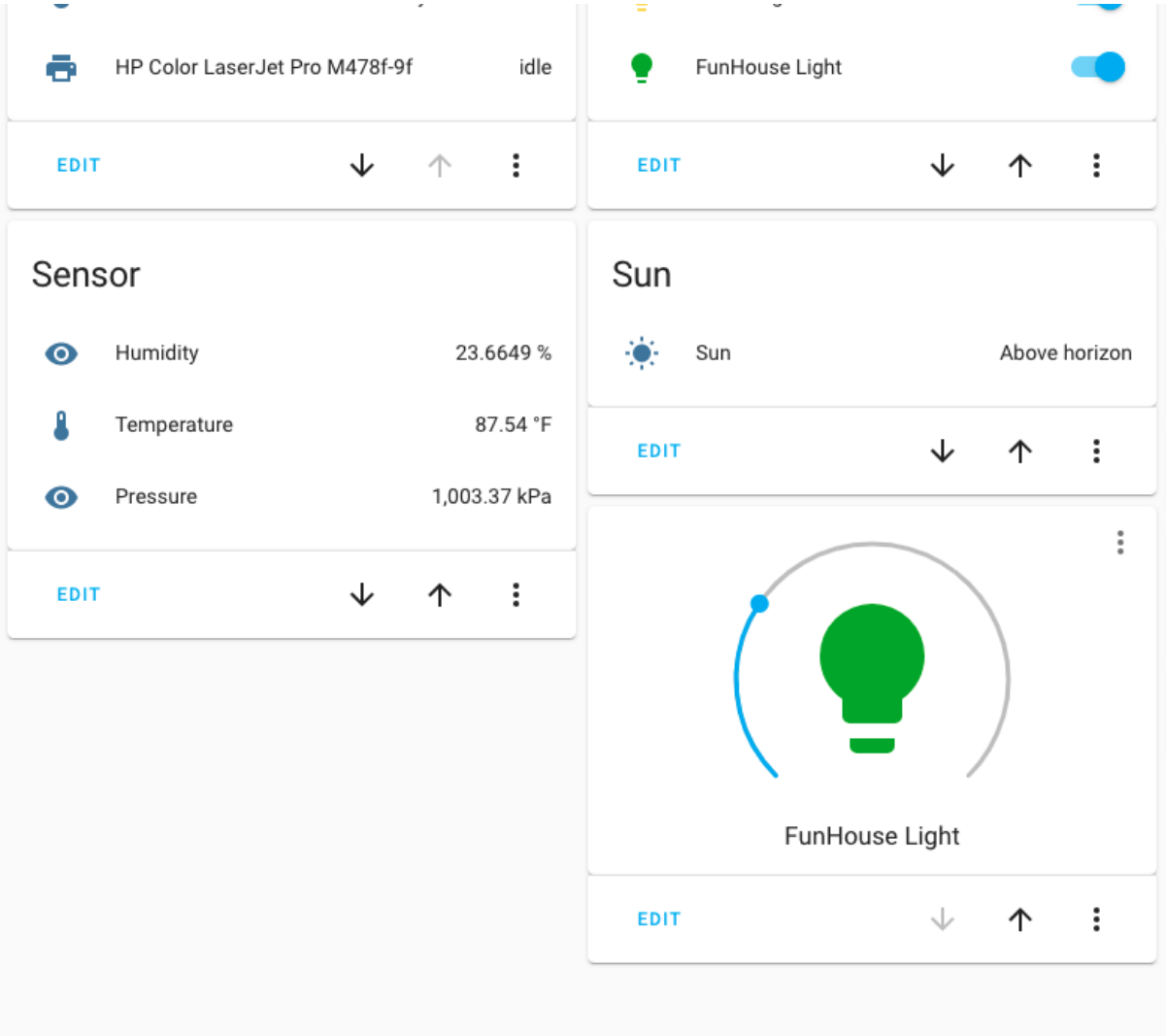
Using the Adafruit FunHouse with Home Assistant

Created by Melissa LeBlanc-Williams

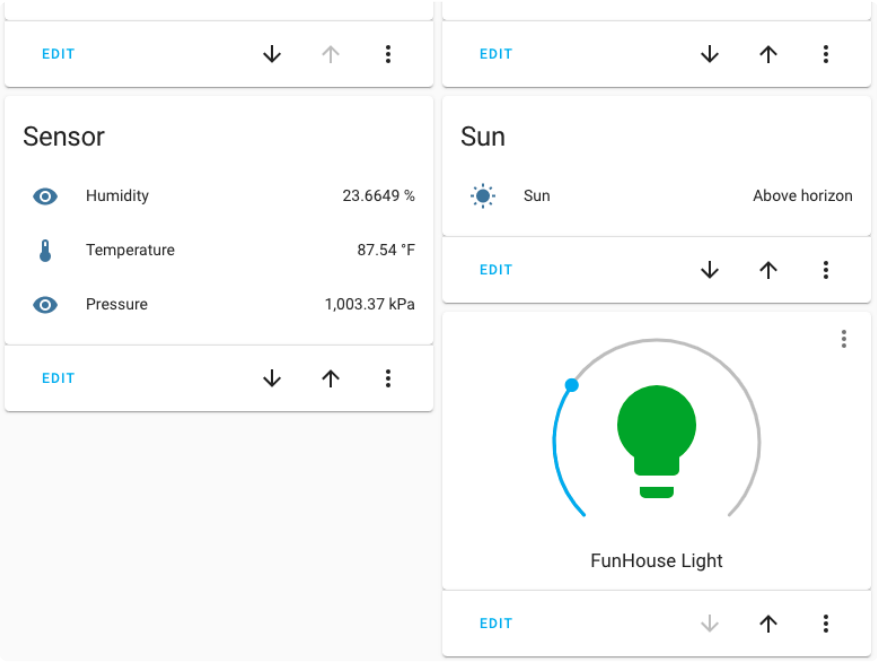


https://learn.adafruit.com/using-the-adafruit-funhouse-with-home-assistant

Last updated on 2025-12-12 06:14:52 PM UTC

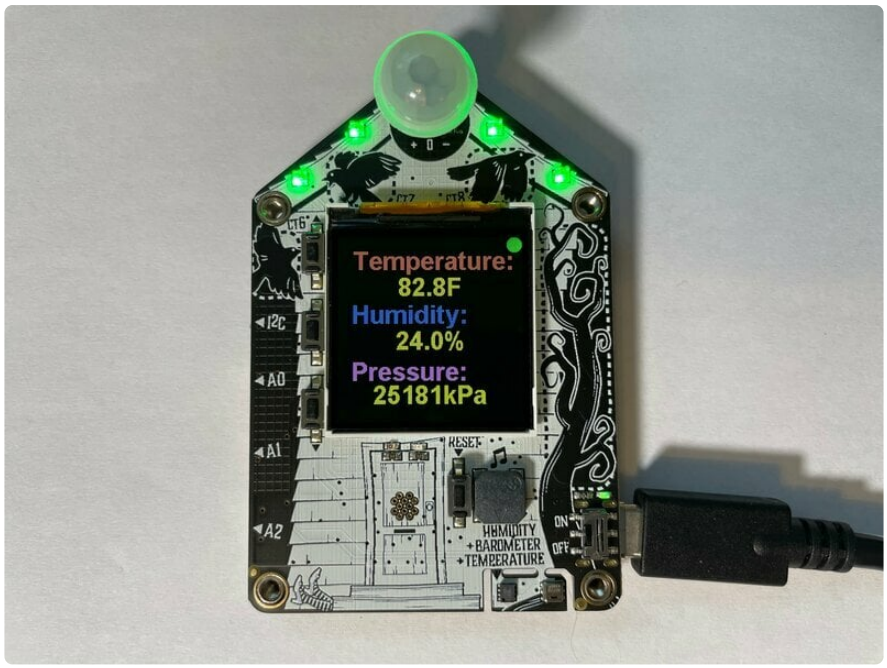
Table of Contents

Overview

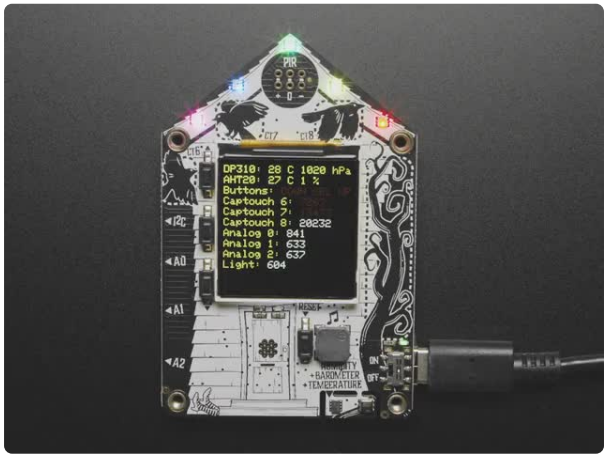


The Adafruit FunHouse is the ideal companion to your Home Assistant setup. You can send not just the temperature, humidity, and barometric pressure using the onboard sensors, but you can also map the attached buttons, slider, and capacitive touch pads to control different parts of your home. You can also use the PIR Sensor to detect motion.

This guide explains how to configure your FunHouse to work with Home Assistant to send environmental data, trigger home automation events using the onboard sensors, and even use the embedded DotStars to emulate an RGB lightbulb that you can control with Home Assistant.



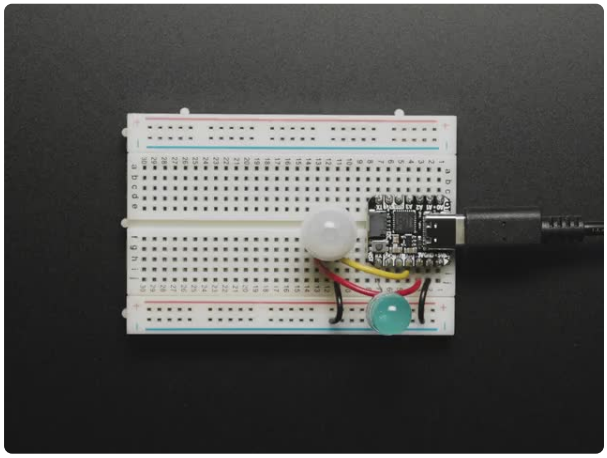
Parts Needed



[Adafruit FunHouse - WiFi Home Automation Development Board]

Home is where the heart is...it's also where we keep all our electronic bits. So why not wire it up with sensors and actuators to turn our house into an electronic wonderland....

https://www.adafruit.com/product/4985



[Breadboard-friendly Mini PIR Motion Sensor with 3 Pin Header]

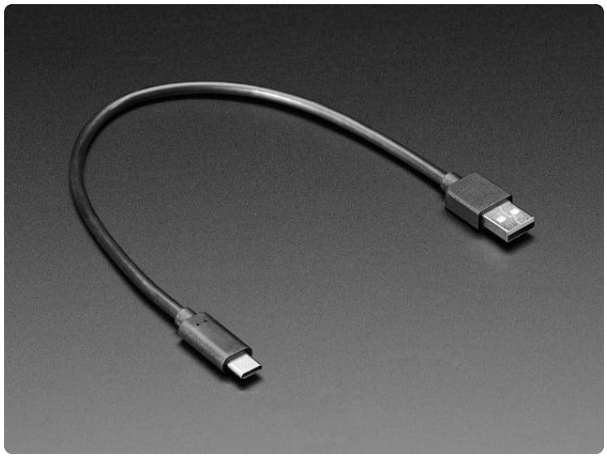
PIR sensors are used to detect motion from pets/humanoids from about 5 meters away (possibly works on zombies, not guaranteed). This sensor is much smaller than most PIR modules, which...

https://www.adafruit.com/product/4871

Mini Magnet Feet for RGB LED Matrices (Pack of 4)

Got a glorious RGB Matrix project you want to mount and display in your workspace or home? If you have one of the matrix panels listed below, you'll need a pack of these...

https://www.adafruit.com/product/4631



USB Type A to Type C Cable - 1ft - 0.3 meter

As technology changes and adapts, so does Adafruit. This USB Type A to Type C cable will help you with the transition to USB C, even if you're still...

https://www.adafruit.com/product/4473

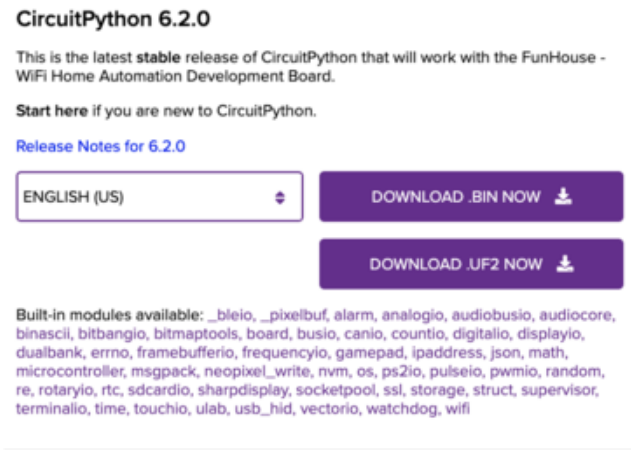
CircuitPython Setup

CircuitPython (https://adafru.it/tB7) is a derivative of MicroPython (https://adafru.it/f9W) 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 FunHouse.

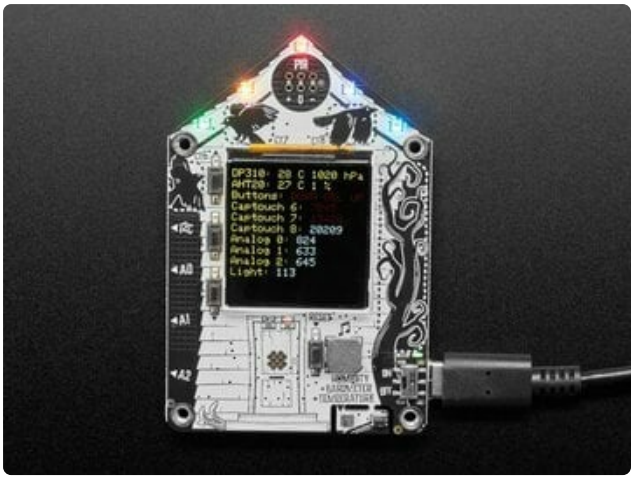
https://adafru.it/RKB



Click the link above and download the latest .BIN and .UF2 file

(depending on how you program the ESP32S2 board you may need one or the other, might as well get both)

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



Plug your FunHouse 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.

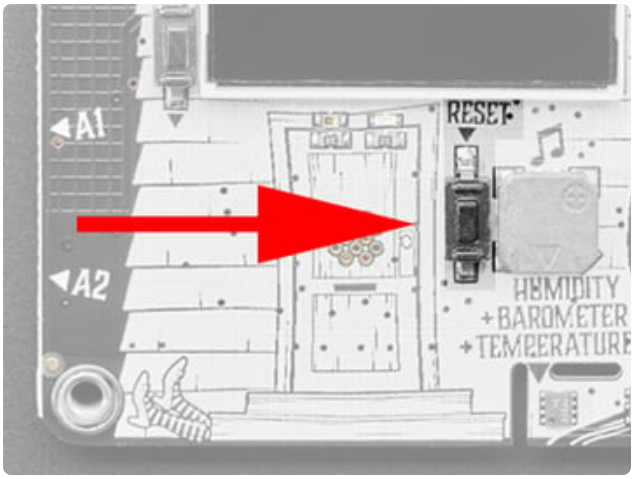
Option 1 - Load with UF2 Bootloader

This is by far the easiest way to load CircuitPython. **However it requires your board has the UF2 bootloader installed. Some early boards do not (we hadn't written UF2 yet!) - in which case you can load using the built in ROM bootloader.**

Still, try this first!

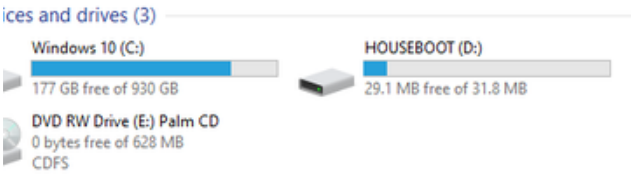
Try Launching UF2 Bootloader

Loading CircuitPython by drag-n-drop UF2 bootloader is the easier way and we recommend it.



Launch UF2 by **double-clicking** the Reset button (the one next to the USB C port). You may have to try a few times to get the timing right.

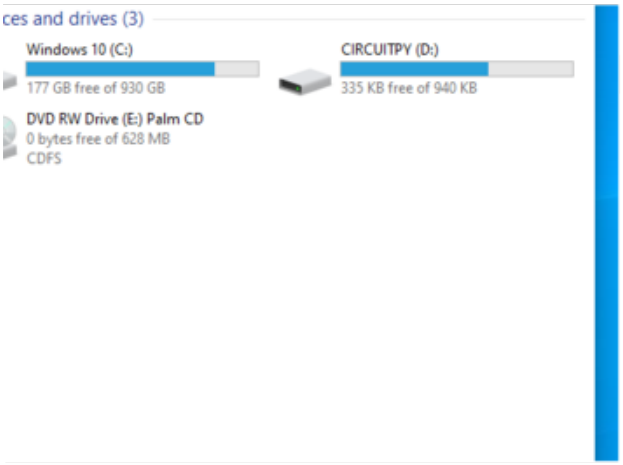
About a half second pause between clicks while the DotStars are purple seems to work well.



If the UF2 bootloader is installed, you will see a new disk drive appear called **HOUSEBOOT**

Copy the **UF2** file you downloaded at the first step of this tutorial onto the **HOUSEBOOT** drive

If you're using Windows and you get an error at the end of the file copy that says **Error from the file copy, Error 0x800701B1: A device which does not exist was specified.** You can ignore this error, the bootloader sometimes disconnects without telling Windows, the install completed just fine and you can continue. [If its really annoying, you can also upgrade the bootloader (the latest version of the UF2 bootloader fixes this warning)](https://adafru.it/RLc) (https://adafru.it/RLc)



Your board should auto-reset into CircuitPython, or you may need to press reset. A **CIRCUITPY** drive will appear. You're done! Go to the next pages.

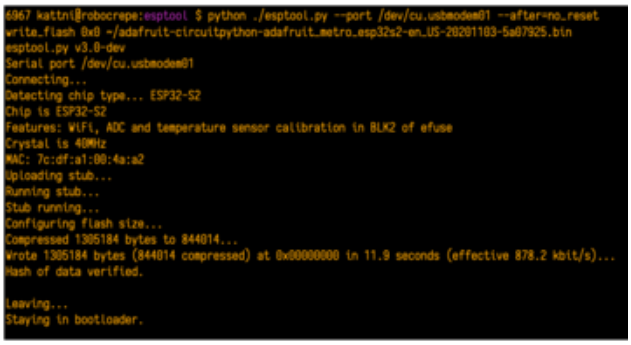
Option 2 - Use Chrome Browser To Upload BIN file

will need to do a full erase prior to uploading new firmware.

The next best option is to try using the Chrome-browser version of esptool we have written. This is handy if you don't have Python on your computer, or something is really weird with your setup that makes esptool not run (which happens sometimes and isn't worth debugging!) You can follow along on the Install UF2 Bootloader (https://adafru.it/RLc) page and either load the UF2 bootloader and then come back to Option 1 on this page, or you can download the CircuitPython BIN file directly using the tool in the same manner as the bootloader.

Option 3 - Use esptool to load BIN file

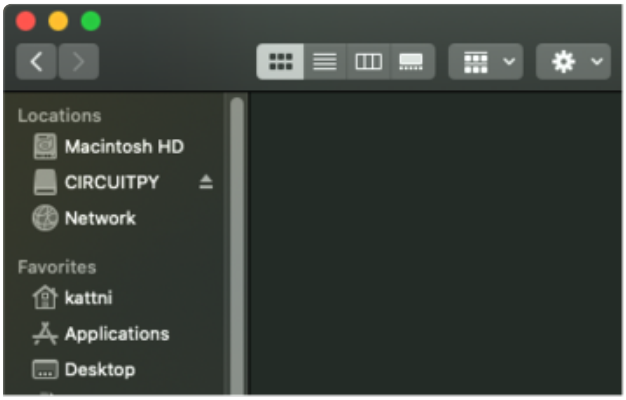
For more advanced users, you can upload with **esptool** to the ROM (hardware) bootloader instead!



Follow the initial steps found in the Run esptool and check connection section of the Install UF2 Bootloader page (https://adafru.it/RLc) 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! Go to the next pages.

CircuitPython Internet Test

One of the great things about most Espressif microcontrollers are their 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 (it will error until WiFi details are added). Click the **Download Project Bundle** button 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.

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

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

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

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

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

In order, the example code...

Checks the ESP32'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 the Sensor

Now let's go over the code that runs on the sensor. The code checks the temperature and humidity, formats it, then publishes directly to the MQTT server.

MQTT Settings in settings.toml

Since the code publishes directly to the MQTT server, there are a few more settings that the code expects to find. If your MQTT server has no username and password, you can change the value to `None`, however in general, the Home Assistant MQTT broker is setup to be password protected by default.

```
MQTT_BROKER = "192.168.1.1"
MQTT_PORT = 1883
```

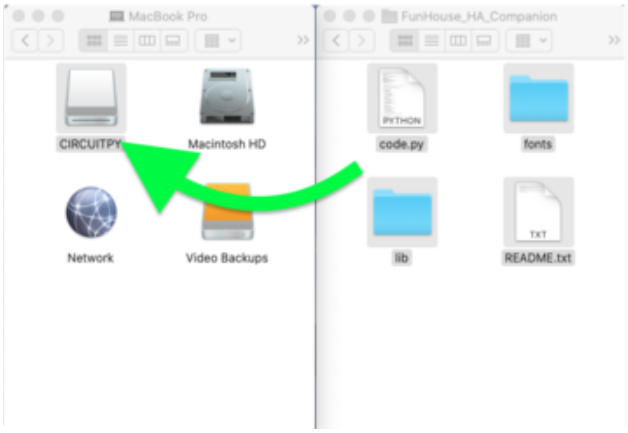
```
MQTT_USERNAME = "myusername"
MQTT_PASSWORD = "mypassword"
```

Temperature Reading

You may notice the temperature comes across a bit higher than the actual room temperature. To see some techniques to improve it, take a look at our Temperature Logger example (https://adafru.it/Sxd).

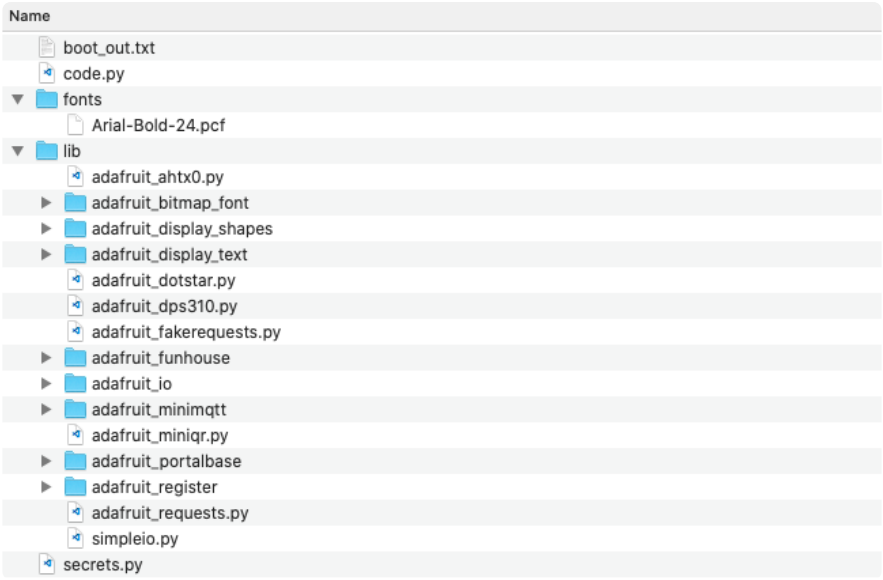
Download the Project Bundle

Your project will use a specific set of CircuitPython libraries and the `code.py` file. In order to get the libraries you need, click on the **Download Project Bundle** link below, and uncompress the .zip file.



Next, drag the contents of the uncompressed bundle directory onto you microcontroller board's CIRCUITPY drive, replacing any existing files or directories with the same names, and adding any new ones that are necessary.

The files on your FunHouse should look like this:




```
# SPDX-FileCopyrightText: 2017 Scott Shawcroft, written for Adafruit Industries
# SPDX-FileCopyrightText: Copyright (c) 2021 Melissa LeBlanc-Williams for Adafruit
Industries
#
# SPDX-License-Identifier: MIT
import os
import time
import json
from displayio import CIRCUITPYTHON_TERMINAL
from adafruit_display_shapes.circle import Circle
from adafruit_funhouse import FunHouse


PUBLISH_DELAY = 60
ENVIRONMENT_CHECK_DELAY = 5
ENABLE_PIR = True
MQTT_TOPIC = "funhouse/state"
LIGHT_STATE_TOPIC = "funhouse/light/state"
LIGHT_COMMAND_TOPIC = "funhouse/light/set"
INITIAL_LIGHT_COLOR = 0x008000
USE_FAHRENHEIT = True

funhouse = FunHouse(default_bg=0x0F0F00)
funhouse.peripherals.dotstars.fill(INITIAL_LIGHT_COLOR)

# Don't display the splash yet to avoid
# redrawing labels after each one is added
funhouse.display.root_group = CIRCUITPYTHON_TERMINAL

# Add the labels
funhouse.add_text(
    text="Temperature:",
    text_position=(20, 30),
    text_color=0xFF8888,
    text_font="fonts/Arial-Bold-24.pcf",
)
temp_label = funhouse.add_text(
    text_position=(120, 60),
    text_anchor_point=(0.5, 0.5),
    text_color=0xFFFF00,
    text_font="fonts/Arial-Bold-24.pcf",
)
funhouse.add_text(
    text="Humidity:",
    text_position=(20, 100),
    text_color=0x8888FF,
```

```
    text_font="fonts/Arial-Bold-24.pcf",
)
hum_label = funhouse.add_text(
    text_position=(120, 130),
    text_anchor_point=(0.5, 0.5),
    text_color=0xFFFF00,
    text_font="fonts/Arial-Bold-24.pcf",
)
funhouse.add_text(
    text="Pressure:",
    text_position=(20, 170),
    text_color=0xFF88FF,
    text_font="fonts/Arial-Bold-24.pcf",
)
pres_label = funhouse.add_text(
    text_position=(120, 200),
    text_anchor_point=(0.5, 0.5),
    text_color=0xFFFF00,
    text_font="fonts/Arial-Bold-24.pcf",
)

# Now display the splash to draw all labels at once
funhouse.display.root_group = funhouse.graphics.root_group

status = Circle(229, 10, 10, fill=0xFF0000, outline=0x880000)
funhouse.graphics.root_group.append(status)

def connected(client, _userdata, _result, _payload):
    status.fill = 0x00FF00
    status.outline = 0x008800
    print("Connected to MQTT! Subscribing...")
    client.subscribe(LIGHT_COMMAND_TOPIC)


def disconnected(_client):
    status.fill = 0xFF0000
    status.outline = 0x880000


def message(_client, topic, payload):
    print("Topic {0} received new value: {1}".format(topic, payload))
    if topic == LIGHT_COMMAND_TOPIC:
        settings = json.loads(payload)
        if settings["state"] == "on":
            if "brightness" in settings:
                funhouse.peripherals.dotstars.brightness = settings["brightness"] /
255
            else:
                funhouse.peripherals.dotstars.brightness = 0.3
            if "color" in settings:
                funhouse.peripherals.dotstars.fill(settings["color"])
        else:
            funhouse.peripherals.dotstars.brightness = 0
        publish_light_state()


def publish_light_state():
    funhouse.peripherals.led = True
    publish_output = {
        "brightness": round(funhouse.peripherals.dotstars.brightness * 255),
        "state": "on" if funhouse.peripherals.dotstars.brightness > 0 else "off",
        "color": funhouse.peripherals.dotstars[0],
    }
    # Publish the Dotstar State
    print("Publishing to {}".format(LIGHT_STATE_TOPIC))
    funhouse.network.mqtt_publish(LIGHT_STATE_TOPIC, json.dumps(publish_output))
    funhouse.peripherals.led = False

# Initialize a new MQTT Client object
```

```
funhouse.network.init_mqtt(
    os.getenv("MQTT_BROKER"),
    os.getenv("MQTT_PORT"),
    os.getenv("MQTT_USERNAME"),
    os.getenv("MQTT_PASSWORD"),
)
funhouse.network.on_mqtt_connect = connected
funhouse.network.on_mqtt_disconnect = disconnected
funhouse.network.on_mqtt_message = message

print("Attempting to connect to {}".format(os.getenv("MQTT_BROKER")))
funhouse.network.mqtt_connect()

last_publish_timestamp = None

last_peripheral_state = {
    "button_up": funhouse.peripherals.button_up,
    "button_down": funhouse.peripherals.button_down,
    "button_sel": funhouse.peripherals.button_sel,
    "captouch6": funhouse.peripherals.captouch6,
    "captouch7": funhouse.peripherals.captouch7,
    "captouch8": funhouse.peripherals.captouch8,
}

if ENABLE_PIR:
    last_peripheral_state["pir_sensor"] = funhouse.peripherals.pir_sensor

environment = {}
last_environment_timestamp = time.monotonic()

# Provide Initial light state
publish_light_state()

while True:
    if not environment or (
        time.monotonic() - last_environment_timestamp > ENVIRONMENT_CHECK_DELAY
    ):
        temp = funhouse.peripherals.temperature
        unit = "C"
        if USE_FAHRENHEIT:
            temp = temp * (9 / 5) + 32
            unit = "F"

        environment["temperature"] = temp
        environment["pressure"] = funhouse.peripherals.pressure
        environment["humidity"] = funhouse.peripherals.relative_humidity
        environment["light"] = funhouse.peripherals.light

        funhouse.set_text("{:.1f}{}".format(environment["temperature"], unit),
temp_label)
        funhouse.set_text("{:.1f}%".format(environment["humidity"]), hum_label)
        funhouse.set_text("{}hPa".format(environment["pressure"]), pres_label)
        last_environment_timestamp = time.monotonic()
    output = environment

    peripheral_state_changed = False
    for peripheral in last_peripheral_state:
        current_item_state = getattr(funhouse.peripherals, peripheral)
        output[peripheral] = "on" if current_item_state else "off"
        if last_peripheral_state[peripheral] != current_item_state:
            peripheral_state_changed = True
            last_peripheral_state[peripheral] = current_item_state

    if funhouse.peripherals.slider is not None:
        output["slider"] = funhouse.peripherals.slider
        peripheral_state_changed = True

    # Every PUBLISH_DELAY, write temp/hum/press/light or if a peripheral changed
    if (
```

```
        last_publish_timestamp is None
        or peripheral_state_changed
        or (time.monotonic() - last_publish_timestamp) > PUBLISH_DELAY
    ):
        funhouse.peripherals.led = True
        print("Publishing to {}".format(MQTT_TOPIC))
        funhouse.network.mqtt_publish(MQTT_TOPIC, json.dumps(output))
        funhouse.peripherals.led = False
        last_publish_timestamp = time.monotonic()

    # Check any topics we are subscribed to
    funhouse.network.mqtt_loop(0.5)
```

How the Code Works

First there are our imports. Many of the imports include built-in modules such as `json` as well as the `adafruit_display_shapes`, and the `adafruit_funhouse` libraries.

```
import os
import time
import json
from adafruit_display_shapes.circle import Circle
from adafruit_funhouse import FunHouse
```

In the next section, there are quite a few settings that you can adjust.

First, the `PUBLISH_DELAY` setting is the amount of time in seconds to wait before updating the temperature and humidity.

The `ENVIRONMENT_CHECK_DELAY` is the amount of time to delay in between reading the environmental sensors such as temperature and humidity and updating the labels. The reason we have a delay is to give time to other events.

The `ENABLE_PIR` setting can be set to False if you either do not have a PIR sensor or would not like the sensor to automatically trigger anything.

The `MQTT_TOPIC` is the topic that holds the state of the FunHouse sensors. To read more about MQTT Topics, you can check out the MQTT Topics section of our All the Internet of Things Protocols (https://adafru.it/Fmb) guide.

The `LIGHT_STATE_TOPIC` is the topic that holds the state of the DotStar LEDs and is tells Home Assistant what the current state of the LEDS is to allow it to act like an RGB bulb.

The `LIGHT_COMMAND_TOPIC` is the topic that the FunHouse listens to in order to change the state of the DotStar LEDs through Home Assistant.

The `INITIAL_LIGHT_COLOR` setting is the initial color of the DotStar LEDs.

If `USE_FAHRENHEIT` is set to True, the temperature displayed on the screen will be in Fahrenheit. **Changing this setting also affects the units that are sent to Home Assistant.**

```
PUBLISH_DELAY = 60
ENVIRONMENT_CHECK_DELAY = 5
ENABLE_PIR = True
MQTT_TOPIC = "funhouse/state"
LIGHT_STATE_TOPIC = "funhouse/light/state"
LIGHT_COMMAND_TOPIC = "funhouse/light/set"
INITIAL_LIGHT_COLOR = 0x008000
USE_FAHRENHEIT = True
```

Next the FunHouse is initialized with a default background and the `INITIAL_LIGHT_COLOR`.

```
funhouse = FunHouse(default_bg=0x0F0F00)
funhouse.peripherals.dotstars.fill(INITIAL_LIGHT_COLOR)
```

In this next section, the display labels are created. `funhouse.display.root_group = CIRCUITPYTHON_TERMINAL` is called first first so that it doesn't draw each label sequentially and end the section with `funhouse.display.root_group = funhouse.graphics.root_group` to trigger drawing everything that's in the `root_group`, which is the group all labels are added to when created.

```
funhouse.display.root_group = CIRCUITPYTHON_TERMINAL
funhouse.add_text(
    text="Temperature:",
    text_position=(20, 30),
    text_color=0xFF8888,
    text_font="fonts/Arial-Bold-24.pcf",
)
temp_label = funhouse.add_text(
    text_position=(120, 60),
    text_anchor_point=(0.5, 0.5),
    text_color=0xFFFF00,
    text_font="fonts/Arial-Bold-24.pcf",
)
funhouse.add_text(
    text="Humidity:",
    text_position=(20, 100),
    text_color=0x8888FF,
    text_font="fonts/Arial-Bold-24.pcf",
)
hum_label = funhouse.add_text(
    text_position=(120, 130),
    text_anchor_point=(0.5, 0.5),
    text_color=0xFFFF00,
    text_font="fonts/Arial-Bold-24.pcf",
)
funhouse.add_text(
    text="Pressure:",
    text_position=(20, 170),
    text_color=0xFF88FF,
    text_font="fonts/Arial-Bold-24.pcf",
```

```
)
pres_label = funhouse.add_text(
    text_position=(120, 200),
    text_anchor_point=(0.5, 0.5),
    text_color=0xFFFF00,
    text_font="fonts/Arial-Bold-24.pcf",
)
funhouse.display.root_group = funhouse.graphics.root_group
```

Next the script creates the circle to indicated whether the board is connected to the MQTT server or not. It is created with a default color of red.

```
status = Circle(229, 10, 10, fill=0xFF0000, outline=0x880000)
funhouse.graphics.root_group.append(status)
```

After that are the MQTT `connect` and `disconnect` handlers. These handle updating the connection status indicator and subscribing to any topics.

```
def connected(client, _userdata, _result, _payload):
    status.fill = 0x00FF00
    status.outline = 0x008800
    print("Connected to MQTT! Subscribing...")
    client.subscribe(LIGHT_COMMAND_TOPIC)


def disconnected(_client):
    status.fill = 0xFF0000
    status.outline = 0x880000
```

For the message handler, it will wait for the any new message that it is subscribed to. Then it will make sure the topic is the value of the `LIGHT_COMMAND_TOPIC` before proceeding. Next it will decode the payload as JSON. If the **state** is `"on"`, it will look for additional settings in the payload.

If it finds **brightness**, it will convert it from 0-255 to 0-1.0 and set the DotStars to that brightness. If it finds **color**, it will set the DotStars to that color using the `fill()` command. After it changes the DotStars, it will publish the light state so the Home Assistant UI can update its settings.

```
def message(_client, topic, payload):
    print("Topic {0} received new value: {1}".format(topic, payload))
    if topic == LIGHT_COMMAND_TOPIC:
        settings = json.loads(payload)
        print(settings)
        if settings["state"] == "on":
            if "brightness" in settings:
                funhouse.peripherals.dotstars.brightness = settings["brightness"] /
255
            else:
                funhouse.peripherals.dotstars.brightness = 0.3
            if "color" in settings:
```

```
                funhouse.peripherals.dotstars.fill(settings["color"])
        else:
            funhouse.peripherals.dotstars.brightness = 0
        publish_light_state()
```

The `publish_light_state()` function will gather the current state of the DotStars and publish it in a way that is meaningful to Home Assistant.

```
def publish_light_state():
    funhouse.peripherals.led = True
    publish_output = {
        "brightness": round(funhouse.peripherals.dotstars.brightness * 255),
        "state": "on" if funhouse.peripherals.dotstars.brightness &gt; 0 else "off",
        "color": funhouse.peripherals.dotstars[0],
    }
    # Publish the Dotstar State
    print("Publishing to {}".format(LIGHT_STATE_TOPIC))
    funhouse.network.mqtt_publish(LIGHT_STATE_TOPIC, json.dumps(publish_output))
    funhouse.peripherals.led = False
```

With the functions all defined, MQTT is initialized in the FunHouse library using the settings in the secrets file. The handler function are then set and the code attempts to connect to the MQTT server. The `os.getenv()` function is used to get settings from settings.toml.

```
# Initialize a new MQTT Client object
funhouse.network.init_mqtt(
    os.getenv("MQTT_BROKER"),
    os.getenv("MQTT_PORT"),
    os.getenv("MQTT_USERNAME"),
    os.getenv("MQTT_PASSWORD"),
)
funhouse.network.on_mqtt_connect = connected
funhouse.network.on_mqtt_disconnect = disconnected
funhouse.network.on_mqtt_message = message

print("Attempting to connect to {}".format(os.getenv("MQTT_BROKER")))
funhouse.network.mqtt_connect()
```

After that, some data variables are set up with initial states. This includes a variable to hold the last time that we published the general FunHouse state to MQTT and the last values of the peripheral so we can monitor when they change.

We also update the environment and display with the `update_enviro()` function we defined earlier.

Finally we publish the current state of the DotStars with `publish_light_state()`.

```
last_publish_timestamp = None

last_peripheral_state = {
```

```
    "button_up": funhouse.peripherals.button_up,
    "button_down": funhouse.peripherals.button_down,
    "button_sel": funhouse.peripherals.button_sel,
    "captouch6": funhouse.peripherals.captouch6,
    "captouch7": funhouse.peripherals.captouch7,
    "captouch8": funhouse.peripherals.captouch8,
}

if ENABLE_PIR:
    last_peripheral_state["pir_sensor"] = funhouse.peripherals.pir_sensor

environment = {}
last_environment_timestamp = time.monotonic()

# Provide Initial light state
publish_light_state()
```

Now we get to the main loop.

First the code checks if the amount of time in `ENVIRONMENT_CHECK_DELAY` has passed and if so, it will gather the environmental and light sensor settings, change the temperature to Fahrenheit if set to do so, update the `environment` dictionary, and change the labels on the display. The current environment dictionary is used as the basis for the output data.

```
if not environment or (time.monotonic() - last_environment_timestamp &gt;
ENVIRONMENT_CHECK_DELAY):
    temp = funhouse.peripherals.temperature
    unit = "C"
    if USE_FAHRENHEIT:
        temp = temp * (9 / 5) + 32
        unit = "F"

    environment["temperature"] = temp
    environment["pressure"] = funhouse.peripherals.pressure
    environment["humidity"] = funhouse.peripherals.relative_humidity
    environment["light"] = funhouse.peripherals.light

    funhouse.set_text("{:.1f}{}".format(environment["temperature"], unit),
temp_label)
    funhouse.set_text("{:.1f}%".format(environment["humidity"]), hum_label)
    funhouse.set_text("{}hPa".format(environment["pressure"]), pres_label)
    last_environment_timestamp = time.monotonic()
output = environment
```

In the next section each of the peripherals is checked and updated. If anything has changed, the `peripheral_state_changed` variable is set to **True** which triggers publishing an update a bit later on.

```
peripheral_state_changed = False
for peripheral in last_peripheral_state:
    current_item_state = getattr(funhouse.peripherals, peripheral)
    output[peripheral] = "on" if current_item_state else "off"
    if last_peripheral_state[peripheral] != current_item_state:
```

```
        peripheral_state_changed = True
        last_peripheral_state[peripheral] = current_item_state
```

Since the slider returns a number between 0-1 or None if not touched, this needs to be checked separately from the other Peripherals and is only added if it is being used.

```
if funhouse.peripherals.slider is not None:
    output["slider"] = funhouse.peripherals.slider
    peripheral_state_changed = True
```

If the `peripheral_state_changed` variable is **True** or the amount of time in `PUBLISH_DELAY` has elapsed, the state of the FunHouse is published.

```
if (
    last_publish_timestamp is None
    or peripheral_state_changed
    or (time.monotonic() - last_publish_timestamp) &gt; PUBLISH_DELAY
):
    funhouse.peripherals.led = True
    print("Publishing to {}".format(MQTT_TOPIC))
    funhouse.network.mqtt_publish(MQTT_TOPIC, json.dumps(output))
    funhouse.peripherals.led = False
    last_publish_timestamp = time.monotonic()
```

Finally the MQTT loop is run. In order to increase the responsiveness of the buttons, a `timeout` value of **0.5** seems to work well. If the value is too small, it won't have enough time to respond to MQTT messages and if it is too large, it is possible to miss peripheral change events.

```
# Check any topics we are subscribed to
funhouse.network.mqtt_loop(0.5)
```

Debugging the Sensor

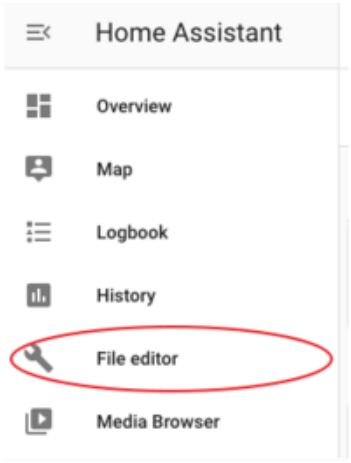
If you would like to monitor what the sensor is doing, you can look at our guide on Connecting to the Serial Console (https://adafru.it/Bec) with CircuitPython. Once you are connected, it can help with any troubleshooting.

Home Assistant Configuration

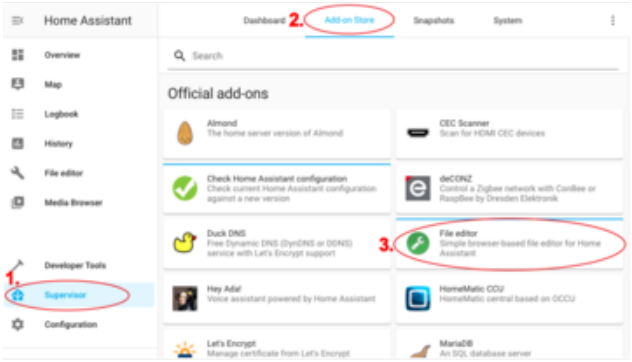
This guide assumes you already have a working and running Home Assistant server. If you don't, be sure to visit our Set up Home Assistant with a Raspberry Pi (https://adafru.it/lbd) guide first.

Check Your Add-Ons

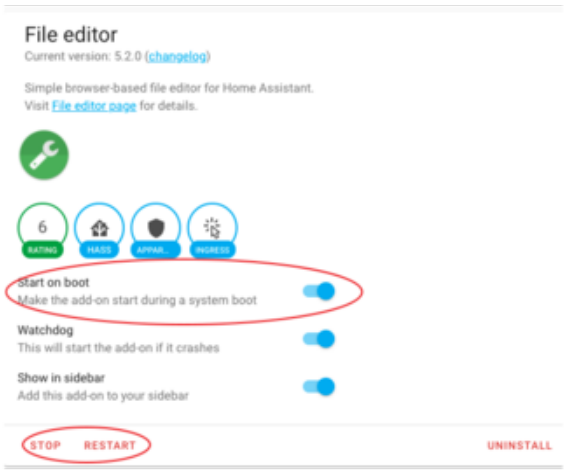
Start out by logging in and opening up your Home Assistant dashboard and checking that the File editor is installed.



As part of the setup, you should have an add-on either called **configurator** or **File editor** with a wrench icon next to it. Go ahead and select it.



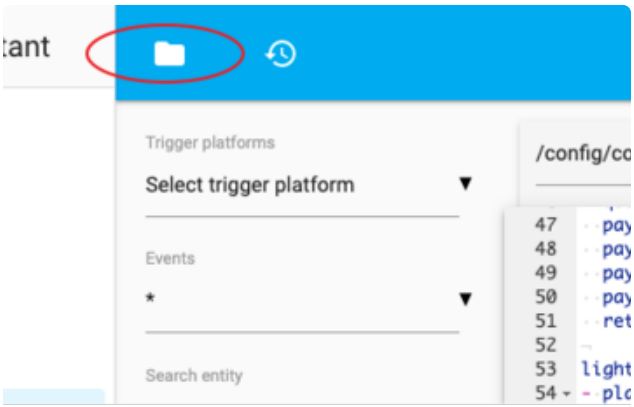
If you don't see it, it may not be installed. You can find it under **Settings → Add-ons → Add-on Store → File editor** and go through the installation procedure.

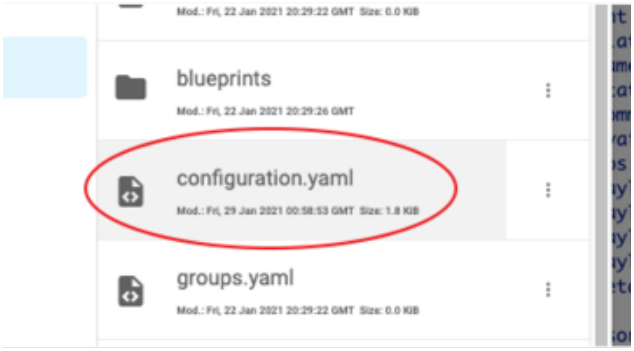


If you already have it, but it's just not showing up, be sure it is started and the option to show in the sidebar is selected.

Environment Data Setup

Next you'll want to open up the File Editor and add the Environment Data.

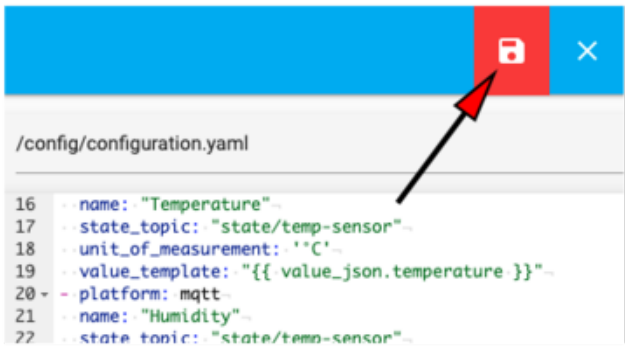




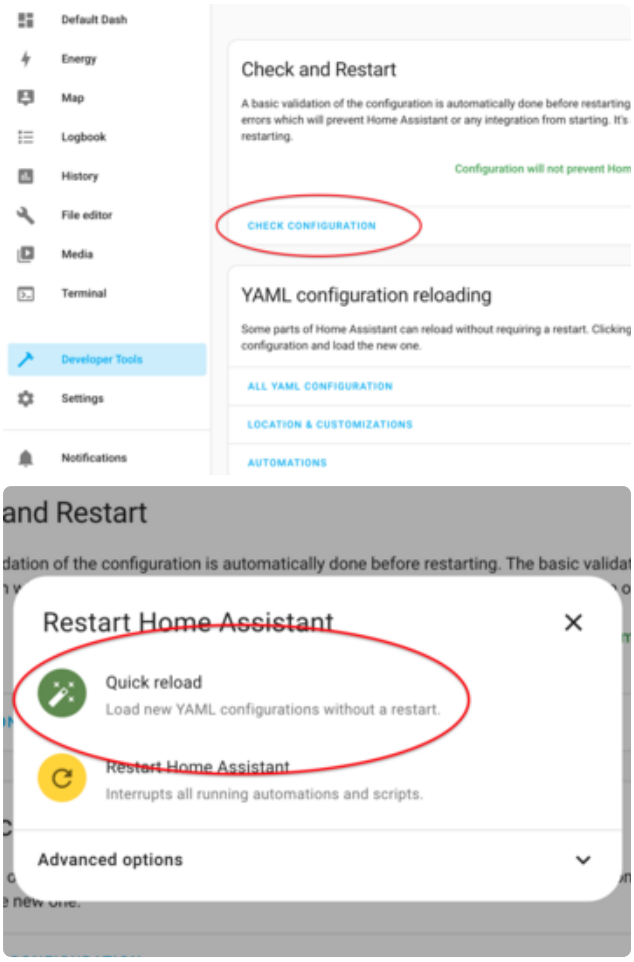
Click on the Folder Icon at the top and select **configuration.yaml**, then click on an area to the right of the file list to close it.

Add the following code to the bottom of the configuration file. Make sure the `state_topic` values match the `MQTT_TOPIC` value you used in the sensor code.

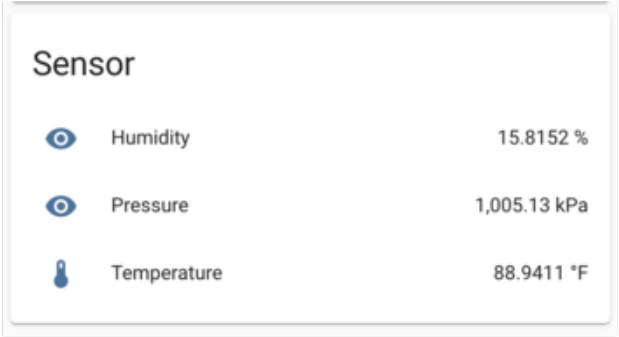
```
mqtt:
  sensor:
  - name: "Temperature"
    state_topic: "funhouse/state"
    unit_of_measurement: '°F'
    value_template: "{{ value_json.temperature }}"
  - name: "Humidity"
    state_topic: "funhouse/state"
    unit_of_measurement: '%'
    value_template: "{{ value_json.humidity }}"
  - name: "Pressure"
    state_topic: "funhouse/state"
    unit_of_measurement: 'hPa'
    value_template: "{{ value_json.pressure }}"
```



Click the save button at the top.



From the **Developer Tools** menu, you can check that the configuration is valid and click on **Restart** to load the configuration changes you made. You can just click **Quick reload** to reload any changes you made.



With the latest releases of Home Assistant, a LoveLace dashboard was added. If you haven't edited the Dashboard, it should automatically appear.

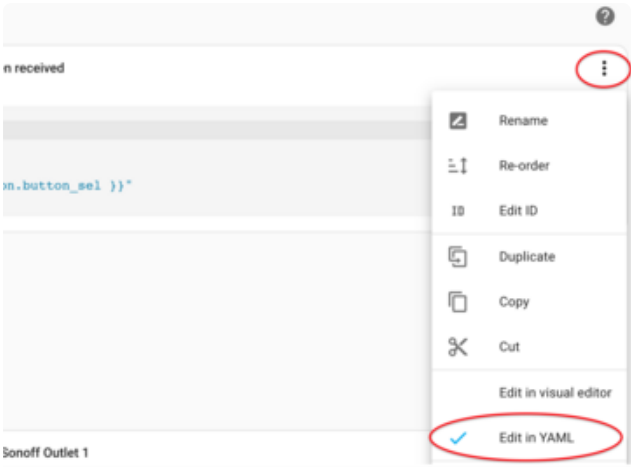
Otherwise, you may need to manually add a card to the dashboard.

Using the Peripherals for Automation

To create an automation event and use one of the peripherals to trigger an existing entity, is pretty easy. For instance, if you had a light named **My_Light** that you wanted to toggle when the **Select button** is pressed, you would add some code to **automations.yaml** similar to the following:

```
- id: button_sel
  alias: "Switch Shortcut"
```

```
  trigger:
  - platform: mqtt
    topic: funhouse/state
    payload: 'on'
    value_template: '{{ value_json.button_sel }}'
  action:
  - service: light.toggle
    data: {}
    target:
      entity_id: light.My_Light
  mode: single
```



You can also add automations by going to the Settings menu and choosing **Automations & Scenes**. The visual editor gives you less control so that you aren't able to enter a `value_template` on the MQTT trigger. However, if you select the 3 dot menu, you can can choose **Edit in YAML**, which will allow you to enter this value.

ther useful trigger is switch.toggle.

This adds an Automation trigger that checks the **button_sel** JSON value in the `funhouse/state` MQTT topic for a value of **on**. When these conditions are met, the action is triggered.

In the action section, the **light.toggle** event is fired on the **light.My_Light** entity.

Be sure to **Restart your Server** as described in the Environment Data Setup section.

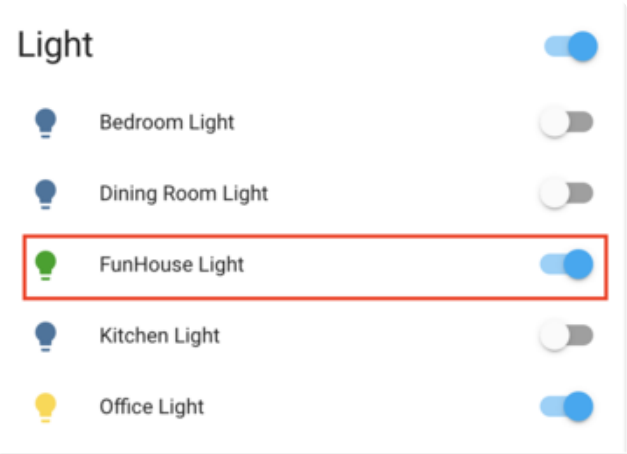
Once you have restarted, try pressing the Middle button on your FunHouse. It should toggle your light. You may need to hold it down for a second or so for it to activate.

Emulating an RGB Bulb

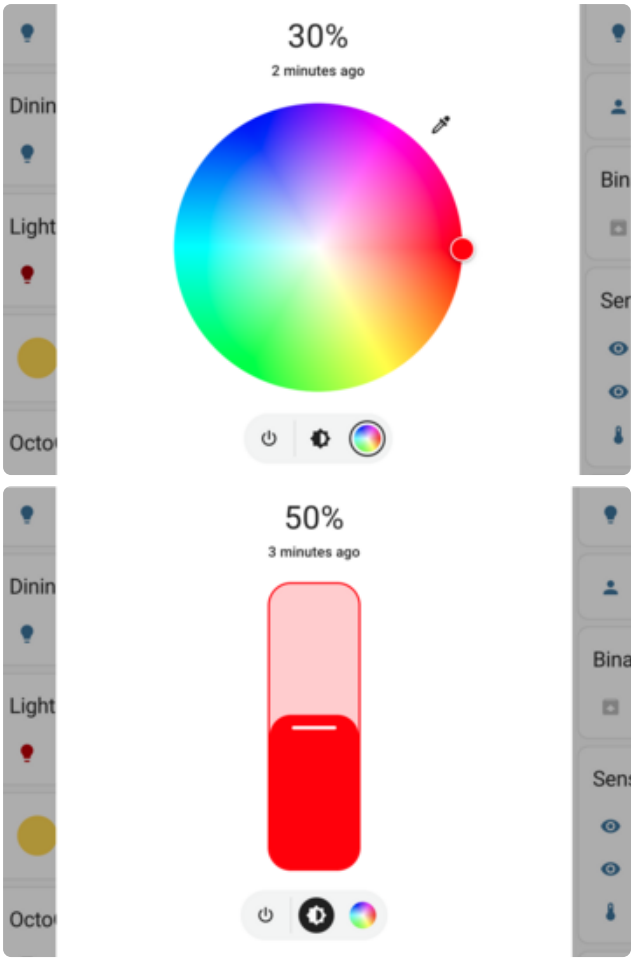
You can also emulate an RGB light with the FunHouse DotStar LEDs. You'll want to add the following code to your configuration.

mqtt sections of your config will need to be combined or it will show as an r in your configuration.

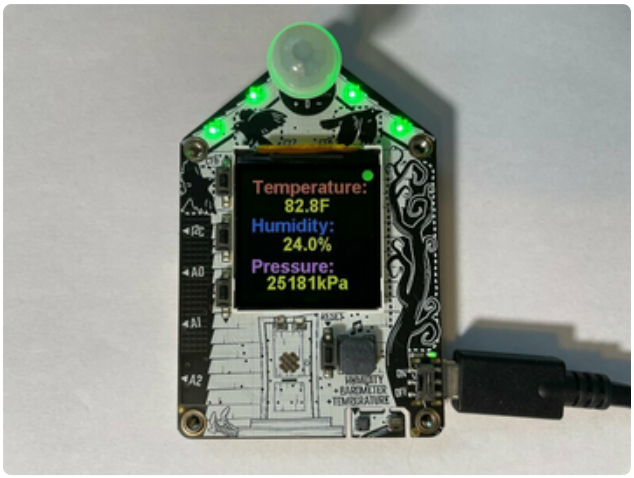
```
mqtt:
  light:
  - name: "FunHouse Light"
    schema: template
    command_topic: "funhouse/light/set"
    state_topic: "funhouse/light/state"
    command_on_template: &gt;
      {"state": "on"
      {%- if brightness is defined -%}
      , "brightness": {{ brightness }}
      {%- endif -%}
      {%- if red is defined and green is defined and blue is defined -%}
      , "color": [{{ red }}, {{ green }}, {{ blue }}]
      {%- endif -%}
      }
    command_off_template: '{"state": "off"}'
    state_template: "{{ value_json.state }}"
    brightness_template: '{{ value_json.brightness }}'
    red_template: '{{ value_json.color[0] }}'
    green_template: '{{ value_json.color[1] }}'
    blue_template: '{{ value_json.color[2] }}'
```



Once you have added that to your configuration, go ahead and restart the server. The **FunHouse Light** should appear under your lights.



Go ahead and click on the bulb icon and a Color and Brightness dialog should come up. Go ahead and change the values.



Changing these values should update the value on your FunHouse.

Troubleshooting

If you see the icons, but there is no data, it is easiest to start by checking the MQTT messages. We have a guide on how to use Desktop MQTT Client for Adafruit.io (https://adafru.it/kID), which can be used for the Home Assistant MQTT server as well.

Go ahead and configure a username and password to match your MQTT server and connect. Under **subscribe**, you can subscribe to the `#` topic to get all messages.

If you are seeing messages from the sensor, you may want to double check your Home Assistant configuration.

If you don't see any messages, you will want to follow the debugging section on the **Code the Sensor** page.