Motion Activated Outlet with the Adafruit FunHouse

Created by Melissa LeBlanc-Williams

https://learn.adafruit.com/motion-activated-outlet-with-the-adafruit-funhouse

Last updated on 2025-04-19 08:46:44 PM EDT

Table of Contents

Overview 3

- Parts

Wiring 6

CircuitPython 7

- Set Up CircuitPython
- Option 1 - Load with UF2 Bootloader
- Option 2 - Use Chrome Browser To Upload BIN file
- Option 3 - Use esptool to load BIN file

CircuitPython Internet Test 11

- The settings.toml File
- IPv6 Networking

Coding the Motion Sensor 18

- MQTT Secrets Settings
- Code Walkthrough

Home Assistant Configuration 23

- Check Your Add-Ons
- Creating a Switched Outlet
- Troubleshooting

Motion Sensor Usage 27

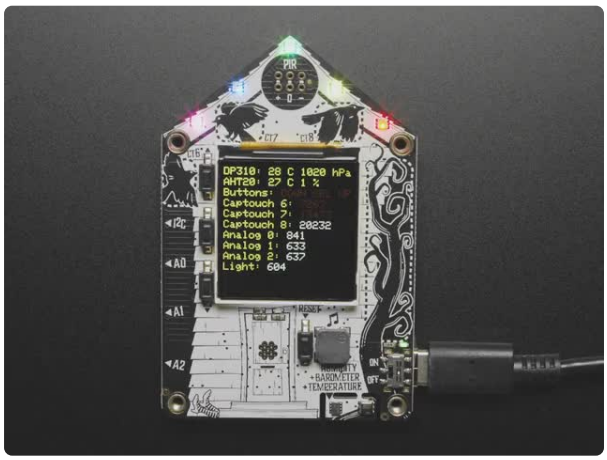
Overview

Motion-activated devices have been around for a while. They are an excellent way to save money if you have a habit of leaving the lights or fan on while you're not in the room.

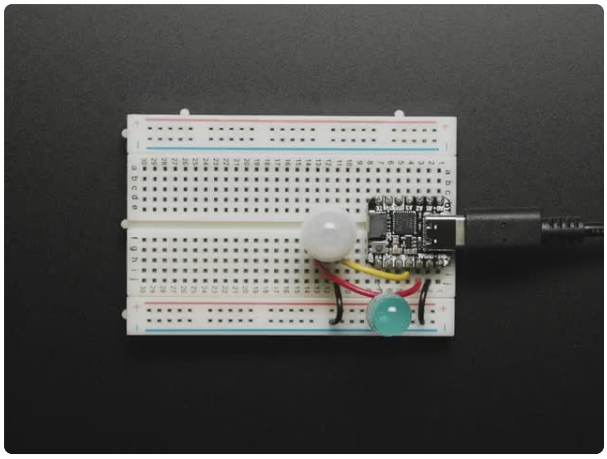
With the FunHouse, it's easy to set up your own motion activated device using a PIR sensor. This guide will take you through setting up a FunHouse board to control an outlet strip that you can plug a light or fan into. For this guide, we're going to use a fan.

This project is designed so you can either use the FunHouse as a standalone device or interface with Home Assistant to optionally control the device as well. If you want to get creative, you can use it to do things like automatically turning off your TV to encourage you to keep moving around every so often.

Parts



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

Controllable Four Outlet Power Relay Module version 2

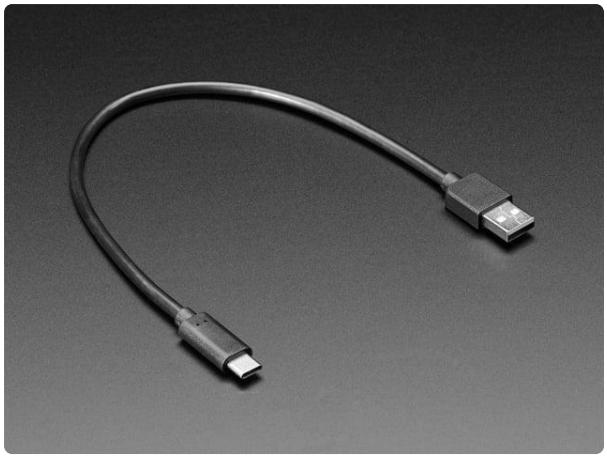
Say goodbye to hazardous high voltage wiring and create the Internet of Things with safe, reliable power control. The IoT...
https://www.adafruit.com/product/2935

STEMMA JST PH 2mm 3-Pin to Male Header Cable - 200mm

This cable will let you turn a JST PH 3-pin cable port into 3 individual wires with high-quality 0.1" male header plugs on the end. We're carrying these to match up with our...
https://www.adafruit.com/product/3893

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

Extending the Wires

If you would like to extend the connection between the FunHouse and the Outlet Relay Module, Adafruit has the parts for that too.

[Stranded-Core Wire Spool - 25ft - 22AWG - White]
This wire is flexible, strong, and super easy to solder, too!Stranded core wire is best used for wiring jigs where theres bending or movement expected. Works well with terminal...
https://www.adafruit.com/product/2997

[Stranded-Core Wire Spool - 25ft - 22AWG - Black]
This wire is flexible, strong, and super easy to solder, too! Stranded core wire is best used for wiring jigs where theres bending or movement expected. Works well with...
https://www.adafruit.com/product/2976

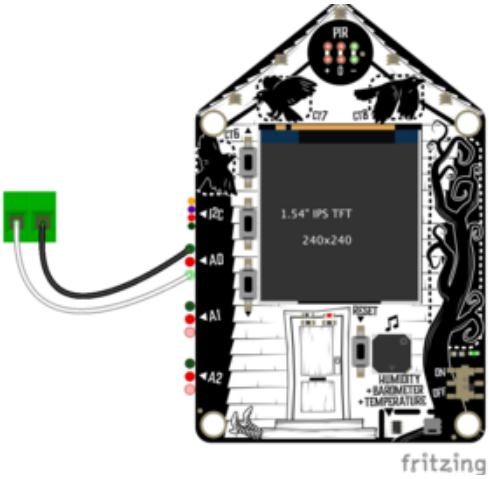
[2.0mm Pitch Connector Kit - JST PH Compatible - 220 Piece Kit]
Totally 220 pieces, this 2.0mm Pitch Kit with JST-PH Connectors is a must-have for your workstation. You'll have enough sockets and...
https://www.adafruit.com/product/4422

[Universal Micro Crimping Pliers - 1.0 to 1.9mm Size Contacts]
Make cables and wiring harnesses with ease using the best crimping pliers we've used. These precision machined crimpers will turn you into a pro, with no more painful struggles...
https://www.adafruit.com/product/350

[Wire Ferrule Kit - 800 pieces]
Crimping's not just for 80s hair metal bands! When your project requires repeated insertion and removal with minimal strain, why not check out the reliable wire...
https://www.adafruit.com/product/5131

Wiring

Wiring the FunHouse to the outlet is really simple and there are a number of options, but the wiring is the same. There is a small terminal block that can be removed from the side of the outlet to make connecting the wire much easier.



Connect the **white wire** from the JST connector to the positive side of the terminal.
Connect the **black wire** from the JST connector to the ground side of the terminal.
Leave the **red wire** disconnected

CircuitPython

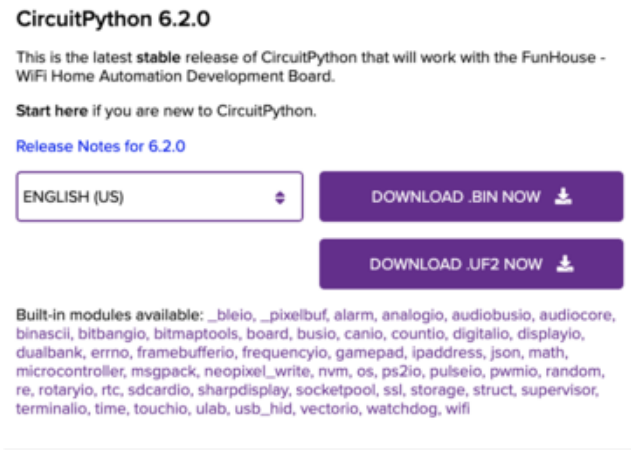
CircuitPython (https://adafru.it/tB7) is a derivative of MicroPython (https://adafru.it/f9W) designed to simplify experimentation and education on low-cost microcontrollers. It makes it easier than ever to get prototyping by requiring no upfront desktop software downloads. Simply copy and edit files on the **CIRCUITPY** drive to iterate.

Set Up CircuitPython

Follow the steps to get CircuitPython installed on your FunHouse.

Download the latest CircuitPython for your board from circuitpython.org

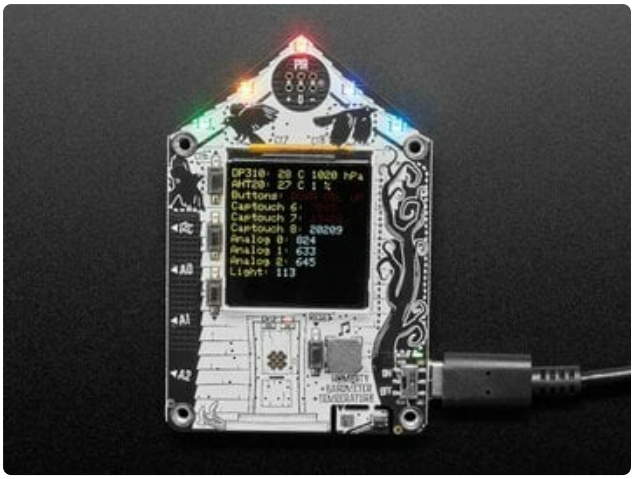
https://adafru.it/RKB



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.

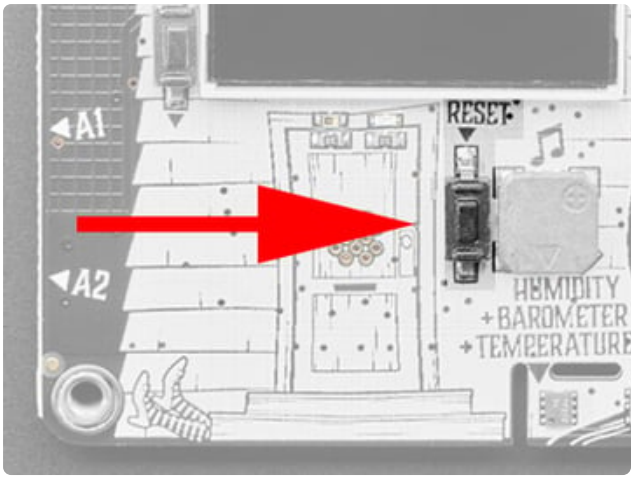
Option 1 - Load with UF2 Bootloader

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!

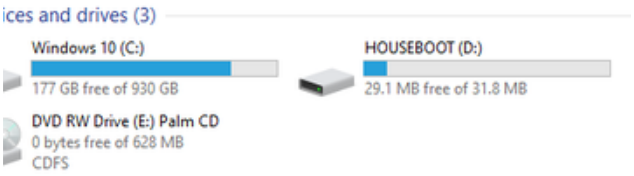
Try Launching UF2 Bootloader

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.

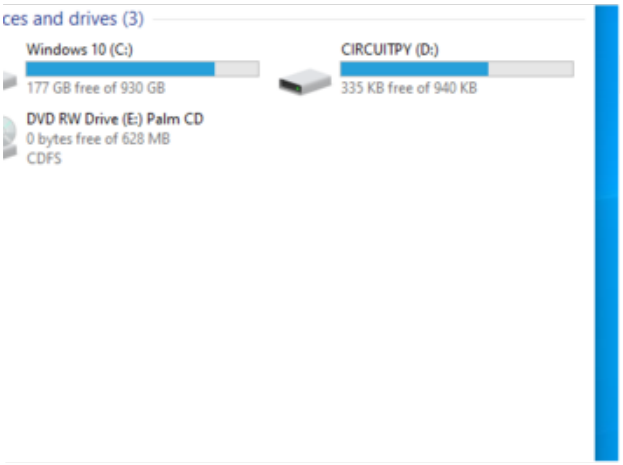
About a half second pause between clicks while the DotStars are purple seems to work well.



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.

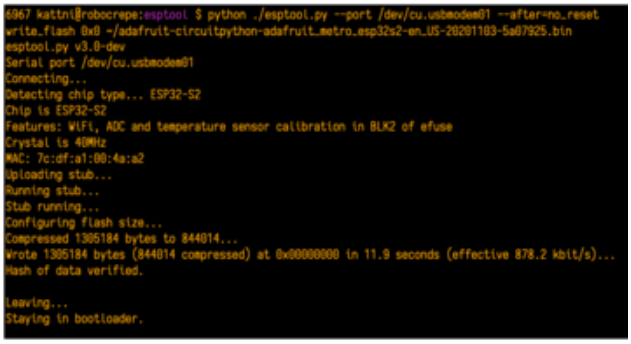
Option 2 - Use Chrome Browser To Upload BIN file

You 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

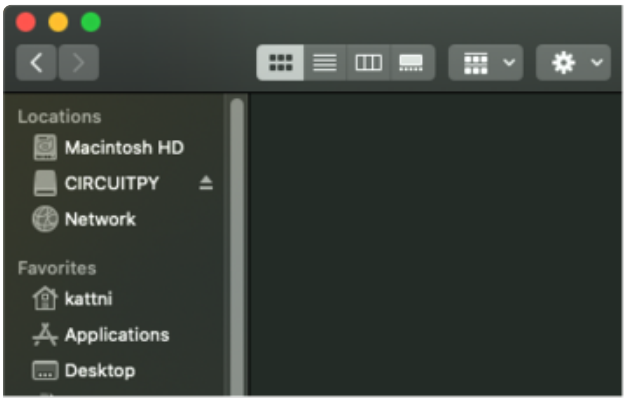
For more advanced users, you can upload with **esptool** to the ROM (hardware) bootloader instead!



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. 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!

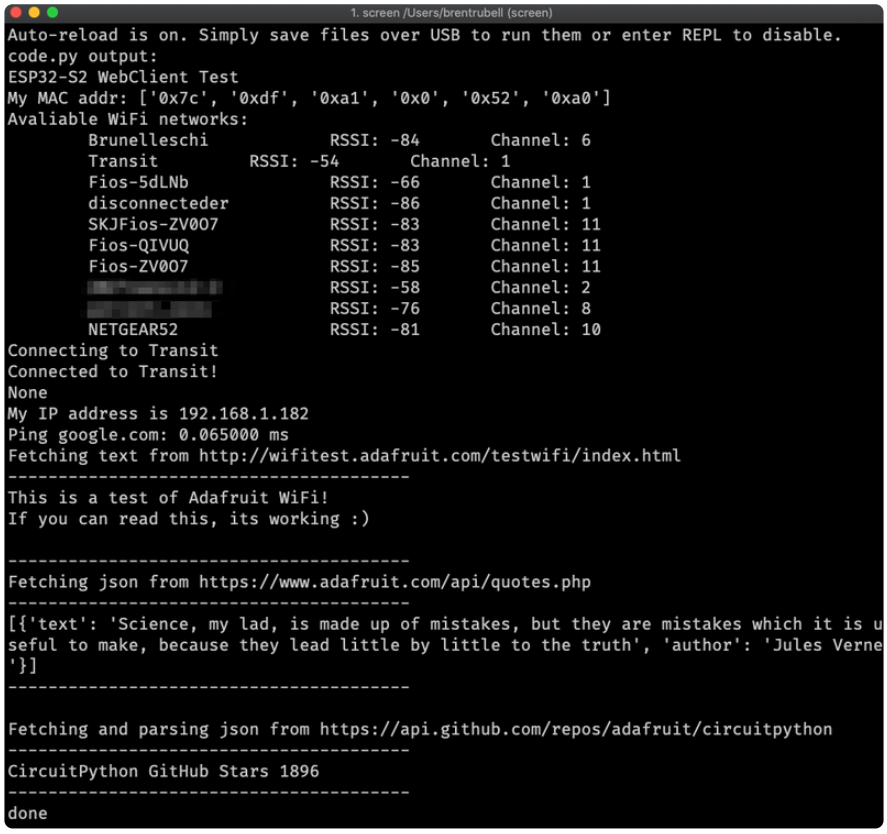
As you make projects you may need more tokens and keys, just add them one line at a time. See for example other tokens such as one for accessing GitHub or the Hackaday API. Other non-secret data like your timezone can also go here.

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')
```

Coding the Motion Sensor

Let's start out with the code that goes onto the FunHouse. This code can run with or without Home Assistant and we'll go over the options.

MQTT Secrets Settings

Since the code publishes directly to the MQTT server, there are a few more **secrets.py** file 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 set up to be password protected by default.

```
MQTT_BROKER = "192.168.1.1"
MQTT_PORT = 1883
MQTT_USERNAME = "myusername"
MQTT_PASSWORD = "mypassword"
```

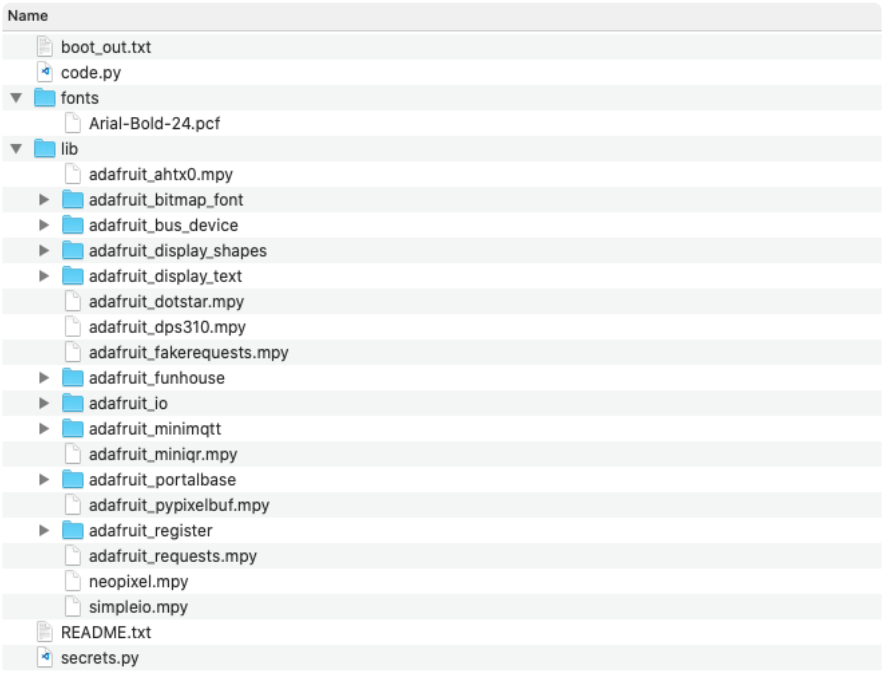
To add code and libraries to your FunHouse, click the **Download Project Bundle** button to get the code and all of the libraries.

```
# 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 board
import digitalio
from displayio import CIRCUITPYTHON_TERMINAL
from adafruit_display_shapes.circle import Circle
from adafruit_funhouse import FunHouse

OUTLET_STATE_TOPIC = "funhouse/outlet/state"
OUTLET_COMMAND_TOPIC = "funhouse/outlet/set"
MOTION_TIMEOUT = 300  # Timeout in seconds
USE_MQTT = True

# Use dict to avoid reassigning the variable
timestamps = {
    "last_pir": None
}

def set_outlet_state(value):
    if value:
        funhouse.peripherals.dotstars.fill(0x00FF00)
        timestamps["last_pir"] = time.monotonic()
    else:
        funhouse.peripherals.dotstars.fill(0xFF0000)
        timestamps["last_pir"] = time.monotonic() - MOTION_TIMEOUT

    outlet.value = value
    publish_outlet_state()

def publish_outlet_state():
    if USE_MQTT:
        funhouse.peripherals.led = True
```

```
        output = "on" if outlet.value else "off"
        # Publish the Dotstar State
        print("Publishing to {}".format(OUTLET_STATE_TOPIC))
        funhouse.network.mqtt_publish(OUTLET_STATE_TOPIC, output)
        funhouse.peripherals.led = False

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

def timeleft():
    seconds = int(timestamps["last_pir"] + MOTION_TIMEOUT - time.monotonic())
    if outlet.value and seconds >= 0:
        minutes = seconds // 60
        seconds -= minutes * 60
        return "{:01}:{:02}".format(minutes, seconds)
    return "Off"

# Set Initial States
funhouse = FunHouse(default_bg=0x0F0F00)
funhouse.peripherals.dotstars.fill(0)
outlet = digitalio.DigitalInOut(board.A0)
outlet.direction = digitalio.Direction.OUTPUT
funhouse.display.root_group = CIRCUITPYTHON_TERMINAL
funhouse.add_text(
    text="Timeout Left:",
    text_position=(20, 60),
    text_color=0xFF0000,
    text_font="fonts/Arial-Bold-24.pcf",
)
countdown_label = funhouse.add_text(
    text_position=(120, 100),
    text_anchor_point=(0.5, 0.5),
    text_color=0xFFFF00,
    text_font="fonts/Arial-Bold-24.pcf",
)
funhouse.display.root_group = funhouse.splash

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

# Initialize a new MQTT Client object
if USE_MQTT:
    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()
set_outlet_state(False)

while True:
```

```
    if funhouse.peripherals.pir_sensor:
        timestamps["last_pir"] = time.monotonic()
        if not outlet.value:
            set_outlet_state(True)
    if outlet.value and time.monotonic() >= timestamps["last_pir"] + MOTION_TIMEOUT:
        set_outlet_state(False)
    funhouse.set_text(timeleft(), countdown_label)
    # Check any topics we are subscribed to
    if USE_MQTT:
        funhouse.network.mqtt_loop(0.5)
```

Copy these over to the **CIRCUITPY** drive for your FunHouse board in the root directory along with your **secrets.py** file. The files on your board should look like this:



Code Walkthrough

Now to cover the code in sections. First are library imports. This includes the `FunHouse` library, the `Circle` to display if the board is connected, `time` for checking in intervals, and finally `board` and `digitalio` for controlling the outlet itself.

```
import os
import time
import board
import digitalio
from adafruit_display_shapes.circle import Circle
from adafruit_funhouse import FunHouse
```

Next up are the MQTT topics, `OUTLET_STATE_TOPIC` and `OUTLET_COMMAND_TOPIC`. If you're not connecting to MQTT, you can leave these alone. The script will subscribe to the **command topic** to listen for any commands to turn the outlet on or off and will publish to the **state topic** to let Home Assistant know the state of the outlet.

`MOTION_TIMEOUT` is the amount of time in seconds for the FunHouse to wait before turning off the outlet. Any movement that triggers the PIR sensor will reset the timeout to this value.

`USE_MQTT` should be set to `False` if you do not plan on using Home Assistant.

```
OUTLET_STATE_TOPIC = "funhouse/outlet/state"
OUTLET_COMMAND_TOPIC = "funhouse/outlet/set"
MOTION_TIMEOUT = 300  # Timeout in seconds
USE_MQTT = True
```

In order to allow changing a value inside of a function and avoid using the `global` keyword, a dict key is used. This trick allows changing the value of a variable without changing the memory location of that variable, so global isn't needed.

```
# Use dict to avoid reassigning the variable
timestamps = {
    "last_pir": None
}
```

In this section, the script to set the outlet state is defined. It does a few things beside just turning the outlet on or off. It changes the color of the DotStars, changes the time left on the clock to either 0 or the value `MOTION_TIMEOUT` depending on whether it is being set off or on. It also publishes the state of the outlet to MQTT if it's connected.

```
def set_outlet_state(value):
    if value:
        funhouse.peripherals.dotstars.fill(0x00FF00)
        timestamps["last_pir"] = time.monotonic()
    else:
        funhouse.peripherals.dotstars.fill(0xFF0000)
        timestamps["last_pir"] = time.monotonic() - MOTION_TIMEOUT

    outlet.value = value
    publish_outlet_state()
```

This function will publish the current state of the outlet to MQTT, and thus Home Assistant. It will only do so if `USE_MQTT` is set to `True`. The output in this case is a raw **on** or `off` string value that is published to the `OUTLET_STATE_TOPIC`.

```
def publish_outlet_state():
    if USE_MQTT:
        funhouse.peripherals.led = True
        output = "on" if outlet.value else "off"
        # Publish the Dotstar State
        print("Publishing to {}".format(OUTLET_STATE_TOPIC))
        funhouse.network.mqtt_publish(OUTLET_STATE_TOPIC, output)
        funhouse.peripherals.led = False
```

The next few functions are used for changing the circle to red or green depending on the connection status and subscribing to the `OUTLET_COMMAND_TOPIC`.

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

The code in the `timeleft()` function is meant to return either the amount of time left formatted in minutes and seconds or return the string `Off` if there is no time left.

```
def timeleft():
    seconds = int(timestamps["last_pir"] + MOTION_TIMEOUT - time.monotonic())
    if outlet.value and seconds &gt;= 0:
        minutes = seconds // 60
        seconds -= minutes * 60
        return "{:01}:{:02}".format(minutes, seconds)
    return "Off"
```

The next bit of code creates a few of the variables with their initial states, including the `funhouse` object, the outlet Digital IO, and creates and draws the text labels. The DotStar LEDs are set to off and are lit up red once initialization is done to indicate that motion sensing will now work, but it is off.

```
# Set Initial States
funhouse = FunHouse(default_bg=0x0F0F00)
funhouse.peripherals.dotstars.fill(0)
outlet = digitalio.DigitalInOut(board.A0)
outlet.direction = digitalio.Direction.OUTPUT
funhouse.display.root_group = CIRCUITPYTHON_TERMINAL
funhouse.add_text(
    text="Timeout Left:",
    text_position=(20, 60),
    text_color=0xFF0000,
    text_font="fonts/Arial-Bold-24.pcf",
)
countdown_label = funhouse.add_text(
    text_position=(120, 100),
    text_anchor_point=(0.5, 0.5),
    text_color=0xFFFF00,
    text_font="fonts/Arial-Bold-24.pcf",
)
funhouse.display.root_group = funhouse.splash
```

This section initializes MQTT using the secrets if `USE_MQTT` is set to `True`, and sets up the handler functions that were defined earlier, and connects. Once that is through, the initial outlet state is set to `False`, which sets the DotStars red. The `os.getenv()` function is used to get settings from settings.toml.

```
# Initialize a new MQTT Client object
if USE_MQTT:
    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()
set_outlet_state(False)
```

Finally, there is the main loop. In the loop, the PIR sensor is checked. If it detects motion, the time is extended and if it was off, the new Outlet State is set to True, which also publishes to MQTT if it is connected.

It also checks to see if enough time has elapsed and turns off the outlet if it is currently on. The reason for checking the outlet state is to avoid flooding MQTT with messages and only when it changes.

The label is updated with the amount of time left and finally if MQTT is connected, the MQTT loop is run with a half second timeout. This has a default timeout of 1 second, but with the other tasks in the loop, it can take more than that. This results in the display timer appearing to skip seconds, when it just not updating often enough.

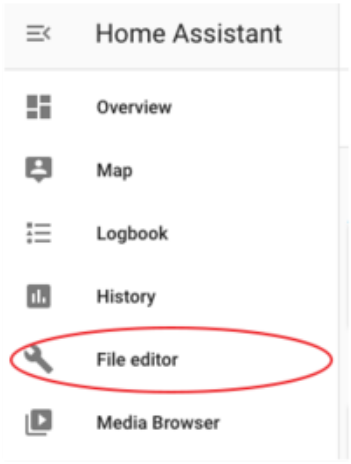
```
while True:
    if funhouse.peripherals.pir_sensor:
        timestamps["last_pir"] = time.monotonic()
        if not outlet.value:
            set_outlet_state(True)
    if outlet.value and time.monotonic() &gt;= timestamps["last_pir"] +
MOTION_TIMEOUT:
        set_outlet_state(False)
    funhouse.set_text(timeleft(), countdown_label)
    # Check any topics we are subscribed to
    if USE_MQTT:
        funhouse.network.mqtt_loop(0.5)
```

Home Assistant Configuration

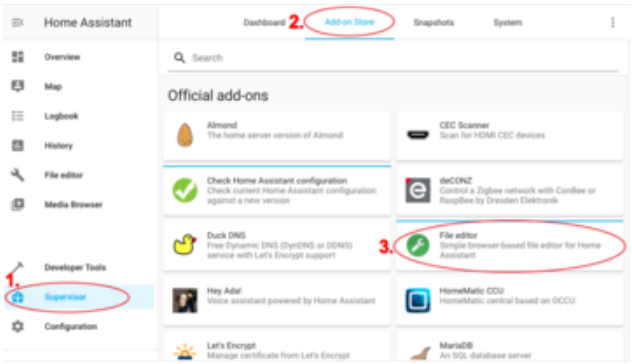
Configuring Home Assistant is purely optional as this project will work great even without it. This guide assumes you already have a working and running Home Assistant server. If you don't, be sure to visit our Set up Home Assistant with a Raspberry Pi (https://adafru.it/lbd) guide first.

Check Your Add-Ons

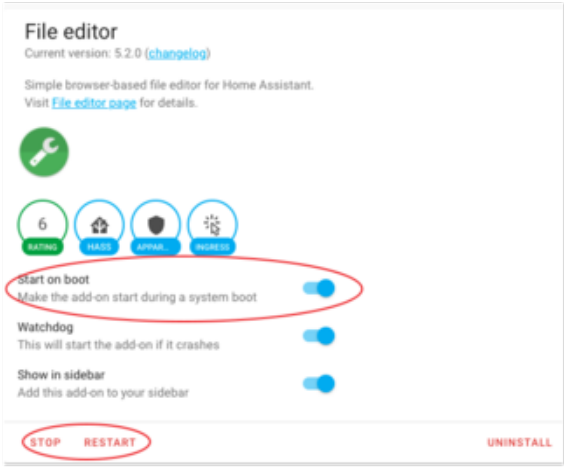
Start out by logging in and opening up your Home Assistant dashboard and checking that the File editor is installed.



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.

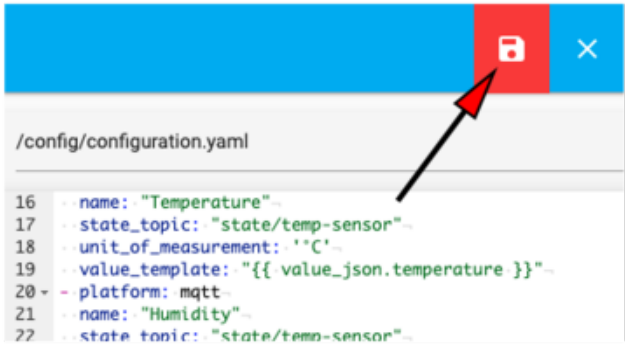


If you already have it, but it's just not showing up, be sure it is started and the option to show in the sidebar is selected.

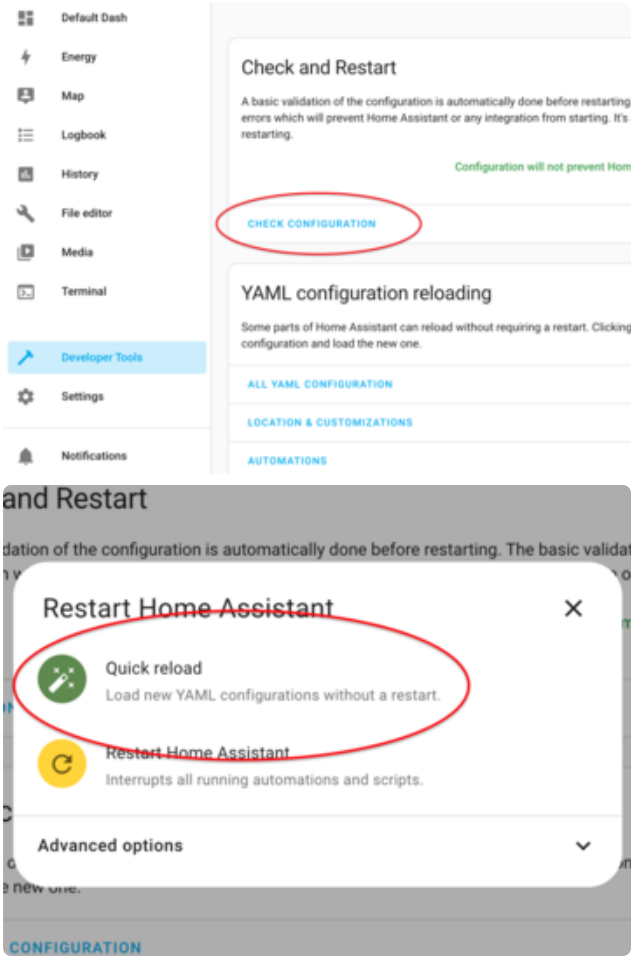
Creating a Switched Outlet

In order to control the switched outlet with Home Assistant, you'll want to add the following code to your configuration.

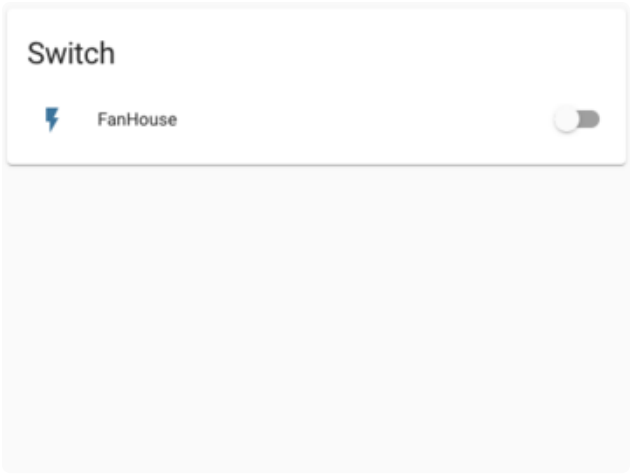
```
mqtt:
  switch:
  - name: "FanHouse"
    unique_id: "funhouse_outlet"
    command_topic: "funhouse/outlet/set"
    state_topic: "funhouse/outlet/state"
    payload_on: "on"
    payload_off: "off"
```



Click the save button at the top.



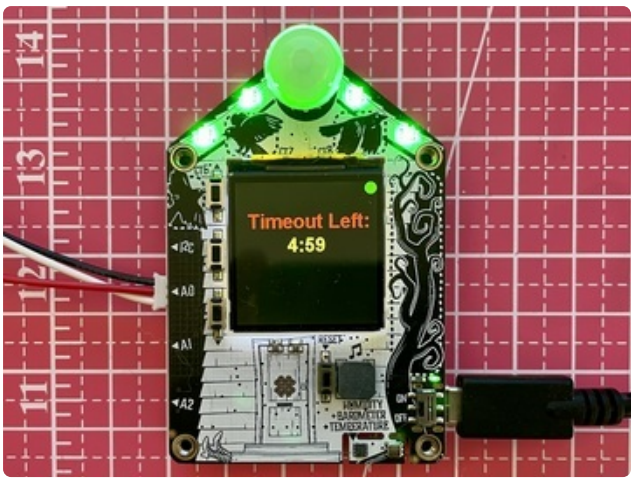
From the **Developer Tools** menu, you can check that the configuration is valid and click on **Restart** to load the configuration changes you made. You can just click **Quick reload** to reload any changes you made.



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 switch card to the dashboard.

The **FanHouse** switch should appear under your switches.



Turning the switch on should turn your FunHouse on and set the timer to the timeout value and turning it off should set the timer to zero and turn the funhouse off.

Troubleshooting

First, make sure you set `MQTT = True` in the CircuitPython code. It will not send or respond to MQTT commands otherwise.

If you see the icons, but there is no data, it is easiest to start by checking the MQTT messages. Adafruit has 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.

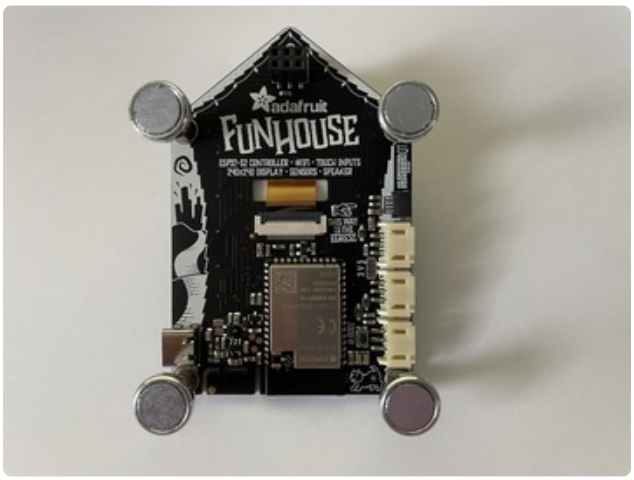
Go ahead and configure a username and password to match your MQTT server and connect. Under **subscribe**, you can subscribe to the `#` topic to get all messages.

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

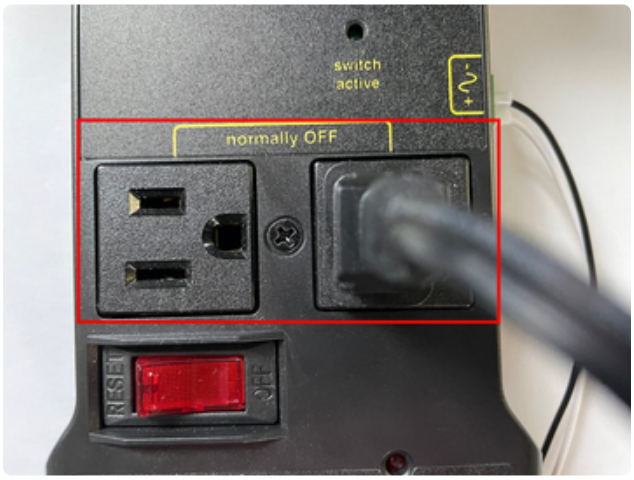
If you don't see any messages, you will want to follow the debugging section on the **Code the Sensor** page.

Motion Sensor Usage

Using the motion sensor is pretty easy. You just make sure everything is on and motion causes the outlet to turn on and begin counting down. If no motion is detected, it will turn off, otherwise the timer will reset.

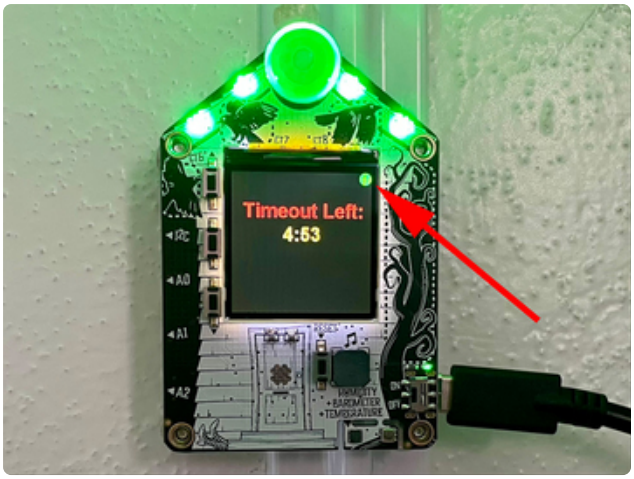


Start by mounting your FunHouse motion sensor in a location where you want the motion sensing to take place. A set of magnetic feet can be really useful for this.

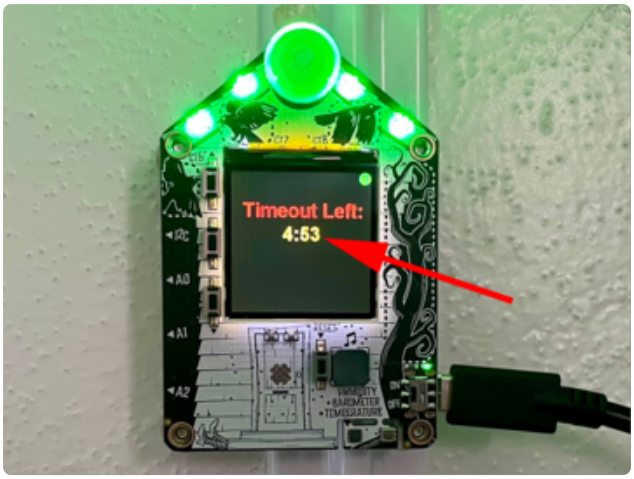


Plug in an appliance such as a fan into one of the outlets labeled **normally OFF**.

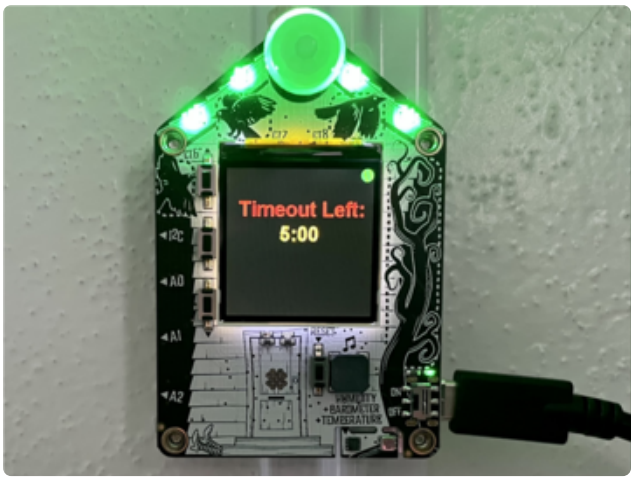
Make sure the Power Switch is in the On position for both the outlet and the appliance. The Power LED should be lit.



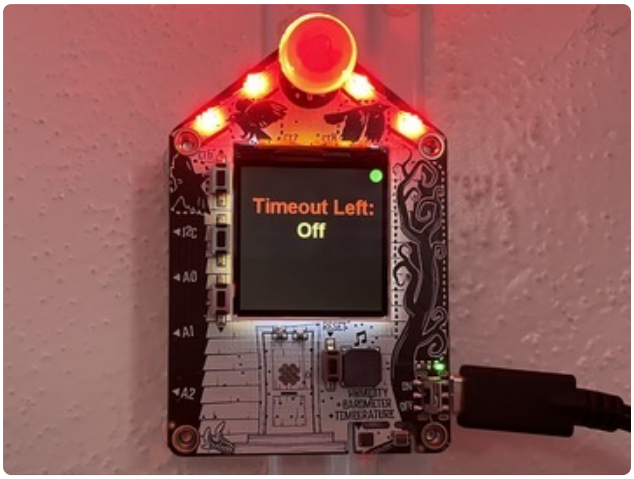
If you connected using Home Assistant, wait until it is connected and the circle turns green.



When the FunHouse senses motion, the lights should light up green and a timer should start counting down as soon as it stops sensing movement.



If you move around, the timer will reset to the full time.



Once the timer reaches 0, the DotStars should turn red and the outlet should turn off.

If you would like to test your setup without standing still for 5 minutes, you can change the timeout to a lower value.