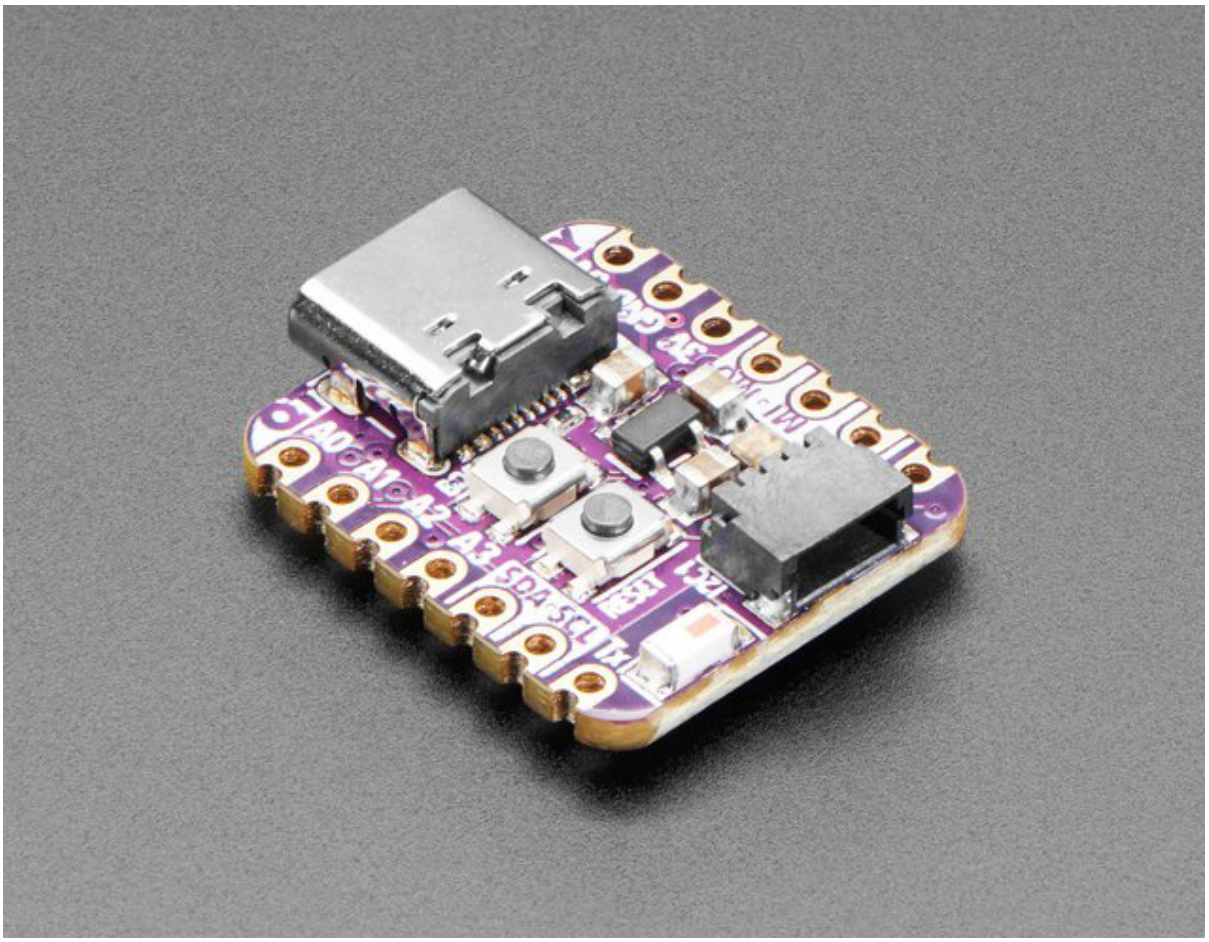
Adafruit QT Py ESP32 Pico

Created by Kattni Rembor

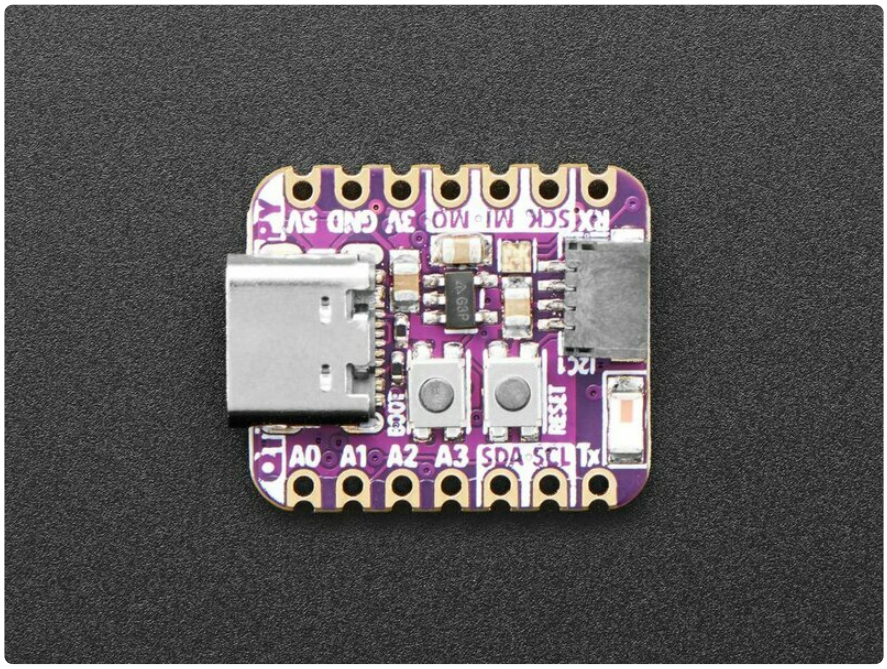


https://learn.adafruit.com/adafruit-qt-py-esp32-pico

Last updated on 2024-11-29 10:08:31 AM EST

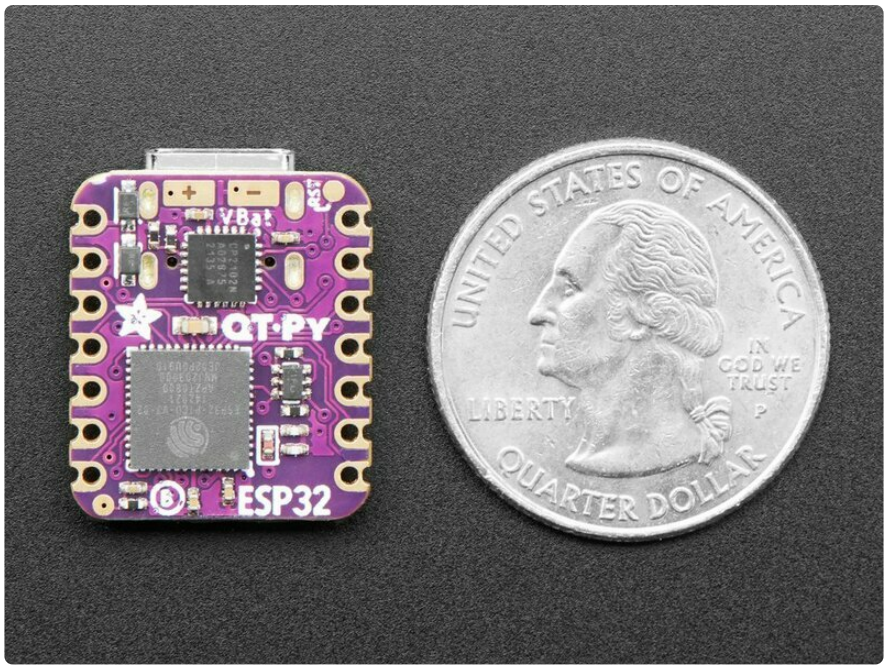
Table of Contents

Overview



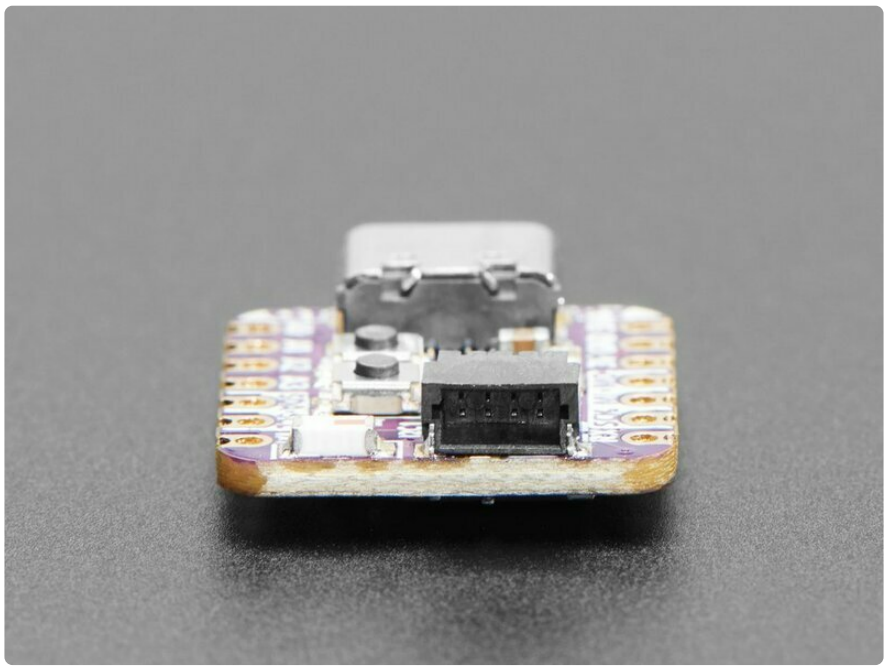
This development board is like when you're watching a super-hero movie and the protagonist shows up in a totally amazing costume in the third act and you're like 'OMG! That's the hero and they're here to kick some serious butt!" but in this case it's a microcontroller.

This QT Py board is a thumbnail-sized PCB that features the **ESP32-Pico-V3-02**, an all-in-one chip that has an **ESP32 chip with dual-core 240MHz Tensilica processor**, **WiFi** and **Bluetooth classic + BLE**, adds a bunch of required passives and oscillator, **8 MB of Flash memory** and **2 MB of PSRAM**. We add a USB to serial converter chip, some more passives, an antenna, USB C, buttons, NeoPixel and QT connector to outfit this super-hero chip for any task you want to throw it at.



At the core of ESP32-PICO-V3-02 is the ESP32 (ECO V3) chip, which is a single 2.4 GHz Wi-Fi and Bluetooth combo chip designed with TSMC's 40 nm low-power technology. ESP32-PICO-V3-02 integrates all peripheral components seamlessly, including a crystal oscillator, flash, PSRAM, filter capacitors, and RF matching links in one single package. This makes it perfect for stuffing into such a small space as the QT Py.

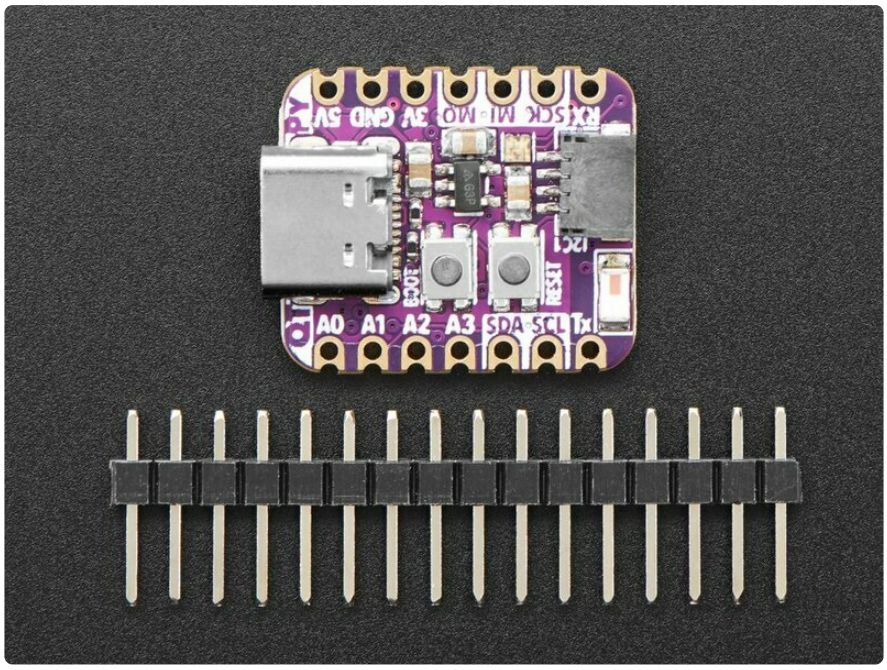
Please note the QT Py ESP32 Pico does not have native USB support - instead there's a USB to serial converter chip. This means it cannot act like a USB keyboard or mouse - but it does have BLE and BT classic so you could do wireless.



OLEDs (https://adafru.it/18eN)! Inertial Measurement Units (https://adafru.it/18eO)! Sensors a-plenty (https://adafru.it/18eP). All plug-and-play thanks to the innovative chainable design: SparkFun Qwiic (https://adafru.it/Fpw)-compatible STEMMA QT (https://adafru.it/Ft4) connectors for the I2C bus so you don't even need to solder! Just plug in a compatible cable and attach it to your MCU of choice, and you're ready to load up some software and measure some light. Seeed Grove I2C boards (http://adafru.it/4528) will also work with this adapter cable.

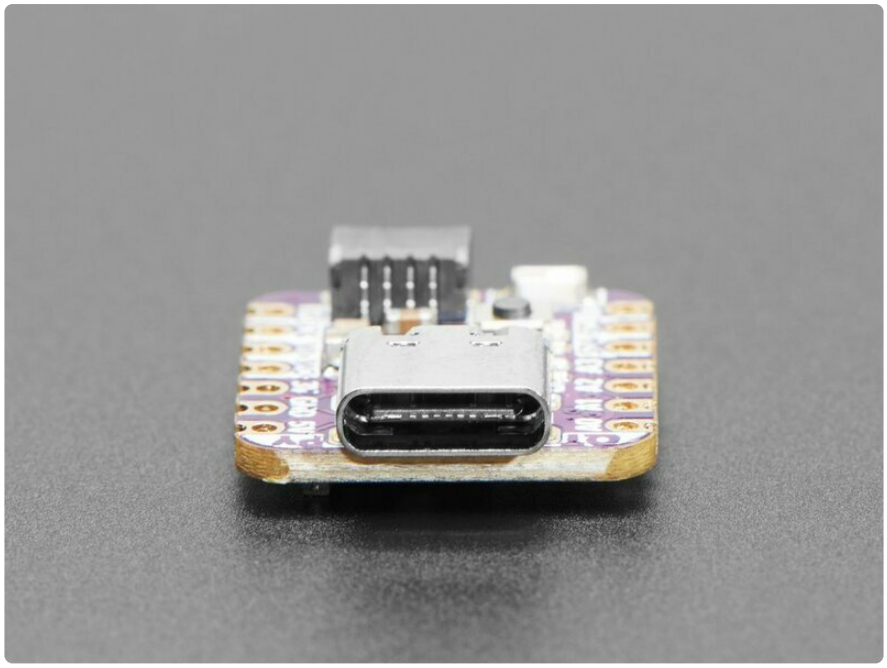
The pinout and shape are Seeed Xiao (https://adafru.it/NC3) compatible, with castellated, solderable pads. In addition to the QT connector, we also added an **RGB NeoPixel** (with controllable power pin to allow for ultra-low-power usage), **a reset button** (great for restarting your program or entering the bootloader), and a button on GPIO 0 for entering the ROM bootloader or for user input.

This board runs Arduino like a dream, or use the ESP IDF for more control over your projects.

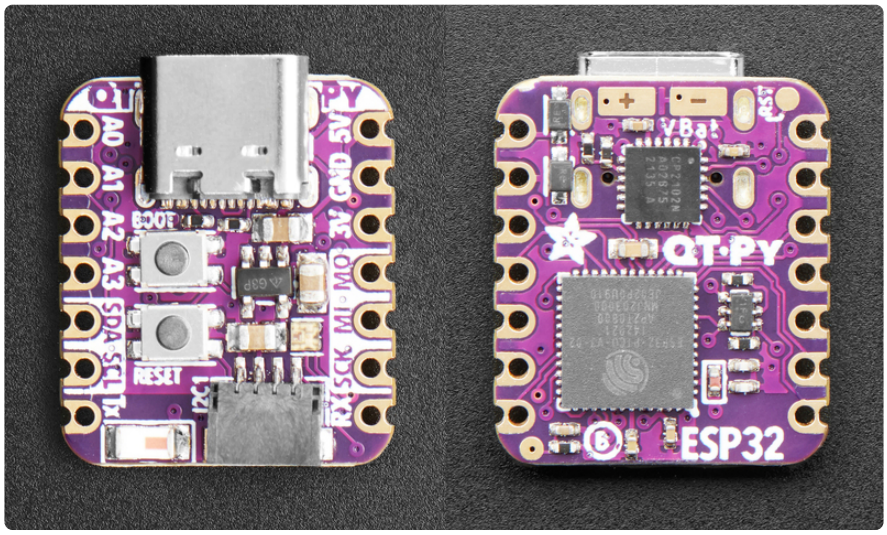


- Same size, form-factor, and pin-out as Seeed Xiao
- **USB Type C connector** - If you have only Micro B cables, this adapter will come in handy (http://adafru.it/4299)!
- **ESP32-V2-03 Dual Core 240MHz Xtensa processor** - the ESP32 you know and love, with the latest engineering fixes. Massive user base and thousands of existing projects and libraries to use.
- WiFi, Bluetooth LE and Classic for any IoT project usage
- **8 MB Flash & 2 MB PSRAM**
- USB to Serial converter built in with high speed UART for debugging and uploading.

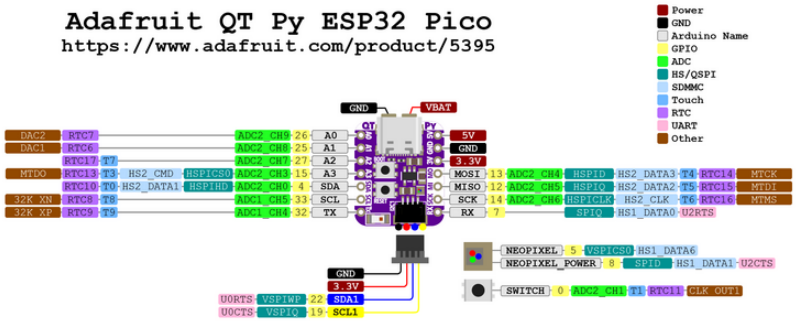
- Can be used with **Arduino IDE** or **MicroPython**
- **Built-in RGB NeoPixel LED** with power control to reduce quiescent power in deep sleep
- Battery input pads on underside with diode protection for external battery packs up to 6V input
- **13 GPIO pins:**
    - 11 on breakout pads, 2 more on QT connector
    - 10 x 12-bit analog inputs
    - Dual 8-bit analog output DACs on A0/A1
    - PWM outputs on any pin
    - Two I2C ports, one on the breakout pads, and another with STEMMA QT plug-n-play connector
    - Hardware UART in addition to the USB-serial UART
    - Hardware SPI on the high speed SPI peripheral pins
    - Hardware I2S on any pins
    - 8 x Capacitive Touch with no additional components required
- 3.3V regulator with **600mA peak output** (https://adafru.it/NC4)
- Light sleep at 4mA, deep sleep at ~70uA
- **Reset switch** for starting your project code over, boot 0 button for entering bootloader mode or for user reading
- **Really really small**



Pinouts

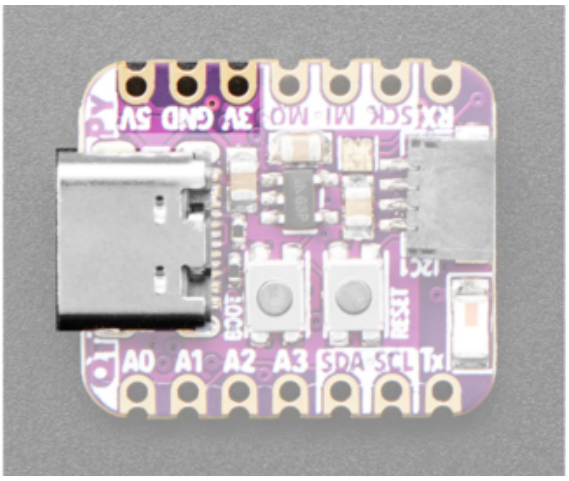


The Adafruit QT Py ESP32 Pico is jam-packed with pins, connectors and buttons. This page covers all the features!



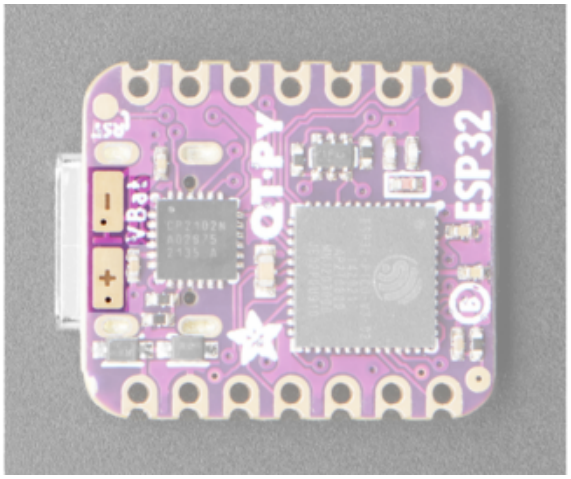
PrettyPins PDF on GitHub (https://adafru.it/-IF).

Power



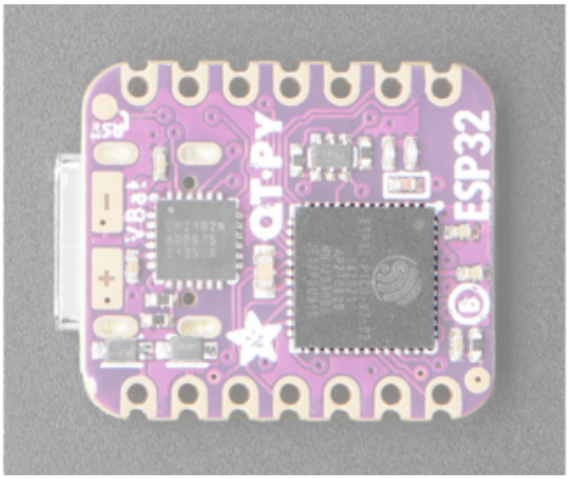
USB-C port - This is used for both powering and programming the board. You can power it with any USB C cable.
3.3V - This pin is the output from the 3.3V regulator, it can supply 600mA peak.
GND - This is the common ground for all power and logic.
5V - This is 5V out from the USB port.

You can also use the 5V pin as a voltage input but you must have some sort of diode (schottky, signal, power, really anything) between your external power source and this pin with anode to battery, cathode to 5V pin. Note that you cannot power the USB port by supplying 5V to this pin: there is a protection diode that prevents the 5V from reaching the USB connector. This is to protect host computer USB ports, etc. You can draw 1A peak through the diode, but we recommend keeping it lower than that, about 500mA.



VBAT pads - On the back of the board are two pads labeled + and - (ground). These are the battery input pads with diode protection for external battery packs from 3V to 6V input.

ESP32 Module

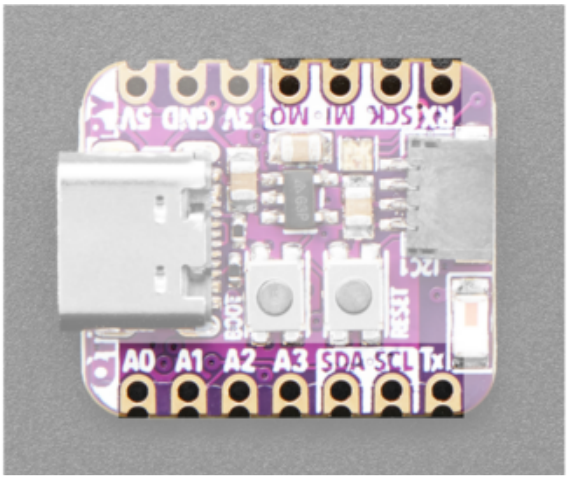


The **ESP32-Pico-V3-02** is an all-in-one chip that has an **ESP32 chip with dual-core 240MHz Tensilica processor**, **WiFi** and **Bluetooth classic** + **BLE**, as well as **8 MB of Flash memory** and **2 MB of PSRAM.**

At the core of ESP32-PICO-V3-02 is the ESP32 (ECO V3) chip, which is a single 2.4 GHz Wi-Fi and Bluetooth combo chip designed with TSMC's 40 nm low-power technology. ESP32-PICO-V3-02 integrates all peripheral components seamlessly, including a crystal oscillator, flash, PSRAM, filter capacitors, and RF matching links in one single package. The 8 MB of flash is inside the chip and is used for **both** program firmware and filesystem storage.

The ESP32-Pico is an ideal choice for a wide variety of application scenarios relating to the Internet of Things (IoT) (https://adafru.it/Bwq), wearable electronics (https://adafru.it/Osb), and smart homes.

Logic Pins



There are eleven GPIO pins broken out to pads. There is hardware I2C, UART, and SPI.

Ten pads are 12-bit analog inputs.

You can do PWM output or hardware I2S on any pin.

There are eight pins (A2, A3, SCL, SDA, TX, MOSI, MISO, SCK) that can do capacitive touch without any external components needed.

That's the general overview. Now for the details!

There are four analog pins.

- **A0 and A1** are the only DAC output pins. These can be used as 8-bit true analog outputs. No other pins can do so. These pins can also be analog inputs. A0 and A1 are on ADC2.
- **A2 and A3** can also be analog inputs. A2 and A3 are on ADC2. These two pins can do capacitive touch.

The I2C pins. **These are NOT shared by the STEMMA QT connector!**

- **SCL** - This is the I2C clock pin. There is no pull-up on this pin, so for I2C please add an external pull-up if the breakout doesn't have one already. This pin can do capacitive touch and also analog input
- **SDA** - This is the I2C data pin.There is no pull-up on this pin, so for I2C please add an external pull-up if the breakout doesn't have one already. This pin can do capacitive touch and also analog input

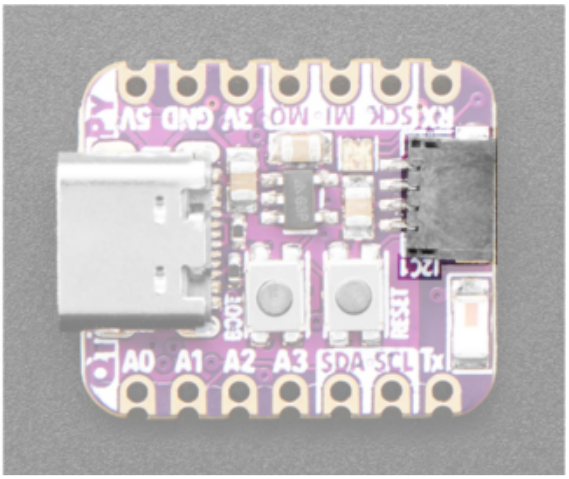
The UART interface.

- **RX** - This is the UART receive pin. Connect to TX (transmit) pin on your sensor or breakout.
- **TX** - This is the UART transmit pin. Connect to RX (receive) pin on your sensor or breakout. This pin can do capacitive touch.

The SPI pins are on the ESP32 high-speed peripheral. You can set any pins to be the low-speed peripheral but you won't get the speedy interface!

- **SCK** - This is the SPI clock pin. This pin can do capacitive touch.
- **MI** - This is the SPI **M**icrocontroller **I**n / **S**ensor **O**ut pin. This pin can do capacitive touch.
- **MO** - This is the SPI **M**icrocontroller **O**ut / **S**ensor **I**n pin. This pin can do capacitive touch.

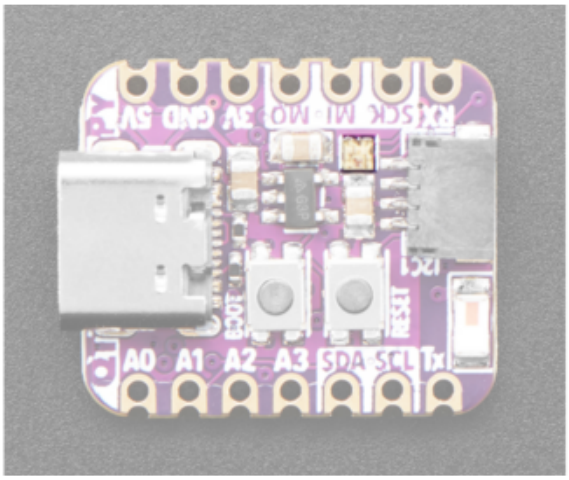
STEMMA QT Connector



This **JST SH 4-pin** STEMMA QT **(https://adafru.it/Ft4) connector** breaks out a second I2C interface (SCL1, SDA1, 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).

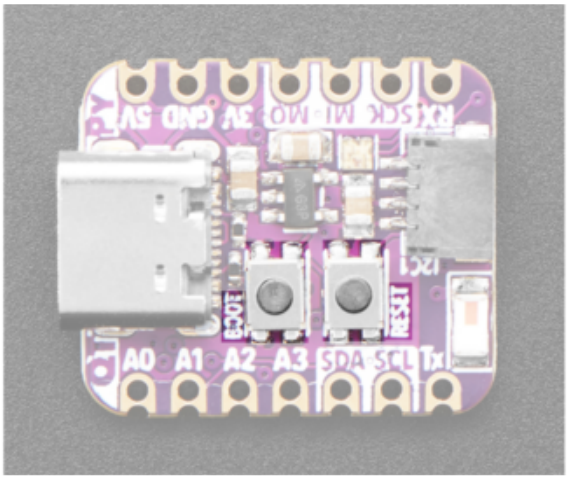
The STEMMA QT connector IO pins in Arduino are `19` (SCL1) and `22` (SDA1) and are available on `Wire1`.

NeoPixel LED



Above the SCK and MI on the silk, is the **RGB NeoPixel LED**. This addressable LED can be controlled with code. It is available in Arduino as `PIN_NEOPIXEL`.
There is a `NEOPIXEL_POWER` **power pin** that is pulled high in Arduino by default. It is necessary for the NeoPixel to work. This is so it can be fully de-powered for low power usage.

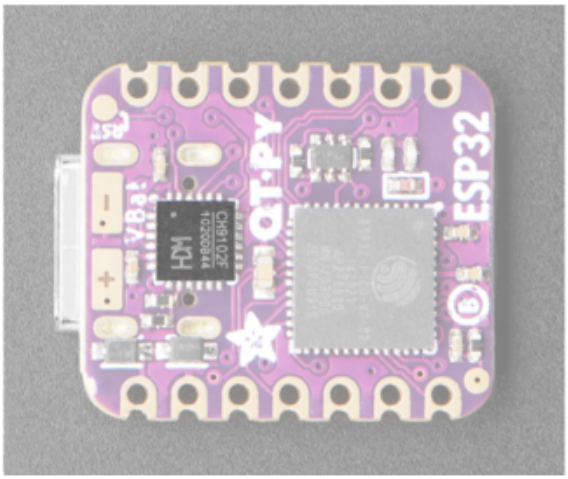
Buttons



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.
BOOT button - This button can be read in code as `SWITCH` or GPIO 0 (set it to be an input-with-pullup).

Both buttons combined can be used to enter the ROM bootloader. To enter ROM bootloader mode, press and continue to hold down the BOOT button while tapping the reset button. When in the ROM bootloader, you can upload code and query the chip using `esptool`.

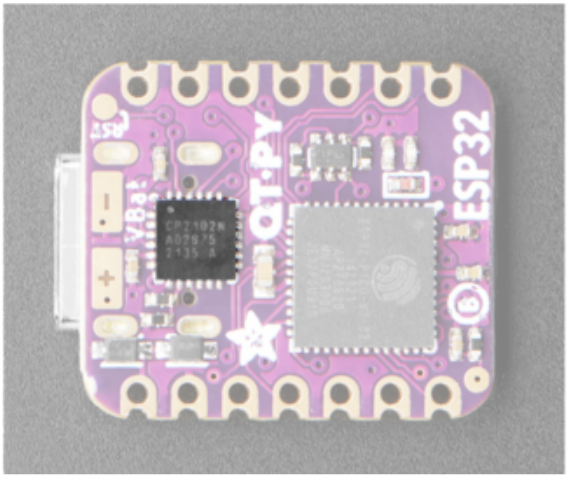
CH9102 USB-to-Serial Converter



The **CH9102F USB to serial converter** can handle 50bps to 4Mbps max rate.

Boards purchased before May 26th, 2022 will have the CP2102N USB to serial converter shown below.

CP2102 USB-to-Serial Converter



The **CP2102N USB to serial converter** can handle 3 mbps max rate.

Low Power Usage

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

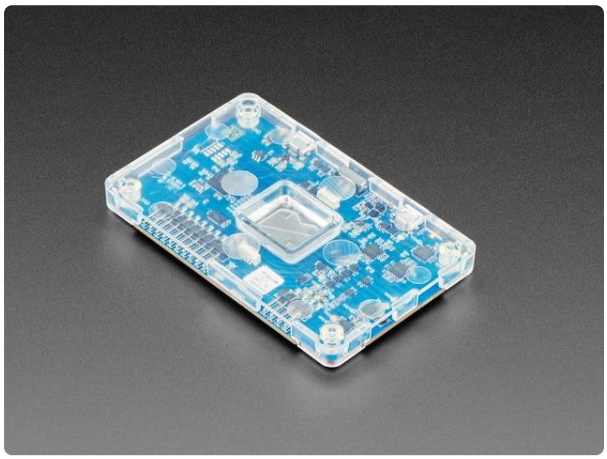
There is a NeoPixel power pin available to cut power to the NeoPixel if desired.

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() {
```

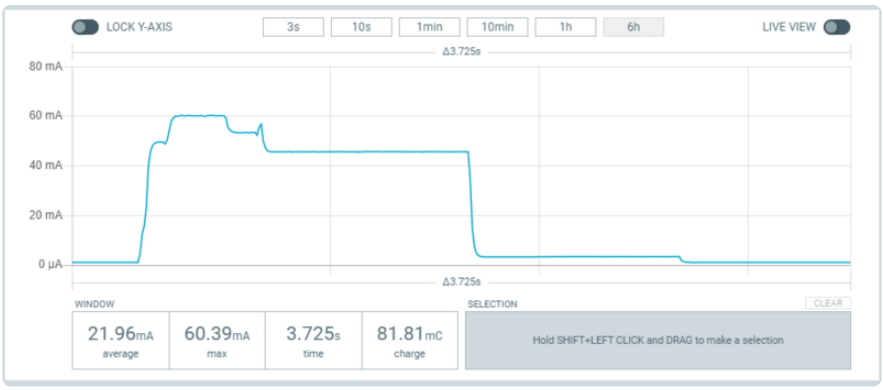
```
#if defined(NEOPIXEL_POWER)
  pinMode(NEOPIXEL_POWER, OUTPUT);
  digitalWrite(NEOPIXEL_POWER, HIGH);
#endif

#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 on 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:

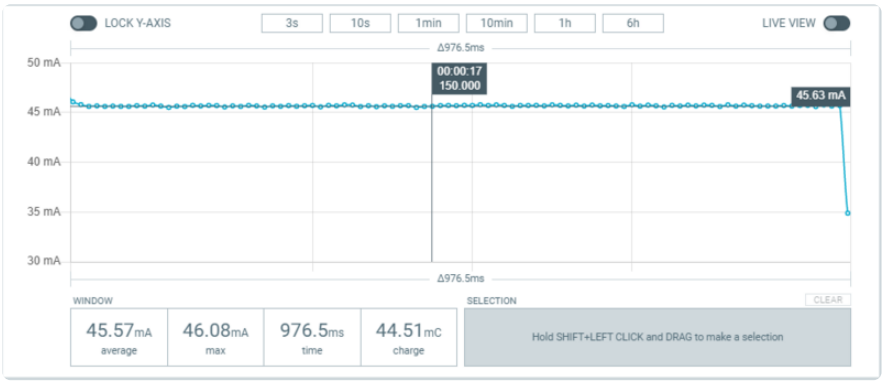


The big pulse is normal mode, you can see the ESP32 booting up, loading code, and then pausing 1 second. Then there's a big drop for one sec to light sleep, and finally one more 1 second pause at deep sleep.

Power Draw for QT Py ESP32 Pico

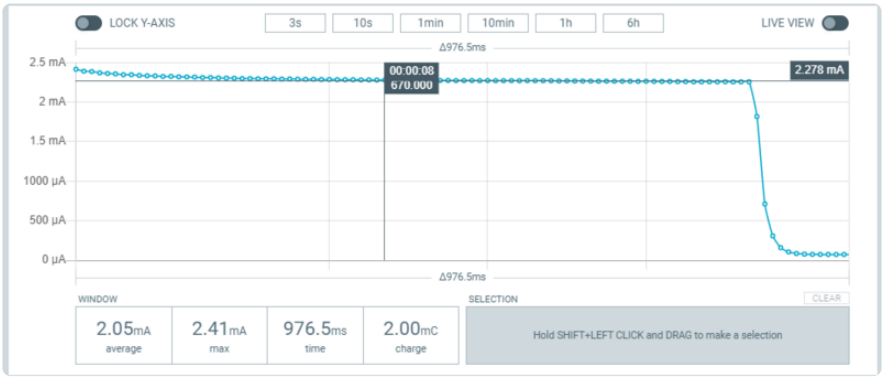
The following graphs show the power draw for the QT Py ESP32 Pico in normal power mode, light sleep mode, and deep sleep mode.

Normal Mode



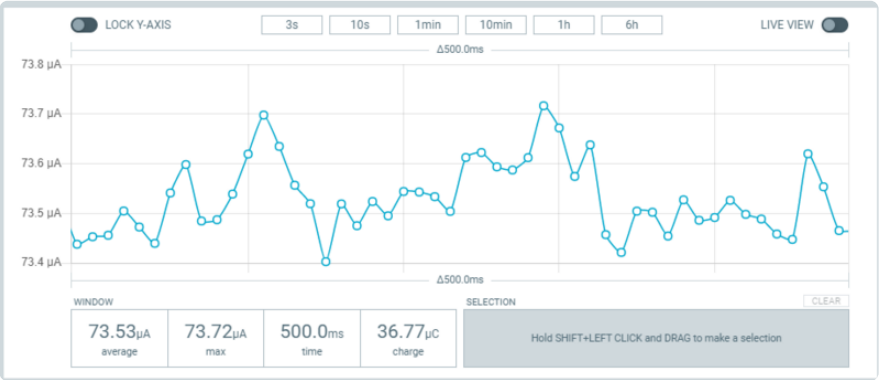
The power draw, running normally, is 45.5mA.

Light Sleep



The power draw in light sleep mode is 2mA.

Deep Sleep



The power draw in deep sleep is 73.5uA.

Arduino IDE Setup

You need to install the right USB-to-serial driver for your chip in addition to the Arduino IDE. If you are unsure which is the right one, install both!

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 CP2104 / CP2102N USB Driver

The USB-to-Serial converter that talks to the ESP32 chip itself will need a driver on your computer's operating system. The driver is available for Mac and Windows. It is already built into Linux.

Click here to download the CP2104 USB Driver

https://adafru.it/vrf

Install CH9102 / CH34X USB Driver

Newer ESP32 boards have a different USB-to-serial converter that talks to the chip itself, and will need a driver on your computer's operating system. The driver is available for Mac and Windows. It is already built into Linux.

If you would like more detail, check out the guide on installing these drivers (https://adafru.it/-f8).

Click here to download the Windows driver

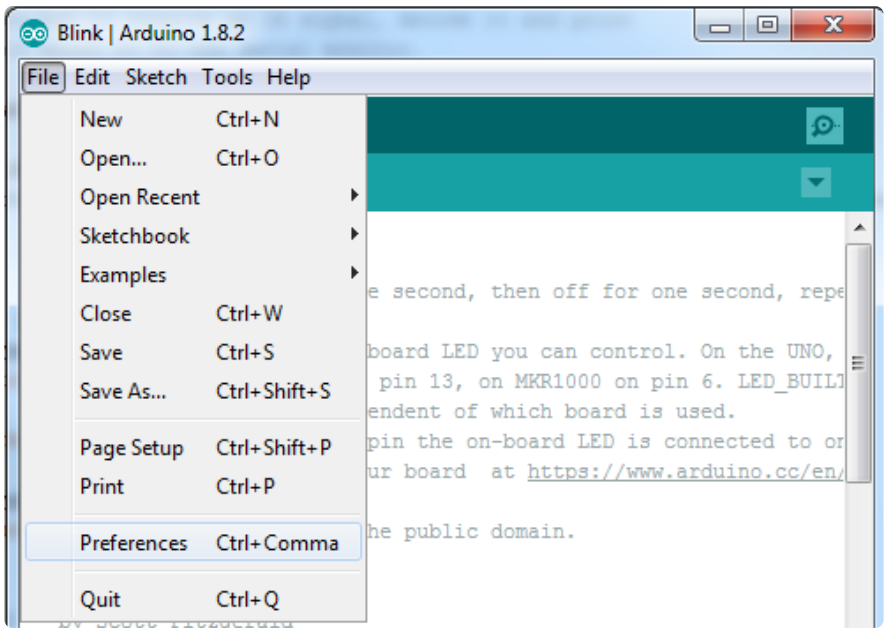
https://adafru.it/-f9

Click here to download the Mac driver

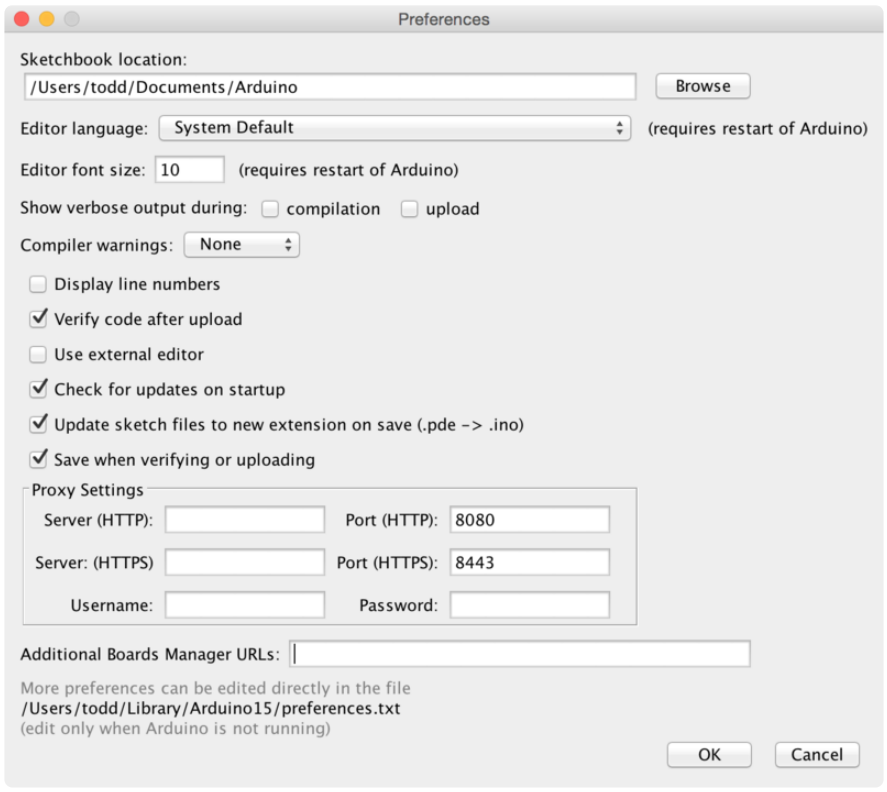
https://adafru.it/-ed

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.



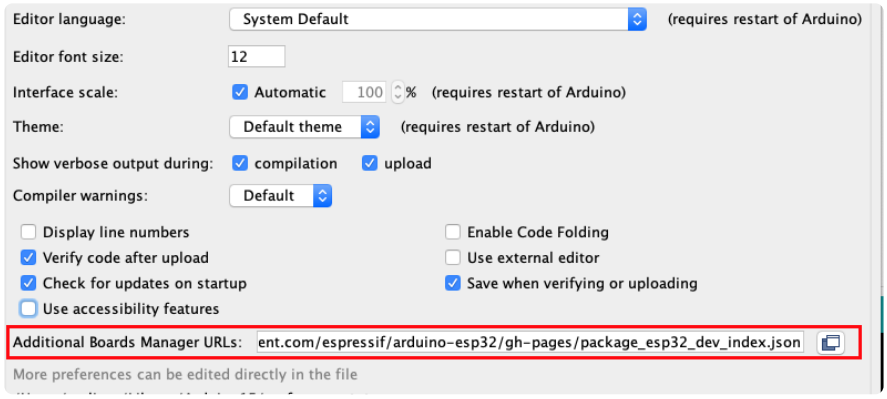
A dialog will pop up just like the one shown below.



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.

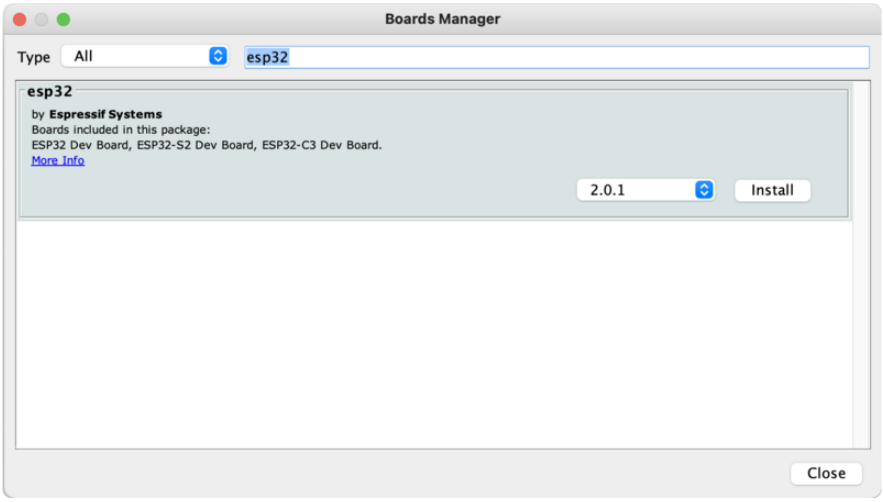
```
https://raw.githubusercontent.com/espressif/arduino-esp32/gh-pages/package_esp32_dev_index.json
```



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.

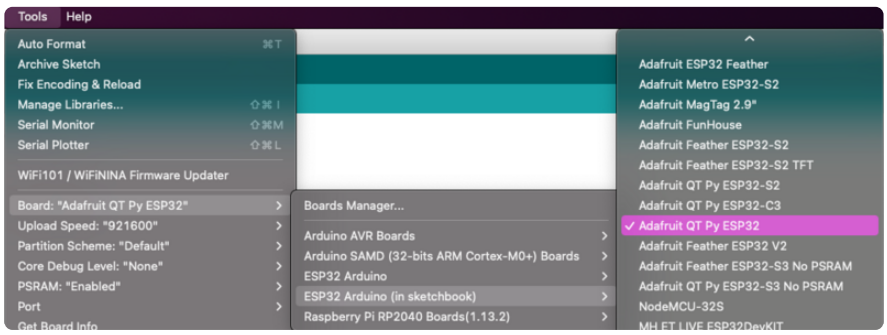
The next step is to actually install the Board Support Package (BSP). Go to the **Tools → Board → Board Manager** submenu. A dialog should come up with various BSPs. Search for **esp32**.



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 QT Py ESP32.



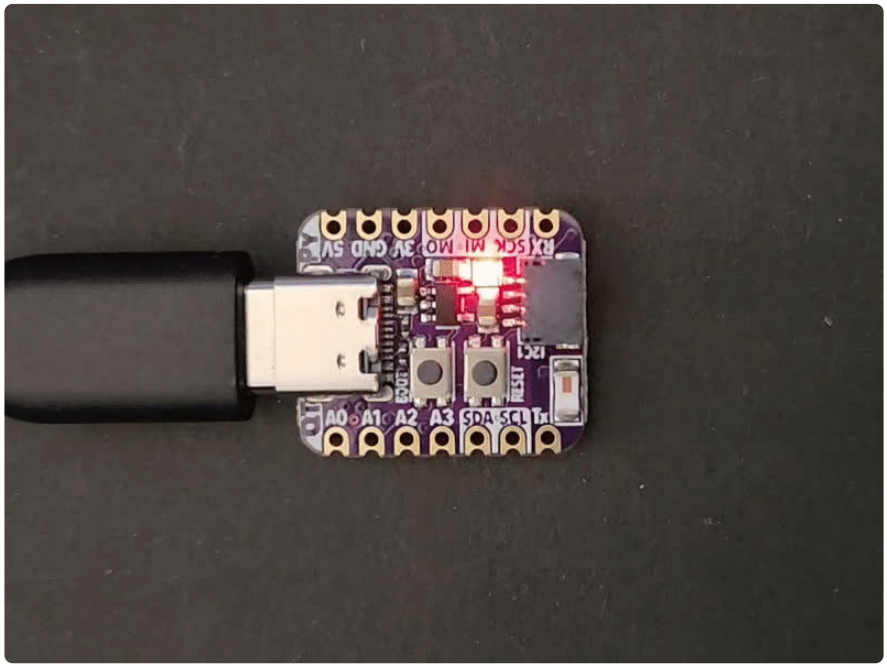
The upload speed can be changed: faster speed makes uploads take less time but sometimes can cause upload issues. **921600** should work fine, but if you're having issues, you can drop down lower.

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!

Now traditionally you would use an onboard LED to make a blink occur. **However, this board does not have an onboard single-color LED**, so we will 'mimic' the same blink sketch but instead of using a digital output pin, we will use NeoPixel support to blink the onboard RGB LED!



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!

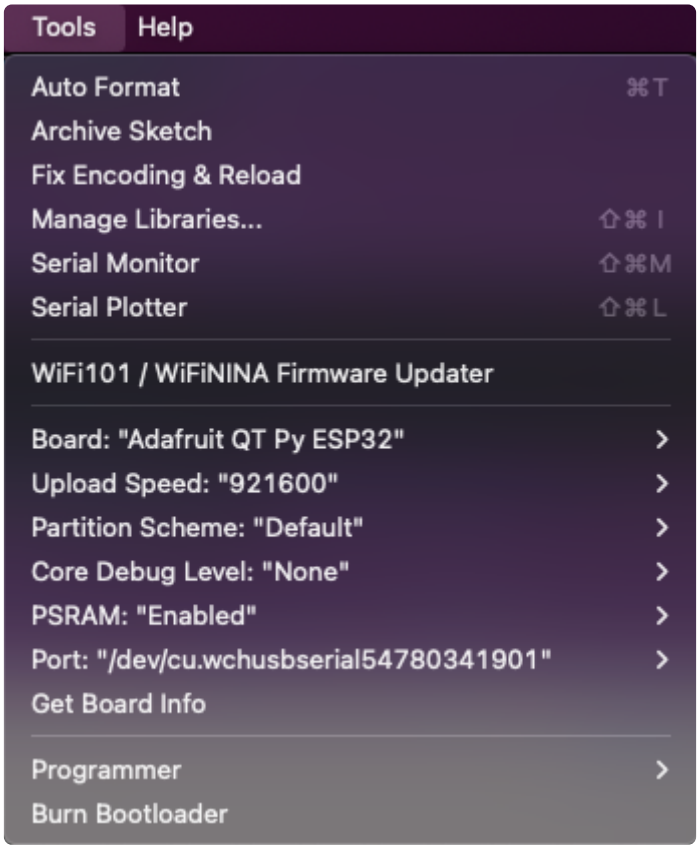
There is a NeoPixel power pin which must be pulled high to enable the NeoPixel LED.

The QT Py ESP32 Pico does not have a power LED.

Start up Arduino IDE and Select Board/Port

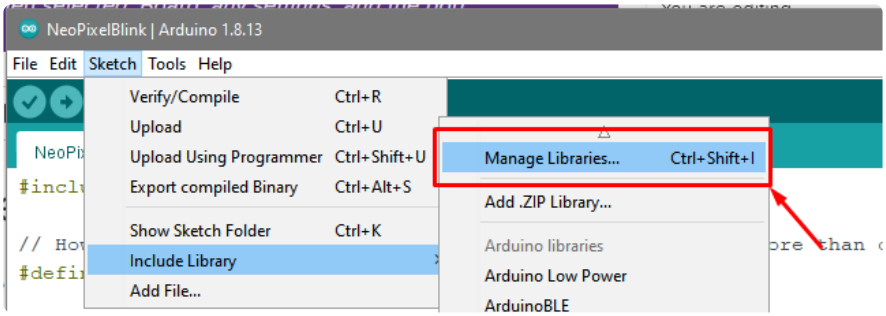
OK, now you are prepared! Open the Arduino IDE on your computer. Now you have to tell the IDE what board you are using, and how you want to connect to it.

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.

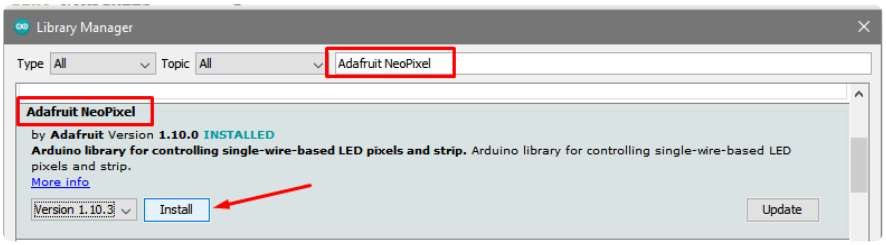


Install NeoPixel Library

Despite their popularity, NeoPixel RGB LEDs are not supported 'out of the box' in Arduino IDE! You will need to add support by installing the library. Good news it is very easy to do it. Go to the **Library Manager** here

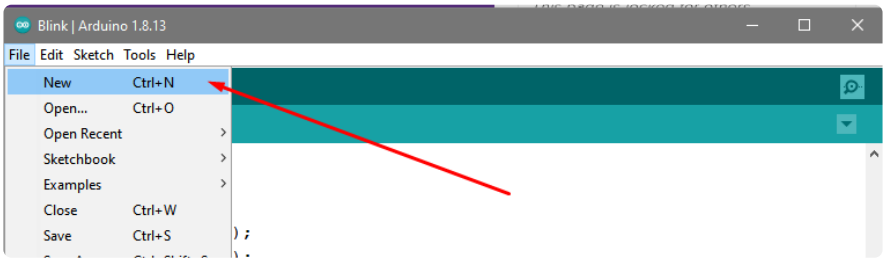


Search for and install the **Adafruit NeoPixel** library. It might not be first in the list so make sure you get the name matched up right!



New NeoPixel 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:

```
#include <Adafruit_NeoPixel.h>

// How many internal neopixels do we have? some boards have more than one!
#define NUMPIXELS        1

Adafruit_NeoPixel pixels(NUMPIXELS, PIN_NEOPIXEL, NEO_GRB + NEO_KHZ800);

// the setup routine runs once when you press reset:
void setup() {
  Serial.begin(115200);

#if defined(NEOPIXEL_POWER)
  // If this board has a power control pin, we must set it to output and high
```

```
  // in order to enable the NeoPixels. We put this in an #if defined so it can
  // be reused for other boards without compilation errors
  pinMode(NEOPIXEL_POWER, OUTPUT);
  digitalWrite(NEOPIXEL_POWER, HIGH);
#endif

  pixels.begin(); // INITIALIZE NeoPixel strip object (REQUIRED)
  pixels.setBrightness(20); // not so bright
}

// the loop routine runs over and over again forever:
void loop() {
  // say hi
  Serial.println("Hello!");

  // set color to red
  pixels.fill(0xFF0000);
  pixels.show();
  delay(500); // wait half a second

  // turn off
  pixels.fill(0x000000);
  pixels.show();
  delay(500); // wait half a second
}
```

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

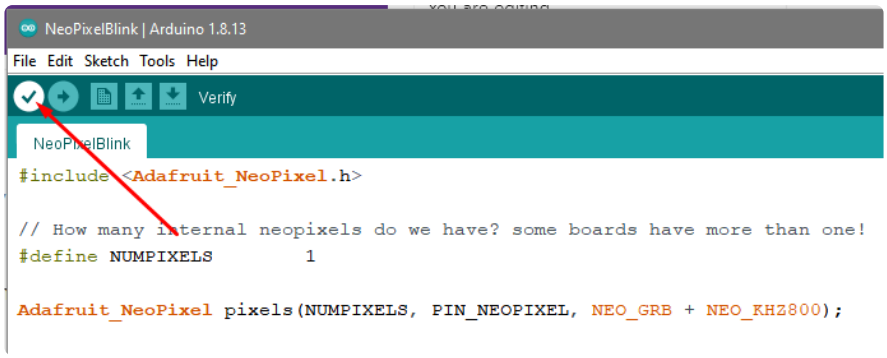
One note you'll see is that we reference the LED with the constant `PIN_NEOPIXEL` rather than a number. That's because each board could have the built in NeoPixels on a different pin and this makes the code a little more portable!

The NeoPixel LED is on pin 5. The NeoPixel power pin is on pin 8.

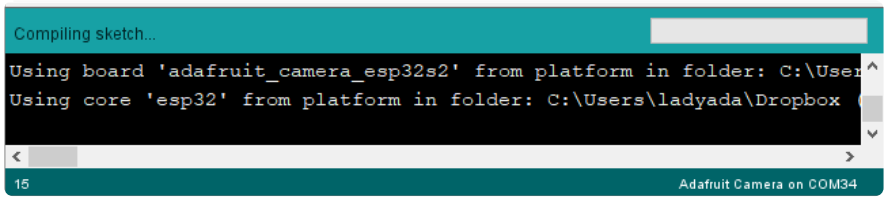
Verify (Compile) Sketch

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 these terms will be used interchangeably.

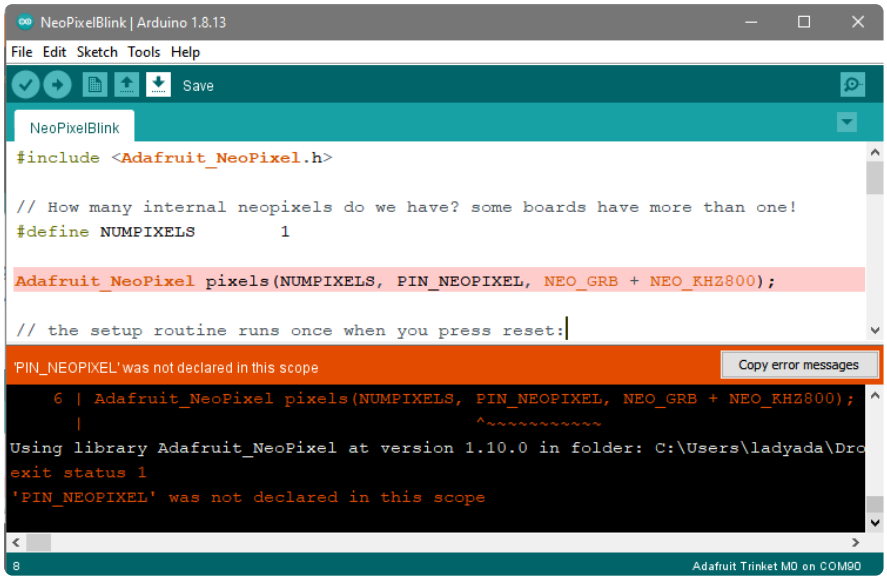


During verification/compilation, the computer will do a bunch of work to collect all the libraries and code and the results will appear in the bottom window of the IDE.

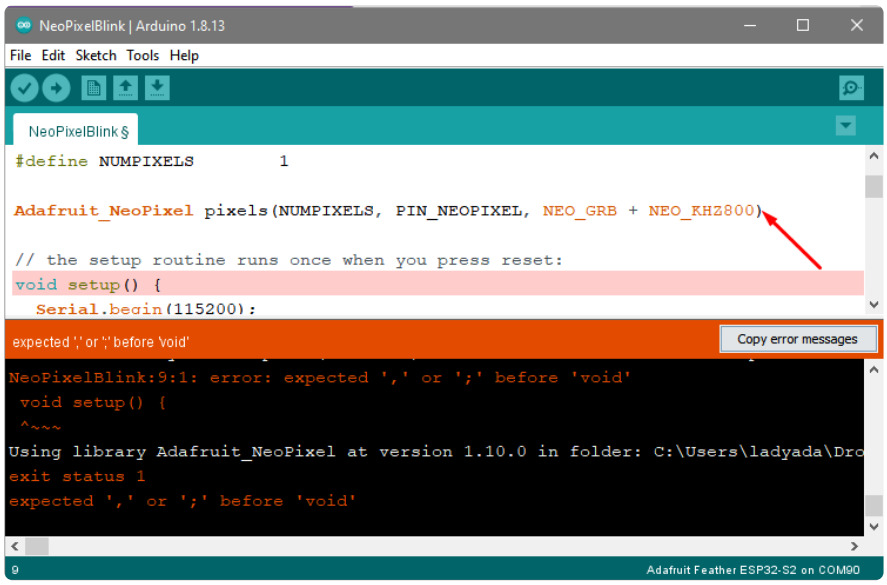


If something went wrong with compilation, you will get red warning/error text in the bottom window letting you know what the error was. It will also highlight the line with an error.

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

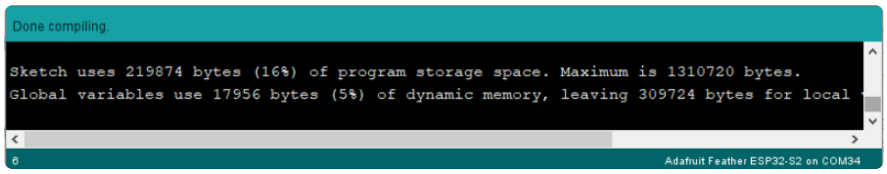


Here's another common error, in my haste I forgot to add a `;` at the end of a line. The compiler warns me that it's looking for one - note that the error is actually a few lines up!



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.



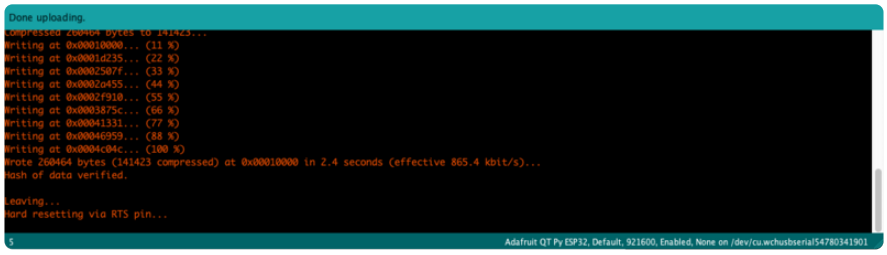
Upload Sketch

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, start with what it looks like: 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`

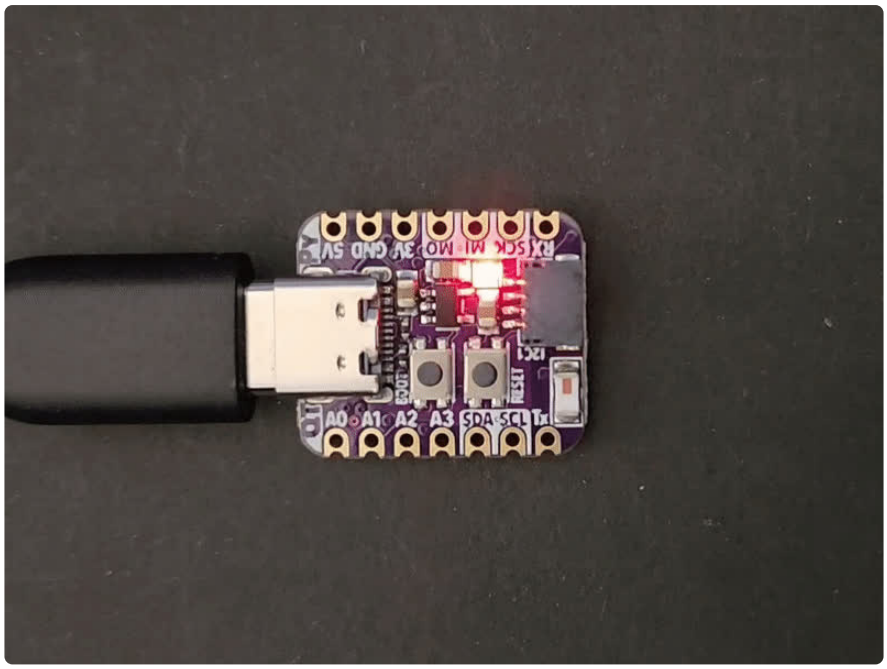
This could be a few things.

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.

Then give it another try!

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 Scan Test

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

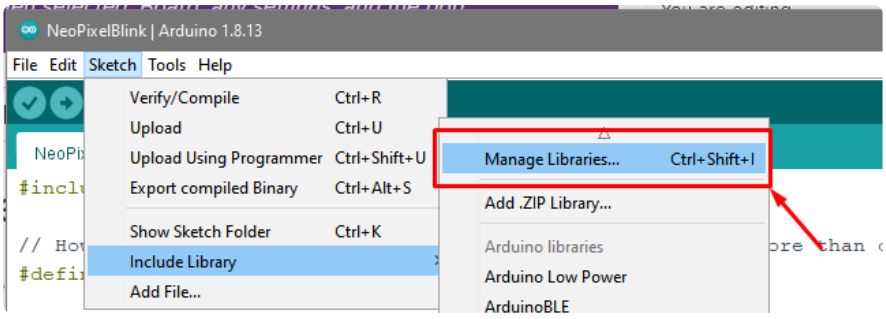
- **Do you have an address collision?** You can only have one board per address. So you cannot, say, connect two AHT20's to one I2C port because they have the same address and will interfere. Check the sensor or documentation for the address. Sometimes there are ways to adjust the address.
- **Does your 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 QT Py ESP32 Pico has two I2C ports. One port is on the SCL/SDA pin pads. The second port is on the STEMMA QT connector which is available as SCL1/SDA1.

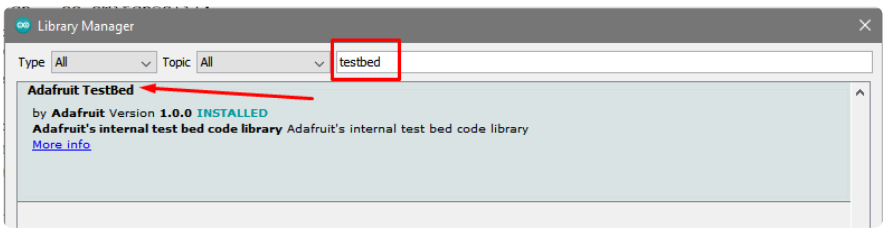
Perform an I2C scan!

Install TestBed Library

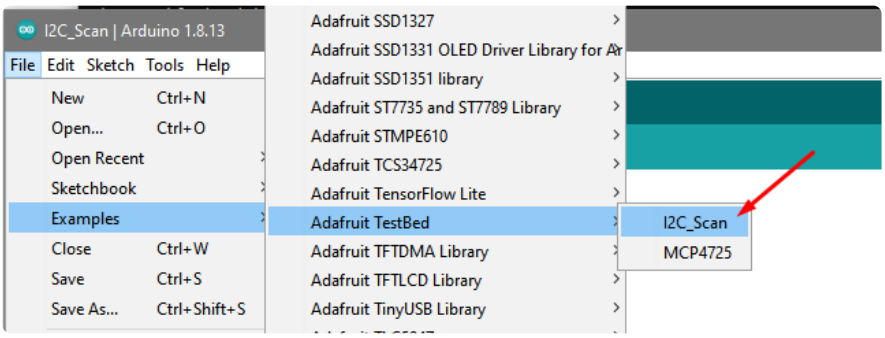
To scan I2C, the Adafruit TestBed library is used. This library and example just makes the scan a little easier to run because it takes care of some of the basics. You will need to add support by installing the library. Good news: it is very easy to do it. Go to the **Arduino Library Manager.**



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



Now open up the I2C Scan example



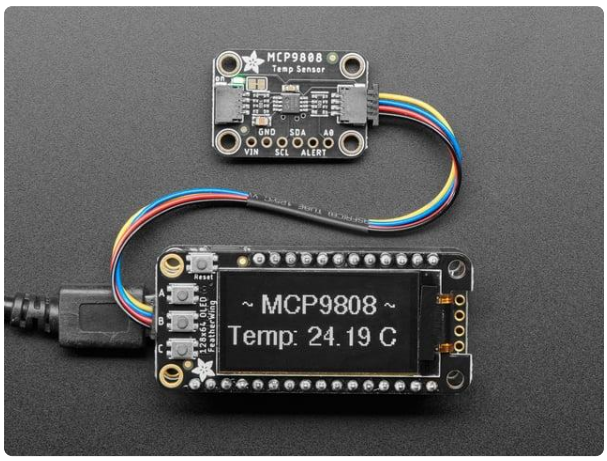

```
// SPDX-FileCopyrightText: 2023 Carter Nelson for Adafruit Industries
//
// SPDX-License-Identifier: MIT
// -------------------------------------
// i2c_scanner
//
// Modified from https://playground.arduino.cc/Main/I2cScanner/
// -------------------------------------

#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.

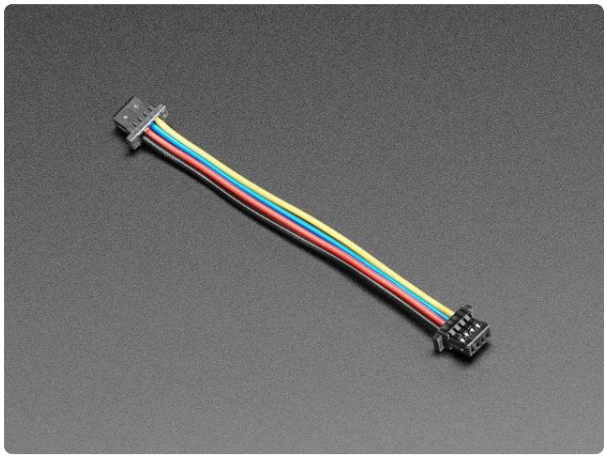
The first thing you'll want to do is get the sensor connected so your board has I2C to talk to.



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



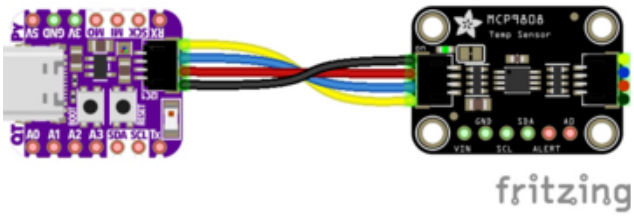
STEMMA QT / Qwiic JST SH 4-Pin Cable - 50mm Long

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.

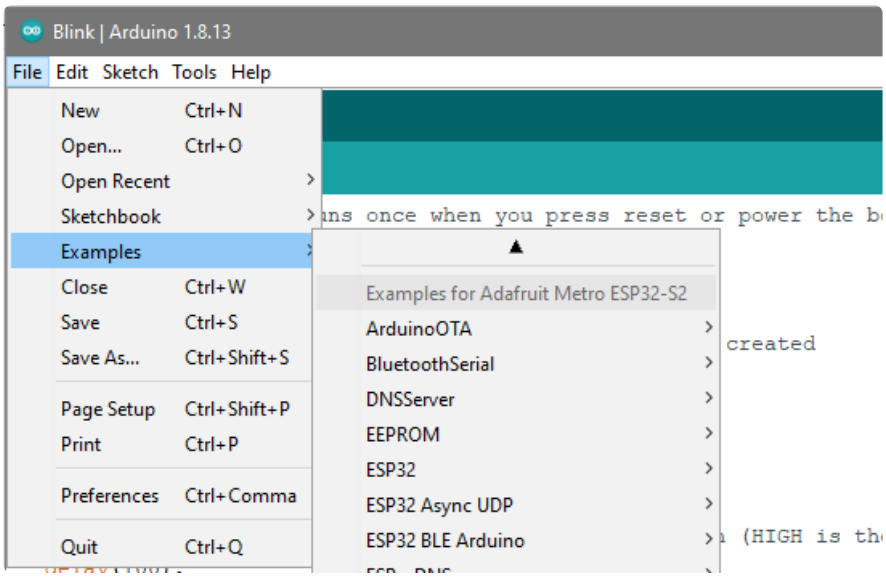


Simply plug a **STEMMA QT to STEMMA QT cable** into **the port on the QT Py**, and **a port on the MCP9808 breakout**.

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

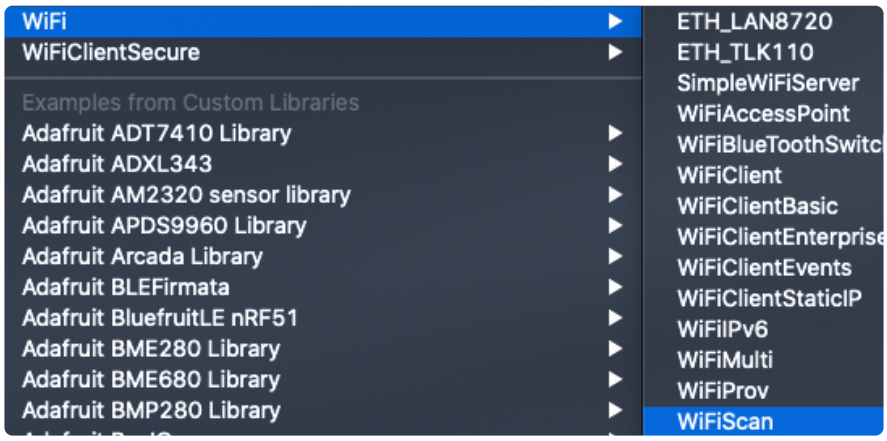
WiFi Test

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



Let's start by scanning the local networks.

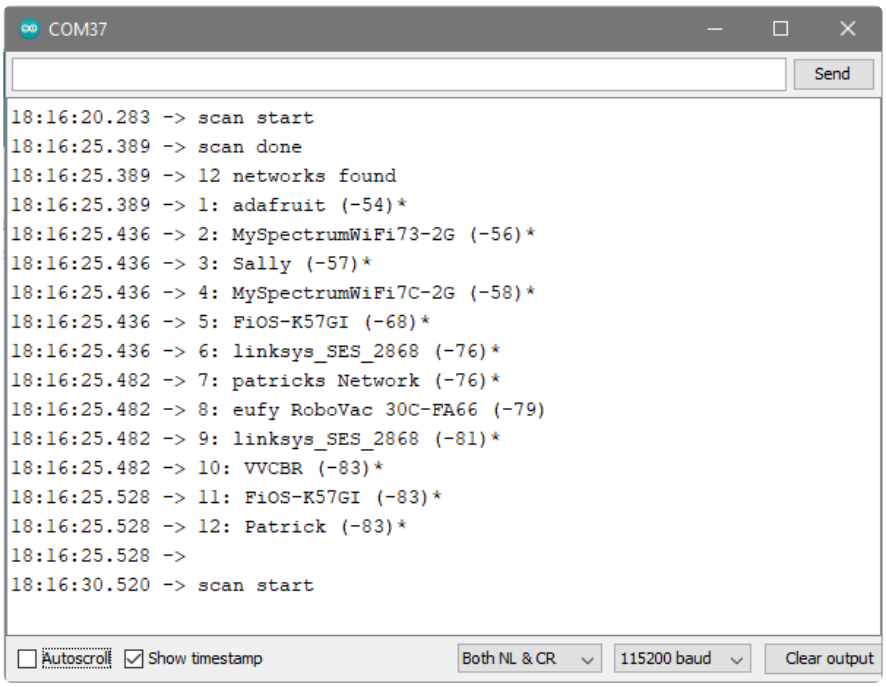
Load up the WiFiScan example under **Examples->Examples for YOUR BOARDNAME->WiFi->WiFiScan**



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

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

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



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

WiFi Connection Test

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

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

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


/*
  Web client

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

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

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

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

#include <WiFi.h>

// Enter your WiFi SSID and password
char ssid[] = "YOUR_SSID";             // your network SSID (name)
char pass[] = "YOUR_SSID_PASSWORD";    // your network password (use for WPA, or
```

```
use as key for WEP)
int keyIndex = 0;                       // your network key Index number (needed
only for WEP)

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

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

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

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

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

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

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

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

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

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

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

```
}


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

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

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

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

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

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

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

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

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

disconnecting from server.
```

Secure Connection Example

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

Copy and paste it into the Arduino IDE:

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

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

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

Written by Arturo Guadalupi
last revision November 2015

*/

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

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

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

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

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

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

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

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

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

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

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

uint32_t bytes = 0;

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

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

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


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

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

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

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

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

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

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

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

WipperSnapper Setup

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

What is WipperSnapper

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

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

From there, you can add components to your board such as buttons, switches, potentiometers, sensors, and more! Components are dynamically added to hardware,

so you can immediately start interacting, logging, and streaming the data your projects produce without writing code.

Sign up for Adafruit.io

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

Install USB Driver

Install CP2104 / CP2102N USB Driver

Many ESP32 boards have a USB-to-Serial converter that talks to the chip itself, and will need a driver on your computer's operating system. The driver is available for Mac, Windows, and Linux.

Click here to download the CP2104/ CP2102N driver

https://adafru.it/vrf

Install CH9102 / CH34X USB Driver

Newer ESP32 boards have a different USB-to-serial converter that talks to the chip itself, and will need a driver on your computer's operating system. The driver is available for Mac and Windows. It is already built into Linux.

If you would like more detail, check out the guide on installing these drivers (https://adafru.it/-f8).

Click here to download the Windows driver

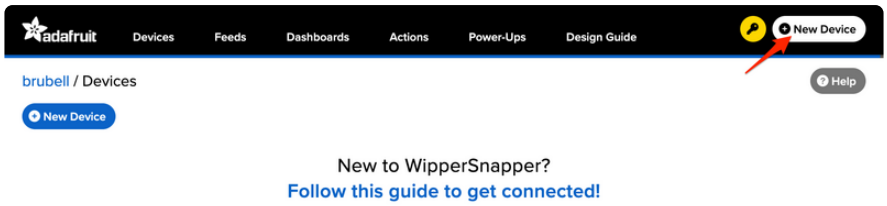
https://adafru.it/-f9

Click here to download the Mac driver

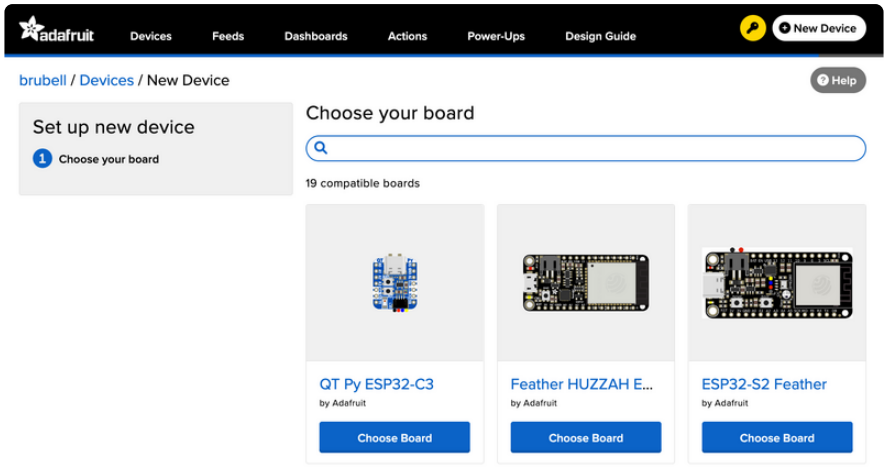
https://adafru.it/-ed

Add a New Device to Adafruit IO

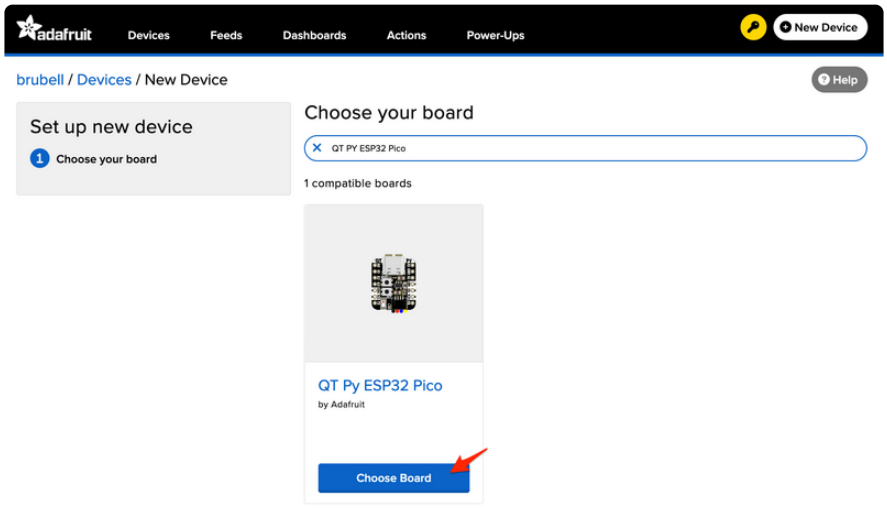
Log into your Adafruit IO (https://adafru.it/fsU) account. Click the New Device button at the top of the page.



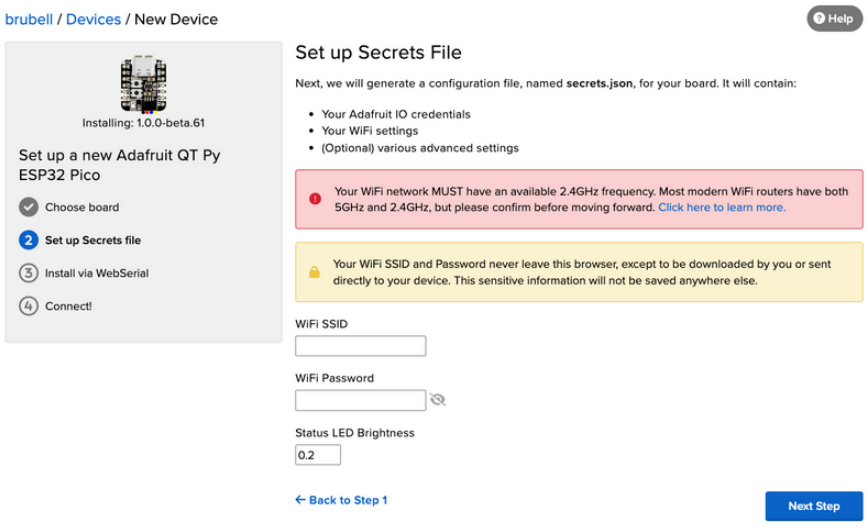
After clicking New Device, you should be on the board selector page. This page displays every board that is compatible with the WipperSnapper firmware.



In the board selector page's search bar, search for the QT Py ESP32 Pico. Once you've located the board you'd like to install WipperSnapper on, click the Choose Board button to bring you to the self-guided installation wizard.



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

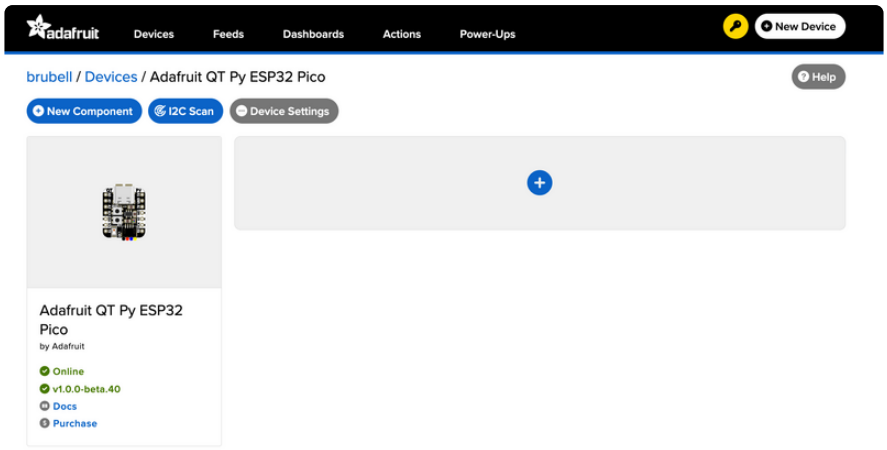


If the installation was successful, a popover should appear displaying that your board has successfully been detected by Adafruit IO.

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

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

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



Feedback

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

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

Troubleshooting

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

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

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

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

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

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

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

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

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

"Uninstalling" WipperSnapper

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

Moving from WipperSnapper to CircuitPython

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

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

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

Moving from WipperSnapper to Arduino

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

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

Arduino IDE Setup

https://adafru.it/10aQ

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

Upload Arduino "Blink" Sketch

https://adafru.it/10aR

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

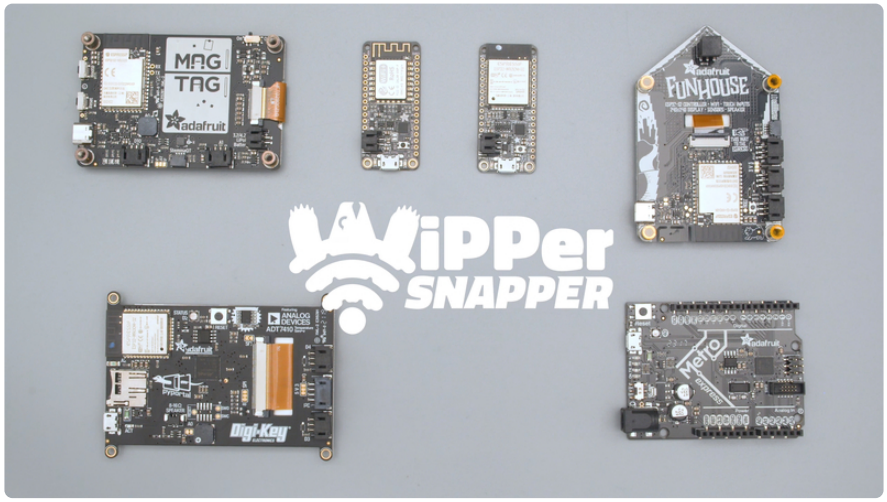
Factory Resetting a WipperSnapper Board

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

Factory Reset Adafruit QT Py ESP32 Pico

https://adafru.it/10aS

WipperSnapper Essentials



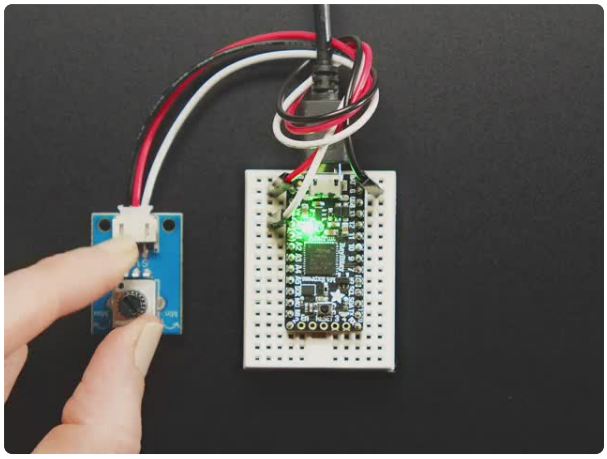
You've installed WipperSnapper firmware on your board and connected it to Adafruit IO. Next, let's learn how to use Adafruit IO!

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

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

Parts

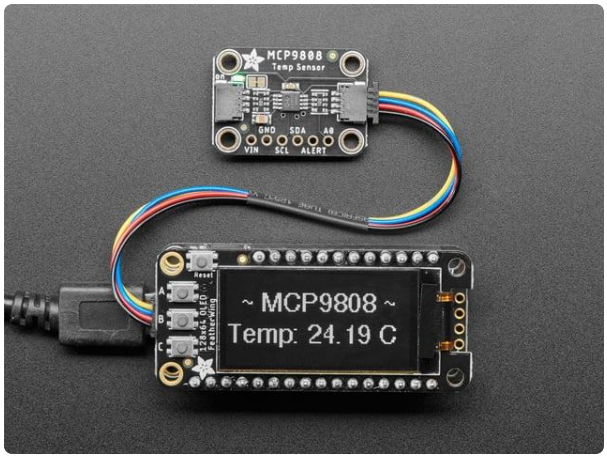
The following parts are **required** to complete the WipperSnapper essentials pages for this board:



STEMMA Wired Potentiometer Breakout Board - 10K ohm Linear

For the easiest way possible to measure twists, turn to this STEMMA potentiometer breakout (ha!). This plug-n-play pot comes with a JST-PH 2mm connector and a matching

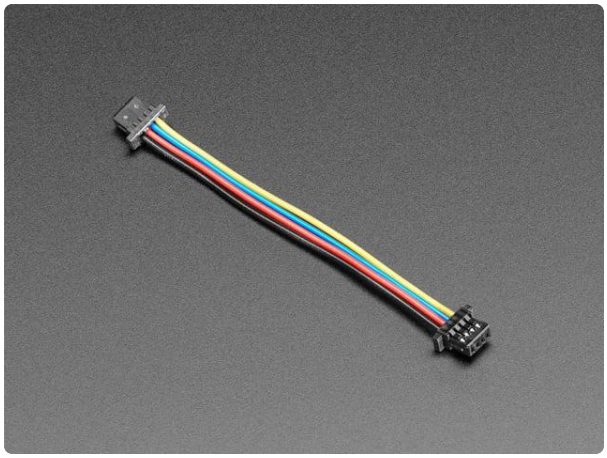
https://www.adafruit.com/product/4493



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...

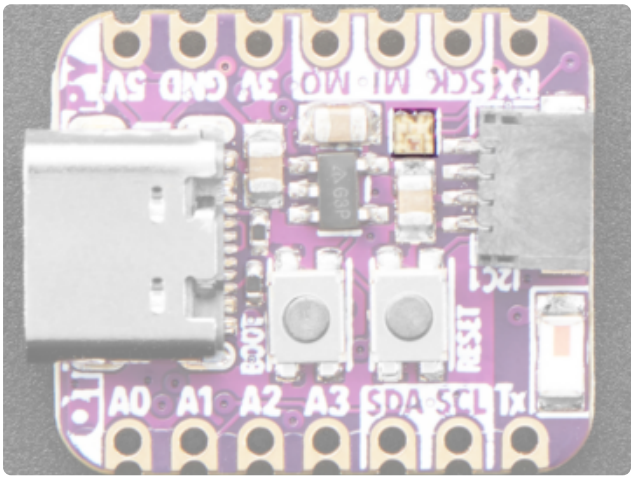
https://www.adafruit.com/product/4399

NeoPixel LED

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

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

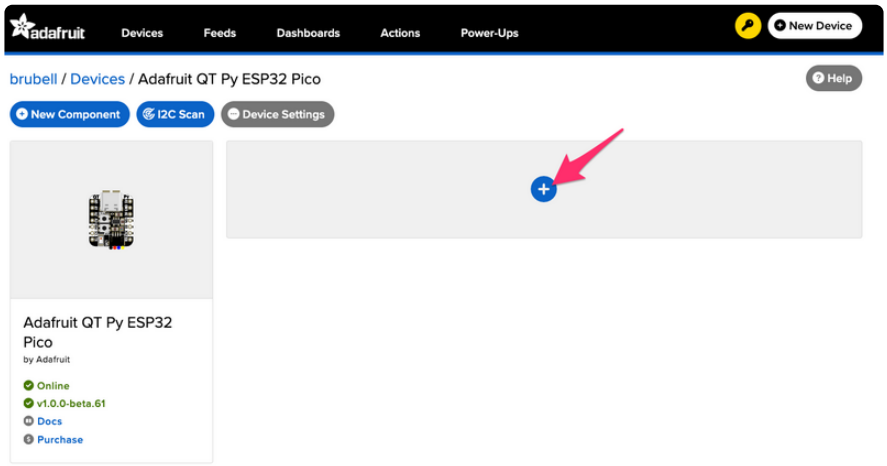
Where is the NeoPixel on my board?

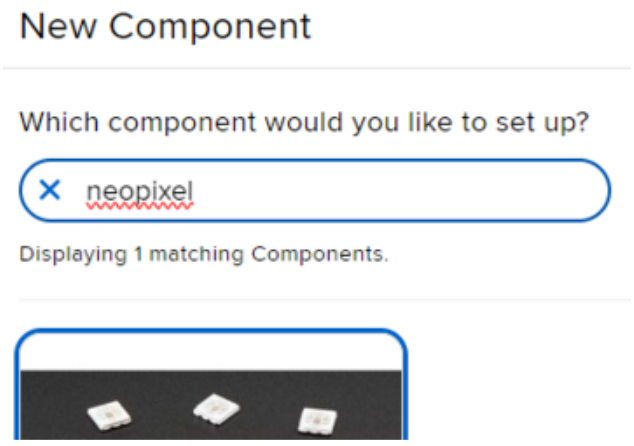


Above the SCK and MI on the silk, is the **RGB NeoPixel LED** (outlined in black).

Create the NeoPixel Component

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





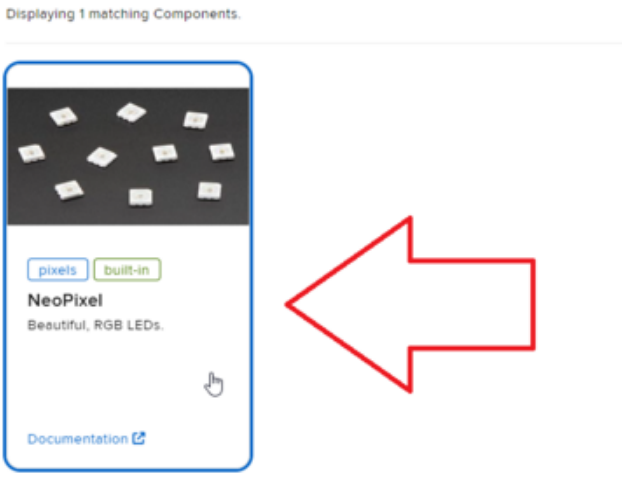
Search for the component name by entering `neopixel` into the text box on the component picker, the list of components should update as soon as you stop typing

Filtering and searching for components

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

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

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

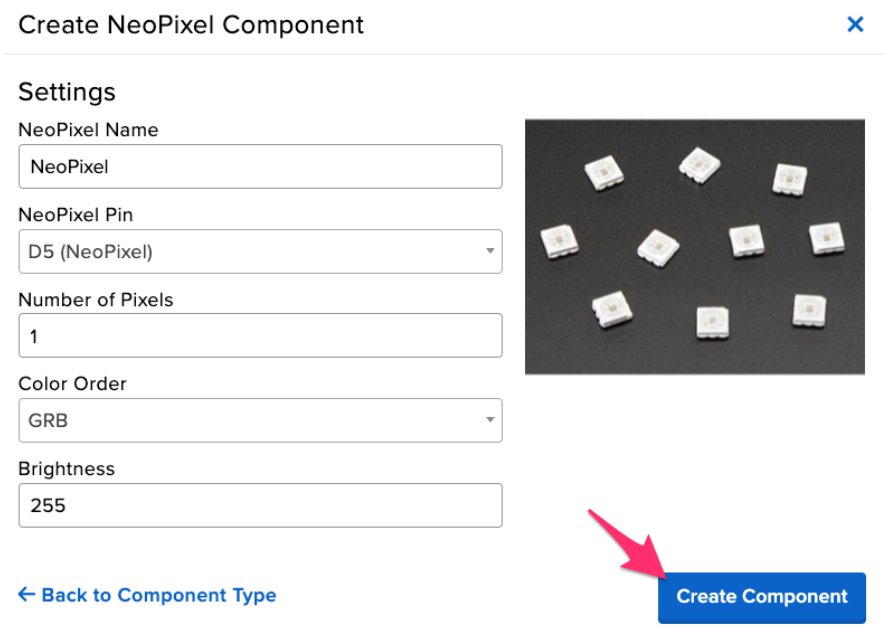


Select the **NeoPixel** from the list of results to go to the component configuration page.

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

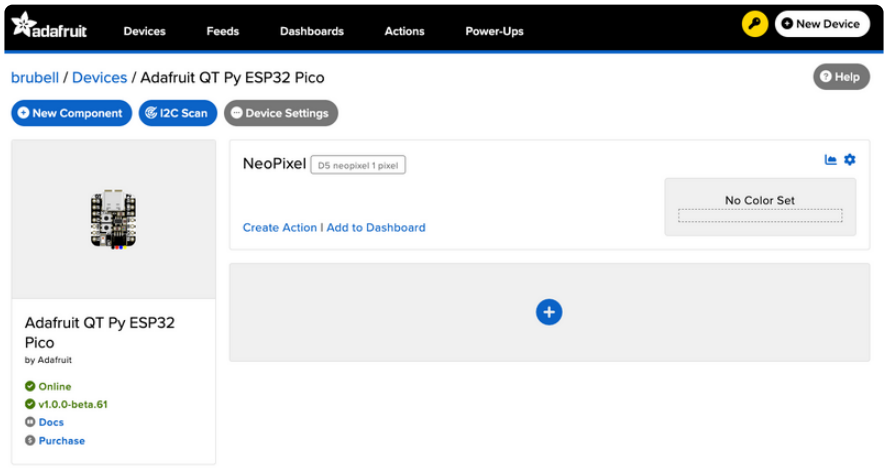
The board NeoPixel pin is automatically found and selected.

Click **Create Component**



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

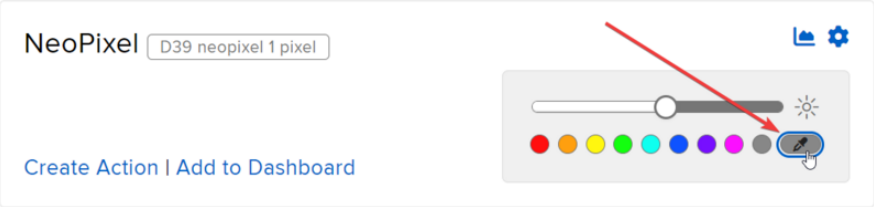
The Device page shows the NeoPixel component.



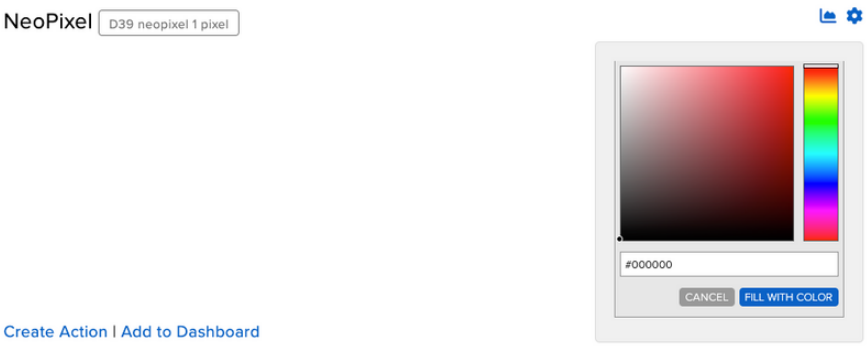
Set the NeoPixel's RGB Color

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

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



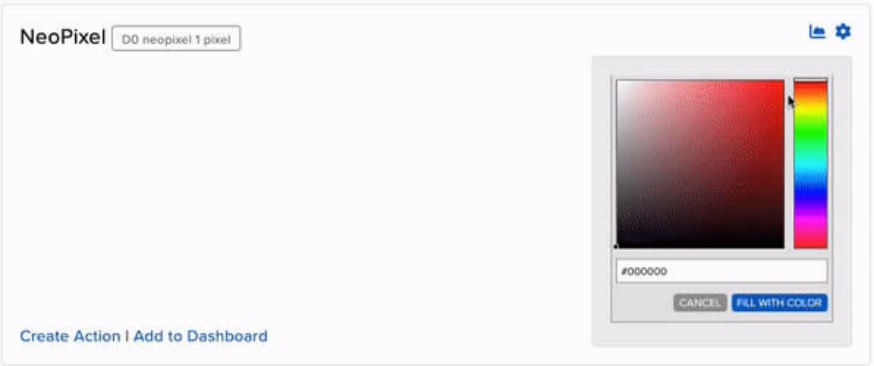
A color picker pops open! Next: learning how Adafruit IO uses hex color codes to represent the colors on your NeoPixel.



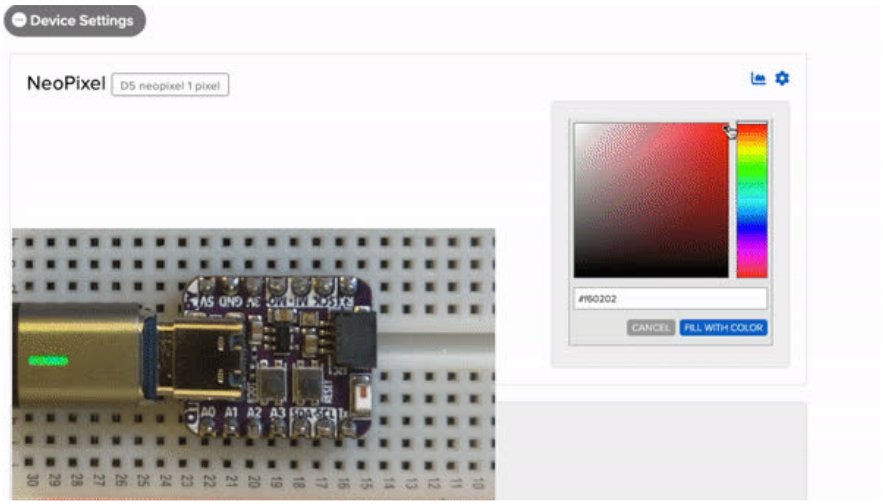
Hex Colors 101

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

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

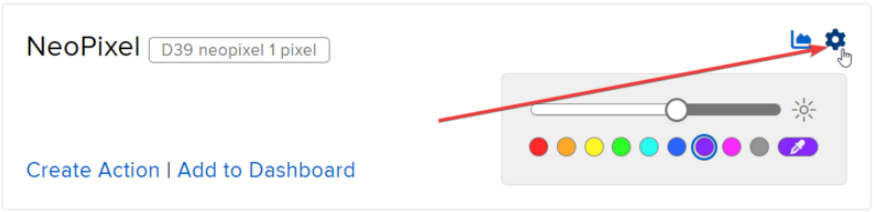


When you're ready to set the color of your device's NeoPixel, click **FILL WITH COLOR**. The NeoPixel will immediately glow!



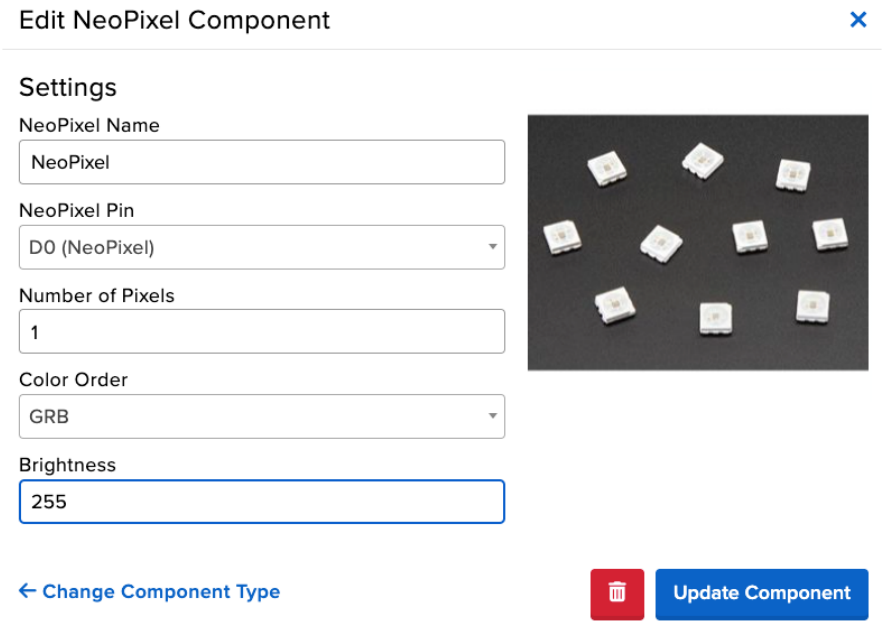
Set NeoPixel Brightness

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



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

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



Read a Push-button

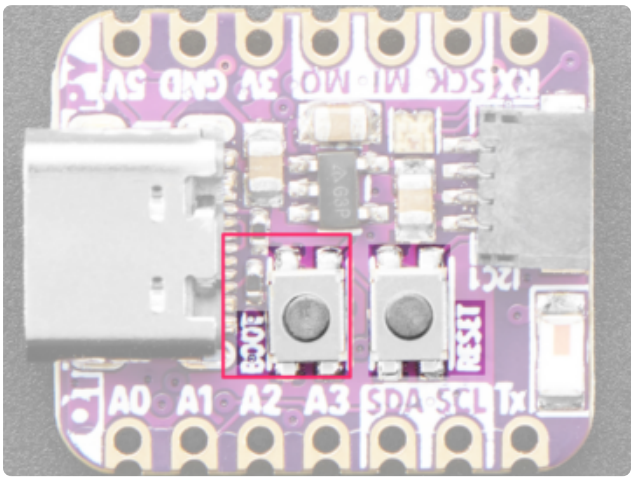
This demo shows reading the state of a push-button from WipperSnapper. But the same kind of control can be used for reading switches, break beam sensors, and other digital sensors.

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

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

Button Location

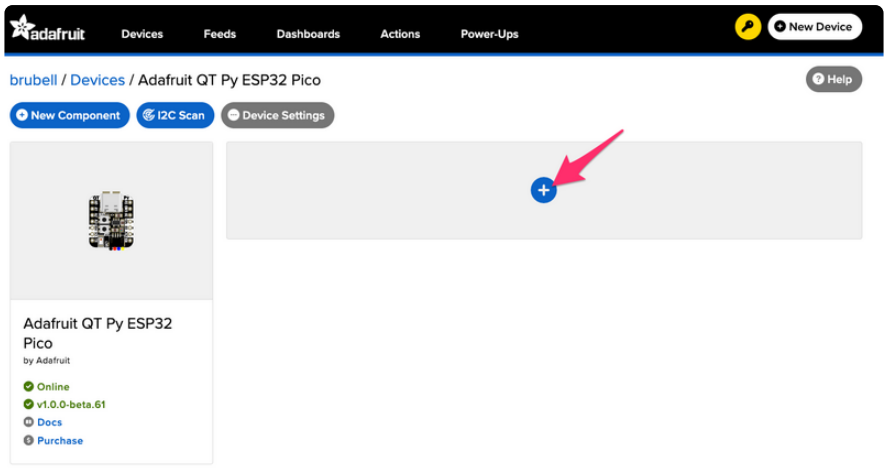
This example uses the board's built-in push-button and internal pull-up resistor instead of wiring a push-button up.

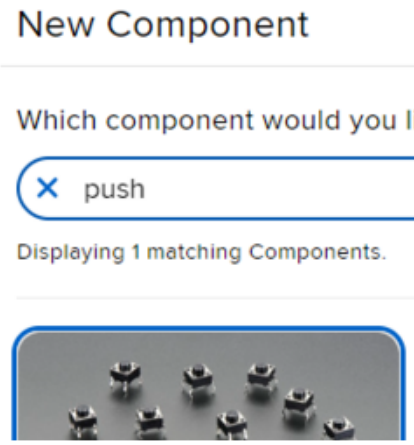


The BOOT button on the QT Py ESP32 Pico is located underneath the USB-C port.

Create a Push-button Component on Adafruit IO

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





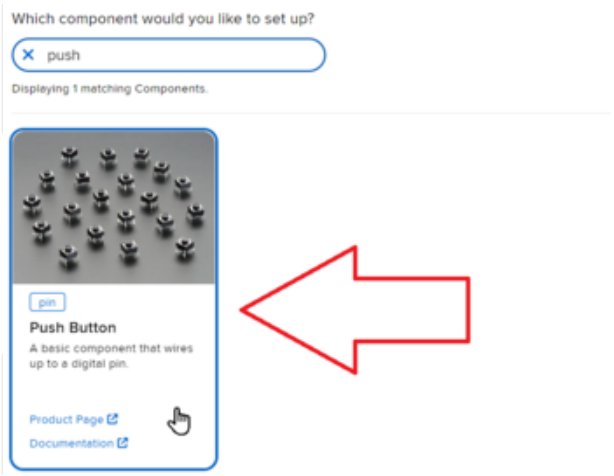
Search for the component name by entering `push` into the text box on the component picker, the list of components should update as soon as you stop typing.

Filtering and searching for components

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

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

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

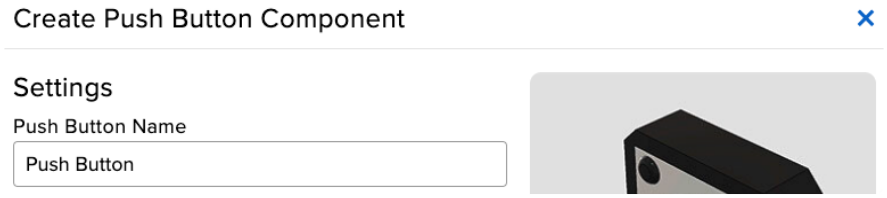


Select the **Push Button** from the list of results to go to the component configuration page.

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

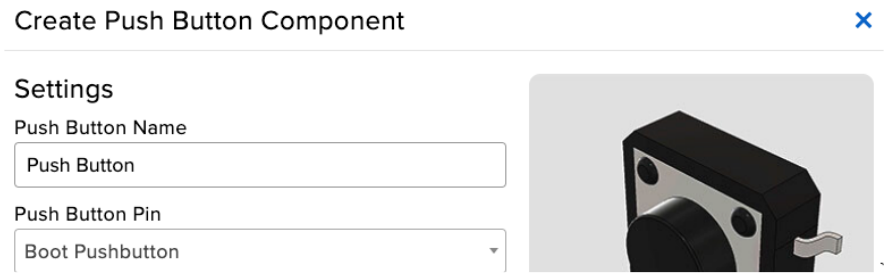
The "Create Push Button Component" form presents you with options for configuring the push button.

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

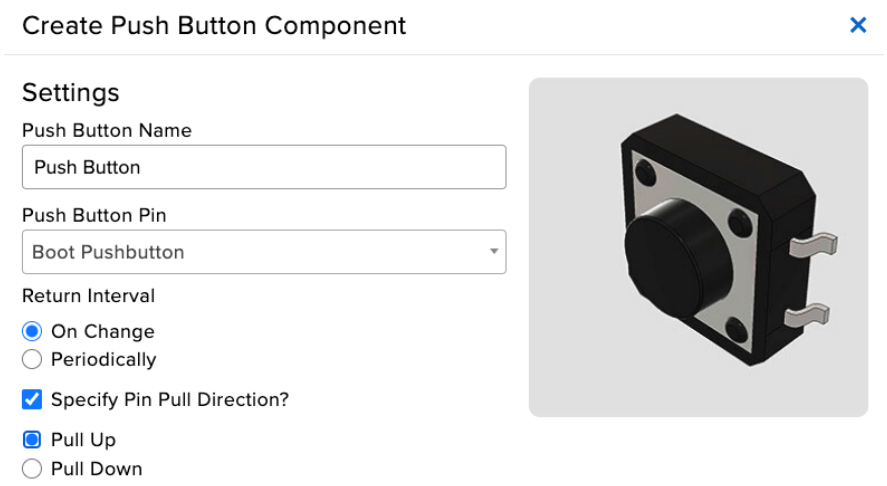


The Return Interval dictates how frequently the value of the push-button will be sent from the board to Adafruit IO.

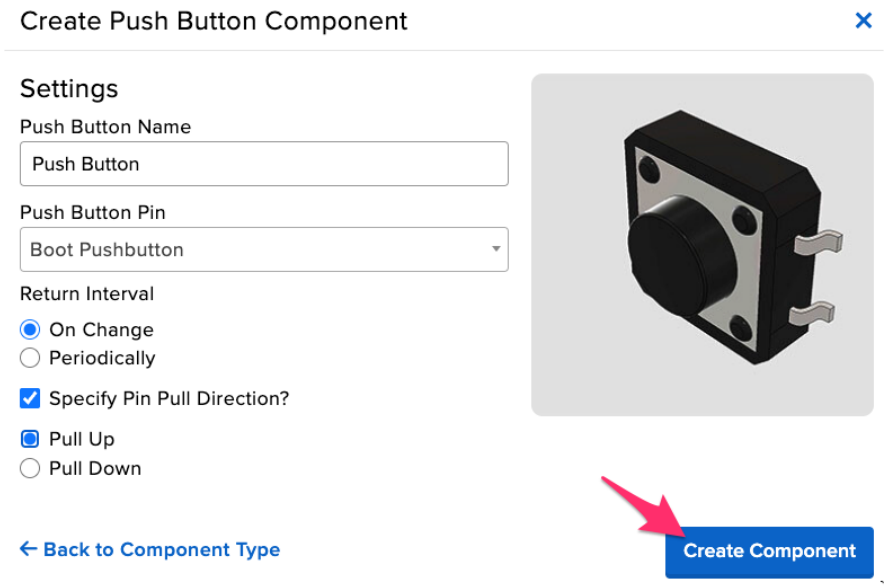
For this example, you will configure the push button's value to be only sent when the value changes (i.e. when it's either pressed or depressed).



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

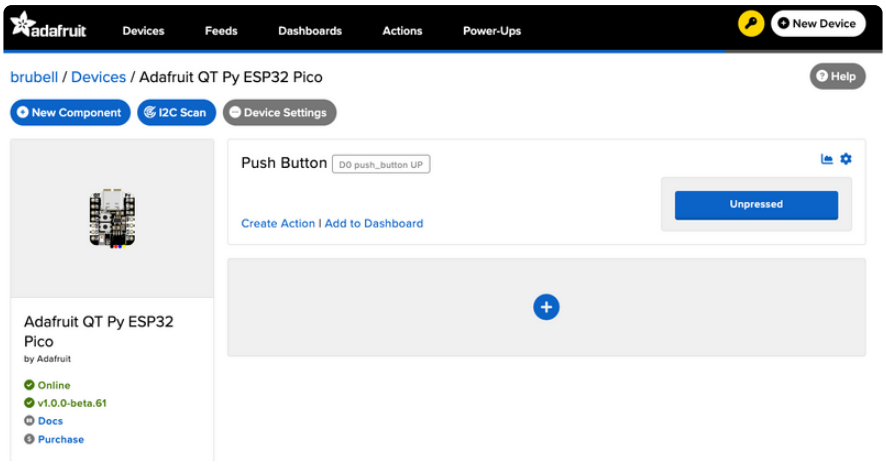


Make sure the form's settings look like the following screenshot. Then, click **Create Component**.

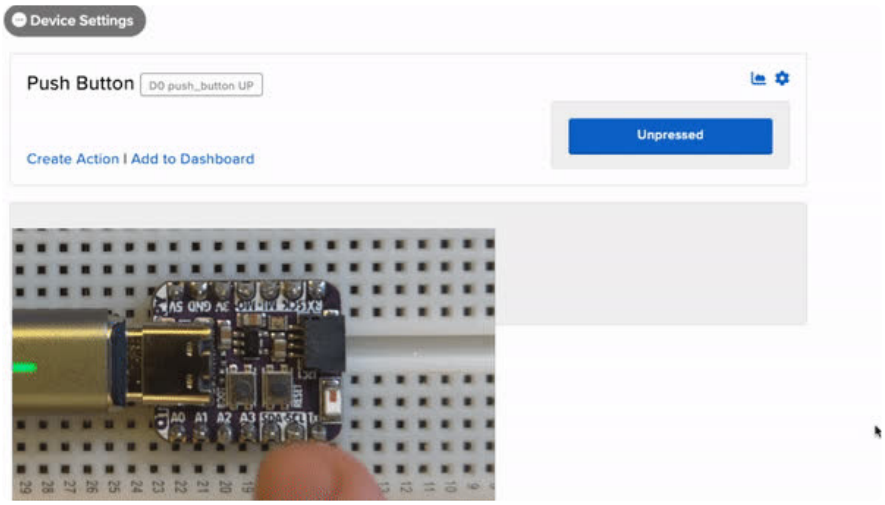


Adafruit IO sends a command to your WipperSnapper board, telling it to configure the GPIO pin you selected to behave as a digital input pin and to enable it to pull up the internal resistor.

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



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



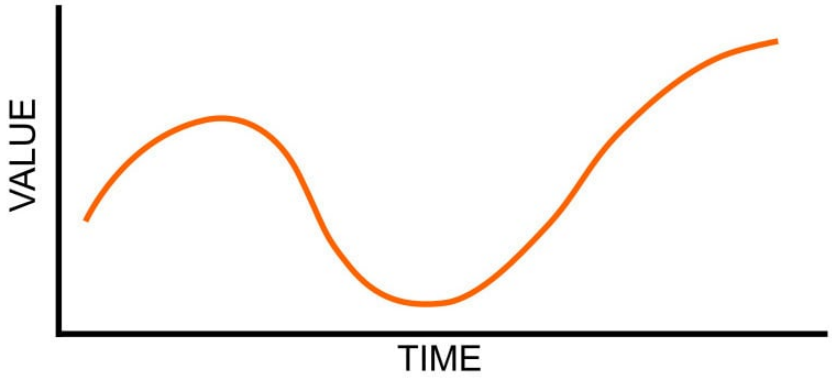
Analog Input

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

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

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

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



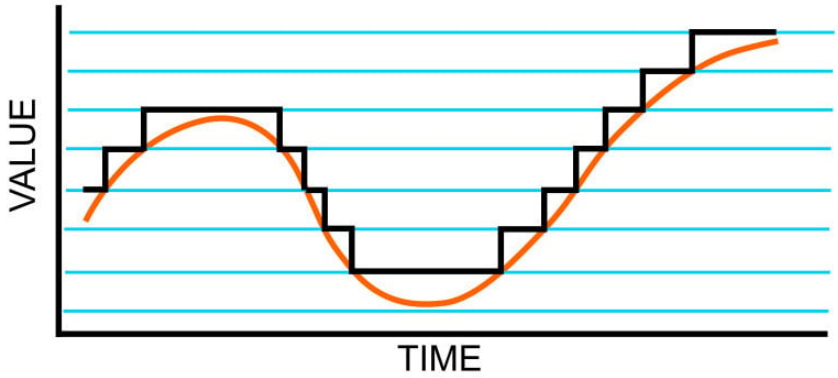
Analog signals are continuous values which means they can be an infinite number of different voltages. Think of analog signals like a floating point or fractional number, they can smoothly transiting to any in-between value like 1.8V, 1.81V, 1.801V, 1.8001V, 1.80001V and so forth to infinity.

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

Analog to Digital Converter (ADC)

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

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

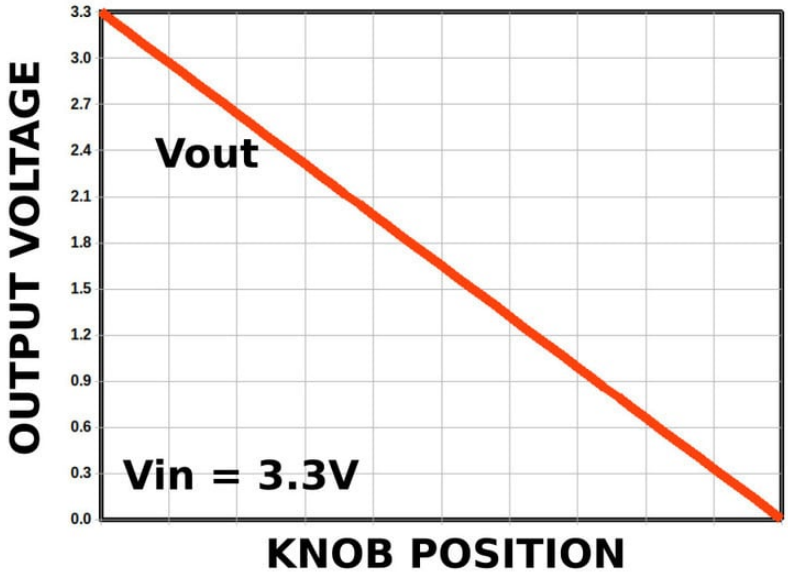


Once that analog signal has been converted by the ADC, the microcontroller can use those digital values any way you like!

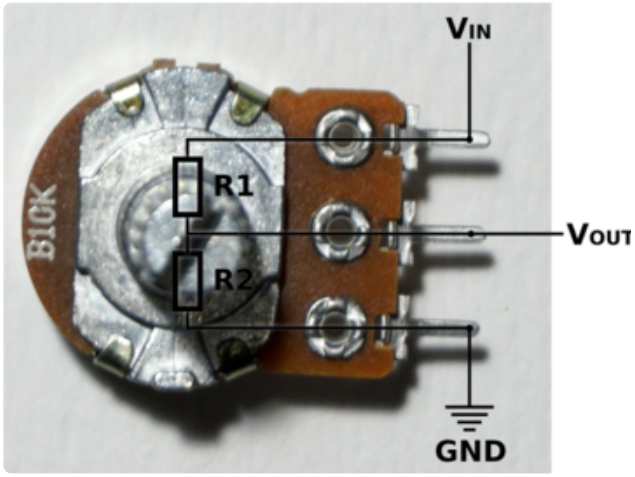
Potentiometers

A potentiometer is a small variable resistor that you can twist a knob or shaft to change its resistance. It has three pins. By twisting the knob on the potentiometer you can change the resistance of the middle pin (called the wiper) to be anywhere within the range of resistance of the potentiometer.

By wiring the potentiometer to your board in a special way (called a voltage divider) you can turn the change in resistance into a change in voltage that your board's analog to digital converter can read.



To wire up a potentiometer as a voltage divider:

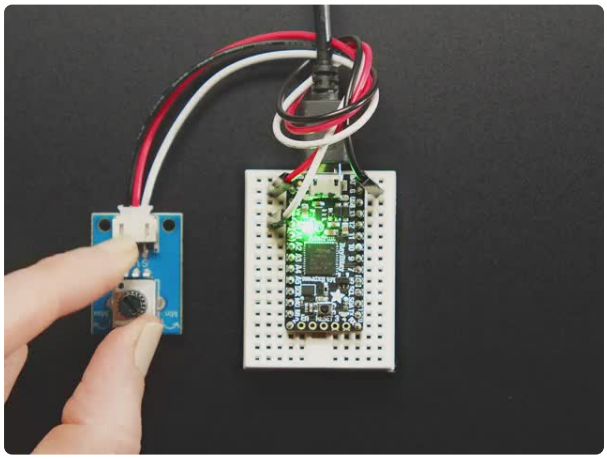


Connect **one outside pin** to **ground**
Connect **the other outside pin** to **voltage in (e.g. 3.3V)**
Connect **the middle pin** to **an analog pin (e.g. A0)**

Hardware

In addition to your microcontroller board, you will need the following hardware to follow along with this example.

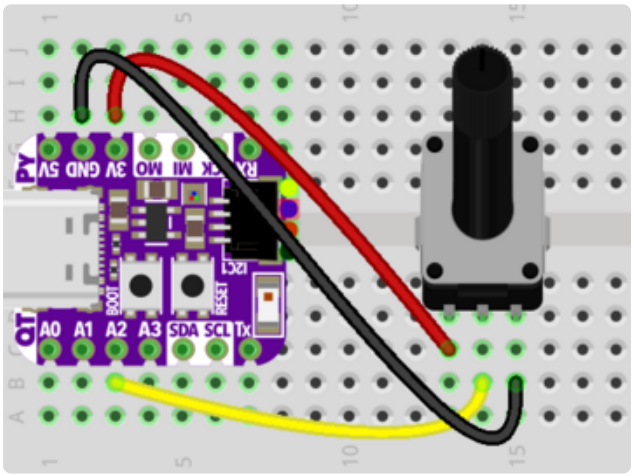
Potentiometer



STEMMA Wired Potentiometer Breakout Board - 10K ohm Linear
For the easiest way possible to measure twists, turn to this STEMMA potentiometer breakout (ha!). This plug-n-play pot comes with a JST-PH 2mm connector and a matching
https://www.adafruit.com/product/4493

Wire Up the Potentiometer

Connect the potentiometer to your board as follows.



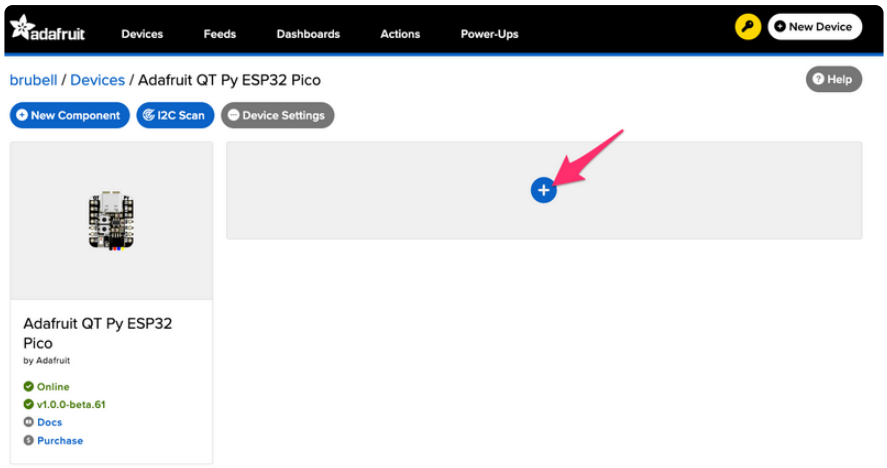
QT Py 3V to **potentiometer left pin**
QT Py A2 to **potentiometer middle pin**
QT Py GND to **potentiometer right pin**

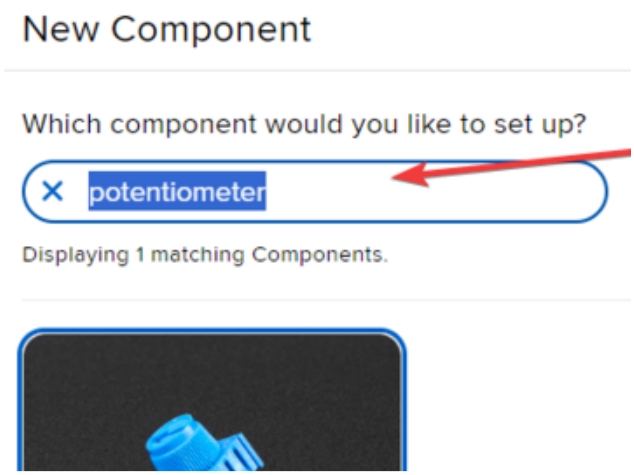
Note: On WipperSnapper, the ESP32's ADC1 pins are unusable for analog input since WiFi is constantly running.

For information on what Analog Input pins can (and can't) be used on this board, click here >>> (https://adafru.it/18eQ)

Create a Potentiometer Component on Adafruit IO

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





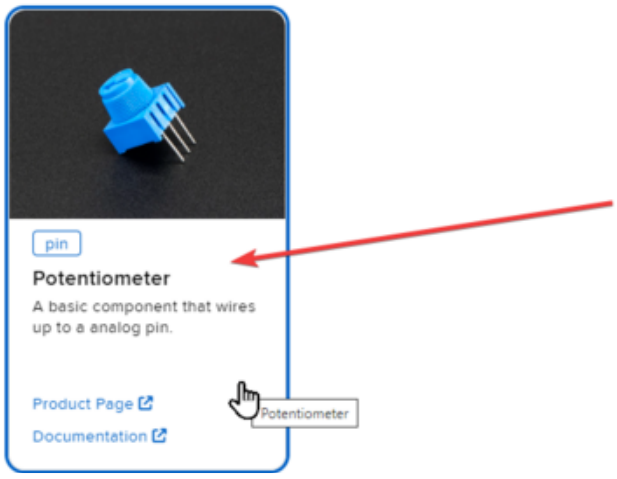
Search for the component name by entering `potentiometer` into the text box on the component picker, the list of components should update as soon as you stop typing.

Filtering and searching for components

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

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

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



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

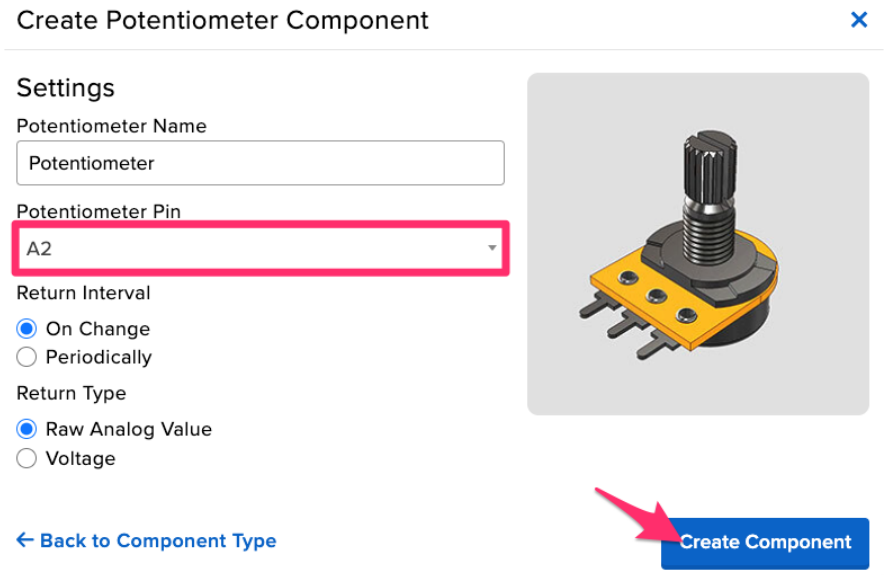
There will be a back button if you select the wrong component, and you can use the Edit component icon (⚙) on the device page to update the component configuration in the future.

On the Create Potentiometer Component form:

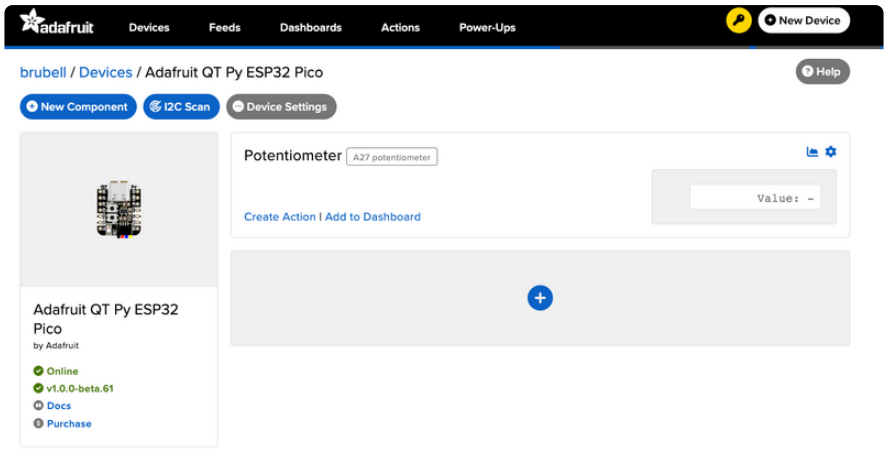
- Set **Potentiometer Pin** to **A2**

- Select **"On Change"** as the **Return Interval**
- Select **Raw Analog Value** as the **Return Type**

Then, click Create Component

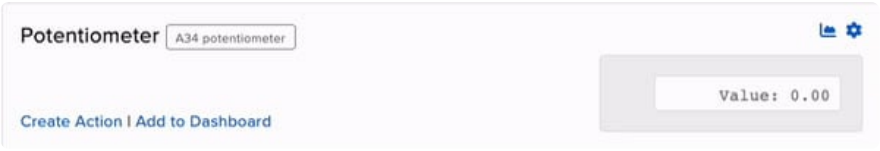


The potentiometer component appears on your board page! Next, learning to read values from it.



Read Analog Pin Values

Rotate the potentiometer to see the value change.

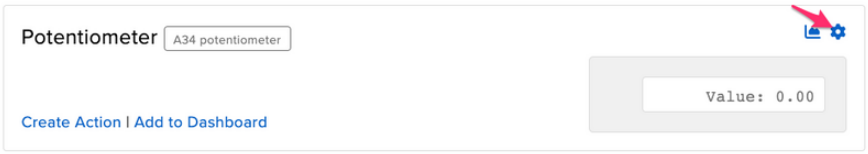


What do these values mean?

WipperSnapper reports ADC "raw values" as 16-bit unsigned integer values. Your potentiometer will read between 0 (twisting the pot to the leftmost position) and 65535 (twisting the pot to the rightmost position).

Read Analog Pin Voltage Values

You can update the potentiometer component (or any analog pin component in WipperSnapper) to report values in Volts. To do this, on the right-hand side of the potentiometer component, click the cog button.



Under **Return Type**, click Voltage.

Click Update Component to send the updated component settings to your board running WipperSnapper.

Now, twist the potentiometer to see the value reported to Adafruit IO in Volts!

I2C Sensors

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

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

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

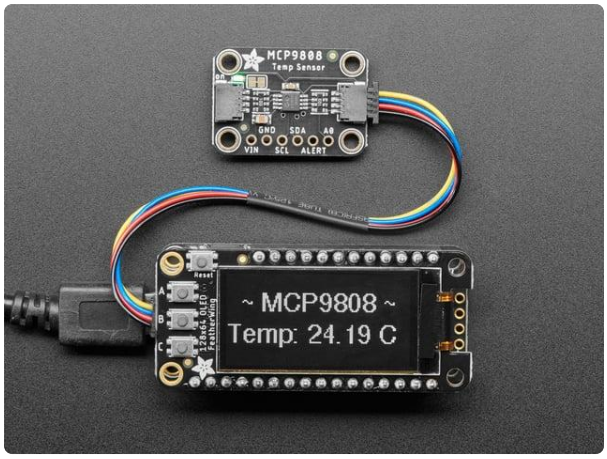
The WipperSnapper firmware supports a number of I2C sensors, viewable in list format here (https://adafru.it/Zbq).

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

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

Parts

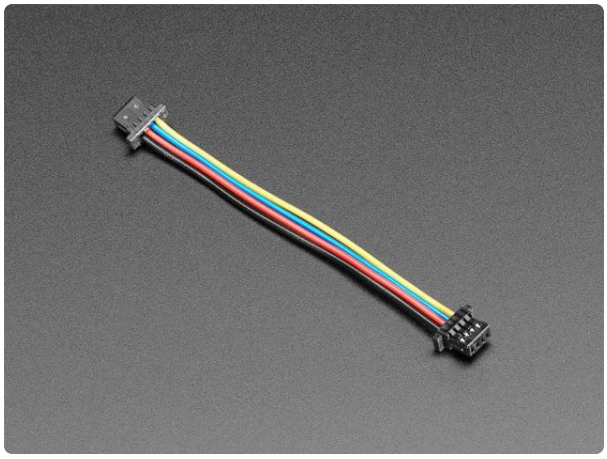
You will need the following parts to complete this page:



Adafruit MCP9808 High Accuracy I2C Temperature Sensor Breakout

The MCP9808 digital temperature sensor is one of the more accurate/precise we've ever seen, with a typical accuracy of ±0.25°C over the sensor's -40°C to...

https://www.adafruit.com/product/5027

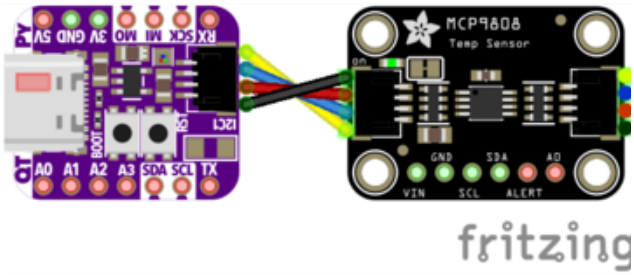


STEMMA QT / Qwiic JST SH 4-Pin Cable - 50mm Long

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

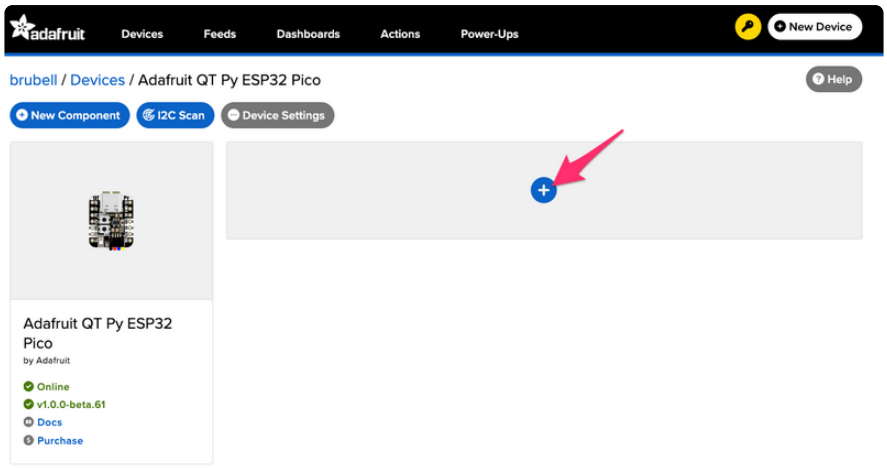


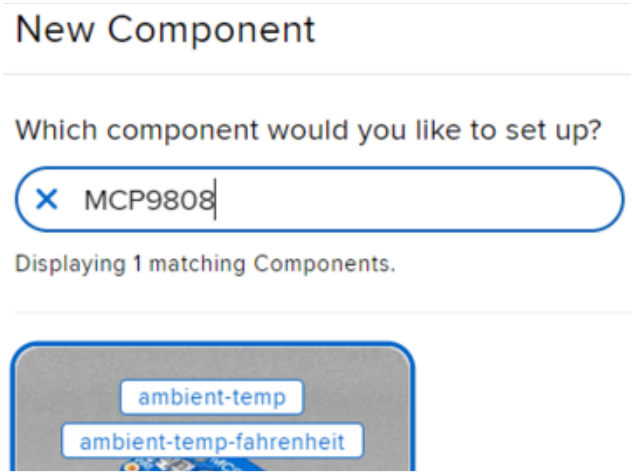
Connect the **QT Py's STEMMA QT Port** to **MCP9808's STEMMA QT Port**

NOTE: At the time of writing, **the QT Py's second I2C port** (mapped to the SCL/SDA pins on the board) **is not supported by WipperSnapper.** Only the STEMMA I2C port is supported.

Add an MCP9808 Component

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





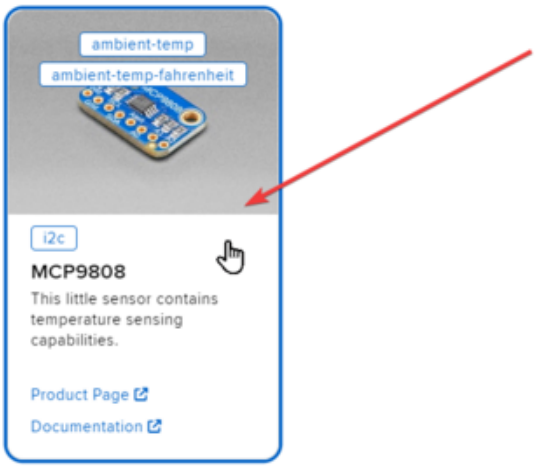
Search for the component name by entering `MCP9808` into the text box on the component picker, the list of components should update as soon as you stop typing.

Filtering and searching for components

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

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

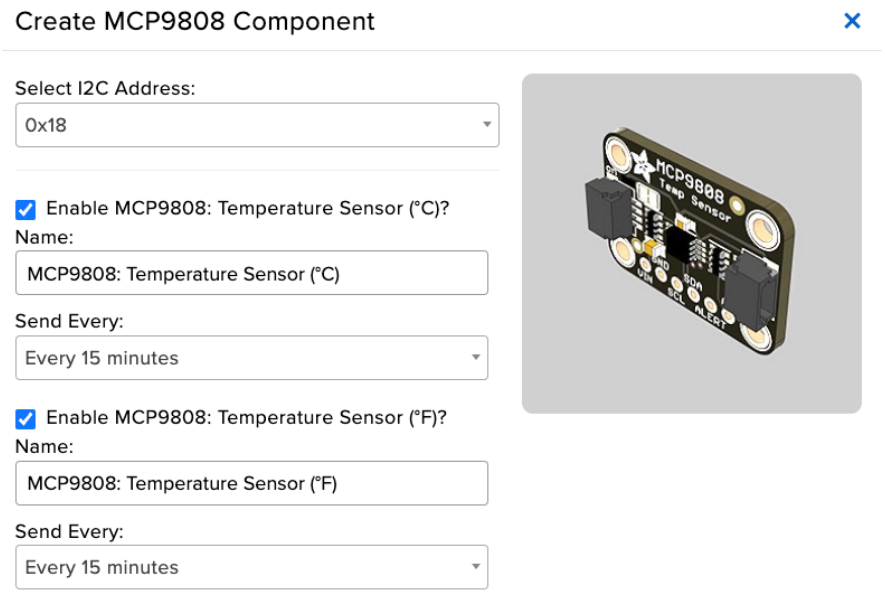
There are added product and documentation links for every component, follow the links beneath the component descriptions to be taken to the appropriate product page or Learn Guide.



Select the **MCP9808** from the list of results to go to the component configuration page.

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

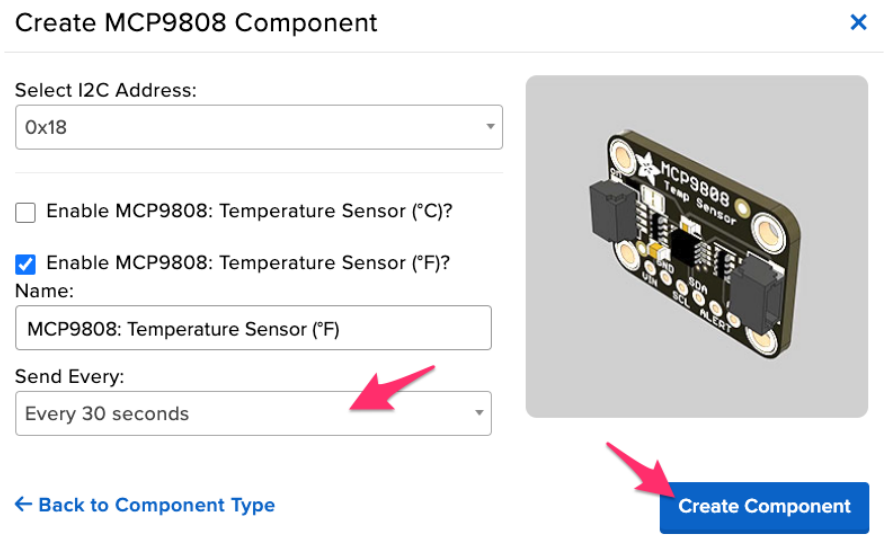
On the component configuration page, the MCP9808's I2C sensor address should be listed along with the sensor's settings.



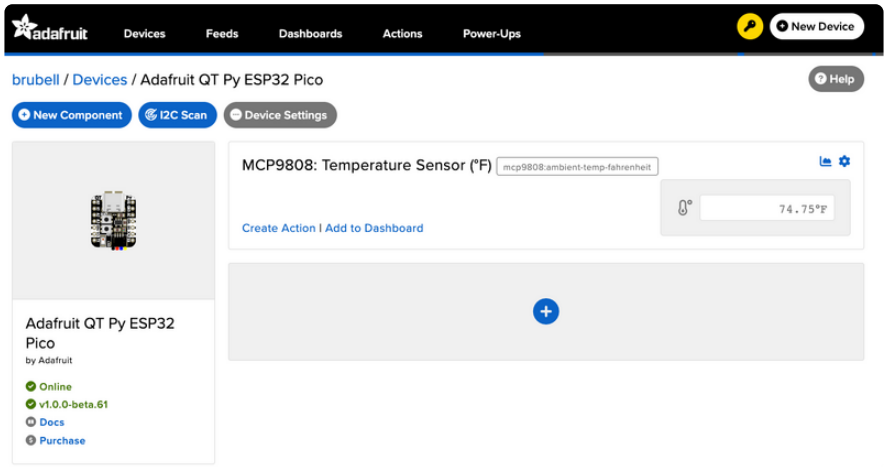
The MCP9808 sensor can measure ambient temperature. This page has individual options for reading the ambient temperature, in either Celsius or Fahrenheit. You may select the readings which are appropriate to your application and region.

The **Send Every** option is specific to each sensor measurement. This option will tell the board how often it should read from the sensor and send the data to Adafruit IO. Measurements can range from every 30 seconds to every 24 hours.

For this example, set the **Send Every** interval for both seconds to **Every 30 seconds.** Click **Create Component**.



The board page should now show the MCP9808 component you created. After the interval you configured elapses, the WipperSnapper firmware running on your board automatically reads values from the sensor and sends them to Adafruit IO.



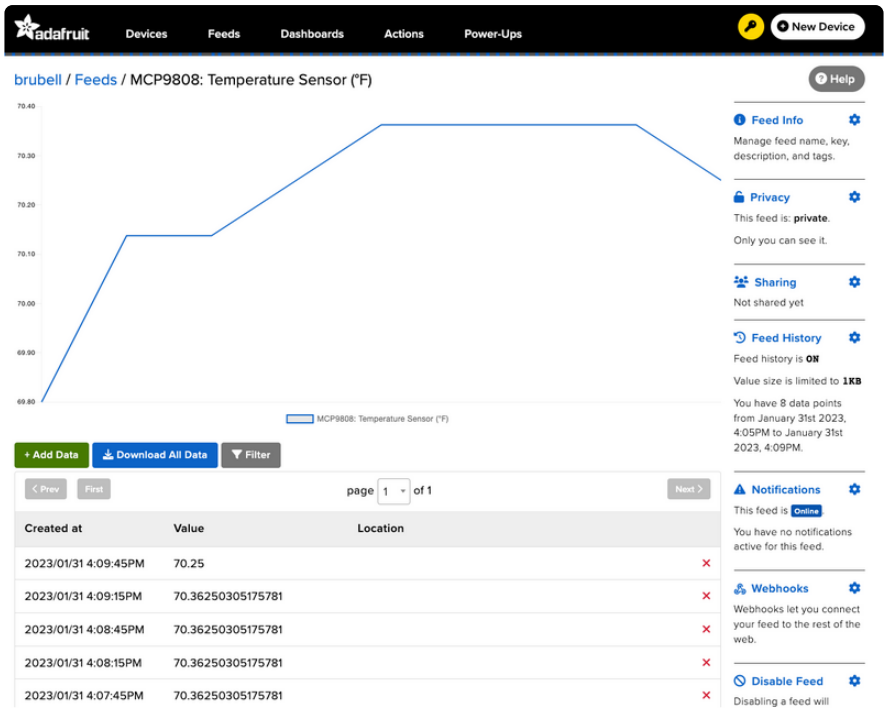
Read I2C Sensor Values

Now to look behind the scenes at a powerful element of using Adafruit IO and WipperSnapper. When a new component is created on Adafruit IO, an Adafruit IO Feed (https://adafru.it/ioA) is also created. This Feed holds your sensor component values for long-term storage (30 days of storage for Adafruit IO Free and 60 days for Adafruit IO Plus plans).

Aside from holding the **values** read by a sensor, the component's feed also holds **metadata** about the data pushed to Adafruit IO. This includes settings for whether the data is public or private, what license the stored sensor data falls under, and a general description of the data.

Next, to look at the sensor temperature feed. To navigate to a component's feed, click on the chart icon in the upper-right-hand corner of the component.

On the component's feed page, you'll each data point read by your sensor and when they were reported to Adafruit IO.



Doing more with your sensor's Adafruit IO Feed

This only scratches the surface of what Adafruit IO Feeds can accomplish for your IoT projects. For a complete overview of Adafruit IO Feeds, including tasks like downloading feed data, sharing a feed, removing erroneous data points from a feed, and more, head over to the "Adafruit IO Basics: Feed" learning guide (https://adafru.it/ioA).

MicroPython Setup

This board is also supported by MicroPython. This page covers the basics to get you started.

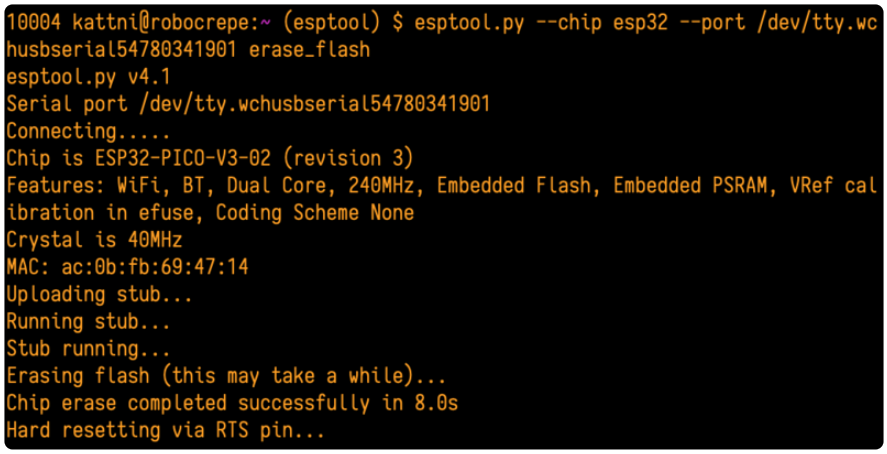
Load MicroPython using `esptool`

Before you can load MicroPython, you'll need to install `esptool`. Follow the instructions found here (https://adafru.it/ZeI).

Once you have `esptool` installed, you will first want to erase the flash on your QT Py. You can do so by running something similar to the following command. Make sure

you update the port to match the serial port to which your board is connected, i.e. change `/dev/tty.wchusbserial54780341901` to match your connection.

```
esptool.py --chip esp32 --port /dev/tty.wchusbserial547803419010 erase_flash
```



Once the flash is successfully erased, you can load MicroPython by running something similar to the command below. Make sure you update the following:

- The port to match the serial port to which your board is connected, i.e. change /**dev/tty.wchusbserial54780341901** to match your connection.
- The **firmware.bin** file name to match the MicroPython firmware you downloaded, i.e. change **firmware.bin** to something like **esp32-20220117-v1.18.bin**.

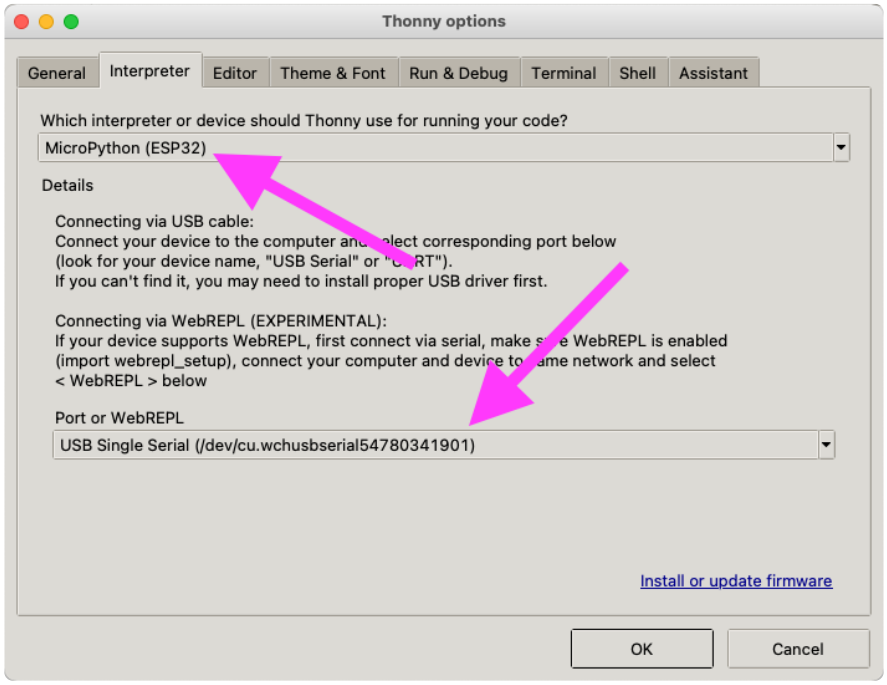
Load MicroPython using something similar to the following command, with the above changes:

```
esptool.py --chip esp32  --port /dev/tty.wchusbserial547803419010 write_flash -z 0x1000 firmware.bin
```

```
10005 kattni@robocrepe:~ (esptool) $ esptool.py --chip esp32 --port /dev/tty.wc
husbserial54780341901 --baud 460800 write_flash -z 0x1000 Downloads/esp32-20220
117-v1.18.bin
esptool.py v4.1
Serial port /dev/tty.wchusbserial54780341901
Connecting....
Chip is ESP32-PICO-V3-02 (revision 3)
Features: WiFi, BT, Dual Core, 240MHz, Embedded Flash, Embedded PSRAM, VRef cal
ibration in efuse, Coding Scheme None
Crystal is 40MHz
MAC: ac:0b:fb:69:47:14
Uploading stub...
Running stub...
Stub running...
Changing baud rate to 460800
Changed.
Configuring flash size...
Flash will be erased from 0x00001000 to 0x0017cfff...
Compressed 1555136 bytes to 1022998...
Wrote 1555136 bytes (1022998 compressed) at 0x00001000 in 23.7 seconds (effecti
ve 524.8 kbit/s)...
Hash of data verified.

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

MicroPython Code

To manage and run code, you can use Thonny - set up to be a **MicroPython (ESP32)** board, then pick the matching serial port.



That's all there is to getting Thonny set up to work with MicroPython!

MicroPython Blink

The first example is blinking the onboard NeoPixel LED.

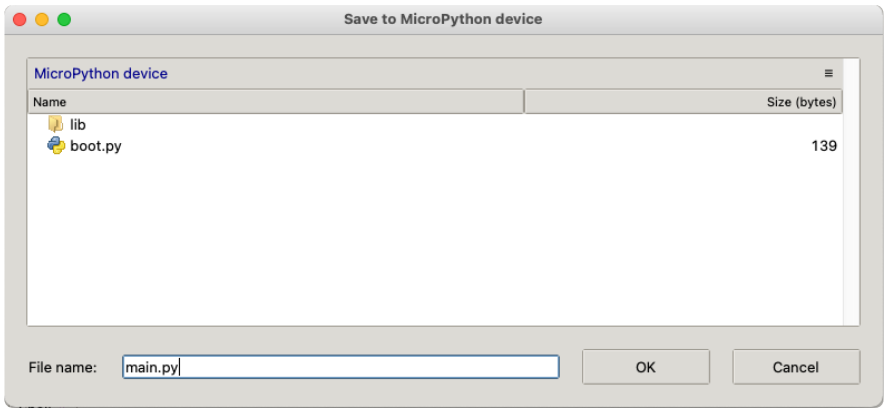
Download the following code and save it to your QT Py as **main.py**

```
# SPDX-FileCopyrightText: 2022 Kattni Rembor for Adafruit Industries
# SPDX-License-Identifier: MIT
import time
from machine import Pin
from neopixel import NeoPixel

power_pin = Pin(8, Pin.OUT)  # NeoPixel power is on pin 8
power_pin.on()  # Enable the NeoPixel Power

pin = Pin(5, Pin.OUT)  # Onboard NeoPixel is on pin 5
np = NeoPixel(pin, 1)  # create NeoPixel driver on pin 5 for 1 pixel

while True:
    np.fill((0, 0, 150))  # Set the NeoPixel blue
    np.write()  # Write data to the NeoPixel
    time.sleep(0.5)  # Pause for 0.5 seconds
    np.fill((0, 0, 0))  # Turn the NeoPixel off
    np.write()  # Write data to the NeoPixel
    time.sleep(0.5)  # Pause for 0.5 seconds
```

Once you have the **main.py** file on your computer, open the **main.py** file into Thonny, and click **Save**.

The first time you do this, you will receive the following popup. Click **MicroPython Device**.

The following window will show up next, allowing you to save the **main.py** file to your QT Py.



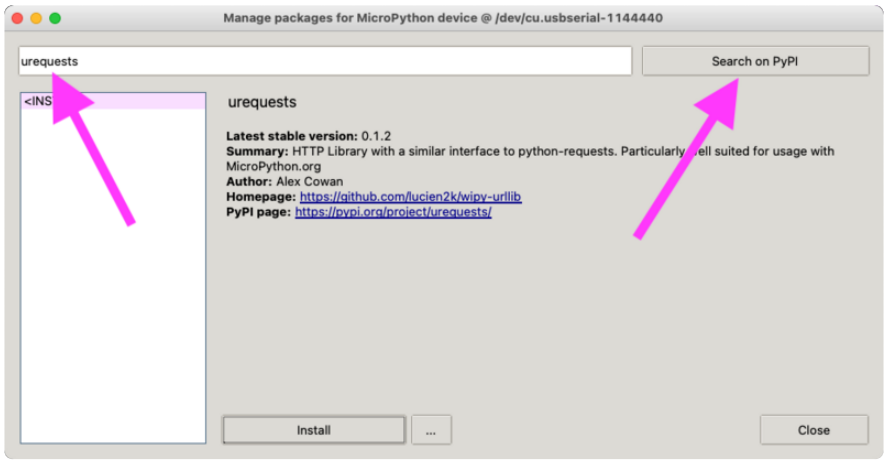
Once saved to your QT Py, in Thonny, click **STOP (red button)** followed by **Run (green arrow button)** to load the **main.py** file and run it.

The NeoPixel LED will begin blinking blue.

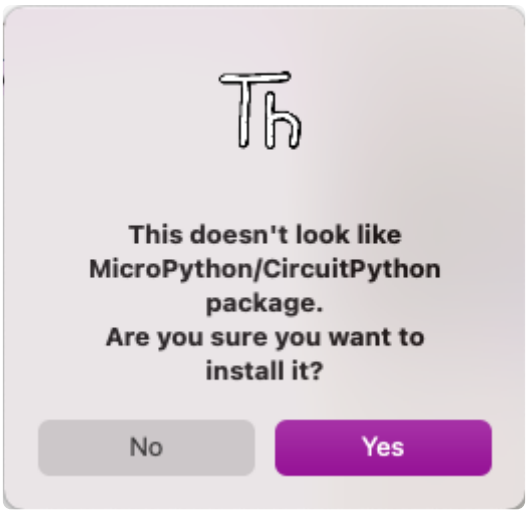
That's all there is to blinking the onboard NeoPixel LED using MicroPython!

MicroPython WiFi Connection

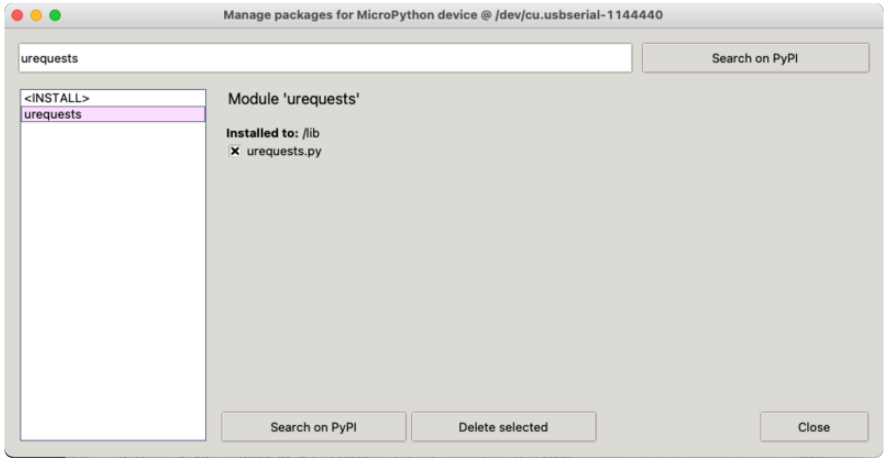
First, install the `urequests` library via the package manager. Click **Tools** > **Manage Packages** to open the package manager. Type **urequests** into the search bar, and click **Search on PyPI**.



You may be greeted with the following popup. Click **Yes**.



Once installed, the package manager will show it as installed.



Click **Close**.

Download the following code and save it to your ESP32 as **main.py**

Don't forget to change `MY_SSID` and `MY_PASSWORD` to your WiFi access point name and credentials.

```
# SPDX-FileCopyrightText: 2022 Ladyada for Adafruit Industries
# SPDX-License-Identifier: MIT
"""MicroPython simple WiFi connection demo"""
import time
import network
import urequests

station = network.WLAN(network.STA_IF)
station.active(True)

# Network settings
wifi_ssid = "MY_SSID"
wifi_password = "MY_PASSWORD"
url = "http://wifitest.adafruit.com/testwifi/index.html"

print("Scanning for WiFi networks, please wait...")
authmodes = ['Open', 'WEP', 'WPA-PSK', 'WPA2-PSK4', 'WPA/WPA2-PSK']
for (ssid, bssid, channel, RSSI, authmode, hidden) in station.scan():
```

```
    print("* {:s}".format(ssid))
    print("   - Channel: {}".format(channel))
    print("   - RSSI: {}".format(RSSI))
    print("   - BSSID: {:02x}:{:02x}:{:02x}:{:02x}:{:02x}:{:02x}".format(*bssid))
    print("   - Auth: {}".format(authmodes[authmode]))
    print()

# Continually try to connect to WiFi access point
while not station.isconnected():
    # Try to connect to WiFi access point
    print("Connecting...")
    station.connect(wifi_ssid, wifi_password)
    time.sleep(10)

# Display connection details
print("Connected!")
print("My IP Address:", station.ifconfig()[0])

# Perform HTTP GET request on a non-SSL web
response = urequests.get(url)

# Display the contents of the page
print(response.text)
```

Follow the same instructions above to load the WiFi code onto your QT Py.

In the Shell, you should see something like the following indicating it could connect to WiFI and read the data from our webserver.

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

#include <Arduino.h>
```

```
#include <Adafruit_NeoPixel.h>
#include "Adafruit_TestBed.h"
#include <WiFi.h>
#include <WiFiAP.h>
#include <esp_wifi.h>

extern Adafruit_TestBed TB;

IPAddress myIP;


void setup() {
  Serial.begin(115200);
  //while (! Serial) delay(10);

  delay(100);
  Serial.println("QT Py ESP32 Pico factory test!");

  Serial.printf("Total heap: %d\n", ESP.getHeapSize());
  Serial.printf("Free heap: %d\n", ESP.getFreeHeap());
  Serial.printf("Total PSRAM: %d\n", ESP.getPsramSize());
  Serial.printf("Free PSRAM: %d\n", ESP.getFreePsram());

  pinMode(NEOPIXEL_POWER, OUTPUT);
  digitalWrite(NEOPIXEL_POWER, HIGH);

  TB.neopixelPin = PIN_NEOPIXEL;
  TB.neopixelNum = 1;
  TB.begin();
  TB.setColor(0x0);
}

uint8_t j = 0;

void loop() {
  if (j == 255) {
    TB.setColor(GREEN);
    Serial.println("scan start");
    // WiFi.scanNetworks will return the number of networks found
    int n = WiFi.scanNetworks();
    Serial.println("scan done");
    if (n == 0) {
      Serial.println("no networks found");
    } else {
      Serial.print(n);
      Serial.println(" networks found");
      for (int i = 0; i < n; ++i) {
          // Print SSID and RSSI for each network found
          Serial.print(i + 1);
          Serial.print(": ");
          Serial.print(WiFi.SSID(i));
          Serial.print(" (");
          Serial.print(WiFi.RSSI(i));
          Serial.print(")");
          Serial.println((WiFi.encryptionType(i) == WIFI_AUTH_OPEN)?" ":"*");
          delay(10);
      }
    }
    Serial.println("");
  }

  TB.setColor(TB.Wheel(j++));
  delay(10);
}
```

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 qt-py-esp32-pico-factory-reset.bin

https://adafru.it/-Az

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.

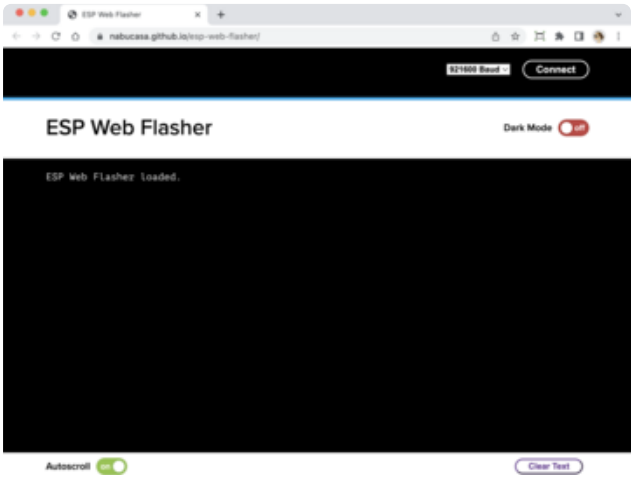
You will have to use a Chromium browser (like Chrome, Opera, Edge...) for this to work, Safari and Firefox, etc. are not supported because we need Web Serial and only Chromium is supporting it to the level needed.

Follow the steps to complete the factory reset.

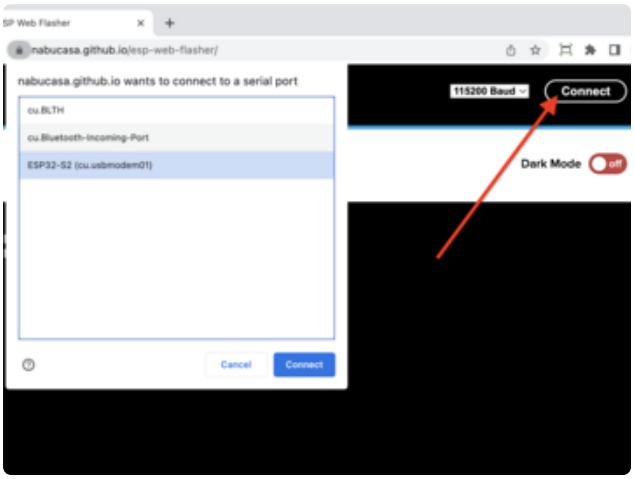
> If you're using Chrome 88 or older, see the Older Versions of Chrome section at the end of this page for instructions on enabling Web Serial.

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.
```

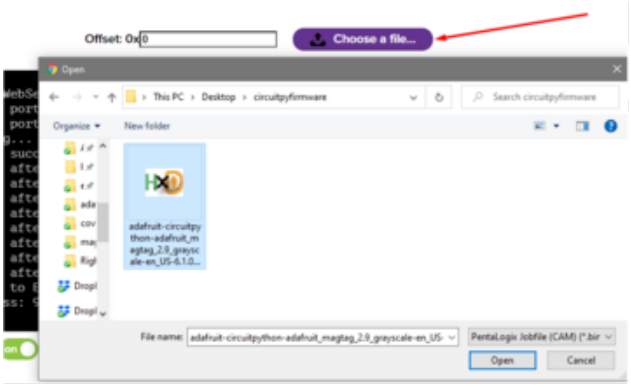
You'll see "Erasing flash memory. Please wait..." This will eventually be followed by "Finished." and the amount of time it took to erase.

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

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

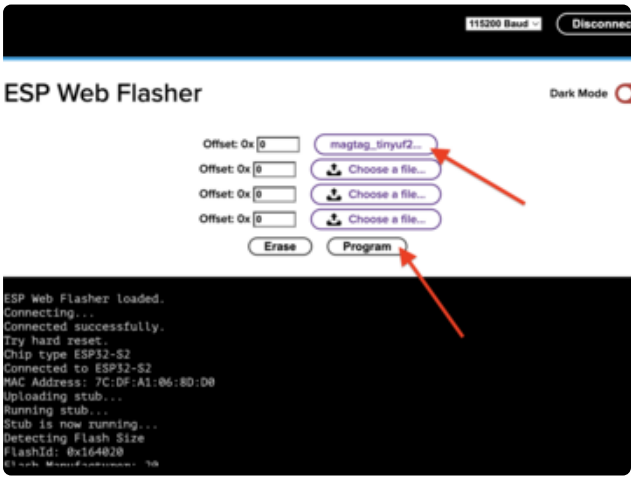
Program the ESP32

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:

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 QT Py ESP32 Pico will light up green, followed by a rainbow swirl.

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

Older Versions of Chrome

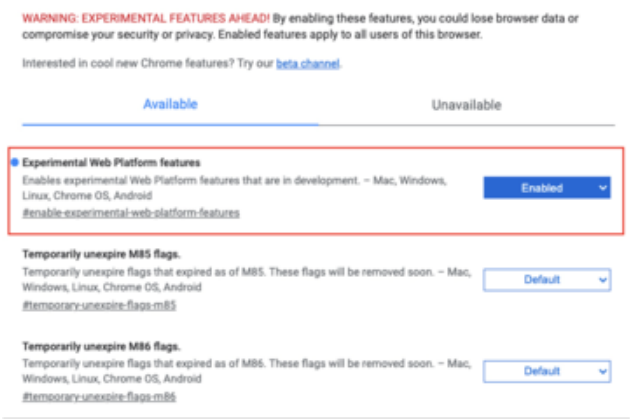
> As of chrome 89, Web Serial is already enabled, so this step is only necessary on older browsers.

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:

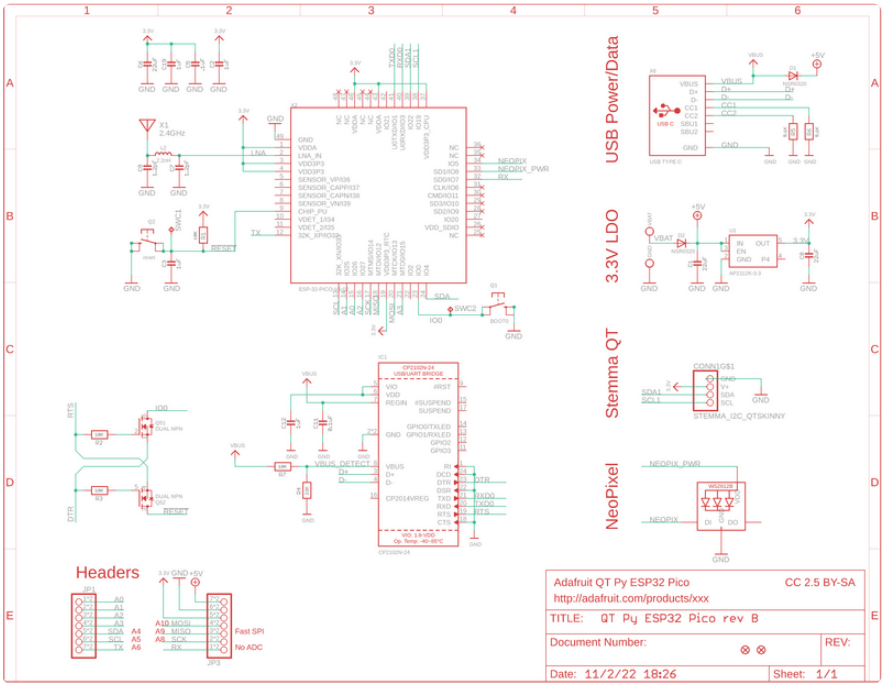
Visit **chrome://flags** from within Chrome.
Find and enable the **Experimental Web Platform features**
Restart Chrome

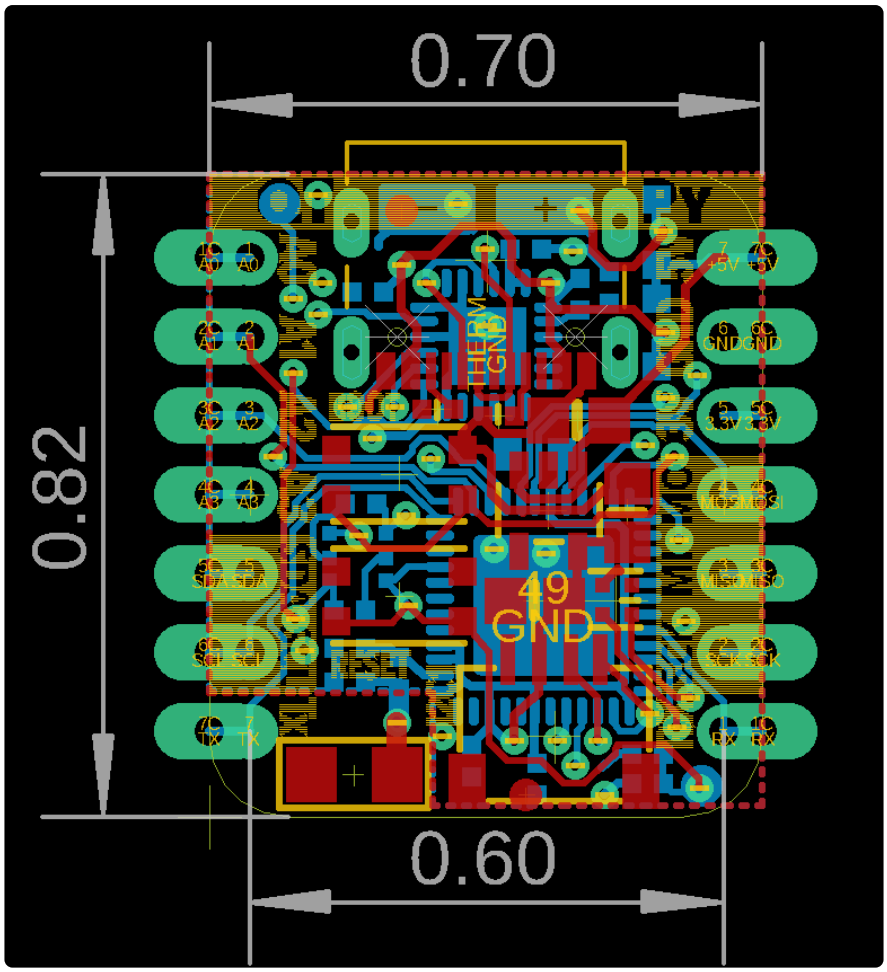
Downloads

Files

- ESP32 Pico datasheet (https://adafru.it/18fm)
- CP2102N datasheet (https://adafru.it/YQB)
- EagleCAD PCB files on GitHub (https://adafru.it/-mb)
- 3D Models on GitHub (https://adafru.it/18fn)
- Fritzing object in the Adafruit Fritzing Library (https://adafru.it/-mc)
- PrettyPins pinout PDF on GitHub (https://adafru.it/-lF)
- PrettyPins pinout SVG (https://adafru.it/-AT)

Schematic and Fab Print





3D Model