Developing an IoT System

Uli Raich

Two lectures on Hardware and Software for
the Internet of Things

Lecture 2: Accessing the “things” through the Internet

Presented online at the African Internet Summit Johannesburg 2023

The “I” in IoT

To communicate with the IoT node over we Internet we must

- Connect the Node to the WiFi network
- Provide a TCP or WEB server (which is a particular type of TCP server!)
- We may need additional protocols like “server side events” or WEB sockets accessed through JavaScript

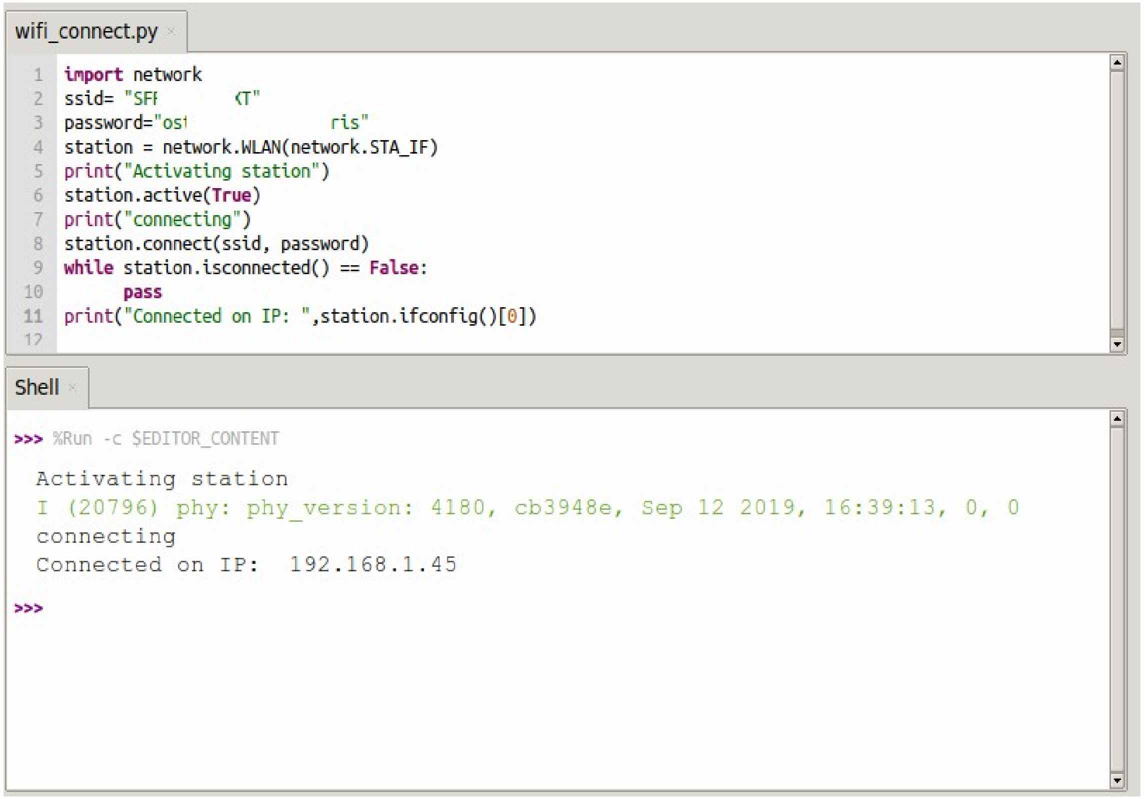
  or

- Communicate to an MQTT broker, which in turn sends or receives data from an MQTT publish or subscribe client

Connecting to the WiFi network

Micropython provides the WLAN class giving access to methods to

- Activate the WiFi station and check the activation
- Scan for nodes in the neighborhood
- connect/disconnect
- Get the network status
- Get/set IP level information



wifi_connect.py

```python
import network
ssid= "SF[redacted]T"
password="os[redacted]ris"
station = network.WLAN(network.STA_IF)
print("Activating station")
station.active(True)
print("connecting")
station.connect(ssid, password)
while station.isconnected() == False:
      pass
print("Connected on IP: ",station.ifconfig()[0])
```

Shell

```
>>> %Run -c $EDITOR_CONTENT

 Activating station
 I (20796) phy: phy_version: 4180, cb3948e, Sep 12 2019, 16:39:13, 0, 0
 connecting
 Connected on IP:  192.168.1.45

>>>
```

A WiFi connection class

When working on IoT you must connect to the network very often.

I therefore wrote and integrated a module named *wifi_connect*

This makes connecting to the network super-simple:

```
from wifi_connect import *

connect()
```

connect also gets the current time from ntp and sets the real time clock on the ESP32

You can get the IP address with

getIPAddress()

or the current time with

gmtTime() or

cetTime()

d

A simple TCP server/client example

The server:

- create a socket
- bind the host address to a port
- listen for connection requests
- accept the connection
- receive data from the connection
- send data to the connection
- Close the connection

A simple TCP server/client example

The client:

- create a socket
- connect to the server
- send data
- receive data

Let’s try it on the PC first!

d

TCP server on the ESP32

There is no difference with respect to the code on the PC

Except prior connection to the WiFi network.

Now we can create a TCP server on the ESP32 that reads some sensors and sends the results to the PC

On the PC we can have a user friendly GUI application which treats and displays the data.

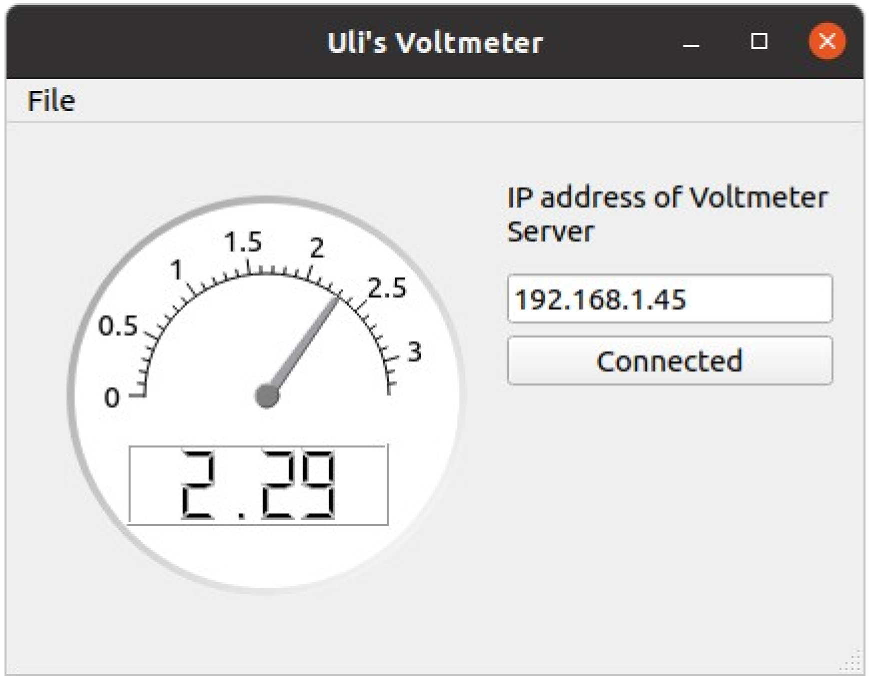
Example: A simple voltmeter. The analogue signal level is read from the ADC and its digitized value is displayed on the Voltmeter application.

The voltmeter application

Like all the other examples the code is entirely written in Python

It uses the Qt5 widget set with the PyQt5 Python language binding

A full description on how to develop a Qt5 application in Python exceeds the scope of these lectures.

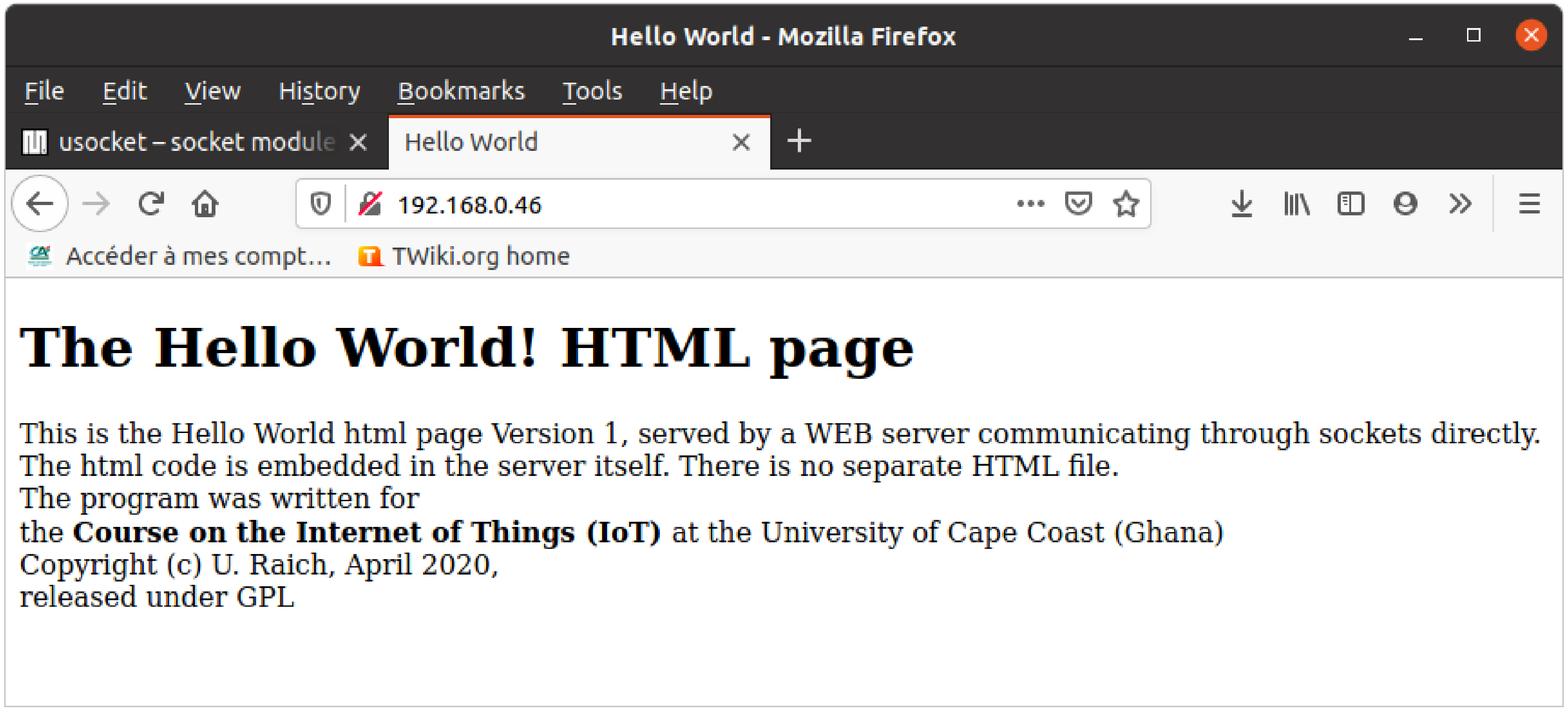


A simple WEB server

As we have seen, MicroPython contains a socket class for network access and that is all that is needed to implement a simple WEB server.

To make things even simpler a basic framework name *picoweb* is available on github.
I integrated this framework into the MicroPython binary to make it globally accessible

The first WEB page



The Hello World! HTML page

This is the Hello World html page Version 1, served by a WEB server communicating through sockets directly.
The html code is embedded in the server itself. There is no separate HTML file.
The program was written for
the **Course on the Internet of Things (IoT)** at the University of Cape Coast (Ghana)
Copyright (c) U. Raich, April 2020,
released under GPL

picoweb

The picoweb module is a framework for writing WEB servers

If contains functionality to

- Create and listen to HTTP requests on a socket
- Handling routing
- Parse HTTP requests
- Prepare HTTP responses by sending the necessary header
- Send HTTP pages stored in files
- Handle templates

d

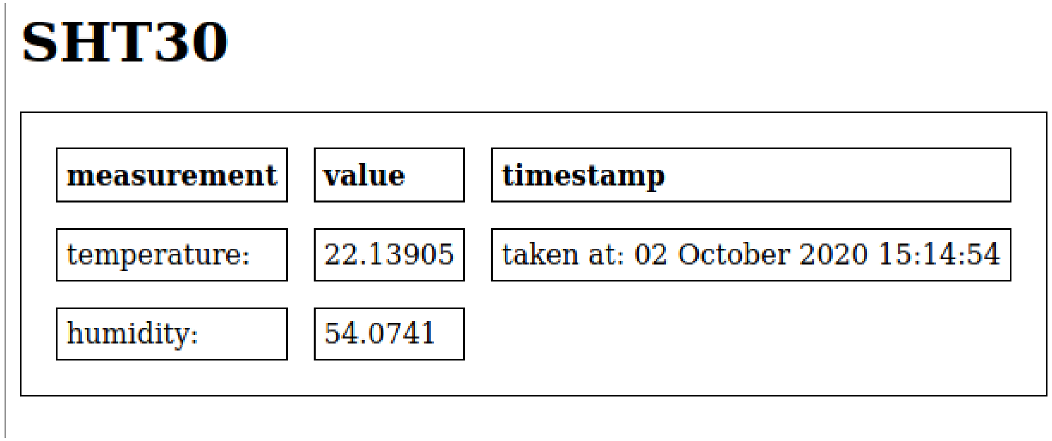
Integrating measurements into the WEB page

That is already not too bad!

However, we want to integrate measurements into the WEB page.

This can be done through templates

We define a HTML table and fill the entries with measurements made be the SHT30.



SHT30

measurement	value	timestamp
temperature:	22.13905	taken at: 02 October 2020 15:14:54
humidity:	54.0741	

d

Server Side events

This is still not perfect because we have to update the whole HTML page if we want to get new measurements

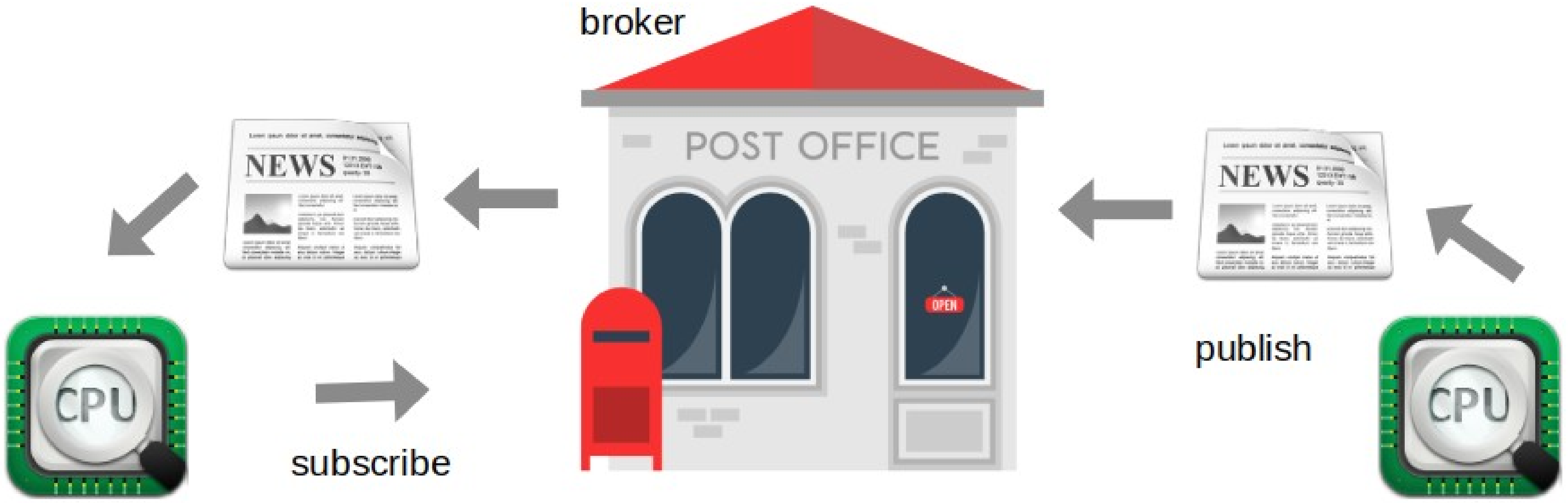
We would like the WEB server make periodic measurements, which update the page on the browser (client) side whenever they are sent

This can be achieved through server side events

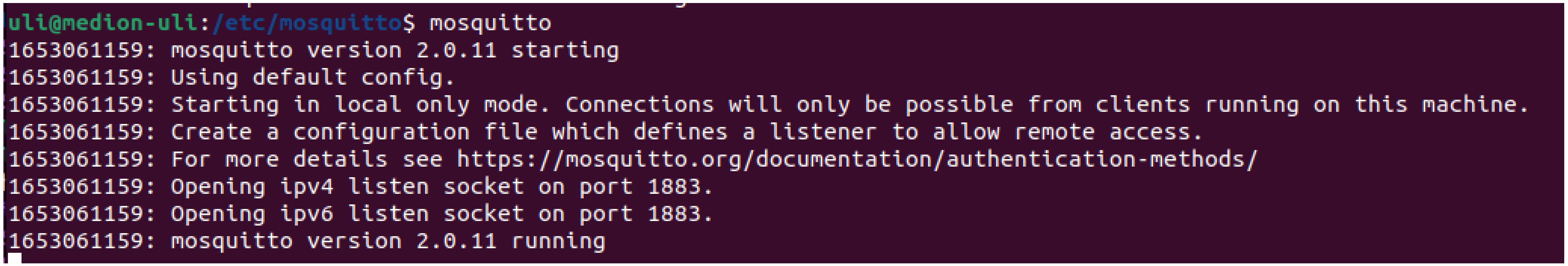
d

MQTT, another way to go online

Message Queuing Telemetry Transport: a publish-subscribe Protocol for IoT



Setting up mosquitto



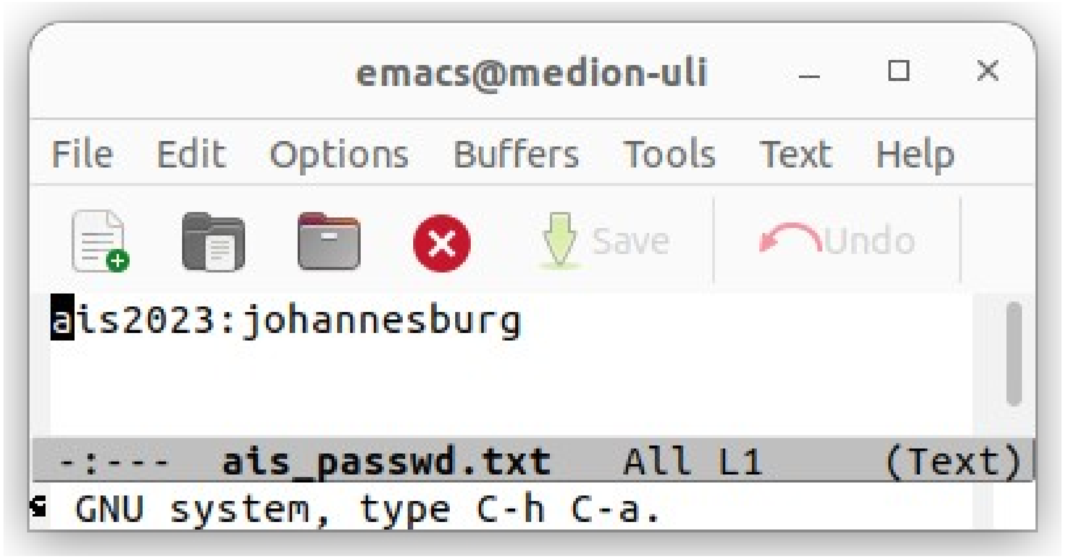
```
uli@medion-uli:/etc/mosquitto$ mosquitto
1653061159: mosquitto version 2.0.11 starting
1653061159: Using default config.
1653061159: Starting in local only mode. Connections will only be possible from clients running on this machine.
1653061159: Create a configuration file which defines a listener to allow remote access.
1653061159: For more details see https://mosquitto.org/documentation/authentication-methods/
1653061159: Opening ipv4 listen socket on port 1883.
1653061159: Opening ipv6 listen socket on port 1883.
1653061159: mosquitto version 2.0.11 running
```

When starting mosquitto you see that we have to define a listener and an authentication method if we want to access the broker from a remote machine like the ESP32

The easiest way to accomplish this is a password file

mosquitto password file

First we create a simple text file with a user name (ais2023) and a password (johannesburg)



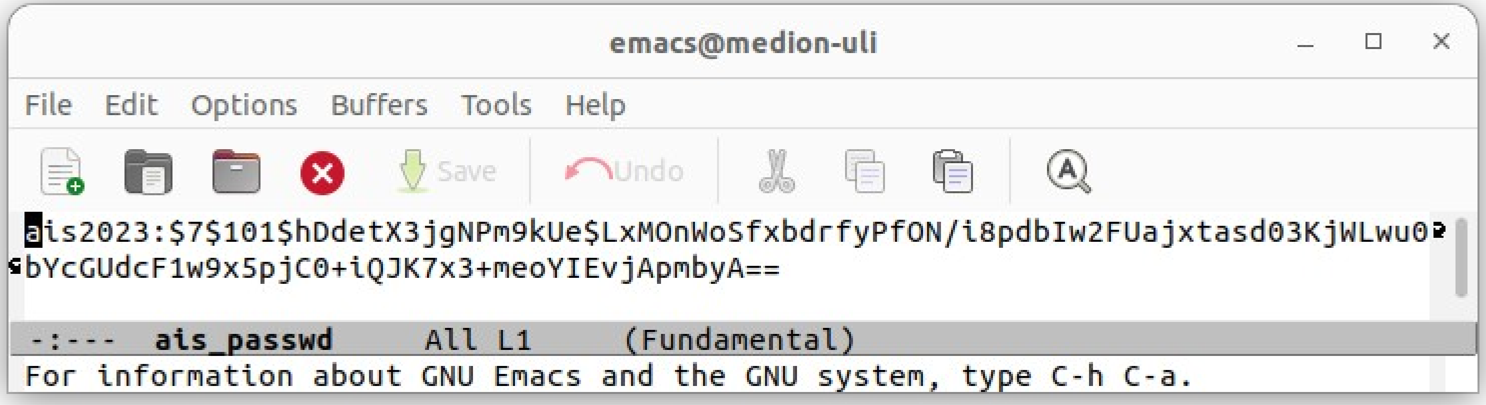
```
ais2023:johannesburg
```

```
-:---  ais_passwd.txt   All L1     (Text)
GNU system, type C-h C-a.
```

Encode password file

Then we encode it with mosquitto_passwd:

cp ais_passwd.txt ais_passwd # the file will be overwritten by the encoded password file
mosquitto_passwd -U ais_passwd

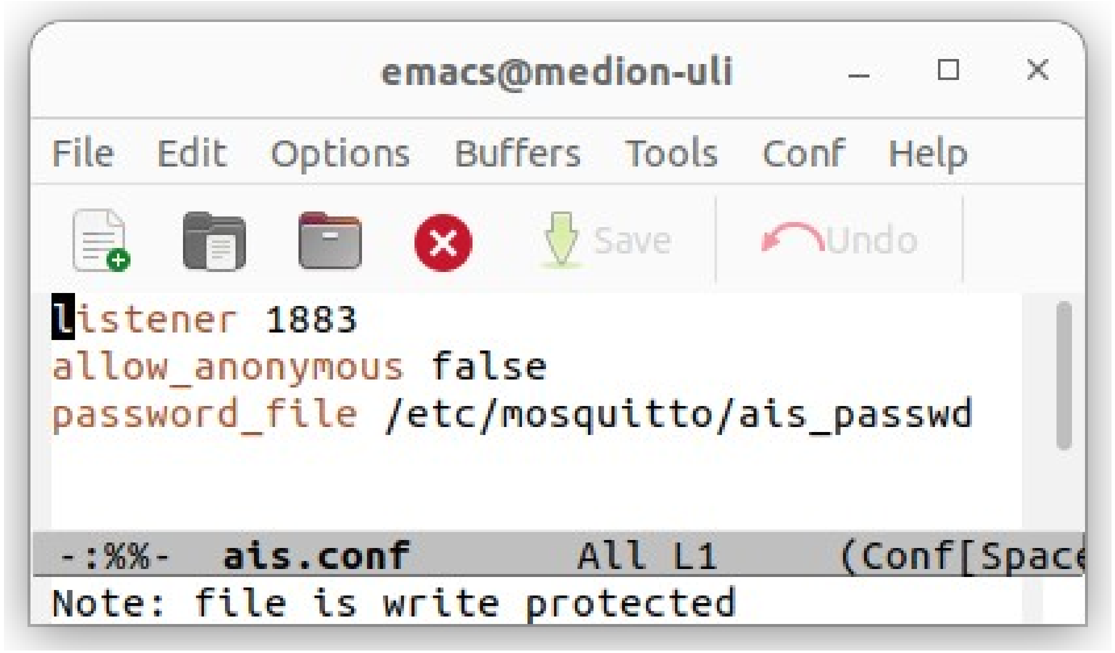


```
ais2023:$7$101$hDdetX3jgNPm9kUe$LxMOnWoSfxbdrfyPfON/i8pdbIw2FUajxtasd03KjWLwu0
bYcGUdcF1w9x5pjC0+iQJK7x3+meoYIEvjApmbyA==
-:---  ais_passwd     All L1     (Fundamental)
For information about GNU Emacs and the GNU system, type C-h C-a.
```

… and we copy it to /etc/mosquitto/

Adapting the config file

Finally we create a custom mosquitto config file, which is located in /etc/mosquitto/conf.d enabling the password file.

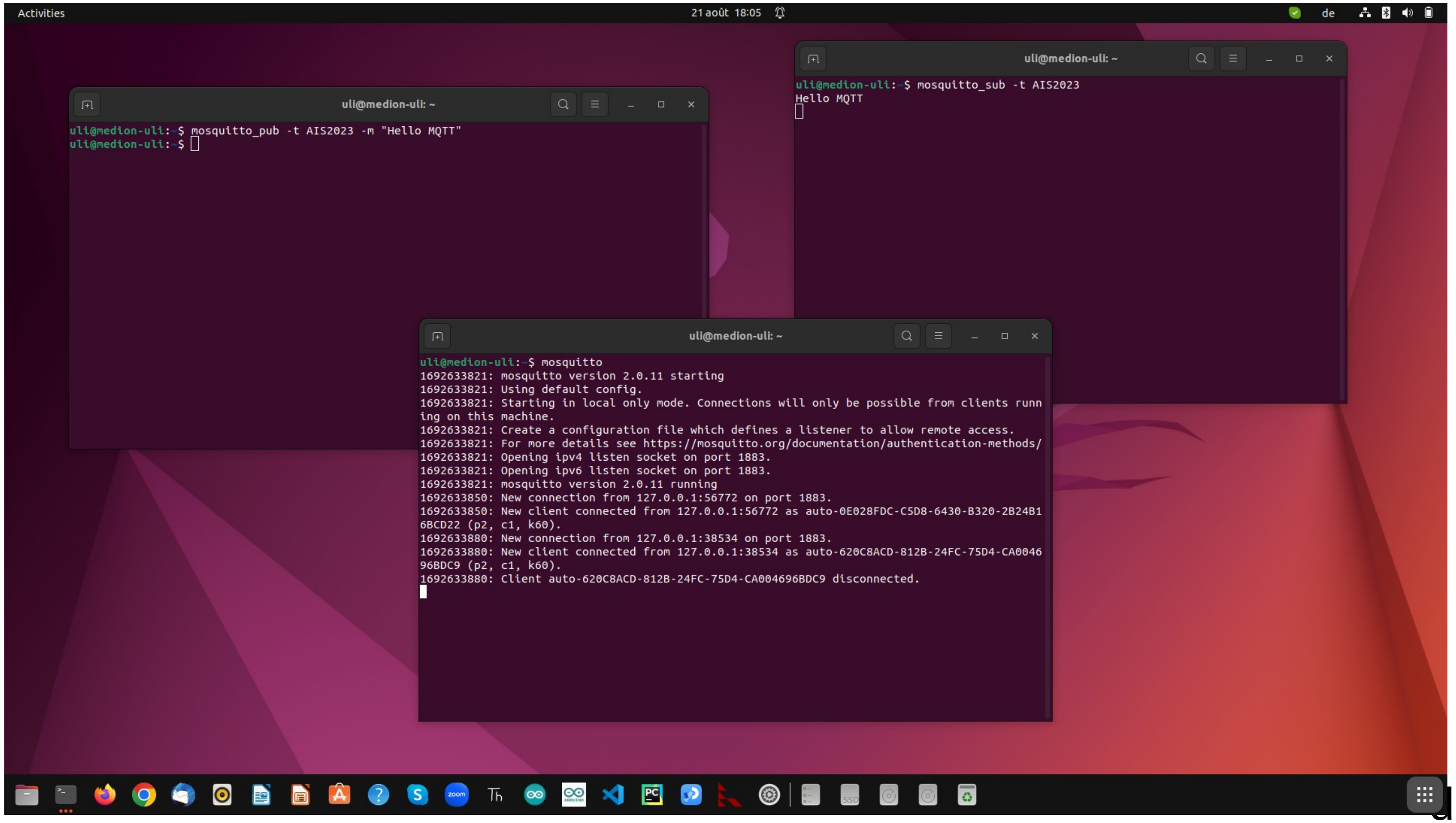


```
listener 1883
allow_anonymous false
password_file /etc/mosquitto/ais_passwd
```

```
-:%%-  ais.conf        All L1      (Conf[Spac
Note: file is write protected
```

The mosquitto broker

This works only on the local machine



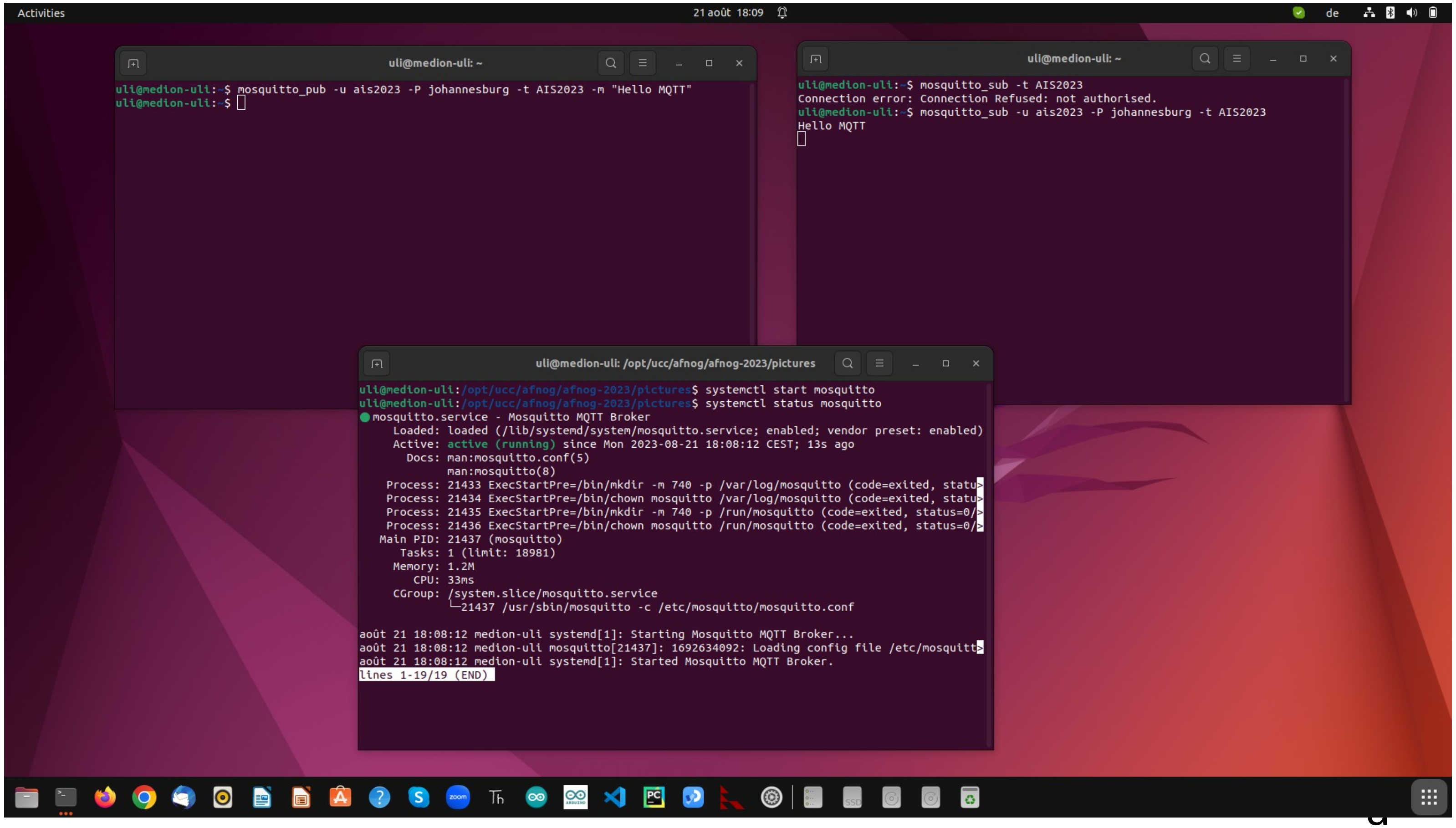
```
uli@medion-uli:~$ mosquitto_pub -t AIS2023 -m "Hello MQTT"
uli@medion-uli:~$
```

```
uli@medion-uli:~$ mosquitto_sub -t AIS2023
Hello MQTT
```

```
uli@medion-uli:~$ mosquitto
1692633821: mosquitto version 2.0.11 starting
1692633821: Using default config.
1692633821: Starting in local only mode. Connections will only be possible from clients runn
ing on this machine.
1692633821: Create a configuration file which defines a listener to allow remote access.
1692633821: For more details see https://mosquitto.org/documentation/authentication-methods/
1692633821: Opening ipv4 listen socket on port 1883.
1692633821: Opening ipv6 listen socket on port 1883.
1692633821: mosquitto version 2.0.11 running
1692633850: New connection from 127.0.0.1:56772 on port 1883.
1692633850: New client connected from 127.0.0.1:56772 as auto-0E028FDC-C5D8-6430-B320-2B24B1
6BCD22 (p2, c1, k60).
1692633880: New connection from 127.0.0.1:38534 on port 1883.
1692633880: New client connected from 127.0.0.1:38534 as auto-620C8ACD-812B-24FC-75D4-CA0046
96BDC9 (p2, c1, k60).
1692633880: Client auto-620C8ACD-812B-24FC-75D4-CA004696BDC9 disconnected.
```

The mosquitto broker with password

This works also with a remote subscriber or publisher



```
uli@medion-uli:~$ mosquitto_pub -u ais2023 -P johannesburg -t AIS2023 -m "Hello MQTT"
uli@medion-uli:~$ 
```

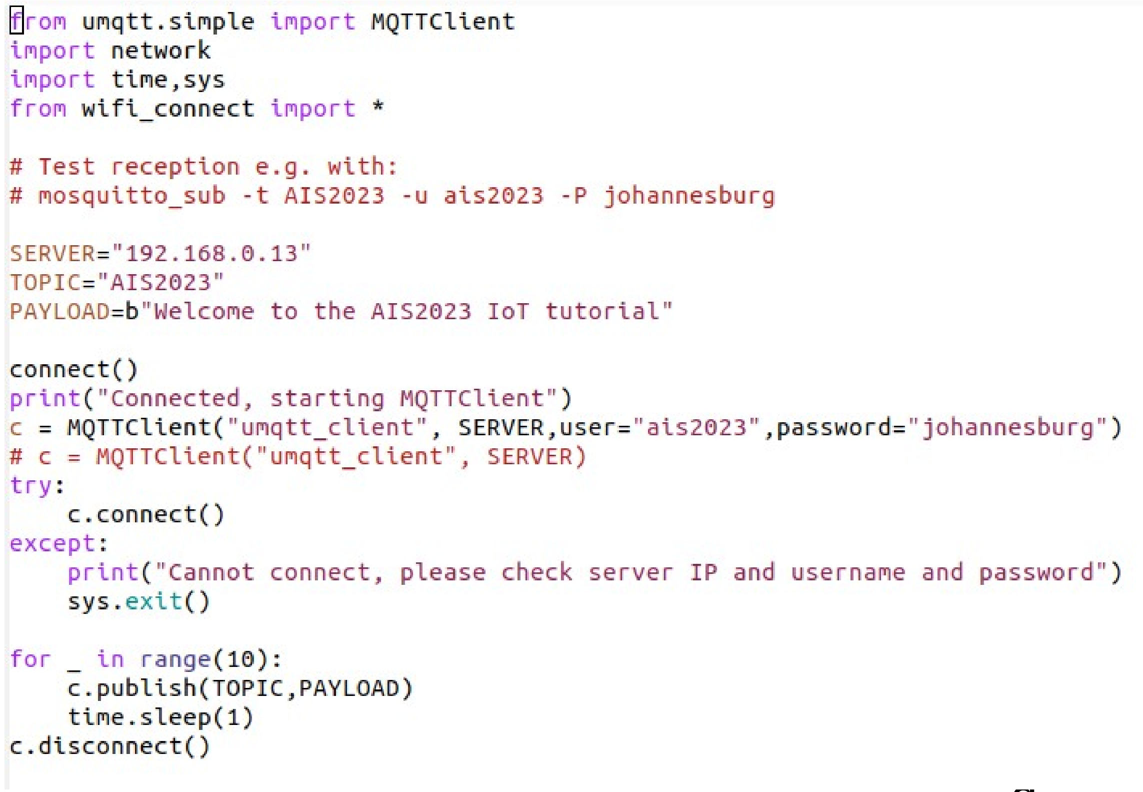
```
uli@medion-uli:~$ mosquitto_sub -t AIS2023
Connection error: Connection Refused: not authorised.
uli@medion-uli:~$ mosquitto_sub -u ais2023 -P johannesburg -t AIS2023
Hello MQTT
```

```
uli@medion-uli:/opt/ucc/afnog/afnog-2023/pictures$ systemctl start mosquitto
uli@medion-uli:/opt/ucc/afnog/afnog-2023/pictures$ systemctl status mosquitto
● mosquitto.service - Mosquitto MQTT Broker
     Loaded: loaded (/lib/systemd/system/mosquitto.service; enabled; vendor preset: enabled)
     Active: active (running) since Mon 2023-08-21 18:08:12 CEST; 13s ago
       Docs: man:mosquitto.conf(5)
             man:mosquitto(8)
    Process: 21433 ExecStartPre=/bin/mkdir -m 740 -p /var/log/mosquitto (code=exited, statu>
    Process: 21434 ExecStartPre=/bin/chown mosquitto /var/log/mosquitto (code=exited, statu>
    Process: 21435 ExecStartPre=/bin/mkdir -m 740 -p /run/mosquitto (code=exited, status=0/>
    Process: 21436 ExecStartPre=/bin/chown mosquitto /run/mosquitto (code=exited, status=0/>
   Main PID: 21437 (mosquitto)
      Tasks: 1 (limit: 18981)
     Memory: 1.2M
        CPU: 33ms
     CGroup: /system.slice/mosquitto.service
             └─21437 /usr/sbin/mosquitto -c /etc/mosquitto/mosquitto.conf

août 21 18:08:12 medion-uli systemd[1]: Starting Mosquitto MQTT Broker...
août 21 18:08:12 medion-uli mosquitto[21437]: 1692634092: Loading config file /etc/mosquitt>
août 21 18:08:12 medion-uli systemd[1]: Started Mosquitto MQTT Broker.
lines 1-19/19 (END)
```

MQTT client on the ESP32

… and if I could have the MQTT client on the ESP32.

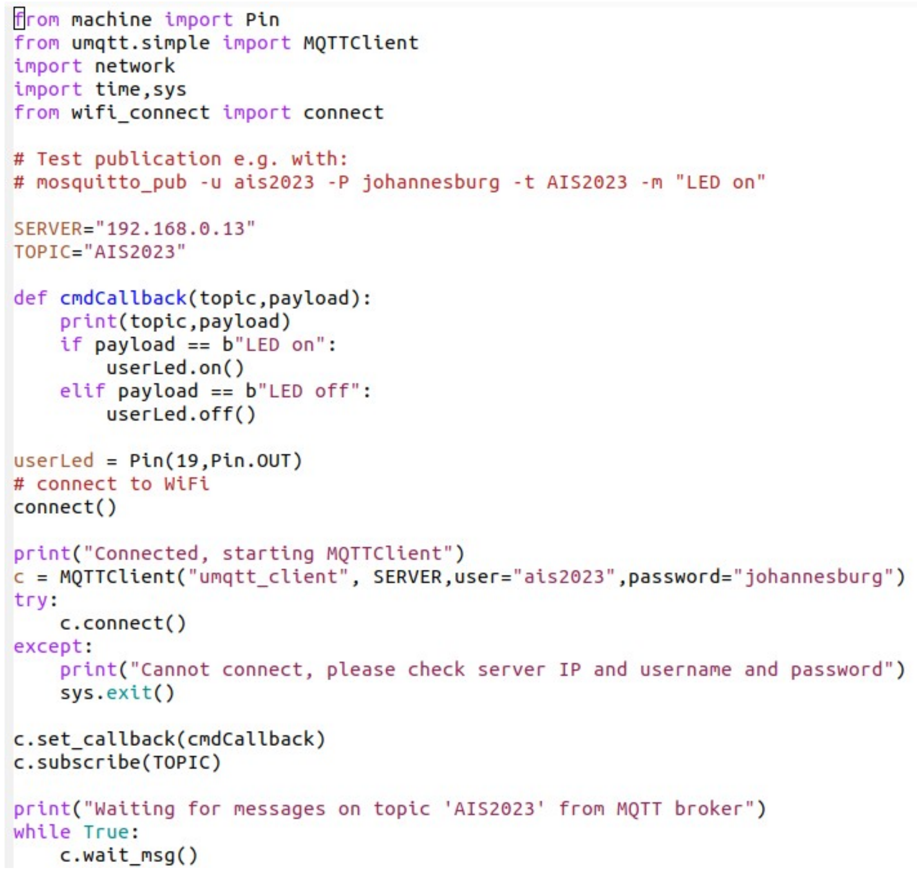
- If it was the publishing client it could send measurements to the broker and thus to any subscribed client
- If it was the subscribing client it could receive commands from the broker and thus from any publishing client
- micropython-lib supplies the umqtt library giving us access to MQTT



```
from umqtt.simple import MQTTClient
import network
import time,sys
from wifi_connect import *

# Test reception e.g. with:
# mosquitto_sub -t AIS2023 -u ais2023 -P johannesburg

SERVER="192.168.0.13"
TOPIC="AIS2023"
PAYLOAD=b"Welcome to the AIS2023 IoT tutorial"

connect()
print("Connected, starting MQTTClient")
c = MQTTClient("umqtt_client", SERVER,user="ais2023",password="johannesburg")
# c = MQTTClient("umqtt_client", SERVER)
try:
    c.connect()
except:
    print("Cannot connect, please check server IP and username and password")
    sys.exit()

for _ in range(10):
    c.publish(TOPIC,PAYLOAD)
    time.sleep(1)
c.disconnect()
```

a

Subscribing client on the ESP32



```
from machine import Pin
from umqtt.simple import MQTTClient
import network
import time,sys
from wifi_connect import connect

# Test publication e.g. with:
# mosquitto_pub -u ais2023 -P johannesburg -t AIS2023 -m "LED on"

SERVER="192.168.0.13"
TOPIC="AIS2023"

def cmdCallback(topic,payload):
    print(topic,payload)
    if payload == b"LED on":
        userLed.on()
    elif payload == b"LED off":
        userLed.off()

userLed = Pin(19,Pin.OUT)
# connect to WiFi
connect()

print("Connected, starting MQTTClient")
c = MQTTClient("umqtt_client", SERVER,user="ais2023",password="johannesburg")
try:
    c.connect()
except:
    print("Cannot connect, please check server IP and username and password")
    sys.exit()

c.set_callback(cmdCallback)
c.subscribe(TOPIC)

print("Waiting for messages on topic 'AIS2023' from MQTT broker")
while True:
    c.wait_msg()
```

d

Cayenne

myDevices Cayenne claims to be the world's first drag and drop IoT builder.

It provides the MQTT broker and uses its own protocol on top of the MQTT messages to transfer information between itself and the IoT system.

It also supplies a dash board with widgets to graphically represent measured parameters.

It can work on many different types of micro-controllers and has language bindings for

- C, C++
- Arduino IDE
- Python

Unfortunately the Cayenne Python library depends on the Eclipse Paho MQTT library and cannot be used with MicroPython without modification

However, the library is OpenSource and I managed to adapt it to MicroPython's umqtt

… this needs an update!

MyDevices Cayenne end of life

ThingSpeak

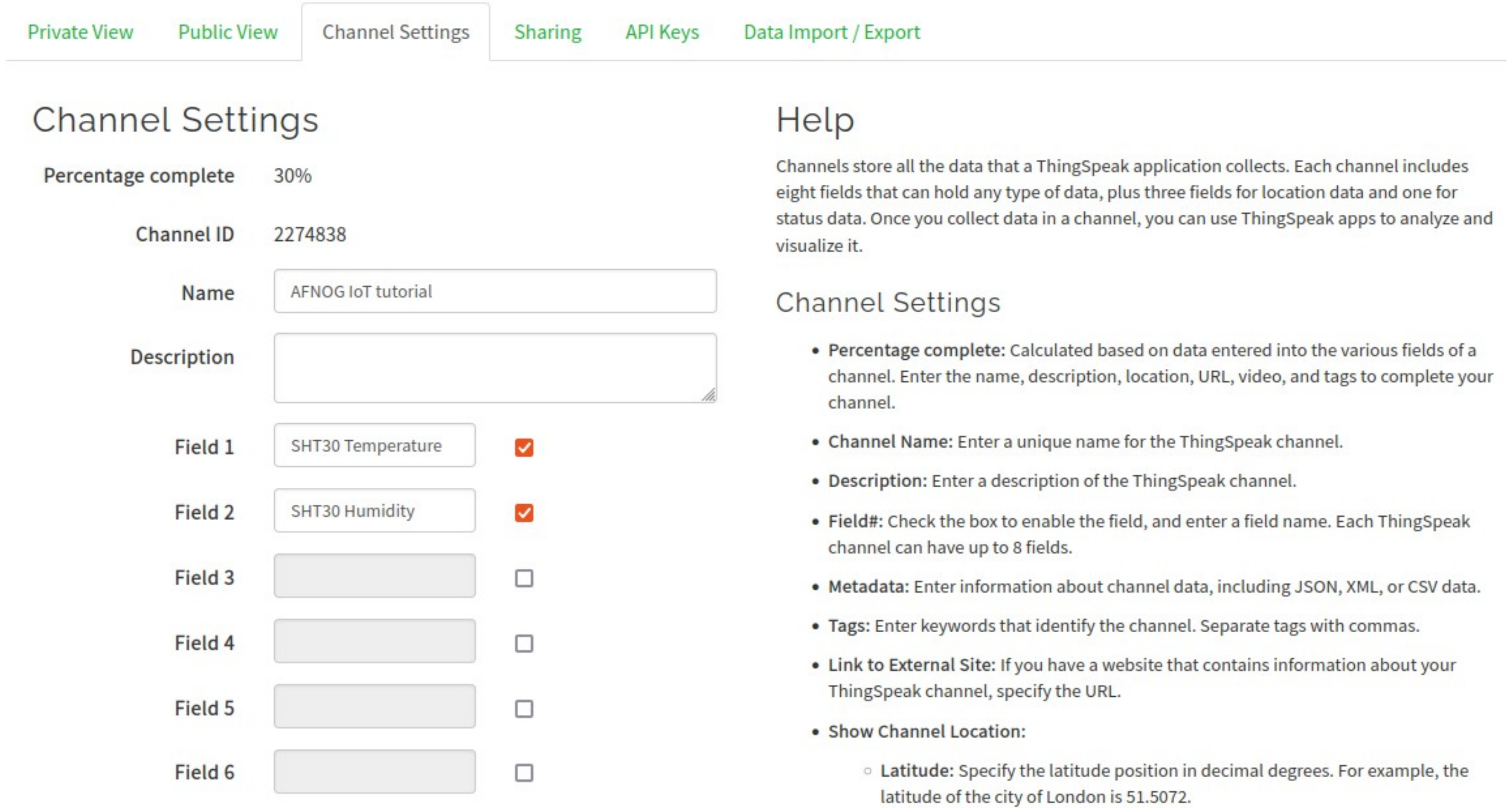
My experience with ThingSpeak is less than 1 day!

You can communicate with the ThingSpeak server using the

- HTTP protocol
- MQTT

You can transmit the data measured on the ESP32 to a channel on ThingSpeak and display he measurements in graphs or use widgets like a gauge to show current values.

Create a ThingSpeak channel



Private View | Public View | Channel Settings | Sharing | API Keys | Data Import / Export

Channel Settings

Percentage complete	30%	
Channel ID	2274838	
Name	AFNOG IoT tutorial	
Description		
Field 1	SHT30 Temperature	✓
Field 2	SHT30 Humidity	✓
Field 3		☐
Field 4		☐
Field 5		☐
Field 6		☐

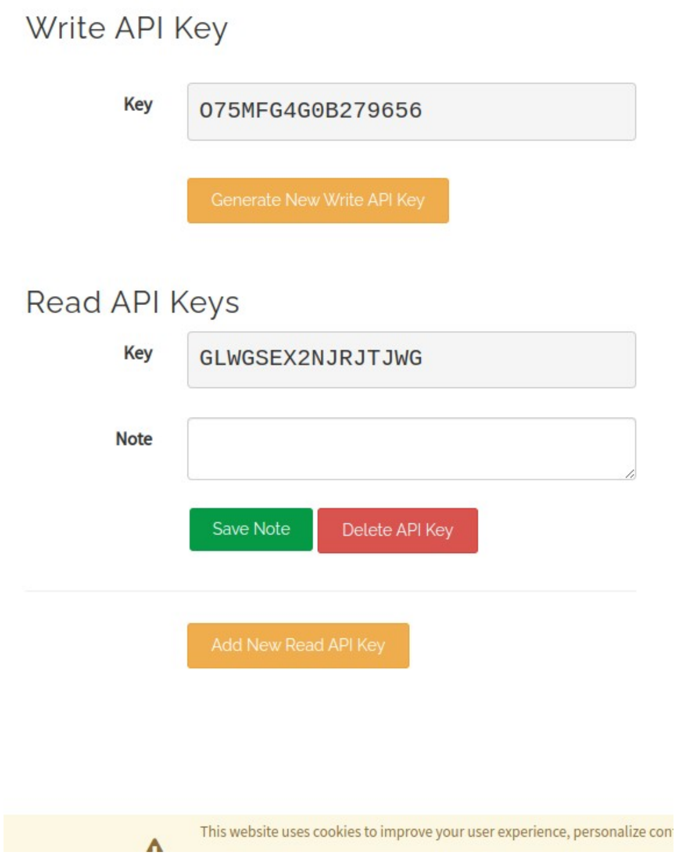
Help

Channels store all the data that a ThingSpeak application collects. Each channel includes eight fields that can hold any type of data, plus three fields for location data and one for status data. Once you collect data in a channel, you can use ThingSpeak apps to analyze and visualize it.

Channel Settings

- **Percentage complete:** Calculated based on data entered into the various fields of a channel. Enter the name, description, location, URL, video, and tags to complete your channel.
- **Channel Name:** Enter a unique name for the ThingSpeak channel.
- **Description:** Enter a description of the ThingSpeak channel.
- **Field#:** Check the box to enable the field, and enter a field name. Each ThingSpeak channel can have up to 8 fields.
- **Metadata:** Enter information about channel data, including JSON, XML, or CSV data.
- **Tags:** Enter keywords that identify the channel. Separate tags with commas.
- **Link to External Site:** If you have a website that contains information about your ThingSpeak channel, specify the URL.
- **Show Channel Location:**
  - **Latitude:** Specify the latitude position in decimal degrees. For example, the latitude of the city of London is 51.5072.

The API keys



Write API Key

Key: 075MFG4G0B279656

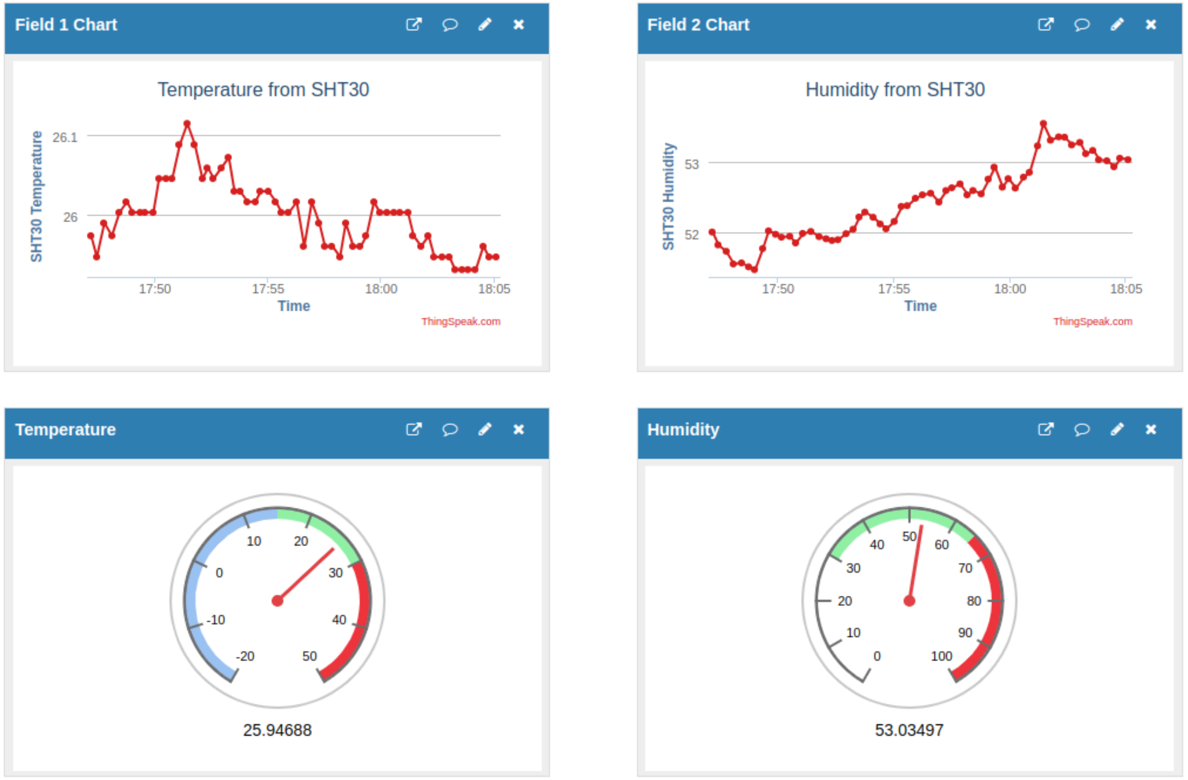
Read API Keys

Key: GLWGSEX2NJRJTJWG

Note:

This website uses cookies to improve your user experience, personalize con

ThingSpeak dash board



What next

This tutorial showed only three different devices:

- The SHT30 temperature and humidity sensor
- The WS2812 addressable rgb LED and the LED ring
- The linear potentiometer in conjunction with the on-chip ADC

However, there are many more WeMos D1 mini shields some of which are demonstrated in the IoT course prepared for the University of Cape Coast, Ghana.

The description is available on the Twiki under IoT Course in English as well as exercises proposed for these devices.

A slightly different french version is also available on the Twiki IoT Course in French

The solutions can be downloaded from my github repository.

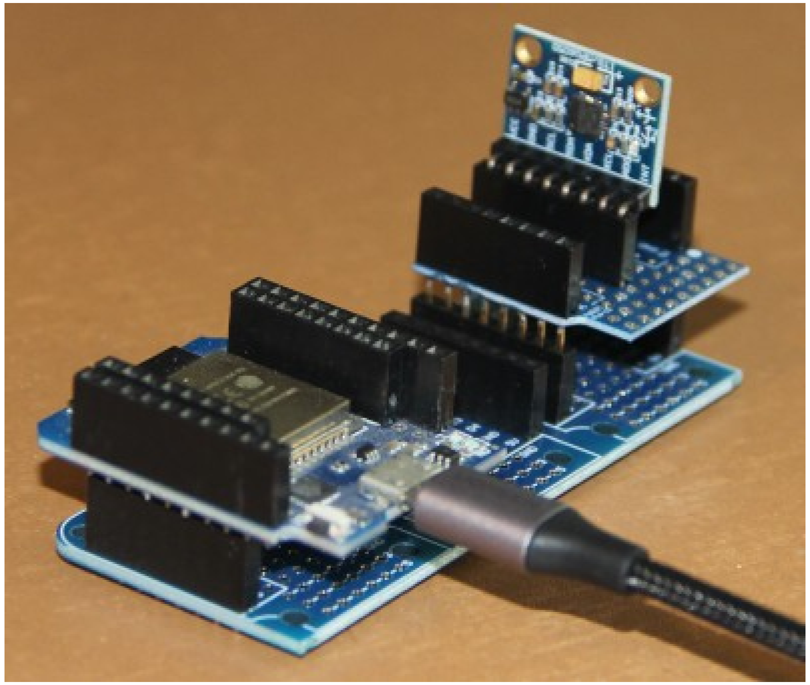
Adapt other sensor boards

There are plenty other sensor and actuator boards available, which can be adapted to the WeMos D1 mini bus using the prototype board.

Here is the example of an accelerometer and gyroscope

Writing drivers for sensor boards is not always a trivial task.

I ported a library for the MPU6050 driver originally written in C++ for the Arduino to MicroPython.
The driver has 6000 lines of MicroPython Code



AFRICA INTERNET SUMMIT'23 26 - 30 SEPTEMBER 2023 JOHANNESBURG

TinyML

The other most fashionable subject in computer science today is **Artificial Intelligence (AI) and Machine Learning (ML)**

We have seen how to read out and interpret data from a temperature and humidity sensor but what do we have to do if we want to

- understand spoken key words captured by a microphone ?
- know if a person is in sight of a camera or not ?
- understand gestures measured with an accelerometer ?

Such type of problems can be solved with Artificial Intelligence.

Hardware for TinyML

Uses the ESP32S3, which has vector instructions speeding up neural network operations

Has 8 MB of octal SPIRAM and therefore plenty of space to store tensorflow lite micro models

Has a camera on board allowing image treatment

TinyML can be run in C++ programs or in MicroPython scripts

MicroPython TinyML requires a specialized firmware with numpy and the tensorflow lite micro library integrated into it.

Creating AI models

Creating machine learning models and making them learn, requires large computing resources and there is no chance to do this on micro-controllers.

The model recognizing a person on an image is fed 120.000 images to make it learn to distinguish images with and without human beings.

My gaming PC with a GPU however can do this. The learning process may take hours or even days.

The final model however can be squeezed down in size in such a way that it fits into the micro-controller memory. The micro-controller code will read images from the camera and feed them into the previously prepared model. The model then answers if a person is there or not.

A lot to learn!

AI and machine learning requires quite a bit of knowledge:

- The mathematics behind the algorithms is not always trivial
- You need to know how to use a big number of Python libraries:
  - numpy
  - pandas
  - matplotlib, seaborn,
  - tensorflow, tensorflow lite, tensorflow lite micro, keras, scikit-learn
  - and quite a few more

Maybe you will come back to AIS-2024 for a tutorial on tinyML?

LOONEY TUNES
Thats all Folks