WiFi-Controlled NeoPixel Matrix LED Sign

Created by Ruiz Brothers

https://learn.adafruit.com/iot-led-sign

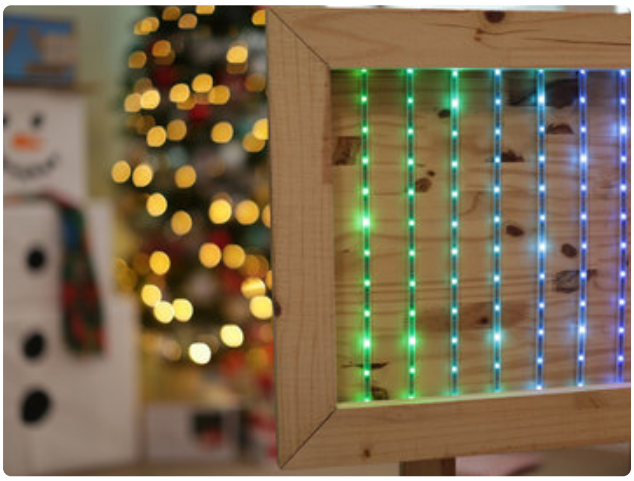
Last updated on 2024-11-29 10:02:26 AM EST

Table of Contents

Overview

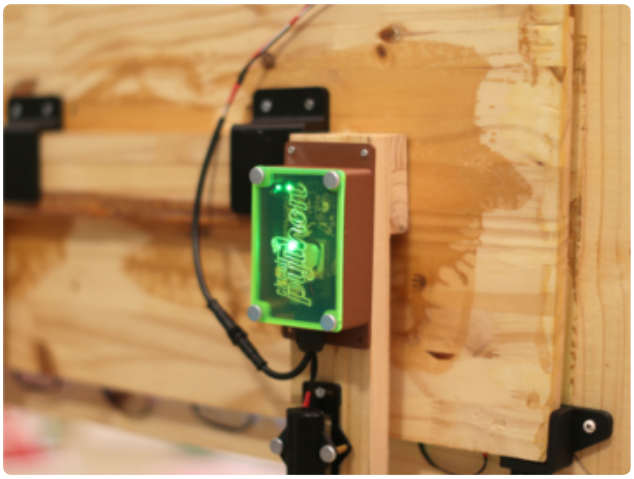
IoT Scrolling Text

In this guide, we'll build an Internet of Things (IoT) sign with NeoPixels and CircuitPython. This is a wooden sign with LED strips that can display scrolling text from Adafruit IO. You can create a dashboard and update the text with your mobile device. With the color picker, you can change the color of the text so you can easily customize your message.



NeoPixel LED Sign with CircuitPython

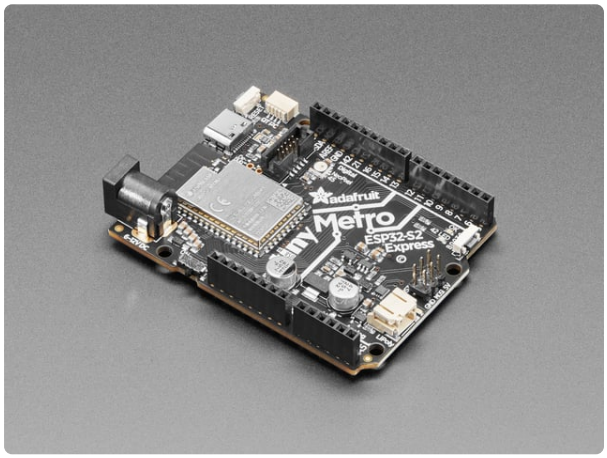
The pixelframe buf library for CircuitPython makes it easy to create custom LED matrices using NeoPixels.



Wifi with Metro ESP32-S2

The Metro ESP32-S2 has built-in WiFi and CircuitPython Support. It’s housed in a 3D printed enclosure and secured to the wooden frame.

Parts



Adafruit Metro ESP32-S2

What's Metro shaped and has an ESP32-S2 WiFi module? What has a STEMMA QT connector for I2C devices, and a Lipoly charger circuit? What has your favorite Espressif WiFi...

https://www.adafruit.com/product/4775

Adafruit Mini Skinny NeoPixel Digital RGB LED Strip - 30 LED/m

So thin. So mini. So teeeeeeny-tiny. It's the 'skinny' version of our classic NeoPixel strips!

https://www.adafruit.com/product/2949

Cable Gland PG-9 size - 0.158" to 0.252" Cable Diameter

We have some great waterproof and weather-proof items in the adafruit shop, but once you have a project built, you'll want...

https://www.adafruit.com/product/761

Waterproof Polarized 4-Wire Cable Set

Outdoor enthusiasts rejoice! We now have very useful 4-wire polarized cable sets in a waterproof variety. These cable sets are ideal for projects that must weather the weather: dust,...

https://www.adafruit.com/product/744

1 x Silicone Ribbon Cable — https://www.adafruit.com/product/3890

Silicone Cover Stranded-Core Ribbon Cable - 10 Wire 1 Meter Long - 28AWG Black

---

1 x USB Micro-B PCB — https://www.adafruit.com/product/1833

USB Micro-B Breakout Board

---

1 x Heat-Set Insert For Soldering Irons — https://www.adafruit.com/product/4239

M3 sized

---

1 x M3 Heat Set Inserts — https://www.adafruit.com/product/4255

M3 x 4mm heat set inserts

---

1 x USB C breakouts — https://www.amazon.com/gp/product/B07T97LC9L

USB 3.1 Type C male Connector

Hardware Screws

Use the following list of hardware to secure the Metro ESP32-S2 to the 3D printed enclosure.

- 8x M3 x 6mm long screws

- 4x M3 x 10mm long FF standoffs
- 4x M3 x 4mm heat set inserts

CAD Files

Source Files

The cad files are available to download using the links below.

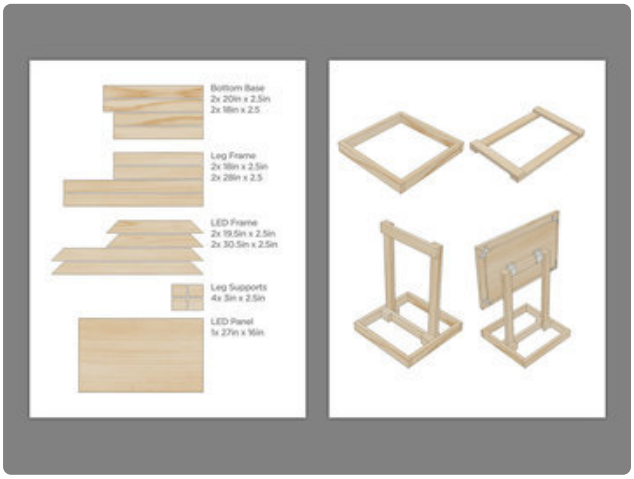
Fusion 360, STEP files
Adjustable User Parameters

Download CAD source files

https://adafru.it/PeO

Download STL files

https://adafru.it/PeP



Plans

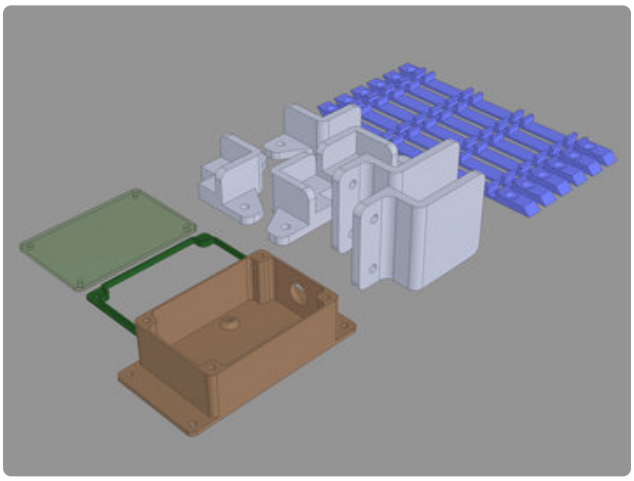
The wood working portion of the project feature a set of plans for printing out. These are handy to reference while cutting and assembling the parts.

Download Parts PDF

https://adafru.it/PeQ

Download Steps PDF

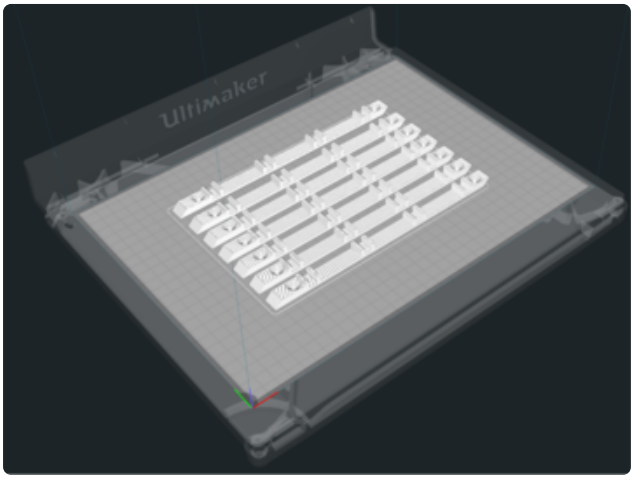
https://adafru.it/PeR



3D Parts List

STL files for 3D printing are oriented to print "as-is" on FDM style machines. Parts are designed to 3D print without any support material. Original design source may be downloaded using the links below.

1x case-box.stl
1x case-gasket.stl
1x case-cover.stl
2x panel-bracket.stl
4x frame-bracket.stl
6x neopixel-holder.stl



Slicing Parts

Slice with setting for PLA material. The parts were sliced using CURA using the slice settings below.

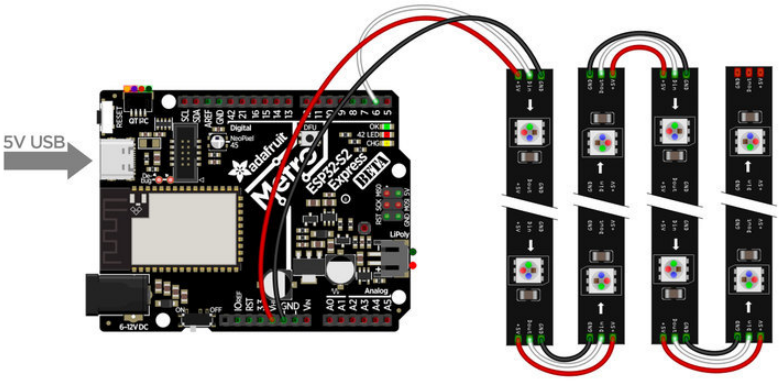
PLA filament
215c extruder
0.2 layer height
10% gyroid infill
60mm/s print speed
60c heated bed

Circuit Diagram

The diagram below provides a visual reference for wiring of the components. This diagram was created using the software package Fritzing (https://adafru.it/oEP).

Adafruit Library for Fritzing

Use Adafruit's Fritzing parts library to create circuit diagrams for your projects. Download the library or just grab individual parts. Get the library and parts from GitHub - Adafruit Fritzing Parts (https://adafru.it/AYZ).



Metro ESP32-S2 features a built-in On/Off switch – Ensure it is set to the ON position in order to power up the board.

Wired Connections

The wiring diagram only shows 4 strips for simplicity. This project is designed to use 12x strips with 12x pixels on each strip. Arrange the NeoPixel strips so the flow of data is in a zigzag pattern.

NeoPixels to Metro

- DIN from NeoPixel strip to Pin 6 on Metro
- 5V from NeoPixel strip to VHI on Metro
- GND from NeoPixel strip to GND on Metro

NeoPixel strip to NeoPixels strip

- DOUT from strip to DIN on strip
- 5V from strip to 5V on strip
- GND from strip to GND on strip

Powering

The Adafruit board can be powered via USB or JST using a 3.7v lipo battery. In this project, a 5V power supply is used. If a lipo battery is being used, it can be rechargeable over the USB-C port on the Metro ESP32-S2.

Adafruit IO Setup

Obtain Adafruit IO Key

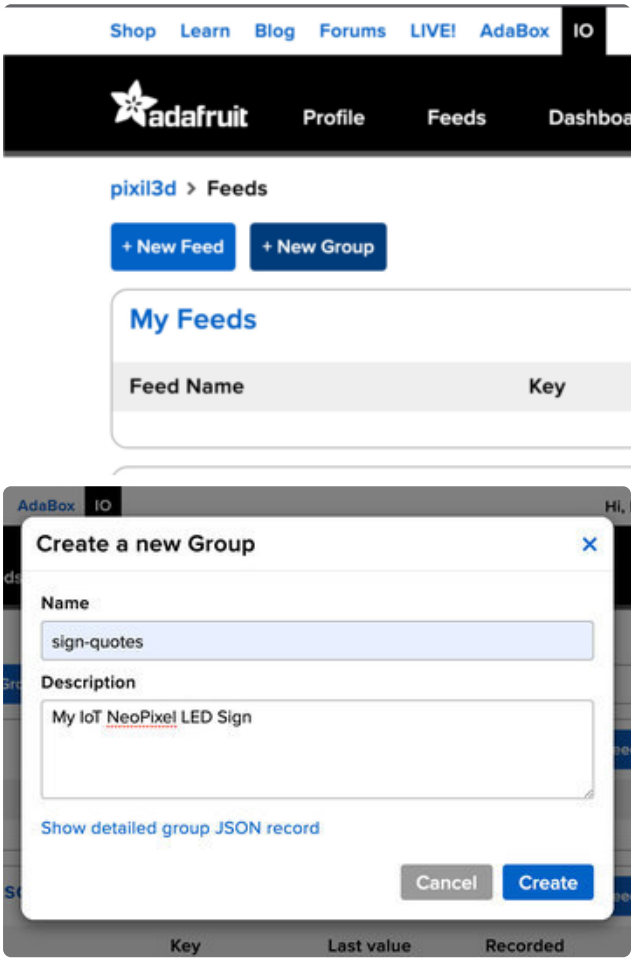
Adafruit IO is Adafruit's free internet of things data portal. If you do not have an account, navigate to https://io.adafruit.com/ (https://adafru.it/fsU) and create one.

You will need your Adafruit IO username and secret API key.

Navigate to Adafruit IO (https://adafru.it/BmD) and **click the Adafruit IO Key button** to retrieve these values. Write them down in a safe place, you'll need them later.

Create Group

This guide will use multiple Adafruit IO feeds to store sensor values. To organize these feeds, you will need to create a new group.



Navigate to your Adafruit IO Feeds page (https://adafru.it/mxC).

Click Actions -> Create a New Group

Name the group sign-quotes. You can optionally set a description.

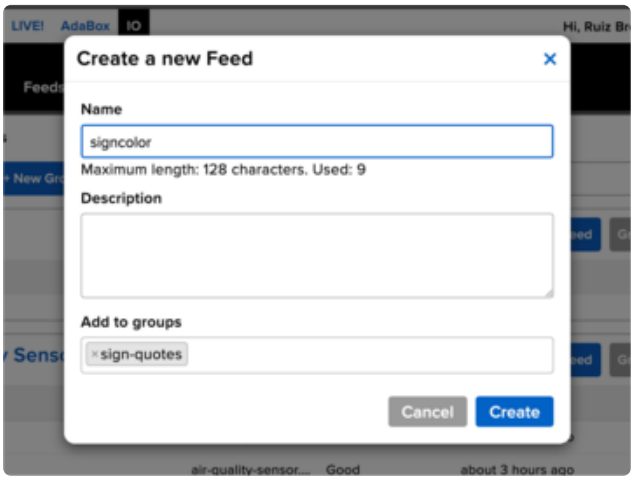
Click Create

Add Feeds to Group

Let's add a few feeds to the **sign-quotes** group to hold NeoPixel color and text.

Click Actions -> Create a New Feed

Name the feed signcolor



Click Add to Groups and select the **sign-quotes** group

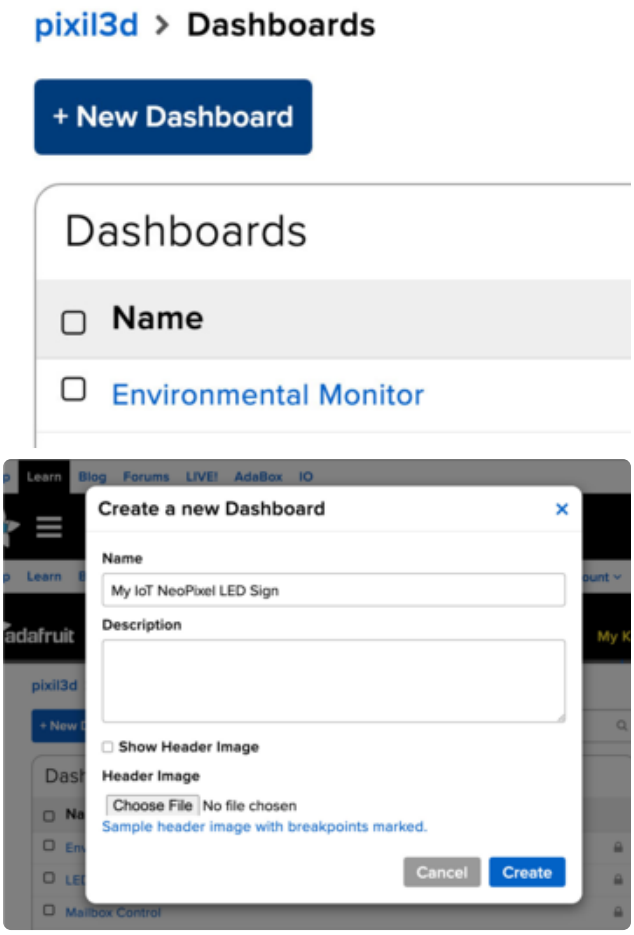
Click Create

Repeat the process in the step above to create a feed for **signtext.**

Before proceeding, make sure your Air Quality Sensor group looks exactly like the screenshot below.

Adafruit IO Dashboard

Dashboards allow you to visualize data and control Adafruit IO connected projects from any modern web browser. We'll be adding a text block to enter text and color picker to choose a color for the NeoPixels.



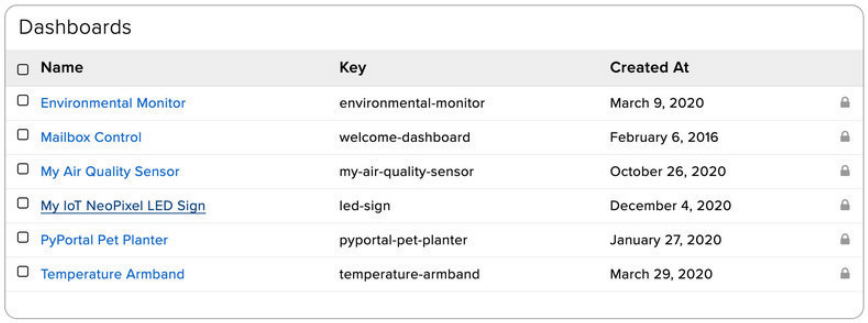
Navigate to the dashboards page on Adafruit IO (https://adafru.it/eIS).

Click Actions -> Create New Dashboard

Name the dashboard My IoT NeoPixel LED Sign

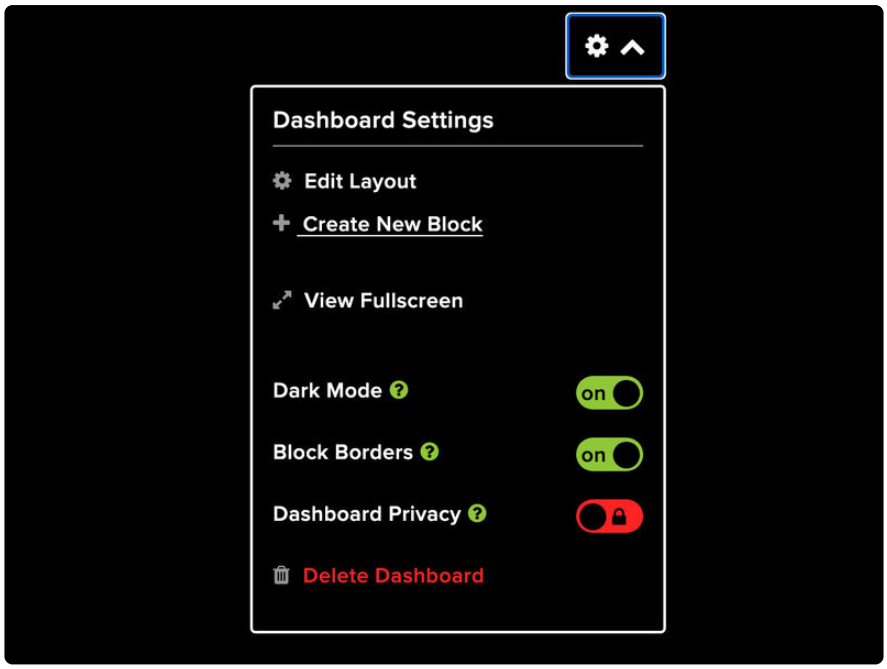
Click Create

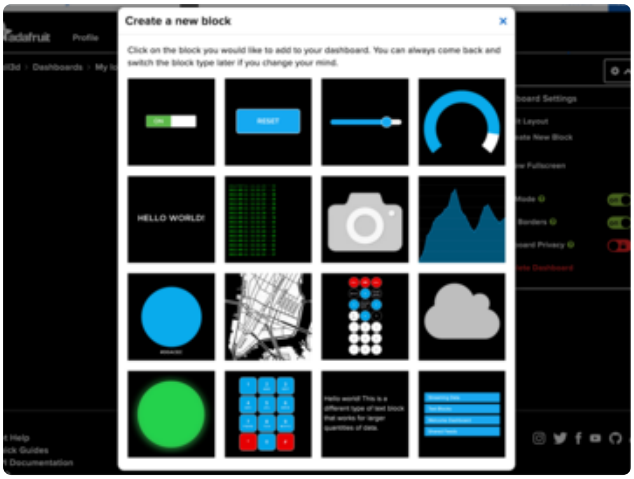
You should see your new dashboard pop-up in the list of Dashboards. **Click the My IoT NeoPixel LED Sign dashboard link** to navigate to the dashboard page.



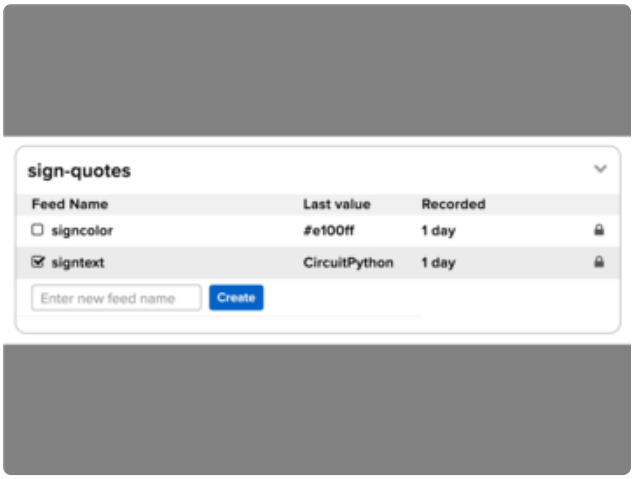
You should see an empty dashboard. Let's fill it with blocks!

Click the '+' button on your dashboard to add a new block.


Dashboard Settings
Edit Layout
Create New Block
View Fullscreen
Dark Mode
on
Block Borders
on
Dashboard Privacy
Delete Dashboard



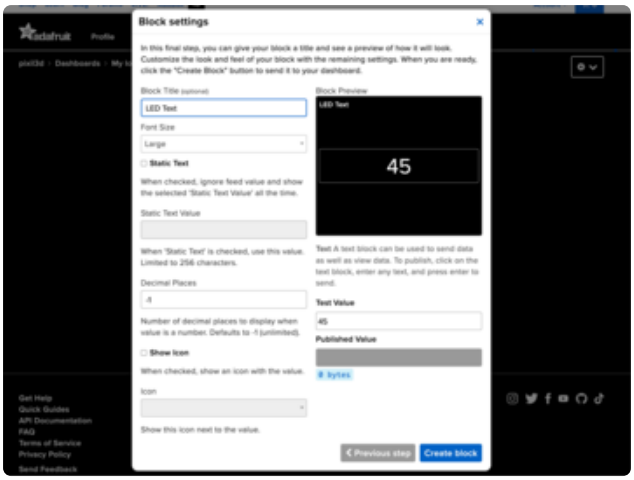
From the Create a New Block picker, **click the Text Block**

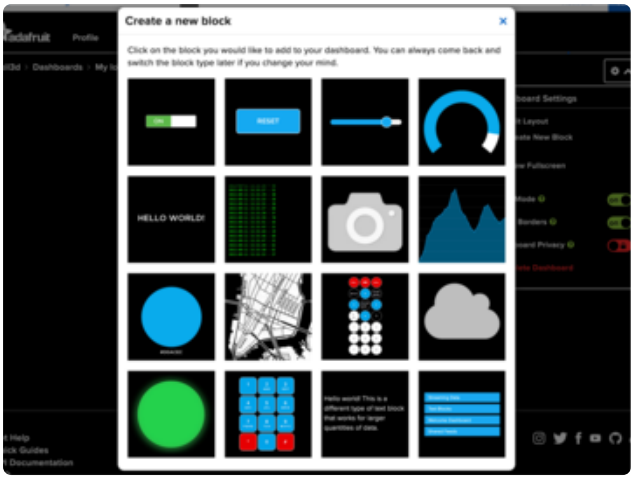


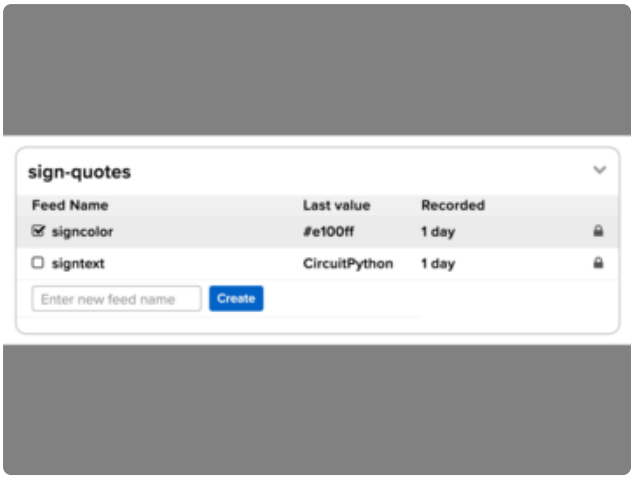
On the Choose Feed picker, **select the signtext feed**

Under Block Settings:

Set Block Title to LED Text
Set Font Size to Large
Click Create Block





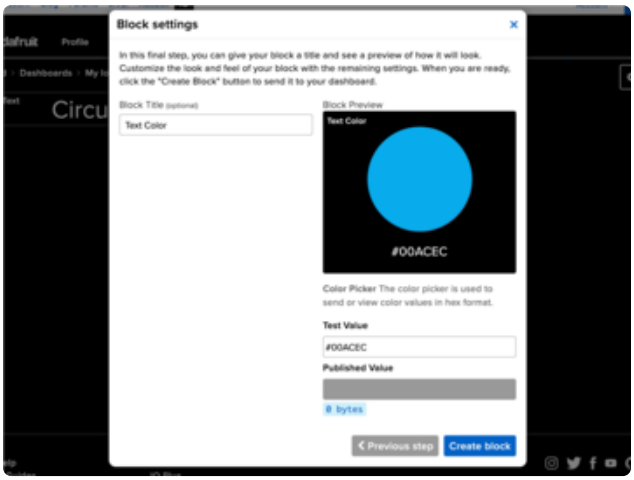


From the Create a New Block picker, click the **Color Picker Block**

On the Choose Feed Picker, select the **signcolor feed**

Name the block **Text Color**

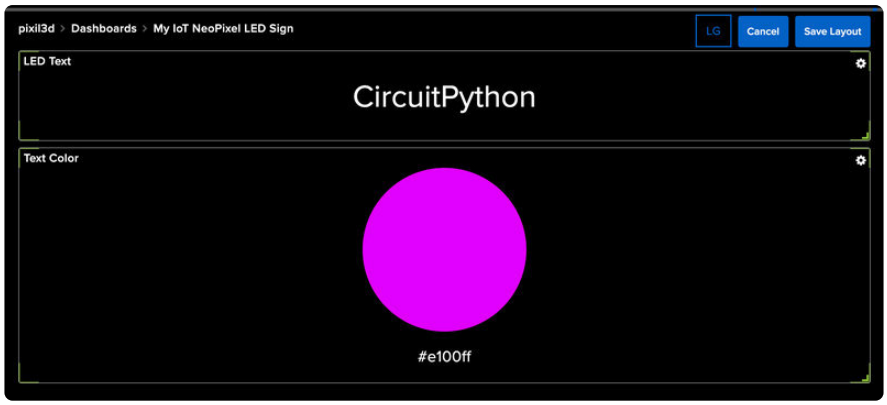
Click Create



Organize Dashboard

You can drag the dashboard blocks around to re-organize your dashboard.

Before moving on, make sure your dashboard contains the same blocks as the screenshot below



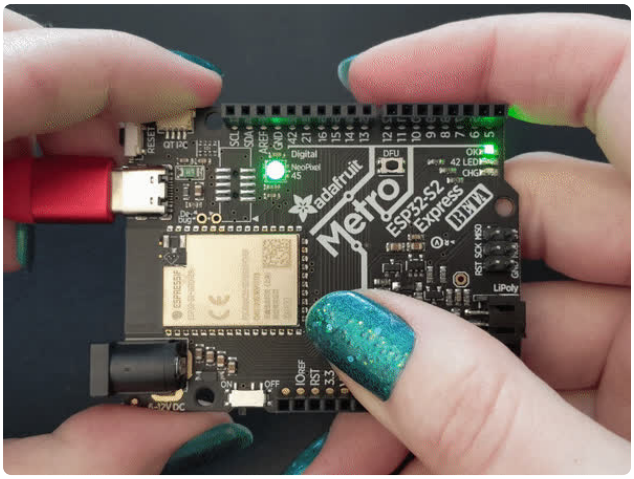
ROM Bootloader

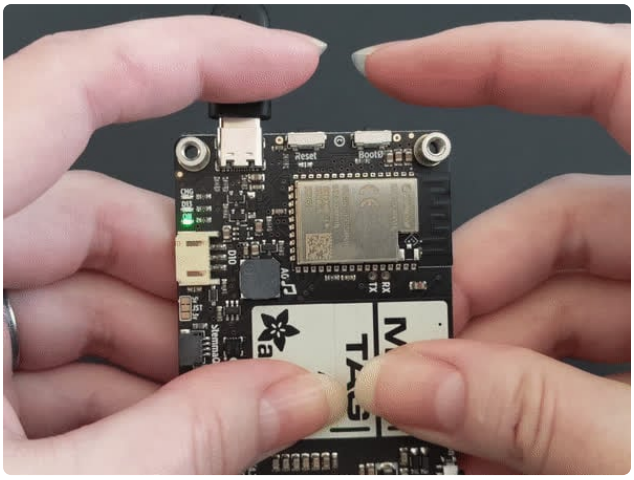
The ESP32-S2 has a built in bootloader, which means you never have to worry about 'bricking' your board. You can use it to load code directly, say CircuitPython or the binary output of an Arduino compilation or you can use it to load a second bootloader on, like UF2 which has a drag-n-drop interface.

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

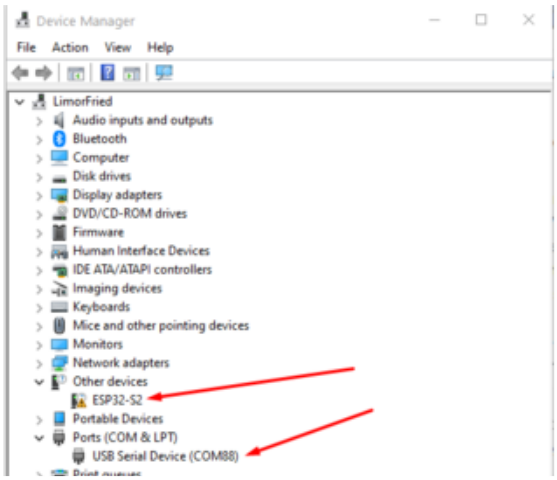
Enter ROM Bootloader Mode

Entering the bootloader is easy. Complete the following steps.



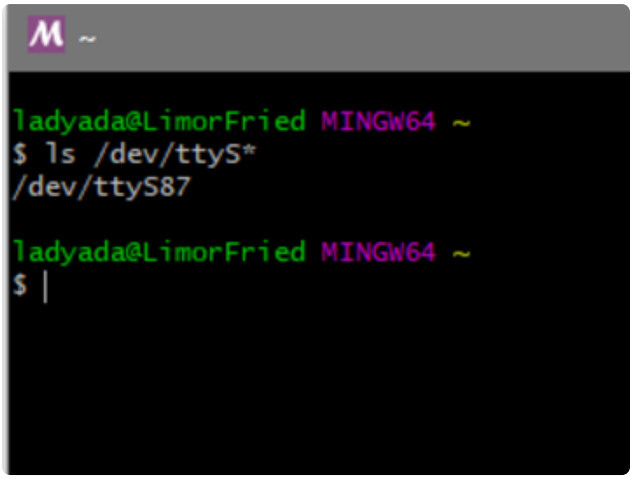


1. **Make sure your ESP32-S2 is plugged into USB port to your computer using a data/sync cable.** Charge-only cables will not work!
2. **Turn on the On/Off switch** - check that you see the OK light on so you know the board is powered, a prerequisite!
3. **Press and hold the DFU / Boot0 button down.**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



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

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



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

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

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

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

Run esptool and check connection

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

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

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

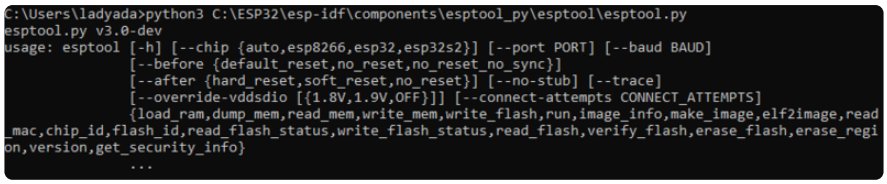
Install the latest version using pip (you may be able to run `pip` without the `3` depending on your setup):

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

Then, you can run:

```
esptool.py
```

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



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

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

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

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

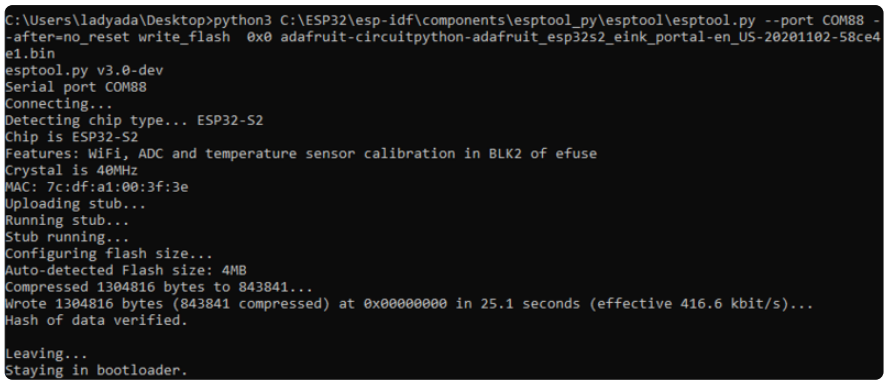
You should get a notice that it connected over that port and found an ESP32-S2

You can now upload a binary file with the following command

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

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

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



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

Install UF2 Bootloader

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

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

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

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

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

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

Step 2. Download the TinyUF2 release for your board

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

Latest MagTag TinyUF2 release

https://adafru.it/TSf

Latest Metro ESP32-S2 TinyUF2 release

https://adafru.it/TSf

Latest Adafruit FunHouse TinyUF2 release

https://adafru.it/TSf

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

Latest TinyUF2 release for all boards

https://adafru.it/TSA

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

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

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

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

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

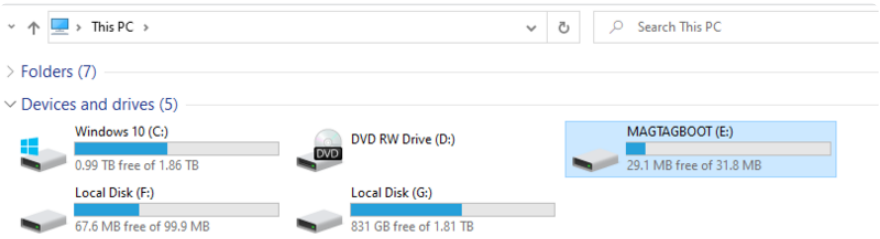
```
esptool.py -p COM88 write_flash 0x0 combined.bin
```

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

You'll finally get an output like this:

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

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

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



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

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

Install CircuitPython

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

Set Up CircuitPython

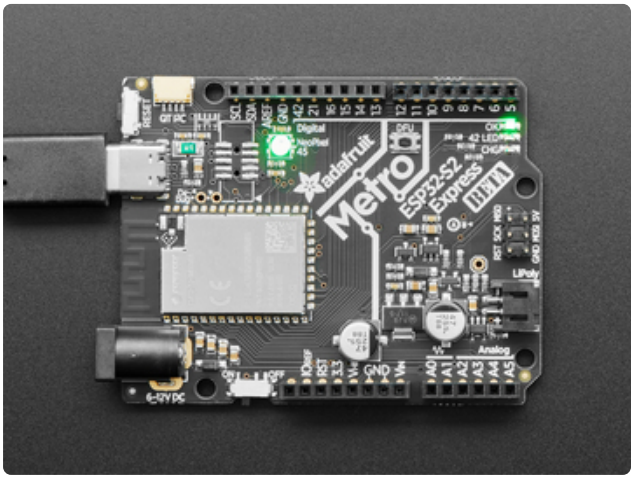
Follow the steps to get CircuitPython installed on your Metro ESP32-S2.

Download the latest CircuitPython for your board from circuitpython.org

https://adafru.it/OsB

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

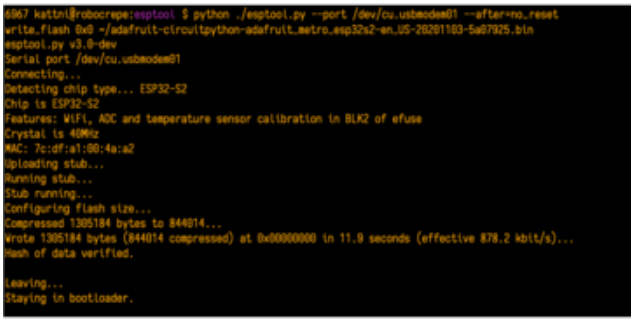
Download and save it to your desktop (or wherever is handy).



Plug your Metro ESP32-S2 into your computer using a known-good USB cable.

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

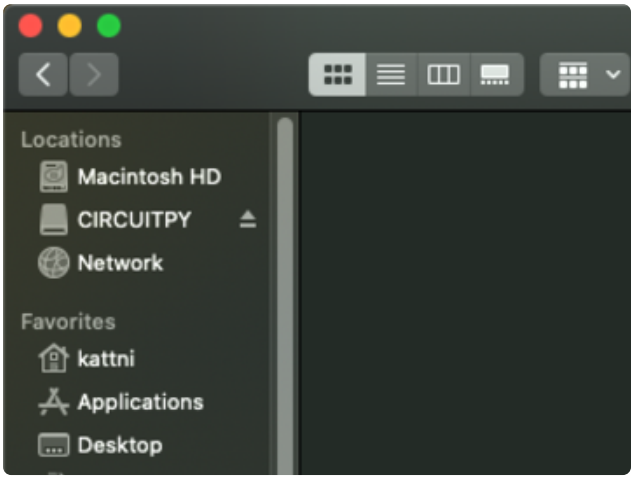
Follow the 6 steps listed in the Enter the ROM Bootloader section of the ROM Bootloader page (https://adafru.it/OsC) to enter the bootloader.



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

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

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



Press reset to exit the bootloader.

Your **CIRCUITPY** drive should appear!

You're all set!

CircuitPython Internet Test

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

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

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

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

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

print("ESP32-S2 WebClient Test")

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

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

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

ping_ip = ipaddress.IPv4Address("8.8.8.8")
```

```
ping = wifi.radio.ping(ip=ping_ip)

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

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

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

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

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

print()

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

print("Done")
```

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

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

The settings.toml File

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

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

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

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

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

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

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

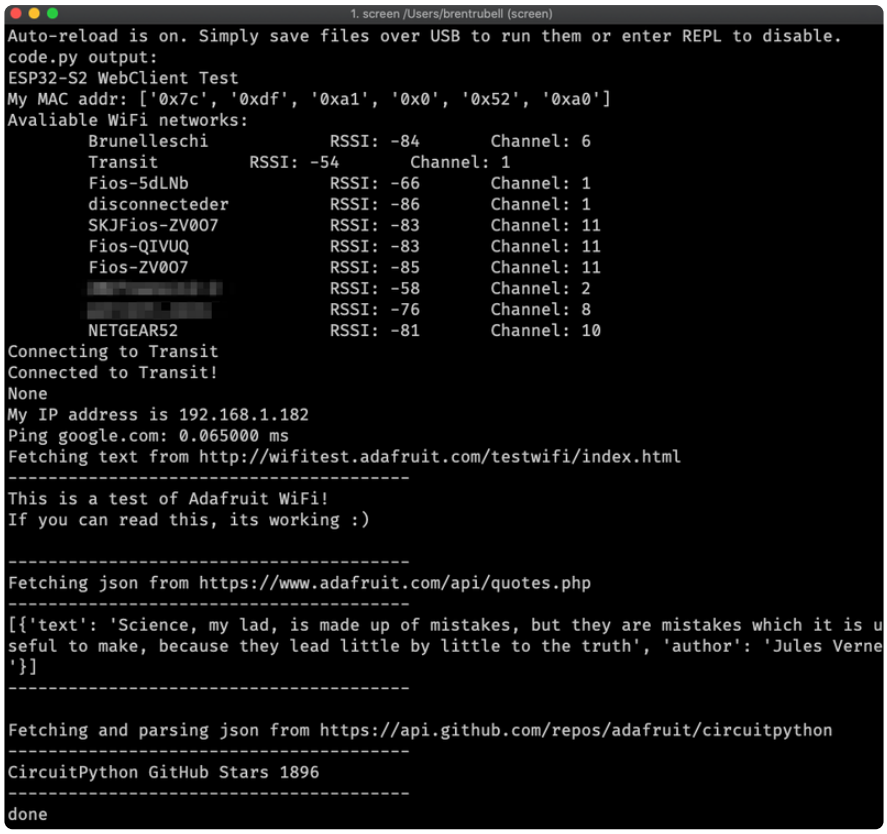
As you make projects you may need more tokens and keys, just add them one line at a time. See for example other tokens such as one for accessing GitHub or the Hackaday API. Other non-secret data like your timezone can also go here.

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

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

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

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



In order, the example code...

Checks the ESP32's MAC address.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

IPv6 Networking

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

IPv6 connectivity & privacy

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

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

Enable IPv6 networking

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

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

Check IP addresses

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

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

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

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

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

Ping v6 networks

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

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

Create & use IPv6 sockets

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

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

Code

CircuitPython Install Check

You should ensure you have CircuitPython 6.1.0 or greater on your board. Plug your board in with a known good data + power cable (not the cheesy USB cable that comes with USB power packs, they are power only). You should see a new flash drive pop up.

If the drive loads, then open the **boot_out.txt** file to ensure the version number is 6.1.0 or greater.

```
Adafruit CircuitPython 6.1.0-beta.1 on 2020-11-20; Metro ESP32S2 with ESP32S2
```

Add Font

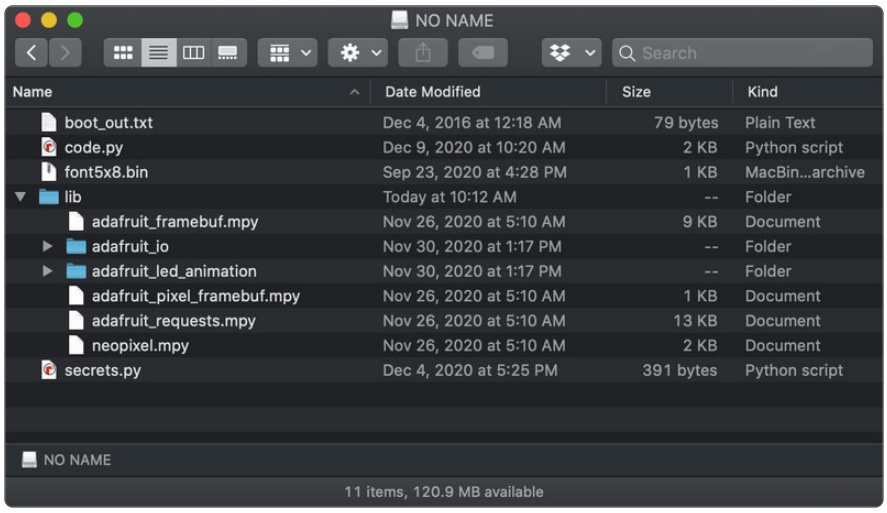
Download the font file below and add it to the root of the CircuitPython drive.

Download Font File

https://adafru.it/DvA

Required Libraries

- **adafruit_framebuf.mpy**
- **adafruit_io**
- **adafruit_led_animation**
- **adafruit_pixel_framebuf.mpy**
- **adafruit_requests.mpy**
- **neopixel.mpy**



Once we have all the files we need, a directory listing will look similar to above as far as files and directories.

Upload Code

Click on the Download Project Zip link below to grab the main code directly from GitHub. **Rename the file** to **code.py** and drop it onto the main (root) directory of the **CIRCUITPY** drive that appeared in your computer file explorer/finder when the board was plugged in via a known, good USB cable The code will run properly when all of the files have been uploaded including libraries.

Use any text editor or favorite IDE to modify the code. We suggest using Mu (https://adafru.it/ANO).

```
# SPDX-FileCopyrightText: 2020 Noe Ruiz for Adafruit Industries
#
# SPDX-License-Identifier: MIT
```

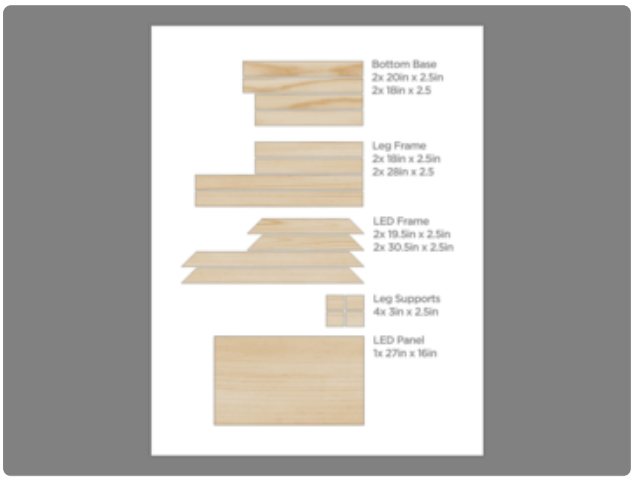
```
import ssl
import board
import neopixel
import adafruit_requests
import socketpool
import wifi
from adafruit_io.adafruit_io import IO_HTTP
from adafruit_pixel_framebuf import PixelFramebuffer
# adafruit_circuitpython_adafruitio usage with native wifi networking

# Neopixel matrix configuration
PIXEL_PIN = board.IO6
PIXEL_WIDTH = 12
PIXEL_HEIGHT = 12

# secrets.py has SSID/password and adafruit.io
try:
    from secrets import secrets
except ImportError:
    print("WiFi secrets are kept in secrets.py, please add them there!")
    raise
AIO_USERNAME = secrets["aio_username"]
AIO_KEY = secrets["aio_key"]

# LED matrix creation
PIXELS = neopixel.NeoPixel(
    PIXEL_PIN, PIXEL_WIDTH * PIXEL_HEIGHT, brightness=0.5, auto_write=False,
)

PIXEL_FRAMEBUF = PixelFramebuffer(
    PIXELS,
    PIXEL_WIDTH,
    PIXEL_HEIGHT,
    alternating=True,
    rotation=1,
    reverse_x=True
    )

# Adafruit.io feeds setup
QUOTE_FEED = "sign-quotes.signtext"
COLOR_FEED = "sign-quotes.signcolor"
CURRENT_TEXT = "Merry Christmas!"
CURRENT_COLOR = 0xFFFFFF

# Helper function to get updated data from Adafruit.io
def update_data():
    global CURRENT_TEXT, CURRENT_COLOR
    print("Updating data from Adafruit IO")
    try:
        quote_feed = IO.get_feed(QUOTE_FEED)
        quotes_data = IO.receive_data(quote_feed["key"])
        CURRENT_TEXT = quotes_data["value"]
        color_feed = IO.get_feed(COLOR_FEED)
        color_data = IO.receive_data(color_feed["key"])
        CURRENT_COLOR = int(color_data["value"][1:], 16)
    # pylint: disable=broad-except
    except Exception as error:
        print(error)


# Connect to WiFi
print("Connecting to %s" % secrets["ssid"])
wifi.radio.connect(secrets["ssid"], secrets["password"])
print("Connected to %s!" % secrets["ssid"])

# Setup Adafruit IO connection
POOL = socketpool.SocketPool(wifi.radio)
REQUESTS = adafruit_requests.Session(POOL, ssl.create_default_context())
```

```
# Initialize an Adafruit IO HTTP API object
IO = IO_HTTP(AIO_USERNAME, AIO_KEY, REQUESTS)


while True:
    update_data()
    print("Displaying", CURRENT_TEXT, "in", hex(CURRENT_COLOR))

    for i in range(12 * len(CURRENT_TEXT) + PIXEL_WIDTH):
        PIXEL_FRAMEBUF.fill(0x000000)
        PIXEL_FRAMEBUF.pixel(0, 0, 0x000000)
        PIXEL_FRAMEBUF.text(CURRENT_TEXT, PIXEL_WIDTH - i, 3, CURRENT_COLOR)
        PIXEL_FRAMEBUF.display()
```

Building the Wooden Sign

Materials

The wood used to build the sign feature the same material width and thickness. The lengths of the each piece are listed below and in the downloadable plans.



Bottom Base

2x 20in x 2.5in x 1in
2x 18in x 2.5in x 1in

Leg Frame

2x 18in x 2.5in x 1in
2x 28in x 2.5 x 1in

LED Frame

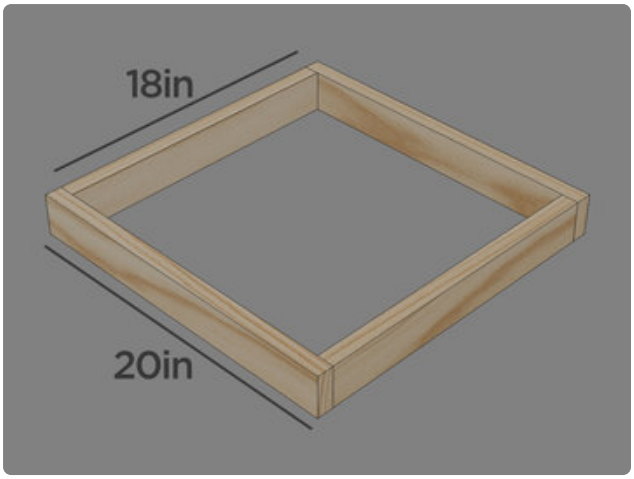
2x 19.5in x 2.5in x 1in
2x 30.5in x 2.5in x 1in

Leg Supports

4x 3in x 2.5in x 1in

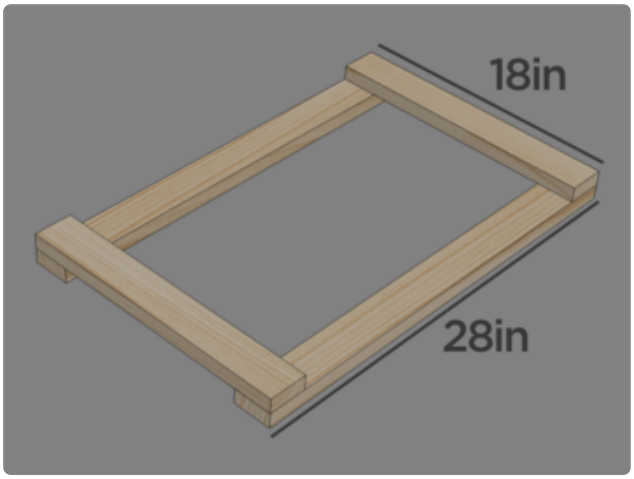
LED Panel

1x 27in x 16in x 1/2in



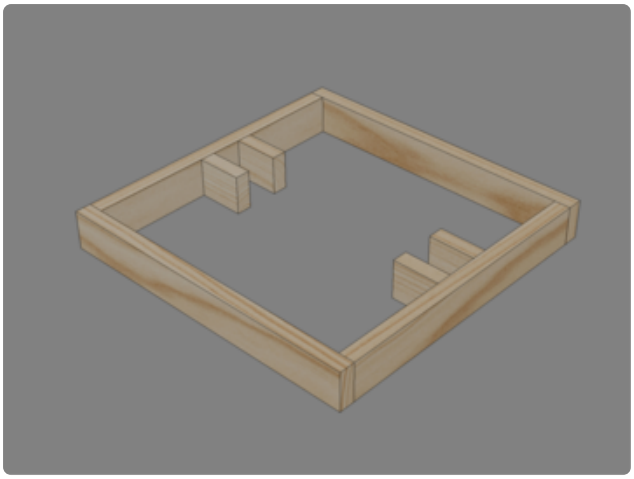
Bottom Base

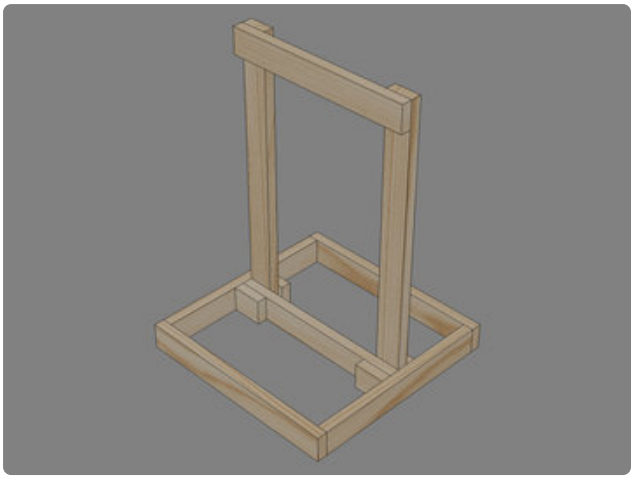
Use four boards to create the bottom base. Secure the boards together using basic butt joints.



Leg Frame

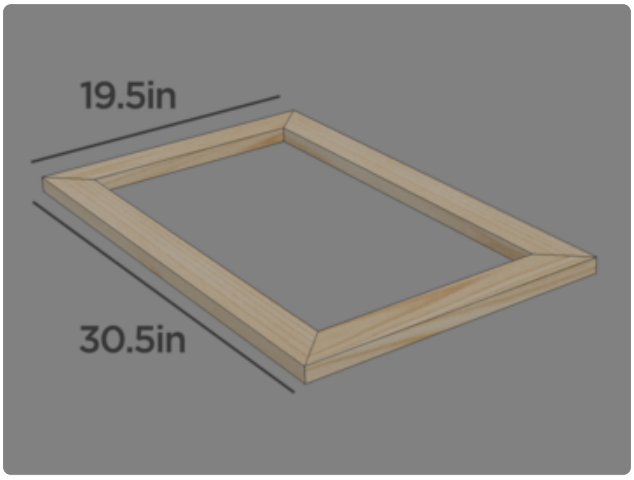
Use four boards to create the leg frame. Place the 28in boards down first and lay the 18in over them. Secure the boards together using nails or screws.





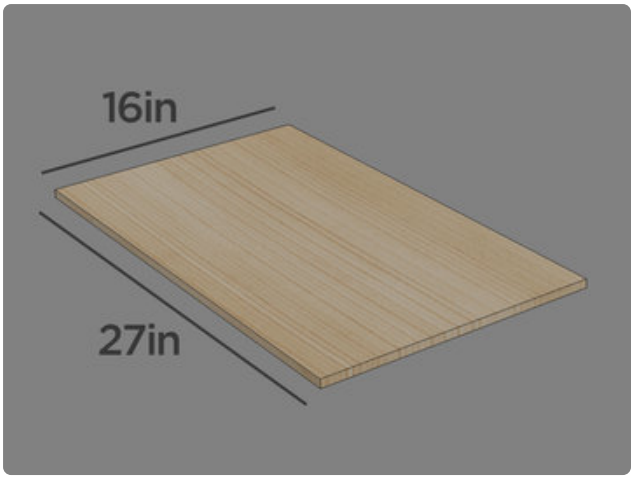
Frame Supports

Use four small cutoff pieces to create the supports. Place the leg frame inside the bottom base to gauge a centered position. Secure the supports to the bottom base using nails or screws.



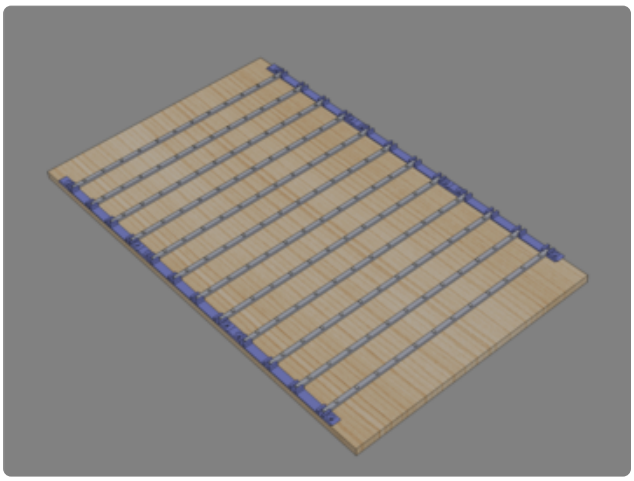
LED Frame

Use four boards with 45 degree mitered cuts to create the LED frame. Secure the boards together using wood glue and nails.



LED Panel

Use a single sheet of wood 1/5in thick and cut to size.

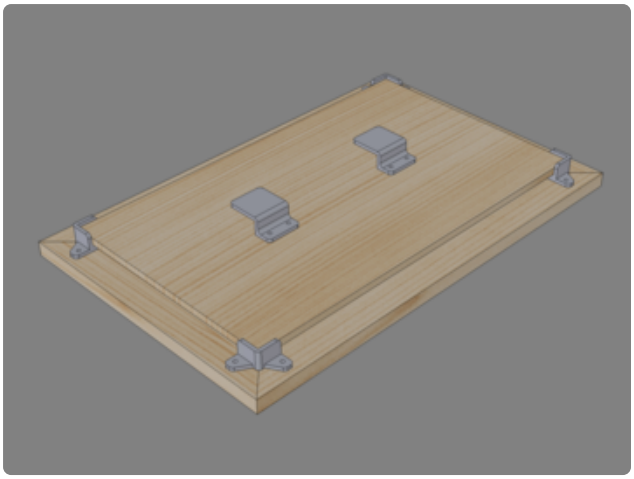


Install Strip Holders

Lay a row of strip holders over the panel and align them with the edge of the wood. Position the set of holder so they're centered with the panel. Use screws to secure the holders to the panel.

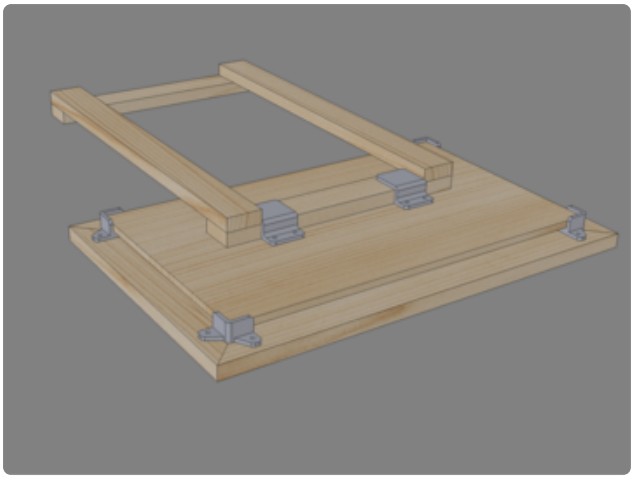
Lay second row of strip holders and align them with the edge. Use a straight edge guide to line up the holders with the first row. Use screws to secure the second set of holders to the panel.

The LED strips do not need to be installed yet, the images are for visual reference.

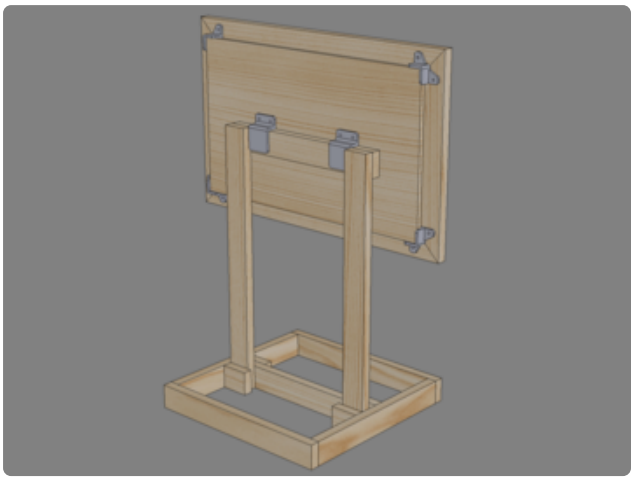


Secure Brackets

Place the four frame brackets onto the corners of the LED frame. Then place the LED panel onto the brackets. Position the two so they're centered. Secure the frame brackets using screws.



Place the leg frame over the LED panel to gauge a good position to secure the panel brackets. Use screws to secure the brackets onto the LED panel.



Assembled Sign

Place the leg frame into the bottom base. Then, fit the LED panel onto the leg frame. Place the LED frame onto the LED panel by fitting onto the brackets. The sign is now ready for NeoPixels and the Metro ESP32-S2 board.

Assembly Notes

A nail gun can be used to help speed up the assembly. Drilling pilot holes is helpful for preventing crooked screws. Material thickness and board width don't have to be exact as the design can be adapted to fit different dimensions.

Building Metro Control Box

Cable for NeoPixels

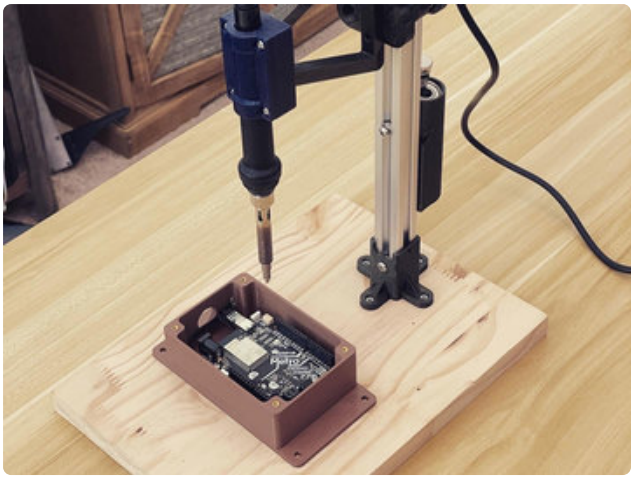
Use the 4-wire weather proof cable to connect the NeoPixel strip to the Metro board. Use a piece of silicone ribbon cable to join the weather proof cable to a 3-pin strip of headers – This makes it easy to connect to the headers on the Metro board. Use heat shrink to insulate exposed connections. The overall length of the cable is 15cm (6in).

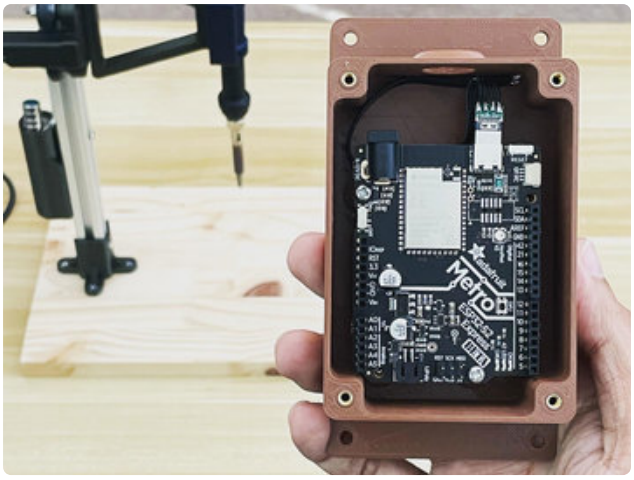
USB-C Cable

Create a slim DIY USB cable to power the Metro board. The Metro is designed to fit a USB C type plug.

The other end can be any type of connector. Ideally the connector should fit an extra long USB cable for power.

The overall length of the cable is 15cm (6in).





Install Heat Inserts

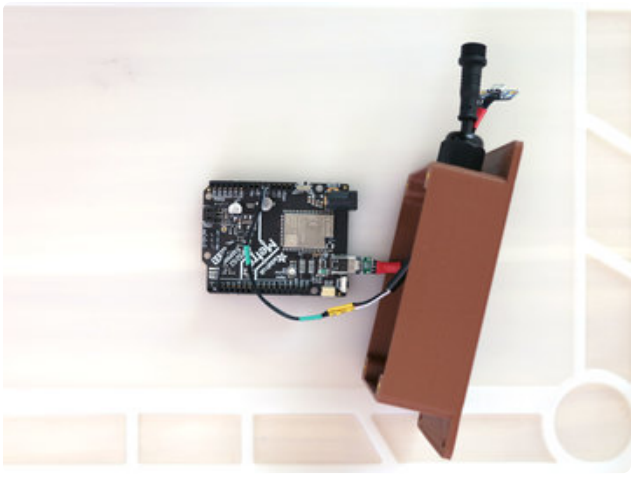
The case uses 4x M3 heat set inserts to secure the top cover. These are installed using a soldering iron with a special heat insert tip. Inserts allow for strong joints with reliable repeatability and no wear out.

A heat insert rig (https://adafru.it/Fis) can be used to make precise installations.

Install Cable Gland & Cables

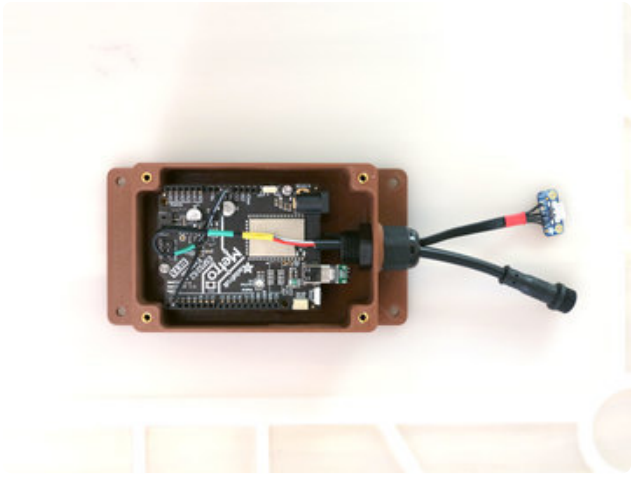
Insert the PG-9 cable gland into the Metro case and secure using the included hex nut.

Fit the USB-C type plug through the cable gland. Then, insert the weather proof cable, header pins first, through the cable gland.



Connect Metro

Fit the header pins from the weather proof cable to Pin 6 (white wire), GND (black wire) and VHI (red wire) pins on the Metro. Connect the USB-C plug to the port on the Metro. Double check the connections from the cable are correct.



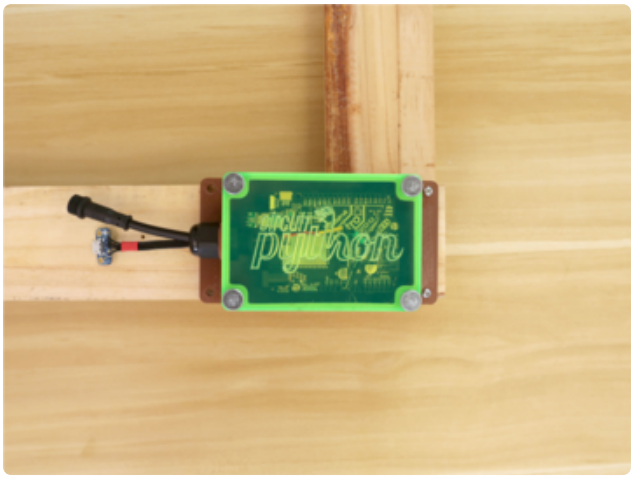
Secure Metro to Case

Orient the Metro board with the mounting holes on the case. Secure the standoffs to the case, then place the metro PCB over them. Secure using machine screws. Use the following hardware to secure the Metro PCB to the case.

8x M3 x 6mm long screws
4x M3 x 10mm long FF standoffs

Secure Cover

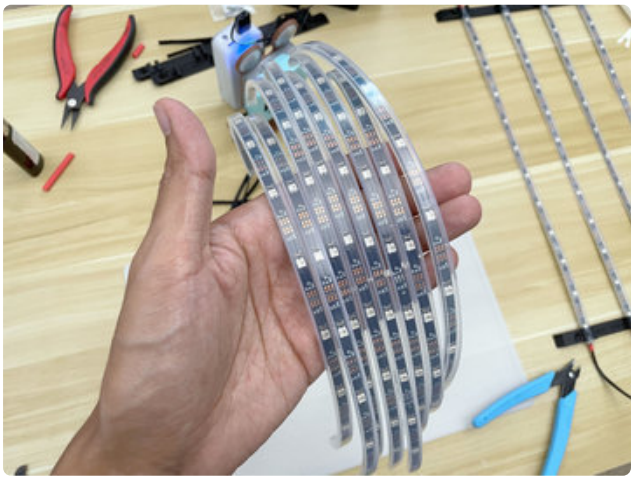
Place the cover on top of the case. Secure using 4x M3 x 6mm long screws. Optionally use the TPU gasket to make a weather proof seal between the cover and case.



Secure Enclosure to Frame

Place the enclosure onto the leg frame in a desired location. The enclosue features four mounting holes for attaching. Secure the enclosure to the frame using #4-40 (M3) wooden screws.

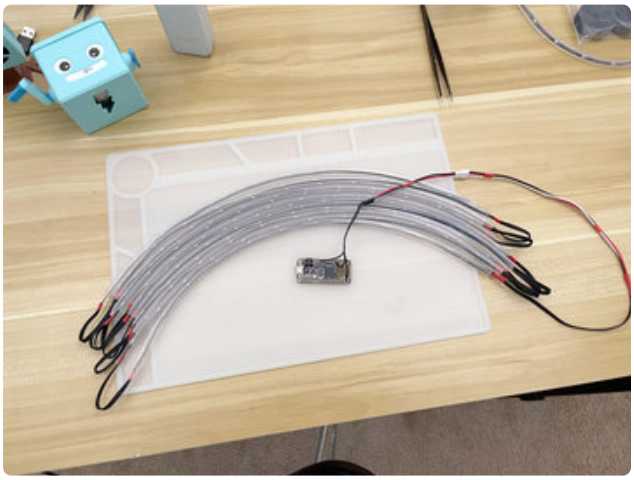
Building NeoPixel Grid



Cut NeoPixel Strips

Use flush snips to cut the NeoPixel strips. Follow the cut markings on the flexible PCB. Thoroughly count each pixel before cutting strips. This project specifically uses a 12x strips, each strip containing 12x NeoPixel LEDs.

Make sure to keep the silicone sheathing fitted over each strip.

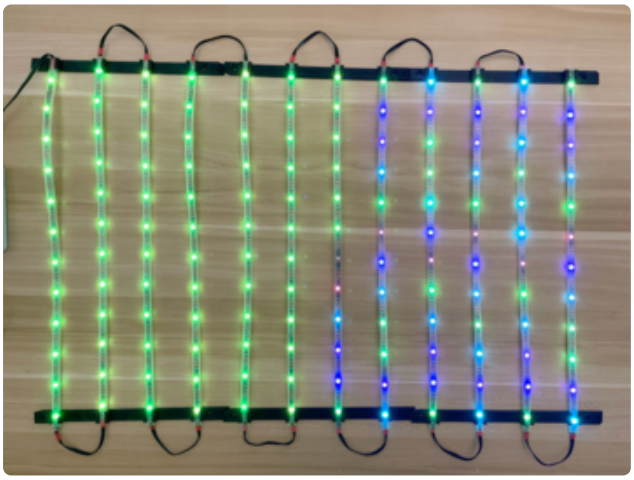


Wiring NeoPixel Strips

Solder the other piece of the weather proof cable to the first NeoPixel strip in the chain.

Solder the 12x strips together using pieces of silicone ribbon wire. Follow the circuit diagram to reference the flow of data. Use pieces of heat shrink to insulate the exposed connections.

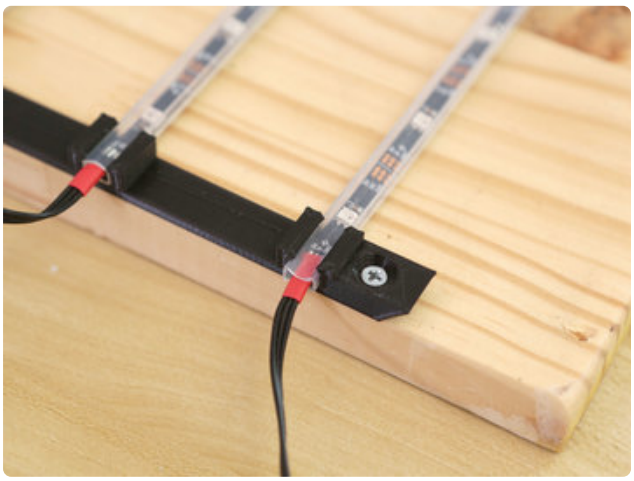
Each set of wires are approximately 5cm (2in) in length.



Test NeoPixel Strips

Connect the first NeoPixel strip to the Metro board. Use a 5V USB power supply to power on the Metro and strips. Thoroughly check each strip for any damaged pixels. If a dead pixel is encountered, swap out the strip with a new one.

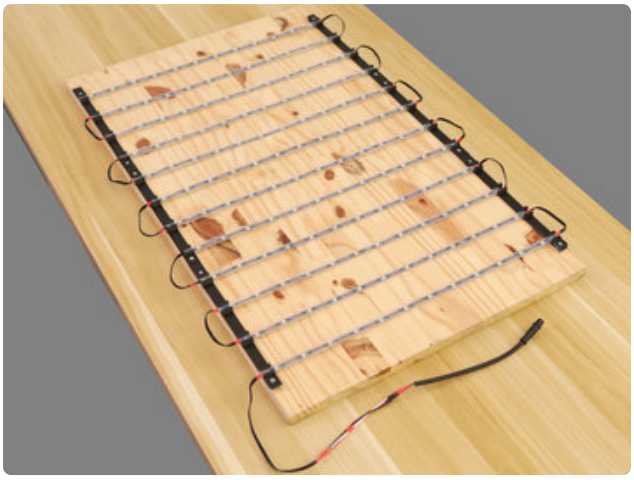
Ideally use the strandtest demo code (https://adafru.it/CYZ) to make all of the NeoPixel a solid color. This a great way to catch any dead pixels in the bunch.



Installing NeoPixel Strips

The NeoPixels are inserted into the 3D printed strip holders. The silicone sheathing is press fitted through the clips. The fitting is snug and should keep the strips nice and taught.

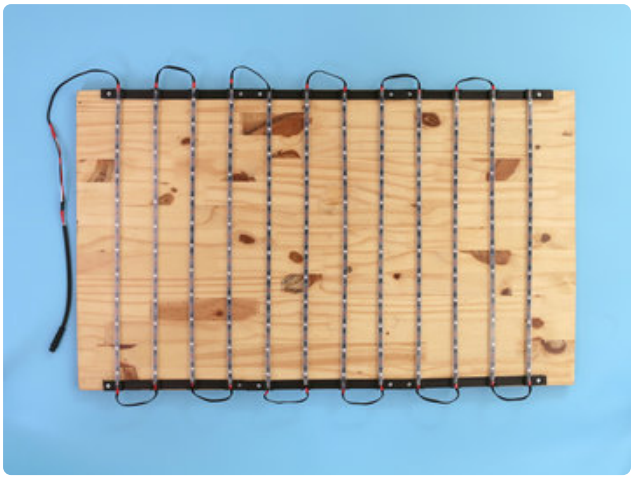
The first NeoPixel strip is secured to the top left corner followed by the proceeding strip in a zigzag order.



Install All Strips

Repeat the installation process for the remaining strips of NeoPixels. Be sure to check the flow of data in each strip so they all follow a zigzag order.

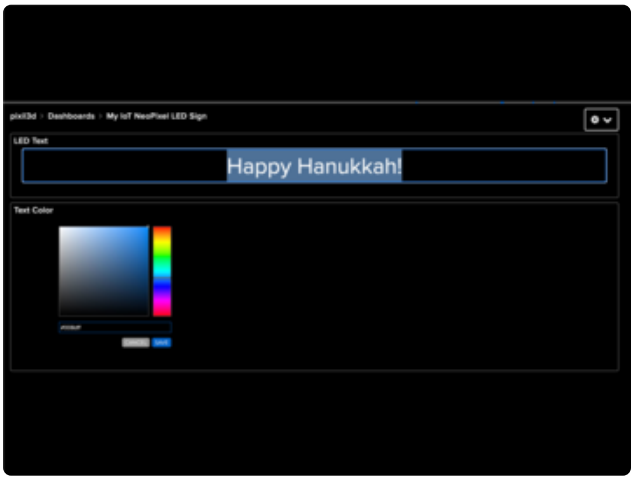
The NeoPixel flexible PCBs can shift during the installation and may need to be adjusted. Align all of the strips for they look straight.



Weather Proofing

If the sign is going to be used outdoors, the NeoPixels will need to be weatherproofed. An easy way to do that is to use hot glue to seal each of the strips. Make sure all of the pixels has been thoroughly tested before adding hot glue.

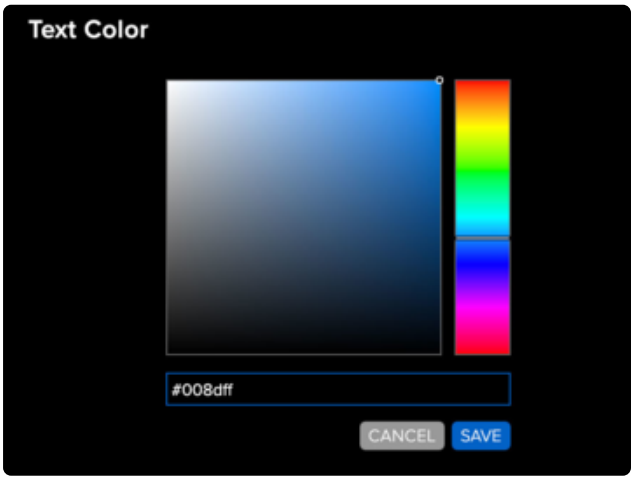
Usage



Updating Scrolling Text

Open up Adafruit IO at https://io.adafruit.com/ (https://adafru.it/fsU) on your phone, tablet, or computer.

Click or tap on the blank text input box and type in your message using a keyboard. Press the enter / return key to save the text. The enter/return key must be entered in order for the text to push to Adafruit IO.



Updating Text Color

Click or tap on the big colored circle to pull up the color picker. Drag the eye dropper around to change the shade of color. Use the slider in the color bar to adjust the hue of the color set. Optionally enter a HEX value in the text input. Click save button to save and update the text color.