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
(The “I” part of IoT)

Presented at the African Internet Summit Online 2021

AFRICA INTERNET SUMMIT'21 ONLINE 31 MAY - 4 JUNE 2021

The “I” in IoT

To communicate with the IoT node over we Internet we must

- Connect the Node to the WiFi network
- Create and connect to a TCP socket
- Provide a WEB 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 and/or receives data from an MQTT publish or subscribe client

AFRICA INTERNET SUMMIT'21
ONLINE
31 MAY - 4 JUNE 2021

Connecting to the WiFi network

MicroPython's network module has the functions we need to connect to the WiFi network

wifi_connect.py

```
1  import network
2  ssid= "SFI        KT"
3  password="ost                ris"
4  station = network.WLAN(network.STA_IF)
5  print("Activating station")
6  station.active(True)
7  print("connecting")
8  station.connect(ssid, password)
9  while station.isconnected() == False:
10         pass
11 print("Connected on IP: ",station.ifconfig()[0])
12
```

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

>>>
```

Connect to WiFi (2)

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 current time with

gmtTime() or

cetTime()

A simple TCP server/client example

We make use of the usocket module in MicroPython

The server:

- Connect to the Network (WiFi)
- 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

AFRICA INTERNET SUMMIT'21 ONLINE 31 MAY - 4 JUNE 2021

The server code

```
def server_program():
    # get the hostname
    host = socket.gethostname()
    port = 5000  # initiate port no above 1024

    server_socket = socket.socket()  # get instance
    # look closely. The bind() function takes tuple as argument
    server_socket.bind(('', port))  # bind host address and port together

    print("Server running on " + host + " with IP: " + get_ip())
    print("Listening to any machine on port ",port)
    server_socket.listen(1)
    conn, address = server_socket.accept()  # accept new connection
    print("Connection from: " + str(address))
    conn.send(("From server: Connected to " + get_ip()).encode())

    while True:
        # receive data stream. it won't accept data packet greater than 1024 byt
es
        data = conn.recv(1024).decode()
        if not data:
            # if data is not received break
            break
        print("from connected user: " + str(data))
        data = input(' -> ')
        conn.send(data.encode())  # send data to the client

    conn.close()  # close the connection
```

A simple TCP server/client example

The client:

- Connect to the Network (WiFi)
- create a socket
- connect to the server
- send data
- receive data

Let’s try it on the PC first!

The client code

```
def client_program(host_ip):
    host = socket.gethostname()  # as both code is running on same pc
    port = 5000  # socket server port number

    client_socket = socket.socket()  # instantiate
    print("Connecting to ",host,":",port)
    try:
        client_socket.connect((host_ip, port))  # connect to the server
    except OSError as error:
        print("Connection failed, please check IP address and port number")
        sys.exit()

    data = client_socket.recv(1024).decode()  # receive response
    print(data)

    message = input(" -> ")  # take input

    while message.lower().strip() != 'bye':
        client_socket.send((message + '\r\n').encode())  # send message
        data = client_socket.recv(1024).decode()  # receive response

        print('Received from server: ' + data)  # show in terminal

        message = input(" -> ")  # again take input

    client_socket.close()  # close the connection
```

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 TCP client on PC.

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

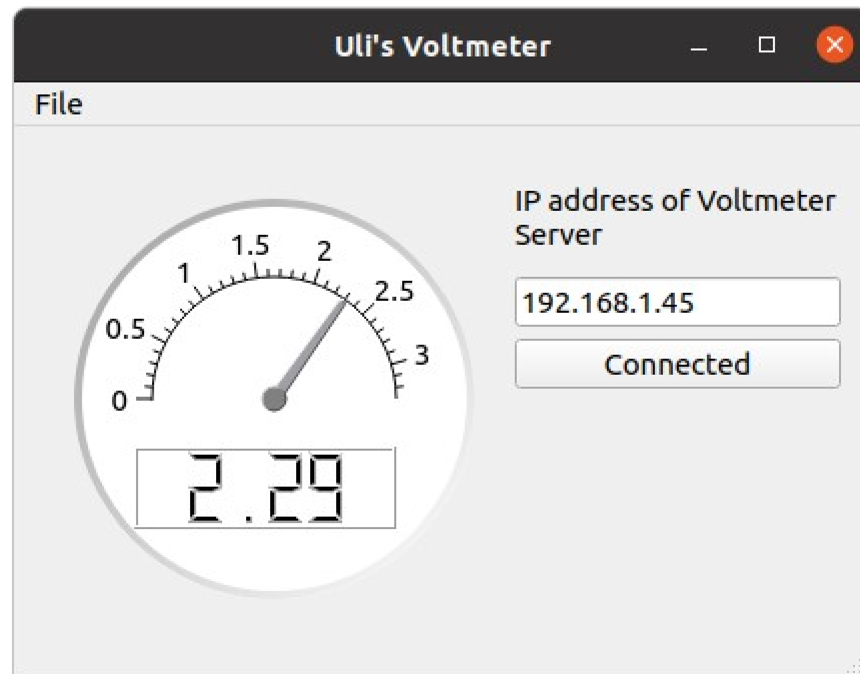
Example: A simple voltmeter. The analogue signal level is read from the ADC on the ESP32 and its digitized value is send to the TCP client on the PC, where it is displayed on the Voltmeter application.

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

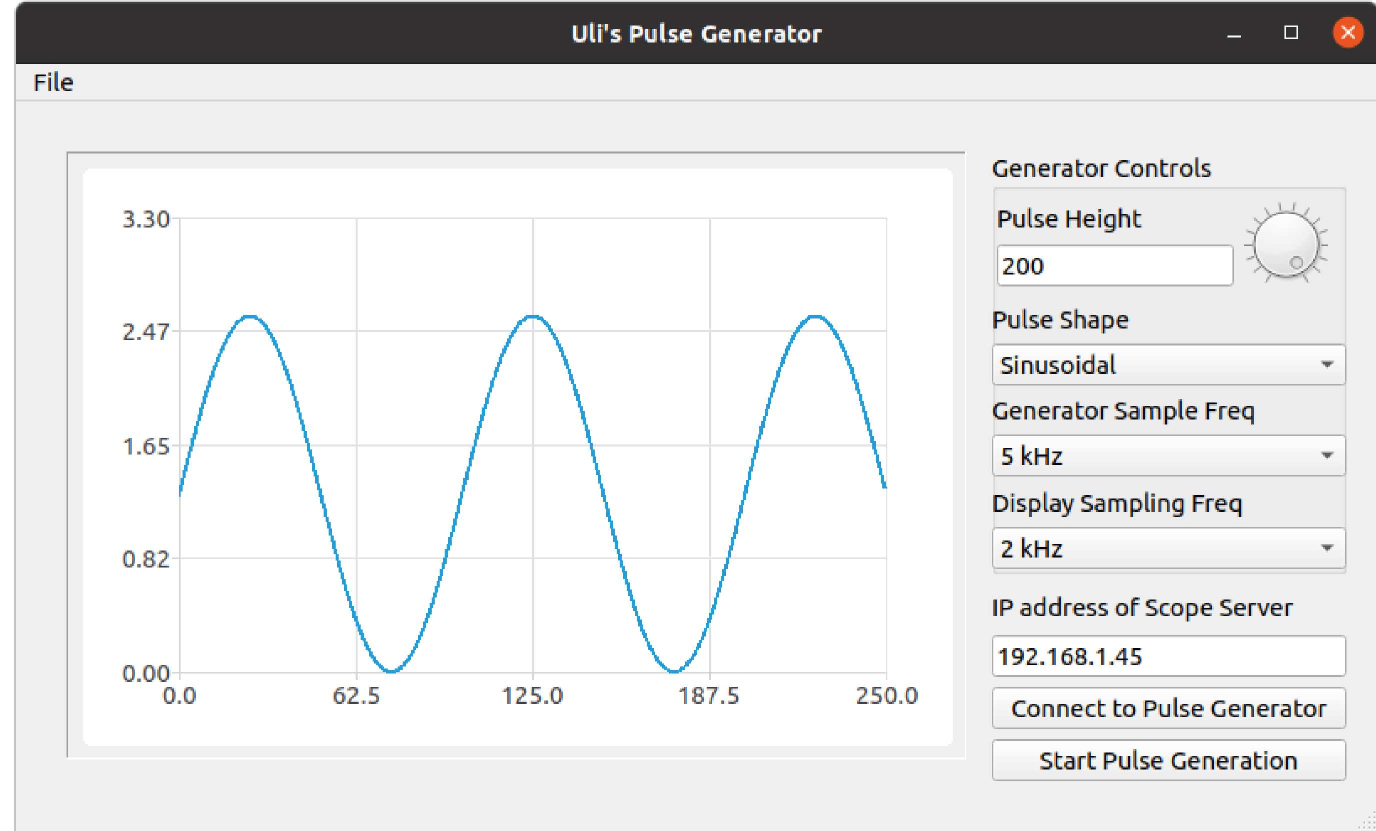
A full description on how to develop a Qt5 application in Python exceeds the scope of these lectures.



AFRICA INTERNET SUMMIT'21 ONLINE 31 MAY - 4 JUNE 2021

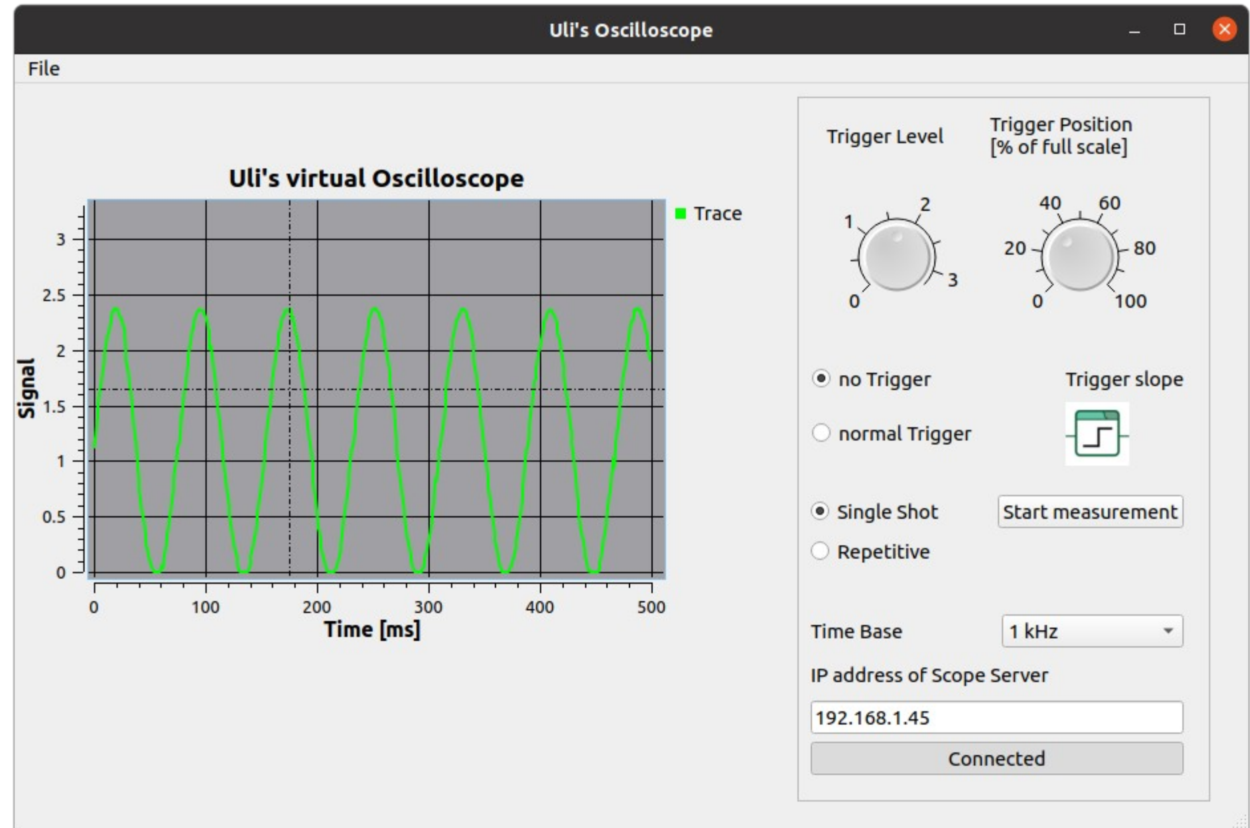
Other virtual instruments

A pulse generator



A virtual oscilloscope

A virtual oscilloscope showing the output of the pulse generator

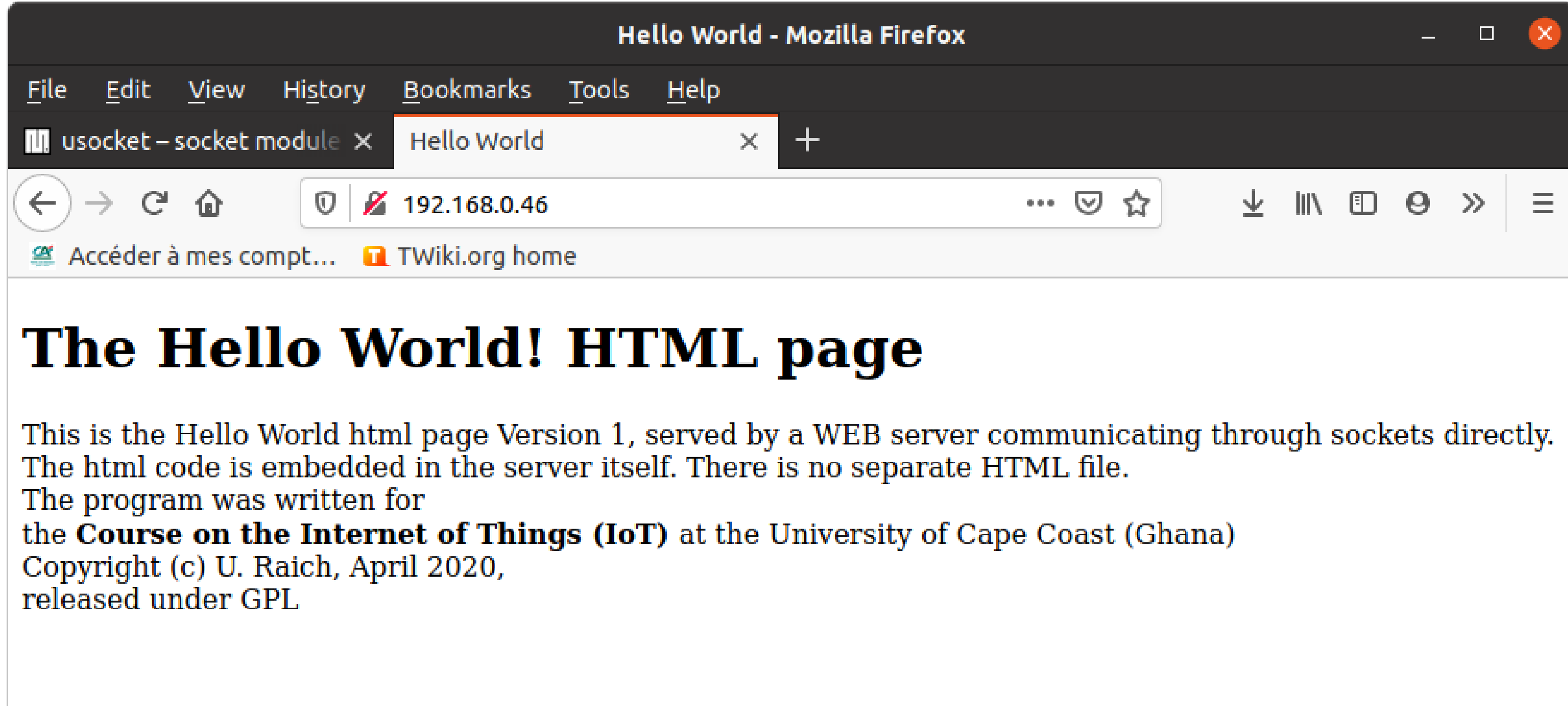


A simple WEB server

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



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 by the SHT30.

SHT30

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

The template

```python
@app.route("/")
@app.route("/temp")
def html(req, resp):
    t,h = sht30.getTempAndHumi(clockStretching=SHT3X.CLOCK_STRETCH,
                               repeatability=SHT3X.REP_S_HIGH)
    print("Temperature: ",t,"°C, Humidity: ",h,"%")
    tm = time.localtime(time.time())
    timeStamp = '{0:02d} {1} {2:04d} {3:02d}:{4:02d}:{5:02d}'.format(
        tm[2],monthTable[tm[1]],tm[0],tm[3],tm[4],tm[5])
    sensor={"tmpr":t,"hmdty":h,"timeStamp":timeStamp}
    msg = (b'{0:3.1f} {1:3.1f}'.format(t,h))
    print(msg)
    yield from picoweb.start_response(resp,
                    content_type = "text/html; charset=utf-8")
    yield from app.render_template(resp, "sensor.tpl", (sensor,))

app.run(debug=True, host =ipaddr,port=80)
```

```html
</head>
<body>
<h1> SHT30 </h1>
<table>
  <tr>
    <th align="left">measurement</th>
    <th align="left">value</th>
    <th align="left">timestamp</th>
  </tr>
  <tr>
    <td>temperature:</td>
    <td>{{sensor['tmpr']}}</td>
    <td>taken at: {{sensor['timeStamp']}}</td>
  </tr>
  <tr>
    <td>humidity:</td>
    <td>{{sensor['hmdty']}}</td>
  </tr>
</table>

</body>
```

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

Javascript to decode measurement text

```
tempC, humi = sht30.getTempAndHumi(clockStretching=SHT3X.CLOCK_STRETCH,repeatability=SHT3X.REP_S_HIGH)
timeStamp=dateString(cetTime())
measurement="data: temperature={:2.1f},humidity={:2.1f},timeStamp={:s}\n\n".format(tempC,humi,timeStamp)
```

```
    <div>
      <p id="measText">Measurement</p>
    </div>

<script>
  var source = new EventSource("events");
  source.onmessage = function(event) {
      var meas = event.data;
      document.getElementById("measText").innerHTML = meas;
      var measArray=meas.split(",");

      var tempArray=measArray[0].split("=");
      document.getElementById("temp").innerHTML = tempArray[1];
```

Plotting the data

We can still do better:

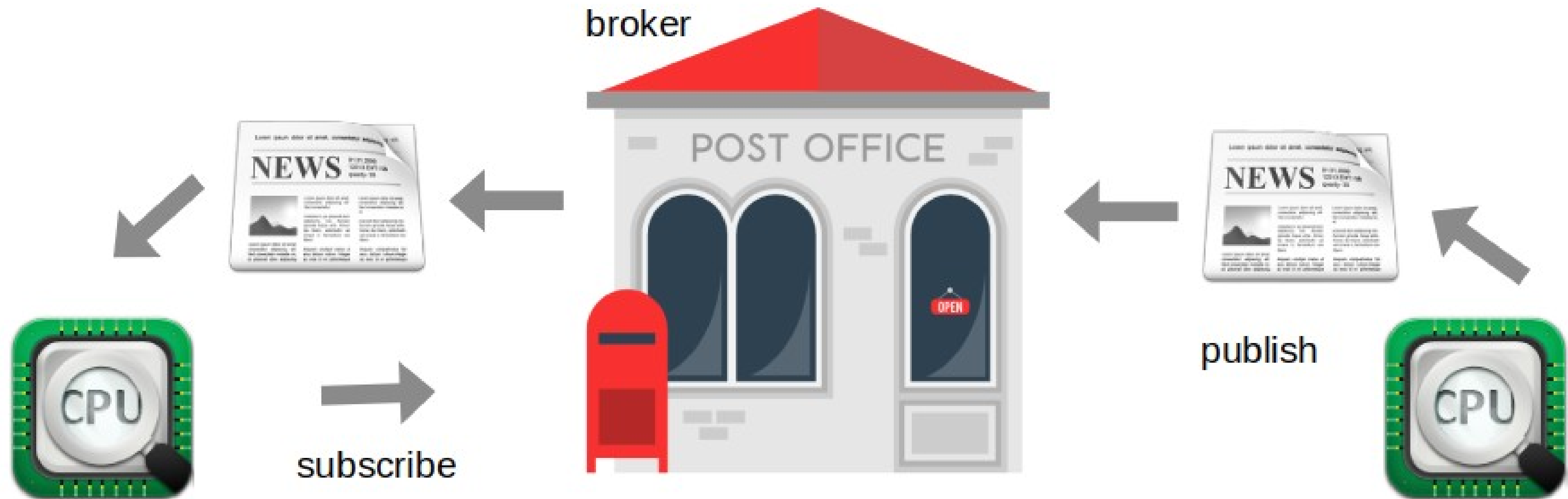
- Collect the data and plot them with HighCharts

HighCharts is a JavaScript plotting library

This is implemented on the page *sht30MeasAndPlot*

MQTT, another way to go online

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



AFRICA
INTERNET
SUMMIT'21
ONLINE
31 MAY - 4 JUNE 2021

The mosquitto broker

uli@medionUli: ~

```
uli@medionUli:~$ mosquitto
[33026.989067]~DLT~10939~INFO     ~FIFO /tmp/dlt cannot be opened. Retrying late
r...
1599072108: mosquitto version 1.6.9 starting
1599072108: Using default config.
1599072108: Opening ipv4 listen socket on port 1883.
1599072108: Opening ipv6 listen socket on port 1883.
1599072154: New connection from 127.0.0.1 on port 1883.
1599072154: New client connected from 127.0.0.1 as mosq-fcjCTujdXatJ1iNvoT (p2,
c1, k60).
1599072249: New connection from 127.0.0.1 on port 1883.
1599072249: New client connected from 127.0.0.1 as mosq-Dfo9FBzO6V4K9EsJfK (p2,
c1, k60).
1599072249: Client mosq-Dfo9FBzO6V4K9EsJfK disconnected.
```

uli@medionUli: ~

```
uli@medionUli:~$ mosquitto_sub -t "DCSIT"
Welcome to the Department of Computer Science and Information Technology a UCC
```

uli@medionUli: ~

```
uli@medionUli:~$ mosquitto_pub -t "DCSIT" -m "Welcome to the Department of Compu
ter Science and Information Technology a UCC"
uli@medionUli:~$
```

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
from wifi_connect import *

# Test reception e.g. with:
# mosquitto_sub -t AIS2021

SERVER="192.168.1.36"
TOPIC="AIS2021"
PAYLOAD=b"Welcome to the AIS2021 IoT lecture"

connect()
print("Connected, starting MQTTClient")
c = MQTTClient("umqtt_client", SERVER)
c.connect()
for _ in range(10):
    c.publish(TOPIC,PAYLOAD)
    time.sleep(1)
c.disconnect()
```

AFRICA INTERNET SUMMIT'21 ONLINE 31 MAY - 4 JUNE 2021

Subscribing client on the ESP32

```
from machine import Pin
from umqtt.simple import MQTTClient
import network
import time
from wifi_connect import *

# Test publication e.g. with:
# mosquitto_pub -t AIS2021 -m "LED on"

SERVER="192.168.1.36"
TOPIC="AIS2021"

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()

print("Connected, starting MQTTClient")
c = MQTTClient("umqtt_client", SERVER)
c.connect()

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

print("Waiting for messages on topic 'AIS2021' from MQTT broker")
while True:
    c.wait_msg()
```

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 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

First steps

Register!

Registration will supply you with an

- MQTT user name
- MQTT password

Cayenne
myDevices

Register

First name

Last name

Email

Password

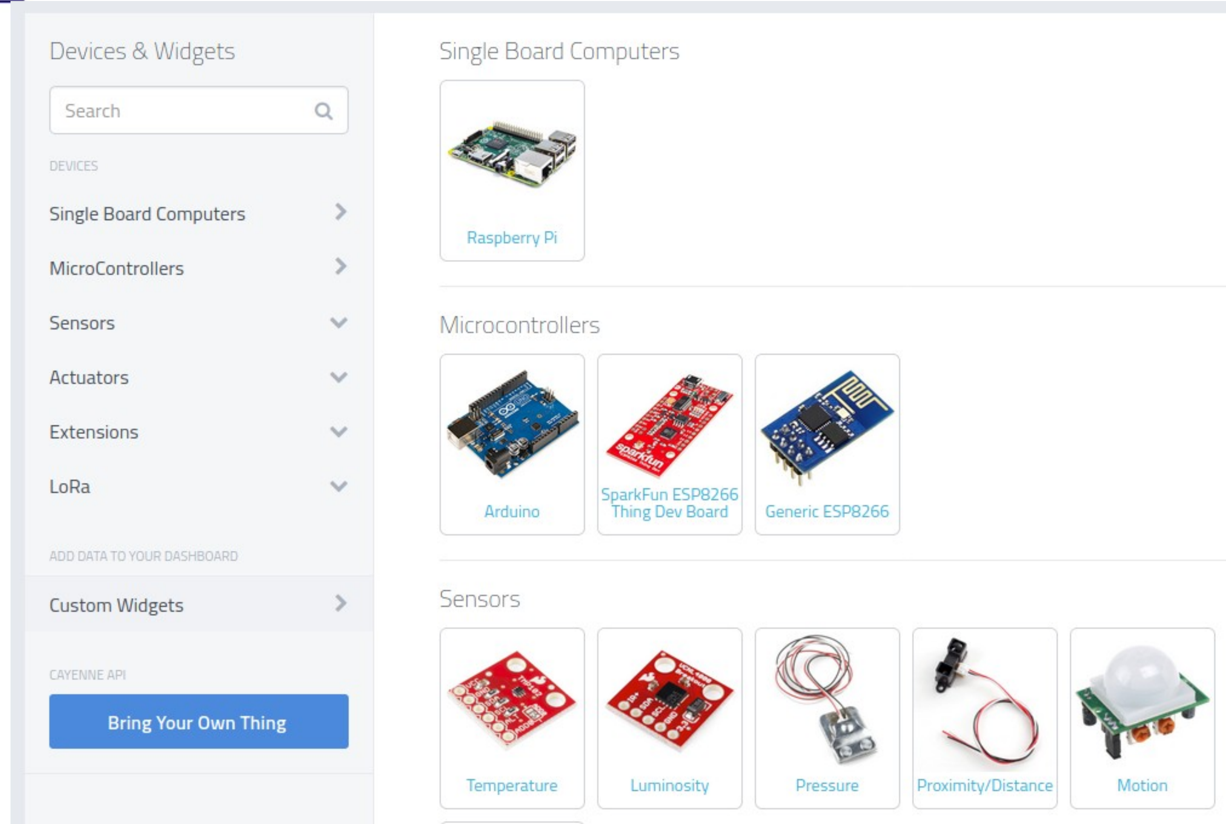
Confirm password

« Back to Login

Register

AFRICA INTERNET SUMMIT'21 ONLINE 31 MAY - 4 JUNE 2021

Create a new project

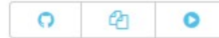


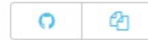
Cavenne credentials

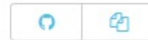
Step 2: Connect your Device

OFFICIAL SDKS

Arduino MQTT

Cayenne MQTT mbed 

Embedded C 

C++ 

Cayenne MQTT Python

Node.JS

View all SDKs on GitHub

NEED HELP?

MQTT API Docs

Ask our community

MQTT USERNAME:

7c70a)10d

MQTT PASSWORD:

32d184 :62876a4

CLIENT ID:

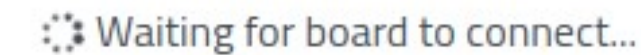
d3e948d0-f209-11ea-883c-638d8ce4c23d

MQTT SERVER: mqtt.mydevices.com

MQTT PORT: 1883

NAME YOUR DEVICE (optional):

New Project



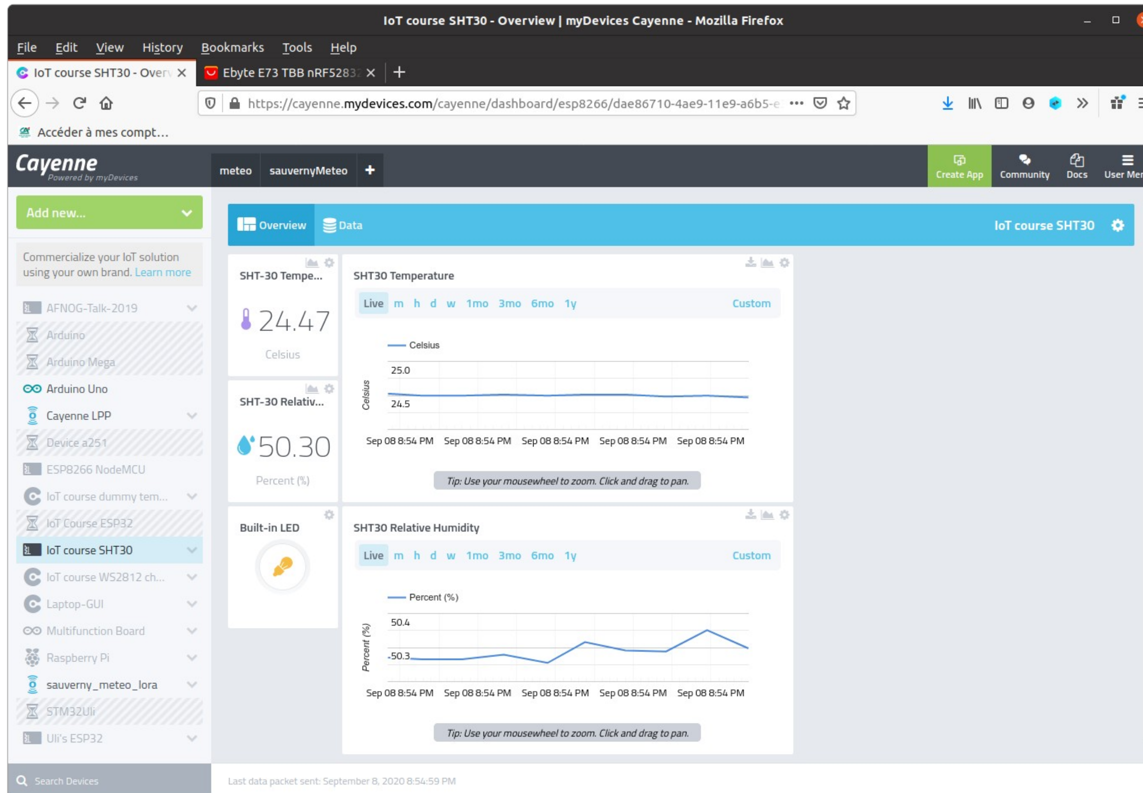
Connecting to Cayenne

```
client = cayenne.client.CayenneMQTTClient()
client.begin(MQTT_USERNAME, MQTT_PASSWORD, MQTT_CLIENT_ID, loglevel=logging.INFO)
print("Successfully connected to myDevices MQTT broker")
# register callback
client.on_message=on_message

def senddata():
  sht30Temperature, sht30Humidity = sht30.getTempAndHumi(clockStretching=SHT3X.NO_CLOCK_STRETCH,
                                                         repeatability=SHT3X.REP_S_HIGH)
  print("Temperature: %6.3f"%sht30Temperature)
  client.celsiusWrite(sht30TempChannel,sht30Temperature)
  client.celsiusWrite(sht30TempChannel+1,sht30Temperature)
  print("Relative humidity: %6.3f"%sht30Humidity + '%')
  client.humidityWrite(sht30HumidityChannel,sht30Humidity)
  client.humidityWrite(sht30HumidityChannel+1,sht30Humidity)
count = 0
while True:
    try:
        client.loop()
        time.sleep(0.5)
        count += 1
        if count == 10:
            count = 0
            senddata()
    except OSError:
        pass
```

AFRICA INTERNET SUMMIT'21 ONLINE 31 MAY - 4 JUNE 2021

Cayenne dash board



What next?

A first course

- The IoT course has been given for the first time at the beginning of 2021
- Student feedback should be taken and evaluated to improve the course
- The slides should be uploaded to the TWiki or better: Full course notes should be written for self-study.
- The students were allowed to take the equipment home for experimentation

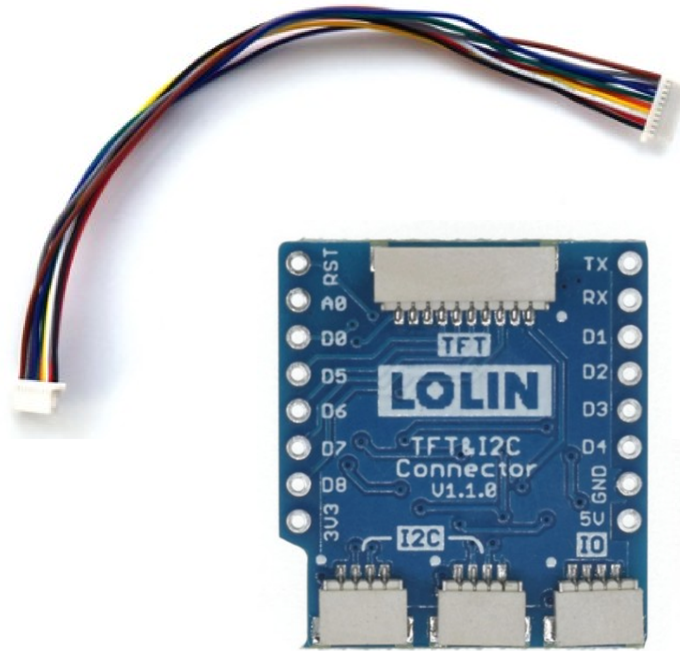
Promote the courses

- Try to get other universities on board and establish collaborations between them to upgrade the course
- Currently there is an effort to bring the course to ‘Université Cheikh Anta Diop, Dakar, Sénégal with documentation in French.
- Create new exercises and/or lab projects

GUI on the IoT node

The Lolin 2.4 " display features a touch panel in addition
It uses an ili9341 graphics controller and has 320x240 pixel resolution

The lvgl GUI library supports this device and makes running of GUI applications on the ESP32 possible



Understanding GUI development

lvgl is written in C but has a MicroPython language binding, where the Python calls are implemented as C code with a Python interface.

There is a MicroPython variant called lv_micropython which has lvgl included in its firmware.

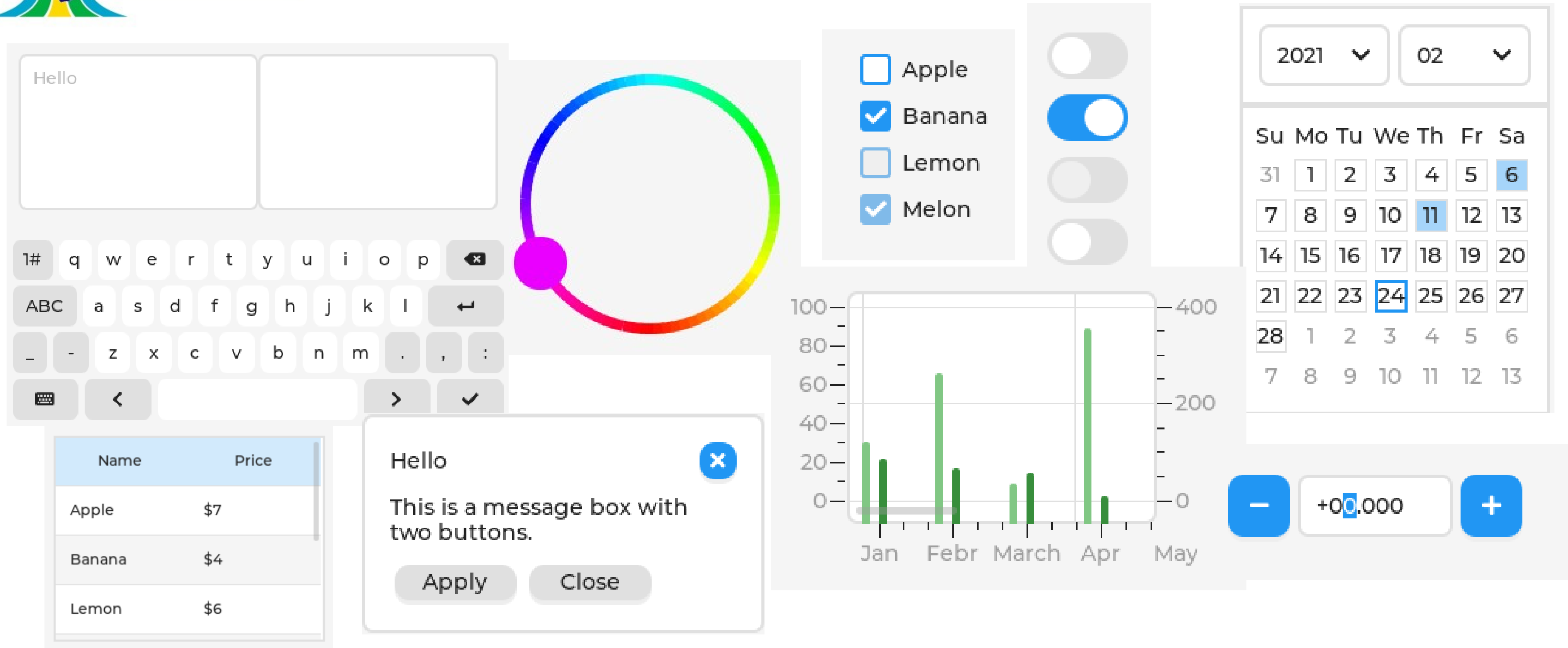
lvgl is available for several graphics controllers and it has a “simulator” using SDL to display its graphics on the unix port of lv_micropython.

Program development can be comfortably done on the PC and only the final program is transferred to the IoT hardware.

The system is so huge that another full semester course is needed to study it.

AFRICA INTERNET SUMMIT'21 ONLINE 31 MAY - 4 JUNE 2021

A few widget examples



Wearable devices?

This is not a watch!

It is a programmable, wearable development device from which you can build a watch (and much more)

TFT_MISO	NULL
TFT_MOSI	IO19
TFT_SCLK	IO18
TFT_CS	IO05
TFT_DC	IO27
TFT_RST	NULL
TFT_BL	IO12

ST7789V 1.54 LCD

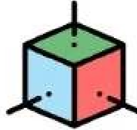
Vibration Motor:GPIO4

T-Watch 2020
LILYGO®

PMU:AXP202

Interrupt	IO35
I2C_SDA	IO21
I2C_SCL	IO22

IR:IO13

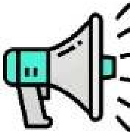


BMA423 AxisSensor

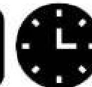
Interrupt	IO39
I2C_SDA	IO21
I2C_SCL	IO22

I2S Class Max98357A

I2S_BCK	IO26
I2S_WS	IO25
I2C_DOUT	IO33



Interrupt RTC:IO37



Touch Board

Interrupt	IO38
I2C_SDA	IO23
I2C_SCL	IO32

T-watch 2020

The t-watch 2020 is again based on the ESP32

It integrates the following devices:

- 240x240 pixel display controlled by an st7789 display controller
- Touch screen controlled by a ft6236 controller
- axp202 power management unit, controlling and measuring battery power of a lithium battery
- bma423 triaxial accelerometer
- RTC clock module pcf8563
- Vibration motor
- max98357a sound system

Available software

Various software bits and pieces are available:

- An Arduino program based on platformio is currently under development as reference software
- lv_micropython works on the t-watch
- The Micropython drivers for the devices are available
- Wasp-OS, a framework to develop watch software and based on Micropython is available
- Much work is needed!

Enough for a full semester course on wearable devices

Cost of the watch: 22.50 Euros

This could be the basis for a course on wearable devices.

- Study and employ the drivers for the different devices
- Study and employ the wasp framework for application development
- Setup of the GUI library with a driver for the st7789 display controller (done in the meantime)
- Example programs for the GUI library e.g. for a settings screen to control display luminosity, active time before sleep, wifi setup …
- A web server running on the watch allowing to control the settings and provide screen dumps
- BlueTooth access
- Control through the IR receiver
- Run the vibration motor and the sound system e.g. for alarms
- Application development: analogue watch, calendar, calculator, settings, battery status ...