# Table of Contents

## Overview
- Parts List

## Sensor Setup
- Wiring
- CircuitPython Installation
- Library Installation

## CircuitPython Internet Test
- The settings.toml File

## Code the Sensor
- MQTT Secrets Settings
- Full Code Listing
- How the Code Works
- Debugging the Sensor
- Using Other Sensors

## Home Assistant Configuration
- Testing the Sensor
- Troubleshooting
- Going Further
This guide will cover how to add a temperature/humidity sensor to Home Assistant using MQTT and Circuit Python. It is assumed that you already have a Home Assistant server up and running and that you have already installed the Mosquitto broker Add-on.

No soldering is required for this project, its plug-and-play thanks to the Metro ESP32-S2's STEMMA QT port and the many QT-friendly sensors we've got. Simply plug in the sensor and run the code to make customizable projects with sensing and Home Assistant!
This guide shows you how to configure the SHTC3 Humidity and Temperature sensor for Home Assistant, though other sensors can be used with just a little modification.

This particular sensor has a wide range of temperatures, excellent accuracy, has low power consumption and has a Stemma QT port for plug and play without soldering.

You can read more about this sensor in our Adafruit Sensirion SHTC3 - Temperature & Humidity Sensor Breakout guide.

For the main board, an ESP32-S2 is used because of the great combination of the built-in WiFi and the StemmaQT, but if you have another board that can connect to WiFi, runs CircuitPython, and can have your sensor wired up to it, that will work as well.

Parts List

Adafruit Metro ESP32-S2

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

https://www.adafruit.com/product/4775
Adafruit Sensirion SHTC3 Temperature & Humidity Sensor
Sensirion Temperature/Humidity sensors are some of the finest & highest-accuracy devices you can get. And finally, we have some that have a true I2C interface for easy...
https://www.adafruit.com/product/4636

STEMMA QT / Qwiic JST SH 4-pin Cable - 100mm Long
This 4-wire cable is a little over 100mm / 4“ long and fitted with JST-SH female 4-pin connectors on both ends. Compared with the chunkier JST-PH these are 1mm pitch instead of...
https://www.adafruit.com/product/4210

Optionally add a battery for mobility or backup protection.

Lithium Ion Polymer Battery - 3.7v 1200mAh
Lithium-ion polymer (also known as 'lipo' or 'lipoly') batteries are thin, light, and powerful. The output ranges from 4.2V when completely charged to 3.7V. This...
https://www.adafruit.com/product/258
Sensor Setup

Wiring

ESP32-S2 3V to sensor VIN (red wire)
ESP32-S2 GND to sensor GND (black wire)
ESP32-S2 SCL to sensor SCL (yellow wire)
ESP32-S2 SDA to sensor SDA (blue wire)

CircuitPython Installation

You will need to have CircuitPython installed and running. If you haven't set up CircuitPython on a board with an ESP32-S2 processor, the process is a little bit different than other boards.

The first step involves installing the Bootloader. We recommend starting with the Install UF2 Bootloader page of the ESP32-S2 Metro guide. After you have the bootloader installed, the other step is installing CircuitPython itself.

Library Installation

To use the internet-connectivity built into your ESP32-S2 with CircuitPython, you must first install a number of libraries.

Adafruit CircuitPython Library Bundle

Download the Adafruit CircuitPython Bundle. You can find the latest release here:

Download latest CircuitPython Library Bundle

Download the adafruit-circuitpython-bundle-version-mpy-*.zip bundle zip file, and unzip a folder of the same name. Inside you'll find a lib folder. The entire collection of
libraries is too large to fit on the CIRCUITPY drive. Instead, add each library as you
need it, this will reduce the space usage but you’ll need to put in a little more effort.

You will need MiniMQTT version 5.0.0 or later to work with the ESP32-S2.

At a minimum we recommend the following libraries, in fact we more than
recommend. They’re basically required. So grab them and install them into CIRCUITPY/lib now!

- adafruit_minimqtt - MQTT library required for communicating with the MQTT Server
- adafruit_shtc3.py - Temperature/Humidity Sensor library

CircuitPython Internet Test

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

The first thing you need to do is update your code.py to the following. Click the Downl
oad 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.

```python
# 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	RSSI: %d\tChannel: %d\t\% (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) * 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")

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.

```python
# 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 = (equal 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 ()] 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.

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

```python
print("Available WiFi networks:")
for network in wifi.radio.start_scanning_networks():
    print("\t%s		RSSI: %d	Channel: %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.

```python
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.

```python
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() interface - which makes getting data from the internet really really easy.

```python
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.

```python
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`.

```python
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.

```python
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!

## 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 Secrets Settings

Since the code publishes directly to the MQTT server, there are a few more secret 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.

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

### Full Code Listing

```python
# SPDX-FileCopyrightText: 2021 Melissa LeBlanc-Williams for Adafruit Industries
#
# SPDX-License-Identifier: MIT
===
SHTC3 Temperature/Humidity Sensor Example for
```
import os
import time
import ssl
import json
import alarm
import board
import socketpool
import wifi
import adafruit_minimqtt.adafruit_minimqtt as MQTT
import adafruit_shtc3

PUBLISH_DELAY = 60
MQTT_TOPIC = "state/temp-sensor"
USE_DEEP_SLEEP = True

# Connect to the Sensor
i2c = board.I2C()  # uses board.SCL and board.SDA
# i2c = board.STEMMA_I2C()  # For using the built-in STEMMA QT connector on a
# microcontroller
sht = adafruit_shtc3.SHTC3(i2c)

wifi.radio.connect(os.getenv("CIRCUITPY_WIFI_SSID"),
os.getenv("CIRCUITPY_WIFI_PASSWORD"))
print("Connected to %s!" % os.getenv("CIRCUITPY_WIFI_SSID"))

# Create a socket pool
pool = socketpool.SocketPool(wifi.radio)

# Set up a MiniMQTT Client
mqtt_client = MQTT.MQTT(
broker=os.getenv("MQTT_BROKER"),
port=os.getenv("MQTT_PORT"),
username=os.getenv("MQTT_USERNAME"),
password=os.getenv("MQTT_PASSWORD"),
socket_pool=pool,
ssl_context=ssl.create_default_context(),
)

print("Attempting to connect to %s" % mqtt_client.broker)
mqtt_client.connect()

while True:
    temperature, relative_humidity = sht.measurements
    output = {
        "temperature": temperature,
        "humidity": relative_humidity,
    }
    print("Publishing to %s" % MQTT_TOPIC)
    mqtt_client.publish(MQTT_TOPIC, json.dumps(output))

    if USE_DEEP_SLEEP:
        mqtt_client.disconnect()
        pause = alarm.time.TimeAlarm(monotonic_time=time.monotonic() +
PUBLISH_DELAY)
        alarm.exit_and_deep_sleep_until_alarms(pause)
    else:
        last_update = time.monotonic()
        while time.monotonic() < last_update + PUBLISH_DELAY:
            mqtt_client.loop()
How the Code Works

First we start with our imports. Many of the imports are ESP32-S2 specific because of the built Wi-Fi functionality, but this list also includes `json`, `adafruit_minimqtt`, and `adafruit_shtc3`, which we'll go over later.

```python
import os
import time
import ssl
import json
import alarm
import board
import socketpool
import wifi
import adafruit_minimqtt.adafruit_minimqtt as MQTT
import adafruit_shtc3
```

In the next section, there are 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 `MQTT_TOPIC` is the topic that is published on the MQTT server. To read more about MQTT Topics, you can check out the MQTT Topics section of our All the Internet of Things Protocols guide.

The `USE_DEEP_SLEEP` setting defines how we want to wait until we publish. If the setting is `True`, it uses Deep Sleep to stop execution, put the board into a low power mode, and then restart the script after a certain amount of time, so it will need a little additional time to reconnect to WiFi. If the setting is `False`, it will just wait the amount of time in `PUBLISH_DELAY` and the publish again. You can read more about the Deep Sleep feature in our Deep Sleep with CircuitPython guide.

If you plan to modify the code to respond to MQTT requests, you will want to have `USE_DEEP_SLEEP` set to false. However, setting up an MQTT subscription will not be covered in this guide because Home Assistant won't be polling for the Temperature/Humidity sensor.

```python
PUBLISH_DELAY = 60
MQTT_TOPIC = "state/temp-sensor"
USE_DEEP_SLEEP = True
```

The next line, we initialize to the sensor by passing in the I2C bus.

```python
# Connect to the Sensor
sht = adafruit_shtc3.SHTC3(board.I2C())
```
Now that the code has secrets, it uses that to connect to the WiFi access point and create a socket pool. Sockets are how CircuitPython establishes communication over the internet.

```python
wifi.radio.connect(secrets["ssid"], secrets["password"])  
print("Connected to %s!" % secrets["ssid"])  

# Create a socket pool  
pool = socketpool.SocketPool(wifi.radio)
```

The MQTT library is initialized next using the settings in the secrets file and the socket pool. Once it is initialized, the code attempts to connect to the MQTT server.

```python
# Set up a MiniMQTT Client  
mqtt_client = MQTT.MQTT(  
    broker=os.getenv("MQTT_BROKER"),  
    port=os.getenv("MQTT_PORT"),  
    username=os.getenv("MQTT_USERNAME"),  
    password=os.getenv("MQTT_PASSWORD"),  
    socket_pool=pool,  
    ssl_context=ssl.create_default_context(),  
)  
print("Attempting to connect to %s" % mqtt_client.broker)  
mqtt_client.connect()
```

Now we get to the main loop. The loop will really only come into play if `USE_DEEP_SLEEEP` is False which we’ll explain in the next section.

First we grab our temperature and humidity from the sensor. Then we create a dict that will hold the structure for our JSON output. We are adding a temperature and humidity settings to the dict. Finally we use the `json.dumps()` function to convert the dict structure to a JSON string.

```python
temperature, relative_humidity = sht.measurements  
output = {  
    "temperature": temperature,  
    "humidity": relative_humidity,  
}  
print("Publishing to %s" % MQTT_TOPIC)  
mqtt_client.publish(MQTT_TOPIC, json.dumps(output))
```

This last section is all about waiting until it is time to publish the temperature and humidity again.

If `USE_DEEP_SLEEP` is `True`, then we create an alarm to tell the program to stop running and restart from the beginning after a certain amount of time.
Otherwise, we'll make use of the loop. The `last_update` is set to `time.monotonic()`, which is a running counter that keeps the relative time and is useful for measuring elapsed time. Then a simple `while` loop is used to wait until it is time to publish again. Inside this loop, the `mqtt_client.loop()` function is called which is useful if you plan on having the code respond to a subscription.

```python
if USE_DEEP_SLEEP:
    mqtt_client.disconnect()
    pause = alarm.time.TimeAlarm(monotonic_time=time.monotonic() + PUBLISH_DELAY)
    alarm.exit_and_deep_sleep_until_alarms(pause)
else:
    last_update = time.monotonic()
    while time.monotonic() < last_update + PUBLISH_DELAY:
        mqtt_client.loop()
```

**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](#) with CircuitPython. Once you are connected, it can help with any troubleshooting.

**Using Other Sensors**

If you would like to use other Temperature and Humidity sensors, then you can modify the code to do so. You'll need to change the import line to your sensor and a simple test example is usually included with each library for the sensor that allows you to get the values that you need.

Once you have the values, you can just plug them into the dict and the rest should just work.

---

**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](#) guide first.

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.
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 value you used in the sensor code.

```yaml
mqtt:
  sensor:
    - name: "Temperature"
      state_topic: "state/temp-sensor"
      unit_of_measurement: '°C'
      value_template: "{{ value_json.temperature }}"
    - name: "Humidity"
      state_topic: "state/temp-sensor"
      unit_of_measurement: '%'
      value_template: "{{ value_json.humidity }}"
```
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.
Testing the Sensor

Click Overview at the top of the sidebar and you should see some icons at the top. Within those icons, there should be a temperature and humidity icon.

If you click on one of the icons, you can see a graph of how the value has changed over time. It will also show you how long it has been since the last update.

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, 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.
Going Further

You can easily modify the scripts to show additional information such as the battery level of the sensor or even change it to use other sensors. You can also hook up additional sensor and display that information as well.