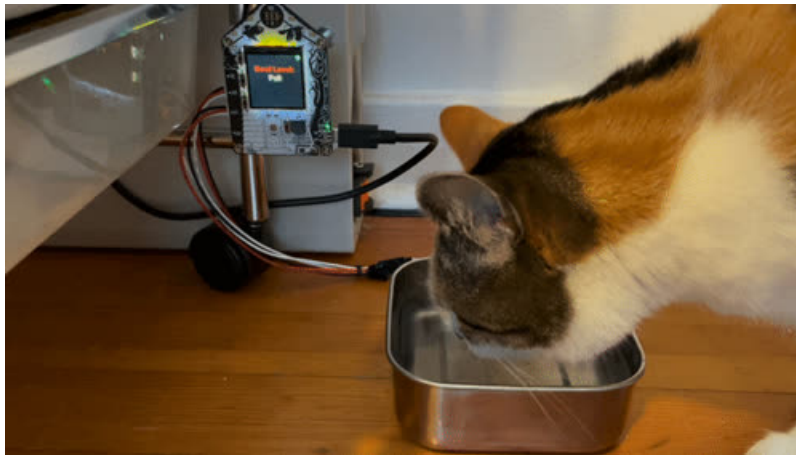




## Pet Bowl Water Level Sensing

Created by Melissa LeBlanc-Williams

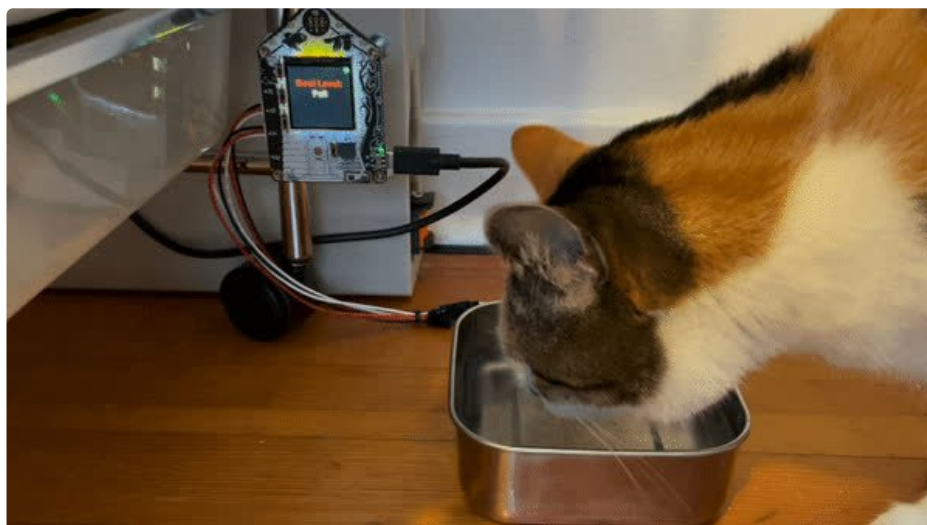


Last updated on 2021-06-07 04:43:43 PM EDT

## Guide Contents

Guide Contents	2
Overview	3
Parts	3
Wiring	5
Attaching the Sensor	5
CircuitPython	7
Set Up CircuitPython	7
Option 1 - Load with UF2 Bootloader	7
Try Launching UF2 Bootloader	8
Option 2 - Use Chrome Browser To Upload BIN file	9
Option 3 - Use esptool to load BIN file	10
CircuitPython Internet Test	11
Secrets File	11
Connect to WiFi	12
Coding the Water Sensor	17
MQTT Secrets Settings	17
Code Walkthrough	18
Home Assistant Configuration	22
Check Your Add-Ons	22
Adding the Sensor	23
Adding Notifications	24
Troubleshooting	26

# Overview



With the FunHouse and Home Assistant, you can automatically be alerted if your pet's bowl gets too low. For this project we are using the Simple Water Sensor. Even though it is designed to be a digital sensor, it will actually work in analog mode to give you a little more granularity. In this way, you can tell if your pet's water level is good, low, or needs water added.

In order to mitigate the effects of running electricity through water, the sensor is only on for a very short period of time to take a reading and turns back off. This is accomplished by using 2 of the ports on the FunHouse. One supplies the voltage to the sensor through the GPIO line and the other takes the reading.

We'll go over how to get your sensor set up and configure Home Assistant to read the status of the sensor and also send you notifications within the Home Assistant environment.

## Parts

Your browser does not support the video tag.

[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...

\$34.95

In Stock

Add to Cart

Your browser does not support the video tag.

[Simple Water Detection Sensor with Digital Output](#)

Keep wet things wet and dry things dry by detecting when the dry things get wet by accident! This palm-sized cherry-red water sensor is simple and easy to implement in your...

\$1.95

In Stock

Add to Cart

---

You'll need 2 of these STEMMA adapters.

### STEMMA JST PH 3-Pin to Female Socket Cable - 200mm

This cable will let you turn a JST PH 3-pin cable port into 3 individual wires with high-quality 0.1" female header sockets on the end. We're carrying these to match up with...

\$1.25

In Stock

Add to Cart

---

### Servo Extension Cable - 50cm / 19.5" long

Stretch out your servo connections with this flexible servo extension cord. It has a 3 pin shrouded "male" connection to plug your servo into and then, 50cm later, a 3 pin...

\$2.50

In Stock

Add to Cart

---

### Clear Adhesive Squares - 6 pack

UGlu Dashes are perfect for a variety of small projects. These adhesive squares provide a stronger bond to most surfaces and are cleaner and easier to...

\$0.50

In Stock

Add to Cart

---

### 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...

\$2.50

In Stock

Add to Cart

---

### 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...

\$3.95

In Stock

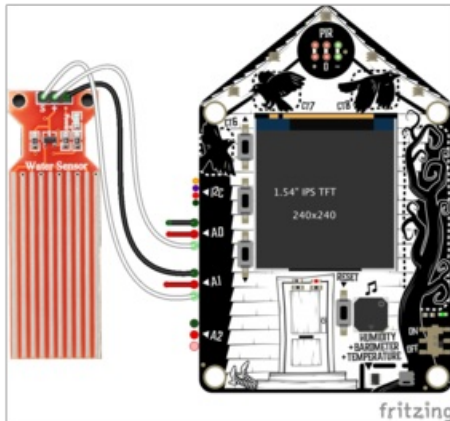
Add to Cart

---

# Wiring

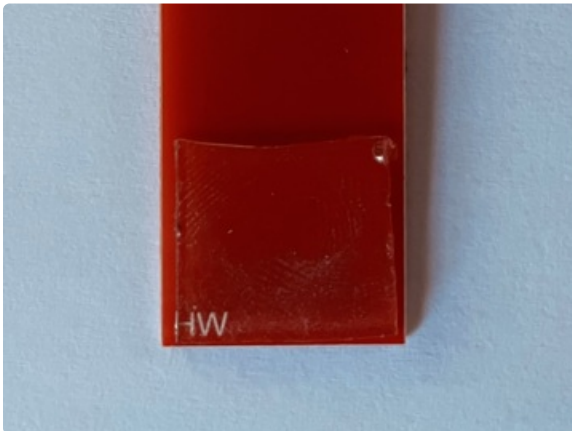
Wiring the FunHouse to the water sensor is pretty straight forward. The only tricky bit we are doing here is we are using 2 JST ports to read the sensor. One of the JST ports is used for reading the sensor and the other port is powering the sensor with its signal wire. This is so the sensor isn't on all the time, which can lead to electrolyzing the traces fairly quickly.

Since this is a water sensor and we're dealing with electronics, we recommend connecting a servo extension cable between the JST cables and the sensor.

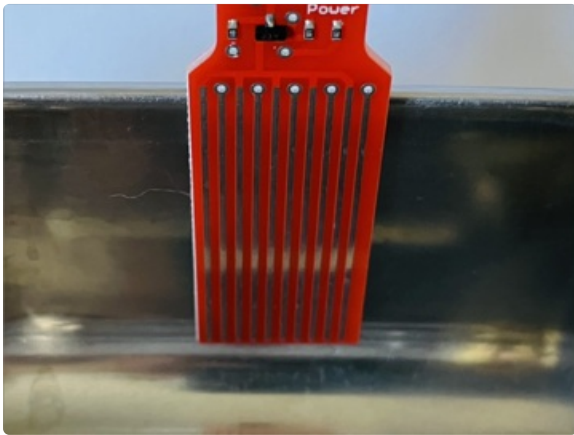


- Connect the **white wire** from the JST connector of **A0** to the positive or + pin on the water sensor.
- Connect the **black wire** from the JST connector of **A1** to the ground or - pin on the water sensor.
- Connect the **white wire** from the JST connector of **A1** to the signal or S pin on the water sensor.
- Leave any remaining wires unconnected.

## Attaching the Sensor

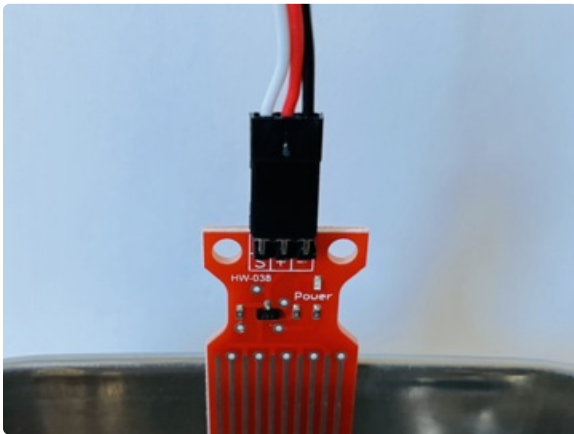


Place some double-sided adhesive to the back side of the sensor.



Attach the sensor to the side of the bowl. If it has trouble sticking due to the curve in the bowl, you can add more adhesive.

You'll want to align the bottom edge around where you would want to be alerted, so if your pet likes it changed more frequently and doesn't drink much, you may want to mount it a bit higher.



Plug the servo extension cable into the sensor as shown in the image.

# CircuitPython

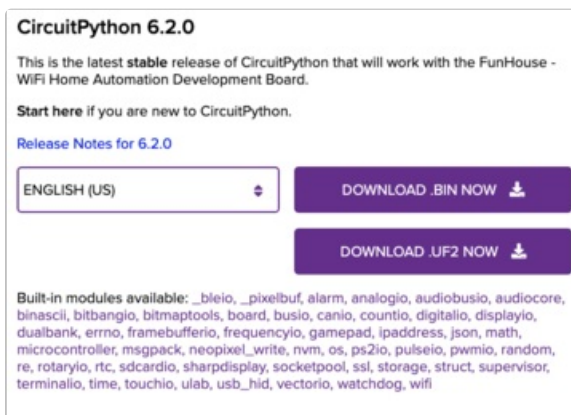
[CircuitPython](https://adafru.it/tB7) (<https://adafru.it/tB7>) is a derivative of [MicroPython](https://adafru.it/f9W) (<https://adafru.it/f9W>) designed to simplify experimentation and education on low-cost microcontrollers. It makes it easier than ever to get prototyping by requiring no upfront desktop 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>

<https://adafru.it/RKB>



**CircuitPython 6.2.0**

This is the latest **stable** release of CircuitPython that will work with the FunHouse - WiFi Home Automation Development Board.

Start here if you are new to CircuitPython.

Release Notes for 6.2.0

ENGLISH (US)

DOWNLOAD .BIN NOW

DOWNLOAD .UF2 NOW

Built-in modules available: \_bleio, \_pixelbuf, alarm, analogio, audiobusio, audiocore, binascii, bitbangio, bitmaptools, board, busio, canio, countio, digitalio, displayio, dualbank, ermo, framebufferio, frequencyio, gamepad, ipaddress, json, math, microcontroller, msgpack, neopixel\_write, nvm, os, ps2io, pulseio, pwmio, random, re, rotaryio, rtc, sdcardio, sharpdisplay, socketpool, ssl, storage, struct, supervisor, terminalio, time, touchio, ulab, usb\_hid, vectorio, watchdog, wifi

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

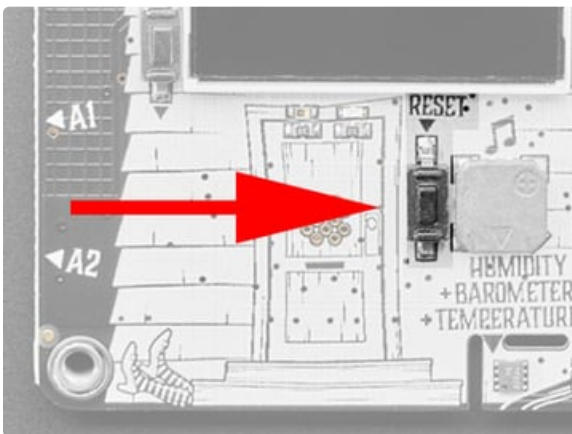
This is by far the easiest way to load CircuitPython. **However it requires your board has the UF2 bootloader installed. Some early boards do not (we hadn't written UF2 yet!) - in which case you can load using the built in ROM bootloader.**

Still, try this first!



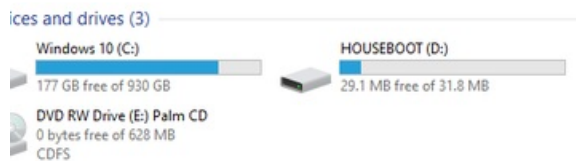
### Try Launching UF2 Bootloader

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



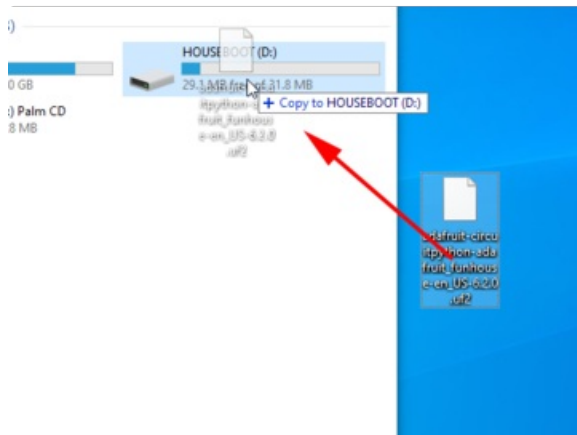
Launch UF2 by **double-clicking** the Reset button (the one next to the USB C port). You may have to try a few times to get the timing right.

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



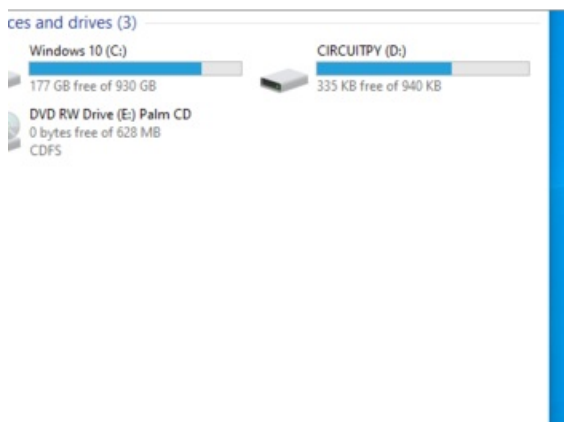
If the UF2 bootloader is installed, you will see a new disk drive appear called **HOUSEBOOT**





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 \(https://adafru.it/RLc\)](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!

```
5967 kattni@roborepe:~$ python ./esptool.py --port /dev/cu.usbmodem01 --after-no_reset
write_flash 0x0 ~/adafruit-circuitpython-adafruit_metro_esp32-en_US-20201103-5a07925.bin
esptool.py v3.0-dev
Serial port /dev/cu.usbmodem01
Connecting...
Detecting chip type... ESP32-S2
Chip is ESP32-S2
Features: WiFi, ADC and temperature sensor calibration in BLK2 of efuse
Crystal is 40MHz
MAC: 7c:df:a1:00:4a:a2
Uploading stub...
Running stub...
Stub running...
Configuring flash size...
Compressed 1305184 bytes to 844014...
Wrote 1305184 bytes (844014 compressed) at 0x00000000 in 11.9 seconds (effective 878.2 kbit/s)...
Hash of data verified.
Leaving...
Staying in bootloader.
```

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.

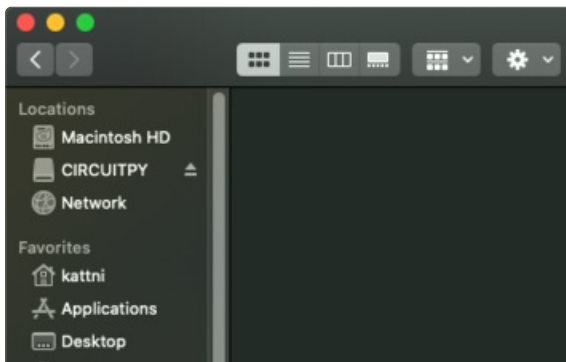
In the final command to write a binary file to the board, replace the port with your port, and replace "firmware.bin" with the the file you downloaded above.

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

Once you have CircuitPython installed and the minimum libraries installed we can get your board connected to the Internet.

To get connected, you will need to start by creating a **secrets.py** file.

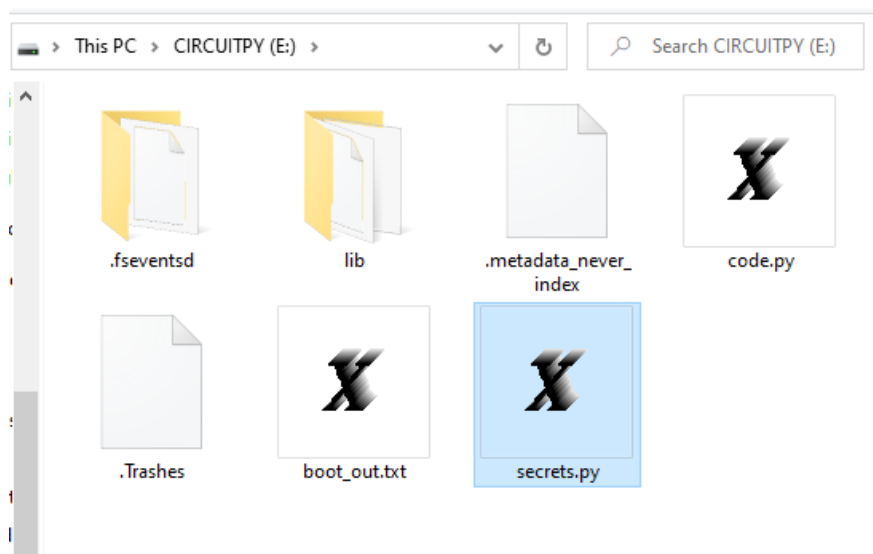
## Secrets File

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 **secrets.py** file, that is in your **CIRCUITPY** drive, to hold secret/private/custom data. That way you can share your main project without worrying about accidentally sharing private stuff.

Your **secrets.py** file should look like this:

```
# This file is where you keep secret settings, passwords, and tokens!  
# If you put them in the code you risk committing that info or sharing it  
  
secrets = {  
    'ssid' : 'home_wifi_network',  
    'password' : 'wifi_password',  
    'aio_username' : 'my_adafruit_io_username',  
    'aio_key' : 'my_adafruit_io_key',  
    'timezone' : "America/New_York", # http://worldtimeapi.org/timezones  
}
```

Copy and paste that text/code into a file called **secrets.py** and save it to your **CIRCUITPY** folder like so:



Inside is a python dictionary named `secrets` with a line for each entry. Each entry has an entry name (say `'ssid'`) and then a colon to separate it from the entry key `'home ssid'` and finally a comma ,

**At a minimum you'll need to adjust the `ssid` and `password` for your local WiFi setup 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, just cause its called secrets doesn't mean you can't have general customization data in there!

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 `secrets.py` - keep that out of GitHub, Discord or other project-sharing sites.

Don't share your secrets.py file, it has your passwords and API keys in it!

## Connect to WiFi

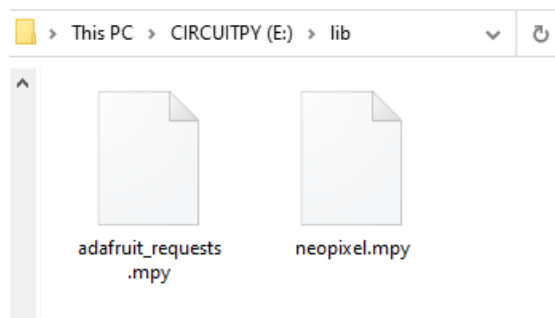
OK now you have your secrets setup - you can connect to the Internet using the Requests module.

First make sure you are running the [latest version of Adafruit CircuitPython](https://adafru.it/Amd) (<https://adafru.it/Amd>) for your board.

Next you'll need to install the necessary libraries to use the hardware--carefully follow the steps to find and install these libraries from [Adafruit's CircuitPython library bundle](https://adafru.it/zdx) (<https://adafru.it/zdx>). Our introduction guide has [a great page on how to install the library bundle](https://adafru.it/ABU) (<https://adafru.it/ABU>).

- `adafruit_requests`
- `neopixel`

**Before continuing make sure your board's `CIRCUITPY/lib` folder or root filesystem has the above files copied over.**



Once that's done, load up the following example using Mu or your favorite editor:

```
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"

# Get wifi details and more from a secrets.py file
try:
    from secrets import secrets
except ImportError:
    print("WiFi secrets are kept in secrets.py, please add them there!")
    raise

print("ESP32-S2 WebClient Test")

print("My MAC addr:", [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("Connecting to %s"%secrets["ssid"])
wifi.radio.connect(secrets["ssid"], secrets["password"])
print("Connected to %s!"%secrets["ssid"])
print("My IP address is", wifi.radio.ipv4_address)

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

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

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

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

print()
```

```

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

print("done")

```

And save it to your board. Make sure the file is named `code.py`.

Open up your REPL, 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
  disconnectededer      RSSI: -86      Channel: 1
  SKJFios-ZV007         RSSI: -83      Channel: 11
  Fios-QIVUQ            RSSI: -83      Channel: 11
  Fios-ZV007           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-S2's MAC address.

```

print("My MAC addr:", [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("Avaliable 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 `secrets.py` file, prints out its local IP address, and attempts to ping google.com to check its network connectivity.

```
print("Connecting to %s"%secrets["ssid"])
wifi.radio.connect(secrets["ssid"], secrets["password"])
print(print("Connected to %s!"%secrets["ssid"]))
print("My IP address is", wifi.radio.ipv4_address)

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

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 new instance of the [requests](https://adafru.it/E9o) (<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 http s url for SSL connectivity.

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

OK you now have your ESP32-S2 board set up with a proper **secrets.py** file and can connect over the Internet. If not, check that your **secrets.py** file has the right ssid and password and retrace your steps until you get the Internet connectivity working!



# Coding the Water Sensor

Let's start out with the code that goes onto the FunHouse.

## MQTT Secrets Settings

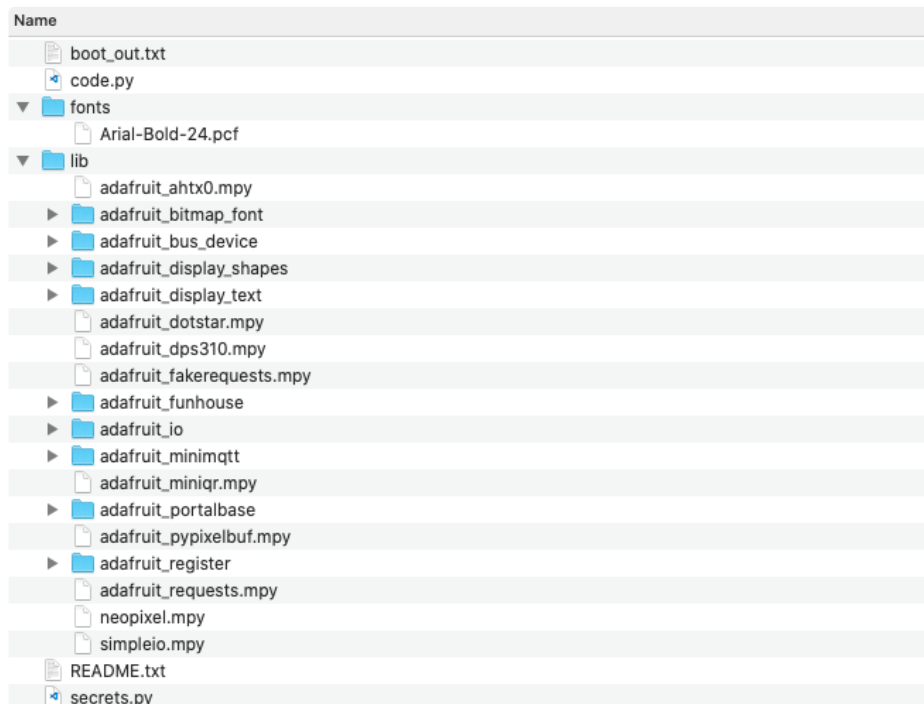
Since the code publishes directly to the MQTT server, there are a few more `secrets.py` file settings that the code expects to find. If your MQTT server has no username and password, you can change the value to `None`, however in general, the Home Assistant MQTT broker is set up to be password protected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

Temporarily unable to load content:

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`, `analogio`, and `digitalio` for enabling and reading the sensor itself.

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

Next up is the MQTT topic, `BOWL_STATE_TOPIC`. The script will publish to the `state` topic if the reading has changed enough to let Home Assistant know the new state of the bowl.

`LOW_VALUE` is the raw analog reading for the sensor when the water is low. Anything above this value is considered full. `EMPTY_VALUE` is the raw analog reading for the sensor when the water is empty. Anything at or below this value is considered empty.

`UPDATE_INTERVAL` is the amount of time to wait in seconds for the FunHouse to check readings. By default this is once every half hour, which should be sufficient for something that changes very little over the course of a day.

```
BOWL_STATE_TOPIC = "funhouse/catbowl/state"
LOW_VALUE = 4000
EMPTY_VALUE = 2000
UPDATE_INTERVAL = 1800 # Every 30 minutes
```

Next up the script attempts to import `secrets.py` to get the MQTT login information.

```
try:
    from secrets import secrets
except ImportError:
    print("WiFi secrets are kept in secrets.py, please add them there!")
    raise
```

This section contains a dict of different states. This is a convenient way to change the text that is shown on the display for each of these states.

```
# Text labels for the Display
states = {
    "empty": "Add Water",
    "low": "Low",
    "full": "Full",
}
```

This function will publish the current state of the bowl to MQTT, and thus Home Assistant. The output in this case is a raw text string value corresponding to one of the keys in the above dict that is published to the `BOWL_STATE_TOPIC`.

```
def publish_bowl_state(bowl_state):
    funhouse.peripherals.led = True
    # Publish the Bowl Level State
    print("Publishing to {}".format(BOWL_STATE_TOPIC))
    funhouse.network.mqtt_publish(BOWL_STATE_TOPIC, bowl_state)
    funhouse.peripherals.led = False
```

The next couple of functions are used for changing the circle to red or green depending on the connection status.

```
def connected(client, userdata, result, payload):
    status.fill = 0x00FF00
    status.outline = 0x008800

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

The code in the `get_bowl_reading()` function is meant to briefly enable the sensor, take a reading, and then shut it off. This is done to reduce electrolysis that may occur with the metal being in the water.

The code in the `get_bowl_state()` function just looks at the `level` value given to it, compares it to the thresholds that were set above and returns the state of the bowl.

The code in the `bowl_level_display()` function checks if the select button is held down and either returns a raw value or the text label for the current state.

```

def get_bowl_reading():
    water_enable.value = True
    level = water_level_sensor.value
    water_enable.value = False
    return level

def get_bowl_state(level):
    if level <= EMPTY_VALUE:
        return "empty"
    elif level <= LOW_VALUE:
        return "low"
    return "full"

def bowl_level_display(water_level):
    if funhouse.peripherals.button_sel:
        return water_level
    return states[get_bowl_state(water_level)]

```

The next bit of code creates a few of the variables with their initial states, including the `funhouse` object, the `water_enable` Digital IO, the `water_level_sensor` Analog IO and creates and draws the text labels.

```

# Set Initial States
funhouse = FunHouse(default_bg=0x0F0F00)
funhouse.peripherals.dotstars.fill(0)
water_enable = digitalio.DigitalInOut(board.A0)
water_enable.switch_to_output()
water_level_sensor = analogio.AnalogIn(board.A1)
funhouse.display.show(None)
funhouse.add_text(
    text="Bowl Level:",
    text_position=(120, 60),
    text_anchor_point=(0.5, 0.5),
    text_color=0xFF0000,
    text_font="fonts/Arial-Bold-24.pcf",
)
level_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.show(funhouse.splash)

```

This section initializes MQTT using the secrets, sets up the handler functions that were defined earlier, and connects. The `last_reading_timestamp` and `last_bowl_state` are then initialized and set to `None`.

```

# Initialize a new MQTT Client object
funhouse.network.init_mqtt(
    secrets["mqtt_broker"],
    secrets["mqtt_port"],
    secrets["mqtt_username"],
    secrets["mqtt_password"],
)
funhouse.network.on_mqtt_connect = connected
funhouse.network.on_mqtt_disconnect = disconnected

print("Attempting to connect to {}".format(secrets["mqtt_broker"]))
funhouse.network.mqtt_connect()

last_reading_timestamp = None
last_bowl_state = None

```

Finally, there is the main loop. In the loop, it keeps looping and waits until the `UPDATE_INTERVAL` has passed. Once it has, it gets a reading and stores it in a variable so the sensor only has to be read once. It sets the display to show the appropriate message depending on the reading.

It next checks the state of the bowl and if the value differs from the last reading, it publishes the result to MQTT.

```

while True:
    if last_reading_timestamp is None or time.monotonic() > last_reading_timestamp +
UPDATE_INTERVAL:
        # Take Reading
        water_level = get_bowl_reading()
        # Update Display
        funhouse.set_text(bowl_level_display(water_level), level_label)
        # If changed, publish new result
        bowl_state = get_bowl_state(water_level)
        if bowl_state != last_bowl_state:
            publish_bowl_state(bowl_state)
            last_bowl_state = bowl_state
        last_reading_timestamp = time.monotonic()

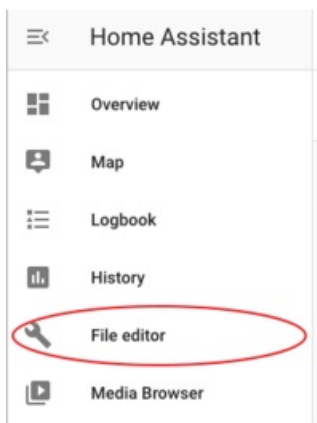
```

# Home Assistant Configuration

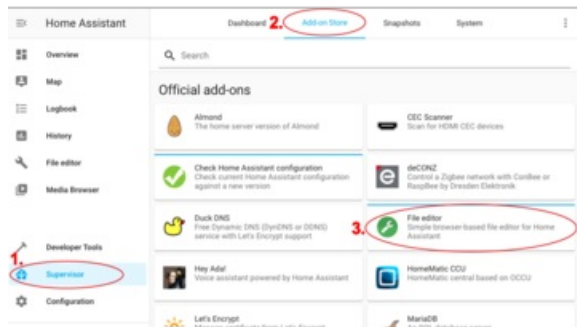
This guide assumes you already have a working and running Home Assistant server. If you don't, be sure to visit our [Set up Home Assistant with a Raspberry Pi \(https://adafru.it/lbd\)](https://adafru.it/lbd) guide first.

## Check Your Add-Ons

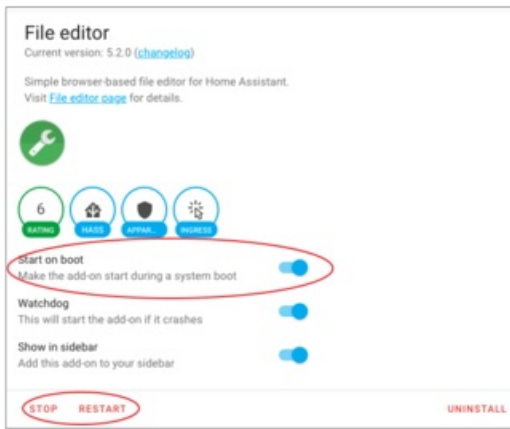
Start out by logging in and opening up your Home Assistant dashboard and checking that the File editor is installed.



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 Supervisor → Add-on Store → File editor and go through the installation procedure.



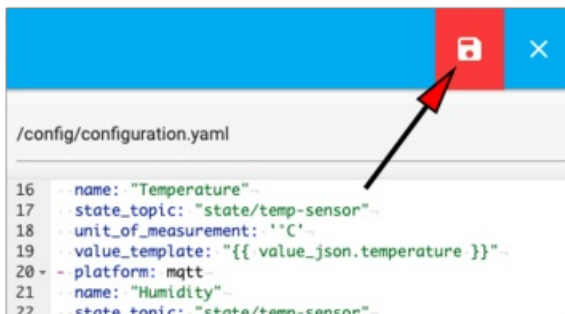
If you already have it, but it's just not showing up, be sure it is started and the option to show in the sidebar is selected.

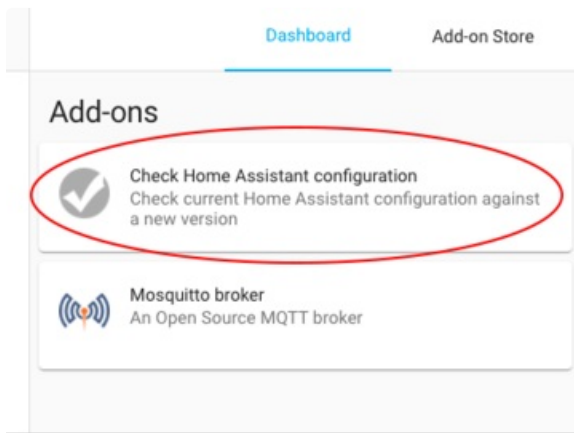
## Adding the Sensor

In order to add the Water Level Sensor to Home Assistant, you'll want to add the following code to your configuration. This will create an MQTT sensor that listens on the `funhouse/catbowl/state` topic. It will translate the value it receives to one of the state labels. Feel free to update the states to your liking.

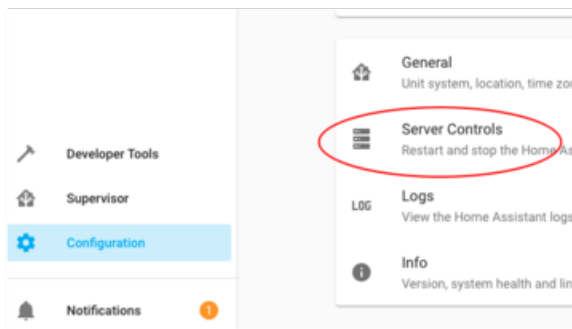
```
sensor funhouse_catwater:
  - platform: mqtt
    name: "Cat Water Bowl"
    state_topic: "funhouse/catbowl/state"
    icon: mdi:bowl
    value_template: >-
      {% set states = {
        'empty' : 'Add Water',
        'low' : 'Water Low',
        'full' : 'Full'
      } %}
      {{ states[value] if value in states else 'Unknown' }}
```

Click the save button at the top.

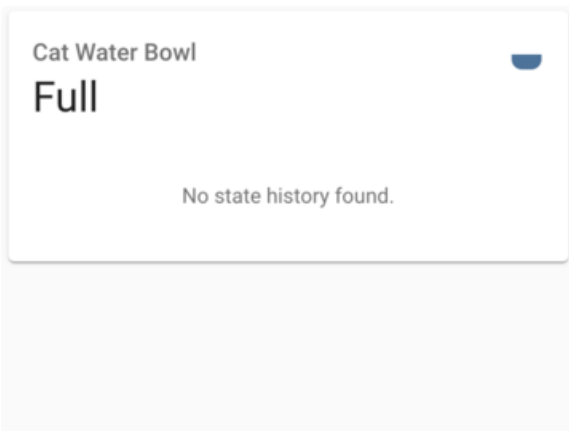




If you have the **Check Home Assistant configuration** tool installed, now would be a good time to run it. It takes several minutes to run and you can check the log tab to see the results.



From the **Configuration** menu, choose **Server Controls**. Here you can check that the configuration is valid and click on **Restart** to load the configuration 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.

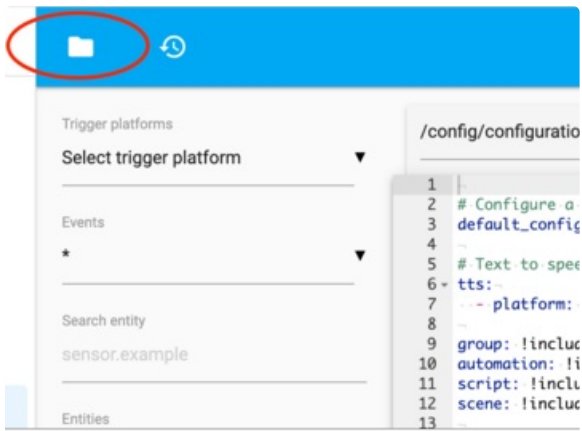
Otherwise, you may need to manually add a **Sensor** card to the dashboard.

The **Cat Water Bowl** sensor should appear under your sensors.

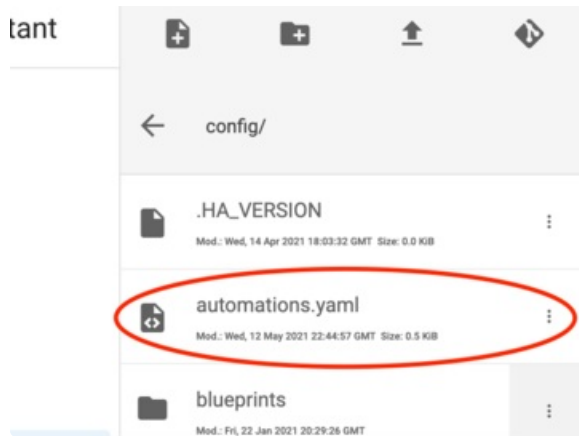
## Adding Notifications

This project wouldn't be complete without having Home Assistant notify you of when the bowl gets low or empty. Home Assistant uses Persistent Notifications as its main way of notifying you, which is what we'll be showing you how to add.





Start by going to the File Editor again and click the folder icon at the top.



Open the `automations.yaml` file.

Add the following code. This will create 2 notification triggers. The first is when the state goes from **full** to **low**. The second is when it goes from any state to **empty**. Feel free to customize the messages.

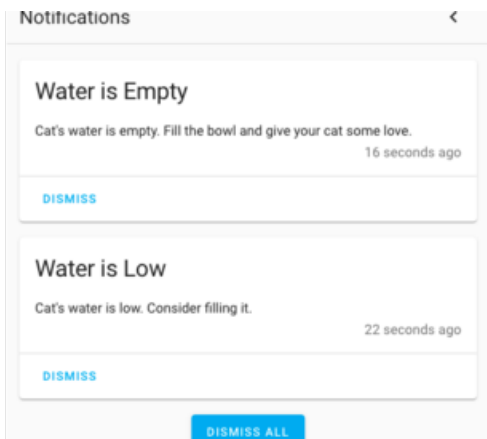
```

- alias: "Cat Water Low"
  trigger:
  - entity_id: sensor.cat_water_bowl
    from: "full"
    platform: state
    to: "low"
  action:
    service: persistent_notification.create
    data:
      title: "Water is Low"
      message: "Cat's water is low. Consider filling it."

- alias: "Cat Water Empty"
  trigger:
  - entity_id: sensor.cat_water_bowl
    platform: state
    to: "empty"
  action:
    service: persistent_notification.create
    data:
      title: "Water is Empty"
      message: "Cat's water is empty. Fill the bowl and give your cat some love."

```

Once that's added, go ahead and check the configuration and restart the server as you did before. If you made changes to the water bowl code above, you may need to update this code to reflect those changes.



To test alerts, you may need to temporarily increase the `UPDATE_INTERVAL` in your sensor code so it checks more frequently.

## Troubleshooting

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 \(https://adafru.it/kID\)](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 **Coding the Water Sensor** page.

