosystem, CircuitPython included.

Open Issues

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



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

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



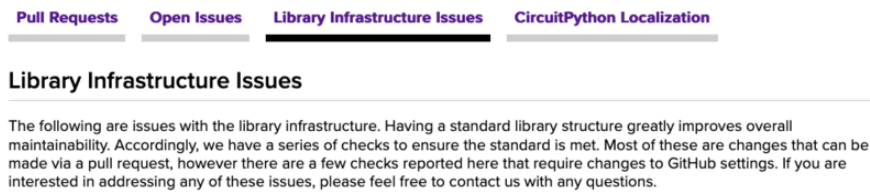
If you're new to everything, new to contributing to open source, or new to contributing to the CircuitPython project, you can choose "Good first issue". Issues with that label are well defined, with a finite scope, and are intended to be easy for someone new to figure out.

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

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

Library Infrastructure Issues

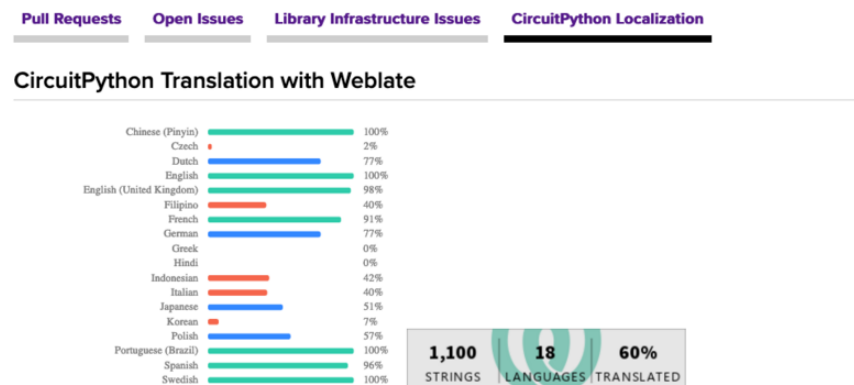
The third tab you'll find is a list of library infrastructure issues.



This section is generated by a script that runs checks on the libraries, and then reports back where there may be issues. It is made up of a list of subsections each containing links to the repositories that are experiencing that particular issue. This page is available mostly for internal use, but you may find some opportunities to contribute on this page. If there's an issue listed that sounds like something you could help with, mention it on Discord, or file an issue on GitHub indicating you're working to resolve that issue. Others can reply either way to let you know what the scope of it might be, and help you resolve it if necessary.

CircuitPython Localization

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

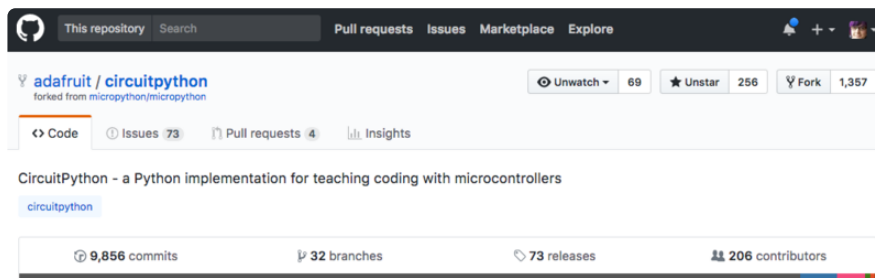


If you speak another language, you can help translate CircuitPython! The translations apply to informational and error messages that are within the CircuitPython core. It means that folks who do not speak English have the opportunity to have these messages shown to them in their own language when using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

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

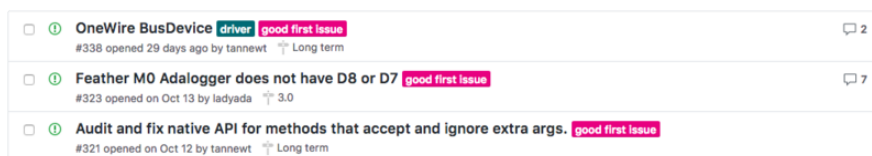
Regardless of your skill level, or how you want to contribute to the CircuitPython project, there is an opportunity available. The [Contributing page \(\)](#) is an excellent place to start!

Adafruit GitHub



Whether you're just beginning or are life-long programmer who would like to contribute, there are ways for everyone to be a part of the CircuitPython project. The CircuitPython core is written in C. The libraries are written in Python. GitHub is the best source of ways to contribute to the [CircuitPython core \(\)](#), and the [CircuitPython libraries \(\)](#). If you need an account, visit [https://github.com/ \(\)](https://github.com/) and sign up.

If you're new to GitHub or programming in general, there are great opportunities for you. For the CircuitPython core, head over to the CircuitPython repository on GitHub, click on "[Issues \(\)](#)", and you'll find a list that includes issues labeled "[good first issue \(\)](#)". For the libraries, head over to the [Contributing page Issues list \(\)](#), and use the drop down menu to search for "[good first issue \(\)](#)". These issues are things that have been identified as something that someone with any level of experience can help with. These issues include options like updating documentation, providing feedback, and fixing simple bugs. If you need help getting started with GitHub, there is an excellent guide on [Contributing to CircuitPython with Git and GitHub \(\)](#).



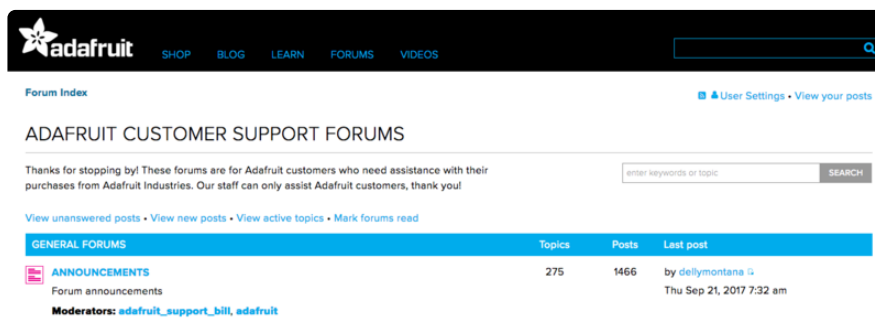
Already experienced and looking for a challenge? Checkout the rest of either issues list and you'll find plenty of ways to contribute. You'll find all sorts of things, from new driver requests, to library bugs, to core module updates. There's plenty of opportunities for everyone at any level!

When working with or using CircuitPython or the CircuitPython libraries, you may find problems. If you find a bug, that's great! The team loves bugs! Posting a detailed issue to GitHub is an invaluable way to contribute to improving CircuitPython. For CircuitPython itself, file an issue [here](#) (). For the libraries, file an issue on the specific library repository on GitHub. Be sure to include the steps to replicate the issue as well as any other information you think is relevant. The more detail, the better!

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

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

Adafruit Forums



The [Adafruit Forums](#) () 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](#) () category under "Supported Products & Projects" is the best place to post your CircuitPython questions.

Forum Index > Supported Products & Projects > Adafruit CircuitPython

Adafuit CircuitPython
Moderators: adafuit_support_bill, adafuit

POST A TOPIC | Search this forum... | SEARCH | Mark topics read • 4154 topics • Page 1 of 84 • 12345 ... 84

Please be positive and constructive with your questions and comments.

ANNOUNCEMENTS		Replies	Views	Last post
	CIRCUITPYTHON 7.2.0 ALPHA 1 RELEASED! by danhalbert • Tue Dec 28, 2021 11:55 pm	0	20	by danhalbert Tue Dec 28, 2021 11:55 pm
	CIRCUITPYTHON 7.1.0 RELEASED! by danhalbert • Tue Dec 28, 2021 12:01 pm	1	32	by rplloverbd Wed Dec 29, 2021 5:53 am
	SAMD51 (M4) BOARD USERS: UPDATE YOUR BOOTLOADERS TO >=V3.9.0 by danhalbert • Fri May 08, 2020 12:55 pm	10	2428	by Guest Sat Aug 15, 2020 11:28 pm

TOPICS	Replies	Views	Last post
--------	---------	-------	-----------

Be sure to include the steps you took to get to where you are. If it involves wiring, post a picture! If your code is giving you trouble, include your code in your post! These are great ways to make sure that there's enough information to help you with your issue.

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

Read the Docs

The screenshot shows the Adafruit CircuitPython documentation interface. On the left is a navigation sidebar with a search bar and a list of modules including 'audioio'. The main content area displays the 'audioio' module page, which includes a title, a brief description, a list of libraries (like 'AudioOut'), and a note about deinitializing classes. Navigation buttons for 'Previous' and 'Next' are visible at the bottom.

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

Here is blinky:

```
import time
import digitalio
import board

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

Create Your settings.toml File

CircuitPython works with WiFi-capable boards to enable you to make projects that have network connectivity. This means working with various passwords and API keys. As of [CircuitPython 8 \(\)](#), there is support for a settings.toml file. This is a file that is stored on your CIRCUITPY drive, that contains all of your secret network information, such as your SSID, SSID password and any API keys for IoT services. It is designed to separate your sensitive information from your code.py file so you are able to share your code without sharing your credentials.

CircuitPython previously used a secrets.py file for this purpose. The settings.toml file is quite similar.

Your settings.toml file should be stored in the main directory of your CIRCUITPY drive. It should not be in a folder.

CircuitPython settings.toml File

This section will provide a couple of examples of what your settings.toml file should look like, specifically for CircuitPython WiFi projects in general.

The most minimal settings.toml file must contain your WiFi SSID and password, as that is the minimum required to connect to WiFi. Copy this example, paste it into your settings.toml, and update:

- your_wifi_ssid
- your_wifi_password

```
CIRCUITPY_WIFI_SSID = "your_wifi_ssid"
CIRCUITPY_WIFI_PASSWORD = "your_wifi_password"
```

Many CircuitPython network-connected projects on the Adafruit Learn System involve using Adafruit IO. For these projects, you must also include your Adafruit IO username and key. Copy the following example, paste it into your settings.toml file, and update:

- `your_wifi_ssid`
- `your_wifi_password`
- `your_aio_username`
- `your_aio_key`

```
CIRCUITPY_WIFI_SSID = "your_wifi_ssid"
CIRCUITPY_WIFI_PASSWORD = "your_wifi_password"
AIO_USERNAME = "your_aio_username"
AIO_KEY = "your_aio_key"
```

Some projects use different variable names for the entries in the settings.toml file. For example, a project might use `AIO_ID` in the place of `AIO_USERNAME`. If you run into connectivity issues, one of the first things to check is that the names in the settings.toml file match the names in the code.

Not every project uses the same variable name for each entry in the settings.toml file! Always verify it matches the code.

settings.toml File Tips

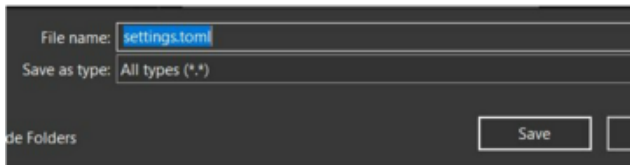
Here is an example settings.toml file.

```
# Comments are supported
CIRCUITPY_WIFI_SSID = "guest wifi"
CIRCUITPY_WIFI_PASSWORD = "guessable"
CIRCUITPY_WEB_API_PORT = 80
CIRCUITPY_WEB_API_PASSWORD = "passw0rd"
test_variable = "this is a test"
thumbs_up = "\U0001f44d"
```

In a settings.toml file, it's important to keep these factors in mind:

- Strings are wrapped in double quotes; ex: `"your-string-here"`
- Integers are not quoted and may be written in decimal with optional sign (`+1`, `-1`, `1000`) or hexadecimal (`0xabcd`).
 - Floats, octal (`0o567`) and binary (`0b11011`) are not supported.

- Use `\u` escapes for weird characters, `\x` and `\ooo` escapes are not available in `.toml` files
 - Example: `\U0001f44d` for 👍 (thumbs up emoji) and `\u20ac` for € (EUR sign)
- Unicode emoji, and non-ASCII characters, stand for themselves as long as you're careful to save in "UTF-8 without BOM" format



When your `settings.toml` file is ready, you can save it in your text editor with the `.toml` extension.

Accessing Your `settings.toml` Information in `code.py`

In your `code.py` file, you'll need to `import` the `os` library to access the `settings.toml` file. Your settings are accessed with the `os.getenv()` function. You'll pass your settings entry to the function to import it into the `code.py` file.

```
import os
print(os.getenv("test_variable"))
```

```
CircuitPython REPL
code.py output:
this is a test

Code done running.

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

In the upcoming CircuitPython WiFi examples, you'll see how the `settings.toml` file is used for connecting to your SSID and accessing your API keys.

CircuitPython Internet Test

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

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

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

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

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

print("ESP32-S2 WebClient Test")

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

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

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

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

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

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

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

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

```

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

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

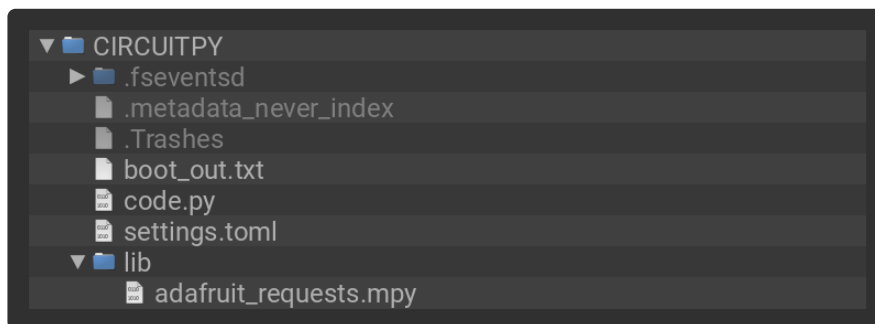
print()

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

print("Done")

```

Your CIRCUITPY drive should resemble the following.



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

The settings.toml File

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

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

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

```

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

# This is where you store the credentials necessary for your code.
# The associated demo only requires WiFi, but you can include any

```

```
# credentials here, such as Adafruit IO username and key, etc.  
CIRCUITPY_WIFI_SSID = "your-wifi-ssid"  
CIRCUITPY_WIFI_PASSWORD = "your-wifi-password"
```

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

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

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

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

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

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

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


```

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
  disconnectededer  RSSI: -86      Channel: 1
  SKJFios-ZV007    RSSI: -83      Channel: 11
  Fios-QIVUQ       RSSI: -83      Channel: 11
  Fios-ZV007       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's MAC address.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Converting Arduino_GFX init strings to CircuitPython

Not all 40-pin displays have the power pins in the same place. Hooking up a non RGB-666 display with the Qualia S3 risks damaging the display.

The conversion script is intended to be run using Python and not CircuitPython on a computer with plenty of memory.

If you would like to generate the init code for CircuitPython, you can do so in a couple of ways with the conversion script below. In both ways you will need to run it using Python on your computer. Start by saving the code below as convert_initcode.py.

```
# convert_initcode.py
# MIT license

import re

(
    BEGIN_WRITE,
    WRITE_COMMAND_8,
    WRITE_COMMAND_16,
    WRITE_DATA_8,
    WRITE_DATA_16,
    WRITE_BYTES,
    WRITE_C8_D8,
    WRITE_C8_D16,
    WRITE_C16_D16,
    END_WRITE,
    DELAY,
) = range(11)

class Encoder:
    def __init__(self):
        self.content = bytearray()
        self.reset()

    def command(self, command):
        if self.pending_command is not None:
            self.flush()
        self.pending_command = command
        self.pending_data = bytearray()

    def data(self, data):
        self.pending_data.append(data)

    def delay(self, value):
        self.pending_delay = value

    def flush(self):
        if self.pending_command is not None:
```

```

        self.content.append(self.pending_command)

        len_and_delay = len(self.pending_data) | (bool(self.pending_delay) << 7)
        self.content.append(len_and_delay)

        self.content.extend(self.pending_data)

        if self.pending_delay:
            self.content.append(self.pending_delay)

    print(f"    {bytes(self.content)}")

    self.reset()

def reset(self):
    self.pending_command = None
    self.pending_data = bytearray()
    self.pending_delay = 0
    self.content = bytearray()

def translate_init_operations(*initcode):
    initcode = iter(initcode)
    encoder = Encoder()

    print("init_code = bytes((")
    for op in initcode:
        if op in (BEGIN_WRITE, END_WRITE):
            continue
        elif op == WRITE_COMMAND_8:
            encoder.command(next(initcode))
        elif op == WRITE_C8_D8:
            encoder.command(next(initcode))
            encoder.data(next(initcode))
        elif op == WRITE_C8_D16:
            encoder.command(next(initcode))
            encoder.data(next(initcode))
            encoder.data(next(initcode))
        elif op == WRITE_BYTES:
            for _ in range(next(initcode)):
                encoder.data(next(initcode))
        elif op == DELAY:
            encoder.delay(next(initcode))
        else:
            raise ValueError(f"Invalid operation 0x{op:02x}")
    encoder.flush()
    print("))")

def translate_init_file(initcode_filename):
    initcode_regex = r"\( *0x([0-9a-fA-F]+) *\);"
    command_data_regex = r"\( *0x([0-9a-fA-F]+), *0x([0-9a-fA-F]+) *\);"
    delay_regex = r"\( *(\d+) *\);"

    # Init code files are inconsistent in their naming of command, data, and delay
    functions
    command8_values = ("SPI_WriteComm", "W_C", "WriteCommand", "WriteComm")
    data_values = ("SPI_WriteData", "W_D", "WriteParameter", "WriteData")
    delay_values = ("Delays", "Delay", "Delay_ms")

    encoder = Encoder()

    def get_command8(line):
        for command in command8_values:
            if command in line:
                encoder.command(get_initcode8(line))
                return True
        return False

    def get_data(line):
        for data in data_values:

```

```

        if data in line:
            encoder.data(get_initcode8(line))
            return True
    return False

def get_delay(line):
    for delay in delay_values:
        if delay in line:
            encoder.delay(get_delay_value(line))
            return True
    return False

def get_initcode8(line):
    match = re.search(initcode_regex, line)
    if match:
        return int(match.group(1), 16)
    raise ValueError(f"Warning: could not parse initcode in line '{line}'")

def get_initcode16(line):
    match = re.search(command_data_regex, line)
    if match:
        command = int(match.group(1), 16)
        data = int(match.group(2), 16)
        return command, data
    raise ValueError(f"Warning: could not parse initcode in line '{line}'")

def get_delay_value(line):
    match = re.search(delay_regex, line)
    if match:
        return int(match.group(1))
    raise ValueError(f"Warning: could not parse delay in line '{line}'")

with open(initcode_filename, encoding='utf-8', errors='ignore') as f:
    print("init_code = bytes((")
    for line in f.readlines():
        line = line.strip()
        # Remove comments or commented out code
        line = line.split("//")[0]
        if get_command8(line):
            continue
        if get_data(line):
            continue
        if get_delay(line):
            continue
        if "Wrt_Reg_3052" in line:
            command, data = get_initcode16(line)
            encoder.command(command)
            encoder.data(data)
    encoder.flush()
    print("))")

```

To use the code, you can just import the conversion script into your own code:

```
from convert_initcode import *
```

Using Arduino_GFX Init Codes

The first method is to convert the codes from the Arduino_GFX library, which can be found at https://github.com/moononournation/Arduino_GFX (). The initialization codes for the dot clock displays are found inside of [src/display/Arduino_RGB_Display.h](#) () by searching for the display's model number.

In your main script, just call `translate_init_operations()` like this:

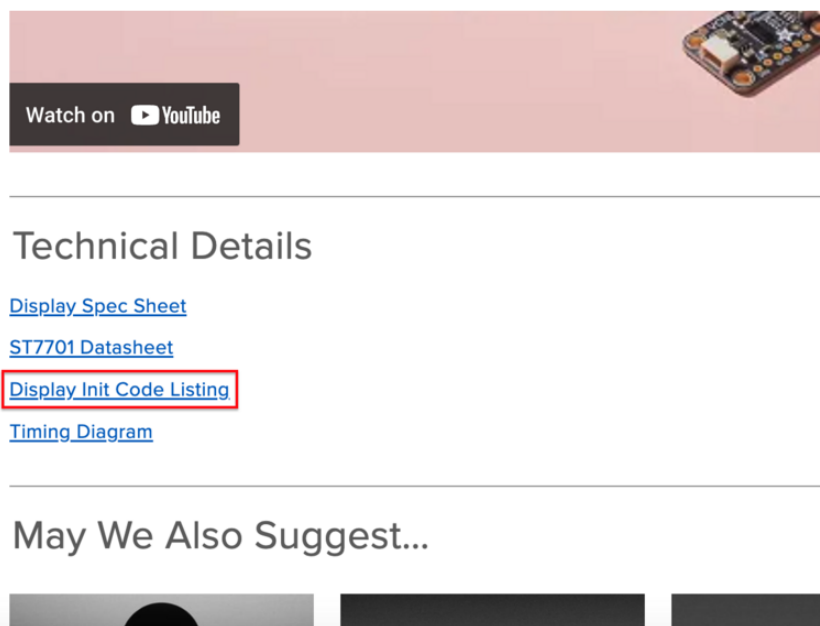
```
translate_init_operations(  
  WRITE_COMMAND_8, 0xFF,  
  WRITE_BYTES, 5, 0x77, 0x01, 0x00, 0x00, 0x13,  
  WRITE_C8_D8, 0xEF, 0x08,  
  ...  
)
```

These operations are supported: `COMMAND_8`, `C8_D8`, `C8_D16`, `WRITE_BYTES`. Adding support for `WRITE_DATA_8` and `WRITE_DATA_16` "should be easy" but it was not used in any examples so far.

It's assumed that `BEGIN_WRITE` / `END_WRITE` are not 'important'. However, `DELAY` is accounted for.

Using Init Code Files

The second method is by using one of the init code files found on the product page for the display. Near the bottom of the page under Technical Details, most of the displays have a link to a file containing the init codes. Just save the file to your computer as something like `display_init_codes.txt`.



Then to convert the file, in your main script, just run `translate_init_file()` like this:

```
translate_init_file("display_init_codes.txt")
```

Script Output

After running your script, you should see output like the following:

```
init_code = bytes((
    b'\xff\x05w\x01\x00\x00\x13'
    b'\xef\x01\x08'
    b'\xff\x05w\x01\x00\x00\x10'
    b'\xc0\x02w\x00'
    b'\xc1\x02\x11\x0c'
    b'\xc2\x02\x07\x02'
    b'\xcc\x010'
    b'\xb0\x10\x06\xcf\x14\x0c\x0f\x03\x00\n\x07\x1b\x03\x12\x10%6\x1e'
    b'\xb1\x10\x0c\xd4\x18\x0c\xe\x06\x03\x06\x08#\x06\x12\x100/\x1f'
    b'\xff\x05w\x01\x00\x00\x11'
    b'\xb0\x01s'
    b'\xb1\x01l'
    b'\xb2\x01x83'
    b'\xb3\x01x80'
```

Determining Timings

If you have your own RGB-666 display, you may wish to use it with the Qualia ESP32-S3. The main pieces of information that you will need to find are:

- Display Width
- Display Height
- Horizontal and Vertical:
 - Sync Pulse Width in milliseconds
 - Front Porch in milliseconds
 - Back Porch in milliseconds

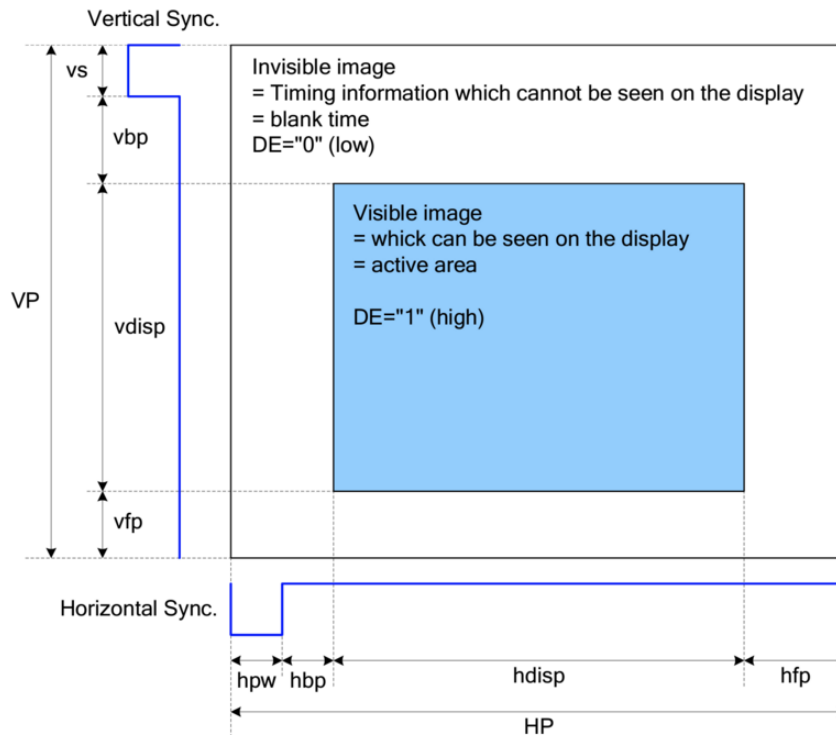
Pieces of Information that are helpful, but can be determined by trial and error include:

- Frequency of the Display Clock
- Signal Polarities for the following:
 - Horizontal Idle
 - Vertical Idle
 - Data Enable Idle
 - Pixel Clock Active
 - Pixel Clock Idle

Some displays ignore many of the timing settings and use ones set through the init codes.

Using a Data Sheet

The one of the best places to start looking for this information is the data sheet for the display. Data sheets may contain a diagram that will give you most of those values:



For the display width and height, these are in pixels and should be easy to find.

In the above diagram, you can see for instance the HP (or Horizontal Period) split up into **hpw** (or Horizontal Sync Pulse Width), **hbp** (or Horizontal Back Porch), **hdisp** (or Horizontal Display, which is the visible area), and **hfp** (or Horizontal Front Porch). For the vertical, this is the same except **vs** is used for the Vertical Sync Pulse Width.

When a display is drawn, the horizontal and vertical periods are split up into these sections. The Sync Pulse Widths are used by the display to keep everything in sync and the Front and Back Porch are blanking periods and are carried over from VGA when CRTs (or Cathode Ray Tubes) were used to give a little extra time for signals to synchronize or allow the electron beam to move to a different place.

While many data sheets will explicitly give you these values, occasionally you may be given values such as the total period time, one of the porch values and the timing of

the display data, which you can use to calculate any missing values, which is why it's important to understand how the timings are used.

You can also use the diagram to figure out the Horizontal and Vertical Idle Polarity by looking at the lines underneath and to the left. In the case of the above diagram, both of the signals have a high idle state, which is the part of the signal where it is out of the sync pulse phase.

Fill in the Settings

For the timings in CircuitPython, a dictionary is used. You can use the following code as a template and you will want to replace anywhere you see `[Number]` with the actual numerical value and anywhere you see `[True/False]` with a boolean value.

```
tft_timings = {
    "frequency": [Number],
    "width": [Number],
    "height": [Number],

    "hsync_pulse_width": [Number],
    "hsync_back_porch": [Number],
    "hsync_front_porch": [Number],
    "hsync_idle_low": [True/False],

    "vsync_pulse_width": [Number],
    "vsync_back_porch": [Number],
    "vsync_front_porch": [Number],
    "vsync_idle_low": [True/False],

    "pclk_active_high": [True/False],
    "pclk_idle_high": [True/False],
    "de_idle_high": [True/False],
}
```

Experimenting with Settings

To get the remainder of the settings, you may need to experiment a bit. You can take a look at some of the other displays that are similar to get a good starting point. From there, start making adjustments until you get an image that looks correct. If you notice that any changes you are making seem to have little effect, then it is likely using settings from the init codes. In this case, you may need to consult the data sheet for the controller and figure out which code is causing issues. You also may try getting the init codes from different sources and see which ones work the best.

Testing your Settings with CircuitPython

So you are at a point where everything seems correct, it's time to test that it all looks good. If the settings are off just a bit, you may notice certain colors look a bit glitchy

and you will need to continue experimenting with the settings to fix it. You can fill in your timing settings and run this script to test that everything looks good:

```
from displayio import release_displays
release_displays()

import random
import displayio
import time
import busio
import board
import dotclockframebuffer
from framebufferio import FramebufferDisplay

init_code = bytes(...))

board.I2C().deinit()
i2c = busio.I2C(board.SCL, board.SDA, frequency=400_000)
tft_io_expander = dict(board.TFT_IO_EXPANDER)
dotclockframebuffer.ioexpander_send_init_sequence(i2c, init_sequence_tl032,
**tft_io_expander)
i2c.deinit()

tft_pins = dict(board.TFT_PINS)

tft_timings = {...}

bitmap = displayio.Bitmap(256, 7*64, 65535)
fb = dotclockframebuffer.DotClockFramebuffer(**tft_pins, **tft_timings)
display = FramebufferDisplay(fb, auto_refresh=False)

# Create a TileGrid to hold the bitmap
tile_grid = displayio.TileGrid(bitmap,
pixel_shader=displayio.ColorConverter(input_colorspace=displayio.Colorspace.RGB565))

# Create a Group to hold the TileGrid
group = displayio.Group()

# Add the TileGrid to the Group
group.append(tile_grid)

# Add the Group to the Display
display.root_group = group

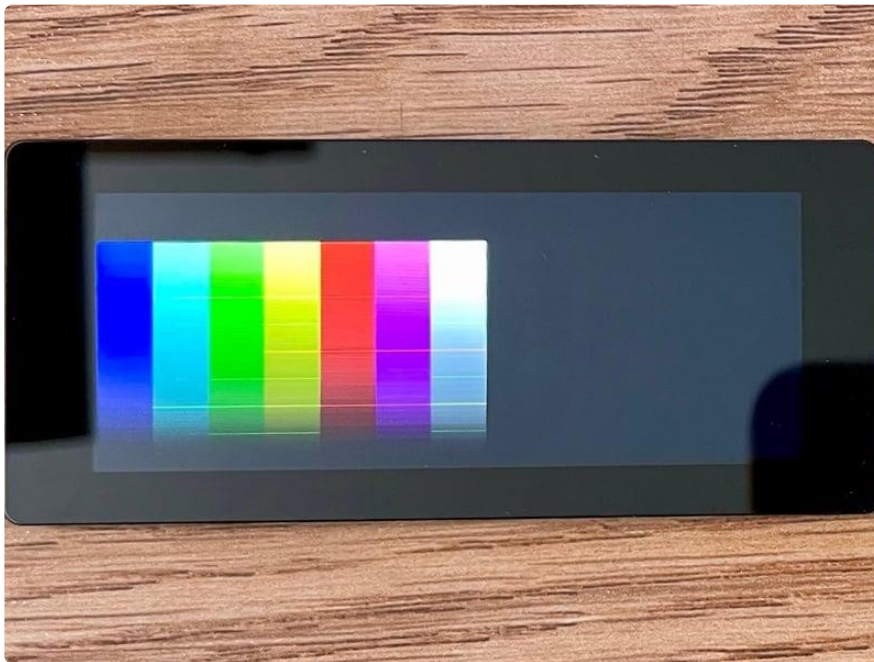
display.auto_refresh = True

for i in range(256):
    b = i >>> 3
    g = (i >>> 2) <<< 5
    r = b <<< 11
    for j in range(64):
        bitmap[i, j] = b
        bitmap[i, j+64] = b|g
        bitmap[i, j+128] = g
        bitmap[i, j+192] = g|r
        bitmap[i, j+256] = r
        bitmap[i, j+320] = r|b
        bitmap[i, j+384] = r|g|b

# Loop forever so you can enjoy your image
while True:
    time.sleep(1)
    display.auto_refresh = False
    group.x = random.randint(0, 32)
    group.y = random.randint(0, 32)
```

```
display.auto_refresh = True
pass
```

If your settings are slightly off, it may look like the following:



Once everything is set correctly, the above image should look like this:



CircuitPython Display Setup

Not all 40-pin displays have the power pins in the same place. Hooking up a non RGB-666 display with the Qualia S3 risks damaging the display.

To set up a display, you need to have several major pieces of information:

- The GPIO connections for the display (TFT_PINS)
- The I/O expander configuration (TFT_IO_EXPANDER)
- The resolution and "timings" of the display (TFT_TIMINGS), which also includes information about the polarity of certain signals.
- The initialization code of the display (TFT_INIT_SEQUENCE)

Luckily, this guide provides all the information for the displays that are sold in the Adafruit shop. However, if you have a different display, you will need to find the information in the data sheet.

If a board is designed for dot clock TFT displays, the GPIO connections are listed in `board.TFT_PINS`. Otherwise, it depends on how the display is connected.

If the board is designed for a single display, then the timings are listed in `board.TFT_TIMINGS`.

These values are used in the display constructor with the `**` so that each element becomes a separate argument to the function.

If the board's built in display requires an initialization sequence, then this is given as `board.TFT_INIT_SEQUENCE`. If the SPI bus is on an I2C I/O expander the settings for the I/O expander are in `board.TFT_IO_EXPANDER`, intended to be expanded with `**`.

If a board is tied to a specific display, then board definition can initialize the dot clock TFT display. For example, this is done with the Espressif ESP32-S3 LCD EV board:

```
#include "py/objtuple.h"
#include "boards/espressif_esp32s3_lcd_ev/board.h"
#include "shared-bindings/board/__init__.h"
#include "shared-module/displayio/__init__.h"

STATIC const mp_rom_map_elem_t tft_io_expander_table[] = {
    { MP_ROM_QSTR(MP_QSTR_i2c_address), MP_ROM_INT(0x20)},
    { MP_ROM_QSTR(MP_QSTR_gpio_address), MP_ROM_INT(1)},
    { MP_ROM_QSTR(MP_QSTR_gpio_data_len), MP_ROM_INT(1)},
    { MP_ROM_QSTR(MP_QSTR_gpio_data), MP_ROM_INT(0xF1)},
    { MP_ROM_QSTR(MP_QSTR_cs_bit), MP_ROM_INT(1)},
    { MP_ROM_QSTR(MP_QSTR_mosi_bit), MP_ROM_INT(3)},
    { MP_ROM_QSTR(MP_QSTR_clk_bit), MP_ROM_INT(2)},
    { MP_ROM_QSTR(MP_QSTR_i2c_init_sequence), &i2c_init_byte_obj},
};
MP_DEFINE_CONST_DICT(tft_io_expander_dict, tft_io_expander_table);

STATIC const mp_rom_obj_tuple_t tft_r_pins = {
    {&mp_type_tuple},
    5,
```

```

    {
        MP_ROM_PTR(&pin_GPI01),
        MP_ROM_PTR(&pin_GPI02),
        MP_ROM_PTR(&pin_GPI042),
        MP_ROM_PTR(&pin_GPI041),
        MP_ROM_PTR(&pin_GPI040),
    }
};

STATIC const mp_rom_obj_tuple_t tft_g_pins = {
    {&mp_type_tuple},
    6,
    {
        MP_ROM_PTR(&pin_GPI021),
        MP_ROM_PTR(&pin_GPI047),
        MP_ROM_PTR(&pin_GPI048),
        MP_ROM_PTR(&pin_GPI045),
        MP_ROM_PTR(&pin_GPI038),
        MP_ROM_PTR(&pin_GPI039),
    }
};

STATIC const mp_rom_obj_tuple_t tft_b_pins = {
    {&mp_type_tuple},
    5,
    {
        MP_ROM_PTR(&pin_GPI010),
        MP_ROM_PTR(&pin_GPI011),
        MP_ROM_PTR(&pin_GPI012),
        MP_ROM_PTR(&pin_GPI013),
        MP_ROM_PTR(&pin_GPI014),
    }
};

STATIC const mp_rom_map_elem_t tft_pins_table[] = {
    { MP_ROM_QSTR(MP_QSTR_de), MP_ROM_PTR(&pin_GPI017) },
    { MP_ROM_QSTR(MP_QSTR_vsync), MP_ROM_PTR(&pin_GPI03) },
    { MP_ROM_QSTR(MP_QSTR_hsync), MP_ROM_PTR(&pin_GPI046) },
    { MP_ROM_QSTR(MP_QSTR_dclk), MP_ROM_PTR(&pin_GPI09) },
    { MP_ROM_QSTR(MP_QSTR_red), MP_ROM_PTR(&tft_r_pins) },
    { MP_ROM_QSTR(MP_QSTR_green), MP_ROM_PTR(&tft_g_pins) },
    { MP_ROM_QSTR(MP_QSTR_blue), MP_ROM_PTR(&tft_b_pins) },
};
MP_DEFINE_CONST_DICT(tft_pins_dict, tft_pins_table);

STATIC const mp_rom_map_elem_t tft_timings_table[] = {
    { MP_ROM_QSTR(MP_QSTR_frequency), MP_ROM_INT(6500000) }, // nominal 16MHz, but
    display is unstable/tears at that frequency
    { MP_ROM_QSTR(MP_QSTR_width), MP_ROM_INT(480) },
    { MP_ROM_QSTR(MP_QSTR_height), MP_ROM_INT(480) },
    { MP_ROM_QSTR(MP_QSTR_hsync_pulse_width), MP_ROM_INT(13) },
    { MP_ROM_QSTR(MP_QSTR_hsync_front_porch), MP_ROM_INT(20) },
    { MP_ROM_QSTR(MP_QSTR_hsync_back_porch), MP_ROM_INT(40) },
    { MP_ROM_QSTR(MP_QSTR_hsync_idle_low), MP_ROM_FALSE },
    { MP_ROM_QSTR(MP_QSTR_vsync_pulse_width), MP_ROM_INT(15) },
    { MP_ROM_QSTR(MP_QSTR_vsync_front_porch), MP_ROM_INT(20) },
    { MP_ROM_QSTR(MP_QSTR_vsync_back_porch), MP_ROM_INT(40) },
    { MP_ROM_QSTR(MP_QSTR_vsync_idle_low), MP_ROM_FALSE },
    { MP_ROM_QSTR(MP_QSTR_de_idle_high), MP_ROM_FALSE },
    { MP_ROM_QSTR(MP_QSTR_pclk_active_high), MP_ROM_FALSE },
    { MP_ROM_QSTR(MP_QSTR_pclk_idle_high), MP_ROM_FALSE },
};
MP_DEFINE_CONST_DICT(tft_timings_dict, tft_timings_table);

STATIC const mp_rom_map_elem_t board_module_globals_table[] = {
    CIRCUITPYTHON_BOARD_DICT_STANDARD_ITEMS

    { MP_ROM_QSTR(MP_QSTR_TFT_PINS), MP_ROM_PTR(&tft_pins_dict) },
    { MP_ROM_QSTR(MP_QSTR_TFT_TIMINGS), MP_ROM_PTR(&tft_timings_dict) },
};

```

```

    { MP_ROM_QSTR(MP_QSTR_TFT_IO_EXPANDER), MP_ROM_PTR(&tft_io_expander_dict) },
    { MP_ROM_QSTR(MP_QSTR_TFT_INIT_SEQUENCE), &display_init_byte_obj},

    { MP_ROM_QSTR(MP_QSTR_I2S_SCK), MP_ROM_PTR(&pin_GPIO16) },
    { MP_ROM_QSTR(MP_QSTR_I2S_MCLK), MP_ROM_PTR(&pin_GPIO5) },
    { MP_ROM_QSTR(MP_QSTR_I2S_WS), MP_ROM_PTR(&pin_GPIO7) },
    { MP_ROM_QSTR(MP_QSTR_I2S_SDO), MP_ROM_PTR(&pin_GPIO6) },

    { MP_ROM_QSTR(MP_QSTR_TX), MP_ROM_PTR(&pin_GPIO43) },
    { MP_ROM_QSTR(MP_QSTR_RX), MP_ROM_PTR(&pin_GPIO44) },

    { MP_ROM_QSTR(MP_QSTR_SCL), MP_ROM_PTR(DEFAULT_I2C_BUS_SCL) },
    { MP_ROM_QSTR(MP_QSTR_SDA), MP_ROM_PTR(DEFAULT_I2C_BUS_SDA) },

    { MP_ROM_QSTR(MP_QSTR_DISPLAY), MP_ROM_PTR(&displays[0].display) },

    // boot mode button can be used in SW as well
    { MP_ROM_QSTR(MP_QSTR_BUTTON), MP_ROM_PTR(&pin_GPIO0) },

    { MP_ROM_QSTR(MP_QSTR_I2C), MP_ROM_PTR(&board_i2c_obj) },
};
MP_DEFINE_CONST_DICT(board_module_globals, board_module_globals_table);

```

```

/*
 * This file is part of the MicroPython project, http://micropython.org/
 *
 * The MIT License (MIT)
 *
 * Copyright (c) 2020 Scott Shawcroft for Adafruit Industries
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 * of this software and associated documentation files (the "Software"), to deal
 * in the Software without restriction, including without limitation the rights
 * to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
 * copies of the Software, and to permit persons to whom the Software is
 * furnished to do so, subject to the following conditions:
 *
 * The above copyright notice and this permission notice shall be included in
 * all copies or substantial portions of the Software.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
 * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
 * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
 * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
 * OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
 * THE SOFTWARE.
 */

#include "supervisor/board.h"
#include "mpconfigboard.h"
#include "shared-bindings/board/__init__.h"
#include "shared-bindings/busio/I2C.h"
#include "shared-bindings/dotclockframebuffer/DotClockFramebuffer.h"
#include "shared-bindings/dotclockframebuffer/__init__.h"
#include "shared-bindings/framebufferio/FramebufferDisplay.h"
#include "shared-bindings/microcontroller/Pin.h"
#include "shared-module/displayio/__init__.h"
#include "boards/espressif_esp32s3_lcd_ev/board.h"

#define MP_DEFINE_BYTES_OBJ(obj_name, bin) mp_obj_str_t obj_name =
    {{&mp_type_bytes}, 0, sizeof(bin) - 1, (const byte *)bin}

static const uint8_t display_init_sequence[] = {
    0xf0, 5, 0x55, 0xaa, 0x52, 0x08, 0x00,
    0xf6, 2, 0x5a, 0x87,
    0xc1, 1, 0x3f,

```

```

0xc2, 1, 0x0e,
0xc6, 1, 0xf8,
0xc9, 1, 0x10,
0xcd, 1, 0x25,
0xf8, 1, 0x8a,
0xac, 1, 0x45,
0xa0, 1, 0xdd,
0xa7, 1, 0x47,
0xfa, 4, 0x00, 0x00, 0x00, 0x04,
0x86, 4, 0x99, 0xa3, 0xa3, 0x51,
0xa3, 1, 0xee,
0xfd, 3, 0x3c, 0x3c, 0x00,
0x71, 1, 0x48,
0x72, 1, 0x48,
0x73, 2, 0x00, 0x44,
0x97, 1, 0xee,
0x83, 1, 0x93,
0x9a, 1, 0x72,
0x9b, 1, 0x5a,
0x82, 2, 0x2c, 0x2c,
0xb1, 1, 0x10,
0x6d, 32, 0x00, 0x1f, 0x19, 0x1a, 0x10, 0x0e, 0x0c, 0x0a, 0x02, 0x07, 0x1e,
0x1e, 0x1e, 0x1e, 0x1e, 0x1e, 0x1e, 0x1e, 0x1e, 0x1e, 0x1e, 0x08, 0x01, 0x09,
0x0b, 0x0d, 0x0f, 0x1a, 0x19, 0x1f, 0x00,
0x64, 16, 0x38, 0x05, 0x01, 0xdb, 0x03, 0x03, 0x38, 0x04, 0x01, 0xdc, 0x03,
0x03, 0x7a, 0x7a, 0x7a, 0x7a,
0x65, 16, 0x38, 0x03, 0x01, 0xdd, 0x03, 0x03, 0x38, 0x02, 0x01, 0xde, 0x03,
0x03, 0x7a, 0x7a, 0x7a, 0x7a,
0x66, 16, 0x38, 0x01, 0x01, 0xdf, 0x03, 0x03, 0x38, 0x00, 0x01, 0xe0, 0x03,
0x03, 0x7a, 0x7a, 0x7a, 0x7a,
0x67, 16, 0x30, 0x01, 0x01, 0xe1, 0x03, 0x03, 0x30, 0x02, 0x01, 0xe2, 0x03,
0x03, 0x7a, 0x7a, 0x7a, 0x7a,
0x68, 13, 0x00, 0x08, 0x15, 0x08, 0x15, 0x7a, 0x7a, 0x08, 0x15, 0x08, 0x15,
0x7a, 0x7a,
0x60, 8, 0x38, 0x08, 0x7a, 0x7a, 0x38, 0x09, 0x7a, 0x7a,
0x63, 8, 0x31, 0xe4, 0x7a, 0x7a, 0x31, 0xe5, 0x7a, 0x7a,
0x69, 7, 0x04, 0x22, 0x14, 0x22, 0x14, 0x22, 0x08,
0x6b, 1, 0x07,
0x7a, 2, 0x08, 0x13,
0x7b, 2, 0x08, 0x13,
0xd1, 52, 0x00, 0x00, 0x00, 0x04, 0x00, 0x12, 0x00, 0x18, 0x00, 0x21, 0x00,
0x2a, 0x00, 0x35, 0x00, 0x47, 0x00, 0x56, 0x00, 0x90, 0x00, 0xe5, 0x01, 0x68, 0x01,
0xd5, 0x01, 0xd7, 0x02, 0x36, 0x02, 0xa6, 0x02, 0xee, 0x03, 0x48, 0x03, 0xa0, 0x03,
0xba, 0x03, 0xc5, 0x03, 0xd0, 0x03, 0xe0, 0x03, 0xea, 0x03, 0xfa, 0x03, 0xff,
0xd2, 52, 0x00, 0x00, 0x00, 0x04, 0x00, 0x12, 0x00, 0x18, 0x00, 0x21, 0x00,
0x2a, 0x00, 0x35, 0x00, 0x47, 0x00, 0x56, 0x00, 0x90, 0x00, 0xe5, 0x01, 0x68, 0x01,
0xd5, 0x01, 0xd7, 0x02, 0x36, 0x02, 0xa6, 0x02, 0xee, 0x03, 0x48, 0x03, 0xa0, 0x03,
0xba, 0x03, 0xc5, 0x03, 0xd0, 0x03, 0xe0, 0x03, 0xea, 0x03, 0xfa, 0x03, 0xff,
0xd3, 52, 0x00, 0x00, 0x00, 0x04, 0x00, 0x12, 0x00, 0x18, 0x00, 0x21, 0x00,
0x2a, 0x00, 0x35, 0x00, 0x47, 0x00, 0x56, 0x00, 0x90, 0x00, 0xe5, 0x01, 0x68, 0x01,
0xd5, 0x01, 0xd7, 0x02, 0x36, 0x02, 0xa6, 0x02, 0xee, 0x03, 0x48, 0x03, 0xa0, 0x03,
0xba, 0x03, 0xc5, 0x03, 0xd0, 0x03, 0xe0, 0x03, 0xea, 0x03, 0xfa, 0x03, 0xff,
0xd4, 52, 0x00, 0x00, 0x00, 0x04, 0x00, 0x12, 0x00, 0x18, 0x00, 0x21, 0x00,
0x2a, 0x00, 0x35, 0x00, 0x47, 0x00, 0x56, 0x00, 0x90, 0x00, 0xe5, 0x01, 0x68, 0x01,
0xd5, 0x01, 0xd7, 0x02, 0x36, 0x02, 0xa6, 0x02, 0xee, 0x03, 0x48, 0x03, 0xa0, 0x03,
0xba, 0x03, 0xc5, 0x03, 0xd0, 0x03, 0xe0, 0x03, 0xea, 0x03, 0xfa, 0x03, 0xff,
0xd5, 52, 0x00, 0x00, 0x00, 0x04, 0x00, 0x12, 0x00, 0x18, 0x00, 0x21, 0x00,
0x2a, 0x00, 0x35, 0x00, 0x47, 0x00, 0x56, 0x00, 0x90, 0x00, 0xe5, 0x01, 0x68, 0x01,
0xd5, 0x01, 0xd7, 0x02, 0x36, 0x02, 0xa6, 0x02, 0xee, 0x03, 0x48, 0x03, 0xa0, 0x03,
0xba, 0x03, 0xc5, 0x03, 0xd0, 0x03, 0xe0, 0x03, 0xea, 0x03, 0xfa, 0x03, 0xff,
0xd6, 52, 0x00, 0x00, 0x00, 0x04, 0x00, 0x12, 0x00, 0x18, 0x00, 0x21, 0x00,
0x2a, 0x00, 0x35, 0x00, 0x47, 0x00, 0x56, 0x00, 0x90, 0x00, 0xe5, 0x01, 0x68, 0x01,
0xd5, 0x01, 0xd7, 0x02, 0x36, 0x02, 0xa6, 0x02, 0xee, 0x03, 0x48, 0x03, 0xa0, 0x03,
0xba, 0x03, 0xc5, 0x03, 0xd0, 0x03, 0xe0, 0x03, 0xea, 0x03, 0xfa, 0x03, 0xff,
0x3a, 1, 0x66,
0x3a, 1, 0x66,
0x11, 0x80, 120,
0x29, 0x0,
0, // trailing NUL for Python bytes() representation

```

```

};
MP_DEFINE_BYTES_OBJ(display_init_byte_obj, display_init_sequence);

static const char i2c_bus_init_sequence[] = {
    2, 3, 0xf1, // set GPIO direction
    2, 2, 0, // disable all output inversion
    0, // trailing NUL for Python bytes() representation
};
MP_DEFINE_BYTES_OBJ(i2c_init_byte_obj, i2c_bus_init_sequence);

static const mcu_pin_obj_t *red_pins[] = {
    &pin_GPIO1, &pin_GPIO2, &pin_GPIO42, &pin_GPIO41, &pin_GPIO40
};
static const mcu_pin_obj_t *green_pins[] = {
    &pin_GPIO21, &pin_GPIO47, &pin_GPIO48, &pin_GPIO45, &pin_GPIO38, &pin_GPIO39
};
static const mcu_pin_obj_t *blue_pins[] = {
    &pin_GPIO10, &pin_GPIO11, &pin_GPIO12, &pin_GPIO13, &pin_GPIO14
};
void board_init(void) {
    dotclockframebuffer_framebuffer_obj_t *framebuffer =
    &allocate_display_bus_or_raise()->dotclock;
    framebuffer->base.type = &dotclockframebuffer_framebuffer_type;

    common_hal_dotclockframebuffer_framebuffer_construct(
        framebuffer,
        /* de */ &pin_GPIO17,
        /* vsync */ &pin_GPIO3,
        /* hsync */ &pin_GPIO46,
        /* dclk */ &pin_GPIO9,
        /* data */ red_pins, MP_ARRAY_SIZE(red_pins), green_pins,
MP_ARRAY_SIZE(green_pins), blue_pins, MP_ARRAY_SIZE(blue_pins),
        /* frequency */ 12000000,
        /* width x height */ 480, 480,
        /* horizontal: pulse, back & front porch, idle */ 13, 20, 40, false,
        /* vertical: pulse, back & front porch, idle */ 15, 20, 40, false,
        /* de_idle_high */ false,
        /* pclk_active_high */ true,
        /* pclk_idle_high */ false,
        /* overscan_left */ 0
    );

    framebufferio_framebufferdisplay_obj_t *disp = &allocate_display_or_raise()-
    >framebuffer_display;
    disp->base.type = &framebufferio_framebufferdisplay_type;
    common_hal_framebufferio_framebufferdisplay_construct(
        disp,
        framebuffer,
        0,
        true
    );

    busio_i2c_obj_t i2c;
    i2c.base.type = &busio_i2c_type;
    common_hal_busio_i2c_construct(&i2c, DEFAULT_I2C_BUS_SCL, DEFAULT_I2C_BUS_SDA,
400000, 255);
    const int i2c_device_address = 32;

    dotclockframebuffer_ioexpander_spi_bus spibus = {
        .bus = &i2c,
        .i2c_device_address = i2c_device_address,
        .i2c_write_size = 2,
        .addr_reg_shadow = { .u32 = 1 }, // GPIO data at register 1
        .cs_mask = 0x100 << 1, // data payload is at byte 2
        .mosi_mask = 0x100 << 3,
        .clk_mask = 0x100 << 2,
    };
};

static const mp_buffer_info_t bufinfo_display_init = { (void

```



```

*)display_init_sequence, sizeof(display_init_sequence) - 1 };
    static const mp_buffer_info_t bufinfo_i2c_bus_init = { (void
*)i2c_bus_init_sequence, sizeof(i2c_bus_init_sequence) - 1 };
    dotclockframebuffer_ioexpander_send_init_sequence(&spibus,
&bufinfo_i2c_bus_init, &bufinfo_display_init);

    common_hal_busio_i2c_deinit(&i2c);
}

// Use the MP_WEAK supervisor/shared/board.c versions of routines not defined here.

```

Example TFT_PINS

The TFT_PINS should be arranged in a Python dict. For the Qualia ESP32-S3, you can simply use `board.TFT_PINS`. They should be arranged similar to the Espressif LCD EV board's `TFT_PINS`:

```

{
    "de": microcontroller.pin.GPI017,
    "vsync": microcontroller.pin.GPI03,
    "hsync": microcontroller.pin.GPI046,
    "dclk": microcontroller.pin.GPI09,
    "red": (
        microcontroller.pin.GPI01,
        microcontroller.pin.GPI02,
        microcontroller.pin.GPI042,
        microcontroller.pin.GPI041,
        microcontroller.pin.GPI040,
    ),
    "green": (
        microcontroller.pin.GPI021,
        microcontroller.pin.GPI047,
        microcontroller.pin.GPI048,
        microcontroller.pin.GPI045,
        microcontroller.pin.GPI038,
        microcontroller.pin.GPI039,
    ),
    "blue": (
        microcontroller.pin.GPI010,
        microcontroller.pin.GPI011,
        microcontroller.pin.GPI012,
        microcontroller.pin.GPI013,
        microcontroller.pin.GPI014,
    ),
}

```

Example TFT_TIMINGS

The specific timings can be found in the display datasheet or, for displays sold through the Adafruit store, on the page for the specific display in this guide.

As an example, here are the timings for the 480x480 display from the Espressif LCD EVK:

```

TFT_TIMINGS = {
    "frequency": 6_500_000, # should be 18_000_000,

```

```

"width": 480,
"height": 480,
"hsync_pulse_width": 13,
"hsync_front_porch": 40,
"hsync_back_porch": 20,
"vsync_pulse_width": 15,
"vsync_front_porch": 40,
"vsync_back_porch": 20,
"hsync_idle_low": False,
"vsync_idle_low": False,
"de_idle_high": False,
"pclk_active_high": True,
"pclk_idle_high": False,
}

```

Timings for the 720x720 square display, which does not require a SPI init sequence, would look like this:

```

tft_timings = {
"frequency": 6_500_000,
"width": 720,
"height": 720,
"hsync_pulse_width": 20,
"hsync_front_porch": 40,
"hsync_back_porch": 40,
"vsync_pulse_width": 10,
"vsync_front_porch": 40,
"vsync_back_porch": 40,
"hsync_idle_low": False,
"vsync_idle_low": False,
"de_idle_high": False,
"pclk_active_high": False,
"pclk_idle_high": False,
}

```

I/O Expander

The `dotclockframebuffer.ioexpander_send_init_sequence()` () function supports a "generic I2C I/O expander". Generic meaning:

- Any I2C address can be used.
- Any GPIO register address can be used.
- GPIO data can be 1 or 2 bytes (8 or 16 bits).
- Arbitrary I2C registers can be initialized for setting direction, pull, inversion, etc.
- State of other GPIO bits can be specified explicitly to avoid undesirable pin state changes.

Here are some values for a PCA9554 expander. This is the IO expander used on the Qualia ESP32-S3 and the values can be found in `board.TFT_IO_EXPANDER`:

- `i2c_address=0x3f`
- `gpio_address=1` (the GPIO output register address)

- `gpio_data_len=1` (1 byte of data)
- `gpio_data=0xfd` (value of other GPIOs on expander)
- `cs_bit=1` (index of chip select)
- `mosi_bit=7` (index of data out)
- `clk_bit=0` (index of clock)
- `reset_bit=2` (optional index of reset pin)
- `i2c_init_sequence=b'...'` (other register settings, see below)

I2C Initialization Sequence

Using an I2C init sequence lets arbitrary registers on the I/O expander be set.

It is composed of a series of commands, starting with a byte length. Each is sent to the I/O expander I2C address.

Typical for PCA9554 expander:

```
i2c_init_sequence=bytes((
  2, 3, 0x78, # set pin direction (register 3) to 0x78 (0-bit is output mode)
  2, 2, 0     # disable output inverts (register 2) to 0
))
```

Display Initialization Code

Some dot clock displays require "initialization code" to be sent on a unidirectional 3-wire bus. The data is transmitted in "mode 0", which is 9 bits long. The top bit specifies whether the code byte is data or a command, with `0` being command and `1` being data.

The structure of the initialization data is a series of commands. Each command can have associated data and an associated delay:

- First byte: 8-bit command value
- Second byte: 7-bit data length (may be zero). The top bit (0x80) is set if a delay byte follows the data
- Variable number of bytes: 8-bit data values
- Optional: 8-bit delay value.

The delay value, if specified, is in milliseconds. The special delay value of `255` or `0xFF` is treated as 500 milliseconds.

Display initialization codes are the same as the ones used by [displayio.FourWire \(\)](#) except that the default after each data block is no delay instead of 10ms.

Example 1

The following byte sequence sends the command 0xfa followed by 4 bytes of data and no delay:

```
0xfa, 4, 0x00, 0x00, 0x00, 0x04,
```

Example 2

The following byte sequence sends the command 0x11, no data, and then delays by a minimum of 120ms:

```
0x11, 0x80, 120,
```

Sending Initialization Code via I2C IO Expander

There is special support for sending initialization code over an I2C IO expander chip. This requires a series of steps:

1. Construct the I2C bus object.
2. Call `ioexpander_send_init_sequence()` with the appropriate values
3. Optionally, deconstruct the I2C bus object so the pins become available.

The `gpio_data` parameter must be pre-set with the correct value all I/O pins, because it is not assumed that the current output values can be read back.

I2C Bus Speed

The default clock speed of I2C busses in CircuitPython is 100kHz. In practice, using a 400kHz bus for display initialization works, even if some device on the bus only supports 100kHz I2C, because a 100kHz device will not hear its own address on the bus; it will simply stay idle. Doing this can speed display initialization. However, this is not guaranteed by the I2C specification, so if you encounter trouble, try an I2C bus at the regular 100kHz speed instead.

Here is the initialization code for the 480x480 square display on the Espressif LCD EVK, which uses 400kHz for the I2C Bus Speed:

```

init_sequence = bytes((
    0xf0, 5, 0x55, 0xaa, 0x52, 0x08, 0x00,
    0xf6, 2, 0x5a, 0x87,
    0xc1, 1, 0x3f,
    0xc2, 1, 0x0e,
    0xc6, 1, 0xf8,
    0xc9, 1, 0x10,
    0xcd, 1, 0x25,
    0xf8, 1, 0x8a,
    0xac, 1, 0x45,
    0xa0, 1, 0xdd,
    0xa7, 1, 0x47,
    0xfa, 4, 0x00, 0x00, 0x00, 0x04,
    0x86, 4, 0x99, 0xa3, 0xa3, 0x51,
    0xa3, 1, 0xee,
    0xfd, 3, 0x3c, 0x3c, 0x00,
    0x71, 1, 0x48,
    0x72, 1, 0x48,
    0x73, 2, 0x00, 0x44,
    0x97, 1, 0xee,
    0x83, 1, 0x93,
    0x9a, 1, 0x72,
    0x9b, 1, 0x5a,
    0x82, 2, 0x2c, 0x2c,
    0xb1, 1, 0x10,
    0x6d, 32, 0x00, 0x1f, 0x19, 0x1a, 0x10, 0x0e, 0x0c, 0x0a, 0x02, 0x07, 0x1e,
    0x1e, 0x1e, 0x1e, 0x1e, 0x1e, 0x1e, 0x1e, 0x1e, 0x1e, 0x1e, 0x1e, 0x08, 0x01, 0x09,
    0x0b, 0x0d, 0x0f, 0x1a, 0x19, 0x1f, 0x00,
    0x64, 16, 0x38, 0x05, 0x01, 0xdb, 0x03, 0x03, 0x38, 0x04, 0x01, 0xdc, 0x03,
    0x03, 0x7a, 0x7a, 0x7a, 0x7a,
    0x65, 16, 0x38, 0x03, 0x01, 0xdd, 0x03, 0x03, 0x38, 0x02, 0x01, 0xde, 0x03,
    0x03, 0x7a, 0x7a, 0x7a, 0x7a,
    0x66, 16, 0x38, 0x01, 0x01, 0xdf, 0x03, 0x03, 0x38, 0x00, 0x01, 0xe0, 0x03,
    0x03, 0x7a, 0x7a, 0x7a, 0x7a,
    0x67, 16, 0x30, 0x01, 0x01, 0xe1, 0x03, 0x03, 0x30, 0x02, 0x01, 0xe2, 0x03,
    0x03, 0x7a, 0x7a, 0x7a, 0x7a,
    0x68, 13, 0x00, 0x08, 0x15, 0x08, 0x15, 0x7a, 0x7a, 0x08, 0x15, 0x08, 0x15,
    0x7a, 0x7a,
    0x60, 8, 0x38, 0x08, 0x7a, 0x7a, 0x38, 0x09, 0x7a, 0x7a,
    0x63, 8, 0x31, 0xe4, 0x7a, 0x7a, 0x31, 0xe5, 0x7a, 0x7a,
    0x69, 7, 0x04, 0x22, 0x14, 0x22, 0x14, 0x22, 0x08,
    0x6b, 1, 0x07,
    0x7a, 2, 0x08, 0x13,
    0x7b, 2, 0x08, 0x13,
    0xd1, 52, 0x00, 0x00, 0x00, 0x04, 0x00, 0x12, 0x00, 0x18, 0x00, 0x21, 0x00,
    0x2a, 0x00, 0x35, 0x00, 0x47, 0x00, 0x56, 0x00, 0x90, 0x00, 0xe5, 0x01, 0x68, 0x01,
    0xd5, 0x01, 0xd7, 0x02, 0x36, 0x02, 0xa6, 0x02, 0xee, 0x03, 0x48, 0x03, 0xa0, 0x03,
    0xba, 0x03, 0xc5, 0x03, 0xd0, 0x03, 0xe0, 0x03, 0xea, 0x03, 0xfa, 0x03, 0xff,
    0xd2, 52, 0x00, 0x00, 0x00, 0x04, 0x00, 0x12, 0x00, 0x18, 0x00, 0x21, 0x00,
    0x2a, 0x00, 0x35, 0x00, 0x47, 0x00, 0x56, 0x00, 0x90, 0x00, 0xe5, 0x01, 0x68, 0x01,
    0xd5, 0x01, 0xd7, 0x02, 0x36, 0x02, 0xa6, 0x02, 0xee, 0x03, 0x48, 0x03, 0xa0, 0x03,
    0xba, 0x03, 0xc5, 0x03, 0xd0, 0x03, 0xe0, 0x03, 0xea, 0x03, 0xfa, 0x03, 0xff,
    0xd3, 52, 0x00, 0x00, 0x00, 0x04, 0x00, 0x12, 0x00, 0x18, 0x00, 0x21, 0x00,
    0x2a, 0x00, 0x35, 0x00, 0x47, 0x00, 0x56, 0x00, 0x90, 0x00, 0xe5, 0x01, 0x68, 0x01,
    0xd5, 0x01, 0xd7, 0x02, 0x36, 0x02, 0xa6, 0x02, 0xee, 0x03, 0x48, 0x03, 0xa0, 0x03,
    0xba, 0x03, 0xc5, 0x03, 0xd0, 0x03, 0xe0, 0x03, 0xea, 0x03, 0xfa, 0x03, 0xff,
    0xd4, 52, 0x00, 0x00, 0x00, 0x04, 0x00, 0x12, 0x00, 0x18, 0x00, 0x21, 0x00,
    0x2a, 0x00, 0x35, 0x00, 0x47, 0x00, 0x56, 0x00, 0x90, 0x00, 0xe5, 0x01, 0x68, 0x01,
    0xd5, 0x01, 0xd7, 0x02, 0x36, 0x02, 0xa6, 0x02, 0xee, 0x03, 0x48, 0x03, 0xa0, 0x03,
    0xba, 0x03, 0xc5, 0x03, 0xd0, 0x03, 0xe0, 0x03, 0xea, 0x03, 0xfa, 0x03, 0xff,
    0xd5, 52, 0x00, 0x00, 0x00, 0x04, 0x00, 0x12, 0x00, 0x18, 0x00, 0x21, 0x00,
    0x2a, 0x00, 0x35, 0x00, 0x47, 0x00, 0x56, 0x00, 0x90, 0x00, 0xe5, 0x01, 0x68, 0x01,
    0xd5, 0x01, 0xd7, 0x02, 0x36, 0x02, 0xa6, 0x02, 0xee, 0x03, 0x48, 0x03, 0xa0, 0x03,
    0xba, 0x03, 0xc5, 0x03, 0xd0, 0x03, 0xe0, 0x03, 0xea, 0x03, 0xfa, 0x03, 0xff,
    0xd6, 52, 0x00, 0x00, 0x00, 0x04, 0x00, 0x12, 0x00, 0x18, 0x00, 0x21, 0x00,
    0x2a, 0x00, 0x35, 0x00, 0x47, 0x00, 0x56, 0x00, 0x90, 0x00, 0xe5, 0x01, 0x68, 0x01,
    0xd5, 0x01, 0xd7, 0x02, 0x36, 0x02, 0xa6, 0x02, 0xee, 0x03, 0x48, 0x03, 0xa0, 0x03,
    0xba, 0x03, 0xc5, 0x03, 0xd0, 0x03, 0xe0, 0x03, 0xea, 0x03, 0xfa, 0x03, 0xff,

```

```

    0x3a, 1, 0x66,
    0x3a, 1, 0x66,
    0x11, 0x80, 120,
    0x29, 0x80, 20
))

expander_addr = 32
bus = busio.I2C(microcontroller.pin.GPI018, microcontroller.pin.GPI08,
frequency=400_000)

if not bus.try_lock():
    raise RuntimeError("Bus already locked")
# Set direction register
bus.writeto(expander_addr, b"\3\xf1")
# Set pull ups
bus.writeto(expander_addr, b"\2\0")
bus.unlock()

t0 = time.monotonic()
ioexpander_send_init_sequence(
    bus=bus,
    i2c_address=expander_addr,
    gpio_address=1,
    gpio_data_len=1,
    gpio_data=0xf1,
    cs_bit=1,
    mosi_bit=3,
    clk_bit=2,
    init_sequence=init_sequence)
t1 = time.monotonic()
print(t1-t0, "s to send init code")

```

Boards that have a built in display can perform these steps in the board init function such as the [Espressif LCD EV board \(\)](#).

Constructing the framebuffer and the display

Because most of the heavy lifting is done by setting up the init codes, constructing the framebuffer and display only requires a couple of lines of code:

```

fb = DotClockFramebuffer(**TFT_PINS, **TFT_TIMINGS)
disp = FramebufferDisplay(fb, auto_refresh=True)

```

Dot clocks

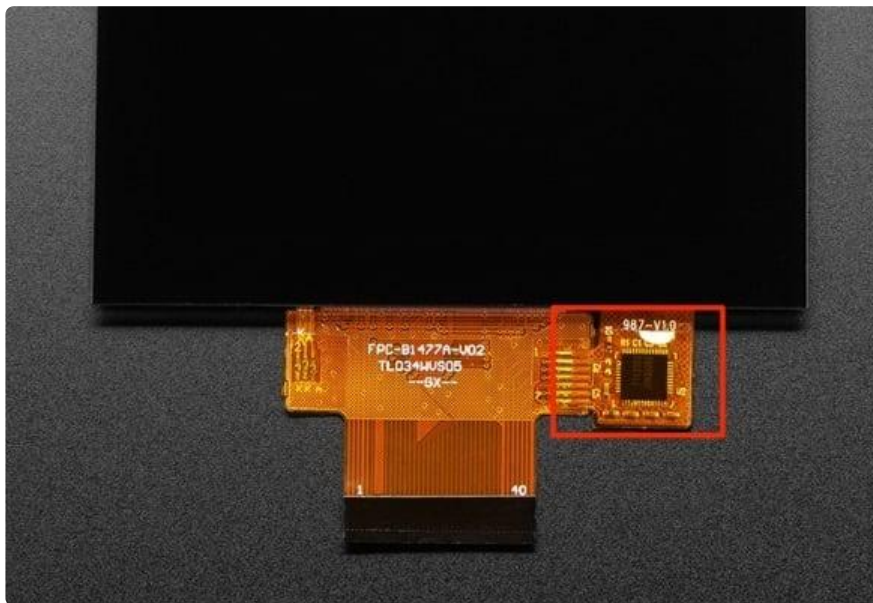
The higher the dot clock frequency, the more susceptible the display is to distortions while doing PSRAM-intensive activities. This looks like portions of the screen shifting horizontally for a frame, then returning to the normal position.

With IDF 5.1, frequencies up to 16MHz mostly work OK.

For most displays, the user can select a lower clock (down to some display-dependent minimum). This decreases refresh rate but reduces the chance of distortion.

CircuitPython Touch Display Usage

If you have a display with touch, you can use the [Adafruit_CircuitPython_FocalTouch](#) library to read the touch data. The FocalTouch capacitive touch controller is communicated to by I2C. If you're not sure if you have a touch display, just check if your display includes a square IC connected off to the side of the main ribbon cable.



Determining the I2C Address

You can scan for I2C devices by connecting to the REPL and typing the following:

```
import board

i2c = board.I2C()
while i2c.try_lock():
    pass

i2c.scan()
```

You should see a couple of devices listed. These will be the GPIO expander and the touch controller. The GPIO Expander is at **0x3F** (or 63 in decimal) by default, though it's possible to change the address with the solderable jumpers on the reverse side. The other address should be the touch controller. On the TL040HDS20 4.0" square display, it shows up as **0x48** (or 72 in decimal), but it's possible it may be a different value on other displays.

Initializing the Touch Controller

In order to use the controller, it will need to first be initialized. You can use the following code to initialize it. If your I2C address differs, change it to the appropriate value.

```
import board
import busio
import adafruit_focaltouch

i2c = busio.I2C(board.SCL, board.SDA, frequency=100_000)
ft = adafruit_focaltouch.Adafruit_FocalTouch(i2c, address=0x48)
```

Likely you will have already initialized I2C for using the GPIO expander, so you can just add the `adafruit_focaltouch` import line and further down add the initialization line like this:

```
import adafruit_focaltouch

...

ft = adafruit_focaltouch.Adafruit_FocalTouch(i2c, address=0x48)
```

Reading from the Touch Controller

To read from the controller, check if it has been `touched` in the main loop and if so, read the touches. Although this controller can support multiple touches, it seems to sometimes have difficulty distinguishing between 2 or more touch points. For each touch point that is reported, you can then read the `x` and `y` coordinates.

```
if ft.touched:
    for touch in ft.touches:
        x = touch["x"]
        y = touch["y"]
```

Example

Here is a paint demo that works on the TL040HDS20 4.0" Square display. Just click the Download Project button, unzip it, and copy it over to your CIRCUITPY drive.

```
# SPDX-FileCopyrightText: 2021 ladyada for Adafruit Industries
# SPDX-License-Identifier: MIT
"""
Simple painting demo that draws on the Adafruit Qualia ESP32-S3 RGB666
with the 4.0" square display and FT6206 captouch driver
"""

import displayio
```



```

import busio
import board
import dotclockframebuffer
from framebufferio import FramebufferDisplay
import adafruit_focaltouch

displayio.release_displays()

# Initialize the Display
tft_pins = dict(board.TFT_PINS)

tft_timings = {
    "frequency": 16000000,
    "width": 720,
    "height": 720,
    "hsync_pulse_width": 2,
    "hsync_front_porch": 46,
    "hsync_back_porch": 44,
    "vsync_pulse_width": 2,
    "vsync_front_porch": 16,
    "vsync_back_porch": 18,
    "hsync_idle_low": False,
    "vsync_idle_low": False,
    "de_idle_high": False,
    "pclk_active_high": False,
    "pclk_idle_high": False,
}

init_sequence_tl040hds20 = bytes()

board.I2C().deinit()
i2c = busio.I2C(board.SCL, board.SDA, frequency=100_000)
tft_io_expander = dict(board.TFT_IO_EXPANDER)
# tft_io_expander['i2c_address'] = 0x38 # uncomment for rev B
dotclockframebuffer.ioexpander_send_init_sequence(
    i2c, init_sequence_tl040hds20, **tft_io_expander
)

fb = dotclockframebuffer.DotClockFramebuffer(**tft_pins, **tft_timings)
display = FramebufferDisplay(fb, auto_refresh=False)

# Main Program
pixel_size = 6
palette_width = 160
palette_height = display.height // 8

bitmap = displayio.Bitmap(display.width, display.height, 65535)

# Create a TileGrid to hold the bitmap
tile_grid = displayio.TileGrid(
    bitmap,

pixel_shader=displayio.ColorConverter(input_colorspace=displayio.Colorspace.RGB565),
)

# Create a Group to hold the TileGrid
group = displayio.Group()

# Add the TileGrid to the Group
group.append(tile_grid)

# Add the Group to the Display
display.root_group = group

display.auto_refresh = True

ft = adafruit_focaltouch.Adafruit_FocalTouch(i2c, address=0x48)

current_color = displayio.ColorConverter().convert(0xFFFFFFFF)

```

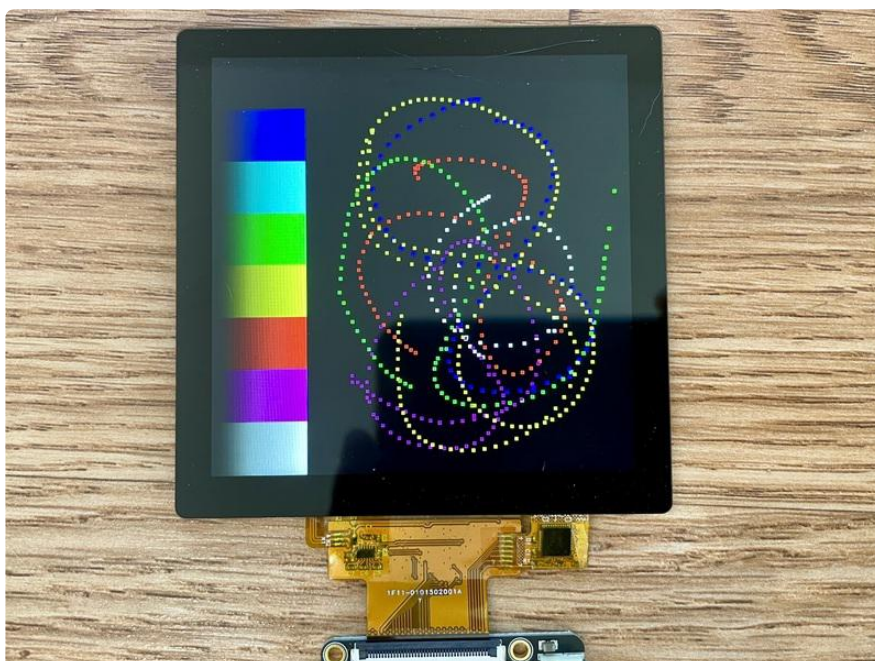
```

for i in range(palette_width):
    color_index = i * 255 // palette_width
    rgb565 = displayio.ColorConverter().convert(
        color_index | color_index << 8 | color_index << 16
    )
    r_mask = 0xF800
    g_mask = 0x07E0
    b_mask = 0x001F
    for j in range(palette_height):
        bitmap[i, j + palette_height] = rgb565 & b_mask
        bitmap[i, j + palette_height * 2] = rgb565 & (b_mask | g_mask)
        bitmap[i, j + palette_height * 3] = rgb565 & g_mask
        bitmap[i, j + palette_height * 4] = rgb565 & (r_mask | g_mask)
        bitmap[i, j + palette_height * 5] = rgb565 & r_mask
        bitmap[i, j + palette_height * 6] = rgb565 & (r_mask | b_mask)
        bitmap[i, j + palette_height * 7] = rgb565

while True:
    if ft.touched:
        try:
            for touch in ft.touches:
                x = touch["x"]
                y = touch["y"]
                if x < palette_width:
                    current_color = bitmap[x, y]
                else:
                    for i in range(pixel_size):
                        for j in range(pixel_size):
                            x_pixel = x - (pixel_size // 2) + i
                            y_pixel = y - (pixel_size // 2) + j
                            if (
                                0 <= x_pixel < display.width
                                and 0 <= y_pixel < display.height
                            ):
                                bitmap[x_pixel, y_pixel] = current_color
        except RuntimeError:
            pass

```

To use, just use your finger to paint on the canvas. You can also select a color from the left. The closer to the edge of the display, the darker, the color will be.



Qualia S3 RGB-666 with TL021WVC02 2.1" 480x480 Round Display

If you have issues running the example, you can always test your hardware by running a UF2 for your display from <https://learn.adafruit.com/adafruit-qualia-esp32-s3-for-rgb666-displays/arduino-rainbow-demo>

Initialization Codes

Here are the init codes for this display:

```
init_sequence_tl021wvc02 = bytes((
    0xff, 0x05, 0x77, 0x01, 0x00, 0x00, 0x10,
    0xc0, 0x02, 0x3b, 0x00,
    0xc1, 0x02, 0x0b, 0x02,
    0xc2, 0x02, 0x00, 0x02,
    0xcc, 0x01, 0x10,
    0xcd, 0x01, 0x08,
    0xb0, 0x10, 0x02, 0x13, 0x1b, 0x0d, 0x10, 0x05, 0x08, 0x07, 0x07, 0x24, 0x04,
    0x11, 0x0e, 0x2c, 0x33, 0x1d,
    0xb1, 0x10, 0x05, 0x13, 0x1b, 0x0d, 0x11, 0x05, 0x08, 0x07, 0x07, 0x24, 0x04,
    0x11, 0x0e, 0x2c, 0x33, 0x1d,
    0xff, 0x05, 0x77, 0x01, 0x00, 0x00, 0x11,
    0xb0, 0x01, 0x5d,
    0xb1, 0x01, 0x43,
    0xb2, 0x01, 0x81,
    0xb3, 0x01, 0x80,
    0xb5, 0x01, 0x43,
    0xb7, 0x01, 0x85,
    0xb8, 0x01, 0x20,
    0xc1, 0x01, 0x78,
    0xc2, 0x01, 0x78,
    0xd0, 0x01, 0x88,
    0xe0, 0x03, 0x00, 0x00, 0x02,
    0xe1, 0x0b, 0x03, 0xa0, 0x00, 0x00, 0x04, 0xa0, 0x00, 0x00, 0x00, 0x20, 0x20,
    0xe2, 0x0d, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00,
    0xe3, 0x04, 0x00, 0x00, 0x11, 0x00,
    0xe4, 0x02, 0x22, 0x00,
    0xe5, 0x10, 0x05, 0xec, 0xa0, 0xa0, 0x07, 0xee, 0xa0, 0xa0, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00,
    0xe6, 0x04, 0x00, 0x00, 0x11, 0x00,
    0xe7, 0x02, 0x22, 0x00,
    0xe8, 0x10, 0x06, 0xed, 0xa0, 0xa0, 0x08, 0xef, 0xa0, 0xa0, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00,
    0xeb, 0x07, 0x00, 0x00, 0x40, 0x40, 0x00, 0x00, 0x00,
    0xed, 0x10, 0xff, 0xff, 0xff, 0xff, 0xba, 0x0a, 0xbf, 0x45, 0xff, 0xff, 0x54, 0xfb,
    0xa0, 0xab, 0xff, 0xff, 0xff,
    0xef, 0x06, 0x10, 0x0d, 0x04, 0x08, 0x3f, 0x1f,
    0xff, 0x05, 0x77, 0x01, 0x00, 0x00, 0x13,
    0xef, 0x01, 0x08,
    0xff, 0x05, 0x77, 0x01, 0x00, 0x00, 0x00,
    0x36, 0x01, 0x00,
    0x3a, 0x01, 0x60,
    0x11, 0x80, 0x64,
    0x29, 0x80, 0x32,
))
```

Timings

Here are the timing settings for this display:

```
tft_timings = {
    "frequency": 16_000_000,
    "width": 480,
    "height": 480,
    "hsync_pulse_width": 20,
    "hsync_front_porch": 40,
    "hsync_back_porch": 40,
    "vsync_pulse_width": 10,
    "vsync_front_porch": 40,
    "vsync_back_porch": 40,
    "hsync_idle_low": False,
    "vsync_idle_low": False,
    "de_idle_high": False,
    "pclk_active_high": True,
    "pclk_idle_high": False,
}
```

Example

Here's an example using those settings:

```
from displayio import release_displays
release_displays()

import time
import busio
import board
import dotclockframebuffer
from framebufferio import FramebufferDisplay

init_sequence_tl021wvc02 = bytes((
    0xff, 0x05, 0x77, 0x01, 0x00, 0x00, 0x10,
    0xc0, 0x02, 0x3b, 0x00,
    0xc1, 0x02, 0x0b, 0x02,
    0xc2, 0x02, 0x00, 0x02,
    0xcc, 0x01, 0x10,
    0xcd, 0x01, 0x08,
    0xb0, 0x10, 0x02, 0x13, 0x1b, 0x0d, 0x10, 0x05, 0x08, 0x07, 0x07, 0x24, 0x04,
    0x11, 0x0e, 0x2c, 0x33, 0x1d,
    0xb1, 0x10, 0x05, 0x13, 0x1b, 0x0d, 0x11, 0x05, 0x08, 0x07, 0x07, 0x24, 0x04,
    0x11, 0x0e, 0x2c, 0x33, 0x1d,
    0xff, 0x05, 0x77, 0x01, 0x00, 0x00, 0x11,
    0xb0, 0x01, 0x5d,
    0xb1, 0x01, 0x43,
    0xb2, 0x01, 0x81,
    0xb3, 0x01, 0x80,
    0xb5, 0x01, 0x43,
    0xb7, 0x01, 0x85,
    0xb8, 0x01, 0x20,
    0xc1, 0x01, 0x78,
    0xc2, 0x01, 0x78,
    0xd0, 0x01, 0x88,
    0xe0, 0x03, 0x00, 0x00, 0x02,
    0xe1, 0x0b, 0x03, 0xa0, 0x00, 0x00, 0x04, 0xa0, 0x00, 0x00, 0x00, 0x20, 0x20,
    0xe2, 0x0d, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00,
```

```

    0xe3, 0x04, 0x00, 0x00, 0x11, 0x00,
    0xe4, 0x02, 0x22, 0x00,
    0xe5, 0x10, 0x05, 0xec, 0xa0, 0xa0, 0x07, 0xee, 0xa0, 0xa0, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00,
    0xe6, 0x04, 0x00, 0x00, 0x11, 0x00,
    0xe7, 0x02, 0x22, 0x00,
    0xe8, 0x10, 0x06, 0xed, 0xa0, 0xa0, 0x08, 0xef, 0xa0, 0xa0, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00,
    0xeb, 0x07, 0x00, 0x00, 0x40, 0x40, 0x00, 0x00, 0x00,
    0xed, 0x10, 0xff, 0xff, 0xff, 0xff, 0xba, 0x0a, 0xbf, 0x45, 0xff, 0xff, 0x54, 0xfb,
0xa0, 0xab, 0xff, 0xff, 0xff,
    0xef, 0x06, 0x10, 0x0d, 0x04, 0x08, 0x3f, 0x1f,
    0xff, 0x05, 0x77, 0x01, 0x00, 0x00, 0x13,
    0xef, 0x01, 0x08,
    0xff, 0x05, 0x77, 0x01, 0x00, 0x00, 0x00,
    0x36, 0x01, 0x00,
    0x3a, 0x01, 0x60,
    0x11, 0x80, 0x64,
    0x29, 0x80, 0x32,
))

```

```

board.I2C().deinit()
i2c = busio.I2C(board.SCL, board.SDA) #, frequency=400_000)
tft_io_expander = dict(board.TFT_IO_EXPANDER)
#tft_io_expander['i2c_address'] = 0x38 # uncomment for rev B
dotclockframebuffer.ioexpander_send_init_sequence(i2c, init_sequence_tl021wvc02,
**tft_io_expander)
i2c.deinit()

tft_pins = dict(board.TFT_PINS)

tft_timings = {
    "frequency": 16_000_000,
    "width": 480,
    "height": 480,
    "hsync_pulse_width": 20,
    "hsync_front_porch": 40,
    "hsync_back_porch": 40,
    "vsync_pulse_width": 10,
    "vsync_front_porch": 40,
    "vsync_back_porch": 40,
    "hsync_idle_low": False,
    "vsync_idle_low": False,
    "de_idle_high": False,
    "pclk_active_high": True,
    "pclk_idle_high": False,
}

fb = dotclockframebuffer.DotClockFramebuffer(**tft_pins, **tft_timings)
disp = FramebufferDisplay(fb, auto_refresh=False)

while True:
    for info in (tft_pins, tft_timings):
        print("\n" * 24)
        for k, v in info.items():
            print(f"{k:<20} {v}")
        disp.auto_refresh = True
        time.sleep(6)
        disp.auto_refresh = False

```

Go ahead and save the example you your CircuitPython code.py and run the code. Your display should now look like this:



Qualia S3 RGB-666 with TL034WVS05 3.4" 480x480 Square Display

If you have issues running the example, you can always test your hardware by running a UF2 for your display from <https://learn.adafruit.com/adafruit-qualia-esp32-s3-for-rgb666-displays/arduino-rainbow-demo>

Initialization Codes

Here are the init codes for this display:

```
init_sequence_tl034wvs05 = bytes((
    b'\xff\x05w\x01\x00\x00\x13'
    b'\xef\x01\x08'
    b'\xff\x05w\x01\x00\x00\x10'
    b'\xc0\x02;\x00'
    b'\xc1\x02\x12\n'
    b'\xc2\x02\x07\x03'
    b'\xc3\x01\x02'
    b'\xcc\x01\x10'
    b'\xcd\x01\x08'
    b'\xb0\x10\x0f\x11\x17\x15\x15\t\x0c\x08\x08&\x04Y\x16f-\x1f'
    b'\xb1\x10\x0f\x11\x17\x15\x15\t\x0c\x08\x08&\x04Y\x16f-\x1f'
    b'\xff\x05w\x01\x00\x00\x11'
    b'\xb0\x01m'
    b'\xb1\x01:'
    b'\xb2\x01\x01'
    b'\xb3\x01\x80'
    b'\xb5\x01I'
    b'\xb7\x01\x85'
    b'\xb8\x01'
    b'\xc1\x01x'
    b'\xc2\x01x'
```

```

b'\xd0\x01\x88'
b'\xe0\x03\x00\x00\x02'
b'\xe1\x0b\x07\x00\t\x00\x06\x00\x08\x00\x0033'
b'\xe2\r\x11\x1133\xf6\x00\xf6\x00\xf6\x00\xf6\x00'
b'\xe3\x04\x00\x00\x11\x11'
b'\xe4\x02DD'
b'\xe5\x10\x0f\xf3=\xff\x11\xf5=\xff\x0b\xef=\xff\r\xf1=\xff'
b'\xe6\x04\x00\x00\x11\x11'
b'\xe7\x02DD'
b'\xe8\x10\x0e\xf2=\xff\x10\xf4=\xff\n\xee=\xff\x0c\xf0=\xff'
b'\xe9\x026\x00'
b'\xeb\x07\x00\x01\xe4\xe4D\xaa\x10'
b'\xec\x02&lt;\x00'
b'\xed\x10\xffEg\xfa\x01+\xcf\xff\xff\xfc\xb2\x10\xafvT\xff'
b'\xef\x06\x10\r\x04\x08?\xf1f'
b'\xff\x05w\x01\x00\x00\x00'
b'5\x01\x00'
b':\x01f'
b'\x11\x80x'
b')\x802'
))

```

Timings

Here are the timing settings for this display:

```

tft_timings = {
    "frequency": 16000000,
    "width": 480,
    "height": 480,
    "hsync_pulse_width": 20,
    "hsync_front_porch": 40,
    "hsync_back_porch": 40,
    "vsync_pulse_width": 10,
    "vsync_front_porch": 40,
    "vsync_back_porch": 40,
    "hsync_idle_low": False,
    "vsync_idle_low": False,
    "de_idle_high": False,
    "pclk_active_high": True,
    "pclk_idle_high": False,
}

```

Example

Here's an example using those settings:

```

from displayio import release_displays
release_displays()

import displayio
import busio
import board
import dotclockframebuffer
from framebufferio import FramebufferDisplay

tft_pins = dict(board.TFT_PINS)

tft_timings = {

```

```

    "frequency": 16000000,
    "width": 480,
    "height": 480,
    "hsync_pulse_width": 20,
    "hsync_front_porch": 40,
    "hsync_back_porch": 40,
    "vsync_pulse_width": 10,
    "vsync_front_porch": 40,
    "vsync_back_porch": 40,
    "hsync_idle_low": False,
    "vsync_idle_low": False,
    "de_idle_high": False,
    "pclk_active_high": True,
    "pclk_idle_high": False,
}

init_sequence_tl034wvs05 = bytes((
    b'\xff\x05w\x01\x00\x00\x13'
    b'\xef\x01\x08'
    b'\xff\x05w\x01\x00\x00\x10'
    b'\xc0\x02;\x00'
    b'\xc1\x02\x12\n'
    b'\xc2\x02\x07\x03'
    b'\xc3\x01\x02'
    b'\xcc\x01\x10'
    b'\xcd\x01\x08'
    b'\xb0\x10\x0f\x11\x17\x15\x15\t\x0c\x08\x08&\x04Y\x16f-\x1f'
    b'\xb1\x10\x0f\x11\x17\x15\x15\t\x0c\x08\x08&\x04Y\x16f-\x1f'
    b'\xff\x05w\x01\x00\x00\x11'
    b'\xb0\x01m'
    b'\xb1\x01:'
    b'\xb2\x01\x01'
    b'\xb3\x01\x80'
    b'\xb5\x01I'
    b'\xb7\x01\x85'
    b'\xb8\x01 '
    b'\xc1\x01x'
    b'\xc2\x01x'
    b'\xd0\x01\x88'
    b'\xe0\x03\x00\x00\x02'
    b'\xe1\x0b\x07\x00\t\x00\x06\x00\x08\x00\x0033'
    b'\xe2\r\x11\x1133\xf6\x00\xf6\x00\xf6\x00\xf6\x00\x00'
    b'\xe3\x04\x00\x00\x11\x11'
    b'\xe4\x02DD'
    b'\xe5\x10\x0f\xf3=\xff\x11\xf5=\xff\x0b\xef=\xff\r\xf1=\xff'
    b'\xe6\x04\x00\x00\x11\x11'
    b'\xe7\x02DD'
    b'\xe8\x10\x0e\xf2=\xff\x10\xf4=\xff\n\xee=\xff\x0c\xf0=\xff'
    b'\xe9\x026\x00'
    b'\xeb\x07\x00\x01\xe4\xe4\xaa\x10'
    b'\xec\x02<\x00'
    b'\xed\x10\xffEg\xfa\x01+\xcf\xff\xff\xfc\xb2\x10\xafvT\xff'
    b'\xef\x06\x10\r\x04\x08?\x1f'
    b'\xff\x05w\x01\x00\x00\x00'
    b'5\x01\x00'
    b':\x01f'
    b'\x11\x80x'
    b')\x802'
))

board.I2C().deinit()
i2c = busio.I2C(board.SCL, board.SDA)
tft_io_expander = dict(board.TFT_IO_EXPANDER)
#tft_io_expander['i2c_address'] = 0x38 # uncomment for rev B
dotclockframebuffer.ioexpander_send_init_sequence(i2c, init_sequence_tl034wvs05,
**tft_io_expander)
i2c.deinit()

bitmap = displayio.OnDiskBitmap("/display-ruler-720p.bmp")

```



```

fb = dotclockframebuffer.DotClockFramebuffer(**tft_pins, **tft_timings)
display = FramebufferDisplay(fb, auto_refresh=False)

# Create a TileGrid to hold the bitmap
tile_grid = displayio.TileGrid(bitmap, pixel_shader=bitmap.pixel_shader)

# Create a Group to hold the TileGrid
group = displayio.Group()

# Add the TileGrid to the Group
group.append(tile_grid)

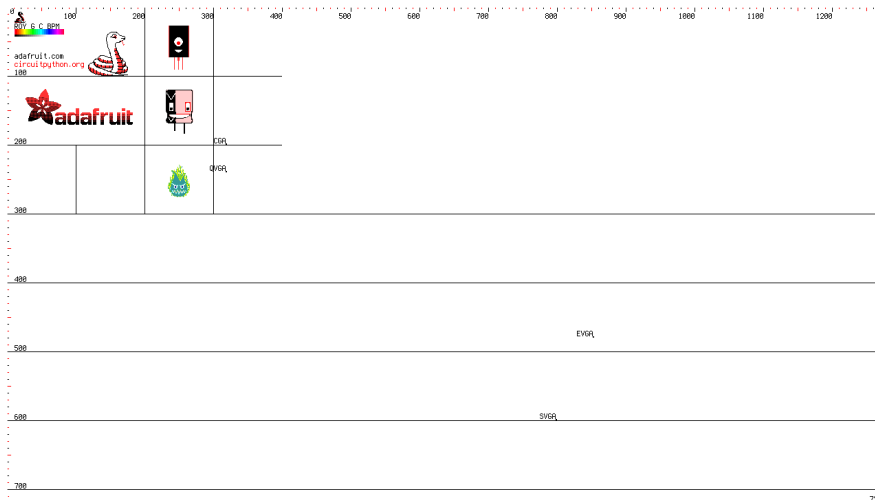
# Add the Group to the Display
display.root_group = group

display.auto_refresh = True

# Loop forever so you can enjoy your image
while True:
    pass

```

Download the following image into the root folder of of your CIRCUITPY drive:



Go ahead and save the example you your CircuitPython code.py and run the code. Your display should now look like this:



Qualia S3 RGB-666 with TL040HDS20 4.0" 720x720 Square Display

If you have issues running the example, you can always test your hardware by running a UF2 for your display from <https://learn.adafruit.com/adafruit-qualia-esp32-s3-for-rgb666-displays/arduino-rainbow-demo>

Initialization Codes

Here are the init codes for this display:

This display is the easiest display and needs no initialization.

```
init_sequence_tl040hds20 = bytes()
```

Timings

Here are the timing settings for this display:

```
tft_timings = {  
    "frequency": 16000000,  
    "width": 720,  
    "height": 720,  
    "hsync_pulse_width": 2,  
    "hsync_front_porch": 46,  
    "hsync_back_porch": 44,  
}
```

```

    "vsync_pulse_width": 2,
    "vsync_front_porch": 16,
    "vsync_back_porch": 18,
    "hsync_idle_low": False,
    "vsync_idle_low": False,
    "de_idle_high": False,
    "pclk_active_high": False,
    "pclk_idle_high": False,
}

```

Example

Here's an example using those settings:

```

from displayio import release_displays
release_displays()

import displayio
import busio
import board
import dotclockframebuffer
from framebufferio import FramebufferDisplay
from microcontroller import pin

tft_pins = dict(board.TFT_PINS)

tft_timings = {
    "frequency": 16000000,
    "width": 720,
    "height": 720,
    "hsync_pulse_width": 2,
    "hsync_front_porch": 46,
    "hsync_back_porch": 44,
    "vsync_pulse_width": 2,
    "vsync_front_porch": 16,
    "vsync_back_porch": 18,
    "hsync_idle_low": False,
    "vsync_idle_low": False,
    "de_idle_high": False,
    "pclk_active_high": False,
    "pclk_idle_high": False,
}

init_sequence_tl040hds20 = bytes()

board.I2C().deinit()
i2c = busio.I2C(board.SCL, board.SDA)
tft_io_expander = dict(board.TFT_IO_EXPANDER)
#tft_io_expander['i2c_address'] = 0x38 # uncomment for rev B
dotclockframebuffer.ioexpander_send_init_sequence(i2c, init_sequence_tl040hds20,
**tft_io_expander)
i2c.deinit()

bitmap = displayio.OnDiskBitmap("/display-ruler-720p.bmp")

fb = dotclockframebuffer.DotClockFramebuffer(**tft_pins, **tft_timings)
display = FramebufferDisplay(fb, auto_refresh=False)

# Create a TileGrid to hold the bitmap
tile_grid = displayio.TileGrid(bitmap, pixel_shader=bitmap.pixel_shader)

# Create a Group to hold the TileGrid
group = displayio.Group()

```

```

# Add the TileGrid to the Group
group.append(tile_grid)

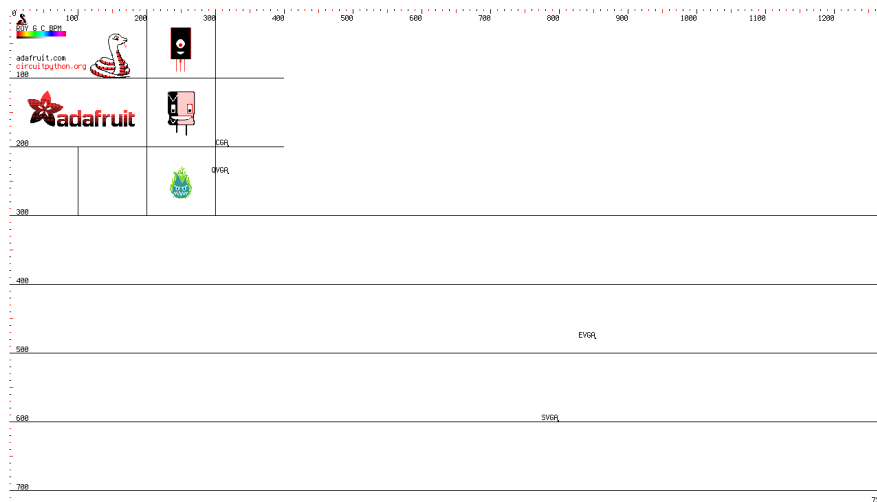
# Add the Group to the Display
display.root_group = group

display.auto_refresh = True

# Loop forever so you can enjoy your image
while True:
    pass

```

Download the following image into the root folder of of your CIRCUITPY drive:



Go ahead and save the example you your CircuitPython code.py and run the code. Your display should now look like this:



Qualia S3 RGB-666 with TL032FWV01 3.2" 320x820 Bar Display

If you have issues running the example, you can always test your hardware by running a UF2 for your display from <https://learn.adafruit.com/adafruit-qualia-esp32-s3-for-rgb666-displays/arduino-rainbow-demo>

Initialization Codes

Here are the init codes for this display:

```
init_sequence_tl032 = bytes((
    b'\x11\x80d'
    b'\xff\x05w\x01\x00\x00\x13'
    b'\xef\x01\x08'
    b'\xff\x05w\x01\x00\x00\x10'
    b'\xc0\x02\xe5\x02'
    b'\xc1\x02\x0c\n'
    b'\xc2\x02\x07\x0f'
    b'\xc3\x01\x02'
    b'\xcc\x01\x10'
    b'\xcd\x01\x08'
    b'\xb0\x10\x00\x080\r\xce\x06\x00\x08\x08\x1d\x02\xd0\x0fo6?'
    b'\xb1\x10\x00\x100\x0c\x11\x05\x00\x07\x07\x1f\x05\xd3\x11n4?'
    b'\xff\x05w\x01\x00\x00\x11'
    b'\xb0\x01M'
    b'\xb1\x01\x1c'
    b'\xb2\x01\x87'
    b'\xb3\x01\x80'
    b'\xb5\x01G'
    b'\xb7\x01\x85'
    b'\xb8\x01!'
    b'\xb9\x01\x10'
    b'\xc1\x01x'
    b'\xc2\x01x'
    b'\xd0\x81\x88d'
    b'\xe0\x03\x80\x00\x02'
    b'\xe1\x0b\x04\xa0\x00\x00\x05\xa0\x00\x00\x00` ` '
    b'\xe2\r00` &lt; \xa0\x00\x00=\xa0\x00\x00\x00'
    b'\xe3\x04\x00\x0033'
    b'\xe4\x02DD'
    b'\xe5\x10\x06&gt; \xa0\xa0\x08@\xa0\xa0\nB\xa0\xa0\x0cD\xa0\xa0'
    b'\xe6\x04\x00\x0033'
    b'\xe7\x02DD'
    b'\xe8\x10\x07?\xa0\xa0\tA\xa0\xa0\x0bC\xa0\xa0rE\xa0\xa0'
    b'\xeb\x07\x00\x01NN\xeed\x00'
    b'\xed\x10\xff\xff\x04Vr\xff\xff\xff\xff' e@\xff\xff"
    b'\xef\x06\x10\r\x04\x08?\x1f'
    b'\xff\x05w\x01\x00\x00\x13'
    b'\xe8\x02\x00\x0e'
    b'\xff\x05w\x01\x00\x00\x00'
    b'\x11\x80x'
    b'\xff\x05w\x01\x00\x00\x13'
    b'\xe8\x82\x00\x0c\n'
    b'\xe8\x02\x00\x00'
    b'\xff\x05w\x01\x00\x00\x00'
    b'6\x01\x00'
    b':\x01f'
```

```
b'\x11\x80x'  
b')\x80x'  
)
```

Timings

Here are the timing settings for this display:

```
tft_timings = {  
    "frequency": 16000000,  
    "width": 320,  
    "height": 820,  
  
    "hsync_pulse_width": 3,  
    "hsync_back_porch": 251,  
    "hsync_front_porch": 150,  
    "hsync_idle_low": False,  
  
    "vsync_pulse_width": 6,  
    "vsync_back_porch": 90,  
    "vsync_front_porch": 100,  
    "vsync_idle_low": False,  
  
    "pclk_active_high": False,  
    "pclk_idle_high": False,  
    "de_idle_high": False,  
}
```

Example

Here's an example using those settings:

```
from displayio import release_displays  
release_displays()  
  
import random  
import displayio  
import time  
import busio  
import board  
import dotclockframebuffer  
from framebufferio import FramebufferDisplay  
  
init_sequence_tl032 = bytes((  
    b'\x11\x80d'  
    b'\xff\x05w\x01\x00\x00\x13'  
    b'\xef\x01\x08'  
    b'\xff\x05w\x01\x00\x00\x10'  
    b'\xc0\x02\xe5\x02'  
    b'\xc1\x02\x0c\n'  
    b'\xc2\x02\x07\x0f'  
    b'\xc3\x01\x02'  
    b'\xcc\x01\x10'  
    b'\xcd\x01\x08'  
    b'\xb0\x10\x00\x08Q\r\xce\x06\x00\x08\x08\x1d\x02\xd0\x0fo6? '  
    b'\xb1\x10\x00\x100\x0c\x11\x05\x00\x07\x07\x1f\x05\xd3\x11n4? '  
    b'\xff\x05w\x01\x00\x00\x11'  
    b'\xb0\x01M'  
    b'\xb1\x01\x1c'
```

```

b'\xb2\x01\x87'
b'\xb3\x01\x80'
b'\xb5\x01G'
b'\xb7\x01\x85'
b'\xb8\x01!'
b'\xb9\x01\x10'
b'\xc1\x01x'
b'\xc2\x01x'
b'\xd0\x81\x88d'
b'\xe0\x03\x80\x00\x02'
b'\xe1\x0b\x04\xa0\x00\x00\x05\xa0\x00\x00\x00` ` '
b'\xe2\r00` `&lt;\xa0\x00\x00=\xa0\x00\x00\x00'
b'\xe3\x04\x00\x0033'
b'\xe4\x02DD'
b'\xe5\x10\x06&gt;\xa0\xa0\x08@\xa0\xa0\nB\xa0\xa0\x0cD\xa0\xa0'
b'\xe6\x04\x00\x0033'
b'\xe7\x02DD'
b'\xe8\x10\x07?\xa0\xa0\tA\xa0\xa0\x0bC\xa0\xa0rE\xa0\xa0'
b'\xeb\x07\x00\x01NN\xeeD\x00'
b'\xed\x10\xff\xff\x04Vr\xff\xff\xff\xff\xff'e@\xff\xff"
b'\xef\x06\x10\r\x04\x08?\x1f'
b'\xff\x05w\x01\x00\x00\x13'
b'\xe8\x02\x00\x0e'
b'\xff\x05w\x01\x00\x00\x00'
b'\x11\x80x'
b'\xff\x05w\x01\x00\x00\x13'
b'\xe8\x82\x00\x0c\n'
b'\xe8\x02\x00\x00'
b'\xff\x05w\x01\x00\x00\x00'
b'6\x01\x00'
b':\x01f'
b'\x11\x80x'
b')\x80x'
) )

board.I2C().deinit()
i2c = busio.I2C(board.SCL, board.SDA, frequency=400_000)
tft_io_expander = dict(board.TFT_IO_EXPANDER)
#tft_io_expander['i2c_address'] = 0x38 # uncomment for rev B
dotclockframebuffer.ioexpander_send_init_sequence(i2c, init_sequence_tl032,
**tft_io_expander)
i2c.deinit()

tft_pins = dict(board.TFT_PINS)

tft_timings = {
    "frequency": 16000000,
    "width": 320,
    "height": 820,

    "hsync_pulse_width": 3,
    "hsync_back_porch": 251,
    "hsync_front_porch": 150,
    "hsync_idle_low": False,

    "vsync_pulse_width": 6,
    "vsync_back_porch": 90,
    "vsync_front_porch": 100,
    "vsync_idle_low": False,

    "pclk_active_high": False,
    "pclk_idle_high": False,
    "de_idle_high": False,
}

#bitmap = displayio.OnDiskBitmap("/display-ruler-720p.bmp")

bitmap = displayio.Bitmap(256, 7*64, 65535)
fb = dotclockframebuffer.DotClockFramebuffer(**tft_pins, **tft_timings)

```

```

display = FramebufferDisplay(fb, auto_refresh=False)

# Create a TileGrid to hold the bitmap
tile_grid = displayio.TileGrid(bitmap,
pixel_shader=displayio.ColorConverter(input_colorspace=displayio.Colorspace.RGB565))

# Create a Group to hold the TileGrid
group = displayio.Group()

# Add the TileGrid to the Group
group.append(tile_grid)

# Add the Group to the Display
display.root_group = group

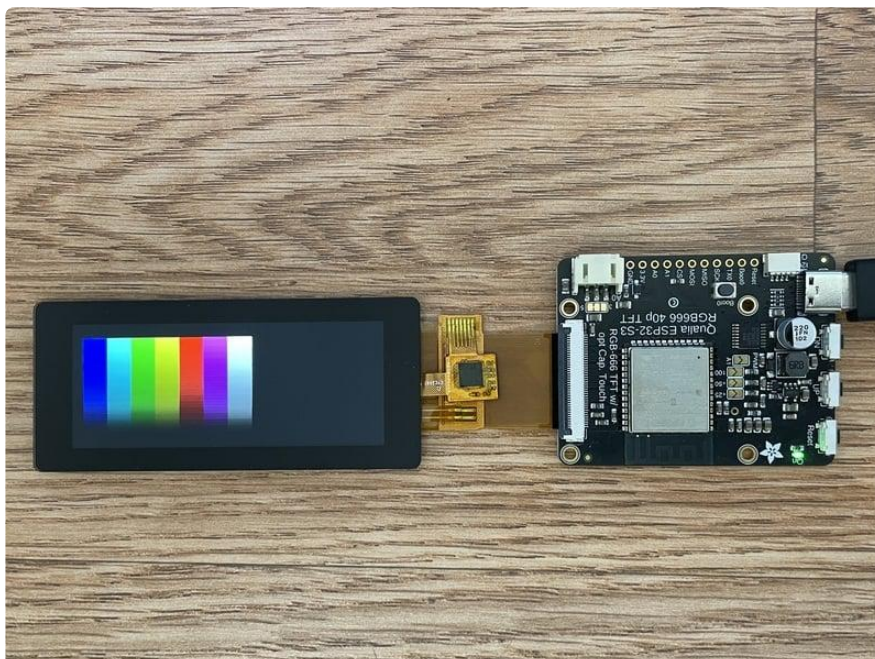
display.auto_refresh = True

for i in range(256):
    b = i >> 3
    g = (i >> 2) << 5
    r = b << 11
    for j in range(64):
        bitmap[i, j] = b
        bitmap[i, j+64] = b|g
        bitmap[i, j+128] = g
        bitmap[i, j+192] = g|r
        bitmap[i, j+256] = r
        bitmap[i, j+320] = r|b
        bitmap[i, j+384] = r|g|b

# Loop forever so you can enjoy your image
while True:
    time.sleep(1)
    display.auto_refresh = False
    group.x = random.randint(0, 32)
    group.y = random.randint(0, 32)
    display.auto_refresh = True
    pass

```

Go ahead and save the example you your CircuitPython code.py and run the code. Your display should now look like this:



Arduino IDE Setup

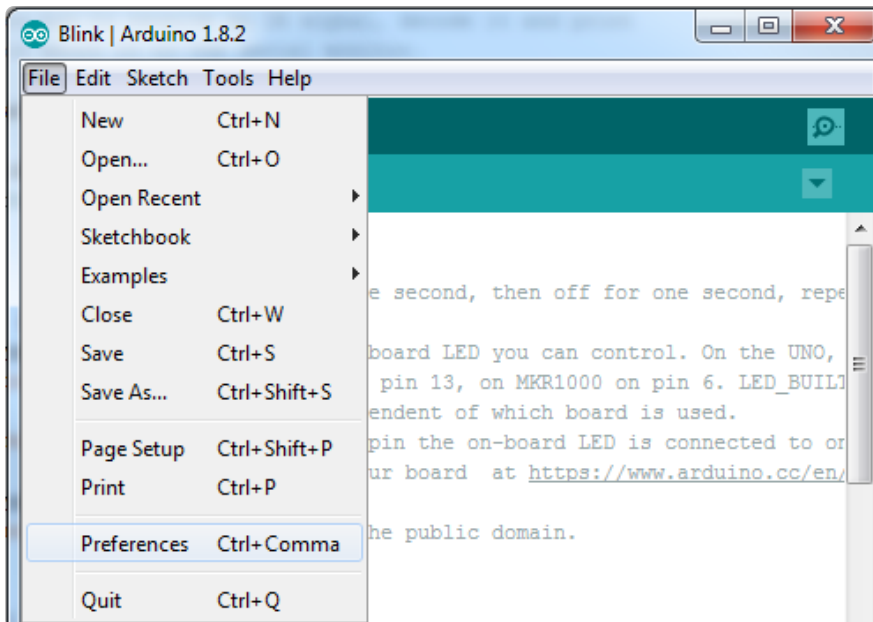
The ESP32-S2/S3 bootloader does not have USB serial support for Windows 7 or 8. (See <https://github.com/espressif/arduino-esp32/issues/5994>) please update to version 10 which is supported by espressif! Alternatively you can try this community-crafted Windows 7 driver (<https://github.com/kutukvpavel/Esp32-Win7-VCP-drivers>)

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

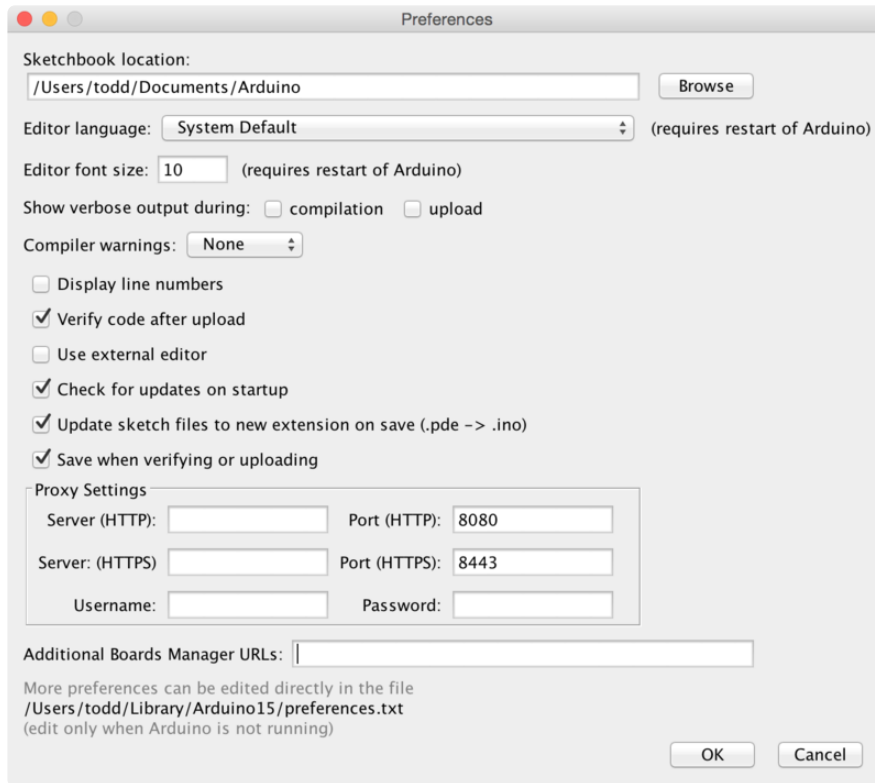
[Arduino IDE Download](#)

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

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.



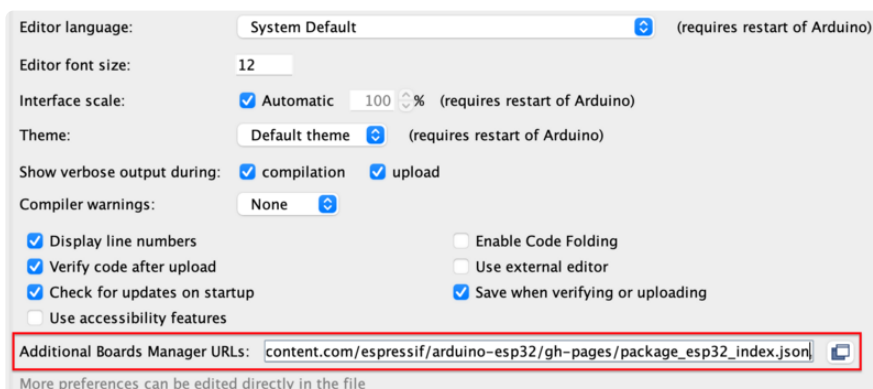
A dialog will pop up just like the one shown below.



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

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



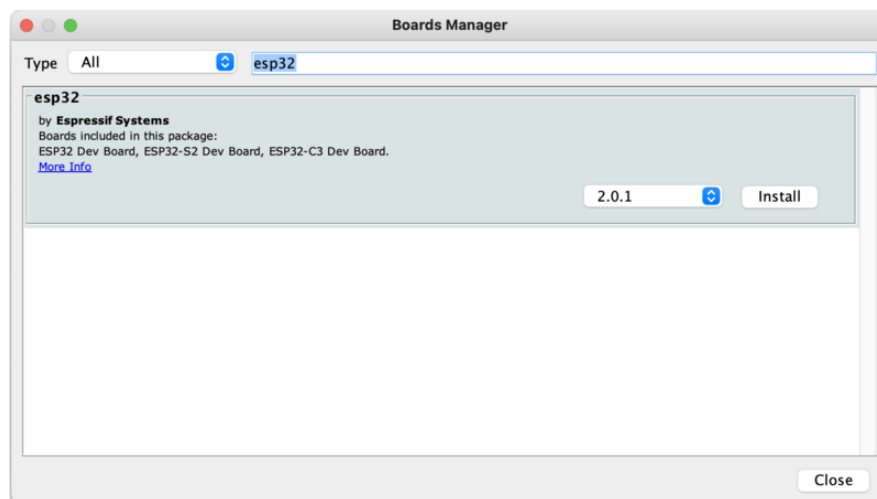
If you're an advanced hacker and want the 'bleeding edge' release that may have fixes (or bugs!) you can check out the dev url instead:

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

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

Once done click OK to save the new preference settings.

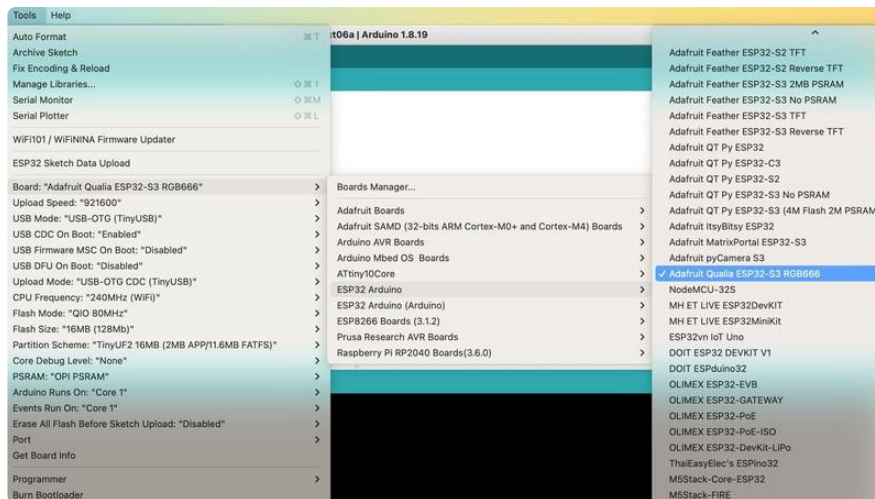
The next step is to actually install the Board Support Package (BSP). Go to the Tools → Board → Board Manager submenu. A dialog should come up with various BSPs. Search for esp32.



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

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

Look for the board called Adafruit Qualia ESP32-S3 RGB666.



Manually Resetting ESP32-S3 Boards

Due to an issue in the Espressif code base, boards with an ESP32-S3 need to be manually reset after uploading code from the Arduino IDE. After your code has been uploaded to the ESP32-S3, press the reset button. After pressing the reset button, your code will begin running.

For additional information, you can track [the issue](#) () on GitHub in the `arduino-esp32` repository.

Make sure to press the reset button after uploading code from the Arduino IDE to the ESP32-S3!

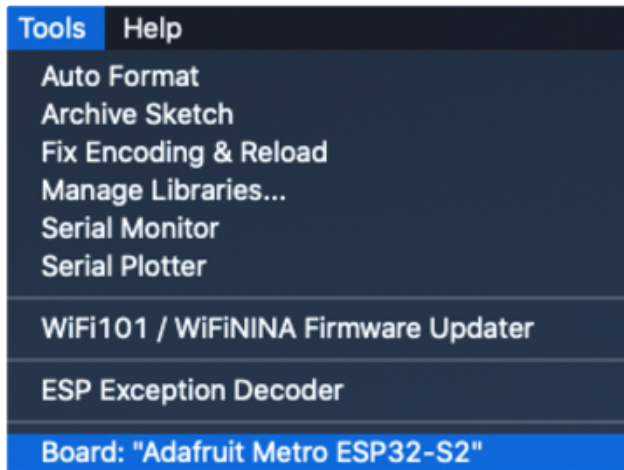
Using with Arduino IDE

Blink

Now you can upload your first blink sketch!

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

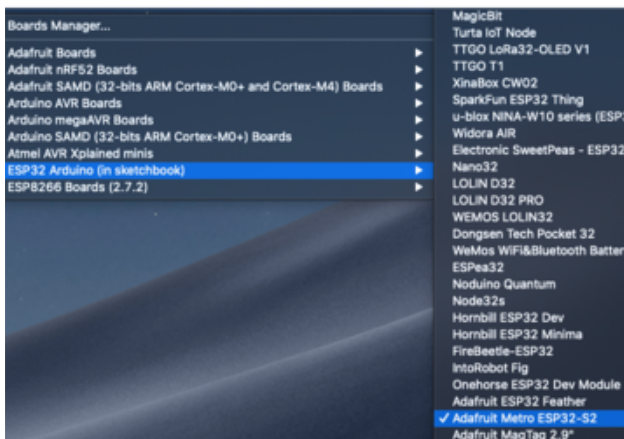
Select ESP32-S2/S3 Board in Arduino IDE



On the Arduino IDE, click:

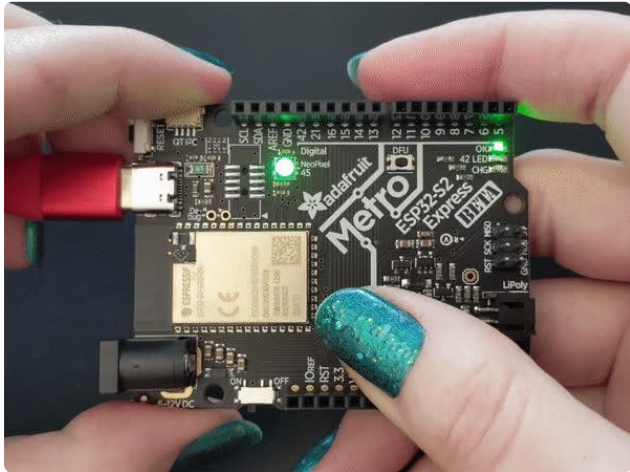
Tools -> Board -> ESP32 Arduino -> Your Adafruit ESP32-S2/S3 board

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



Launch ESP32-S2/S3 ROM Bootloader

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



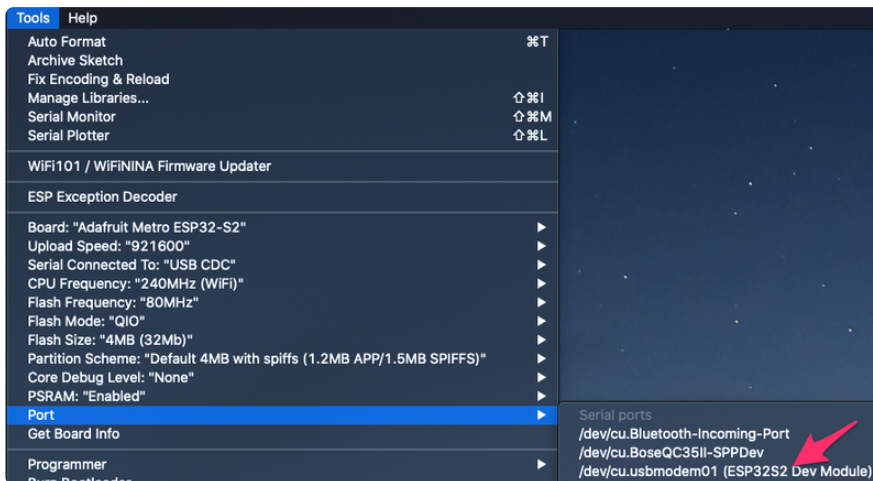
Before we upload a sketch, [place your ESP32-S2/S3 board into ROM bootloader mode \(\)](#).

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

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

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

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



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

Load Blink Sketch

Now open up this Blink example in a new sketch window

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

// the loop function runs over and over again forever
void loop() {
```

```
digitalWrite(LED_BUILTIN, HIGH); // turn the LED on (HIGH is the voltage level)
delay(1000); // wait for a second
digitalWrite(LED_BUILTIN, LOW); // turn the LED off by making the voltage LOW
delay(1000); // wait for a second
}
```

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

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

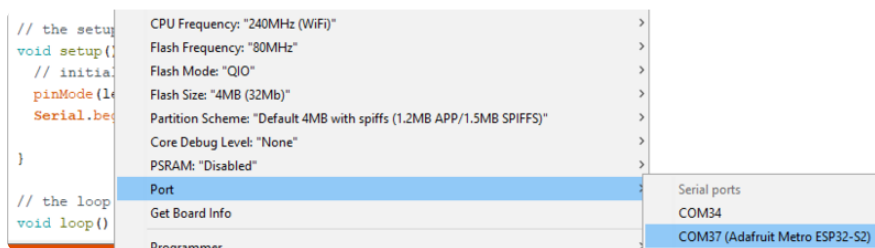
```
To suppress this error, set --after option to 'no_reset'. Copy error messages
Writing at 0x0002b000... (94 %)
Writing at 0x0002b800... (96 %)
Writing at 0x0002c000... (98 %)
Writing at 0x0002c800... (100 %)
Wrote 207904 bytes (117510 compressed) at 0x00010000 in 1.7 seconds (effective
Hash of data verified.
Compressed 3072 bytes to 128...
Writing at 0x00008000... (100 %)
Wrote 3072 bytes (128 compressed) at 0x00008000 in 0.0 seconds (effective 4497
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
To suppress this error, set --after option to 'no_reset'.
To suppress this error, set --after option to 'no_reset'.
```

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

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

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

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



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

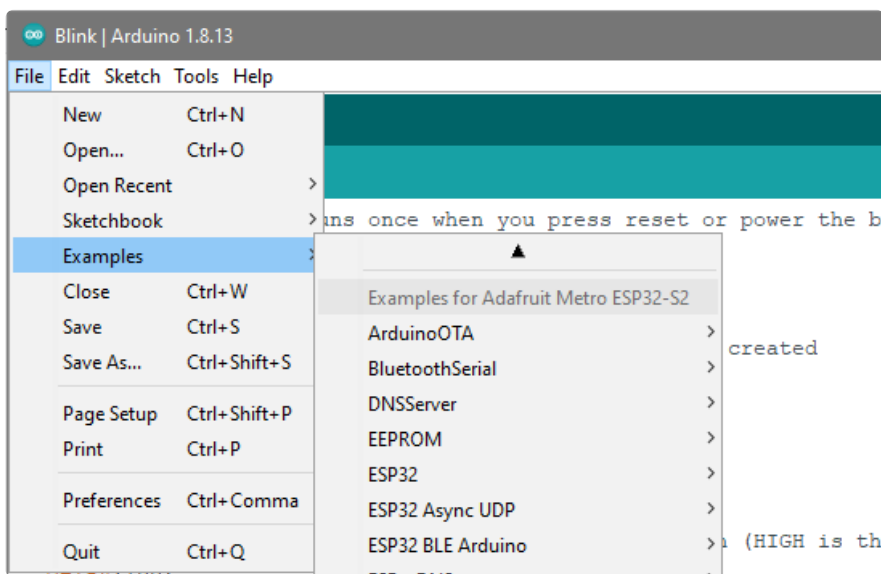
If you ever DON'T see a serial port, or something is not working out with upload you can always manually enter bootloader mode:

- Reset board into ROM bootloader with DFU/BOOT0 + Reset buttons

- Select the ESP32S2/S3 Dev Board ROM bootloader serial port in Tools->Port menu
- Upload sketch
- Click reset button to launch code

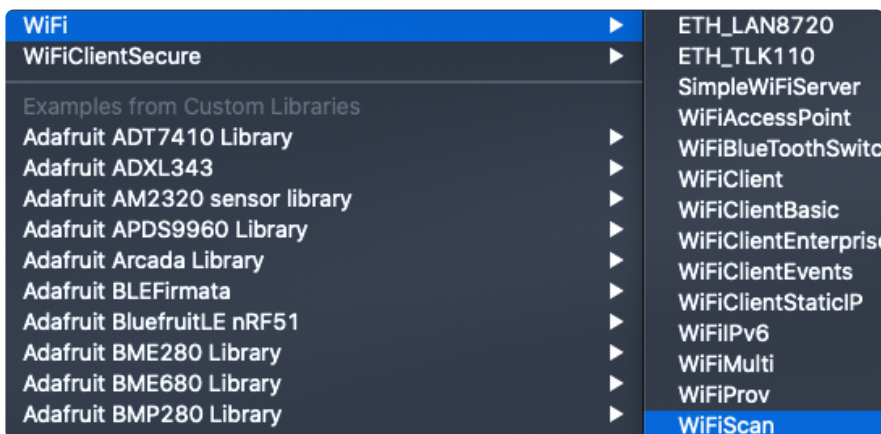
WiFi Test

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



Let's start by scanning the local networks.

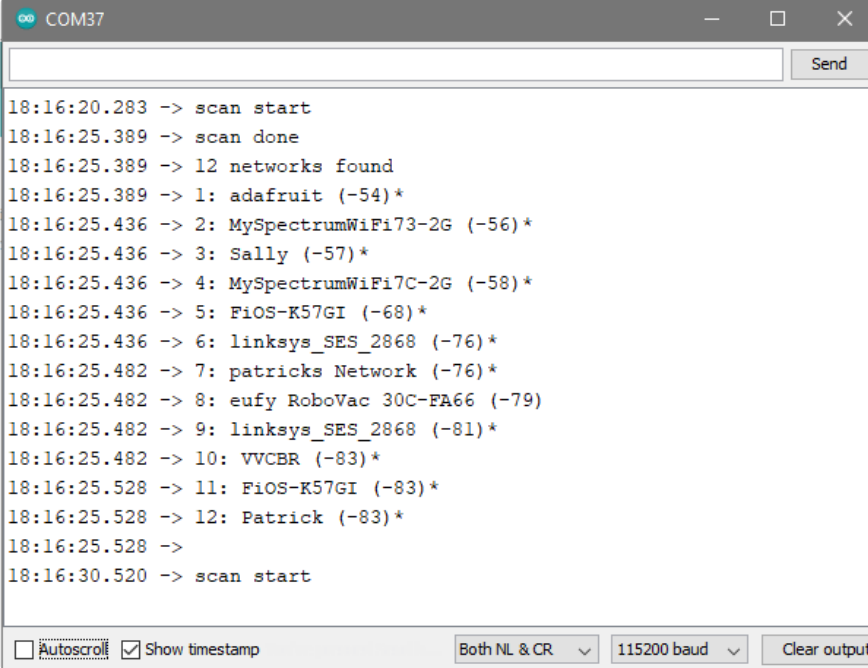
Load up the WiFiScan example under Examples->Examples for YOUR BOARDNAME->WiFi->WiFiScan



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

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

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



```
COM37
18:16:20.283 -> scan start
18:16:25.389 -> scan done
18:16:25.389 -> 12 networks found
18:16:25.389 -> 1: adafruit (-54)*
18:16:25.436 -> 2: MySpectrumWiFi73-2G (-56)*
18:16:25.436 -> 3: Sally (-57)*
18:16:25.436 -> 4: MySpectrumWiFi7C-2G (-58)*
18:16:25.436 -> 5: FiOS-K57GI (-68)*
18:16:25.436 -> 6: linksys_SES_2868 (-76)*
18:16:25.482 -> 7: patricks Network (-76)*
18:16:25.482 -> 8: eufy RoboVac 30C-FA66 (-79)
18:16:25.482 -> 9: linksys_SES_2868 (-81)*
18:16:25.482 -> 10: VVCBR (-83)*
18:16:25.528 -> 11: FiOS-K57GI (-83)*
18:16:25.528 -> 12: Patrick (-83)*
18:16:25.528 ->
18:16:30.520 -> scan start
```

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

WiFi Connection Test

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

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

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

/*
  Web client

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

  This example is written for a network using WPA encryption. For
```

WEP or WPA, change the Wifi.begin() call accordingly.

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

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

#include <WiFi.h>

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

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

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

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

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

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

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

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

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

void loop() {
  // if there are incoming bytes available
  // from the server, read them and print them:
```

```

while (client.available()) {
  char c = client.read();
  Serial.write(c);
}

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

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

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

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

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

```

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

```

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

```

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

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

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

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

disconnecting from server.
```

Secure Connection Example

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

Copy and paste it into the Arduino IDE:

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

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

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

Written by Arturo Guadalupi
last revision November 2015

*/

#include <WiFiClientSecure.h>

// Enter your WiFi SSID and password
```

```

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

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

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

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

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

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

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

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

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

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

uint32_t bytes = 0;

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

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

```

```

    client.stop();
    Serial.print("Read "); Serial.print(bytes); Serial.println(" bytes");

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

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

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

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

```

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

Note we use `WiFiClientSecure client` instead of `WiFiClient client`; to require a SSL connection! 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-Control-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 \(\)](#) 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 \(\)](#).

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

```
// SPDX-FileCopyrightText: 2014 Benoit Blanchon
// SPDX-FileCopyrightText: 2014 Arturo Guadalupi
// SPDX-FileCopyrightText: 2020 Brent Rubell for Adafruit Industries
//
// SPDX-License-Identifier: MIT

/*
This example creates a client object that connects and transfers
data using always SSL, 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>
#include <Wire.h>

// uncomment the next line if you have a 128x32 OLED on the I2C pins
// #define USE_OLED
// uncomment the next line to deep sleep between requests
// #define USE_DEEPSLEEP

#if defined(USE_OLED)
// Some boards have TWO I2C ports, how nifty. We should use the second one sometimes
#if defined(ARDUINO_ADAFRUIT_QTPY_ESP32S2) || \
    defined(ARDUINO_ADAFRUIT_QTPY_ESP32S3_NOPSRAM) || \
    defined(ARDUINO_ADAFRUIT_QTPY_ESP32_PICO)
    #define OLED_I2C_PORT &Wire1
#else
    #define OLED_I2C_PORT &Wire
#endif
#else
    #define OLED_I2C_PORT &Wire
#endif

#include <Adafruit_SSD1306.h>
Adafruit_SSD1306 display = Adafruit_SSD1306(128, 32, OLED_I2C_PORT);
#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"

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

  // Connect to WPA/WPA2 network
  WiFi.begin(ssid, pass);

  #if defined(USE_OLED)
    setupI2C();
    delay(200); // wait for OLED to reset

    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) {
      // wait for serial port to connect. Needed for native USB port only
      delay(10);
    }
  #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

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

  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(1000);

#if defined(USE_DEEPSLEEP)
#if defined(USE_OLED)
  display.clearDisplay();
  display.display();
#endif // OLED
#if defined(NEOPIXEL_POWER)
  digitalWrite(NEOPIXEL_POWER, LOW); // off
#elif defined(NEOPIXEL_I2C_POWER)
  digitalWrite(NEOPIXEL_I2C_POWER, LOW); // off
#endif
  // wake up 1 second later and then go into deep sleep
  esp_sleep_enable_timer_wakeup(10 * 1000UL * 1000UL); // 10 sec
  esp_deep_sleep_start();
#else
  delay(10 * 1000);
#endif
}

void setupI2C() {
  #if defined(ARDUINO_ADAFRUIT_QTPY_ESP32S2) || \
    defined(ARDUINO_ADAFRUIT_QTPY_ESP32S3_NOPSRAM) || \
    defined(ARDUINO_ADAFRUIT_QTPY_ESP32_PICO)
    // ESP32 is kinda odd in that secondary ports must be manually
    // assigned their pins with setPins()!
    Wire1.setPins(SDA1, SCL1);
  #endif

  #if defined(NEOPIXEL_I2C_POWER)
  pinMode(NEOPIXEL_I2C_POWER, OUTPUT);
  digitalWrite(NEOPIXEL_I2C_POWER, HIGH); // on
  #endif

  #if defined(ARDUINO_ADAFRUIT_FEATHER_ESP32S2)
  // turn on the I2C power by setting pin to opposite of 'rest state'
  pinMode(PIN_I2C_POWER, INPUT);
  delay(1);
  bool polarity = digitalRead(PIN_I2C_POWER);
  pinMode(PIN_I2C_POWER, OUTPUT);
  digitalWrite(PIN_I2C_POWER, !polarity);
  #endif
}

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
```

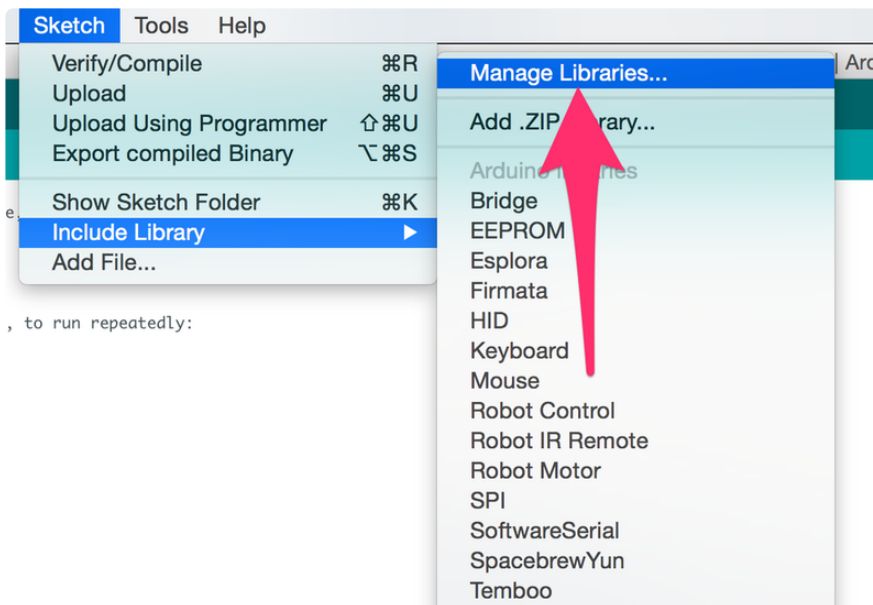
Usage with Adafruit IO

The ESP32-S2/S3 is an affordable, all-in-one, option for connecting your projects to the internet [using our IoT platform, Adafruit IO \(\)](#).

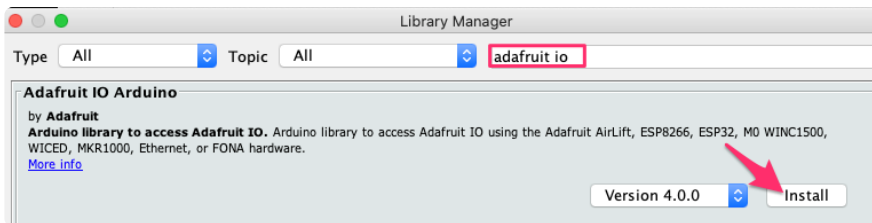
- For more information and guides about Adafruit IO, check out the [Adafruit IO Basics Series. \(\)](#)

Install Libraries

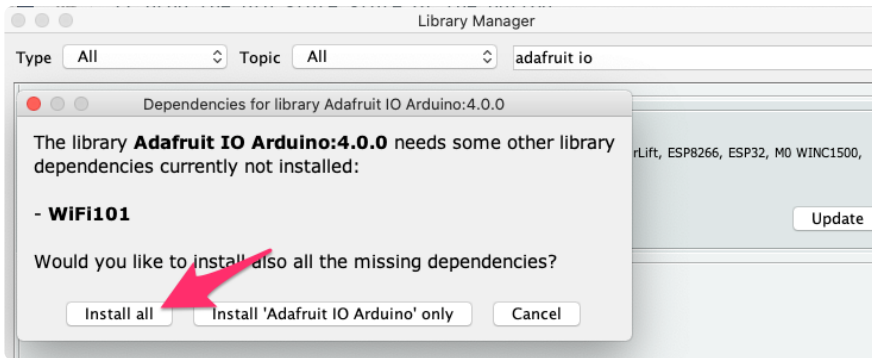
In the Arduino IDE, navigate to Sketch -> Include Library->Manage Libraries...



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 \(\)](#). Next, navigate to the Adafruit IO Dashboards page.

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



brubell > Dashboards



Create a new Dashboard ×

Name

Description

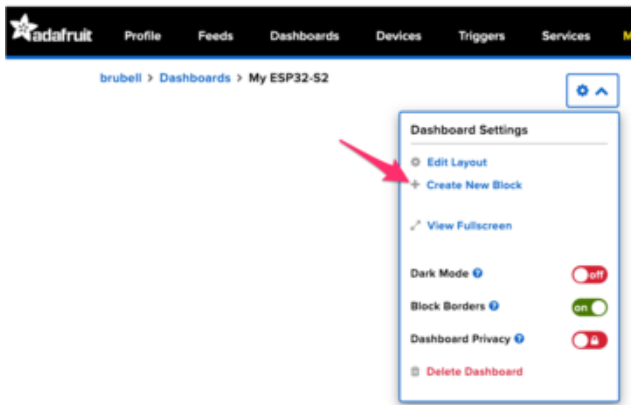
Show Header Image

Header Image
 No file chosen
[Sample header image with breakpoints marked.](#)

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

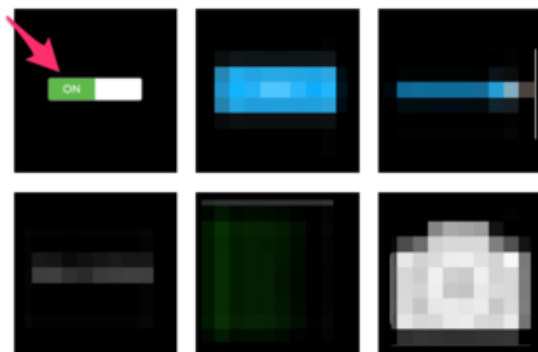
<input type="checkbox"/> LoRa Feather Network	lora-feather-network
<input type="checkbox"/> My Air Quality Sensor	my-air-quality-sensor
<input type="checkbox"/> My ESP32-S2	my-esp32-s2
<input type="checkbox"/> My Garage	my-garage

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.



Create a new block

Click on the block you would like to add to your dashboard. You can switch the block type later if you change your mind.



<input type="checkbox"/>	led	HIGH	9 minutes
<input type="checkbox"/>	lwill	rip	over 1 year
<input type="checkbox"/>	moisture	2	1 day
<input type="checkbox"/>	neopixel		2 days
<input type="checkbox"/>	outdoor-lights	#000000	about 2 ye
<input type="checkbox"/>	relay	morning	about 3 ho
<input type="checkbox"/>	temperature	72	1 day
<input type="checkbox"/>	test	66	2 days
<input type="checkbox"/>	timecube	4.5	almost 2 ye
<input type="checkbox"/>	zapemail	Gary Thompson...	1 day

led

My Feeds

Feed Name	Last value	Recorded
<input type="checkbox"/> battery	55	1 day
<input type="checkbox"/> digital	1	about 17 hours
<input type="checkbox"/> humidity	10	2 days
<input type="checkbox"/> image	/9j/4QAWRXhp...	5 months
<input type="checkbox"/> indoor-lights	#000000	about 2 years
<input checked="" type="checkbox"/> led		less than a min...
<input type="checkbox"/> lwill	rip	over 1 year
<input type="checkbox"/> moisture	2	1 day
<input type="checkbox"/> neopixel		2 days
<input type="checkbox"/> outdoor-lights	#000000	about 2 years


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.

Block Title (optional)

Block Preview

Button On Text

Button Off Text



Toggle A toggle button is useful if you have an ON or OFF type of state. You can configure what values are sent on press and release.

Test Value

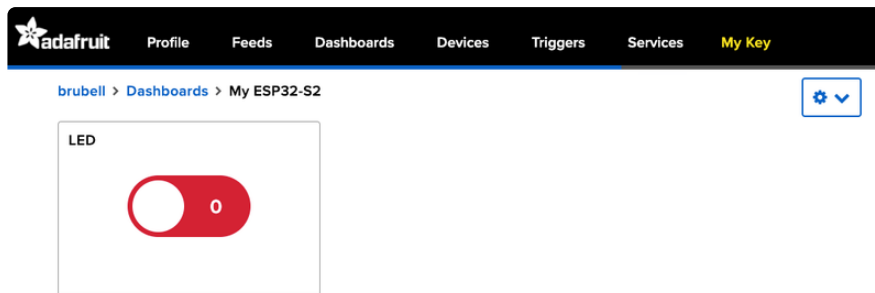
Published Value

0 bytes

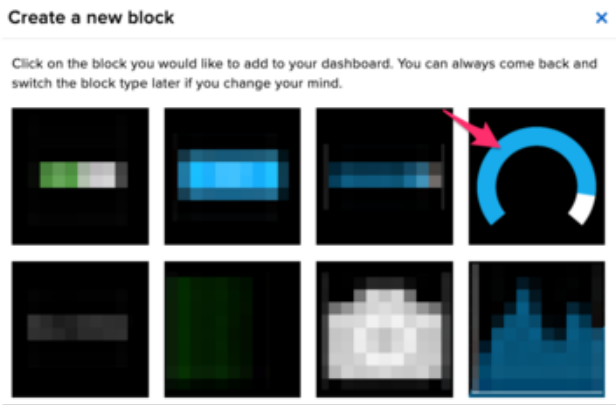
[< Previous step](#) [Create block](#)

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.



Block Name	Feed Name	Last Update	Lock Icon
<input type="checkbox"/> image	/9j/4QAWRXhp...	5 months	🔒
<input type="checkbox"/> indoor-lights	#000000	about 2 years	🔒
<input type="checkbox"/> led		16 minutes	🔒
<input type="checkbox"/> lwill	rip	over 1 year	🔒
<input type="checkbox"/> moisture	2	1 day	🔒
<input type="checkbox"/> neopixel		2 days	🔒
<input type="checkbox"/> outdoor-lights	#000000	about 2 years	🔒
<input type="checkbox"/> relay	morning	about 3 hours	🔒
<input type="checkbox"/> temperature	72	1 day	🔒
<input type="checkbox"/> test	66	2 days	🔒
<input type="checkbox"/> timecube	4.5	almost 2 years	🔒
<input type="checkbox"/> zapemail	Gary Thompson...	1 day	🔒
<input type="text" value="button"/>			

Create

Create a Gauge Block

A gauge is a read only block type that shows a fixed range of values. Choose a single feed you would like to connect to this gauge feed within a group.

My Feeds

Feed Name	Last value
<input type="checkbox"/> battery	55
<input checked="" type="checkbox"/> button	

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.

Block settings

In this final step, you can give your block a title and see a preview of how it will look. Customize the look and feel of your block with the remaining settings. When you are ready, click the "Create Block" button to send it to your dashboard.

Block Title (optional)

Gauge Min Value


Gauge Max Value

Gauge Width

Gauge Label

Low Warning Value

Block Preview



Gauge A gauge is a read only block type that shows a fixed range of values.

Under block settings,

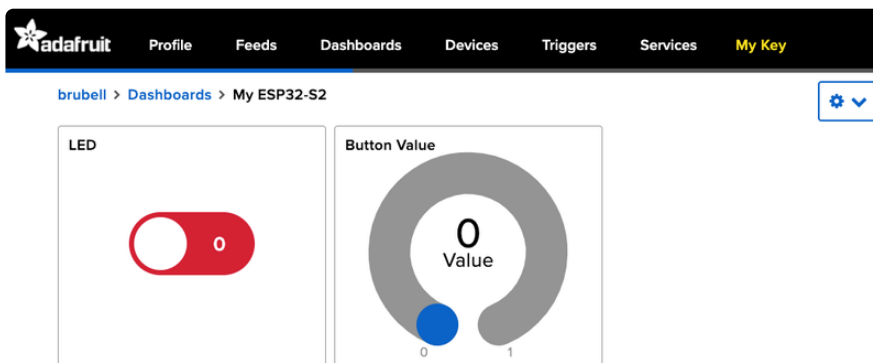
Change Block Title to Button Value

Change Gauge Min Value to 0, the button's state when it's off

Change Gauge Max Value to 1, the button's state when it's on

Click Create block

Your dashboard should look like the following:



Code Usage

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

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

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

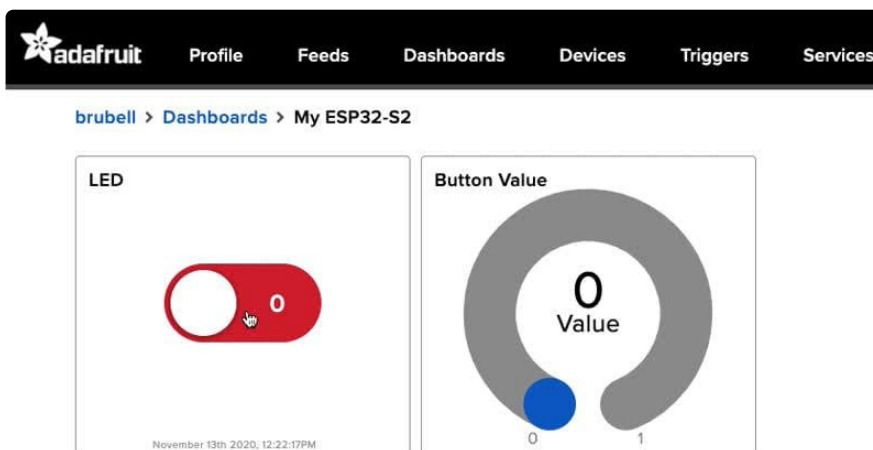
```
adafruitio_26_led_btn - config.h | Arduino 1.8.13
adafruitio_26_led_btn  config.h
1 /***** Adafruit IO Config *****/
2
3 // visit io.adafruit.com if you need to create an account,
4 // or if you need your Adafruit IO key.
5 #define IO_USERNAME "your_username"
6 #define IO_KEY "your_key"
7
```

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

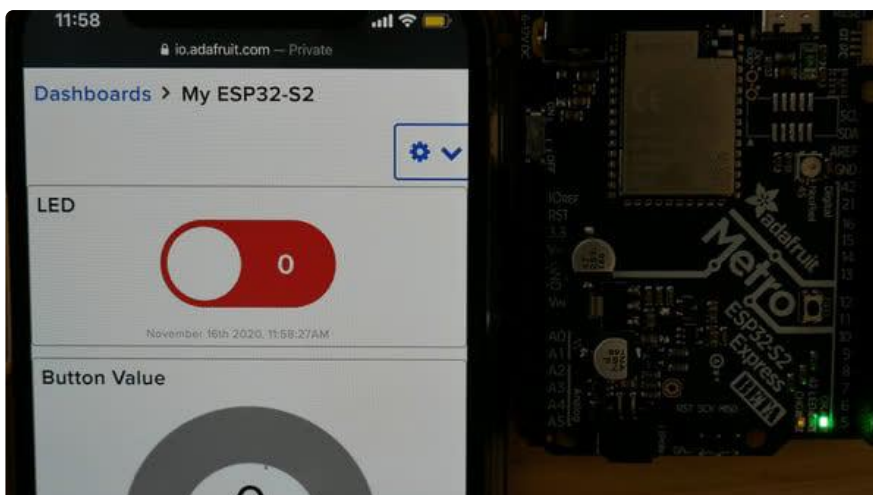
```
adafruitio_26_led_btn config.h
20
21 #define WIFI_SSID "your_ssid"
22 #define WIFI_PASS "your_pass"
23
24 // uncomment the following line if you are using airlift
25 //#define USE_AIRLIFT
26
27 // uncomment the following line if you are using winc1500
28 // #define USE_WINC1500
29
```

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

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



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



Arduino Rainbow Demo

We have a Circular Rainbow demo available for the Qualia ESP32-S3 that runs in Arduino. Below is the code to run it.

This currently will not compile with version 3.0.0 of the ESP32 Board Support Package. Please use version 2 of the BSP.

```
// SPDX-FileCopyrightText: 2023 Limor Fried for Adafruit Industries
//
// SPDX-License-Identifier: MIT

#include <Arduino_GFX_Library.h>
#include <Adafruit_FT6206.h>

Arduino_XCA9554SWSPI *expander = new Arduino_XCA9554SWSPI(
  PCA_TFT_RESET, PCA_TFT_CS, PCA_TFT_SCK, PCA_TFT_MOSI,
  &Wire, 0x3F);

Arduino_ESP32RGBPanel *rgbpanel = new Arduino_ESP32RGBPanel(
  TFT_DE, TFT_VSYNC, TFT_HSYNC, TFT_PCLK,
  TFT_R1, TFT_R2, TFT_R3, TFT_R4, TFT_R5,
  TFT_G0, TFT_G1, TFT_G2, TFT_G3, TFT_G4, TFT_G5,
  TFT_B1, TFT_B2, TFT_B3, TFT_B4, TFT_B5,
  1 /* hsync_polarity */, 50 /* hsync_front_porch */, 2 /* hsync_pulse_width */,
44 /* hsync_back_porch */,
  1 /* vsync_polarity */, 16 /* vsync_front_porch */, 2 /* vsync_pulse_width */,
18 /* vsync_back_porch */
// ,1, 30000000
);

Arduino_RGB_Display *gfx = new Arduino_RGB_Display(
// 2.1" 480x480 round display
  480 /* width */, 480 /* height */, rgbpanel, 0 /* rotation */, true /*
auto_flush */,
  expander, GFX_NOT_DEFINED /* RST */, TL021WVC02_init_operations,
sizeof(TL021WVC02_init_operations));

// 2.8" 480x480 round display
// 480 /* width */, 480 /* height */, rgbpanel, 0 /* rotation */, true /*
auto_flush */,
// expander, GFX_NOT_DEFINED /* RST */, TL028WVC01_init_operations,
sizeof(TL028WVC01_init_operations));

// 3.4" 480x480 square display
// 480 /* width */, 480 /* height */, rgbpanel, 0 /* rotation */, true /*
auto_flush */,
// expander, GFX_NOT_DEFINED /* RST */, tl034wvs05_b1477a_init_operations,
sizeof(tl034wvs05_b1477a_init_operations));

// 3.2" 320x820 rectangle bar display
// 320 /* width */, 820 /* height */, rgbpanel, 0 /* rotation */, true /*
auto_flush */,
// expander, GFX_NOT_DEFINED /* RST */, tl032fwv01_init_operations,
sizeof(tl032fwv01_init_operations));

// 4.0" 720x720 square display
// 720 /* width */, 720 /* height */, rgbpanel, 0 /* rotation */, true /*
auto_flush */,
// expander, GFX_NOT_DEFINED /* RST */, NULL, 0);

// 4.0" 720x720 round display
```

```

// 720 /* width */, 720 /* height */, rgbpanel, 0 /* rotation */, true /*
auto_flush */,
// expander, GFX_NOT_DEFINED /* RST */, hd40015c40_init_operations,
sizeof(hd40015c40_init_operations));
// needs also the rgbpanel to have these pulse/sync values:
// 1 /* hsync_polarity */, 46 /* hsync_front_porch */, 2 /* hsync_pulse_width */,
44 /* hsync_back_porch */,
// 1 /* vsync_polarity */, 50 /* vsync_front_porch */, 16 /* vsync_pulse_width
*/, 16 /* vsync_back_porch */

uint16_t *colorWheel;

// The FTxxxx based CTP overlays uses hardware I2C (SCL/SDA)
#define I2C_TOUCH_ADDR 0x48 // often but not always 0x38!
Adafruit_FT6206 ctp = Adafruit_FT6206(); // this library also supports
FT5336U!
bool touchOK = false; // we will check if the touchscreen exists

void setup(void)
{
  Serial.begin(115200);
  //while (!Serial) delay(100);

#ifdef GFX_EXTRA_PRE_INIT
  GFX_EXTRA_PRE_INIT();
#endif

  Serial.println("Beginning");
  // Init Display

  Wire.setClock(1000000); // speed up I2C
  if (!gfx->begin()) {
    Serial.println("gfx->begin() failed!");
  }

  Serial.println("Initialized!");

  gfx->fillScreen(BLACK);

  expander->pinMode(PCA_TFT_BACKLIGHT, OUTPUT);
  expander->digitalWrite(PCA_TFT_BACKLIGHT, HIGH);

  colorWheel = (uint16_t *) ps_malloc(gfx->width() * gfx->height() *
sizeof(uint16_t));
  if (colorWheel) {
    generateColorWheel(colorWheel);
    gfx->draw16bitRGBBitmap(0, 0, colorWheel, gfx->width(), gfx->height());
  }

  if (!ctp.begin(0, &Wire, I2C_TOUCH_ADDR)) {
    Serial.println("No touchscreen found");
    touchOK = false;
  } else {
    Serial.println("Touchscreen found");
    touchOK = true;
  }
}

void loop()
{
  if (touchOK && ctp.touched()) {
    TS_Point p = ctp.getPoint(0);
    Serial.printf("(%d, %d)\n", p.x, p.y);
    gfx->fillRect(p.x, p.y, 5, 5, WHITE);
  }

  // use the buttons to turn off
  if (!expander->digitalRead(PCA_BUTTON_DOWN)) {

```

```

    expander->digitalWrite(PCA_TFT_BACKLIGHT, LOW);
  }
  // and on the backlight
  if (! expander->digitalRead(PCA_BUTTON_UP)) {
    expander->digitalWrite(PCA_TFT_BACKLIGHT, HIGH);
  }
}

// https://chat.openai.com/share/8edee522-7875-444f-9fea-ae93a8dfa4ec
void generateColorWheel(uint16_t *colorWheel) {
  int width = gfx->width();
  int height = gfx->height();
  int half_width = width / 2;
  int half_height = height / 2;
  float angle;
  uint8_t r, g, b;
  int index, scaled_index;

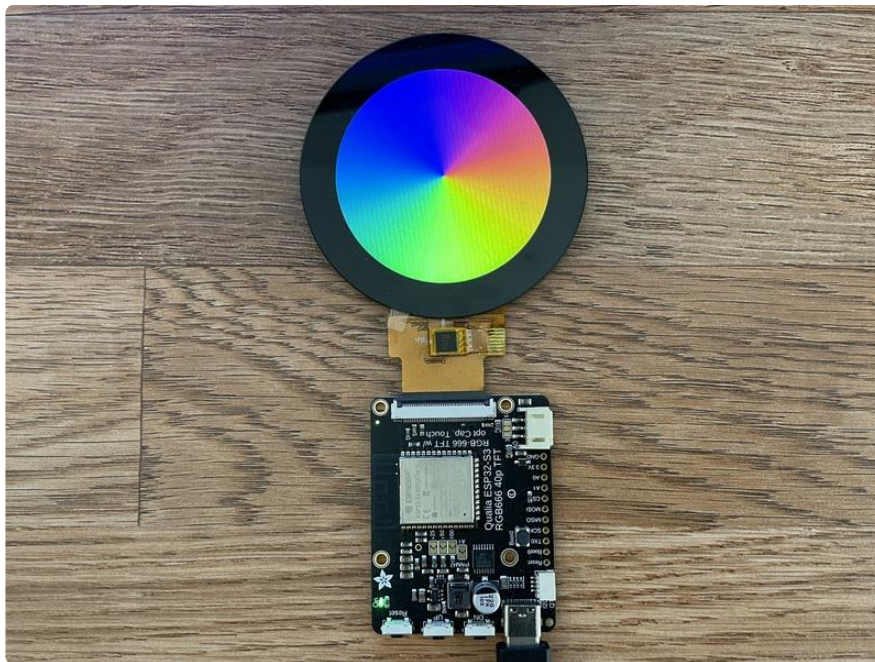
  for(int y = 0; y < half_height; y++) {
    for(int x = 0; x < half_width; x++) {
      index = y * half_width + x;
      angle = atan2(y - half_height / 2, x - half_width / 2);
      r = uint8_t(127.5 * (cos(angle) + 1));
      g = uint8_t(127.5 * (sin(angle) + 1));
      b = uint8_t(255 - (r + g) / 2);
      uint16_t color = RGB565(r, g, b);

      // Scale this pixel into 4 pixels in the full buffer
      for(int dy = 0; dy < 2; dy++) {
        for(int dx = 0; dx < 2; dx++) {
          scaled_index = (y * 2 + dy) * width + (x * 2 + dx);
          colorWheel[scaled_index] = color;
        }
      }
    }
  }
}

```

This sketch was written for either of the 2.1" Round 480x480 RGB-666 displays.

Now upload the sketch to your Qualia ESP32-S3 and make sure a round display is connected. You may need to press the Reset button to reset the microcontroller. You should now see a circular rainbow appear on the display!



Here are some pre-compiled UF2s for various displays so you can instantly test them!

2.1" Round 480x480
TL021WVC02ColorTest.UF2

TL028WVC01ColorTest.UF2

3.2" Bar 820x320
TL032FWV01ColorTest.UF2

3.4" Square 480x480
TL034WVS05ColorTest.UF2

720x720 test codes may be flickery as its compiled with IDF 4:

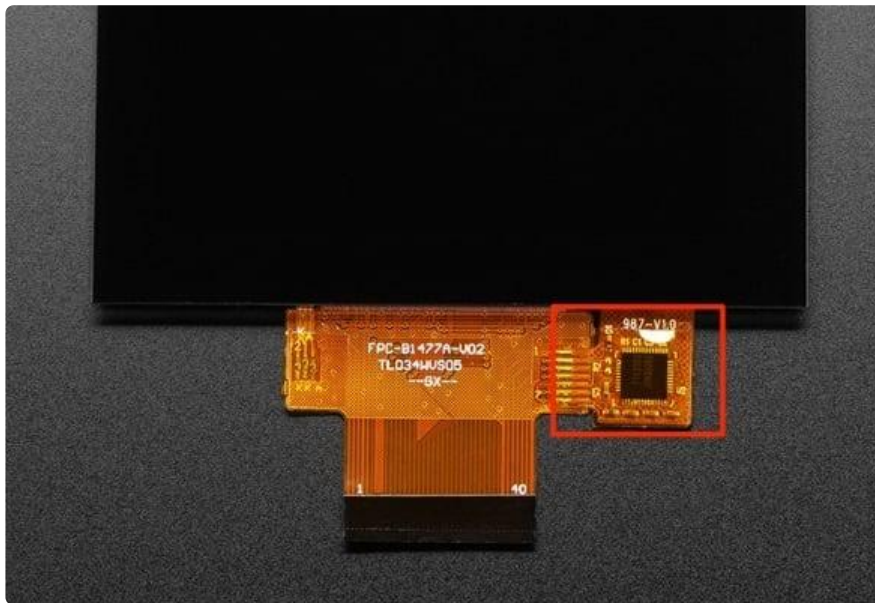
HD40015C40ColorTest.UF2

4" Square 720x720
TL040HDS20ColorTest.UF2

Arduino Touch Display Usage

If you have a display with touch, you can use the [Adafruit_FT6206_Library \(\)](#) library to read the touch data. The Capacitive Touch controller is communicated to by I2C. If

you're not sure if you have a touch display, just check if your includes a square IC connected off to the side of the main ribbon cable.



Determining the I2C Address

You can scan for I2C devices by running the WireScan example. You can find it by going to File → Examples → Wire → WireScan and uploading the sketch. Once it is running, open the serial monitor to see which devices it finds.

```
Scanning for I2C devices ...  
I2C device found at address 0x3F  
I2C device found at address 0x48  
Scanning for I2C devices ...  
I2C device found at address 0x3F  
I2C device found at address 0x48  
Scanning for I2C devices ...
```

You should see a couple of devices listed. These will be the GPIO expander and the touch controller. The GPIO Expander is at **0x3F** by default, though it's possible to change the address with the solderable jumpers on the reverse side. The other address should be the touch controller. On the TL040HDS20 4.0" square display, it shows up as **0x48**, but it's possible it may be a different value on other displays.

Initializing the Touch Controller

In order to use the controller, it will need to first be initialized. You can use the following code to initialize it. If your I2C address differs, change it to the appropriate value.

```
#include <Wire.h> // this is needed for FT6206
#include <Adafruit_FT6206.h>;

#define I2C_TOUCH_ADDR 0x48 // often but not always 0x48!
Adafruit_FT6206 ctp = Adafruit_FT6206(); // this library also supports FT5336U!

Serial.begin(115200); // To print the output

if (!ctp.begin(0, &Wire, I2C_TOUCH_ADDR)) {
  Serial.println("No touchscreen found");
}
```

Reading from the Touch Controller

To read from the controller, check if it has been **touched** in the main loop and if so, read the **x** and **y** coordinates.

```
if (ctp.touched()) {
  TS_Point p = ctp.getPoint(0);
  Serial.printf("(%d, %d)\n", p.x, p.y);
}
```

Example

To see an example, check out the [Arduino Rainbow Demo Page \(\)](#).

Install UF2 Bootloader

If your board has a UF2 bootloader, you do not need to follow the steps on this page. Try to enter the UF2 bootloader before continuing! Double-tap the reset button to do so.

The Qualia ESP32-S3 RGB-666 ships with a UF2 bootloader which allows the board to show up as TFT_S3BOOT when you double-tap the reset button, and enables you to drag and drop UF2 files to update the firmware.

On ESP32-S2/S3, there is no bootloader protection for the UF2 bootloader. That means it is possible to erase or damage the UF2 bootloader, especially if you upload

an Arduino sketch to an ESP32-S2/S3 board that doesn't "know" there's a bootloader it should not overwrite!

It turns out, however, the ESP32-S2/S3 comes with a second bootloader: the ROM bootloader. Thanks to the ROM bootloader, you don't have to worry about damaging the UF2 bootloader. The ROM bootloader can never be disabled or erased, so its always there if you need it! You can simply re-load the UF2 bootloader from the ROM bootloader.

If your UF2 bootloader ends up damaged or overwritten, you can follow the steps found in the [Factory Reset and Bootloader Repair \(\)](#) section of the Factory Reset page in this guide.

Once completed, you'll return to where the board was when you opened the package. Then you'll be back in business, and able to continue with your existing plans!

Factory Reset

The Qualia ESP32-S3 microcontroller ships running a circular rainbow gradient example for the round 480x480 display. It's lovely, but you probably had other plans for the board. As you start working with your board, you may want to return to the original code to begin again, or you may find your board gets into a bad state. Either way, this page has you covered.

You're probably used to seeing the TFT_S3BOOT drive when loading CircuitPython or Arduino. The TFT_S3BOOT drive is part of the UF2 bootloader, and allows you to drag and drop files, such as CircuitPython. However, on the ESP32-S3 the UF2 bootloader can become damaged.

Factory Reset Firmware UF2

If you have a bootloader still installed - which means you can double-click to get the TFT_S3BOOT drive to appear, then you can simply drag this UF2 file over to the BOOT drive.

To enter bootloader mode, plug in the board into a USB cable with data/sync capability. Press the reset button once, wait till the RGB LED turns purple, then press the reset button again. Then drag this file over:

[Qualia S3 RGB-666 Factory Reset](#)

Your board is now back to its factory-shipped state! You can now begin again with your plans for your board.

Factory Reset and Bootloader Repair

What if you tried double-tapping the reset button, and you still can't get into the UF2 bootloader? Whether your board shipped without the UF2 bootloader, or something damaged it, this section has you covered.

There is no bootloader protection for the UF2 bootloader. That means it is possible to erase or damage the UF2 bootloader, especially if you upload an Arduino sketch to an ESP32-S2/S3 board that doesn't "know" there's a bootloader it should not overwrite!

It turns out, however, the ESP32-S2/S3 comes with a second bootloader: the ROM bootloader. Thanks to the ROM bootloader, you don't have to worry about damaging the UF2 bootloader. The ROM bootloader can never be disabled or erased, so it's always there if you need it! You can simply re-load the UF2 bootloader from the ROM bootloader.

Completing a factory reset will erase your board's firmware which is also used for storing CircuitPython/Arduino/Files! Be sure to back up your data first.

There are two ways to do a factory reset and bootloader repair. The first is using WebSerial through a Chromium-based browser, and the second is using `esptool` via command line. We highly recommend using WebSerial through Chrome/Chromium.

The next section walks you through the prerequisite steps needed for both methods.

Download .bin and Enter Bootloader

Step 1. Download the factory-reset-and-bootloader.bin file

Save the following file wherever is convenient for you. You will need to access it from the WebSerial ESPTool.

Note that this file is approximately 3MB. This is not because the bootloader is 3MB, it is because the bootloader is near the end of the available flash. Most of the file is empty but it's easier to program if you use a combined file.

Qualia S3 RGB-666 Factory Reset .bin File

Step 2. Enter ROM bootloader mode

Entering the ROM bootloader is easy. Complete the following steps.

Before you start, make sure your ESP32-S2/S3 is plugged into USB port to your computer using a data/sync cable. Charge-only cables will not work!

To enter the bootloader:

1. Press and hold the BOOT/DFU button down. Don't let go of it yet!
2. Press and release the Reset button. You should still have the BOOT/DFU button pressed while you do this.
3. Now you can release the BOOT/DFU button.



No USB drive will appear when you've entered the ROM bootloader. This is normal!

Now that you've downloaded the .bin file and entered the bootloader, you're ready to continue with the factory reset and bootloader repair process. The next two sections walk you through using WebSerial and [esptool](#).

The WebSerial ESPTool Method

We highly recommend using WebSerial ESPTool method to perform a factory reset and bootloader repair. However, if you'd rather use esptool via command line, you can skip this section.

This method uses the WebSerial ESPTool through Chrome or a Chromium-based browser. The WebSerial ESPTool was designed to be a web-capable option for programming ESP32-S2/S3 boards. It allows you to erase the contents of the microcontroller and program up to four files at different offsets.

You will have to use a Chromium browser (like Chrome, Opera, Edge...) for this to work, Safari and Firefox, etc. are not supported because we need Web Serial and only Chromium is supporting it to the level needed.

Follow the steps to complete the factory reset.

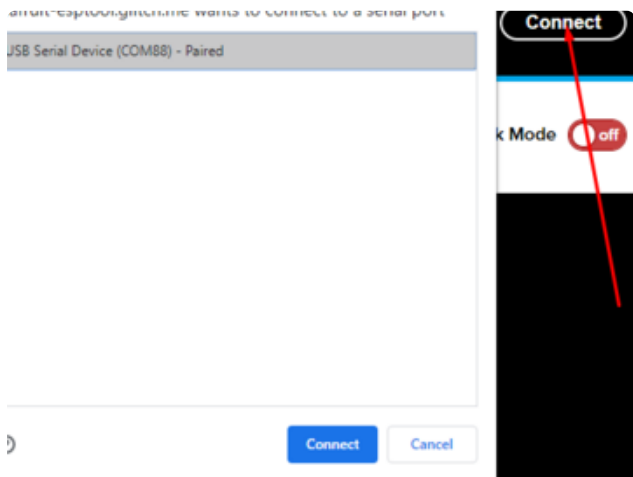
If you're using Chrome 88 or older, see the Older Versions of Chrome section at the end of this page for instructions on enabling Web Serial.

Connect

You should have plugged in only the ESP32-S2/S3 that you intend to flash. That way there's no confusion in picking the proper port when it's time!



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



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.

Remember, you should remove all other USB devices so only the ESP32-S2/S3 board is attached, that way there's no confusion over multiple ports!

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

```

ESP Web Flasher loaded.
Connecting...
Connected successfully.
Try hard reset.
Chip type ESP32-S2
Connected to ESP32-S2
MAC Address: 7C:DF:A1:06:8D:D0
Uploading stub...
Running stub...
Stub is now running...
Detecting Flash Size
FlashId: 0x164020
Flash Manufacturer: 20
Flash Device: 4016
Auto-detected Flash size: 4MB
  
```

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



```

Adafuit WebSerial ESPTool loaded.
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
Timed out after 100 milliseconds
Timed out after 100 milliseconds
Changed baud rate to 921600
Connected to ESP32-S2
MAC Address: E6:85:6D:92:AA:32
  
```

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

Erase the Contents

This will erase everything on your board! If you have access, and wish to keep any code, now is the time to ensure you've backed up everything.



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.

```
Erasing flash memory. Please wait...
Finished. Took 15899ms to erase.
```

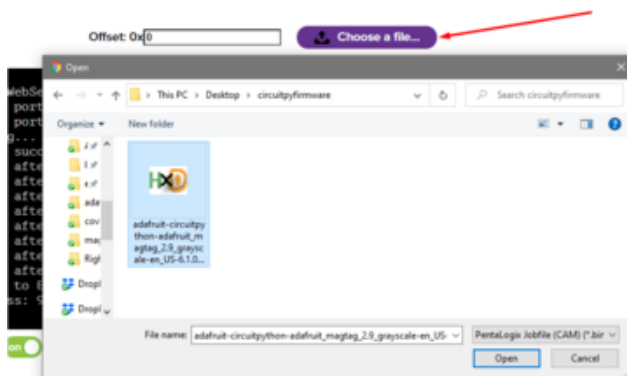
You'll see "Erasing flash memory. Please wait..." This will eventually be followed by "Finished." and the amount of time it took to erase.

Do not disconnect! Immediately continue on to programming the ESP32-S2/S3.

Do not disconnect after erasing! Immediately continue on to the next step!

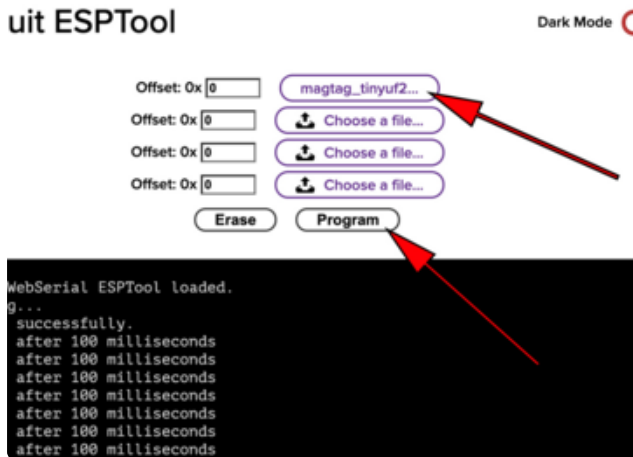
Program the ESP32-S2/S3

Programming the microcontroller can be done with up to four files at different locations, but with the board-specific factory-reset.bin file, which you should have downloaded under Step 1 on this page, you only need to use one file.



Click on the first Choose a file.... (The tool will only attempt to program buttons with a file and a unique location.) Then, select the *-factory-reset.bin file you downloaded in Step 1 that matches your board.

Verify that the Offset box next to the file location you used is (0x) 0.



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



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

Once completed, you can skip down to the section titled Reset the Board.

The `esptool` Method (for advanced users)

If you used WebSerial ESPTool, you do not need to complete the steps in this section!

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

Install ESPTool.py

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
```

Make sure you are running esptool v3.0 or higher, which adds ESP32-S2/S3 support.

Test the Installation

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_region,version,get_security_info}
              ...
```

Connect

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/S3.

```
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'.
```


Erase the Flash

Before programming the board, it is a good idea to erase the flash. Run the following command.

```
esptool.py erase_flash
```

You must be connected (by running the command in the previous section) for this command to work as shown.

```
> esptool.py erase_flash
esptool.py v4.7-dev
Found 2 serial ports
Serial port /dev/cu.usbmodem2121101
Connecting...
Detecting chip type... ESP32-S3
Chip is ESP32-S3 (QFN56) (revision v0.1)
Features: WiFi, BLE, Embedded PSRAM 8MB (AP_3v3)
Crystal is 40MHz
MAC: 34:85:18:9b:f1:7c
Uploading stub...
Running stub...
Stub running...
Erasing flash (this may take a while)...
Chip erase completed successfully in 2.3s
Hard resetting via RTS pin...
```

Installing the Bootloader

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

```
esptool.py --port COM88 write_flash 0x0 tinyuf2_combo.bin
```

Don't forget to change the `--port` name to match.

Adjust the bootloader filename accordingly if it differs from `tinyuf2_combo.bin`.

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

Once completed, you can continue to the next section.

Reset the board

Now that you've reprogrammed the board, you need to reset it to continue. Click the reset button to launch the new firmware.

If you have a 480x480 round display plugged in, you should see a circular rainbow gradient appear on the display.

You've successfully returned your board to a factory reset state!

Older Versions of Chrome

As of chrome 89, Web Serial is already enabled, so this step is only necessary on older browsers.

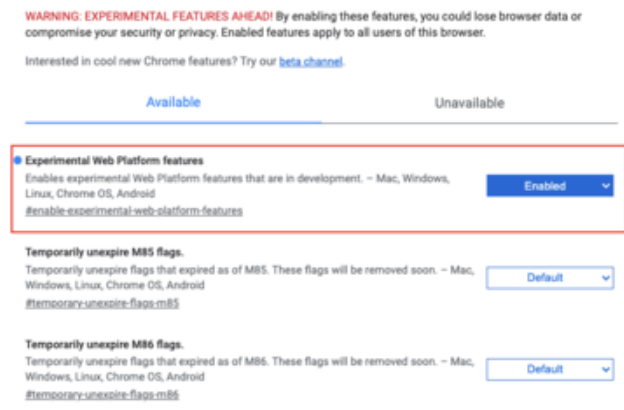
We suggest updating to Chrome 89 or newer, as Web Serial is enabled by default.

If you must continue using an older version of Chrome, follow these steps to enable Web Serial.

Sorry, **Web Serial** is not supported on this device, make sure you're running Chrome 78 or later and have enabled the `#enable-experimental-web-platform-features` flag in `chrome://flags`

If you receive an error like the one shown when you visit the WebSerial ESPTool site, you're likely running an older version of Chrome.

You must be using Chrome 78 or later to use Web Serial.



To enable Web Serial in Chrome versions 78 through 88:

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

The Flash an Arduino Sketch Method

This section outlines flashing an Arduino sketch onto your ESP32-S2/S3 board, which automatically installs the UF2 bootloader as well.

Arduino IDE Setup

If you don't already have the Arduino IDE installed, the first thing you will need to do is to download the latest release of the Arduino IDE. ESP32-S2/S3 requires version 1.8 or higher. Click the link to download the latest.

[Arduino IDE Download](#)

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 > Preferences menu in Windows or Linux, or the Arduino > Preferences menu on OS X.

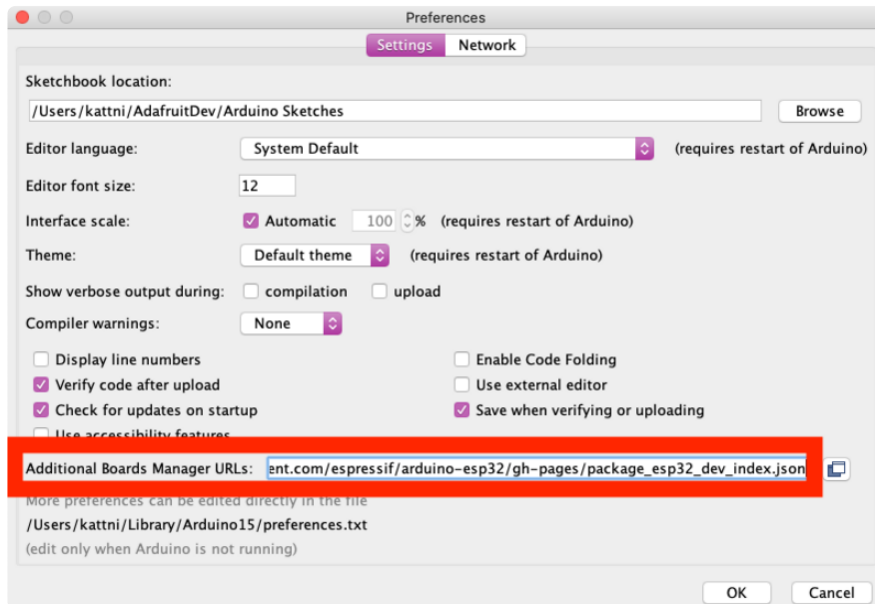
The Preferences window will open.

In the Additional Boards Manager URLs field, you'll want to add a new URL. The list of URLs is comma separated, and you will only have to add each URL once. The URLs point to index files that the Board Manager uses to build the list of available & installed boards.

Copy the following URL.

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

Add the URL to the the Additional Boards Manager URLs field (highlighted in red below).



Click OK to save and close Preferences.

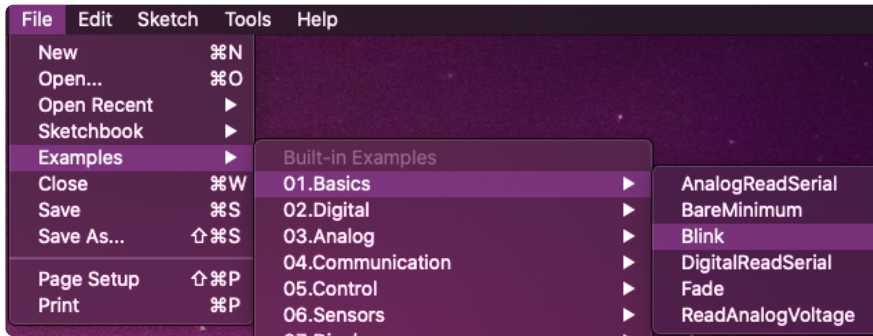
In the Tools > Boards menu you should see the ESP32 Arduino menu. In the expanded menu, it should contain the ESP32 boards along with all the latest ESP32-S2 boards.

Now that your IDE is setup, you can continue on to loading the sketch.

Load the Blink Sketch

In the Tools > Boards menu you should see the ESP32 Arduino menu. In the expanded menu, look for the menu option for the Adafruit Qualia ESP32-S3 RGB666, and click on it to choose it.

Open the Blink sketch by clicking through File > Examples > 01.Basics > Blink.



Once open, click Upload from the sketch window.



Once successfully uploaded, the little red LED will begin blinking once every second. At that point, you can now enter the bootloader.

The Qualia ESP32-S3 RGB-666 does not have a little red LED, so the default Blink sketch will fail.

If you change `LED_BUILTIN` to `13`, the sketch will compile and upload. Be aware that, once the sketch is loaded, nothing will happen on the board. However, you will have a bootloader. The updated code would look like this:

```
void setup() {
  pinMode(13, OUTPUT);
}

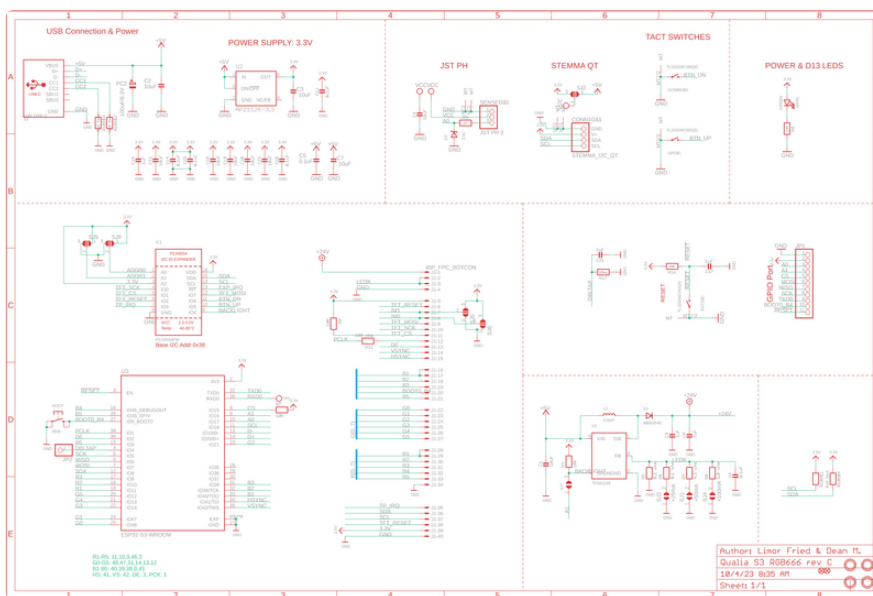
void loop() {
  digitalWrite(13, HIGH);
  delay(1000);
  digitalWrite(13, LOW);
  delay(1000);
}
```

Alternatively, you could load a different sketch. It doesn't matter which sketch you use.

Downloads

- [ESP32-S3 product page with resources \(\)](#)
- [ESP32-S3 datasheet \(\)](#)
- [ESP32-S3 Technical Reference \(\)](#)
- [ST7701 datasheet \(\)](#)
- [NV3052C datasheet \(\)](#)
- [3D models on GitHub \(\)](#)
- [Qualia ESP32-S3 RGB-666 EagleCAD PCB files on GitHub \(\)](#)
- [Qualia ESP32-S3 RGB-666 Fritzing object in the Adafruit Fritzing Library \(\)](#)

Schematic



Fab Print

