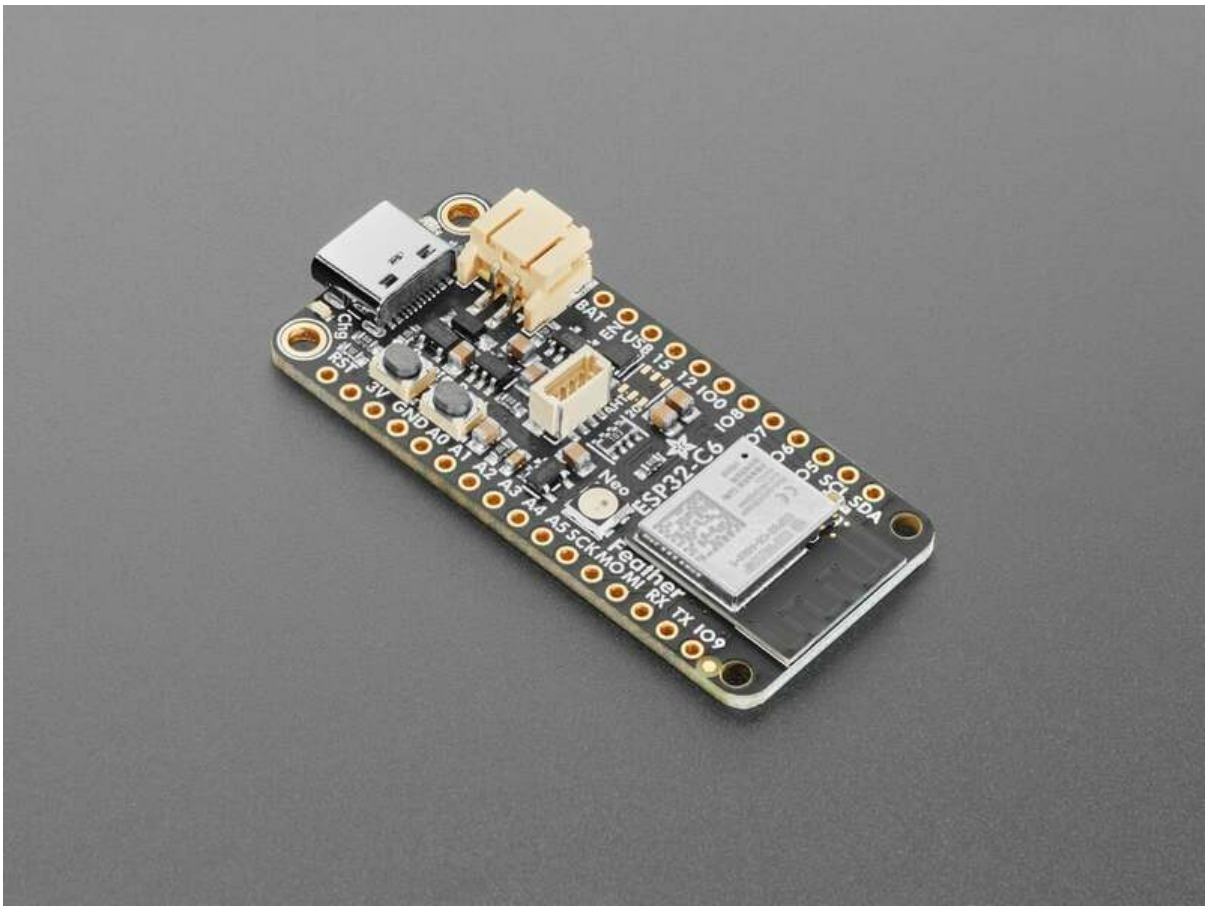
Adafruit ESP32-C6 Feather

Created by Liz Clark

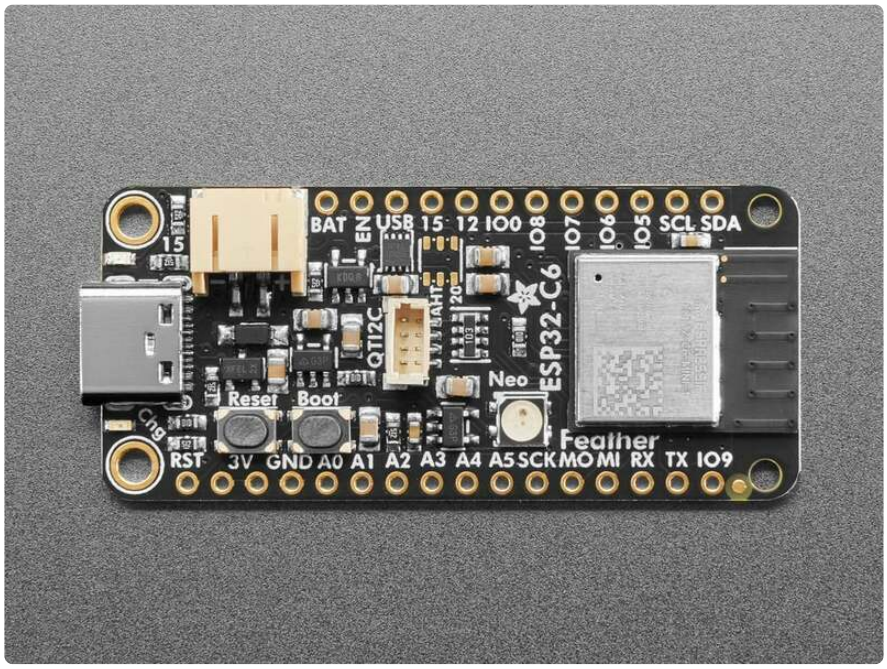


https://learn.adafruit.com/adafruit-esp32-c6-feather

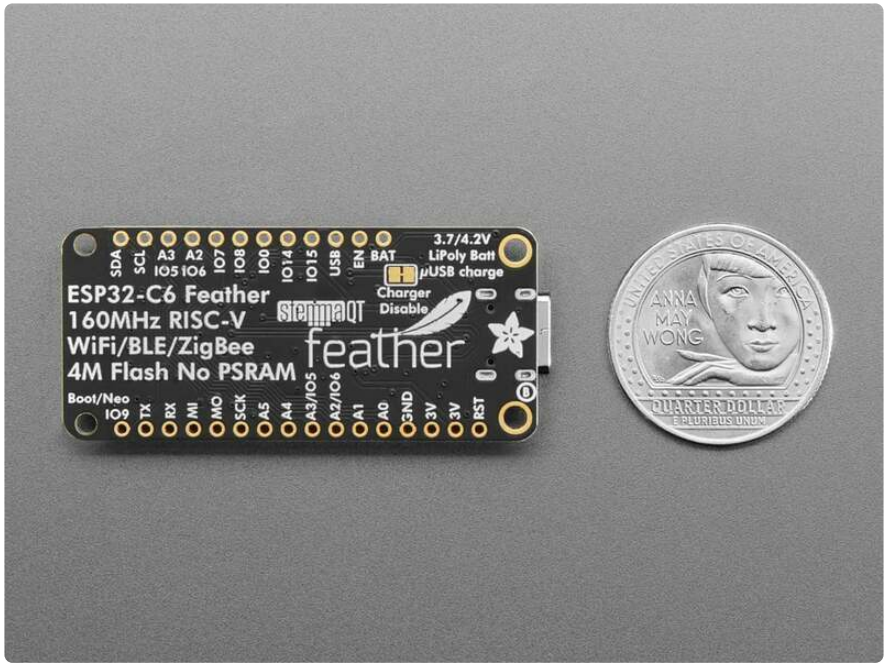
Last updated on 2025-02-18 12:59:54 PM EST

Table of Contents

Overview



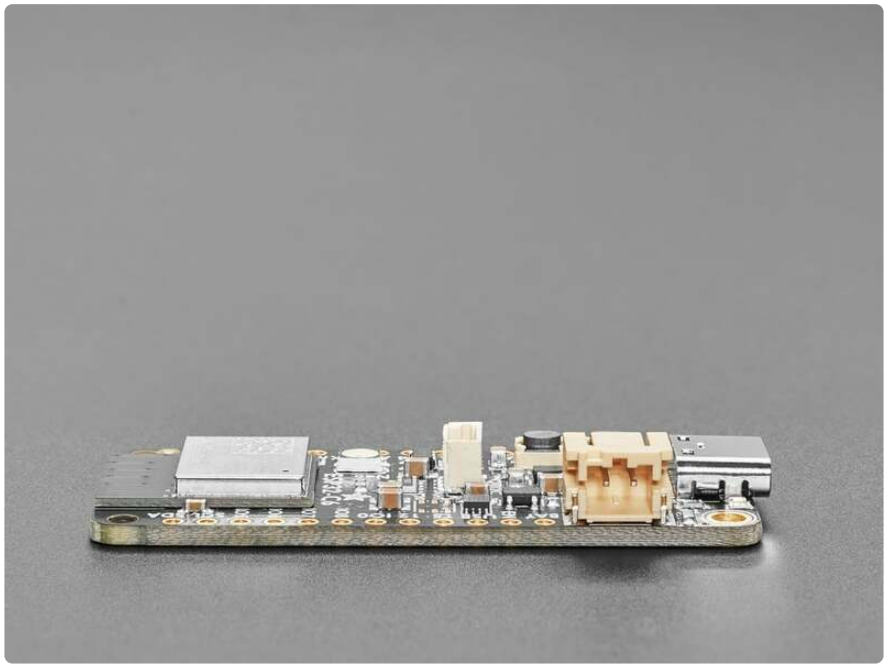
The ESP32-C6 is Espressif's first WiFi 6 SoC integrating 2.4 GHz WiFi 6, Bluetooth 5 (LE) and the 802.15.4 protocol. It brings the goodness you know from the low-cost C3 series (http://adafru.it/5337) and improves it with Zigbee/802.15.4 at 2.4 Ghz. That means it could make for great Matter development hardware (https://adafru.it/1a6C)!



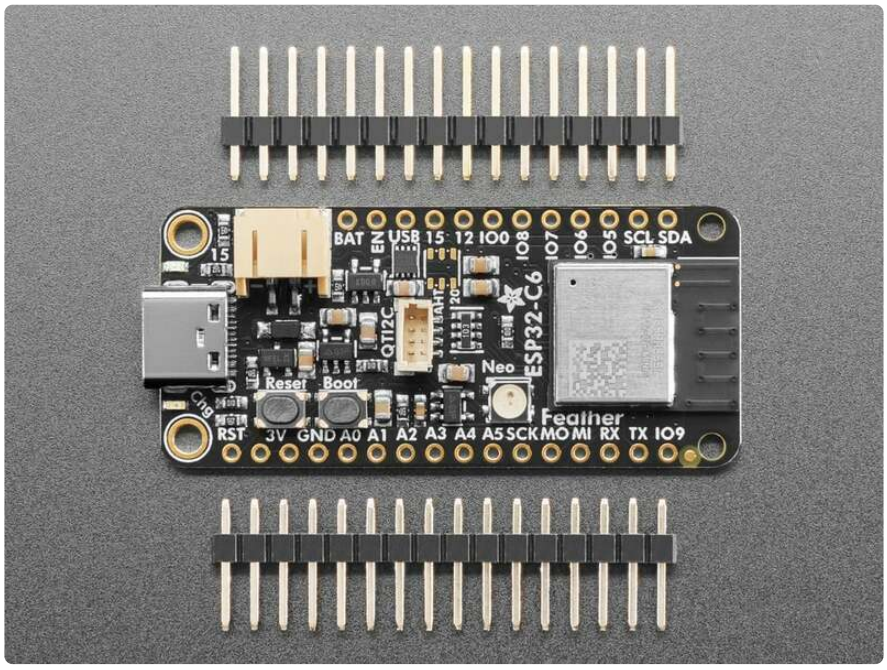
We took our Feather ESP32-S2 (http://adafru.it/5000) and swapped out the 'S2 for a C6. Plus some re-routing and here's what we've got: a C6 Feather with lots of GPIO, lipoly charging and monitoring with the MAX17048 (http://adafru.it/5580), NeoPixel,

I2C Stemma QT port, and a second low-quiescent LDO for disabling the I2C and NeoPixel when we want ultra-low power usage - as low as 17uA in deep sleep.

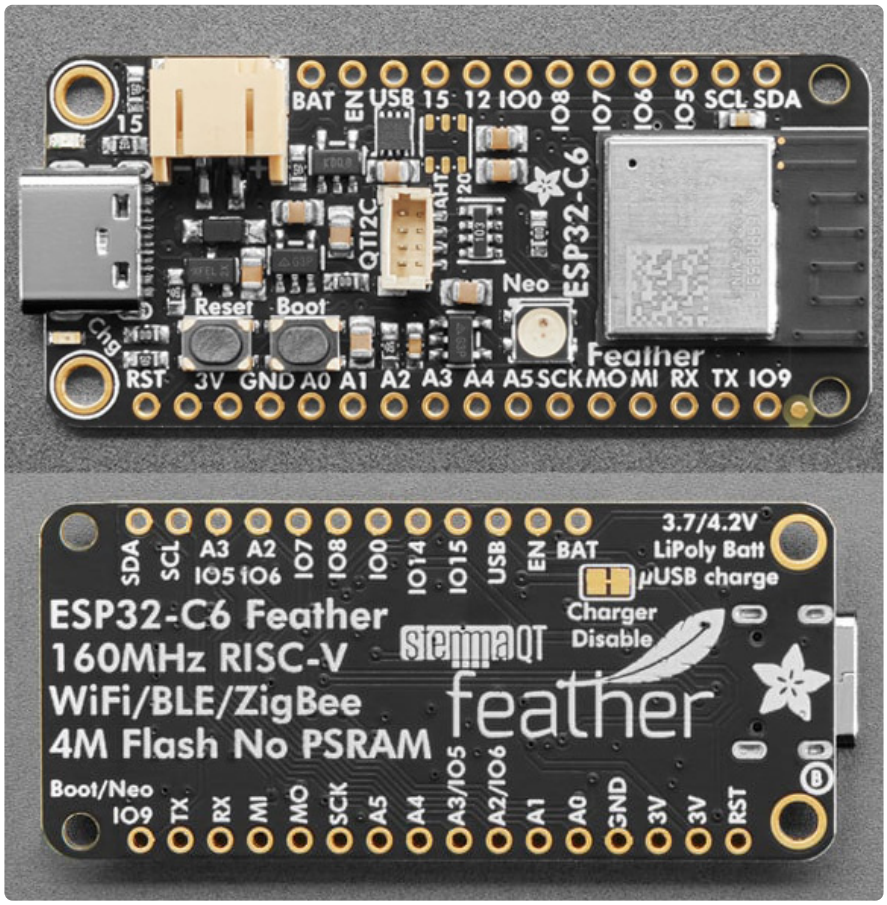
One thing to watch for is that, like the C3, the C6 does not have native USB. It does have a 'built in' USB Serial core that can be used for debugging, but it cannot act like a mouse, keyboard, or disk drive. That means if you are running CircuitPython (https://adafru.it/1a6l) you will need to use WiFi, Bluetooth or WebSerial for transferring files back and forth rather than drag-and-dropping to a drive. Ditto for the bootloader side, this chip cannot run UF2.

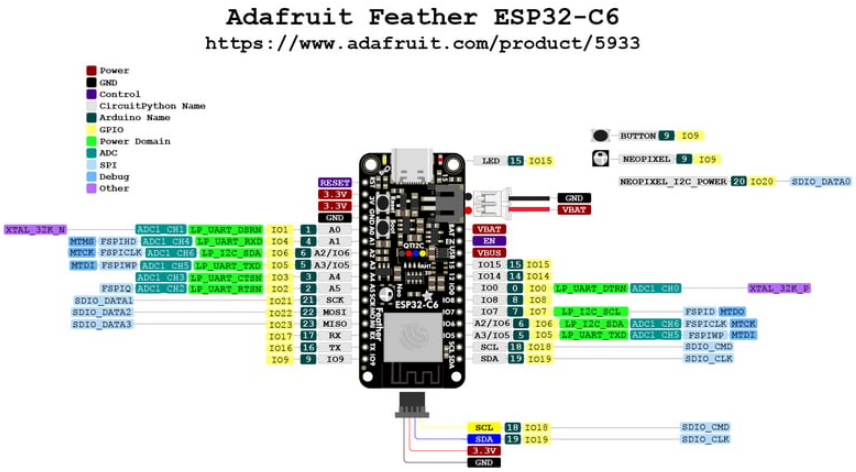


Another thing to be aware of is the ESP32-C6 does not have as many GPIO as the ESP32-S2 or ESP32-S3, so **A2** is the same GPIO pin as **IO6** and **A3** is the same pin as **IO5**. However, this gives it the most compatibility with our existing FeatherWings.



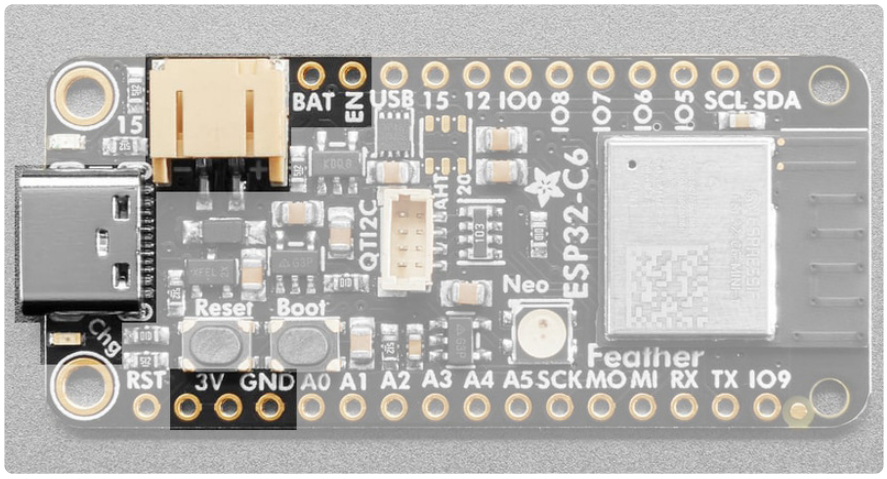
Pinouts





Link to PrettyPins PDF on GitHub (https://adafru.it/1a6D).

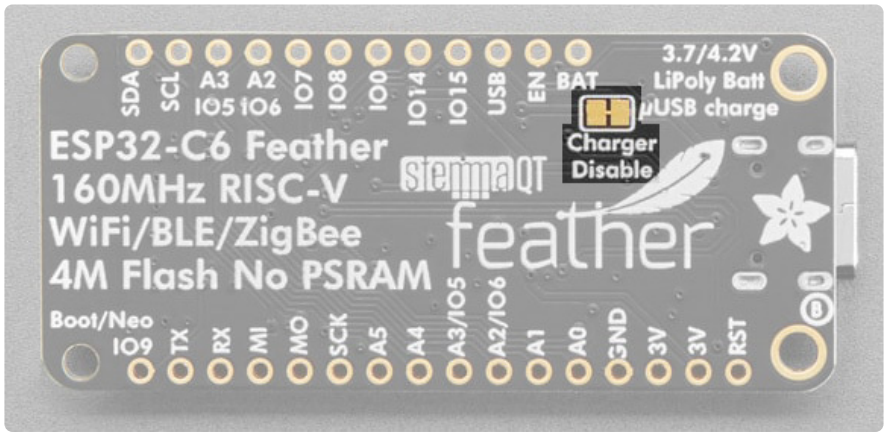
Power



There are two ways you can power the ESP32-C6 Feather, as well as other related pins.

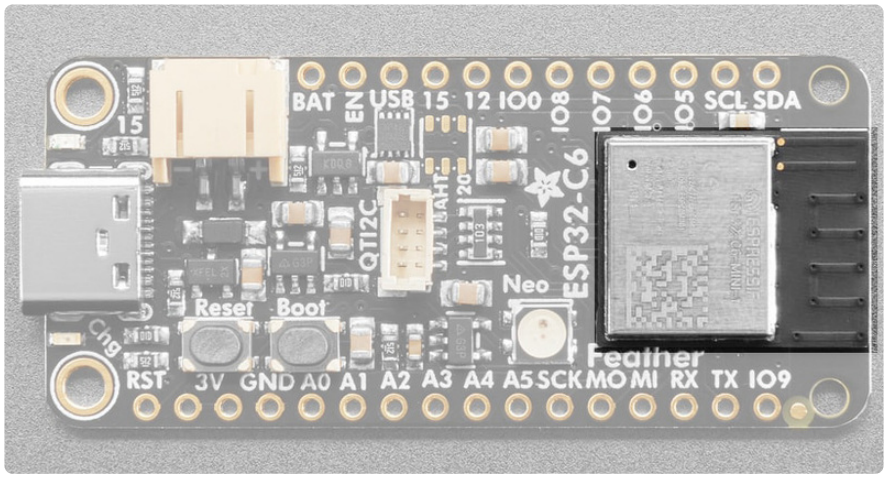
- **USB-C port** - This is used for both powering and programming the board. You can power it with any USB C cable. When USB is plugged in it will charge the Lipoly battery.
- **LiPoly connector/charger** - You can plug in any 250mAh or larger 3.7/4.2V Lipoly battery into this **JST 2-PH port** to both power your Feather and charge the battery. The battery will charge from the USB power when USB is plugged in. If the battery is plugged in and USB is plugged in, the Feather will power itself from USB and it will charge the battery up.
- **Chg LED** - When the battery is charging, the yellow CHG LED will be lit. When charging is complete, the LED will turn off. If there's no battery plugged in, the CHD LED may blink rapidly - this is expected!

- **GND** - This is the common ground for all power and logic.
- **BAT** - This is the positive voltage to/from the 2-pin JST jack for the optional Lipoly battery.
- **USB** - This is the positive voltage to/from the USB C jack, if USB is connected.
- **EN** - This is the 3.3V regulator's enable pin. It's pulled up, so connect to ground to disable the 3.3V regulator.
- **3.3V** - These pins are the output from the 3.3V regulator, they can supply 500mA peak.



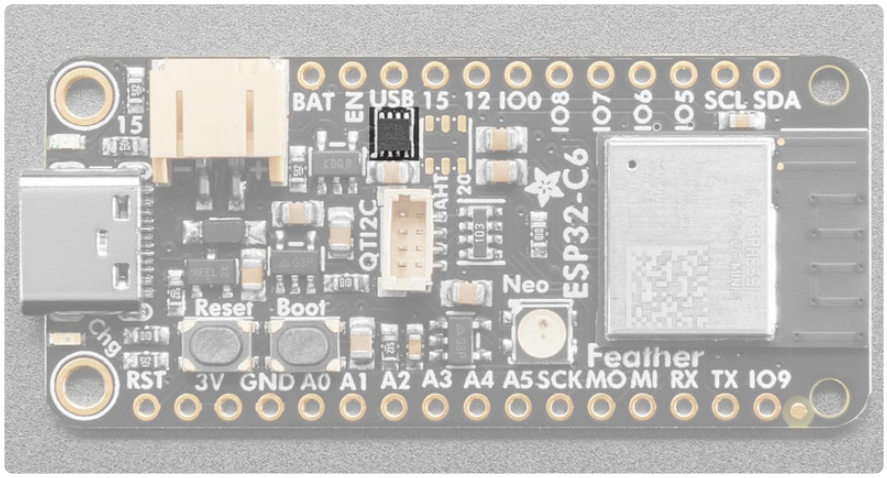
On the back of the Feather is a jumper labeled **Charger Disable**. If you cut this jumper, you'll disable the LiPoly battery charging circuit. This means that you could use the JST-PH battery port with non-LiPoly batteries, such as AA or AAA battery packs.

ESP32-C6 WiFi Module



This is the **ESP32-C6 module**. It is a 32-bit RISC-V single-core processor that operates at up to 160 MHz. This version of the module has 4MB of flash and no PSRAM. It supports WiFi 6 in a 2.4GHz band, Bluetooth 5, Zigbee 3.0, and Thread. It's pin-to-pin compatible with the ESP32-WROOM series module. With low-power consumption, it is an ideal choice for a variety of IoT projects!

MAX17048 Battery Monitor

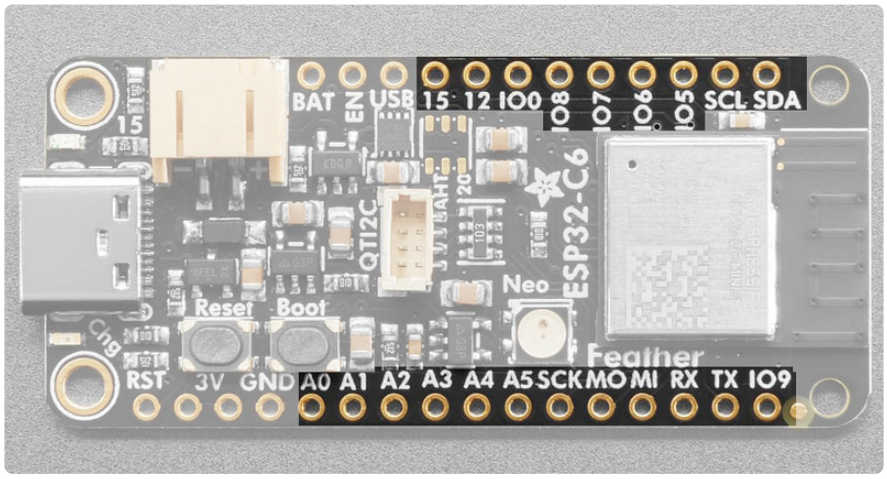


The **Adafruit MAX17048 LiPoly / LiIon Fuel Gauge and Battery Monitor** reports the voltage and charge percent over I2C. Connect it to your Lipoly or LiIon battery (https://adafru.it/NdY) and it will let you know the voltage of the cell, and it does the annoying math of decoding the non-linear voltage to get you a valid percentage as well!

The battery monitor is available over I2C on address **0x36**.

Our Arduino (https://adafru.it/18f1) or CircuitPython/Python (https://adafru.it/10RA) library code allows you to read the voltage and percentage whenever you like. There is no pin on the ESP32-C6 Feather that returns battery voltage, but this I2C monitor makes it super simple to get that data!

Logic Pins



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

There are three sets of shared pins on the Feather: Boot/NeoPixel/IO9, A2/IO6 and A3/IO5. Be careful when using these pins.

There are six analog pins:

- **A0 thru A5** can also be analog inputs.

> The ESP32-C6 Feather does not have a DAC, so you cannot do true analog out.

ESP32 chips allow for 'multiplexing' of almost all signals. There is support for SPI, UART, I2C, I2S, RMT, TWAI, and PWM on any pin. The Feather has a few specially designated pins for pin compatibility with FeatherWings and preexisting code:

The SPI pins:

- **SCK** - This is the SPI clock pin (IO21).
- **MOSI** - This is the SPI **M**icrocontroller **O**ut / **S**ensor **I**n pin (IO22).
- **MISO** - This is the SPI **M**icrocontroller **I**n / **S**ensor **O**ut pin (IO23).

The UART interface:

- **RX** - This is the UART receive pin (IO17). Connect to TX (transmit) pin on your sensor or breakout.
- **TX** - This is the UART transmit pin (IO16). Connect to RX (receive) pin on your sensor or breakout.

The I2C interface. This is shared by the STEMMA QT connector.

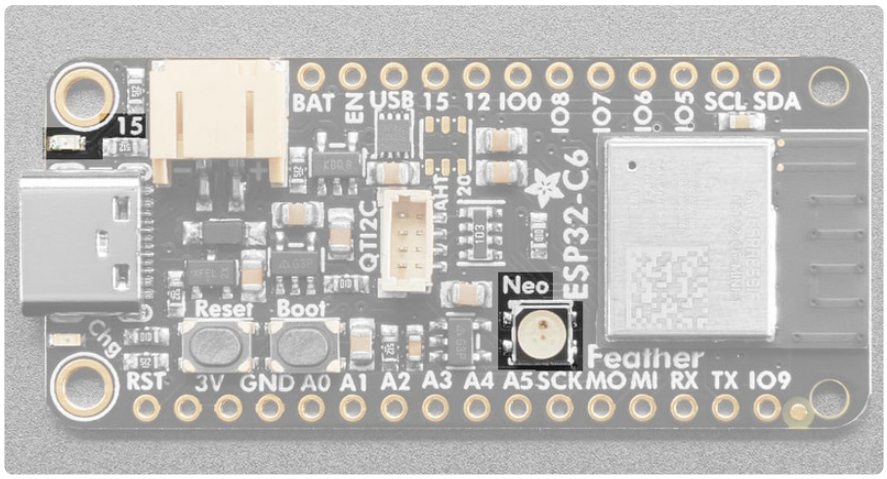
- **SCL** - This is the I2C clock pin (IO18). There is a 5k pullup on this pin.
- **SDA** - This is the I2C data pin (IO19). There is a 5k pullup on this pin.
- In CircuitPython, you can use the STEMMA connector with `board.SCL` and `board.SDA`, or `board.STEMMA_I2C()`. In Arduino, you can access these pins with `Wire`.
- There is an I2C power pin (IO20) that needs to be pulled high for the STEMMA QT connector to work properly. **CircuitPython and Arduino do this automatically.** It is available in CircuitPython and Arduino as `NEOPIXEL_I2C_POWER`. This pin also controls the NeoPixel power.

The digital pins:

- **IO0, IO5-IO9, IO12, IO15** - These are digital pins. IO9 is shared with the Boot button and onboard NeoPixel, IO6 is shared with A2 and IO5 is shared with A3.

Check the ESP32-C6 datasheet or the PrettyPins diagram above for more details on each pin if you need them!

NeoPixel and Red LED

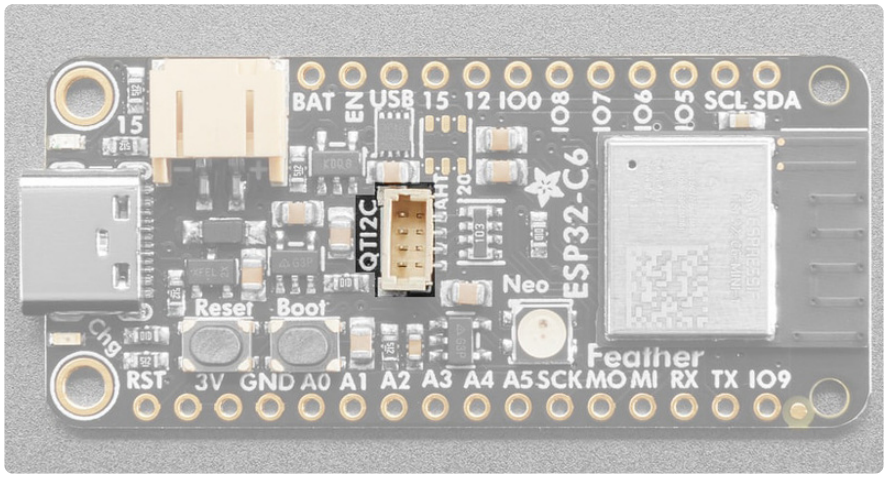


There are two LEDs you can control in code.

- **NeoPixel LED** - This addressable RGB NeoPixel LED, labeled **Neo** on the board, can be controlled with code. It does not act as a status LED in CircuitPython because it shares a pin with the **Boot** button (**IO9**). It is available in CircuitPython as `board.NEOPIXEL`, and in Arduino as `PIN_NEOPIXEL`.
- There is a NeoPixel power pin that needs to be pulled high for the NeoPixel to work. It is the same pin as the I2C power pin (IO20). **This is done automatically by CircuitPython and Arduino.** It is available in CircuitPython and Arduino as `NEOPIXEL_I2C_POWER`.
- **Red LED** - This little red LED, labeled **15** on the board, is on or blinks during certain operations (such as pulsing when in the bootloader), and is controllable in code. It is available in CircuitPython as `board.LED`, and in Arduino as `LED_BUILTIN` or `15`.

The NeoPixel and Boot button are both connected to IO9.

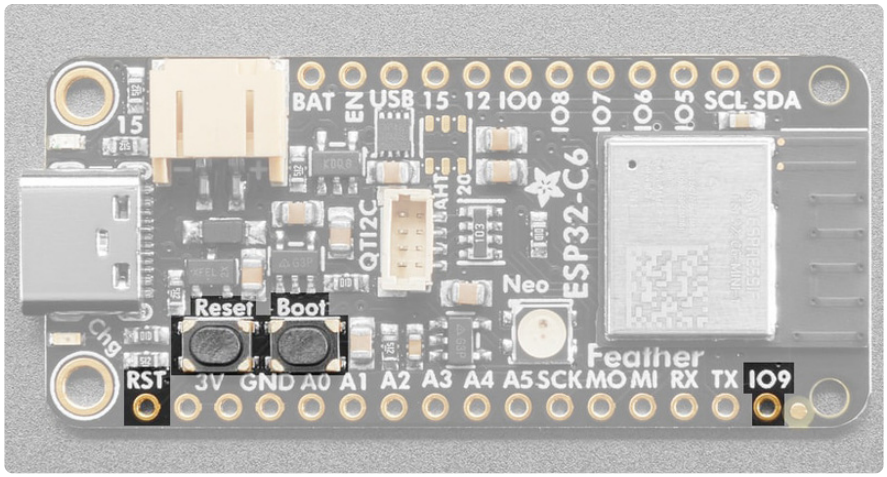
STEMMA QT



This **JST SH 4-pin** STEMMA QT **(https://adafru.it/Ft4) connector** breaks out I2C (SCL, SDA, 3.3V, GND). It allows you to connect to various breakouts and sensors with **STEMMA QT** connectors (https://adafru.it/Qgf) or to other things using assorted associated accessories (https://adafru.it/Ft6). It works great with any STEMMA QT or Qwiic sensor/device. You can also use it with Grove I2C devices thanks to this handy cable (http://adafru.it/4528).

There is a power pin (**IO20**) that must be pulled high for the STEMMA QT connector to work. **This is done automatically in CircuitPython and Arduino.** The pin is available in CircuitPython and Arduino as `NEOPIXEL_I2C_POWER`. You can manually cut power to the QT port completely by setting this pin to an output and low. This will disable power to the connector for low power usage.

Buttons



There are two buttons on the ESP32-C6 Feather.

- **Reset button** - This button restarts the board and helps enter the bootloader. You can click it once to reset the board without unplugging the USB cable or battery.
- The **RST pin** is can be used to reset the board. Tie to ground manually to reset the board.
- **Boot button** - This button can be read as an input in code. It is connected to pin **IO9**, which is also broken out separately on the Feather. It is available as `board.BUTTON` in CircuitPython, and pin `9` in Arduino. Simply set it to be an input with a pullup. **This pin is shared with the onboard NeoPixel.** This button can also be used to put the board into ROM bootloader mode. To enter ROM bootloader mode, hold down DFU button while clicking reset button mentioned above. When in the ROM bootloader, you can upload code and query the chip using `esptool`.

Low Power Use

This microcontroller board can be used for low power usage thanks to the ESP32's multiple sleep modes.

There are three basic operating states to Espressif chips: normal, light sleep and deep sleep.

Normal power usage is as you expect: you can use the chip and run code as you like - connecting to WiFi, reading sensors, etc.

Light sleep is sort of a 'hibernation' - power usage is minimal and WiFi is disconnected, but the internal clock and memory is kept. That means you can wake up where you left off, in the middle of the code as desired. You'll still need to re-initialize any external hardware that got disconnected, and WiFi, but it's often faster than waking from a deep sleep

Deep sleep is the lowest power but the tradeoff is that all memory and state is lost - the only thing that's running is the real time clock that can wake the chip up. When woken up, the chip starts as if it was physically reset - from the beginning of the code. This can be beneficial if you want to have a fresh start each time

A rough guideline is:

- Normal power: 100mA+ can be as much power as need and spike during WiFi connection

- Light sleep: 2mA assuming all external hardware is de-powered
- Deep sleep: 100uA assuming all external hardware is de-powered

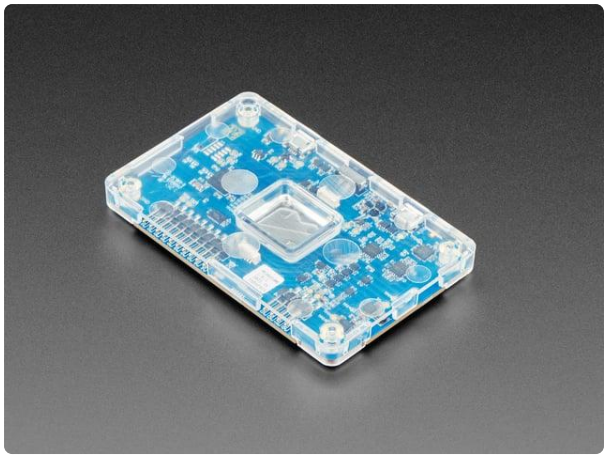
The Adafruit ESP32-C6 Feather has a `NEOPIXEL_I2C_POWER` pin that controls power to I2C and the NeoPixel LED. This pin is automatically pulled HIGH in both CircuitPython and Arduino. Disabling this pin by setting it to an output and LOW allows you to drop the power draw, even when you have I2C sensors or breakouts connected.

Here's a generic sketch we use for all our boards that has a macro-defined section for enabling and disabling all external powered elements. For example, if there's a power pin for NeoPixels, I2C port, TFT, etc...we turn that off before going into light or deep sleep! This will minimize power usage

```
// SPDX-FileCopyrightText: 2022 Limor Fried for Adafruit Industries
//
// SPDX-License-Identifier: MIT

#include <Adafruit_NeoPixel.h>

// While we wait for Feather ESP32 V2 to get added to the Espressif BSP,
// we have to select PICO D4 and UNCOMMENT this line!
//#define ADAFRUIT_FEATHER_ESP32_V2

// then these pins will be defined for us
#if defined(ADAFRUIT_FEATHER_ESP32_V2) or defined(ARDUINO_ADAFRUIT_ITSYBITSY_ESP32)
#define PIN_NEOPIXEL 0
#define NEOPIXEL_I2C_POWER 2
#endif


#if defined(PIN_NEOPIXEL)
  Adafruit_NeoPixel pixel(1, PIN_NEOPIXEL, NEO_GRB + NEO_KHZ800);
#endif

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

  // Turn on any internal power switches for TFT, NeoPixels, I2C, etc!
  enableInternalPower();
}

void loop() {
  LEDon();
  delay(1000);

  disableInternalPower();
  LEDoff();
  esp_sleep_enable_timer_wakeup(1000000); // 1 sec
  esp_light_sleep_start();
  // we'll wake from light sleep here

  // wake up 1 second later and then go into deep sleep
  esp_sleep_enable_timer_wakeup(1000000); // 1 sec
  esp_deep_sleep_start();
  // we never reach here
}

void LEDon() {
#if defined(PIN_NEOPIXEL)
```

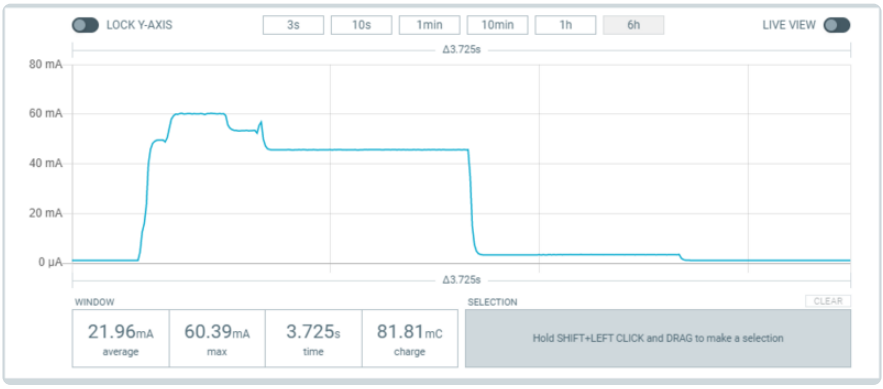
```
  pixel.begin(); // INITIALIZE NeoPixel
  pixel.setBrightness(20); // not so bright
  pixel.setPixelColor(0, 0xFFFFFF);
  pixel.show();
#endif
}

void LEDoff() {
#if defined(PIN_NEOPIXEL)
  pixel.setPixelColor(0, 0x0);
  pixel.show();
#endif
}

void enableInternalPower() {
#if defined(NEOPIXEL_POWER)
  pinMode(NEOPIXEL_POWER, OUTPUT);
  digitalWrite(NEOPIXEL_POWER, HIGH);
#endif

#if defined(NEOPIXEL_I2C_POWER)
  pinMode(NEOPIXEL_I2C_POWER, OUTPUT);
  digitalWrite(NEOPIXEL_I2C_POWER, HIGH);
#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);
  pinMode(NEOPIXEL_POWER, OUTPUT);
  digitalWrite(NEOPIXEL_POWER, HIGH);
#endif
}

void disableInternalPower() {
#if defined(NEOPIXEL_POWER)
  pinMode(NEOPIXEL_POWER, OUTPUT);
  digitalWrite(NEOPIXEL_POWER, LOW);
#endif

#if defined(NEOPIXEL_I2C_POWER)
  pinMode(NEOPIXEL_I2C_POWER, OUTPUT);
  digitalWrite(NEOPIXEL_I2C_POWER, LOW);
#endif

#if defined(ARDUINO_ADAFRUIT_FEATHER_ESP32S2)
  // turn off the I2C power by setting pin to rest state (off)
  pinMode(PIN_I2C_POWER, INPUT);
  pinMode(NEOPIXEL_POWER, OUTPUT);
  digitalWrite(NEOPIXEL_POWER, LOW);
#endif
}
```

The best way to really test power draw is with a specialty power meter such as the Nordic PPK 2



Nordic nRF-PPK2 - Power Profiler Kit II
The Power Profiler Kit II is a standalone unit, which can measure and optionally supply currents all the way from sub-uA and as high as 1A on all Nordic DKs, in...
https://www.adafruit.com/product/5048

When running the above code and monitoring with a PPK, you'll get a graph like this:

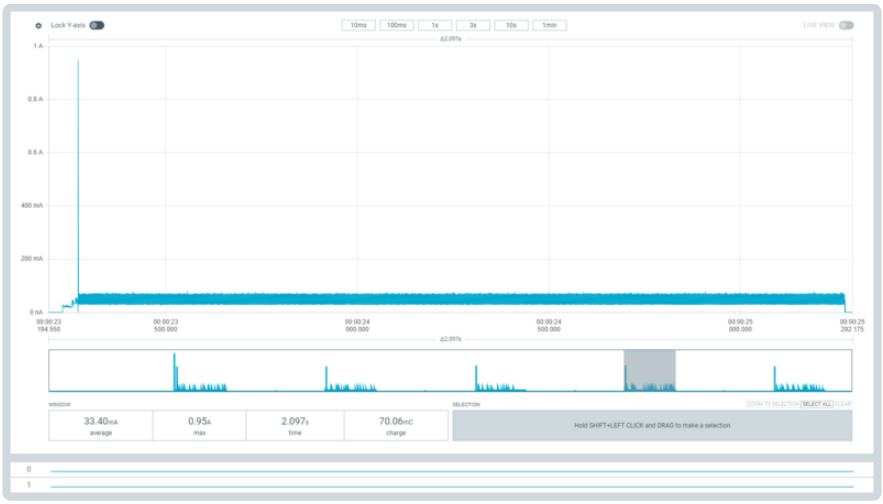


The big pulse is normal mode, you can see the ESP32 booting up, loading code, and then pausing 1 second. Then there's a big drop for one sec to light sleep, and finally one more 1 second pause at deep sleep.

Power Draw for ESP32-C6 Feather

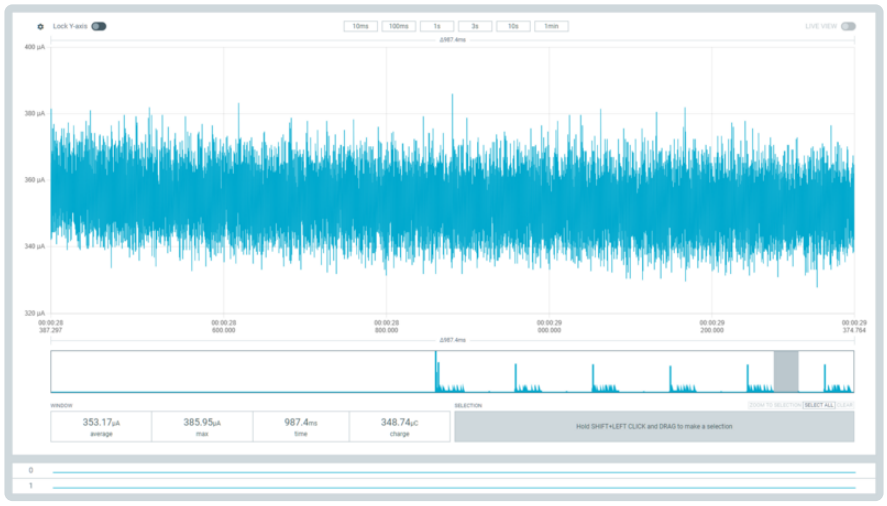
The following graphs show the power draw for the ESP32-C6 Feather V2 in normal power mode, light sleep mode, and deep sleep mode.

Normal Power Mode



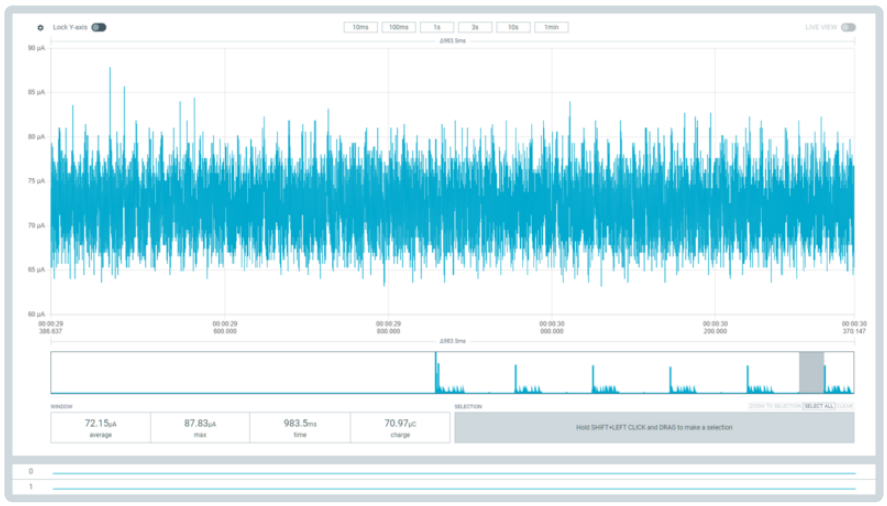
The power draw, running normally (without WiFi), is 33.4mA.

Light Sleep Mode



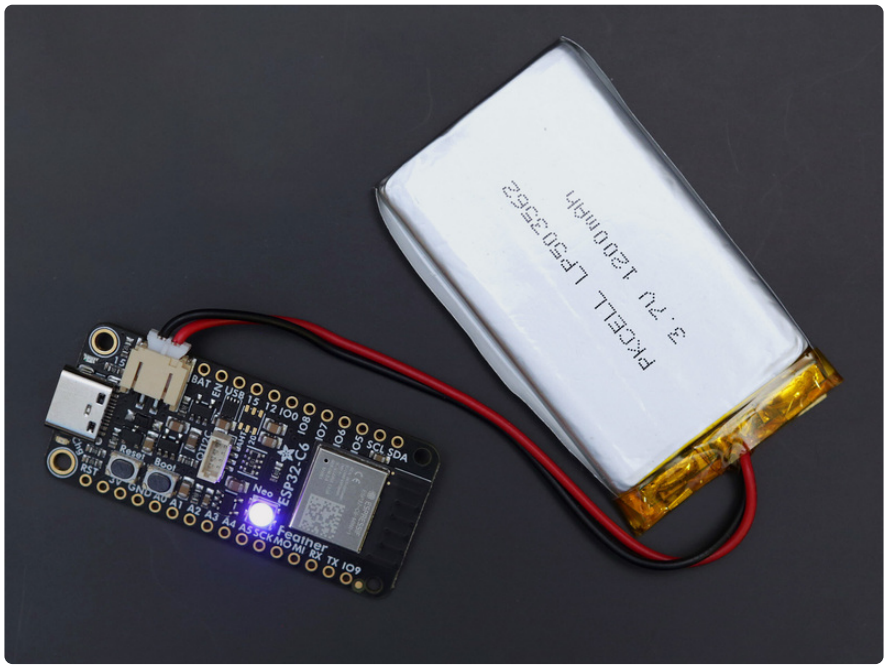
The power draw in light sleep mode is 353.17uA.

Deep Sleep Mode



The power draw in deep sleep mode is 72.15uA.

Power Management



Battery + USB Power

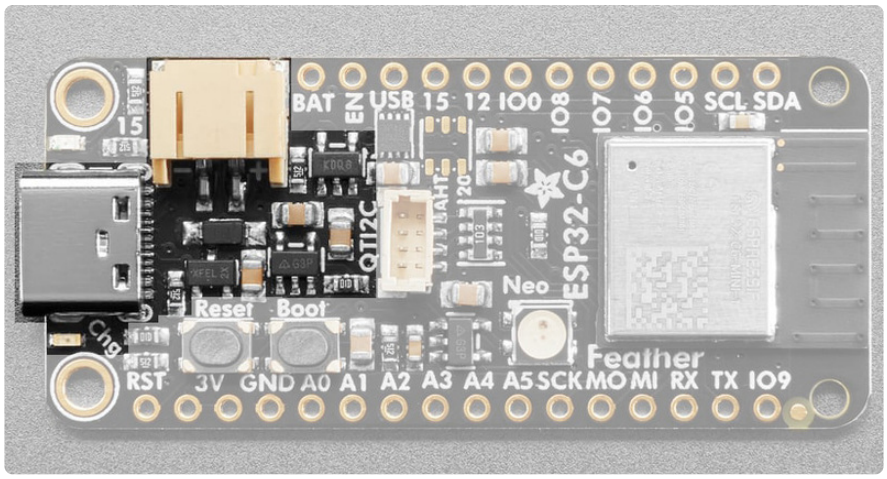
We wanted to make our Feather boards easy to power both when connected to a computer as well as via battery.

There's **two ways to power** a Feather:

1. You can connect with a USB cable (just plug into the jack) and the Feather will regulate the 5V USB down to 3.3V.
2. You can also connect a 4.2/3.7V Lithium Polymer (LiPo/LiPoly) or Lithium Ion (LiIon) battery to the JST jack. This will let the Feather run on a rechargeable battery.

When the USB power is powered, it will automatically switch over to USB for power, as well as start charging the battery (if attached). This happens 'hot-swap' style so you can always keep the LiPoly connected as a 'backup' power that will only get used when USB power is lost.

The JST connector polarity is matched to Adafruit LiPoly batteries. Using wrong polarity batteries can destroy your Feather. Many customers try to save money by purchasing Lipoly batteries from Amazon only to find that they plug them in and the Feather is destroyed!



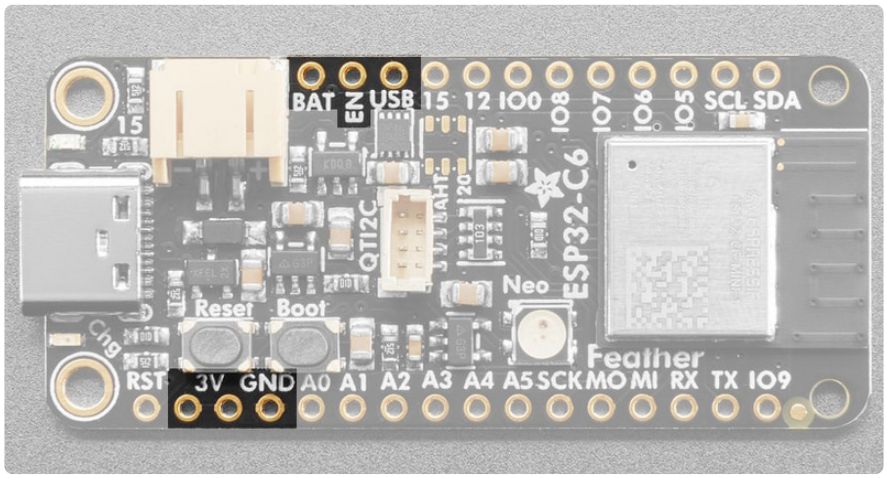
The above shows the USB C jack (left), LiPoly JST jack (top left), as well as the 3.3V regulator and changeover diode (just to the right of the JST jack) and the LiPoly charging circuitry (above the Reset button).

There's also a **CHG** LED next to the USB jack, which will light up while the battery is charging. This LED might also flicker if the battery is not connected, it's normal.

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

Power Supplies

You have a lot of power supply options here! We bring out the **BAT** pin, which is tied to the LiPoly JST connector, as well as **USB** which is the +5V from USB if connected. We also have the **3V** pin which has the output from the 3.3V regulator. We use a 500mA peak regulator. While you can get 500mA from it, you can't do it continuously from 5V as it will overheat the regulator.



Measuring Battery

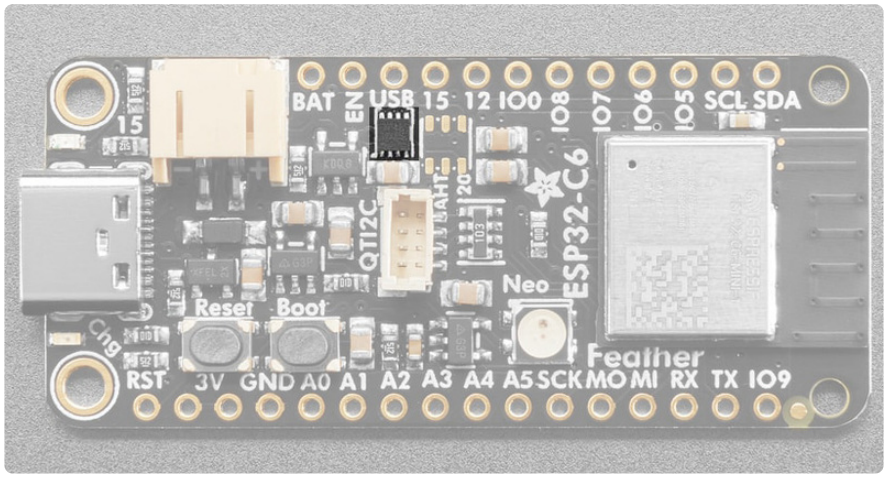
If you're running off of a battery, chances are you wanna know what the voltage is at! That way you can tell when the battery needs recharging. LiPoly batteries are 'maxed out' at 4.2V and stick around 3.7V for much of the battery life, then slowly sink down to 3.2V or so before the protection circuitry cuts it off. By measuring the voltage you can quickly tell when you're heading below 3.7V.

This board includes an **MAX17048 Battery Monitor** that reports the voltage and charge percent over I2C.

The MAX17048 battery monitor is available over I2C on address **0x36**.

Our Arduino MAX1704x (https://adafru.it/18f1) or CircuitPython/Python MAX1704x (https://adafru.it/10RA) library code allows you to read the voltage and percentage whenever you like.

There is no pin on this board that returns battery voltage, but this I2C monitor makes it super simple to get that data!



In Arduino, you can measure the battery voltage using the following script.

```
// SPDX-FileCopyrightText: 2023 Liz Clark for Adafruit Industries
//
// SPDX-License-Identifier: MIT
//
// Adafruit Battery Monitor Demo
// Checks for MAX17048 or LC709203F

#include <Wire.h>
#include "Adafruit_MAX1704X.h"
#include "Adafruit_LC709203F.h"

Adafruit_MAX17048 maxlipo;
Adafruit_LC709203F lc;

// MAX17048 i2c address
bool addr0x36 = true;
```

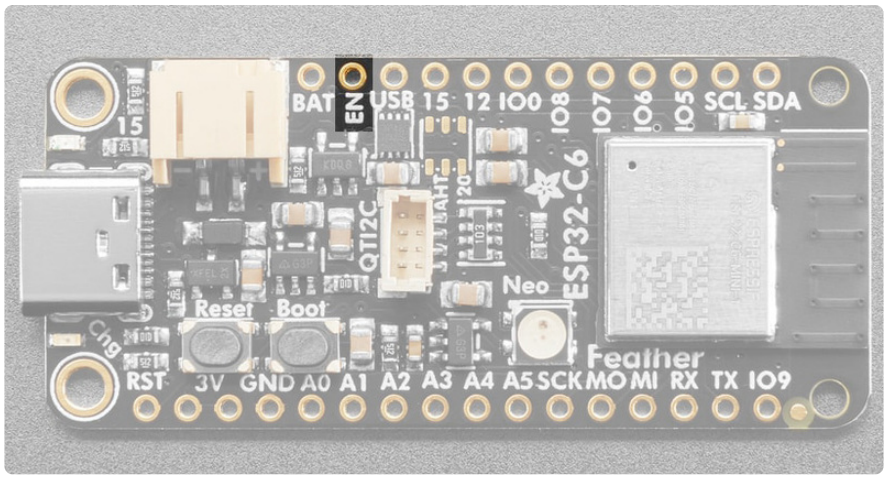
```
void setup() {
  Serial.begin(115200);
  while (!Serial) delay(10);    // wait until serial monitor opens
  Serial.println(F("\nAdafruit Battery Monitor simple demo"));
  // if no max17048..
  if (!maxlipo.begin()) {
    Serial.println(F("Couldnt find Adafruit MAX17048, looking for LC709203F.."));
    // if no lc709203f..
    if (!lc.begin()) {
      Serial.println(F("Couldnt find Adafruit MAX17048 or LC709203F."));
      while (1) delay(10);
    }
    // found lc709203f!
    else {
      addr0x36 = false;
      Serial.println(F("Found LC709203F"));
      Serial.print("Version: 0x"); Serial.println(lc.getICversion(), HEX);
      lc.setThermistorB(3950);
      Serial.print("Thermistor B = "); Serial.println(lc.getThermistorB());
      lc.setPackSize(LC709203F_APA_500MAH);
      lc.setAlarmVoltage(3.8);
    }
  // found max17048!
  }
  else {
    addr0x36 = true;
    Serial.print(F("Found MAX17048"));
    Serial.print(F(" with Chip ID: 0x"));
    Serial.println(maxlipo.getChipID(), HEX);
  }
}

void loop() {
  // if you have the max17048..
  if (addr0x36 == true) {
    max17048();
  }
  // if you have the lc709203f..
  else {
    lc709203f();
  }

  delay(2000);  // dont query too often!

}

void lc709203f() {
  Serial.print("Batt_Voltage:");
  Serial.print(lc.cellVoltage(), 3);
  Serial.print("\t");
  Serial.print("Batt_Percent:");
  Serial.print(lc.cellPercent(), 1);
  Serial.print("\t");
  Serial.print("Batt_Temp:");
  Serial.println(lc.getCellTemperature(), 1);
}

void max17048() {
  Serial.print(F("Batt Voltage: ")); Serial.print(maxlipo.cellVoltage(), 3);
Serial.println(" V");
  Serial.print(F("Batt Percent: ")); Serial.print(maxlipo.cellPercent(), 1);
Serial.println(" %");
  Serial.println();
}
```

For CircuitPython, you can measure it like this.

```
# SPDX-FileCopyrightText: Copyright (c) 2023 Kattni Rembor for Adafruit Industries
#
# SPDX-License-Identifier: Unlicense

import time
import board
from adafruit_max1704x import MAX17048
from adafruit_lc709203f import LC709203F, PackSize

#
i2c = board.I2C()
while not i2c.try_lock():
    pass
i2c_address_list = i2c.scan()
i2c.unlock()

device = None

if 0x0b in i2c_address_list:
    lc709203 = LC709203F(board.I2C())
    # Update to match the mAh of your battery for more accurate readings.
    # Can be MAH100, MAH200, MAH400, MAH500, MAH1000, MAH2000, MAH3000.
    # Choose the closest match. Include "PackSize." before it, as shown.
    lc709203.pack_size = PackSize.MAH400

    device = lc709203
    print("Battery monitor: LC709203")

elif 0x36 in i2c_address_list:
    max17048 = MAX17048(board.I2C())

    device = max17048
    print("Battery monitor: MAX17048")

else:
    raise Exception("Battery monitor not found.")

while device:
    print(f"Battery voltage: {device.cell_voltage:.2f} Volts")
    print(f"Battery percentage: {device.cell_percent:.1f} %")
    print("")
    time.sleep(1)
```

ENable pin

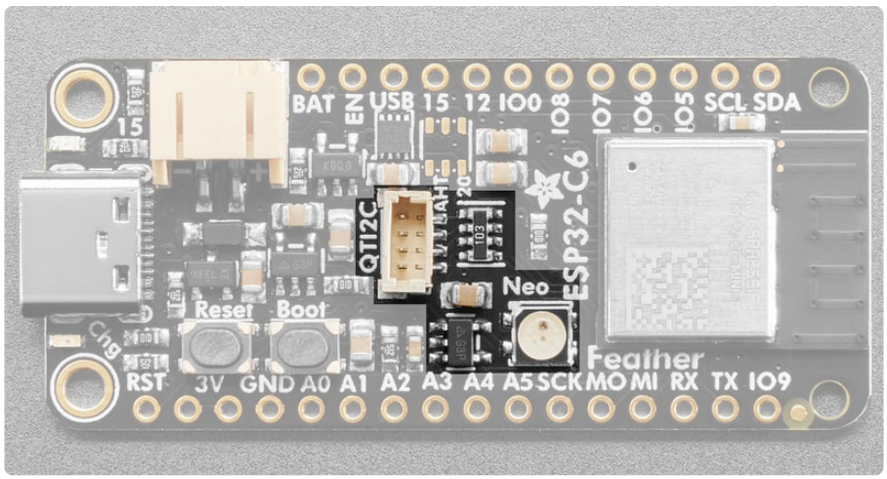
If you'd like to turn off the 3.3V regulator, you can do that with the **EN**(able) pin. Simply tie this pin to **Ground** and it will disable the 3V regulator. The **BAT** and **USB** pins will still be powered.



STEMMA QT and NeoPixel Power

The ESP32-C6 Feather is equipped with a STEMMA QT port and NeoPixel which are both connected to their own regulator. Unlike the one controlled by the ENable pin, this is controlled by GPIO. They are enabled by default in CircuitPython and Arduino. You can disable it manually for low power usage. This pin is available in CircuitPython as `I2C_POWER` and in Arduino as `I2C_NEOPIXEL_POWER`.

If you run into I2C or NeoPixel power issues on Arduino, ensure you are using the latest Espressif board support package. If you are still having issues, you may need to manually pull the pin high in your code.



Alternative Power Options

The two primary ways for powering a feather are a 3.7/4.2V LiPo battery plugged into the JST port or a USB power cable.

If you need other ways to power the Feather, here's what we recommend:

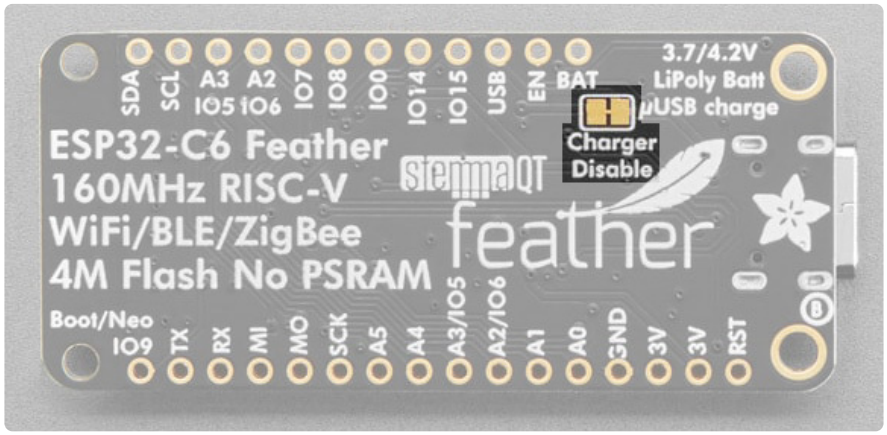
- For permanent installations, a 5V 1A USB wall adapter (http://adafru.it/501) will let you plug in a USB cable for reliable power
- For mobile use, where you don't want a LiPoly, use a USB battery pack! (http://adafru.it/1959)
- If you have a higher voltage power supply, use a 5V buck converter (https://adafru.it/DHs) and wire it to a USB cable's 5V and GND input (http://adafru.it/3972)

Here's what you cannot do:

- **Do not use alkaline or NiMH batteries** and connect to the battery port - this will destroy the LiPoly charger
- **Do not use 7.4V RC batteries on the battery port** - this will destroy the board

The Feather is not designed for external power supplies - this is a design decision to make the board compact and low cost. It is not recommended, but technically possible:

- **Connect an external 3.3V power supply to the 3V and GND pins.** Not recommended, this may cause unexpected behavior and the **EN** pin will no longer work. Also this doesn't provide power on **BAT** or **USB** and some Feathers/Wings use those pins for high current usages. You may end up damaging your Feather.
- **Connect an external 5V power supply to the USB and GND pins.** Not recommended, this may cause unexpected behavior when plugging in the USB port because you will be back-powering the USB port, which could confuse or damage your computer.



If you use **alkaline or NiMH batteries** and connect to the battery port, you'll destroy the LiPoly charger, **unless** you cut the **Charger Disable jumper**. This jumper is located on the back of the board. If you cut it, the LiPoly charger will be disabled and allow you to use alkaline or NiMH batteries.

Install CircuitPython

CircuitPython (https://adafru.it/tB7) is a derivative of MicroPython (https://adafru.it/BeZ) designed to simplify experimentation and education on low-cost microcontrollers. It makes it easier than ever to get prototyping by requiring no upfront desktop software downloads. ESP32 CircuitPython firmware is uploaded to the board via the USB serial port.

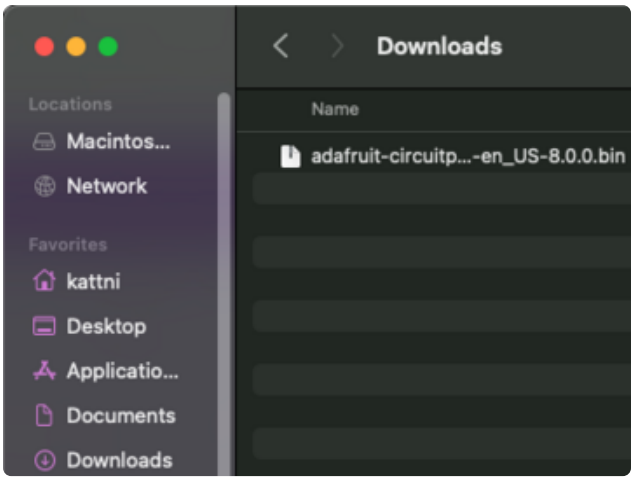
Follow this step-by-step to get CircuitPython running on your board.

CircuitPython Download

Currently we recommend users use an absolute newest version of CircuitPython with the ESP32-C6 Feather. To download this **.BIN** file, click the button below.

adafruit_feather_esp32c6_4mbflash_BIN

https://adafru.it/1a6E

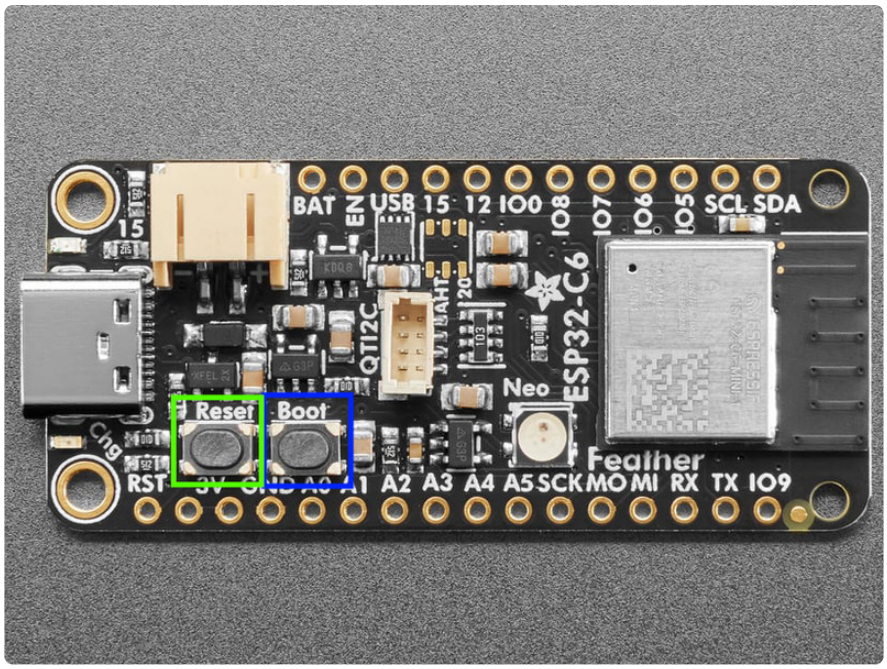


Click the link above to download the latest CircuitPython .bin file.

Save it wherever is convenient for you.

Bootloader Mode

Before connecting to the Web Flasher, you need to put the board into bootloader mode. To do this, hold down the **Boot** button (highlighted in blue). While continuing to hold down the **Boot** button, press and release the **Reset** button (highlighted in green). Then, release the **Boot** button. Your board is now in bootloader mode.



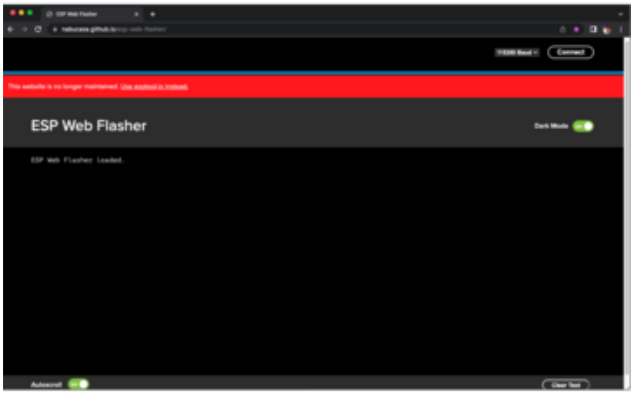
Connecting to the Web Flasher

To begin, **plug your board into your computer via USB, using a known-good data-sync cable**, directly, or via an adapter if needed.

You will have to use the Chrome or a Chromium-based browser to install CircuitPython. For example, Edge and Opera are Chromium based (https://adafru.it/10BL).

Safari and Firefox, etc are not supported - they have not implemented Web Serial (https://adafru.it/10BM)!

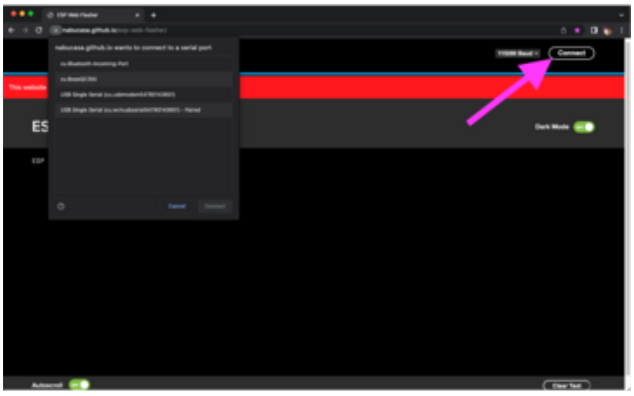
In the **Chrome browser** visit https://adafruit.github.io/Adafruit_WebSerial_ESPTool/ (https://adafru.it/PMB)



The main page of the ESP Web Flasher should look something like this.

Note: The site now displays an alert that it is no longer maintained, and suggests using a different option. The ESP Web Flasher has still proven to be more consistent and easier to use, so it is highly suggested that you continue with this version.

You should remove all other USB devices so only the target board is attached. This eliminates confusion over multiple ports!

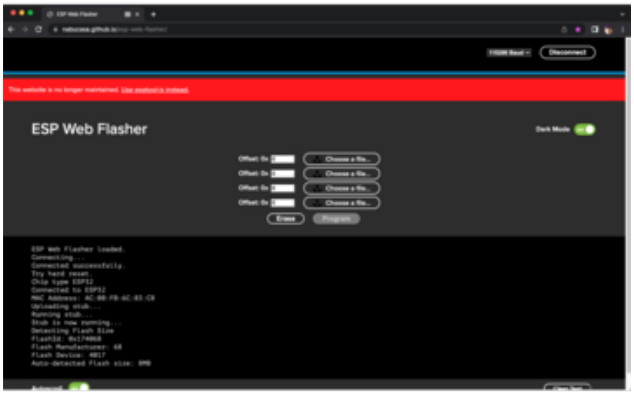


Press the **Connect** button in the top right of the web browser. You will get a pop up asking you to select the COM or Serial port. Look for **USB Single Serial**.

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

```
ESP Web Flasher loaded.
Connecting...
Connected successfully.
Try hard reset.
Chip type ESP32
Connected to ESP32
MAC Address: AC:0B:FB:6C:83:C8
Uploading stub...
Running stub...
Stub is now running...
Detecting Flash Size
FlashId: 0x174068
Flash Manufacturer: 68
Flash Device: 4017
Auto-detected Flash size: 8MB
```

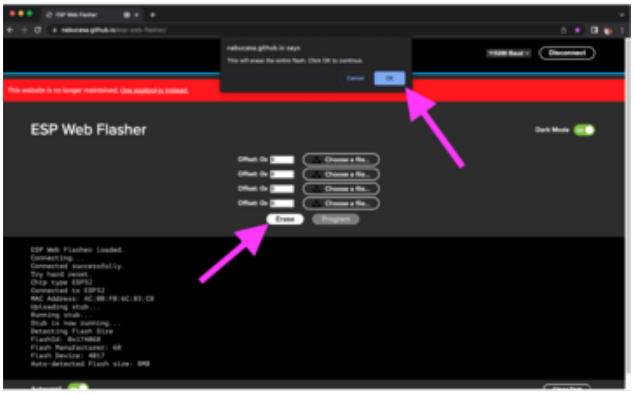
The Javascript code will now try to connect to the board. It may timeout for a bit until it succeeds. On success, you will see that it is **Connected** and will print out a unique **MAC address** identifying the board along with other information that was detected.



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

Erasing the Board Contents

If you would like to erase the entire flash area so that you can start with a clean slate, you can use the erase feature. We recommend doing this every time before installing or updating CircuitPython.



To erase the contents, click the **Erase** button. You will be prompted as to whether you want to continue. Click **OK** to continue. If you do not wish to continue, click Cancel.

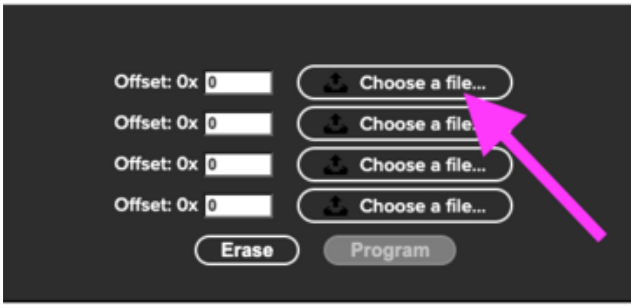
```
MAC Address: AC:0B:FB:6C:83:C8
Uploading stub...
Running stub...
Stub is now running...
Detecting Flash Size
FlashId: 0x174068
Flash Manufacturer: 68
Flash Device: 4017
Auto-detected Flash size: 8MB
Erasing flash memory. Please wait...
Finished. Took 5965ms to erase.
```

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

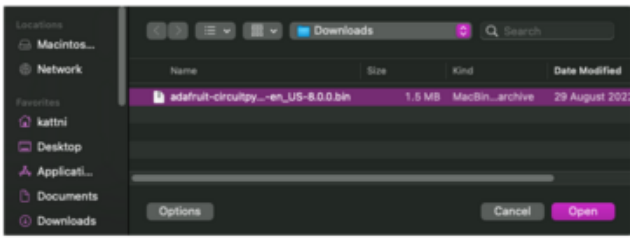
Do not disconnect! Immediately continue on to Programming the Board.

Do not disconnect after erasing! You should immediately continue on to programming your board. If you do not, you may end up with your board in a bad state that makes it more difficult to continue. You can avoid this!

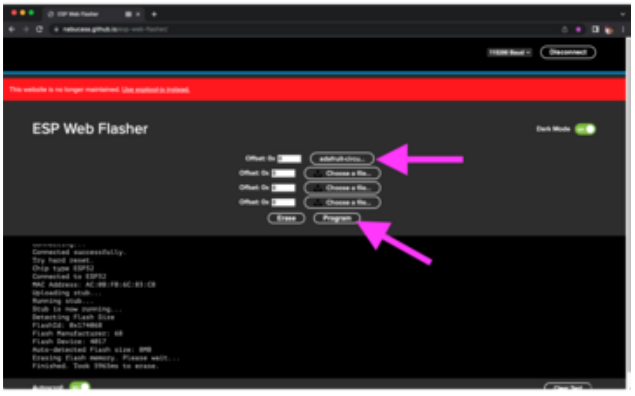
Programming the Board



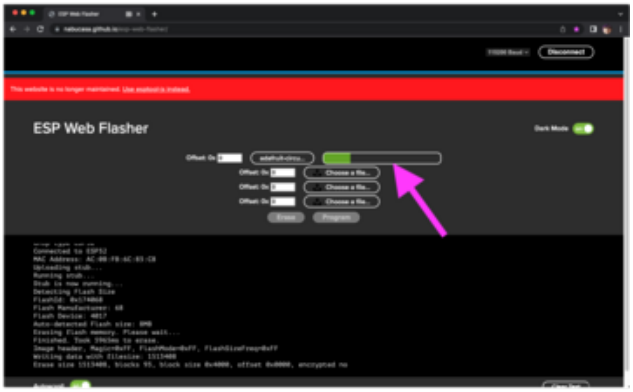
You can click on **Choose a file...** from any of the available buttons. It will only attempt to program buttons with a file and a unique location. Select the **.bin** file you downloaded at the beginning of this page from the file chooser dialogue.



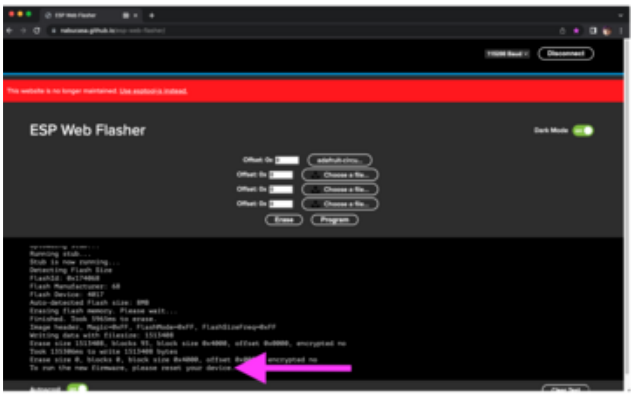
Verify that the **Offset** box next to the file location you used is 0x0. The offset defaults to 0x0, so unless you changed it manually, it should be good to go.



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



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



You've now successfully programmed CircuitPython onto your board! As suggested in the output, press reset to run the new firmware.

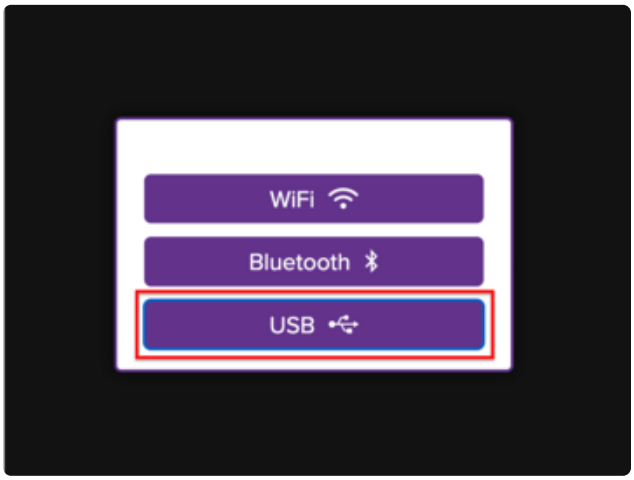
As the ESP32-C6 does not have native USB, no USB drive will show up on your computer when you reset. With CircuitPython firmware loaded, the REPL can be accessed over a serial/COM port.

Don't worry though! We have the CircuitPython USB Workflow Code Editor (https://adafru.it/1a6F) so that you can access the board via USB in your Chromium-based browser.

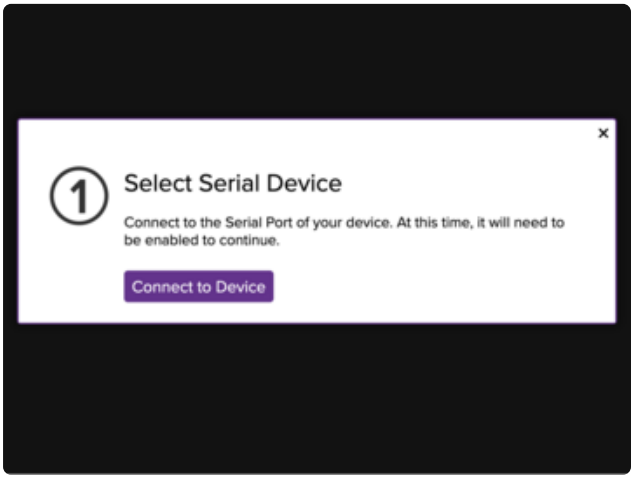
Connecting to the USB Workflow Code Editor

The USB workflow is a new feature and there may be bugs! If you find a bug, please file an issue on GitHub.

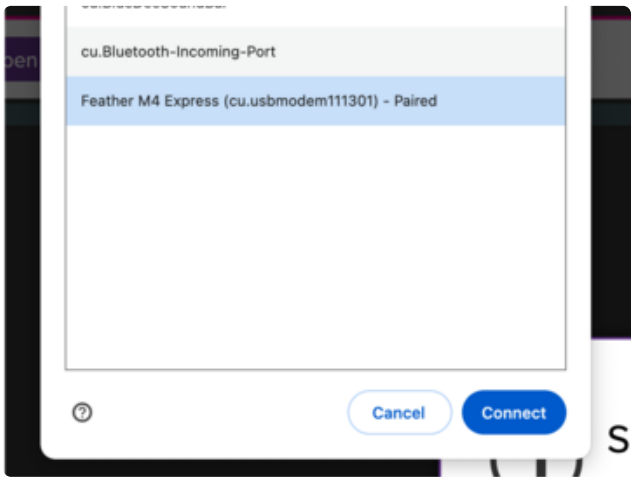
To use the Code Editor, you will need an internet browser such as Google Chrome or Microsoft Edge. It's possible that it may work in other browsers as well, but these have been more thoroughly tested.



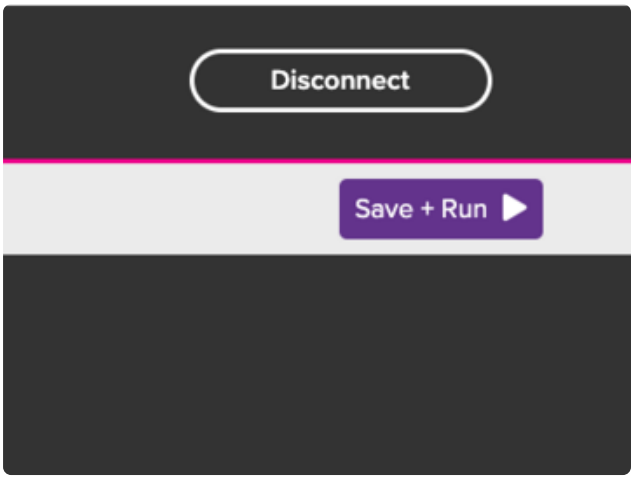
Open your browser and navigate to https://code.circuitpython.org/ (https://adafru.it/10QF). Select **USB** on the dialog prompt that comes up.



This will display a page of instructions along with a **button** to bring up a list of devices to connect to.



Click **Connect to Device** and then **select your board** in the pop-up window. Click **Connect** to connect your board to the editor.

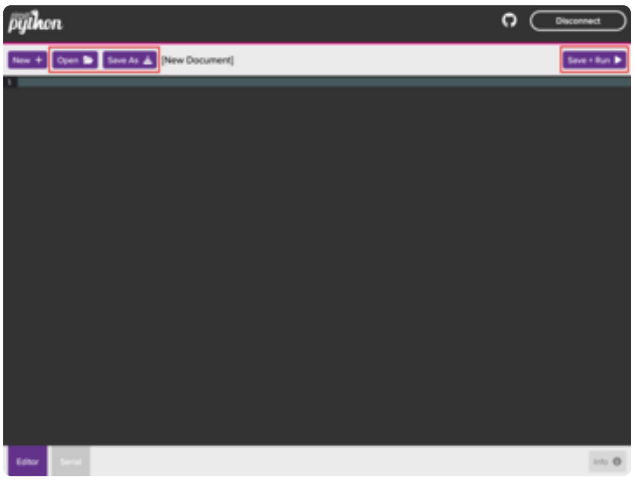


Once you have connected, the **Connect** button in the upper right-hand corner should change to a **Disconnect** button.

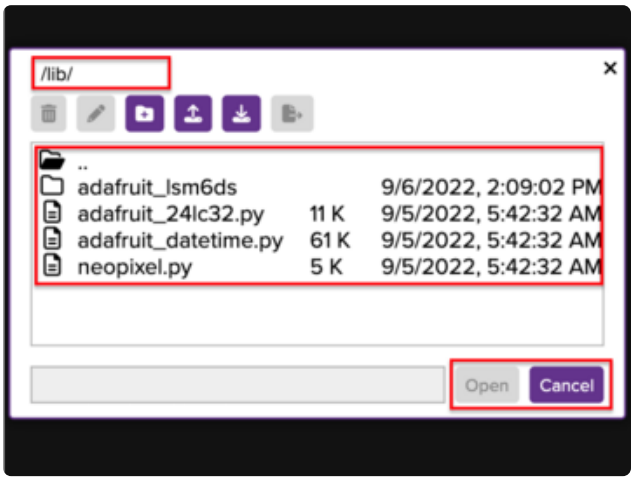
Navigating USB Workflow

Opening and Saving Files

Opening and Saving files is designed to be like to most other applications. Just use the buttons along the top of the editor window.

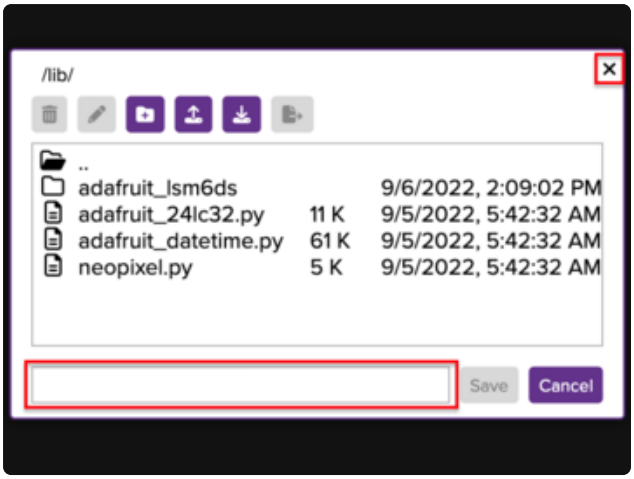


Clicking the **Open** or **Save As** buttons along the top will open the File Dialog. Clicking the **Save + Run** button will save your file and run the code. If your file hasn't been saved yet, this will also bring up the file dialog box.



The file dialog that appears is a simplified dialog that displays the current path at the top, allows you to navigate through the file tree to select the file you would like to open, and has buttons on the bottom to open or save the file you would like to use.

Canceling will tell the editor that you do not want to continue with the current operation.



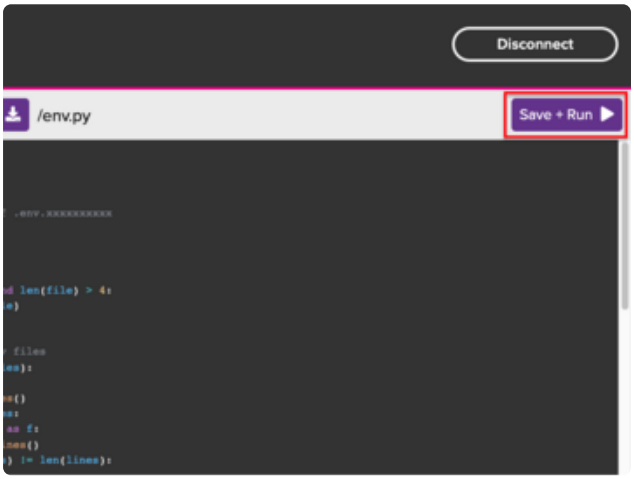
The X at the top performs the same function as the Cancel button as does clicking outside of the dialog.

On the Save As dialog, you can also type in a filename in the field next to the button.

Running Code

As mentioned above, the **Save** + **Run** button will first save your file, then run the code. The logic to run the code however is currently very simplistic in that it will try a couple of basic strategies to run your code, but doesn't currently do much beyond that.

The way it works is if you are working on **code.py** in the root folder, a soft reset will be performed, which automatically runs **code.py**. If you were working on some code in another file, the editor will attempt to perform an import on this code, which should run it. When you run your code, it will automatically switch over to the serial terminal.



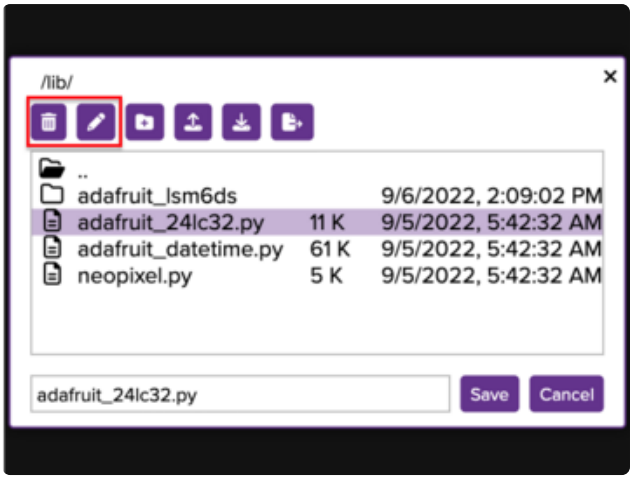
Click the **Save** + **Run** button to save and run the code current code.

File Dialog Toolbar

The file Dialog toolbar along the top allows you to perform common operations on files and folders regardless of whether you are saving or opening. Clicking the cancel button at the bottom will not undo any operations that were performed with these buttons.

Renaming and Deleting Files and Folders

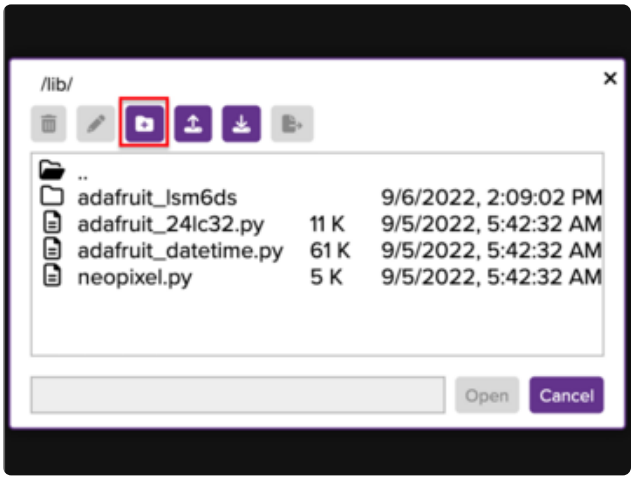
You can rename or delete both files and folders. An item must be selected first for the buttons to become available.

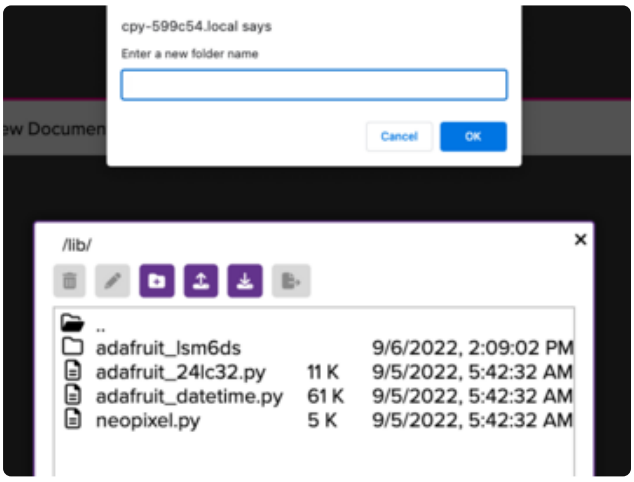


Use the **delete** and **rename** buttons here to perform the corresponding operation on the currently selected file or folder.

Creating New Folders

This feature allows you to create a new folder to store your work inside of.



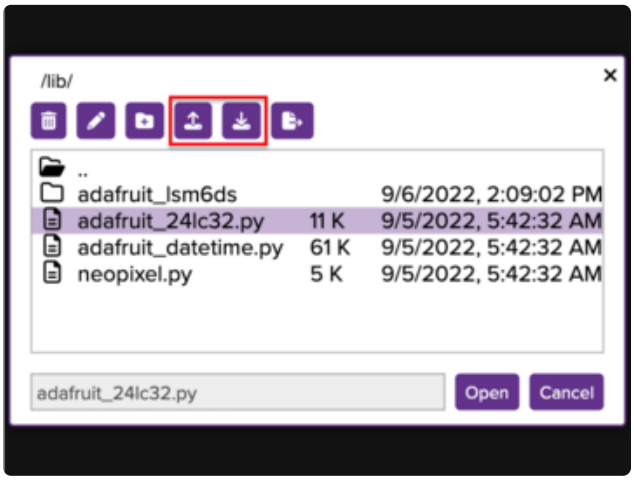


Clicking the **new folder** button at the top will prompt you for a folder name. It will inform you of invalid folder names such as the same name as an existing file or folder or a folder that begins with a period.

Uploading and Downloading Files and Folders

This feature allows you to upload or download files as long as they fit in the available space. If you need to add images or sound files for your project, you can use the upload button to add them. If you need to retrieve a file from your device for whatever reason, the download button will give you access to do that.

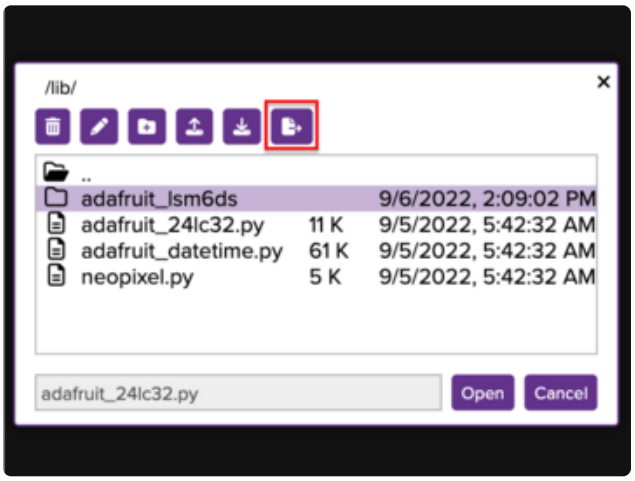
You can also download folders. When you select a folder and click download, the contents of that folder are automatically zipped into a single file. If nothing is selected when you click the download button, the current folder will be used.



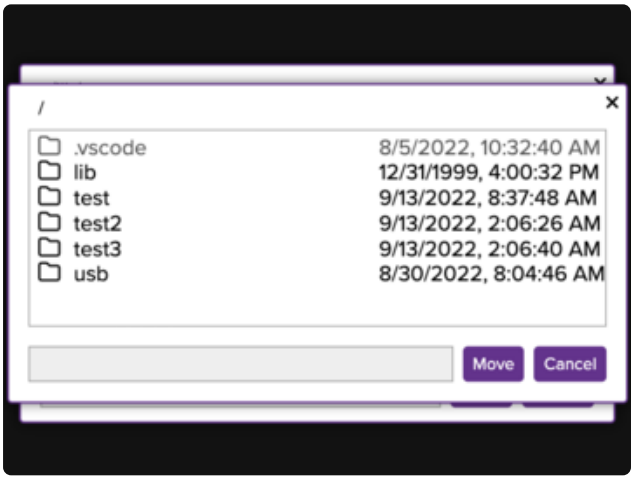
Use the **upload** or **download** buttons to easily add files or retrieve them from your board.

Moving Files and Folders

This feature allows you to move files and folders to a different location on the device. When you click the move button, another prompt will appear on top of the dialog that allows you to navigate to where you would like to move the currently selected item.



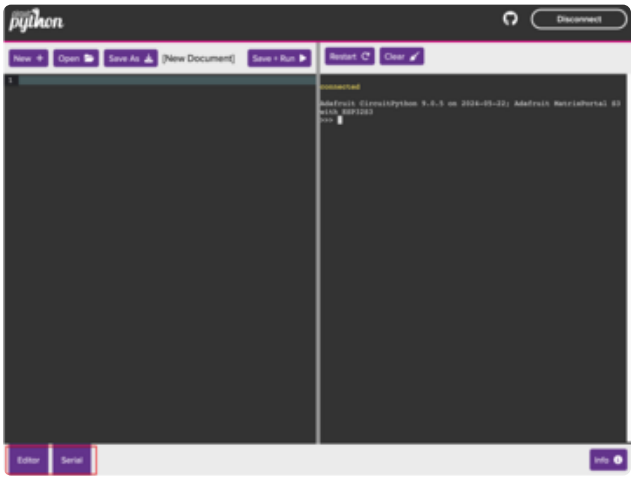
Use the **move** button to move files or folders to a new location on the device.



The second dialog that appears will show only folders and allow you to navigate to where you would like to move the file.

Using the Serial Terminal

The serial terminal allows you to watch the output of your device as well as type inputs just like you can from a separate application like PuTTY, except there's nothing you need to configure. This allows you to access the REPL or view the output of your currently running code.



Use the mode buttons in the bottom left-hand corner to open and close the serial and editor panes.

More Features to Come

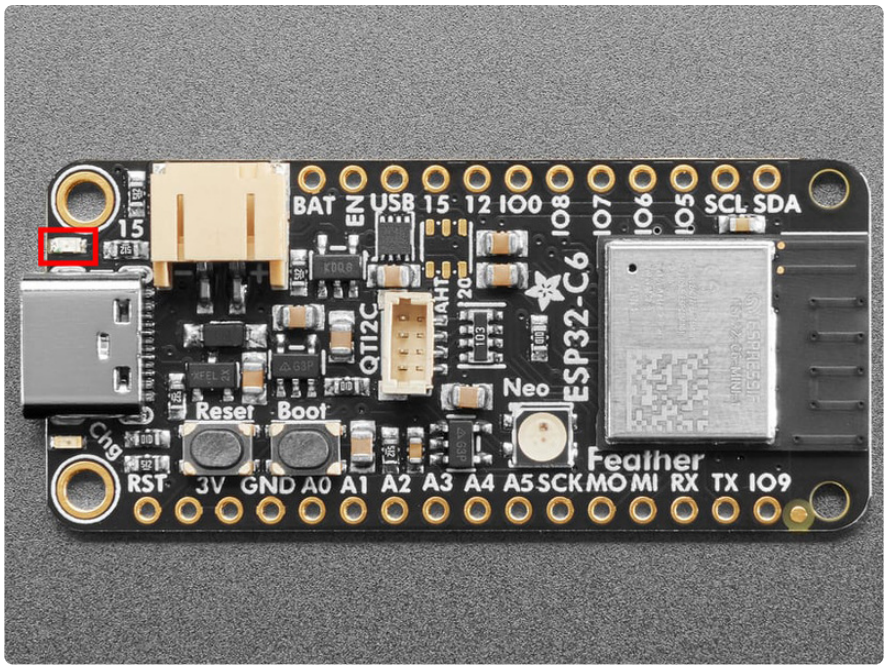
The CircuitPython Code Editor is still under development, so expect more features to be added. If you would like to contribute on GitHub (https://adafru.it/10Rc), you can submit any new issues or pull requests for review.

Blink

In learning any programming language, you often begin with some sort of `Hello, World!` program. In CircuitPython, Hello, World! is blinking an LED. Blink is one of the simplest programs in CircuitPython. It involves three built-in modules, two lines of set up, and a short loop. Despite its simplicity, it shows you many of the basic concepts needed for most CircuitPython programs, and provides a solid basis for more complex projects. Time to get blinky!

LED Location

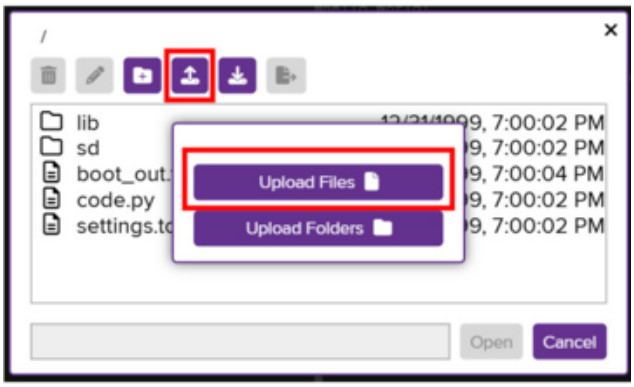
The LED is located between the USB C port and the mounting hole.



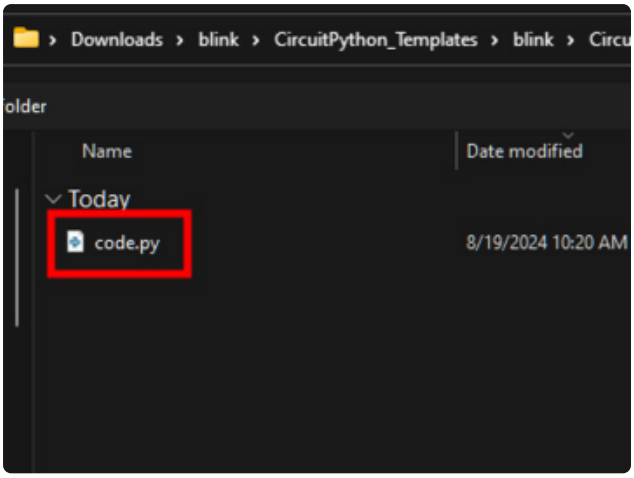
Blinking an LED

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_Templates/blink/** and then click on the directory that matches the version of CircuitPython you're using.

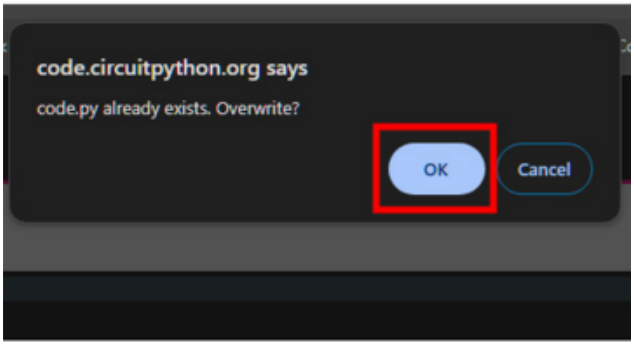
```
# SPDX-FileCopyrightText: 2021 Kattni Rembor for Adafruit Industries
# SPDX-License-Identifier: MIT
"""CircuitPython Blink Example - the CircuitPython 'Hello, World!'"""
import time
import board
import digitalio

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)
```



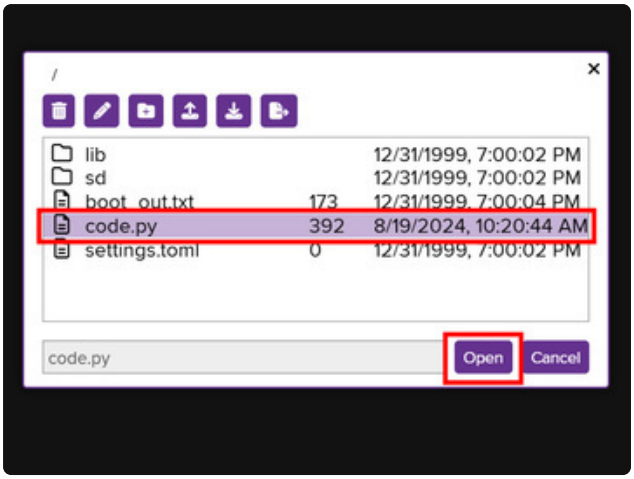
In the editor window in your browser, click the **Open** button to view the file dialog. Then, click the **Upload** button and select **Upload Files**.



Navigate to the project bundle that you downloaded and select the **code.py** file.

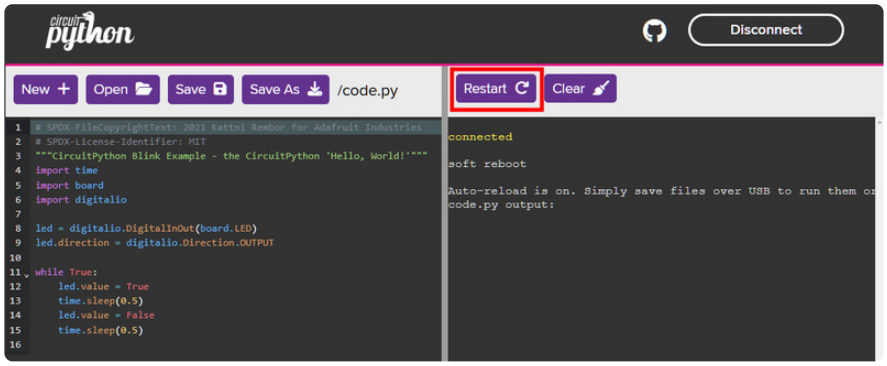


You'll be asked if you want to overwrite the previous **code.py** with the new **code.py** file from the Project Bundle. Click **OK**.



You'll see a new **code.py** file appear in the file browser. **Select it** and click **Open** to view it in the code editor.

You'll see the LED blink **code.py** file contents. Click **Restart** above the Serial monitor to run the LED blink code.



The built-in LED begins blinking!

Note that the code is a little less "Pythonic" than it could be. It could also be written as `led.value = not led.value` with a single `time.sleep(0.5)`. That way is more difficult to understand if you're new to programming, so the example is a bit longer than it needed to be to make it easier to read.

It's important to understand what is going on in this program.

First you `import` three modules: `time`, `board` and `digitalio`. This makes these modules available for use in your code. All three are built-in to CircuitPython, so you don't need to download anything to get started.

Next, you set up the LED. To interact with hardware in CircuitPython, your code must let the board know where to look for the hardware and what to do with it. So, you create a `digitalio.DigitalInOut()` object, provide it the LED pin using the `board` module, and save it to the variable `led`. Then, you tell the pin to act as an `OUTPUT`.

Finally, you create a `while True:` loop. This means all the code inside the loop will repeat indefinitely. Inside the loop, you set `led.value = True` which powers on the

LED. Then, you use `time.sleep(0.5)` to tell the code to wait half a second before moving on to the next line. The next line sets `led.value = False` which turns the LED off. Then you use another `time.sleep(0.5)` to wait half a second before starting the loop over again.

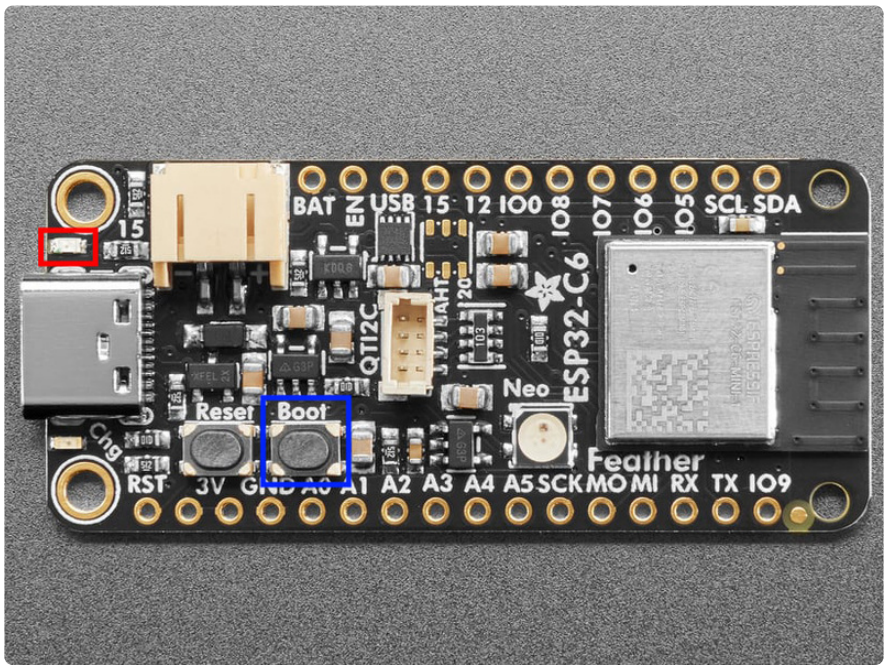
With only a small update, you can control the blink speed. The blink speed is controlled by the amount of time you tell the code to wait before moving on using `time.sleep()`. The example uses `0.5`, which is one half of one second. Try increasing or decreasing these values to see how the blinking changes.

That's all there is to blinking an LED using CircuitPython!

Digital Input

The CircuitPython `digitalio` module has many applications. The basic Blink program sets up the LED as a digital output. You can just as easily set up a **digital input** such as a button to control the LED. This example builds on the basic Blink example, but now includes setup for a button switch. Instead of using the `time` module to blink the LED, it uses the status of the button switch to control whether the LED is turned on or off.

LED and Button

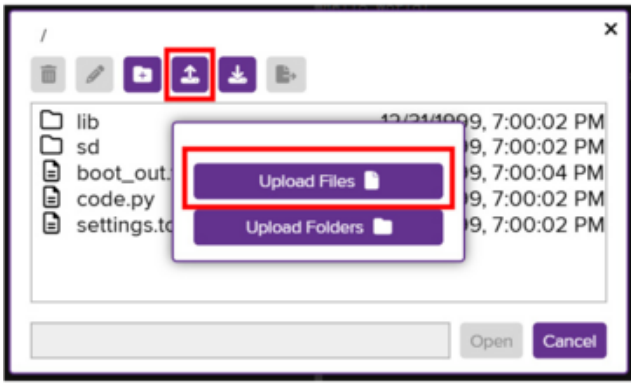


Controlling the LED with a Button

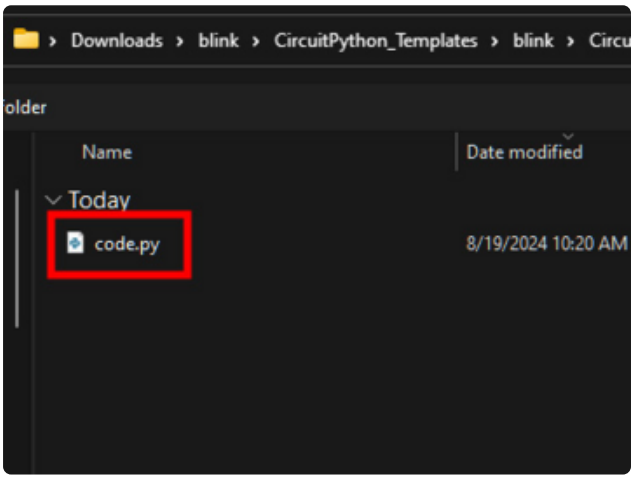
In the example below, click the **Download Project Bundle** button below to download the necessary libraries and the **code.py** file in a zip file. Extract the contents of the zip

file, open the directory **CircuitPython_Templates/digital_input_built_in_button_led/** and then click on the directory that matches the version of CircuitPython you're using.

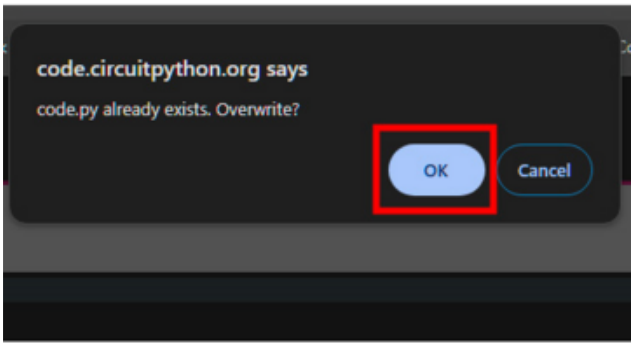
```
# SPDX-FileCopyrightText: 2022 Kattni Rembor for Adafruit Industries
# SPDX-License-Identifier: MIT
"""
CircuitPython Digital Input Example - Blinking an LED using the built-in button.
"""
import board
import digitalio

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

button = digitalio.DigitalInOut(board.BUTTON)
button.switch_to_input(pull=digitalio.Pull.UP)

while True:
    if not button.value:
        led.value = True
    else:
        led.value = False
```



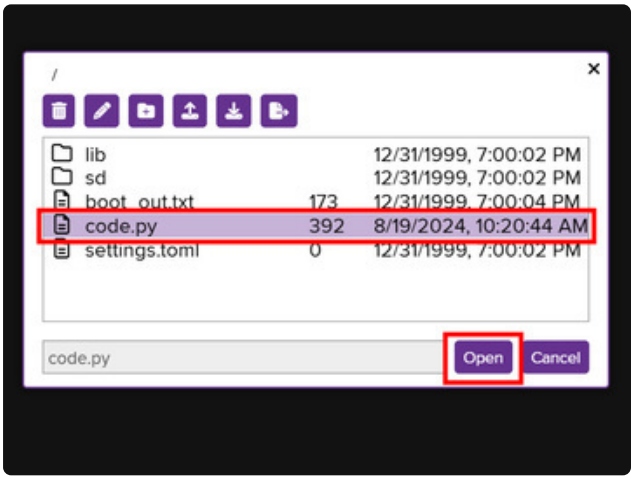
In the editor window in your browser, click the **Open** button to view the file dialog. Then, click the **Upload** button and select **Upload Files**.



Navigate to the project bundle that you downloaded and select the **code.py** file.

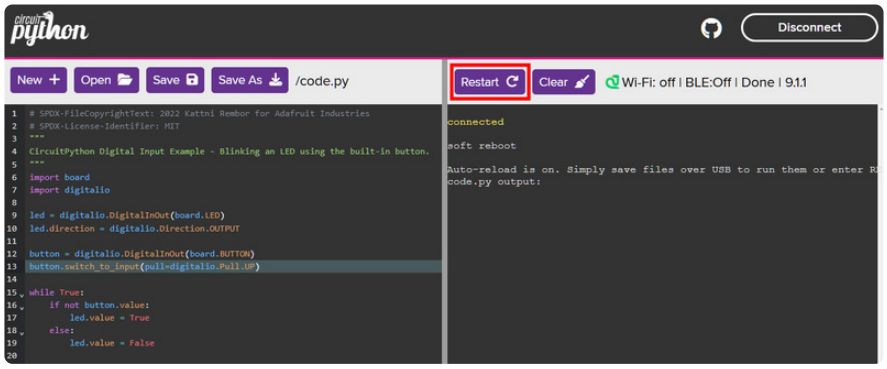


You'll be asked if you want to overwrite the previous **code.py** with the new **code.py** file from the Project Bundle. Click **OK**.

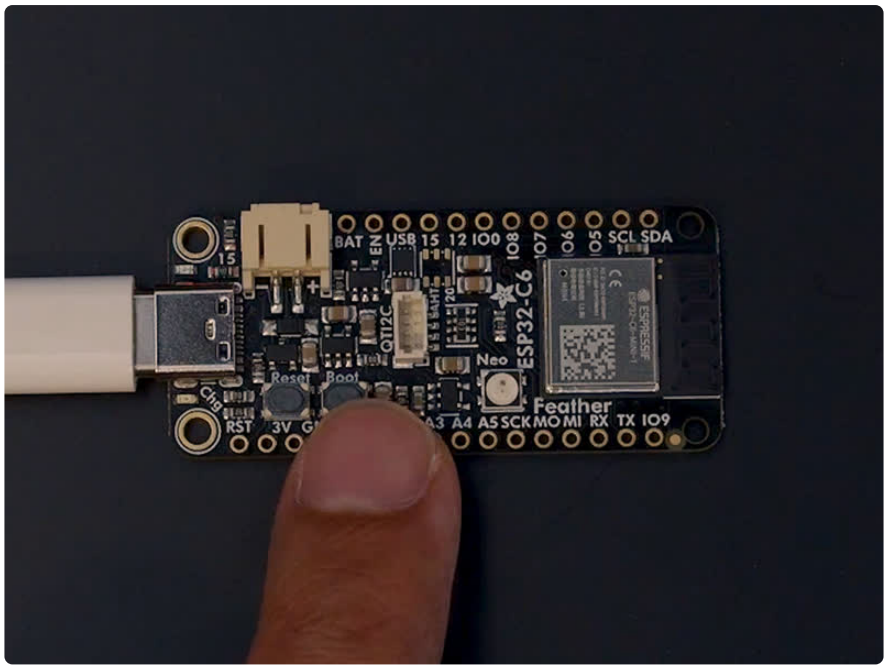


You'll see a new **code.py** file appear in the file browser. **Select it** and click **Open** to view it in the code editor.

You'll see the digital input **code.py** file contents. Click **Restart** above the Serial monitor to run the digital input code.



Now, press the button. The LED lights up! Let go of the button and the LED turns off.



Note that the code is a little less "Pythonic" than it could be. It could also be written as `led.value = not button.value`. That way is more difficult to understand if you're new to programming, so the example is a bit longer than it needed to be to make it easier to read.

First you `import` two modules: `board` and `digitalio`. This makes these modules available for use in your code. Both are built-in to CircuitPython, so you don't need to download anything to get started.

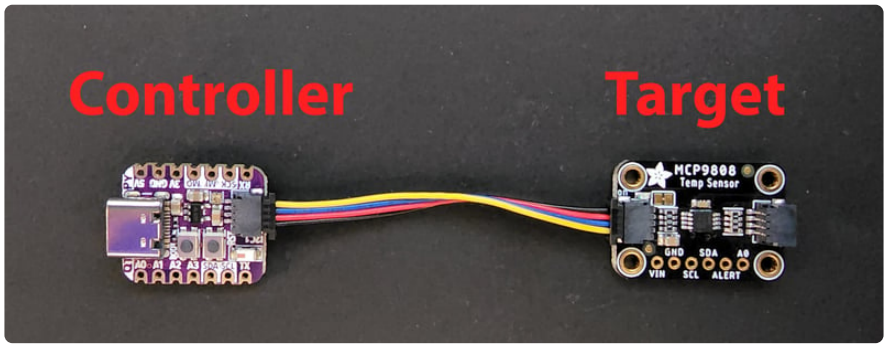
Next, you set up the LED. To interact with hardware in CircuitPython, your code must let the board know where to look for the hardware and what to do with it. So, you create a `digitalio.DigitalInOut()` object, provide it the LED pin using the `board` module, and save it to the variable `led`. Then, you tell the pin to act as an `OUTPUT`.

You include setup for the button as well. It is similar to the LED setup, except the button is an `INPUT`, and requires a pull up.

Inside the loop, you check to see if the button is pressed, and if so, turn on the LED. Otherwise the LED is off.

That's all there is to controlling an LED with a button switch!

I2C Scan



The **I2C**, or inter-integrated circuit (https://adafru.it/u2a), is a 2-wire protocol for communicating with simple sensors and devices, which means it uses two connections, or wires, for transmitting and receiving data. One connection is a clock, called **SCL**. The other is the data line, called **SDA**. Each pair of clock and data pins are referred to as a **bus**.

Typically, there is a device that acts as a **controller** and sends requests to the **target** devices on each bus. In this case, your microcontroller board acts as the controller, and the sensor breakout acts as the target. Historically, the controller is referred to as the master, and the target is referred to as the slave, so you may run into that terminology elsewhere. The official terminology is controller and target (https://adafru.it/TtF).

Multiple I2C devices can be connected to the same clock and data lines. Each I2C device has an address, and as long as the addresses are different, you can connect them at the same time. This means you can have many different sensors and devices all connected to the same two pins.

Both I2C connections require pull-up resistors, and most Adafruit I2C sensors and breakouts have pull-up resistors built in. If you're using one that does not, you'll need to add your own 2.2-10kΩ pull-up resistors from SCL and SDA to 3.3V.

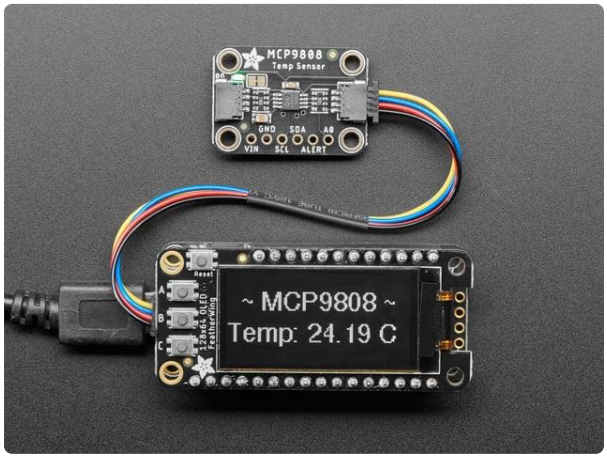
I2C and CircuitPython

CircuitPython supports many I2C devices, and makes it super simple to interact with them. There are libraries available for many I2C devices in the CircuitPython Library Bundle (https://adafru.it/Tra). (If you don't see the sensor you're looking for, keep checking back, more are being written all the time!)

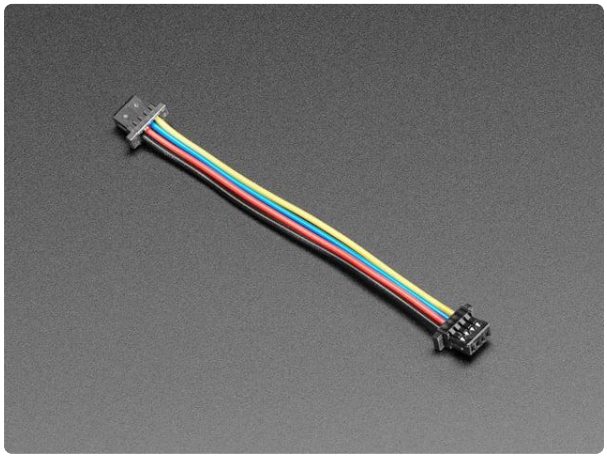
In this section, you'll learn how to scan the I2C bus for all connected devices. Then you'll learn how to interact with an I2C device.

Necessary Hardware

You'll need the following additional hardware to complete the examples on this page.



Adafruit MCP9808 High Accuracy I2C Temperature Sensor Breakout
The MCP9808 digital temperature sensor is one of the more accurate/precise we've ever seen, with a typical accuracy of ±0.25°C over the sensor's -40°C to...
https://www.adafruit.com/product/5027



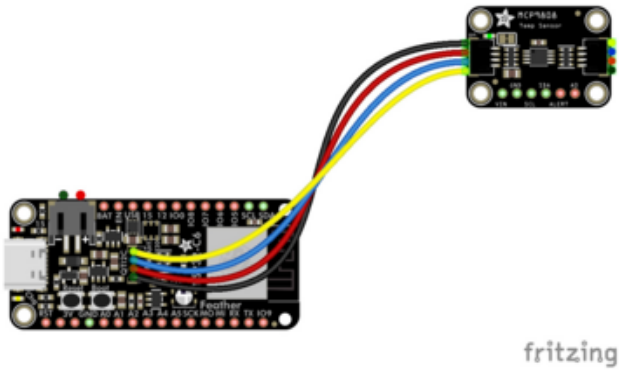
STEMMA QT / Qwiic JST SH 4-Pin Cable - 50mm Long
This 4-wire cable is 50mm / 1.9" long and fitted with JST SH female 4-pin connectors on both ends. Compared with the chunkier JST PH these are 1mm pitch instead of 2mm, but...
https://www.adafruit.com/product/4399

While the examples here will be using the Adafruit MCP9808 (http://adafru.it/5027), a high accuracy temperature sensor, the overall process is the same for just about any I2C sensor or device.

The first thing you'll want to do is get the sensor connected so your board has I2C to talk to.

Wiring the MCP9808

The MCP9808 comes with a STEMMA QT connector, which makes wiring it up quite simple and solder-free.



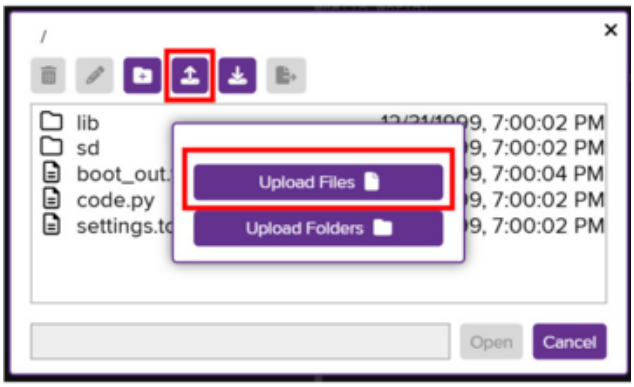
Connect the STEMMA QT cable from **the STEMMA QT port on your board** to **the STEMMA QT port on the MCP9808**.

Find Your Sensor

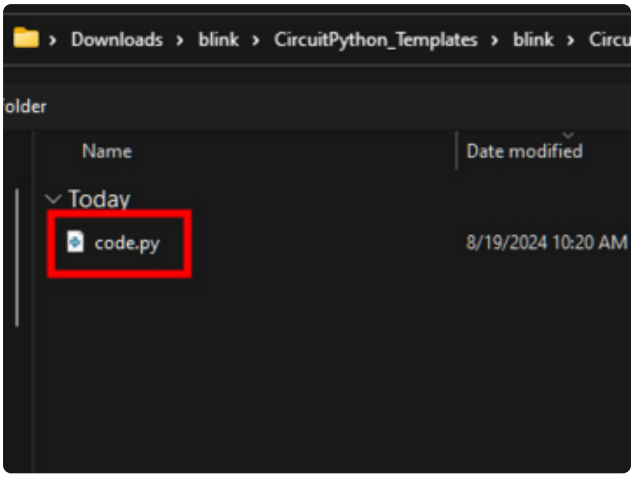
The first thing you'll want to do after getting the sensor wired up, is make sure it's wired correctly. You're going to do an I2C scan to see if the board is detected, and if it is, print out its I2C address.

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_Templates/i2c_scan/** and then click on the directory that matches the version of CircuitPython you're using.

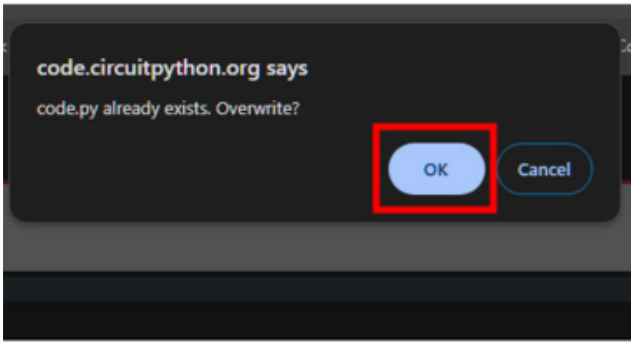
```
# SPDX-FileCopyrightText: 2021 Kattni Rembor for Adafruit Industries
# SPDX-License-Identifier: MIT
"""CircuitPython I2C Device Address Scan"""
import time
import board

i2c = board.I2C()  # uses board.SCL and board.SDA
# i2c = board.STEMMA_I2C()  # For using the built-in STEMMA QT connector on a
microcontroller

# To create I2C bus on specific pins
# import busio
# i2c = busio.I2C(board.GP1, board.GP0)    # Pi Pico RP2040

while not i2c.try_lock():
    pass

try:
    while True:
        print(
            "I2C addresses found:",
            [hex(device_address) for device_address in i2c.scan()],
        )
        time.sleep(2)

finally:  # unlock the i2c bus when ctrl-c'ing out of the loop
    i2c.unlock()
```



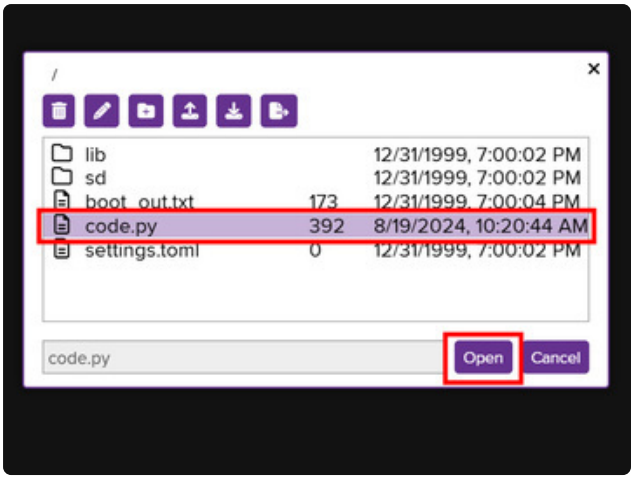
In the editor window in your browser, click the **Open** button to view the file dialog. Then, click the **Upload** button and select **Upload Files**.



Navigate to the project bundle that you downloaded and select the **code.py** file.



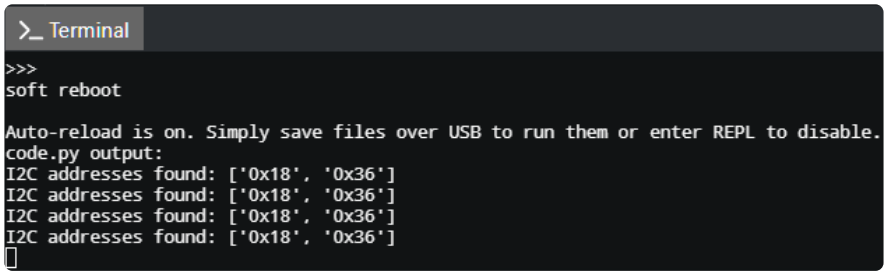
You'll be asked if you want to overwrite the previous **code.py** with the new **code.py** file from the Project Bundle. Click **OK**.



You'll see a new **code.py** file appear in the file browser. **Select it** and click **Open** to view it in the code editor.

The Feather ESP32-C6 comes with 1 I2C sensor built in: the MAX17048. The I2C scan code will show the address from the built in sensor (**0x36**) and the MCP9808 (**0x18**).

> ESP32-C6 Feather comes with an I2C sensor built in: the MAX17048. The I2C scan code will show the addresses from the built-in sensor and the MCP9808.



If you run this and it seems to hang, try manually unlocking your I2C bus by running the following two commands from the REPL.

```
import board
board.I2C().unlock()
```

First you create the `i2c` object, using `board.I2C()`. This convenience routine creates and saves a `busio.I2C` object using the default pins `board.SCL` and `board.SDA`. If the object has already been created, then the existing object is returned. No matter how many times you call `board.I2C()`, it will return the same object. This is called a singleton.

To be able to scan it, you need to lock the I2C down so the only thing accessing it is the code. So next you include a loop that waits until I2C is locked and then continues on to the scan function.

Last, you have the loop that runs the actual scan, `i2c_scan()`. Because I2C typically refers to addresses in hex form, the example includes this bit of code that formats the results into hex format: `[hex(device_address) for device_address in i2c.scan()]`.

Open the serial console to see the results! The code prints out an array of addresses. You've connected the MCP9808 which has a 7-bit I2C address of 0x18. The result for this sensor is `I2C addresses found: ['0x18']`. If no addresses are returned, refer back to the wiring diagrams to make sure you've wired up your sensor correctly.

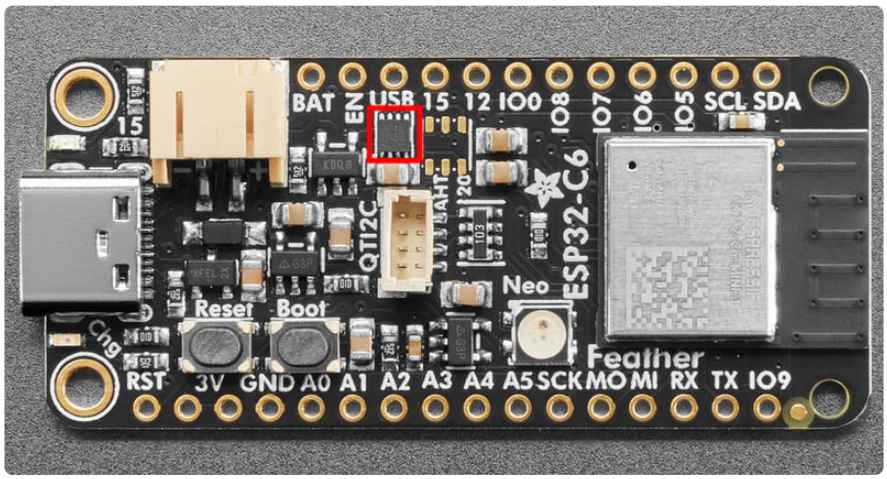
MAX17048 Battery Monitor

Your microcontroller board comes with an **MAX17048 lithium ion polymer (lipoly) battery monitor** built right onto the board. The MAX17048 is available over I2C.

The MAX17048 comes with its own Adafruit CircuitPython library that makes it simple to write code to read data from it. This example will be using, among other things, the Adafruit CircuitPython MAX1704x (https://adafru.it/10RA) library.

The example simply reads data from the battery monitor and prints it to the serial console. It is designed to show you how to get data from the battery monitor.

MAX17048 Location



The **MAX17048 battery monitor** (highlighted in red) is immediately below the **USB** pin label. Its I2C address is **0x36**.

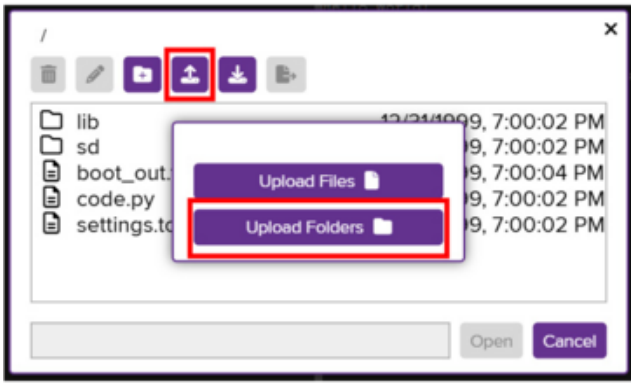
MAX17048 Simple Data Example

To run this example, you need to first install the MAX1704x library into the **lib** folder on your board. Then you need to update **code.py** with the example script.

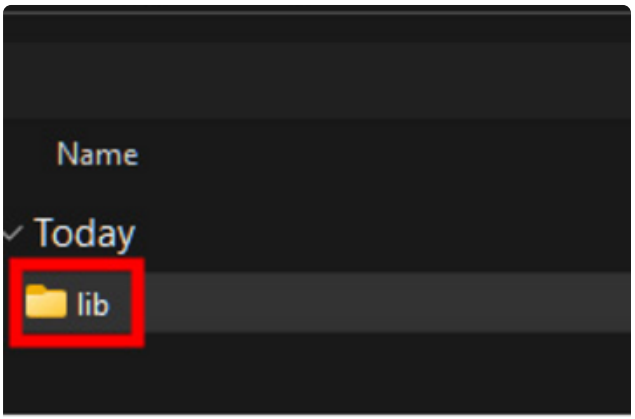
In the example below, click the **Download Project Bundle** button below to download the necessary libraries and the **code.py** file in a zip file. Extract the contents of the zip file.

```
# SPDX-FileCopyrightText: Copyright (c) 2023 Kattni Rembor for Adafruit Industries
#
# SPDX-License-Identifier: Unlicense

import time
import board
import adafruit_max1704x

monitor = adafruit_max1704x.MAX17048(board.I2C())

while True:
    print(f"Battery voltage: {monitor.cell_voltage:.2f} Volts")
    print(f"Battery percentage: {monitor.cell_percent:.1f} %")
    print("")
    time.sleep(1)
```

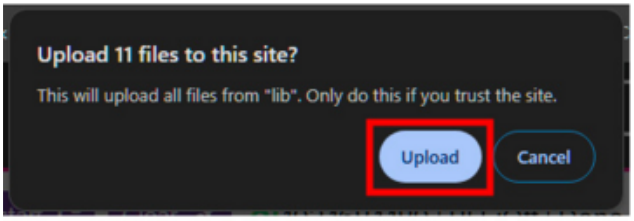
Update the /**lib** Folder



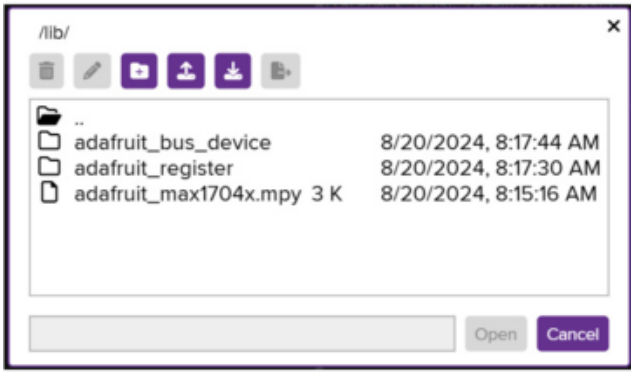
In the editor window in your browser, click the **Open** button to view the file dialog. Then, click the **Upload** button and select **Upload Folders**.



Navigate to the project bundle that you downloaded and select the **/lib folder**.

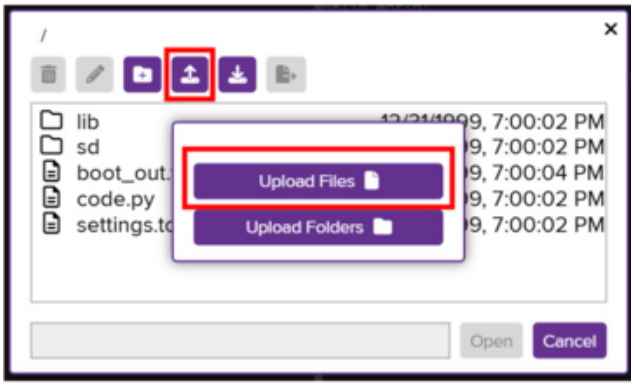


You'll be asked if you want to upload the /**lib** folder from the Project Bundle. Click **Upload**.

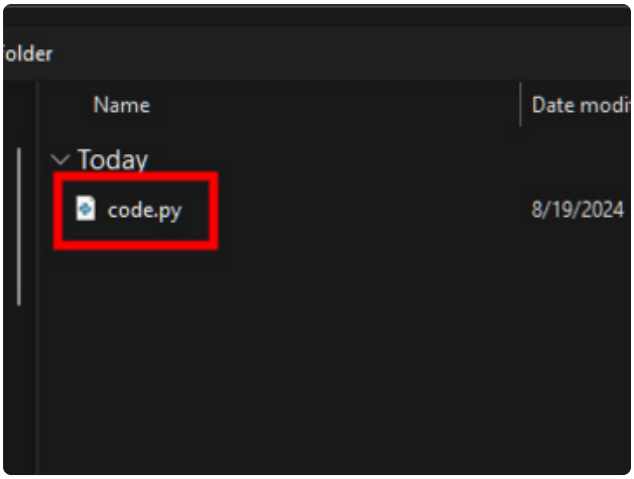


After the upload finishes, you can open the **lib** folder to view the library files required for the MAX17048 example.

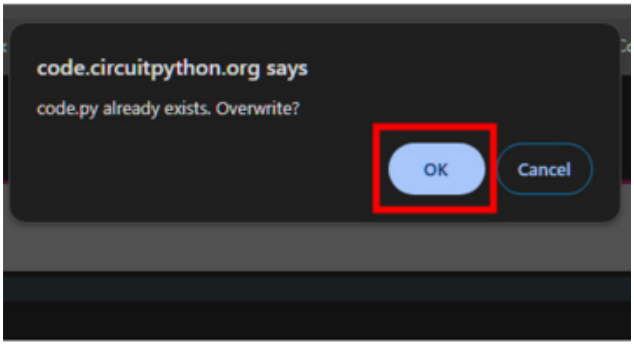
Update **code.py**



In the editor window in your browser, click the **Open** button to view the file dialog. Then, click the **Upload** button and select **Upload Files**.



Navigate to the project bundle that you downloaded and select the **code.py** file.



You'll be asked if you want to overwrite the previous **code.py** with the new **code.py** file from the Project Bundle. Click **OK**.

This code will run without a battery plugged in, and voltage and charge level will be printed to the serial console, but this data does not correlate to anything. Plug in a battery to get useful data!

Open the serial console to see the battery data printed out!

That's all there is to reading the MAX17048 data using CircuitPython!

For more details, check out the guide for the MAX17048 (https://adafru.it/18f8).

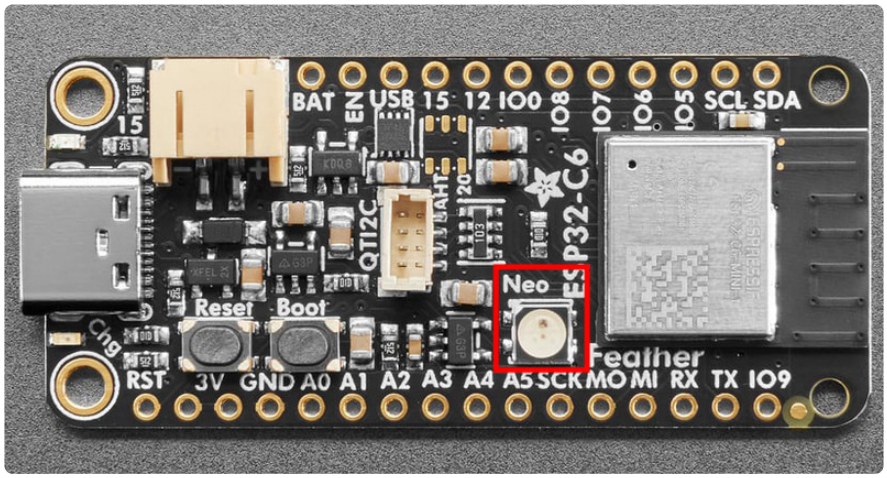
NeoPixel

Your board has a built-in RGB NeoPixel status LED. You can use CircuitPython code to control the color and brightness of this LED. It is also used to indicate the bootloader status and errors in your CircuitPython code.

A NeoPixel is what Adafruit calls the WS281x family of addressable RGB LEDs. It contains three LEDs - a red one, a green one and a blue one - along side a driver chip in a tiny package controlled by a single pin. They can be used individually (as in the built-in LED on your board), or chained together in strips or other creative form factors. NeoPixels do not light up on their own; they require a microcontroller. So, it's super convenient that the NeoPixel is built in to your microcontroller board!

This page will cover using CircuitPython to control the status RGB NeoPixel built into your microcontroller. You'll learn how to change the color and brightness, and how to make a rainbow. Time to get started!

NeoPixel Location



NeoPixel Color and Brightness

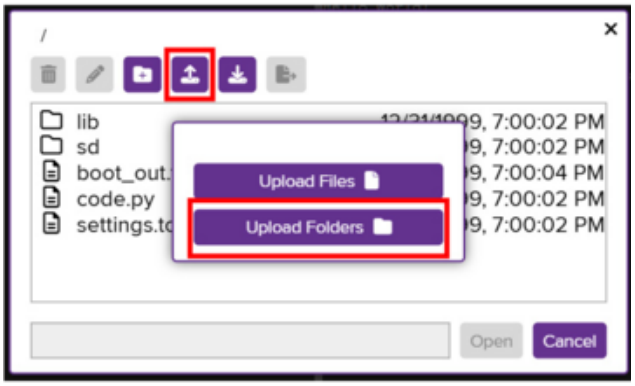
To use with CircuitPython, you need to first install a few libraries, into the **lib** folder on your board. Then you need to update **code.py** with the example script.

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_Templates/status_led_one_neopixel_rgb/** and then click on the directory that matches the version of CircuitPython you're using.

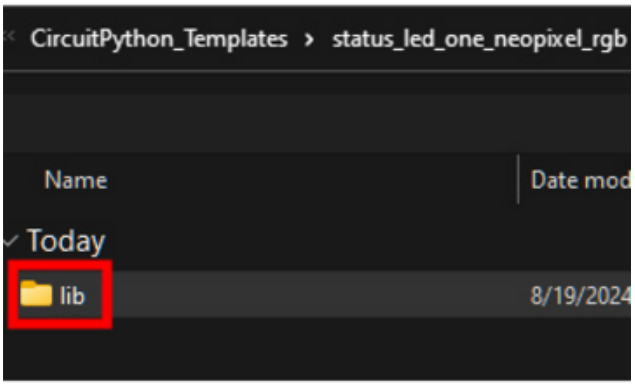
```
# SPDX-FileCopyrightText: 2021 Kattni Rembor for Adafruit Industries
# SPDX-License-Identifier: MIT
"""CircuitPython status NeoPixel red, green, blue example."""
```

```
import time
import board
import neopixel

pixel = neopixel.NeoPixel(board.NEOPIXEL, 1)

pixel.brightness = 0.3

while True:
    pixel.fill((255, 0, 0))
    time.sleep(0.5)
    pixel.fill((0, 255, 0))
    time.sleep(0.5)
    pixel.fill((0, 0, 255))
    time.sleep(0.5)
```

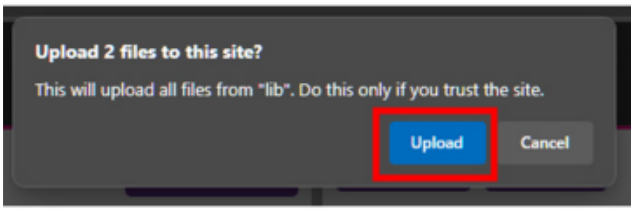
Update the /lib Folder



In the editor window in your browser, click the **Open** button to view the file dialog. Then, click the **Upload** button and select **Upload Folders**.



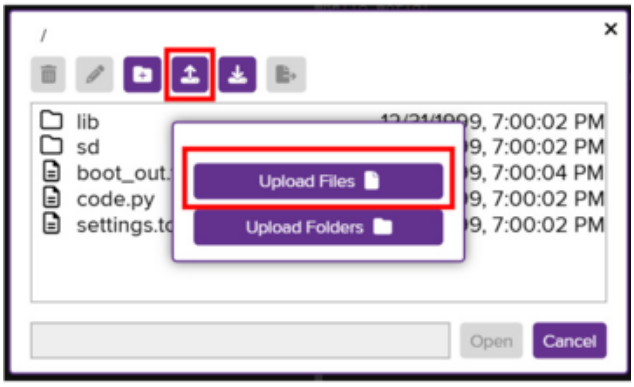
Navigate to the project bundle that you downloaded and select the **/lib folder**.



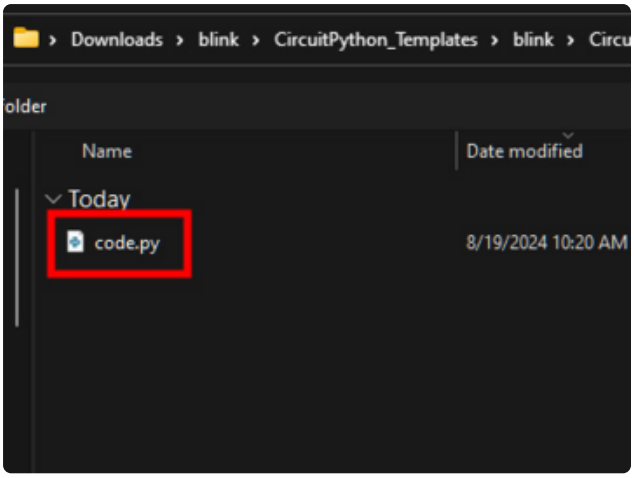
You'll be asked if you want to upload the /**lib** folder from the Project Bundle. Click **Upload**.

After the upload finishes, you can open the **lib** folder to view the two library files required for the NeoPixel examples.

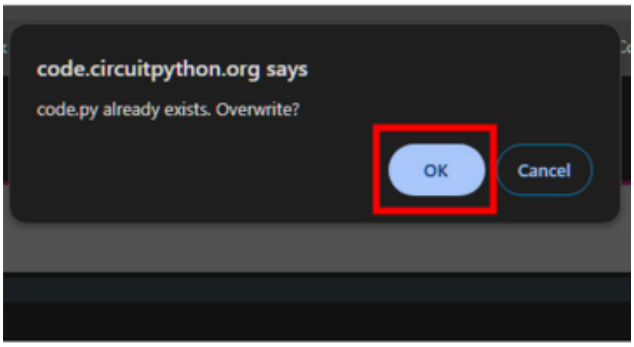
Update **code.py**



In the editor window in your browser, click the **Open** button to view the file dialog. Then, click the **Upload** button and select **Upload Files**.

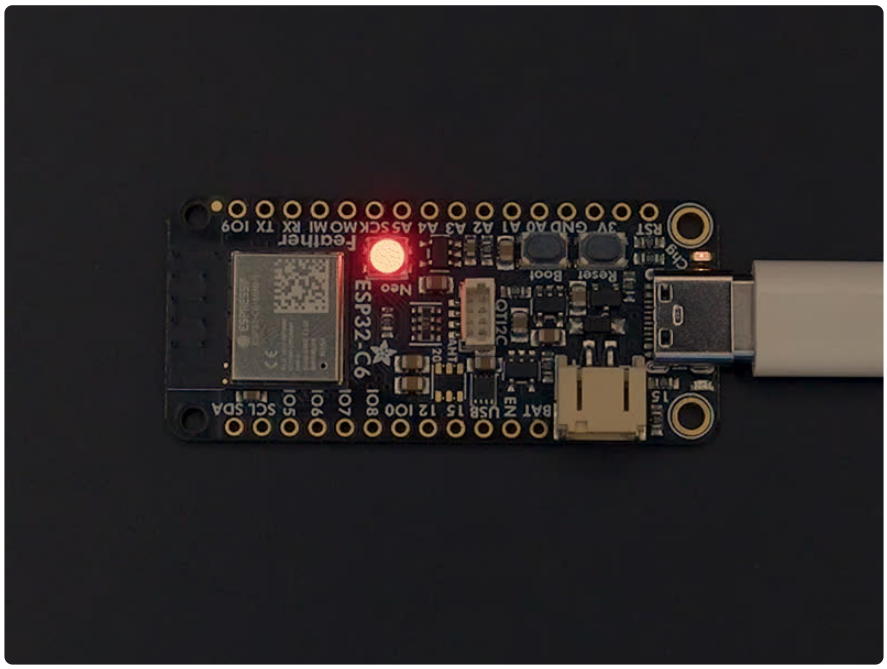


Navigate to the project bundle that you downloaded and select the **code.py** file.



You'll be asked if you want to overwrite the previous **code.py** with the new **code.py** file from the Project Bundle. Click **OK**.

The built-in NeoPixel begins blinking red, then green, then blue, and repeats!



First you import two modules, `time` and `board`, and one library, `neopixel`. This makes these modules and libraries available for use in your code. The first two are modules built-in to CircuitPython, so you don't need to download anything to use

those. The `neopixel` library is separate, which is why you needed to install it before getting started.

Next, you set up the NeoPixel LED. To interact with hardware in CircuitPython, your code must let the board know where to look for the hardware and what to do with it. So, you create a `neopixel.NeoPixel()` object, provide it the NeoPixel LED pin using the `board` module, and tell it the number of LEDs. You save this object to the variable `pixel`.

Then, you set the NeoPixel brightness using the `brightness` attribute. `brightness` expects float between `0` and `1.0`. A float is essentially a number with a decimal in it. The brightness value represents a percentage of maximum brightness; `0` is 0% and `1.0` is 100%. Therefore, setting `pixel.brightness = 0.3` sets the brightness to 30%. The default brightness, which is to say the brightness if you don't explicitly set it, is `1.0`. The default is really bright! That is why there is an option available to easily change the brightness.

Inside the loop, you turn the NeoPixel red for 0.5 seconds, green for 0.5 seconds, and blue for 0.5 seconds.

To turn the NeoPixel red, you "fill" it with an RGB value. Check out the section below for details on RGB colors. The RGB value for red is `(255, 0, 0)`. Note that the RGB value includes the parentheses. The `fill()` attribute expects the full RGB value including those parentheses. That is why there are two pairs of parentheses in the code.

You can change the RGB values to change the colors that the NeoPixel cycles through. Check out the list below for some examples. You can make any color of the rainbow with the right RGB value combination!

That's all there is to changing the color and setting the brightness of the built-in NeoPixel LED!

RGB LED Colors

RGB LED colors are set using a combination of **r**ed, **g**reen, and **b**lue, in the form of an (**R**, **G**, **B**) tuple. Each member of the tuple is set to a number between 0 and 255 that determines the amount of each color present. Red, green and blue in different combinations can create all the colors in the rainbow! So, for example, to set an LED to red, the tuple would be `(255, 0, 0)`, which has the maximum level of red, and no green or blue. Green would be `(0, 255, 0)`, etc. For the colors between, you set a combination, such as cyan which is `(0, 255, 255)`, with equal amounts of green and blue. If you increase all values to the same level, you get white! If you decrease all the values to 0, you turn the LED off.

Common colors include:

- red: `(255, 0, 0)`
- green: `(0, 255, 0)`
- blue: `(0, 0, 255)`
- cyan: `(0, 255, 255)`
- purple: `(255, 0, 255)`
- yellow: `(255, 255, 0)`
- white: `(255, 255, 255)`
- black (off): `(0, 0, 0)`

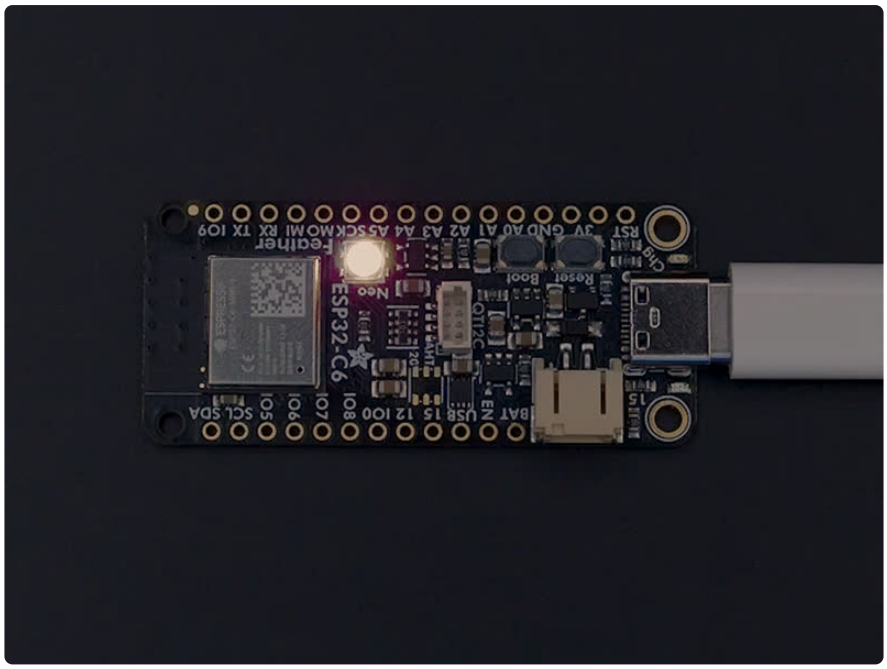
NeoPixel Rainbow

You should have already installed the library necessary to use the built-in NeoPixel LED. If not, follow the steps at the beginning of the NeoPixel Color and Brightness section to install it.

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_Templates/status_led_one_neopixel_rainbow/** and then click on the directory that matches the version of CircuitPython you're using.

```
# SPDX-FileCopyrightText: 2021 Kattni Rembor for Adafruit Industries
# SPDX-License-Identifier: MIT
"""CircuitPython status NeoPixel rainbow example."""
import time
import board
from rainbowio import colorwheel
import neopixel

pixel = neopixel.NeoPixel(board.NEOPIXEL, 1)
pixel.brightness = 0.3


def rainbow(delay):
    for color_value in range(255):
        pixel[0] = colorwheel(color_value)
        time.sleep(delay)


while True:
    rainbow(0.02)
```

Update the **code.py** file in the USB code editor with the rainbow **code.py** file. The same libraries from the RGB blinking example are used. The NeoPixel displays a rainbow cycle!



This example builds on the previous example.

First, you import the same three modules and libraries. In addition to those, you import `colorwheel`.

The NeoPixel hardware setup and brightness setting are the same.

Next, you have the `rainbow()` helper function. This helper displays the rainbow cycle. It expects a `delay` in seconds. The higher the number of seconds provided for `delay`, the slower the rainbow will cycle. The helper cycles through the values of the color wheel to create a rainbow of colors.

Inside the loop, you call the rainbow helper with a 0.2 second delay, by including `rainbow(0.2)`.

That's all there is to making rainbows using the built-in NeoPixel LED!

WiFi Test

In this example, you'll test your ESP32-C6 Feather WiFi connection by connecting to your SSID, printing your MAC address and IP address to the REPL and then pinging Google.

settings.toml File

If you've worked on WiFi projects with CircuitPython before, you're probably familiar with the **secrets.py** file. This file is a Python file that is stored on your **CIRCUITPY** drive that contains all of your secret WiFi information, such as your SSID, SSID password and any API keys for IoT services.

As of CircuitPython 8 (https://adafru.it/Em8), there is support for a **settings.toml** file. Similar to **secrets.py**, the **settings.toml** file separates your sensitive information from your main **code.py** file.

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

settings.toml File Example

Here is an example on how to format your **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:

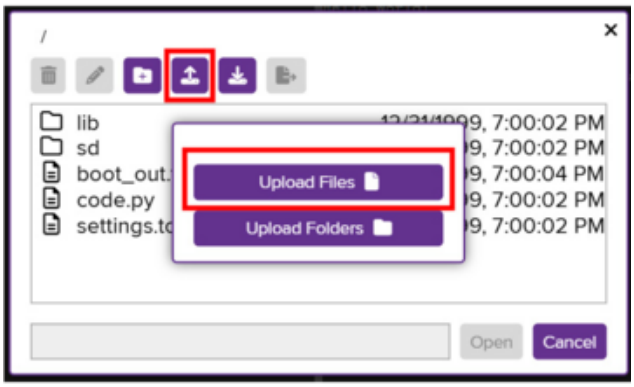
- Strings are wrapped in double quotes; ex: `"your-string-here"`
- Integers are **not** quoted and may be written in decimal with optional sign (`+1`, `-1`, `1000`) or hexadecimal (`0xabcd`).
    - Floats, octal (`0o567`) and binary (`0b11011`) are not supported.
- Use `\u` escapes for weird characters,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

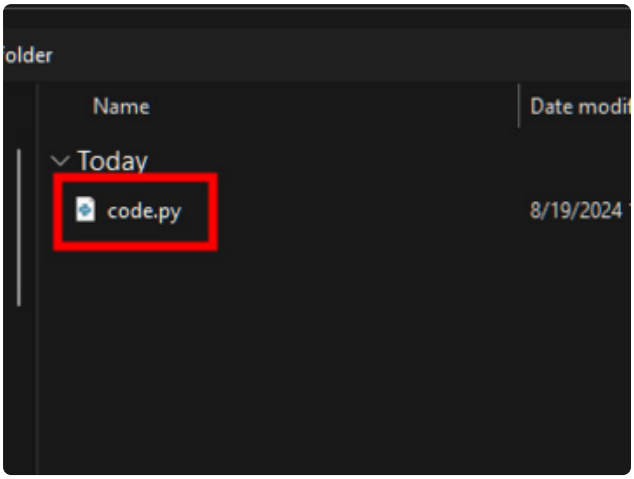
CircuitPython WiFi Example

In the example below, click the **Download Project Bundle** button below to download the **code.py** file in a zip file. Extract the contents of the zip file and then click on the directory that matches the version of CircuitPython you're using.

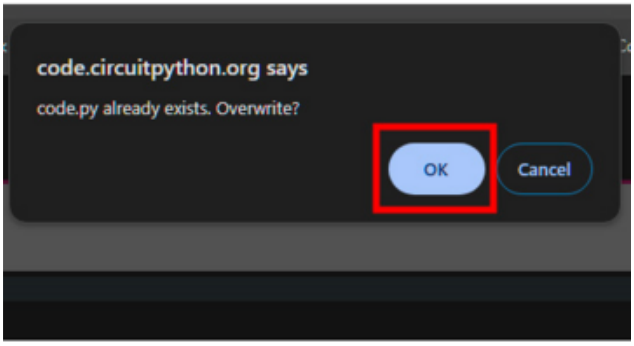
```
# SPDX-FileCopyrightText: 2022 Liz Clark for Adafruit Industries
#
# SPDX-License-Identifier: MIT

import os
import ipaddress
import wifi
import socketpool

print()
print("Connecting to WiFi")

#  connect to your SSID
try:
    wifi.radio.connect(os.getenv('CIRCUITPY_WIFI_SSID'),
os.getenv('CIRCUITPY_WIFI_PASSWORD'))
except TypeError:
    print("Could not find WiFi info. Check your settings.toml file!")
    raise

print("Connected to WiFi")

pool = socketpool.SocketPool(wifi.radio)

#  prints MAC address to REPL
print("My MAC addr:", [hex(i) for i in wifi.radio.mac_address])

#  prints IP address to REPL
print("My IP address is", wifi.radio.ipv4_address)

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



In the editor window in your browser, click the **Open** button to view the file dialog. Then, click the **Upload** button and select **Upload Files**.



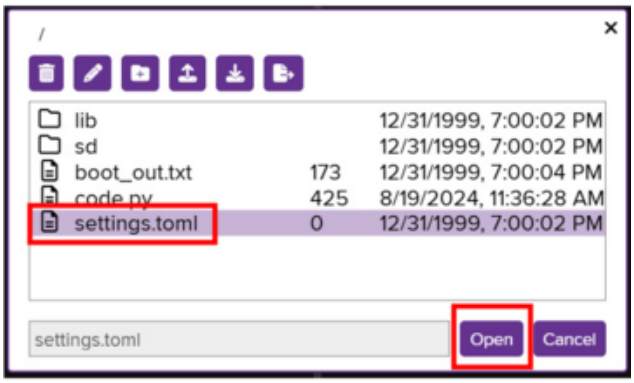
Navigate to the project bundle that you downloaded and select the **code.py** file.



You'll be asked if you want to overwrite the previous **code.py** with the new **code.py** file from the Project Bundle. Click **OK**.

Update Your **settings.toml** File

Remember to add your **settings.toml** file as described earlier in this page. You'll need to include your `CIRCUITPY_WIFI_SSID` and `CIRCUITPY_WIFI_PASSWORD` in the file.



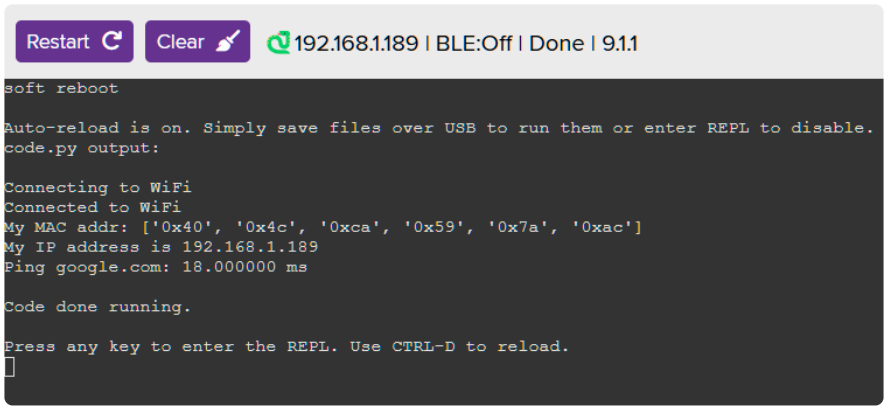
You can edit the file manually in the USB code editor by clicking **Open**, selecting **settings.toml** and clicking **Open** at the bottom of the dialog box.

With **settings.toml** open in the editor, you can add your WiFi credentials:

```
CIRCUITPY_WIFI_SSID = "your-ssid-here"
CIRCUITPY_WIFI_PASSWORD = "your-ssid-password-here"
```

Once your credentials are entered, click **Save** above the editor to save your changes to **settings.toml**.

Once everything is saved to the board, **Restart** the Serial Console to see the data printed out!



How the CircuitPython WiFi Example Works

In the basic WiFi test, the board connects to your SSID by importing your SSID and SSID password from the **settings.toml** file.

```
wifi.radio.connect(os.getenv('CIRCUITPY_WIFI_SSID'),
os.getenv('CIRCUITPY_WIFI_PASSWORD'))
```

Then, your MAC address and IP address are printed to the REPL.

```
#  prints MAC address to REPL
print("My MAC addr:", [hex(i) for i in wifi.radio.mac_address])

#  prints IP address to REPL
print("My IP address is", wifi.radio.ipv4_address)
```

Finally, google.com is pinged. The amount of time it takes to ping is printed to the REPL and the code stops running.

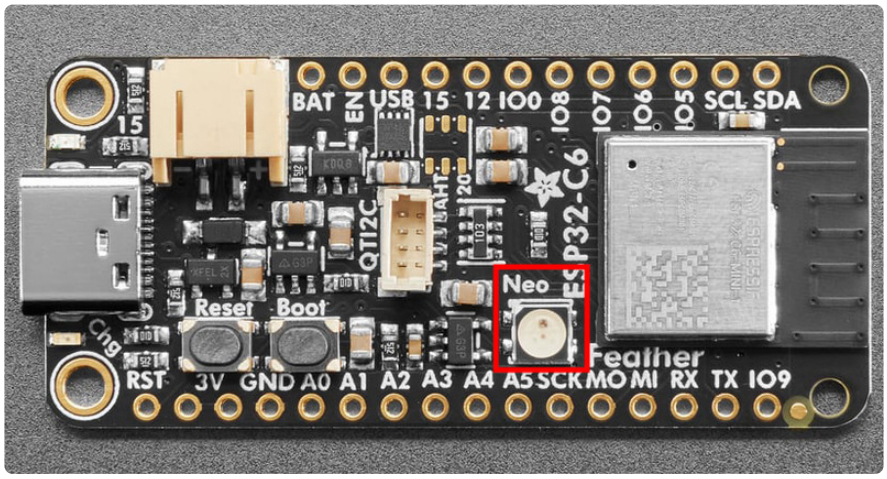
```
#  pings Google
ipv4 = ipaddress.ip_address("8.8.4.4")
print("Ping google.com: %f ms" % (wifi.radio.ping(ipv4)*1000))
```

By successfully running this WiFi test code, you can confirm that your board is connecting to WiFi with CircuitPython successfully and you can move on to more advanced projects.

Adafruit IO

Adafruit IO gives you the option to disconnect your microcontroller from your computer and run it off of USB power or a battery, and still be able to see the data. It also allows you to send data to your microcontroller, such as NeoPixel colors. This example shows how to both send data to and receive data from Adafruit IO. It pulls from a "random" number generator and sends the "random" number to Adafruit IO, while simultaneously listening for NeoPixel color data from Adafruit IO.

NeoPixel Location



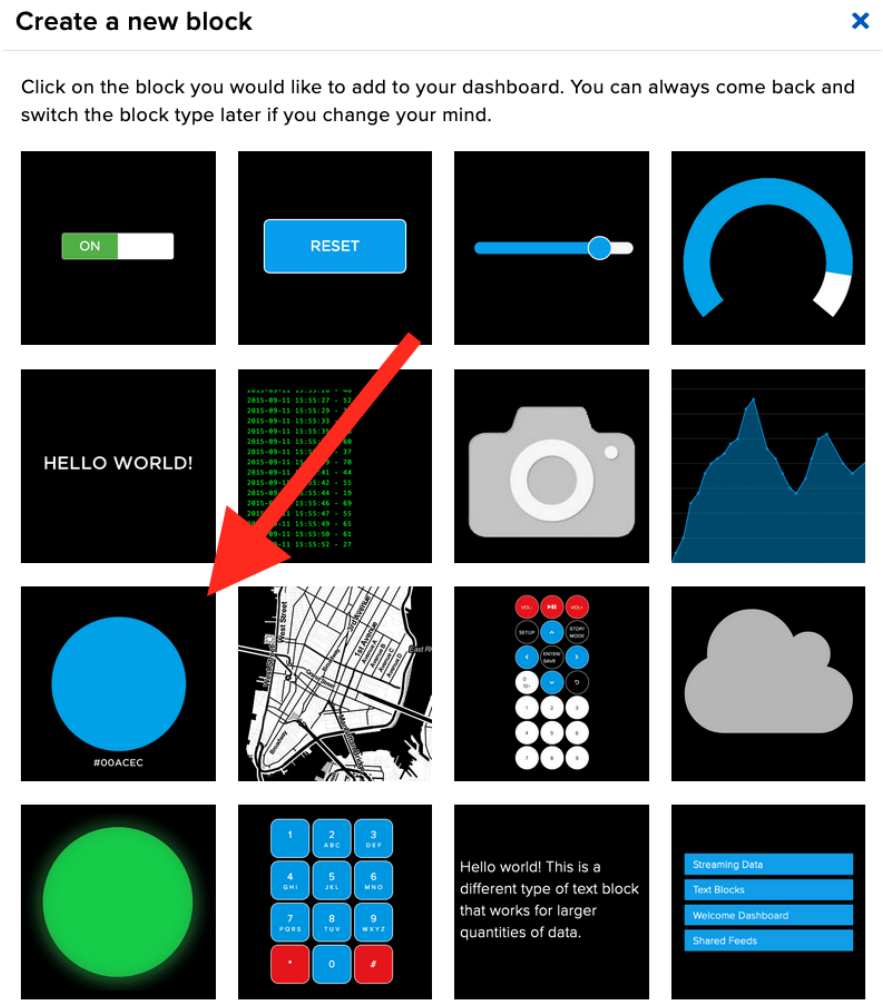
Adafruit IO Feeds and Dashboard

The first thing you'll need to do, is head over to Adafruit IO (https://adafru.it/fsU) and make sure your account is set up.

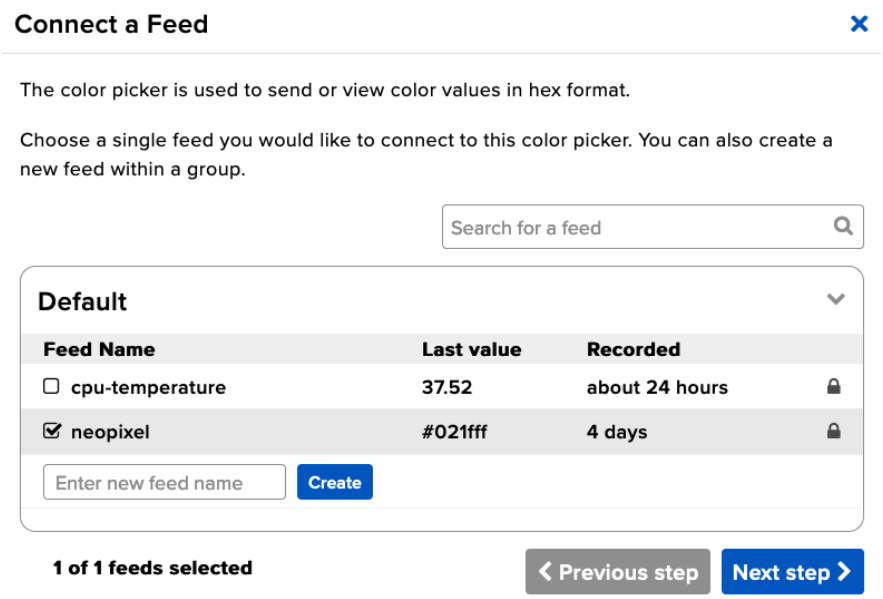
Then, you need to create two feeds (https://adafru.it/f5k) called **neopixel** and **random**. These are case sensitive!

Next, you'll create a dashboard (https://adafru.it/Fm7) for the NeoPixel Color Picker. You can name the dashboard whatever you like.

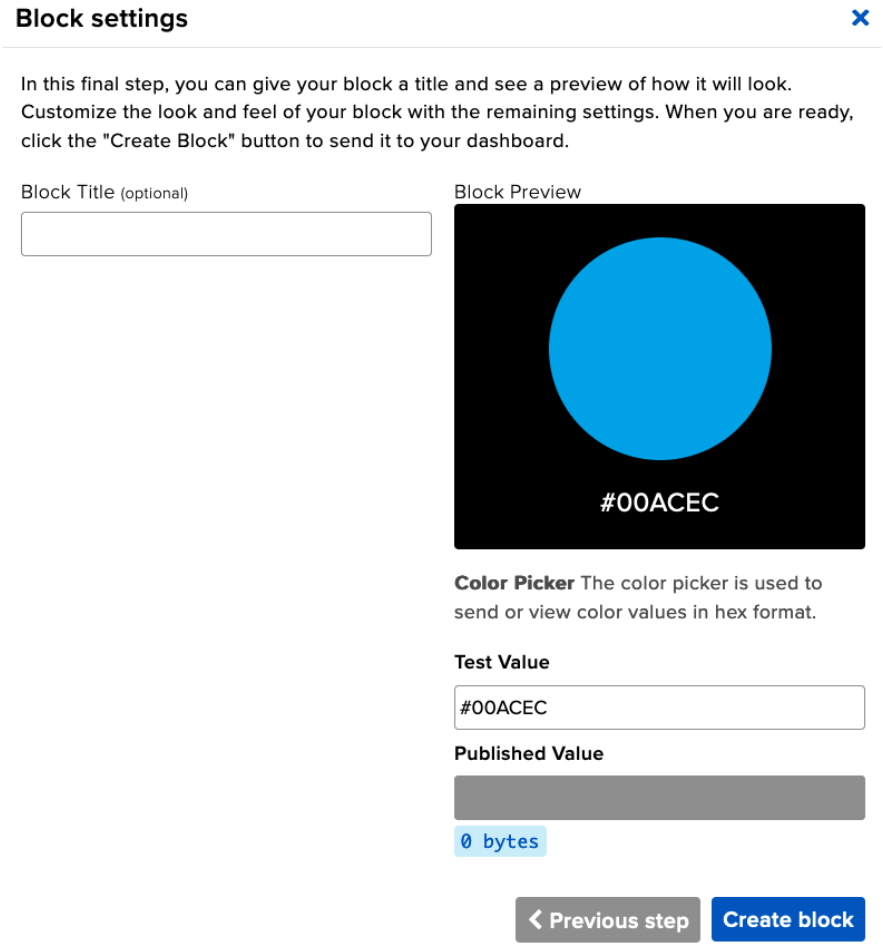
Once the dashboard is created, you'll want to add a color picker block (https://adafru.it/DZe). The color picker block is highlighted by a red arrow in the image below.



Once you choose the color picker block, you'll need to connect a feed to it. Check the box next to **neopixel**.



Finally, a Block Settings page will come up. You can add an optional block title here. Then you press **Create Block**.



The dashboard should look something like the following.

Now that things are set up on the Adafruit IO end, you can continue on to the code on your microcontroller!

Adafruit IO settings.toml

This example requires you to provide your Wi-Fi credentials, and your Adafruit IO username and key. To do this, you'll want to create a **settings.toml** file on your **CIRCUITPY** drive.

To obtain your Adafruit IO key, follow the initial steps on this page (https://adafru.it/XbK).

Your **settings.toml** file should be structured in a certain way, and contain all the necessary information. Follow these instructions to create your settings.toml file (https://adafru.it/18f9).

Adafruit IO Example Code

To run this example, you need to first install the NeoPixel, Adafruit IO, and Adafruit MiniMQTT libraries into the **lib** folder on your board. Then you need to update **code.py** with the example script.

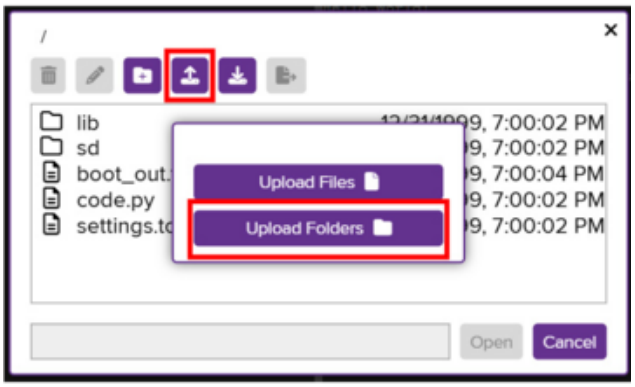
In the example below, click the **Download Project Bundle** button below to download the necessary libraries and the **code.py** file in a zip file. Extract the contents of the zip file. You'll see a **code.py** file and **/lib** folder.

```
# SPDX-FileCopyrightText: 2021 Ladyada for Adafruit Industries
# SPDX-FileCopyrightText: 2022 Kattni Rembor for Adafruit Industries
# SPDX-License-Identifier: MIT
import time
import ssl
import os
from random import randint
import microcontroller
import socketpool
import wifi
import board
import neopixel
import adafruit_minimqtt.adafruit_minimqtt as MQTT
from adafruit_io.adafruit_io import IO_MQTT
```

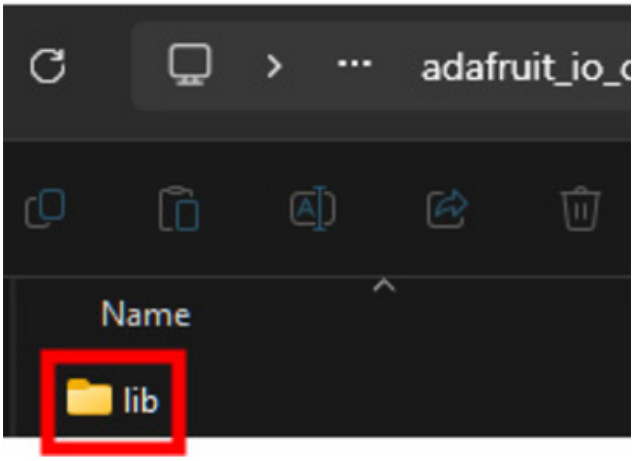
```
# WiFi
try:
    print("Connecting to %s" % os.getenv("CIRCUITPY_WIFI_SSID"))
    wifi.radio.connect(os.getenv("CIRCUITPY_WIFI_SSID"),
os.getenv("CIRCUITPY_WIFI_PASSWORD"))
    print("Connected to %s!" % os.getenv("CIRCUITPY_WIFI_SSID"))
# Wi-Fi connectivity fails with error messages, not specific errors, so this except
is broad.
except Exception as e:  # pylint: disable=broad-except
    print("Failed to connect to WiFi. Error:", e, "\nBoard will hard reset in 30
seconds.")
    time.sleep(30)
    microcontroller.reset()

# Initialise NeoPixel
pixel = neopixel.NeoPixel(board.NEOPIXEL, 1, brightness=0.3)


# Define callback functions which will be called when certain events happen.
def connected(client):
    print("Connected to Adafruit IO!  Listening for NeoPixel changes...")
    # Subscribe to Adafruit IO feed called "neopixel"
    client.subscribe("neopixel")


def message(client, feed_id, payload):  # pylint: disable=unused-argument
    print("Feed {0} received new value: {1}".format(feed_id, payload))
    if feed_id == "neopixel":
        pixel.fill(int(payload[1:], 16))


# Create a socket pool
pool = socketpool.SocketPool(wifi.radio)

# Initialize a new MQTT Client object
mqtt_client = MQTT.MQTT(
    broker="io.adafruit.com",
    username=os.getenv("ADAFRUIT_AIO_USERNAME"),
    password=os.getenv("ADAFRUIT_AIO_KEY"),
    socket_pool=pool,
    ssl_context=ssl.create_default_context(),
)

# Initialize Adafruit IO MQTT "helper"
io = IO_MQTT(mqtt_client)

# Set up the callback methods above
io.on_connect = connected
io.on_message = message

timestamp = 0
while True:
    try:
        # If Adafruit IO is not connected...
        if not io.is_connected:
            # Connect the client to the MQTT broker.
            print("Connecting to Adafruit IO...")
            io.connect()

        # Explicitly pump the message loop.
        io.loop()
        # Obtain the "random" value, print it and publish it to Adafruit IO every
10 seconds.
        if (time.monotonic() - timestamp) >= 10:
            random_number = "{}".format(randint(0, 255))
            print("Current 'random' number: {}".format(random_number))
            io.publish("random", random_number)
            timestamp = time.monotonic()
```

```
    # Adafruit IO fails with internal error types and WiFi fails with specific
messages.
    # This except is broad to handle any possible failure.
    except Exception as e:  # pylint: disable=broad-except
        print("Failed to get or send data, or connect. Error:", e,
              "\nBoard will hard reset in 30 seconds.")
        time.sleep(30)
        microcontroller.reset()
```

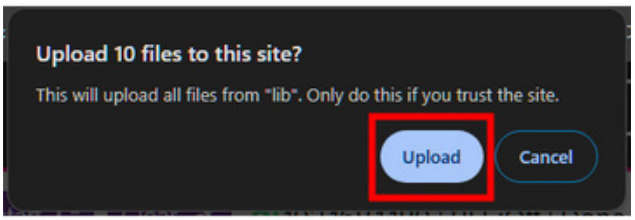
Update the /lib Folder



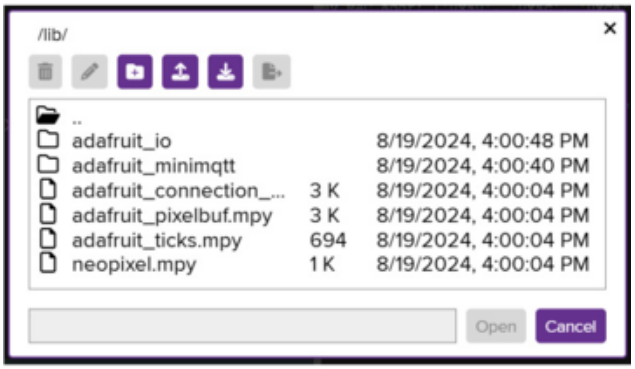
In the editor window in your browser, click the **Open** button to view the file dialog. Then, click the **Upload** button and select **Upload Folders**.



Navigate to the project bundle that you downloaded and select the **/lib folder**.

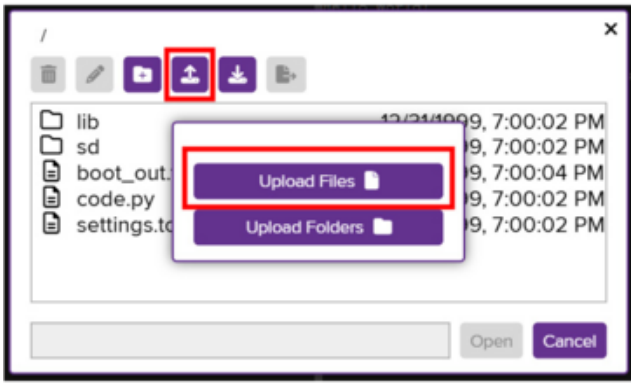


You'll be asked if you want to upload the /**lib** folder from the Project Bundle. Click **Upload**.

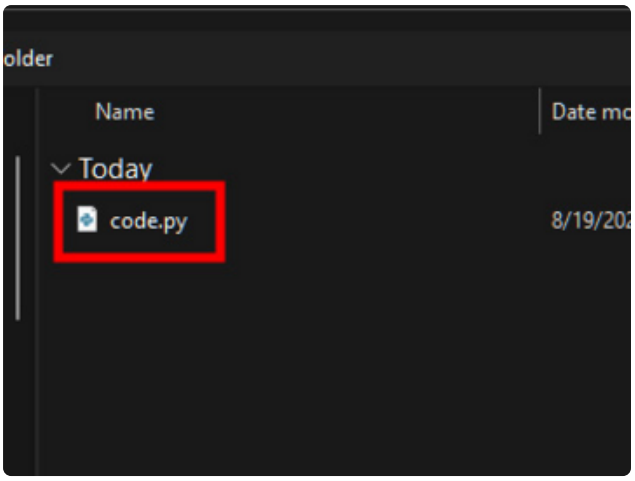


After the upload finishes, you can open the **lib** folder to view the library files required for the Adafruit IO example.

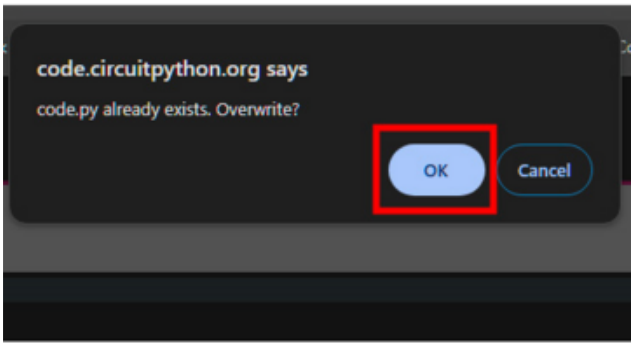
Update **code.py**



In the editor window in your browser, click the **Open** button to view the file dialog. Then, click the **Upload** button and select **Upload Files**.



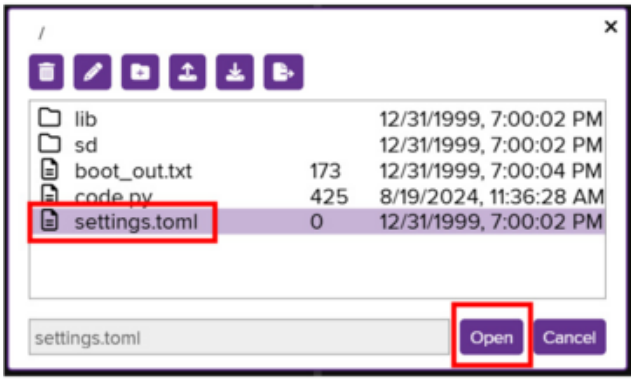
Navigate to the project bundle that you downloaded and select the **code.py** file.



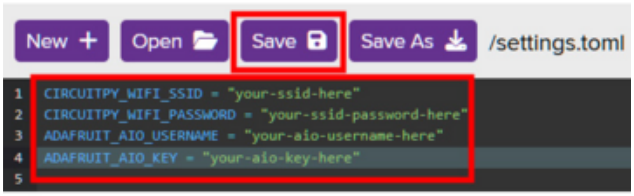
You'll be asked if you want to overwrite the previous **code.py** with the new **code.py** file from the Project Bundle. Click **OK**.

Update Your **settings.toml** File

Remember to add your **settings.toml** file as described earlier in this page. You'll need to include your `ADAFRUIT_AIO_USERNAME`, `ADAFRUIT_AIO_KEY`, `CIRCUITPY_WIFI_SSID` and `CIRCUITPY_WIFI_PASSWORD` in the file.



You can edit the file manually in the USB code editor by clicking **Open**, selecting **settings.toml** and clicking **Open** at the bottom of the dialog box.

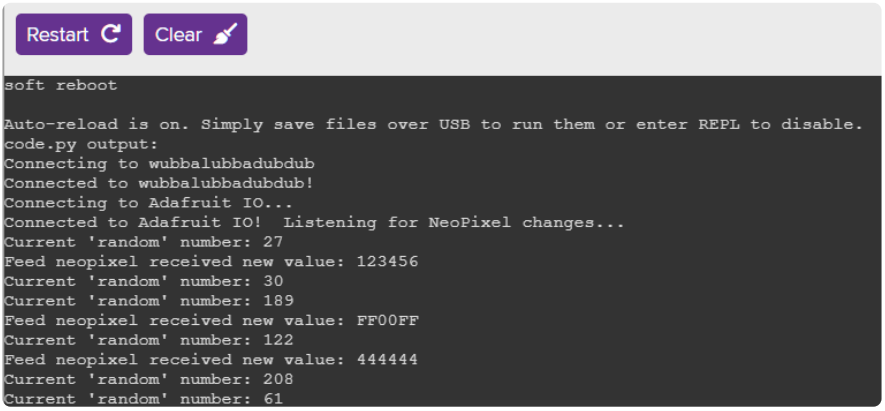


With **settings.toml** open in the editor, you can add your WiFi and Adafruit IO credentials:

```
CIRCUITPY_WIFI_SSID = "your-ssid-here"
CIRCUITPY_WIFI_PASSWORD = "your-ssid-password-here"
ADAFRUIT_AIO_USERNAME = "your-aio-username-here"
ADAFRUIT_AIO_KEY = "your-aio-key-here"
```

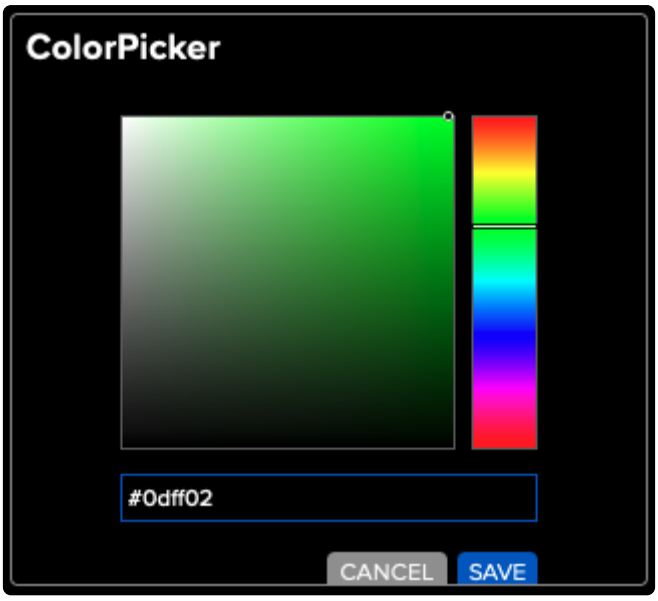
Once your credentials are entered, click **Save** above the editor to save your changes to **settings.toml**.

Once everything is saved to the board, **Restart** the Serial Console to run the new **code.py**. You'll see the connection info and current readings printed out in the console.



NeoPixel Color Change

To change the color of the NeoPixel, go to the NeoPixel Adafruit IO dashboard you created at the beginning, and click on the colored circle in the ColorPicker block. It will bring up the following.



You can move the dot in the box around, and the slider line across the gradient to choose the perfect color. Choose a new color and click **SAVE**.

The NeoPixel color will update, and you will see the new value printed to the serial console, as shown below.

```
Current 'random' number: 4
Current 'random' number: 68
Feed neopixel received new value: #cc02ff
```

Code Walkthrough

This example contains three `try`/`except` blocks. These are included where the code is likely to fail due to WiFi or Adafruit IO connection failures. WiFi can be finicky, and without these code blocks, if the connection was lost, the code would crash. Instead, it is designed to reset the board and start the code over again to reestablish the connection, regardless of the cause. This ensures your code will continue running. The details of these blocks are explained below.

First you import all of the necessary modules and libraries.

```
import time
import ssl
import os
from random import randint
import socketpool
import wifi
import board
import neopixel
import adafruit_minimqtt.adafruit_minimqtt as MQTT
from adafruit_io.adafruit_io import IO_MQTT
```

Note that if a settings.toml file is not present on your CIRCUITPY drive, the code will fail to run, and you will receive an error in the serial console. Add a settings.toml file to your CIRCUITPY drive to resolve this error.

The WiFi attempts to connect, and prints the status to the serial console. If it connects successfully, the code continues onto the NeoPixel set up.

```
try:
    print("Connecting to %s" % os.getenv("CIRCUITPY_WIFI_SSID"))
    wifi.radio.connect(os.getenv("CIRCUITPY_WIFI_SSID"),
os.getenv("CIRCUITPY_WIFI_PASSWORD"))
    print("Connected to %s!" % os.getenv("CIRCUITPY_WIFI_SSID"))
```

If the WiFi connection is not successful, the error will be printed to the serial console, and the board will hard reset after 30 seconds.

```
except Exception as e:  # pylint: disable=broad-except
    print("Failed to connect to WiFi. Error:", e, "\nBoard will hard reset in 30
seconds.")
    time.sleep(30)
    microcontroller.reset()
```

Once the WiFi successfully connects, the NeoPixel object is initiated.

```
pixel = neopixel.NeoPixel(board.NEOPIXEL, 1, brightness=0.3)
```

Following that are two callback methods. For more details, check out this guide (https://adafru.it/FGB). The `connected` method subscribes to the neopixel feed on Adafruit IO. The `message` callback checks for updates to the neopixel feed, and turns the pixel the color from the feed.

```
def connected(client):
    print("Connected to Adafruit IO!  Listening for NeoPixel changes...")
    # Subscribe to Adafruit IO feed called "neopixel"
    client.subscribe("neopixel")


# pylint: disable=unused-argument
def message(client, feed_id, payload):
    print("Feed {0} received new value: {1}".format(feed_id, payload))
    if feed_id == "neopixel":
        pixel.fill(int(payload[1:], 16))
```

You create a socket pool, use that to initialise the new MQTT Client object, and use that to initialise the Adafruit IO MQTT "helper".

```
pool = socketpool.SocketPool(wifi.radio)
```

```
mqtt_client = MQTT.MQTT(
    broker="io.adafruit.com",
    username=os.getenv("ADAFRUIT_AIO_USERNAME"),
    password=os.getenv("ADAFRUIT_AIO_KEY"),
    socket_pool=pool,
    ssl_context=ssl.create_default_context(),
)

io = IO_MQTT(mqtt_client)
```

You set up the callback methods mentioned above.

```
io.on_connect = connected
io.on_message = message
```

Next, you attempt to connect the client to the MQTT broker. If connection is successful, the code continues on to the `timestamp`.

```
try:
    io.connect()
```

If the MQTT broker connection is not successful, the error is printed to the serial console, and the board will hard reset after 30 seconds.

```
except Exception as e:
    print("Failed to connect to Adafruit IO. Error:", e, "\nBoard will hard reset
in 30 seconds.")
    time.sleep(30)
    microcontroller.reset()
```

Once the broker is connected, you set the `timestamp` to `0` immediately before the loop.

```
timestamp = 0
```

Inside the loop, you attempt to do two things. You first explicitly poll the message loop. Check out this guide (https://adafru.it/YF7) for more details on that.

```
while True:
    try:
        io.loop()
```

Second, you have a block of code that runs every 10 seconds. Inside, you obtain a "random" value between 0-255 inclusive, print it to the serial console, and publish it to an Adafruit IO feed. Finally, you reset timestamp so the block of code knows when another 10 seconds has passed, and runs again.

```
[...]
        if (time.monotonic() - timestamp) &gt;= 10:
            random_number = "{}".format(randint(0, 255))
```

```
            print("Current 'random' number: {}".format(random_number))
            io.publish("random", random_number)
            timestamp = time.monotonic()
```

If at any time WiFi or Adafruit IO disconnects, the code will print the error to the serial console, and the board will hard reset after 30 seconds.

```
[...]
    except Exception as e:
        print("Failed to get or send data, or connect. Error:", e,
              "\nBoard will hard reset in 30 seconds.")
        time.sleep(30)
        microcontroller.reset()
```

That's all there is to using CircuitPython and Adafruit IO to send data to Adafruit IO, and receive data from it!

Arduino IDE Setup

Install Arduino IDE

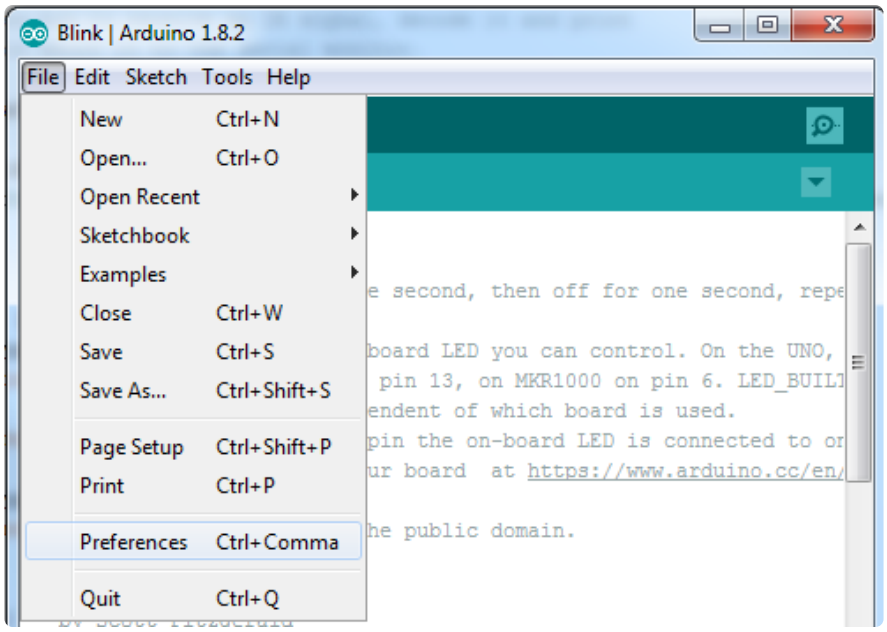
The first thing you will need to do is to download the latest release of the Arduino IDE. You will need to be using **version 1.8** or higher for this guide

Arduino IDE Download

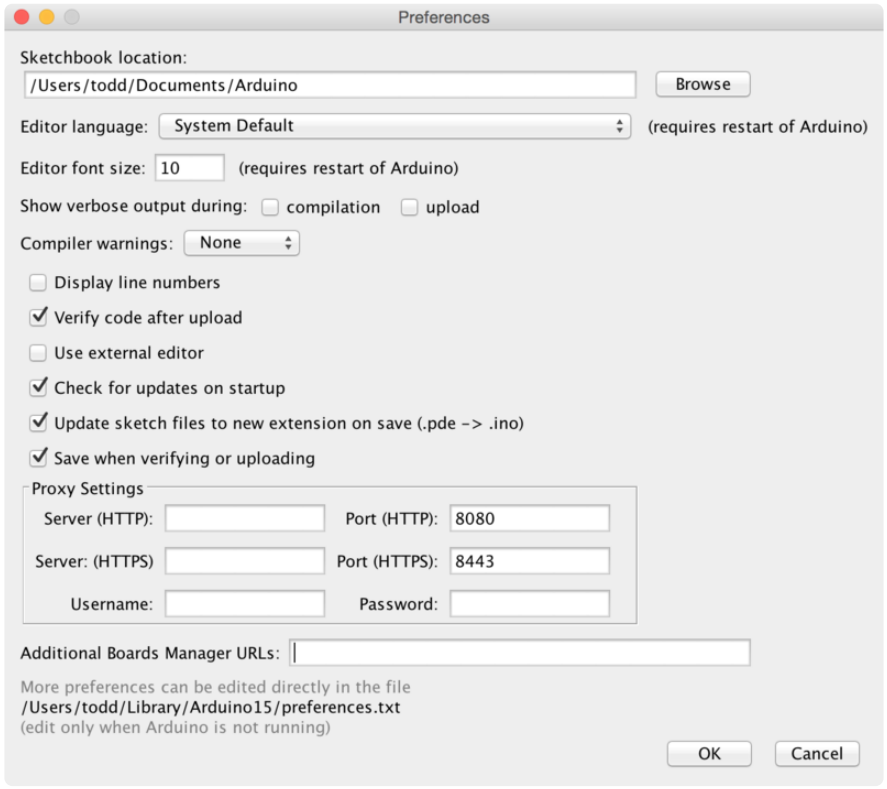
https://adafru.it/f1P

Install ESP32 Board Support Package

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



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

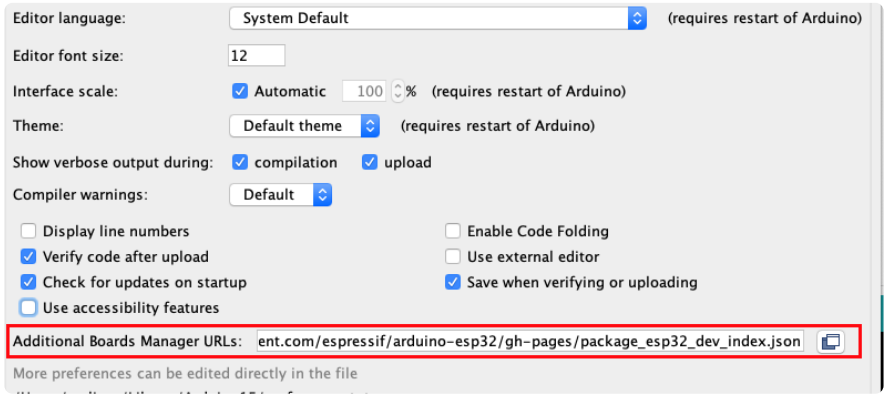


We will be adding a URL to the new **Additional Boards Manager URLs** option. The list of URLs is comma separated, and you will only have to add each URL once. New Adafruit boards and updates to existing boards will automatically be picked up by the Board Manager each time it is opened. The URLs point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

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

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

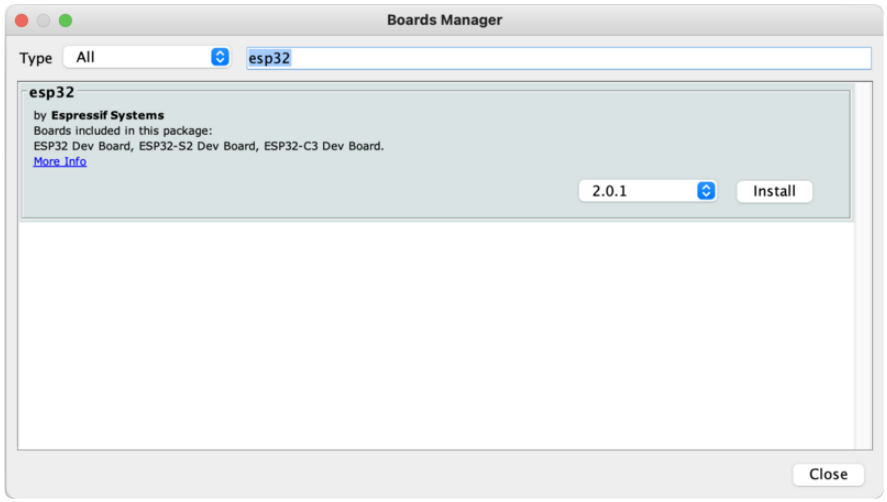
```
https://raw.githubusercontent.com/espressif/arduino-esp32/gh-pages/package_esp32_dev_index.json
```



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.

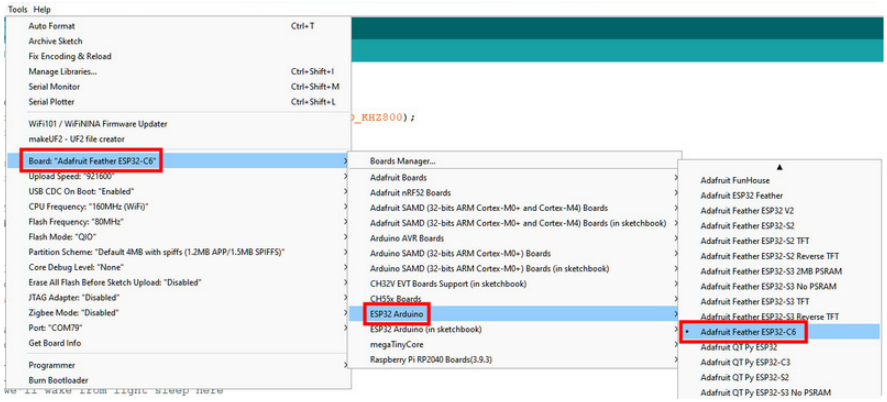
The next step is to actually install the Board Support Package (BSP). Go to the **Tools → Board → Board Manager** submenu. A dialog should come up with various BSPs. Search for **esp32**.



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 boards.

Look for the board called Adafruit Feather ESP32-C6.

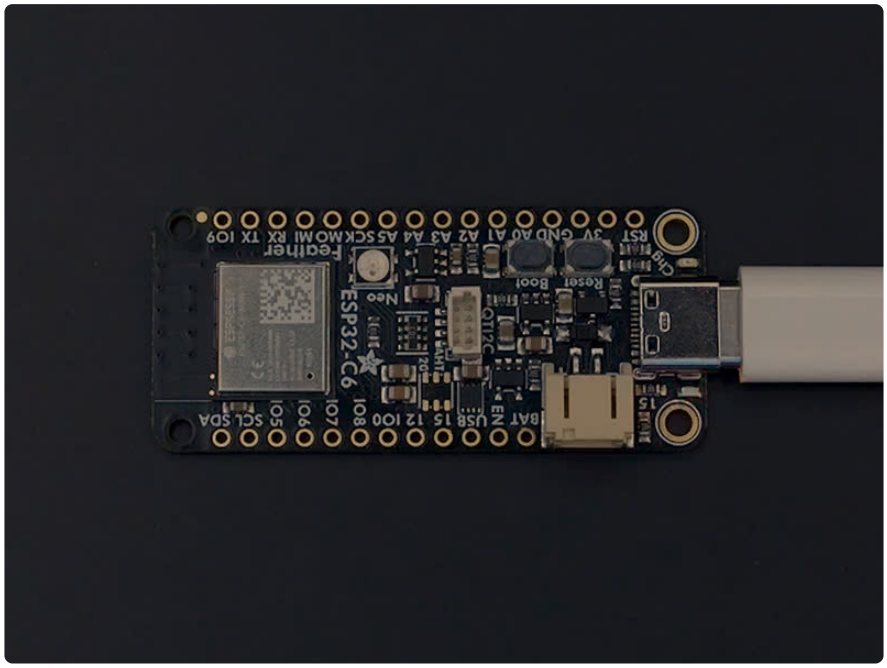


The upload speed can be changed: faster speed makes uploads take less time but sometimes can cause upload issues. **921600** should work fine, but if you're having issues, you can drop down lower.

Blink

The first and most basic program you can upload to your Arduino is the classic Blink sketch. This takes something on the board and makes it, well, blink! On and off. It's a great way to make sure everything is working and you're uploading your sketch to the right board and right configuration.

When all else fails, you can always come back to Blink!



Pre-Flight Check: Get Arduino IDE & Hardware Set Up

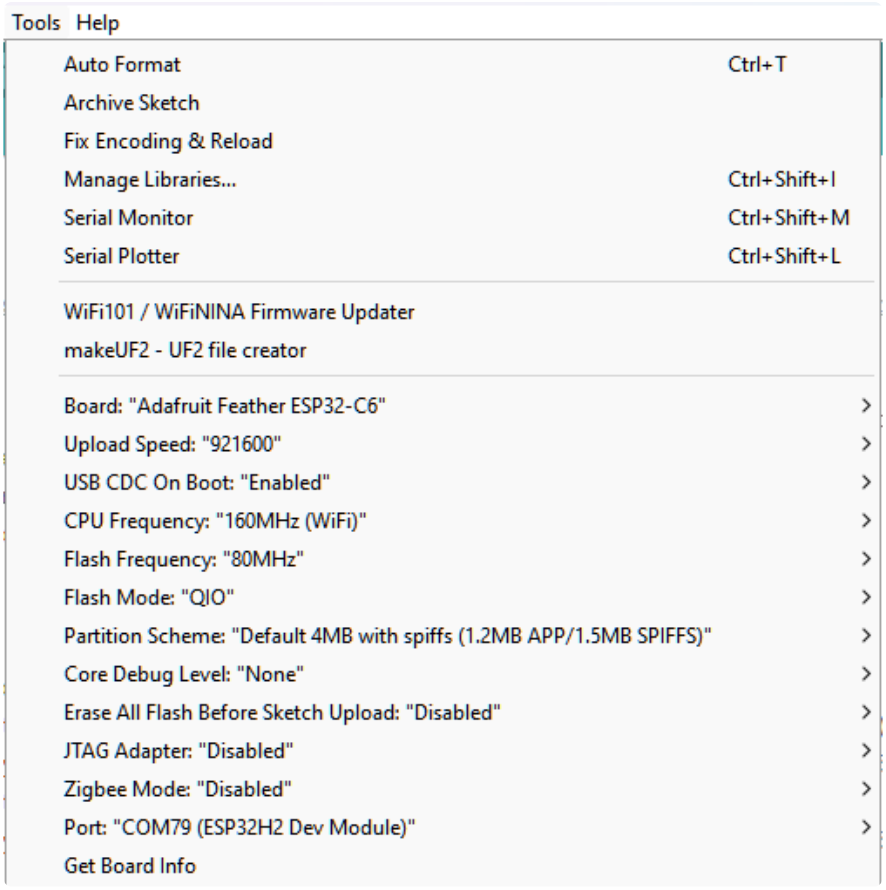
This lesson assumes you have Arduino IDE set up. This is a generalized checklist, some elements may not apply to your hardware. If you haven't yet, check the previous steps in the guide to make sure you:

- **Install the very latest Arduino IDE for Desktop** (not all boards are supported by the Web IDE so we don't recommend it)
- **Install any board support packages (BSP) required for your hardware.** Some boards are built in defaults on the IDE, but lots are not! You may need to install plug-in support which is called the BSP.
- **Get a Data/Sync USB cable for connecting your hardware.** A significant amount of problems folks have stem from not having a USB cable with data pins. Yes, these cursed cables roam the land, making your life hard. If you find a USB cable that doesn't work for data/sync, throw it away immediately! There is no need to keep it around, cables are very inexpensive these days.
- **Install any drivers required** - If you have a board with a FTDI or CP210x chip, you may need to get separate drivers. If your board has native USB, it probably doesn't need anything. After installing, reboot to make sure the driver sinks in.
- **Connect the board to your computer.** If your board has a power LED, make sure its lit. Is there a power switch? Make sure its turned On!

Start up Arduino IDE and Select Board/Port

OK now you are prepared! Open the Arduino IDE on your computer. Now you have to tell the IDE what board you are using, and how you want to connect to it.

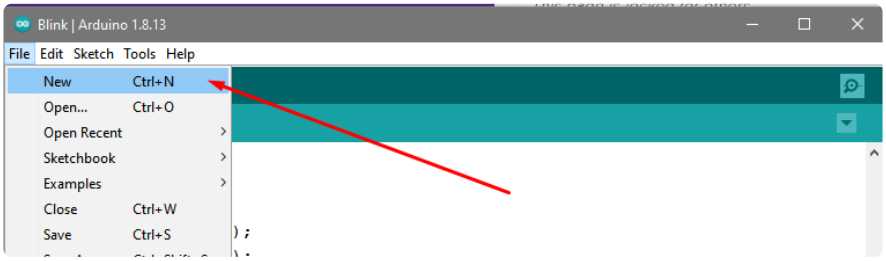
In the IDE find the **Tools** menu. You will use this to select the board. If you switch boards, you must switch the selection! So always double-check before you upload code in a new session.



If you have any issues accessing the Serial Monitor, make sure that USB CDC On Boot is set to Enabled.

New Blink Sketch

OK lets make a new blink sketch! From the **File** menu, select **New**



Then in the new window, copy and paste this text:

```
int led = LED_BUILTIN;

void setup() {
  // Some boards work best if we also make a serial connection
  Serial.begin(115200);
```

```
  // set LED to be an output pin
  pinMode(led, OUTPUT);
}

void loop() {
  // Say hi!
  Serial.println("Hello!");

  digitalWrite(led, HIGH);   // turn the LED on (HIGH is the voltage level)
  delay(500);                // wait for a half second
  digitalWrite(led, LOW);    // turn the LED off by making the voltage LOW
  delay(500);                // wait for a half second
}
```

Note that in this example, we are not only blinking the LED but also printing to the Serial monitor, think of it as a little bonus to test the serial connection.

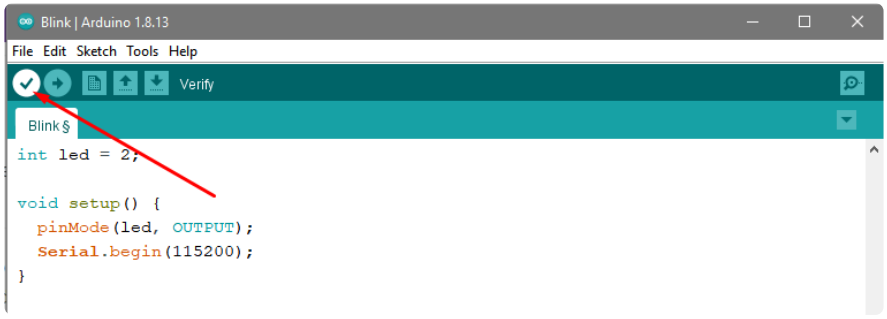
One note you'll see is that we reference the LED with the constant `LED_BUILTIN` rather than a number. That's because, historically, the built in LED was on pin 13 for Arduinos. But in the decades since, boards don't always have a pin 13, or maybe it could not be used for an LED. So the LED could have moved to another pin. It's best to use `LED_BUILTIN` so you don't get the pin number confused!

The red LED on the Feather ESP32-C6 is available as `LED_BUILTIN`, as well as `15`.

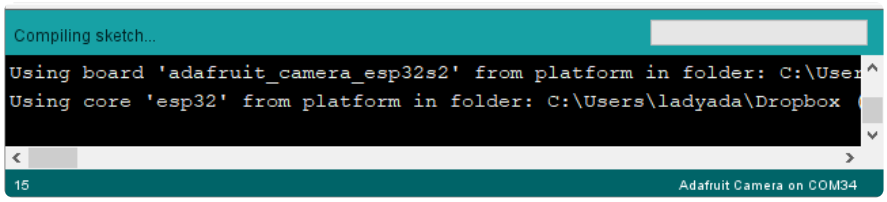
Verify (Compile) Sketch

OK now you can click the Verify button to convert the sketch into binary data to be uploaded to the board.

Note that Verifying a sketch is the same as Compiling a sketch - so we will use the words interchangeably

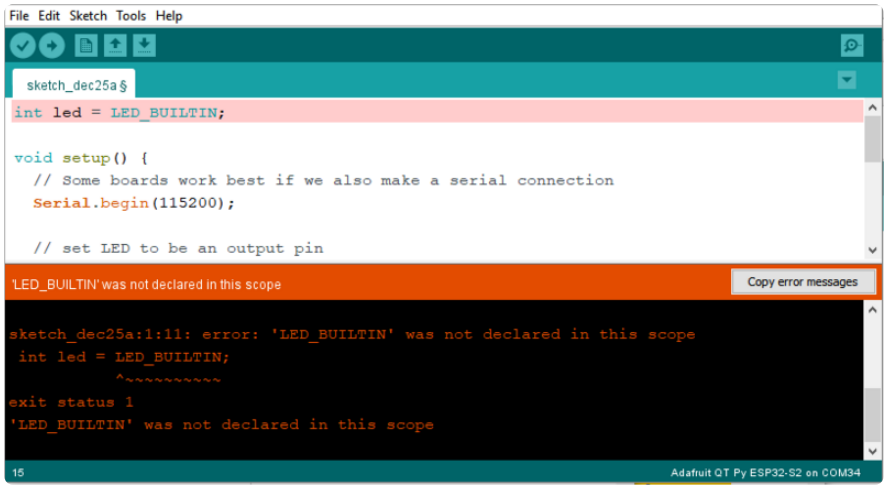


During verification/compilation, the computer will do a bunch of work to collect all the libraries and code and the results will appear in the bottom window of the IDE

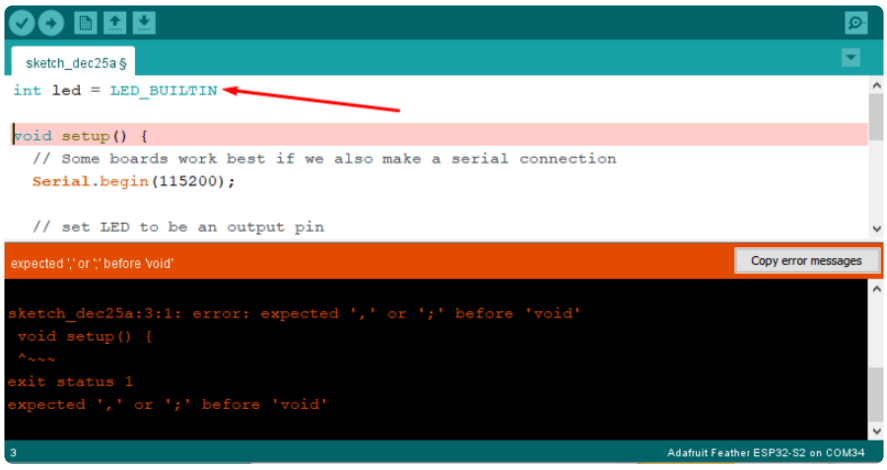


If something went wrong with compilation, you will get red warning/error text in the bottom window letting you know what the error was. It will also highlight the line with an error

For example, here I had the wrong board selected - and the selected board does not have a built in LED!

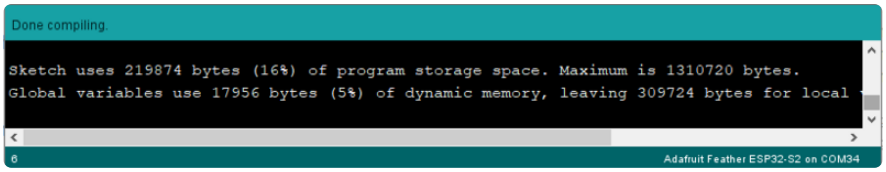


Here's another common error, in my haste I forgot to add a `;` at the end of a line. The compiler warns me that it's looking for one - note that the error is actually a few lines up!



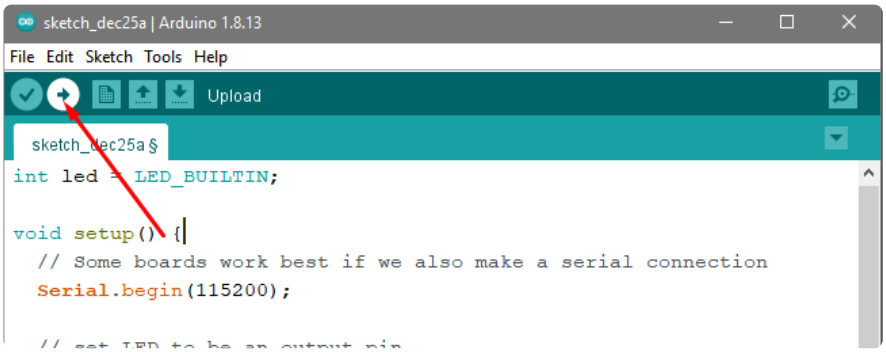
Turning on detailed compilation warnings and output can be very helpful sometimes - Its in Preferences under "Show Verbose Output During:" and check the Compilation button. If you ever need to get help from others, be sure to do this and then provide all the text that is output. It can assist in nailing down what happened!

On success you will see something like this white text output and the message **Done compiling.** in the message area.



Upload Sketch

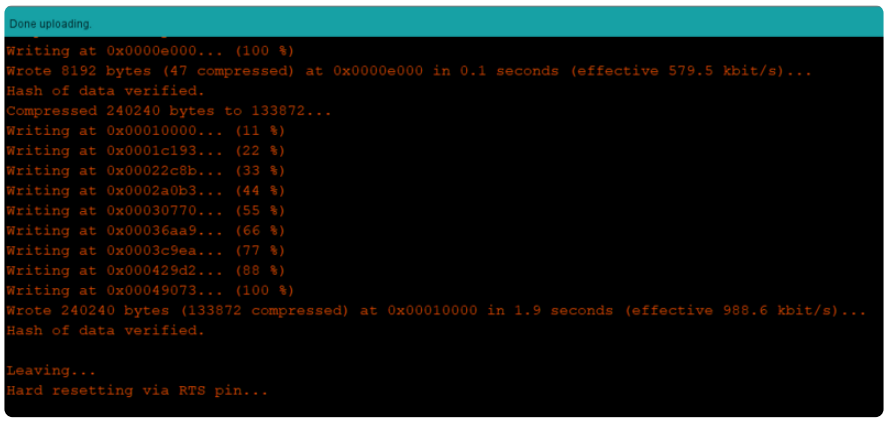
Once the code is verified/compiling cleanly you can upload it to your board. Click the **Upload** button



The IDE will try to compile the sketch again for good measure, then it will try to connect to the board and upload a the file.

This is actually one of the hardest parts for beginners because it's where a lot of things can go wrong.

However, lets start with what it looks like on success! Here's what your board upload process looks like when it goes right:



Often times you will get a warning like this, which is kind of vague:

`No device found on COM66` (or whatever port is selected)
`An error occurred while uploading the sketch`

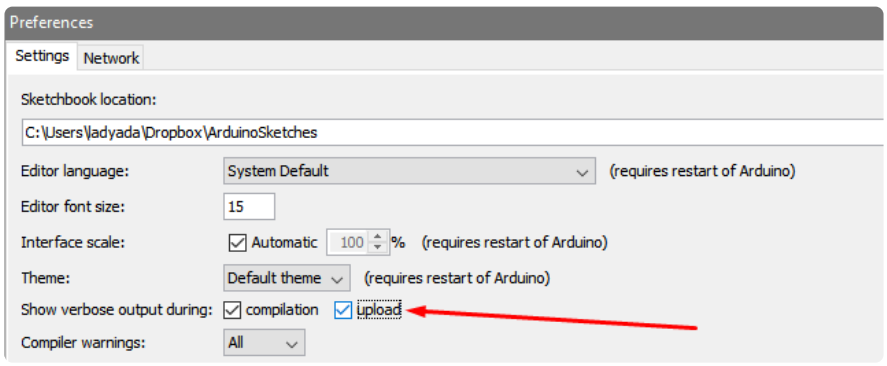
This could be a few things.

First up, **check again that you have the correct board selected!** Many electronics boards have very similar names or look, and often times folks grab a board different from what they thought.

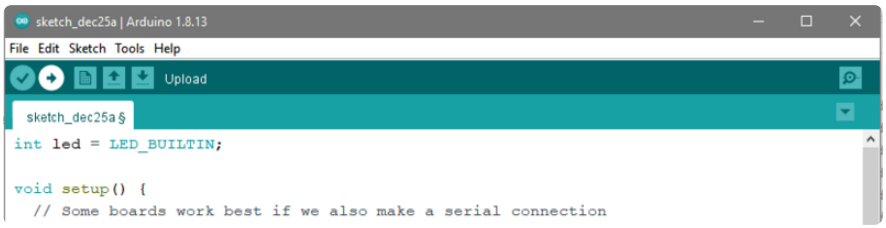
Second, **make sure you selected the right port!** If you have the wrong port or no port selected, Arduino doesn't know where to look for your board.

If both of those are correct, the next step is to enable verbose upload messages.

Before continuing we really really suggest turning on **Verbose Upload** messages, it will help in this process because you will be able to see what the IDE is trying to do. It's a checkbox in the **Preferences** menu.



Now you can try uploading again!

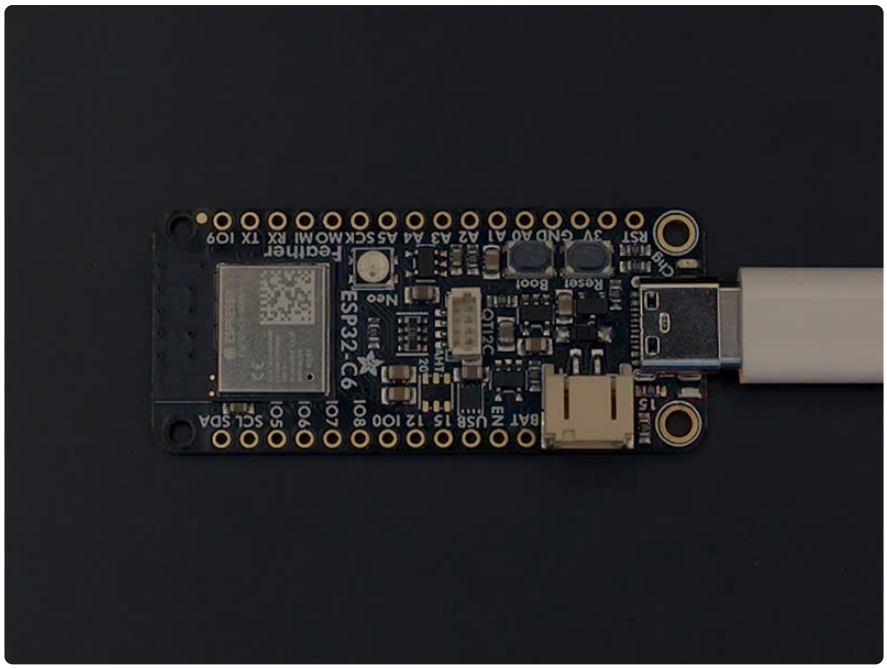


This time, you should have success!

After uploading this way, be sure to **click the reset button** - it sort of makes sure that the board got a good reset and will come back to life nicely.

Finally, a Blink!

OK it was a journey but now we're here and you can enjoy your blinking LED. Next up, try to change the delay between blinks and re-upload. It's a good way to make sure your upload process is smooth and practiced.



I2C

A lot of sensors, displays, and devices can connect over I2C. I2C is a 2-wire 'bus' that allows multiple devices to all connect on one set of pins so it's very convenient for wiring!

When using your board, you'll probably want to connect up I2C devices, and it can be a little tricky the first time. The best way to debug I2C is go through a checklist and then perform an I2C scan

Common I2C Connectivity Issues

- **Have you connected four wires (at a minimum) for each I2C device?** Power the device with whatever is the logic level of your microcontroller board (probably 3.3V), then a ground wire, and a SCL clock wire, and and a SDA data wire.
- **If you're using a STEMMA QT board - check if the power LED is lit.** It's usually a green LED to the left side of the board.
- **Does the STEMMA QT/I2C port have switchable power or pullups?** To reduce power, some boards have the ability to cut power to I2C devices or the pullup resistors. Check the documentation if you have to do something special to turn on the power or pullups.
- **If you are using a DIY I2C device, do you have pullup resistors?** Many boards do not have pullup resistors built in and they are required! We suggest any common 2.2K to 10K resistors. You'll need two: one each connects from SDA to positive power, and SCL to positive power. Again, positive power (a.k.a VCC, VDD or V+) is often 3.3V
- **Do you have an address collision?** You can only have one board per address. So you cannot, say, connect two AHT20's to one I2C port because they have the same address and will interfere. Check the sensor or documentation for the address. Sometimes there are ways to adjust the address.
- **Does your board have multiple I2C ports?** Historically, boards only came with one. But nowadays you can have two or even three! This can help solve the "hey, but what if I want two devices with the same address" problem: just put one on each bus.
- **Are you hot-plugging devices?** I2C does not support dynamic re-connection, you cannot connect and disconnect sensors as you please. They should all be connected on boot and not change. (Only exception is if you're using a hot-plug assistant but that'll cost you (http://adafru.it/5159)).
- **Are you keeping the total bus length reasonable?** I2C was designed for maybe 6" max length. We like to push that with plug-n-play cables, but really please keep them as short as possible! (Only exception is if you're using an active bus extender (http://adafru.it/4756)).

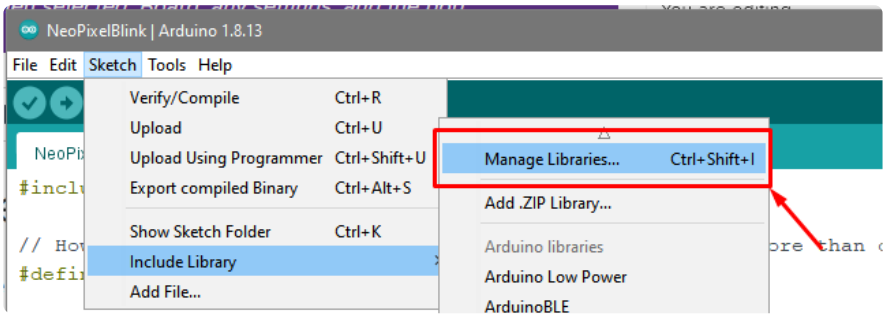
The Feather ESP32-C6 has a `NEOPIXEL_I2C_POWER` pin that must be pulled `HIGH` to enable power to the STEMMA QT port. This is done automatically in the board support package in Arduino, but if you find you're having issues definitely double check that it is pulled `HIGH`!

Perform an I2C scan!

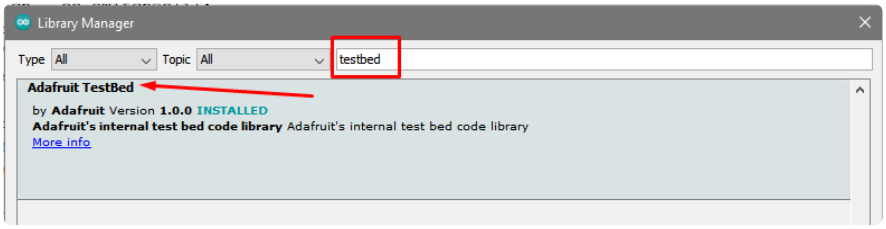
Install TestBed Library

To scan I2C, the Adafruit TestBed library is used. This library and example just makes the scan a little easier to run because it takes care of some of the basics. You will

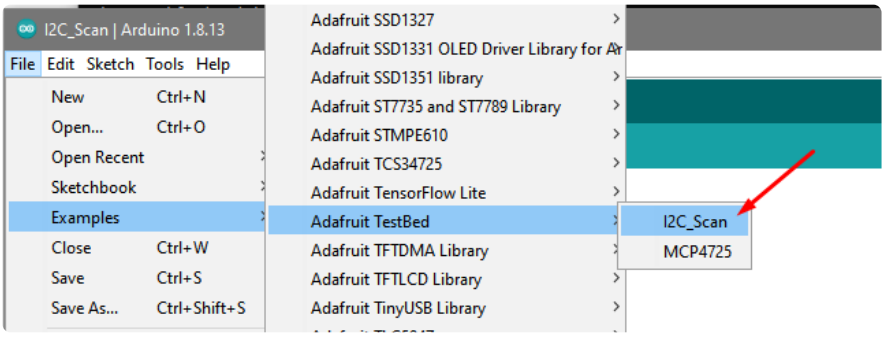
need to add support by installing the library. Good news: it is very easy to do it. Go to the **Arduino Library Manager.**



Search for **TestBed** and install the **Adafruit TestBed** library



Now open up the I2C Scan example



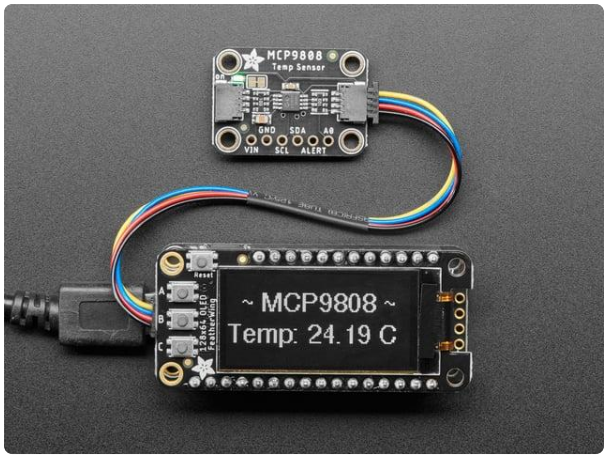

```
// SPDX-FileCopyrightText: 2023 Carter Nelson for Adafruit Industries
//
// SPDX-License-Identifier: MIT
// -------------------------------------
// i2c_scanner
//
// Modified from https://playground.arduino.cc/Main/I2cScanner/
// -------------------------------------

#include <Wire.h>

// Set I2C bus to use: Wire, Wire1, etc.
#define WIRE Wire

void setup() {
  WIRE.begin();

  Serial.begin(9600);
  while (!Serial)
     delay(10);
  Serial.println("\nI2C Scanner");
}
```

```
void loop() {
  byte error, address;
  int nDevices;

  Serial.println("Scanning...");

  nDevices = 0;
  for(address = 1; address < 127; address++ )
  {
    // The i2c_scanner uses the return value of
    // the Write.endTransmisstion to see if
    // a device did acknowledge to the address.
    WIRE.beginTransmission(address);
    error = WIRE.endTransmission();

    if (error == 0)
    {
      Serial.print("I2C device found at address 0x");
      if (address<16)
        Serial.print("0");
      Serial.print(address,HEX);
      Serial.println("  !");

      nDevices++;
    }
    else if (error==4)
    {
      Serial.print("Unknown error at address 0x");
      if (address<16)
        Serial.print("0");
      Serial.println(address,HEX);
    }
  }
  if (nDevices == 0)
    Serial.println("No I2C devices found\n");
  else
    Serial.println("done\n");

  delay(5000);           // wait 5 seconds for next scan
}
```

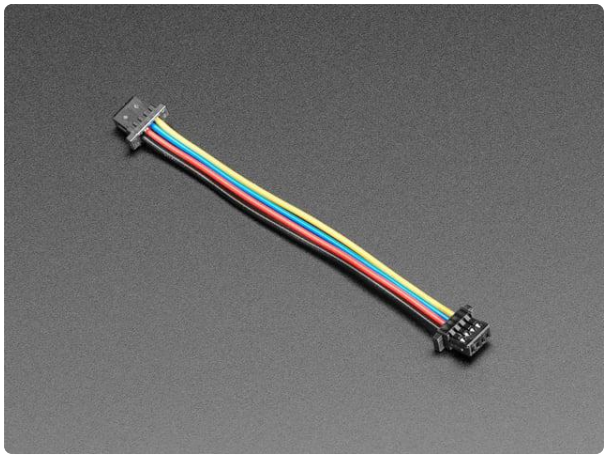
Wire up I2C device

While the examples here will be using the Adafruit MCP9808 (http://adafru.it/5027), a high accuracy temperature sensor, the overall process is the same for just about any I2C sensor or device.

The first thing you'll want to do is get the sensor connected so your board has I2C to talk to.



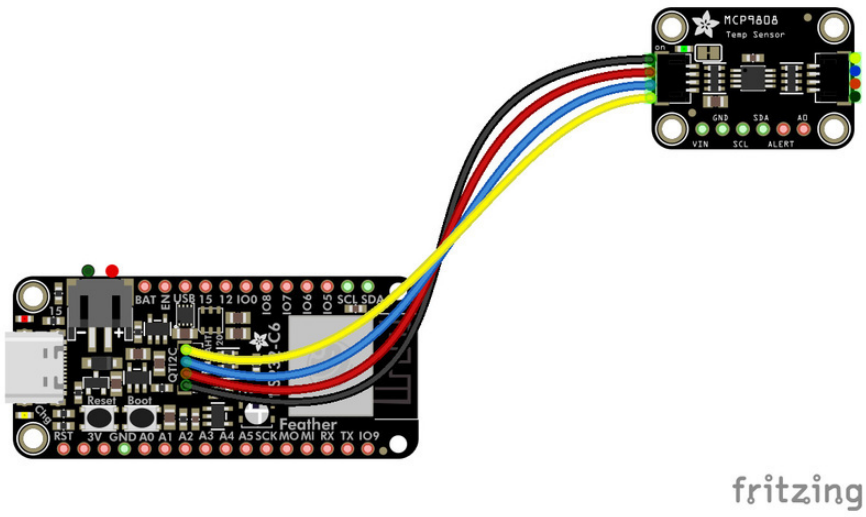
[Adafruit MCP9808 High Accuracy I2C Temperature Sensor Breakout]
The MCP9808 digital temperature sensor is one of the more accurate/precise we've ever seen, with a typical accuracy of ±0.25°C over the sensor's -40°C to...
https://www.adafruit.com/product/5027



[STEMMA QT / Qwiic JST SH 4-Pin Cable - 50mm Long]
This 4-wire cable is 50mm / 1.9" long and fitted with JST SH female 4-pin connectors on both ends. Compared with the chunkier JST PH these are 1mm pitch instead of 2mm, but...
https://www.adafruit.com/product/4399

Wiring the MCP9808

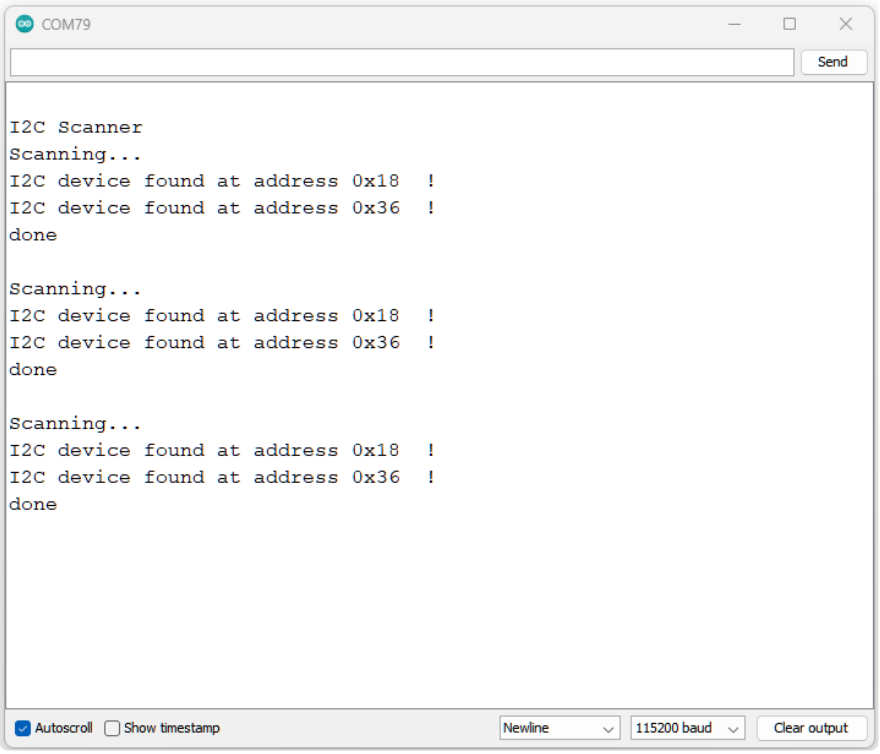
The MCP9808 comes with a STEMMA QT connector, which makes wiring it up quite simple and solder-free.



Now upload the scanning sketch to your microcontroller and open the serial port to see the output. You should see something like this:

You'll see two addresses in the Serial Monitor: **0x18** for the MCP9808 sensor and **0x36** for the built-in MAX17048.

If you have any issues accessing the Serial Monitor, make sure that USB CDC On Boot is set to Enabled for the Feather.

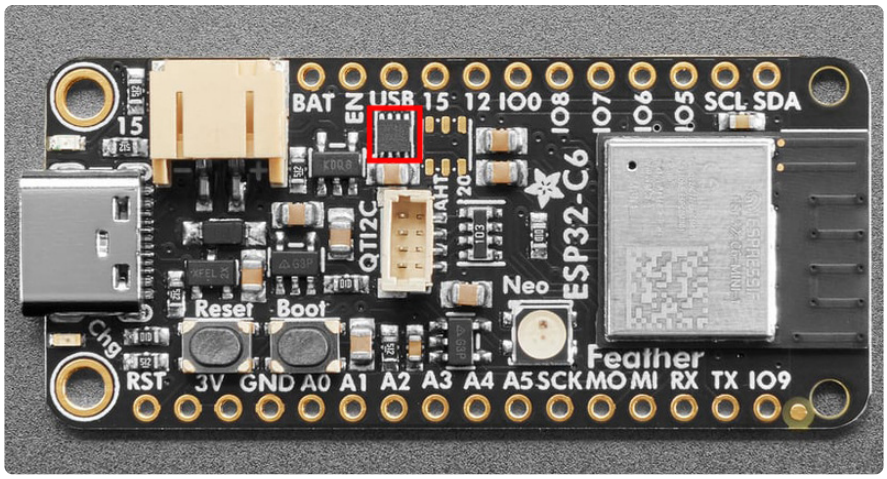


MAX17048 Simple Data

Your microcontroller board comes with a **MAX17048 lithium ion polymer (lipoly) battery monitor** built right onto the board. The MAX17048 is available over I2C.

The sensor comes with its own Adafruit CircuitPython library that makes it simple to write code to read data from it. This example will be using, among other things, the Adafruit_MAX1704X (https://adafru.it/10FG) library.

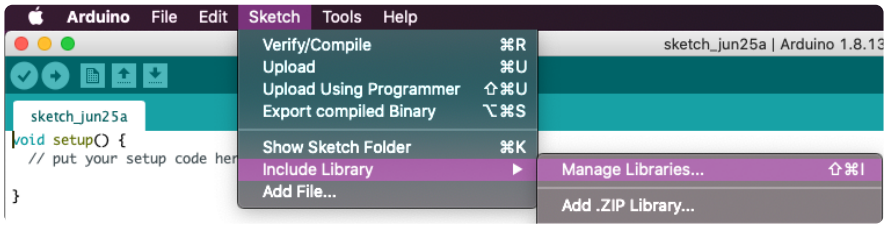
The example simply reads data from the sensor and prints it to the serial console. It is designed to show you how to get data from the sensor.



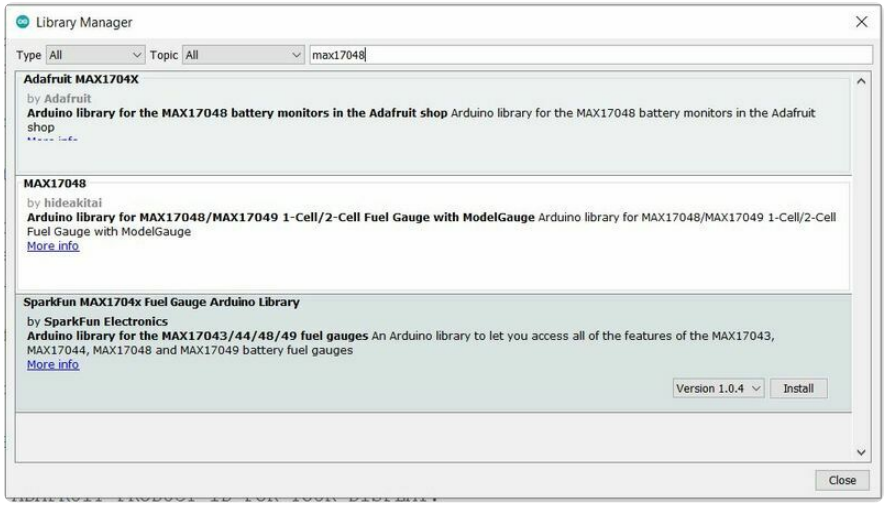
The **MAX17048 battery monitor** (highlighted in red) is immediately below the **USB** pin label. Its I2C address is **0x36**.

Arduino Library Installation

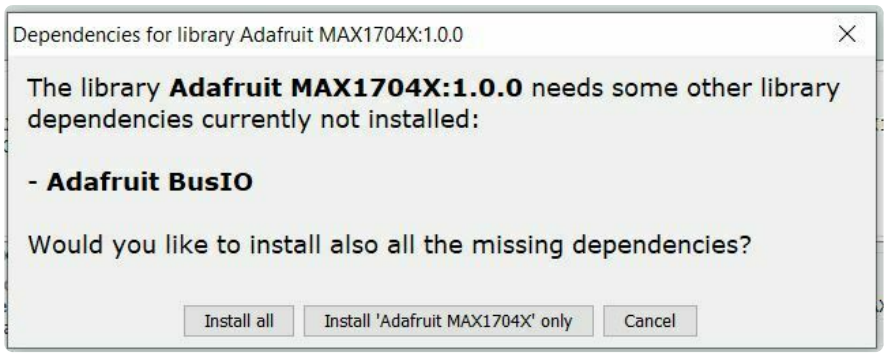
You can install the necessary libraries from the Library Manager. To open, click **Sketch > Include Library > Manage Libraries...**



Search for **MAX17048**, and install the **Adafruit MAX1704X** library.



When asked about installing dependencies, click **Install all**.



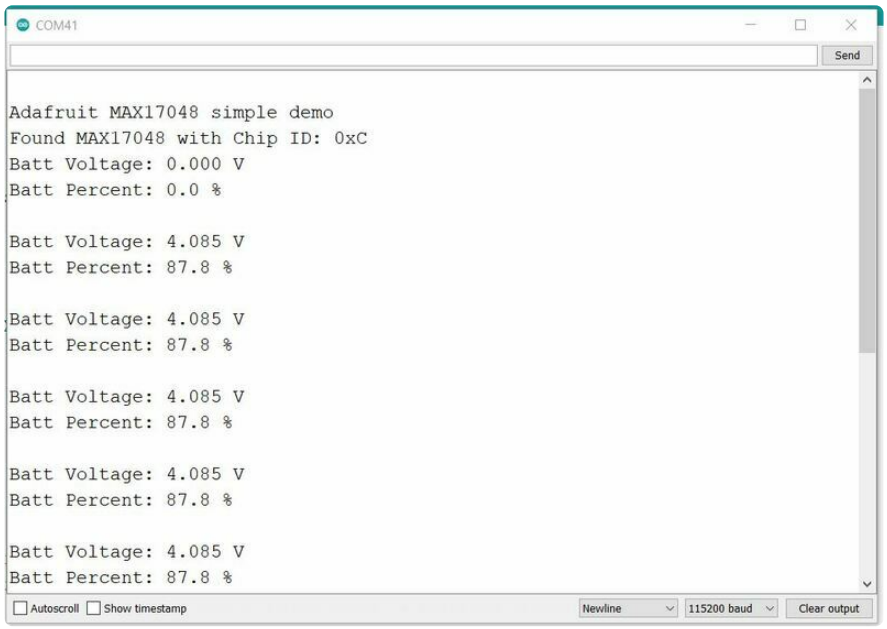
MAX17048 Simple Data Example

Click **File** > **Examples** > **Adafruit MAX1704X** > **MAX17048_basic** to open the example.

```
#include "Adafruit_MAX1704X.h"

Adafruit_MAX17048 maxlipo;

void setup() {
  Serial.begin(115200);
  while (!Serial) delay(10);    // wait until serial monitor opens

  Serial.println(F("\nAdafruit MAX17048 simple demo"));

  while (!maxlipo.begin()) {
    Serial.println(F("Couldnt find Adafruit MAX17048?\nMake sure a battery is
plugged in!"));
    delay(2000);
  }
  Serial.print(F("Found MAX17048"));
  Serial.print(F(" with Chip ID: 0x"));
  Serial.println(maxlipo.getChipID(), HEX);
}

void loop() {
  float cellVoltage = maxlipo.cellVoltage();
  if (isnan(cellVoltage)) {
    Serial.println("Failed to read cell voltage, check battery is connected!");
    delay(2000);
    return;
  }
  Serial.print(F("Batt Voltage: ")); Serial.print(cellVoltage, 3); Serial.println("
V");
  Serial.print(F("Batt Percent: ")); Serial.print(maxlipo.cellPercent(), 1);
Serial.println(" %");
  Serial.println();

  delay(2000);  // dont query too often!
}
```

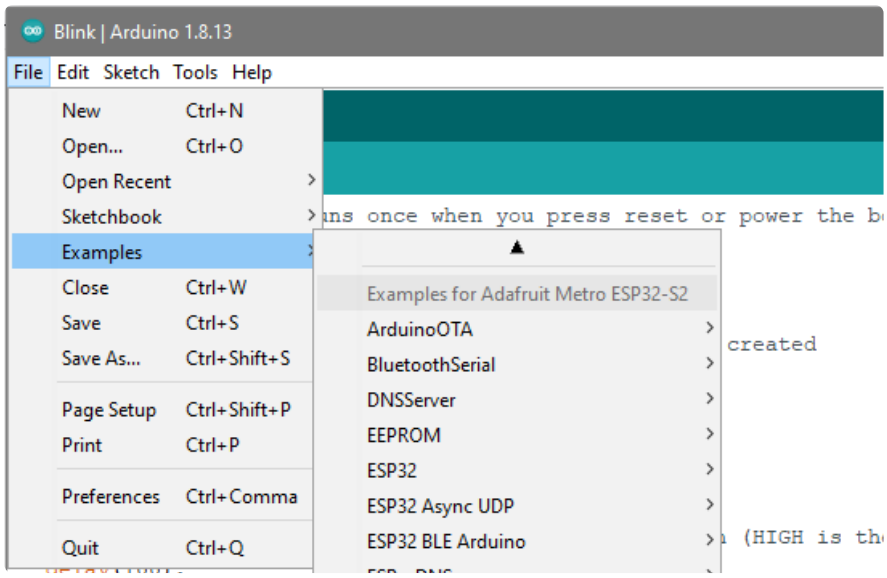
If you have any issues accessing the Serial Monitor, make sure that USB CDC On Boot is set to Enabled.

After opening the **MAX17048_basic** file, upload it to your microcontroller. Open the **Serial Monitor** at **115200 baud**. Plug in a lipo battery to the JST-PH battery port. You should see the battery voltage and percentage data print to the Serial Monitor as the sketch runs.



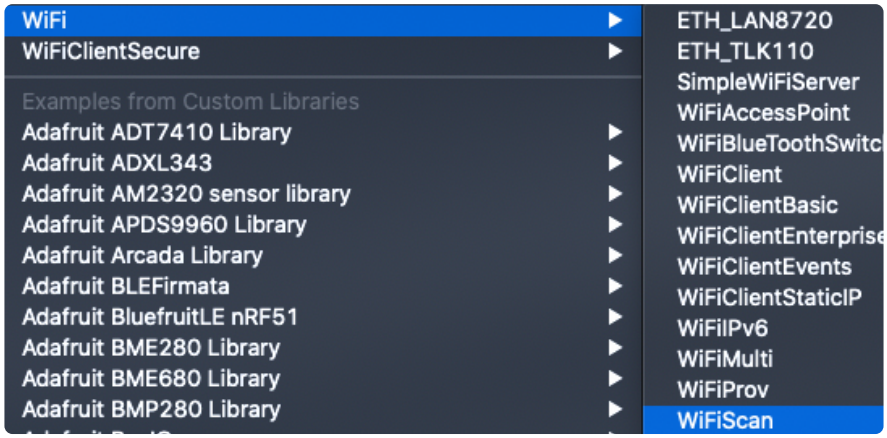
WiFi Test

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.

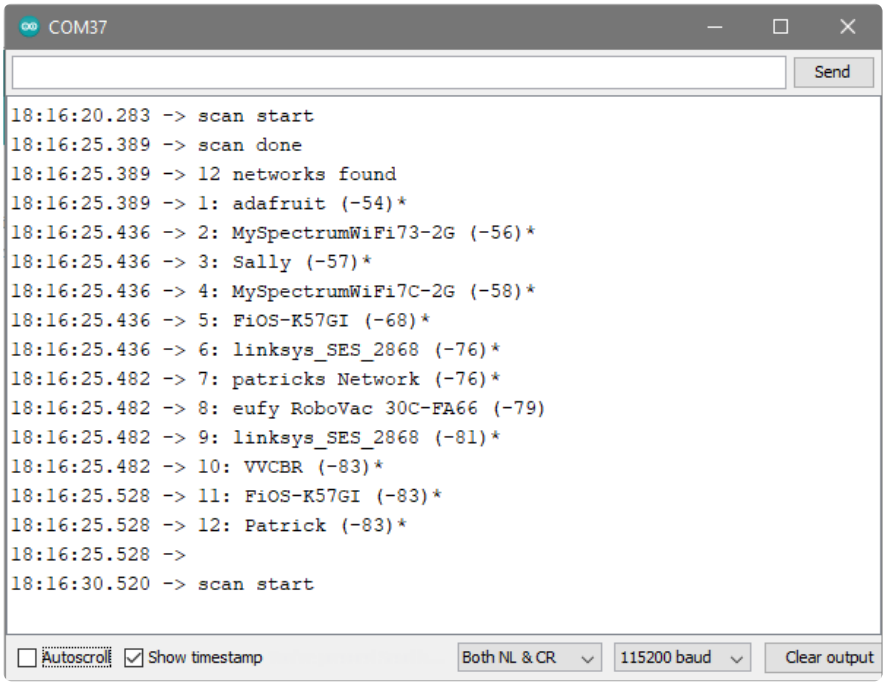
Load up the WiFiScan example under **Examples->Examples for YOUR BOARDNAME->WiFi->WiFiScan**



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.



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.
```

If you have issues establishing a connection, try power cycling the board by unplugging and replugging the USB cable.

Factory Reset

Your microcontroller ships running a factory demo. 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.

Factory Reset Example Code

If you're still able to load Arduino sketches, you can load the following sketch onto your board to return it to its original state.

```
// SPDX-FileCopyrightText: 2024 ladyada for Adafruit Industries
//
// SPDX-License-Identifier: MIT

#include <Adafruit_NeoPixel.h>
#include "Adafruit_TestBed.h"
extern Adafruit_TestBed TB;

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

void setup() {
  //while (! Serial) delay(10);
  Serial.begin(115200);
  TB.neopixelPin = PIN_NEOPIXEL;
  TB.neopixelNum = 1;
  TB.begin();
  TB.setColor(WHITE);
}

uint8_t j = 0;

void loop() {

    TB.setColor(TB.Wheel(j++));
    delay(10);
    if (j == 0) {
      TB.printI2CBusScan();
    }

}
```

Your board is now back to its factory-shipped state! You can now begin again with your plans for your board.

Factory Reset .bin

If your board is in a state where Arduino isn't working, you may need to use these tools to flash a .bin file directly onto your board.

There are two ways to do a factory reset. 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.**

First you'll need to download the factory-reset.bin file. Save the following file wherever is convenient for you. You'll need access to it for both tools.

Click to download the ESP32-C6 Feather Factory Reset .BIN File

https://adafru.it/1a6G

Now that you've downloaded the .bin file, you're ready to continue with the factory reset 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 boards. It allows you to erase the contents of the microcontroller and program up to four files at different offsets.

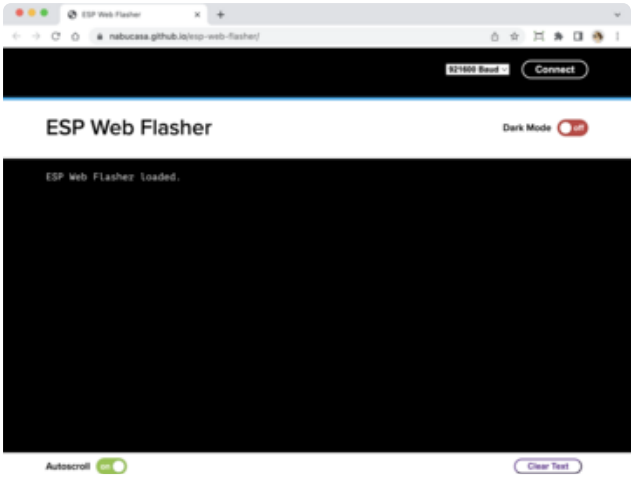
You will have to use a Chromium browser (like Chrome, Opera, Edge...) for this to work, Safari and Firefox, etc. are not supported because we need Web Serial and only Chromium is supporting it to the level needed.

Follow the steps to complete the factory reset.

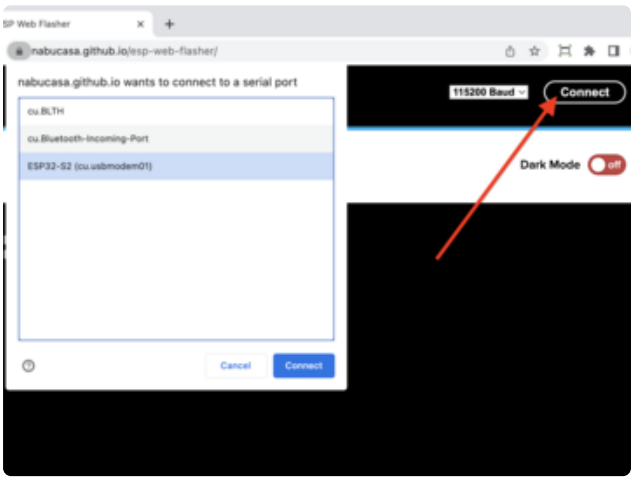
If you're using Chrome 88 or older, see the Older Versions of Chrome section at the end of this page for instructions on enabling Web Serial.

Connect

You should have plugged in **only the ESP32 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/ (https://adafru.it/PMB). 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 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.

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

Erase the Contents

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.
```

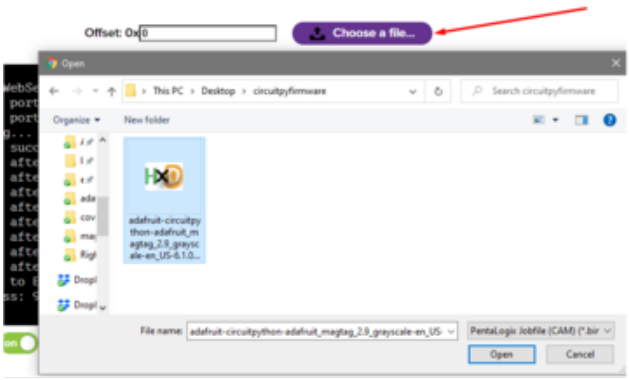
You'll see "Erasing flash memory. Please wait..." This will eventually be followed by "Finished." and the amount of time it took to erase.

Do not disconnect! Immediately continue on to programming the ESP32.

Do not disconnect after erasing! Immediately continue on to the next step!

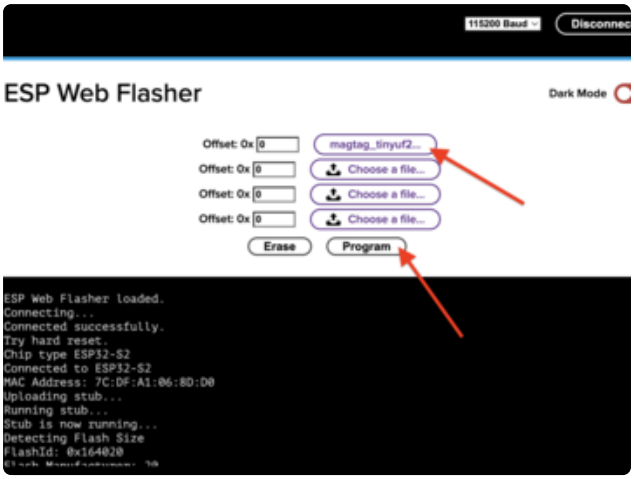
Program the ESP32

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 earlier, 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!

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

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
```

Test the Installation

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

```
9102 kattni@robocrepe:~ (esptool) $ esptool.py
esptool.py v3.0
usage: esptool [-h] [--chip {auto,esp8266,esp32,esp32s2,esp32s3beta2,esp32c3}]
               [--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,m
ake_image,elf2image,read_mac,chip_id,flash_id,read_flash_status,write_flash_statu
s,read_flash,verify_flash,erase_flash,erase_region,version,get_security_info}
               ...
```

Connect

Run the following command, replacing the `COM88` 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.

```
9103 kattni@robocrepe:~ (esptool) $ esptool.py --port /dev/cu.usbserial-1144440 c
hip_id
esptool.py v3.0
Serial port /dev/cu.usbserial-1144440
Connecting....
Detecting chip type... ESP32
Chip is ESP32-PICO-V3-02 (revision 3)
Features: WiFi, BT, Dual Core, 240MHz, Embedded Flash, Embedded PSRAM, VRef calib
ration in efuse, Coding Scheme None
Crystal is 40MHz
MAC: 4c:75:25:be:19:98
Uploading stub...
Running stub...
Stub running...
Warning: ESP32 has no Chip ID. Reading MAC instead.
MAC: 4c:75:25:be:19:98
Hard resetting via RTS pin...
```

Installing the Factory Test file

Run this command and replace the serial port name, `COM88`, with your matching port and `*-factory-reset.bin` with file you just downloaded

```
esptool.py --port COM88 write_flash 0x0 *-factory-reset.bin
```

Don't forget to change the `--port` name to match.

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
Serial port /dev/cu.usbserial-1144440
Connecting.....
Detecting chip type... ESP32
Chip is ESP32-PICO-V3-02 (revision 3)
Features: WiFi, BT, Dual Core, 240MHz, Embedded Flash, Embedded PSRAM, VRef calib
ration in efuse, Coding Scheme None
Crystal is 40MHz
MAC: 4c:75:25:be:19:98
Uploading stub...
Running stub...
Stub running...
Configuring flash size...
Compressed 3084464 bytes to 241457...
Wrote 3084464 bytes (241457 compressed) at 0x00000000 in 26.3 seconds (effective
939.4 kbit/s)...
Hash of data verified.

Leaving...
Hard resetting via RTS pin...
```

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.

In the event that pressing the reset button does not restart the board, unplug the board from USB and plug it back in to get the new firmware to start up.

The NeoPixel LED on the ESP32-C6 Feather will show a rainbow swirl. Every few seconds, you'll see an I2C scan print to the Serial Monitor with address 0x36 for the onboard battery monitor.

You've successfully returned your board to a factory reset state!

Older Versions of Chrome

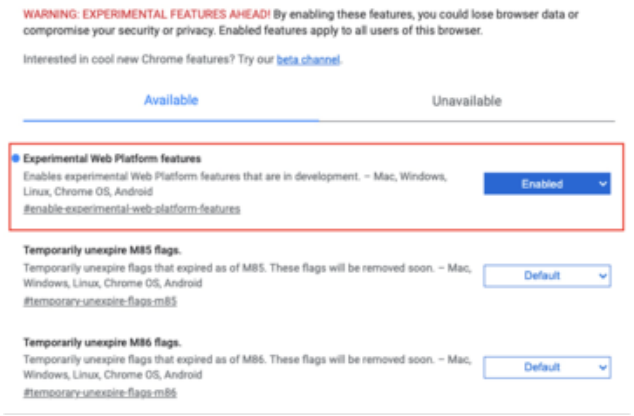
As of chrome 89, Web Serial is already enabled, so this step is only necessary on older browsers.

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.

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:

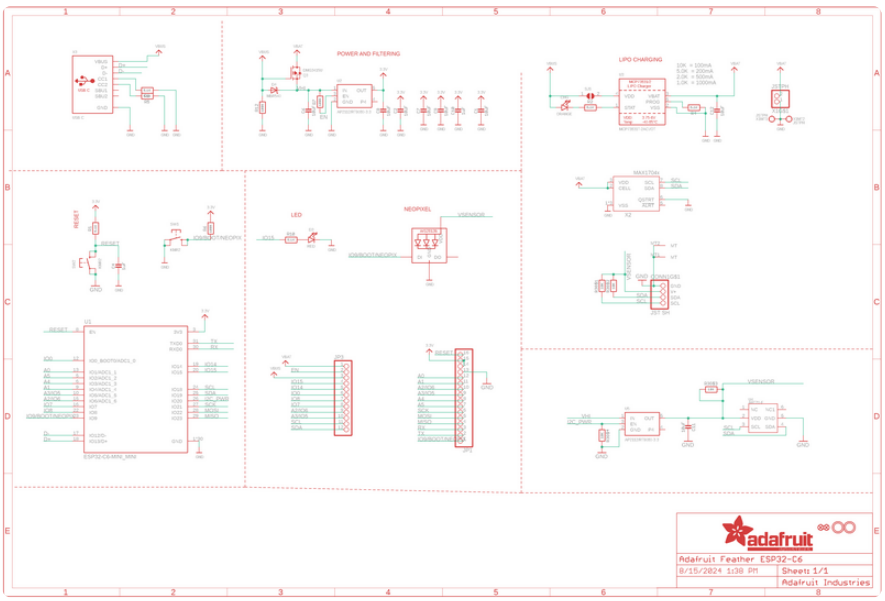
Visit **chrome://flags** from within Chrome.
Find and enable the **Experimental Web Platform features**
Restart Chrome

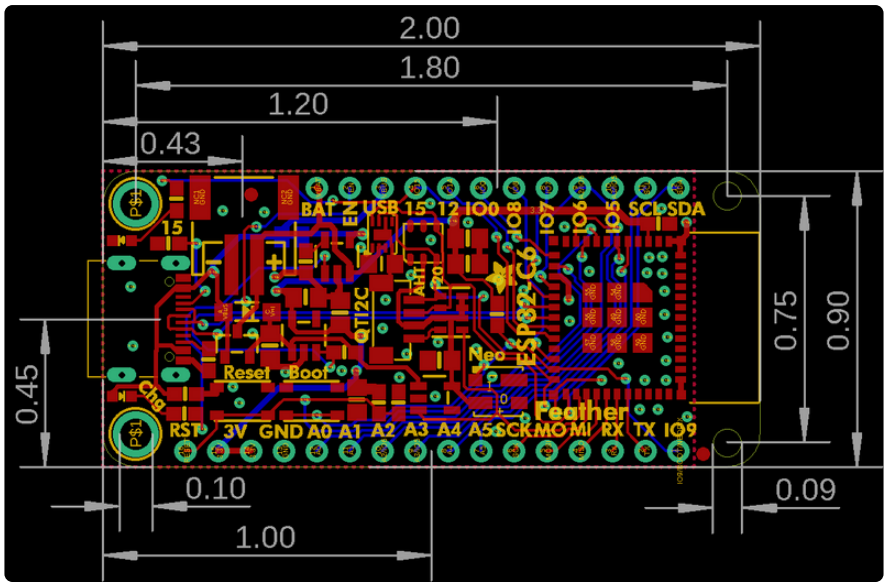
Downloads

Files

- ESP32-C6 datasheet (https://adafru.it/1a94)
- EagleCAD PCB files on GitHub (https://adafru.it/1a6I)
- Fritzing object in the Adafruit Fritzing Library (https://adafru.it/1a6J)
- PrettyPins pinout PDF on GitHub (https://adafru.it/1a6D)
- PrettyPins pinout SVG (https://adafru.it/1a6K)

Schematic and Fab Print




2.00
1.80
1.20
0.43
0.45
0.75
0.90
0.10
0.09
1.00
BAT EN USB 15 12 IO0 IO8 IO7 IO6 IO5 SCL SDA
Reset Boot
Neo
Feather
RST 3V GND A0 A1 A2 A3 A4 A5 SCK MO MI RX TX IO9

X-ON Electronics

Largest Supplier of Electrical and Electronic Components

Click to view similar products for Multiprotocol Development Tools ***category:***

Click to view products by Adafruit ***manufacturer:***

Other Similar products are found below :

EVK-MAYA-W271 FCM360WAATB-0L-04 XKC-M5T-W ATWINC3400-XPRO 2636 RE-WFKIT-9260NVP 2542 5395 5426 5500 5672 5700 5750 5778 5835 irpi01-868 ABX00083 ABX00087 ABX00092 BCM94343WWCD1_EVB XKC-V1T-U ESP32-S3-EYE EV-WT02C40C-eng CY8CPROTO-062S2-43439 INP3010 INP3011 ISP3080-UX-AN ISP3080-UX-DK ISP3080-UX-TB ISP3080-UX-TG ISP4520-AS-DK 453-00011-K1 XPC270300EK MIKROE-2440 MIKROE-4493 MIKROE-5983 MIKROE-6154 MIKROE-6170 MIKROE-6217 MIKROE-6488 MIKROE-6489 MTDOT-BOX-G-868-B MTDOT-BOX-G-915-B LBAA0XV2GT-001EVK LBEE0ZZ2WE-uSD-M2 LBEE0ZZ2WF-uSD-M2 LBEE0ZZ2WS-ST-NCL LBEH5DU1BW-TEMP-DS-SD LBUA2ZZ2DK-EVK NRF5340-DK