

Report Assignment 4

Group Members

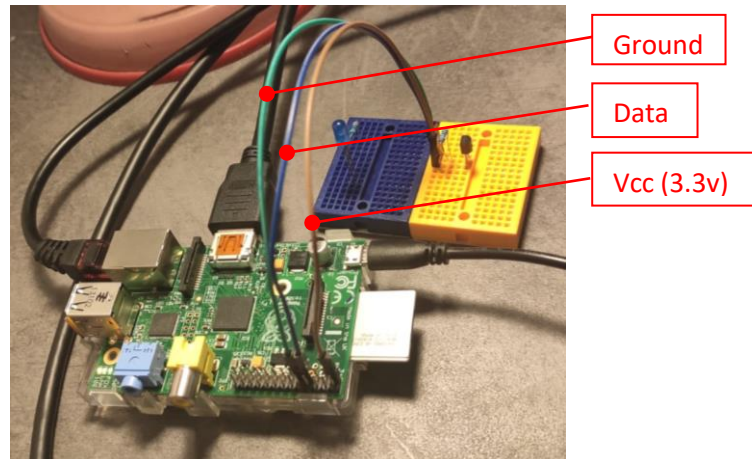
Reindl Hannes	01532129
Sammer Manuel	11903022
Platzer Andreas	11771913

Task 1) Temperature Sensing

Since we missed the deadline for the 3rd assignment, we used the tutorial posted in slack to implement the temperature sensing.

Link: <https://st-page.de/2018/01/20/tutorial-raspberry-pi-temperaturmessung-mit-ds18b20/>

Following the tutorial and the figure, the temperature sensor (DS18B20) was connected to pi depicted below:



After connecting the temperature sensor, the value can be read from terminal. First, we need to know the device ID. Opening the folder in the terminal via “cd /sys/bus/w1/devices/” the connected devices can be seen. Our device has the ID: **28-00000b813dc5**

The command “cat **28-00000b813dc5/w1_slave**” read from the temperature sensor, which can be seen in the next picture:

```

pi@raspberrypi: /sys/bus/w1/devices
Datei  Bearbeiten  Reiter  Hilfe
hwmon      16384  2 wire,raspberrypi_hwmon
uio_pdrv_genirq 16384  0
uio        20480  1 uio_pdrv_genirq
fixed      16384  0
i2c_dev    16384  0
ip_tables  24576  0
x_tables   32768  1 ip_tables
ipv6       446464  24

pi@raspberrypi:~$ cd sys
bash: cd: sys: Datei oder Verzeichnis nicht gefunden
pi@raspberrypi:~$ dir
Desktop  Downloads  Music      Public  thinclient_drives
Documents  MagPi    Pictures  Templates  Videos
pi@raspberrypi:~$ cd /sys/bus/w1/devices/
pi@raspberrypi:/sys/bus/w1/devices$ ls
28-00000b813dc5 w1_bus_master1
pi@raspberrypi:/sys/bus/w1/devices$ cat /w1_slave
cat: /w1_slave: Datei oder Verzeichnis nicht gefunden
pi@raspberrypi:/sys/bus/w1/devices$ cat /28-00000b813dc5/w1_slave
cat: /28-00000b813dc5/w1_slave: Datei oder Verzeichnis nicht gefunden
pi@raspberrypi:/sys/bus/w1/devices$ cat 28-00000b813dc5/w1_slave
9d 01 4b 46 7f ff 03 10 57 : crc=57 YES
9d 01 4b 46 7f ff 03 10 57 t=25812

```

Note that the temperature value is written down as a multiple of 1000. The current temperature is $t = 25.812^{\circ}\text{C}$.

Now to automate the process of reading the temperature, a python script was written, which simple reads via the 1-wire-protocol and slices the given string to only contain the temperature value. The value gets polled every second.

Code file: Task_1/read_temp_python.py

For testing purposes, the temperature was written to the terminal.

```

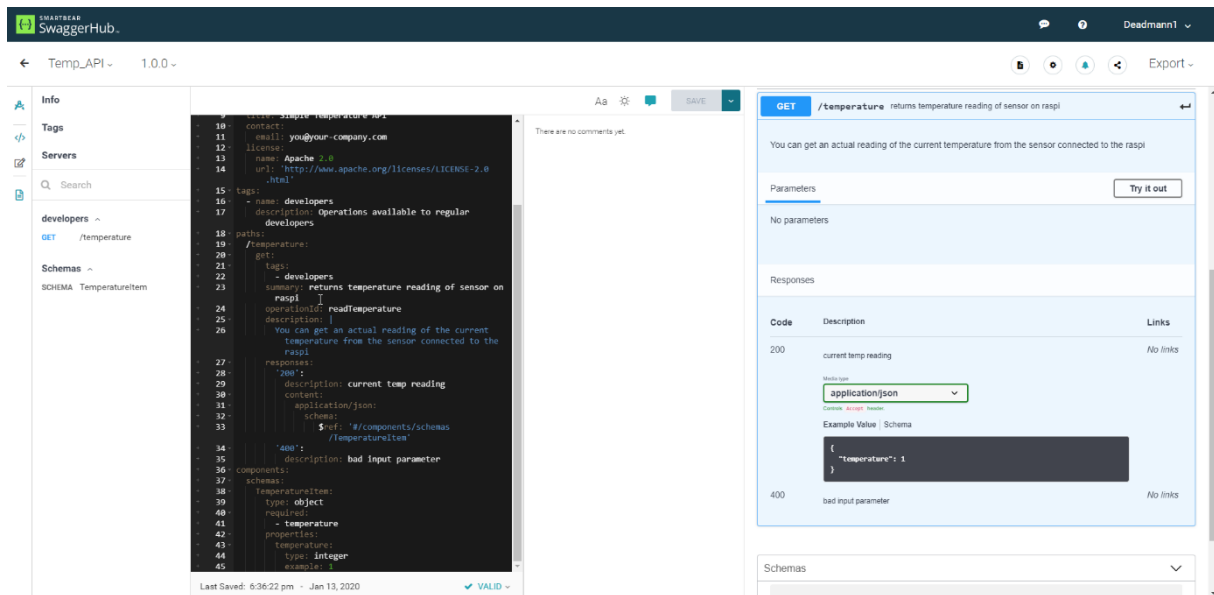
geany_run_script_TW28D0.sh
Datei  Bearbeiten  Reiter  Hilfe
18:36:34 - t = 25 Grad Celsius
18:36:45 - t = 25 Grad Celsius
18:36:56 - t = 25 Grad Celsius
18:37:07 - t = 25 Grad Celsius

```

Task 2) Creating an API

In this task a simple web server should be created which allows to request the temperature via the HTTP protocol.

Going to the recommended website <https://swagger.io/>, and simply changing a given template to fit our purposes the API was created.



We used swagger with Open API 3.0.0 and used the given template “simple template”. Afterwards we deleted the things which were not needed and rewrote a model to our current temperature model. Then we added the path “temperature” which returns an object of the type temperature model.

Using the websites build in simulation function, the API was tested.

Curl

```
curl -X GET "https://virtserver.swaggerhub.com/csn_ddi/Temp_Sensor_API/1.0.0/temperature" -H "accept: application/json"
```

Request URL

```
https://virtserver.swaggerhub.com/csn_ddi/Temp_Sensor_API/1.0.0/temperature
```

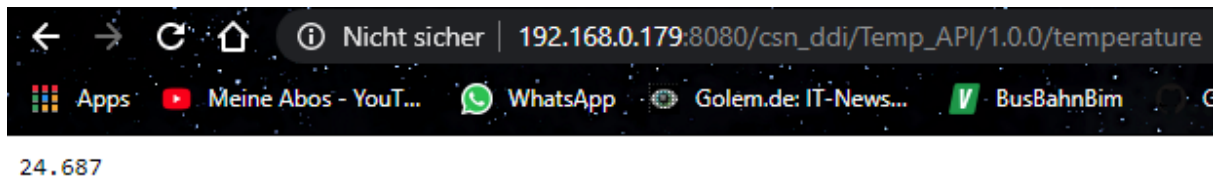
Server response

Code	Details
200	Response body <pre>{ "temperature": 0 }</pre> Download
	Response headers <pre>access-control-allow-credentials: true access-control-allow-headers: X-Requested-With,Content-Type,Accept,Origin access-control-allow-methods: * access-control-allow-origin: * cache-control: no-cache content-encoding: gzip content-length: 43 content-type: application/json; charset=utf-8 date: Sun, 12 Jan 2020 20:14:28 GMT</pre>

The Export button allowed us to download a server stub for python-flask.

From now on, for testing purposes, we used a RPi 3, which was way faster than the RPi 1 and allowed complex tasks (such as moving a window) in much less time. Since this no Bare-Metal programming anymore and the same operating system is used by the two RPi's, it should not make any difference.

Here we see the HTTP Response in the browser. The server can also be seen which is currently working on the request.



```
pi@raspberrypi: ~/Desktop/python-flask-server-generated
Datei Bearbeiten Reiter Hilfe
127.0.0.1 - - [13/Jan/2020 18:28:52] "GET /csn_ddi/Temp_API/1.0.0/temperature HTTP/1.1" 200 -
127.0.0.1 - - [13/Jan/2020 18:28:54] "GET /csn_ddi/Temp_API/1.0.0/temperature HTTP/1.1" 200 -
127.0.0.1 - - [13/Jan/2020 18:28:56] "GET /csn_ddi/Temp_API/1.0.0/temperature HTTP/1.1" 200 -
127.0.0.1 - - [13/Jan/2020 18:28:57] "GET /csn_ddi/Temp_API/1.0.0/temperature HTTP/1.1" 200 -
127.0.0.1 - - [13/Jan/2020 18:28:59] "GET /csn_ddi/Temp_API/1.0.0/temperature HTTP/1.1" 200 -
127.0.0.1 - - [13/Jan/2020 18:29:00] "GET /csn_ddi/Temp_API/1.0.0/temperature HTTP/1.1" 200 -
127.0.0.1 - - [13/Jan/2020 18:29:02] "GET /csn_ddi/Temp_API/1.0.0/temperature HTTP/1.1" 200 -
127.0.0.1 - - [13/Jan/2020 18:29:04] "GET /csn_ddi/Temp_API/1.0.0/temperature HTTP/1.1" 200 -
127.0.0.1 - - [13/Jan/2020 18:29:05] "GET /csn_ddi/Temp_API/1.0.0/temperature HTTP/1.1" 200 -
127.0.0.1 - - [13/Jan/2020 18:29:07] "GET /csn_ddi/Temp_API/1.0.0/temperature HTTP/1.1" 200 -
127.0.0.1 - - [13/Jan/2020 18:29:08] "GET /csn_ddi/Temp_API/1.0.0/temperature HTTP/1.1" 200 -
127.0.0.1 - - [13/Jan/2020 18:29:10] "GET /csn_ddi/Temp_API/1.0.0/temperature HTTP/1.1" 200 -
127.0.0.1 - - [13/Jan/2020 18:29:12] "GET /csn_ddi/Temp_API/1.0.0/temperature HTTP/1.1" 200 -
127.0.0.1 - - [13/Jan/2020 18:29:13] "GET /csn_ddi/Temp_API/1.0.0/temperature HTTP/1.1" 200 -
127.0.0.1 - - [13/Jan/2020 18:29:16] "GET /csn_ddi/Temp_API/1.0.0/temperature HTTP/1.1" 200 -
127.0.0.1 - - [13/Jan/2020 18:29:18] "GET /csn_ddi/Temp_API/1.0.0/temperature HTTP/1.1" 200 -
127.0.0.1 - - [13/Jan/2020 18:29:19] "GET /csn_ddi/Temp_API/1.0.0/temperature HTTP/1.1" 200 -
127.0.0.1 - - [13/Jan/2020 18:34:17] "GET /csn_ddi/Temp_API/1.0.0/temperature HTTP/1.1" 200 -
127.0.0.1 - - [13/Jan/2020 18:34:19] "GET /csn_ddi/Temp_API/1.0.0/temperature HTTP/1.1" 200 -
127.0.0.1 - - [13/Jan/2020 18:34:20] "GET /csn_ddi/Temp_API/1.0.0/temperature HTTP/1.1" 200 -
127.0.0.1 - - [13/Jan/2020 18:34:21] "GET /csn_ddi/Temp_API/1.0.0/temperature HTTP/1.1" 200 -
127.0.0.1 - - [13/Jan/2020 18:34:25] "GET /csn_ddi/Temp_API/1.0.0/temperature HTTP/1.1" 200 -
127.0.0.1 - - [13/Jan/2020 18:34:27] "GET /csn_ddi/Temp_API/1.0.0/temperature HTTP/1.1" 200 -
127.0.0.1 - - [13/Jan/2020 18:34:28] "GET /csn_ddi/Temp_API/1.0.0/temperature HTTP/1.1" 200 -
127.0.0.1 - - [13/Jan/2020 18:34:30] "GET /csn_ddi/Temp_API/1.0.0/temperature HTTP/1.1" 200 -
127.0.0.1 - - [13/Jan/2020 18:34:30] "GET /csn_ddi/Temp_API/1.0.0/temperature HTTP/1.1" 200 -
127.0.0.1 - - [13/Jan/2020 18:34:32] "GET /csn_ddi/Temp_API/1.0.0/temperature HTTP/1.1" 200 -
127.0.0.1 - - [13/Jan/2020 18:34:32] "GET /csn_ddi/Temp_API/1.0.0/temperature HTTP/1.1" 200 -
127.0.0.1 - - [13/Jan/2020 18:34:41] "GET /csn_ddi/Temp_API/1.0.0/temperature HTTP/1.1" 200 -
127.0.0.1 - - [13/Jan/2020 18:34:43] "GET /csn_ddi/Temp_API/1.0.0/temperature HTTP/1.1" 200 -
127.0.0.1 - - [13/Jan/2020 18:34:45] "GET /csn_ddi/Temp_API/1.0.0/temperature HTTP/1.1" 200 -
```

This server was only tested in LAN, because port forwarding was not set up for public use.

Task 3) CoAP Protocol

First the plugin Cu4Cr was installed to enable google chrome sending coap packets and allowed us to debug the code. Link: <https://github.com/mkovatsc/Copper4Cr>

For the CoAP implementation a GitHub repository was used named aiocoap.

Link: <https://github.com/chrysn/aiocoap>

Following the guided tour, the aiocoap package was downloaded onto the RPi using

```
pip3 install --upgrade "aiocoap[all]"
```

and starting the server from the root folder by

```
./server.py
```

The server.py file already has some defined resources, which can be found via the CoAP .well-known/core. Writing this command line into the terminal

```
./aiocoap-client coap://localhost/.well-known/core
```

Shows the following resources:

```
</time>; obs, </.well-known/core>; ct=40, </other/separate>,
</other/block>
```

The resource time and the resource other, which has 2 sub resources. The assignment explicitly states to make sure the server can use the OBSERVER method and implement.

Within the file "server.py" the resources can be defined. Rewriting the time resource, which was defined as an observable resource, and copying the already written program to read the temperature from Task 1, this task was close to being fulfilled. Simply starting the server again and accessing it through the Cu4Cr plugin for Google Chrome we saw this:

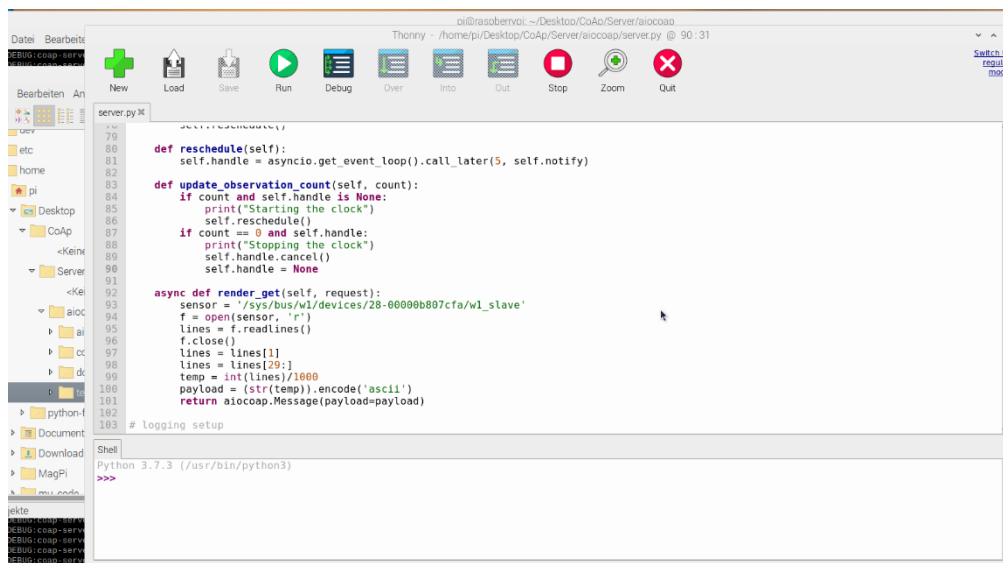
The screenshot shows the Chrome DevTools interface with the Cu4Cr plugin. The 'Status' tab is active, displaying a CoAP message log. The log shows a series of messages between the client and the server. A red box highlights a message with the payload 'Temperature'. Below the log, the 'CoAP Message Log' table is visible, showing the details of the messages. The 'Via Observer Method' text is overlaid on the log.

Time	Type	Code	MID	Token	Options	Payload
20:04:44	ACK	EMPTY	14125	0x2ca33fc	Observe: 11	
20:04:44	CON	2.05 Content	14125	0x2ca33fc		
20:04:38	ACK	EMPTY	14124	0x2ca33fc	Observe: 10	
20:04:38	CON	2.05 Content	14124	0x2ca33fc		
20:04:33	ACK	EMPTY	14123	0x2ca33fc	Observe: 9	
20:04:33	CON	2.05 Content	14123	0x2ca33fc		
20:04:28	ACK	EMPTY	14122	0x2ca33fc	Observe: 8	
20:04:28	CON	2.05 Content	14122	0x2ca33fc		
20:04:23	ACK	EMPTY	14121	0x2ca33fc	Observe: 7	
20:04:23	CON	2.05 Content	14121	0x2ca33fc		
20:04:18	ACK	EMPTY	14120	0x2ca33fc	Observe: 6	
20:04:18	CON	2.05 Content	14120	0x2ca33fc		
20:04:13	ACK	EMPTY	14119	0x2ca33fc	Observe: 5	
20:04:13	CON	2.05 Content	14119	0x2ca33fc		
20:04:08	ACK	EMPTY	14118	0x2ca33fc	Observe: 4	
20:04:08	CON	2.05 Content	14118	0x2ca33fc		
20:04:03	ACK	EMPTY	14117	0x2ca33fc	Observe: 3	
20:04:03	CON	2.05 Content	14117	0x2ca33fc		

And the messages displayed on the terminal which started the server.

```
pi@raspberrypi: ~/Desktop/CoAp/Server/aiocoap
DEBUG:coap-server:Exchange removed, message ID: 14123.
DEBUG:coap-server:Sending message <aiocoap.Message at 0x751d2990: Type.CON 2.05 Content (MID 14124, token 02ca33fc) remote <UDPEndpointAddress 192.168.0.15:56462 with local address>, 1 option(s), 6 byte(s) payload>
DEBUG:coap-server:Exchange added, message ID: 14124.
DEBUG:coap-server:Incoming message <aiocoap.Message at 0x751d2430: Type.ACK EMPTY (MID 14124, token 02ca33fc) remote <UDPEndpointAddress 192.168.0.15:56462 with local address>>
DEBUG:coap-server:New unique message received
DEBUG:coap-server:Exchange removed, message ID: 14124.
DEBUG:coap-server:Sending message <aiocoap.Message at 0x751d2a90: Type.CON 2.05 Content (MID 14125, token 02ca33fc) remote <UDPEndpointAddress 192.168.0.15:56462 with local address>, 1 option(s), 4 byte(s) payload>
DEBUG:coap-server:Exchange added, message ID: 14125.
DEBUG:coap-server:Incoming message <aiocoap.Message at 0x751d2990: Type.ACK EMPTY (MID 14125, token 02ca33fc) remote <UDPEndpointAddress 192.168.0.15:56462 with local address>>
DEBUG:coap-server:New unique message received
DEBUG:coap-server:Exchange removed, message ID: 14125.
DEBUG:coap-server:Sending message <aiocoap.Message at 0x751d2b90: Type.CON 2.05 Content (MID 14126, token 02ca33fc) remote <UDPEndpointAddress 192.168.0.15:56462 with local address>, 1 option(s), 6 byte(s) payload>
DEBUG:coap-server:Exchange added, message ID: 14126.
DEBUG:coap-server:Incoming message <aiocoap.Message at 0x751d2630: Type.ACK EMPTY (MID 14126, token 02ca33fc) remote <UDPEndpointAddress 192.168.0.15:56462 with local address>>
```

This is the function which computes the response, changed to deliver our temperature sensor value.



The screenshot shows the Thonny Python IDE interface. The top toolbar includes icons for New, Load, Save, Run, Debug, Over, Info, Out, Stop, Zoom, and Quit. The left sidebar shows a file explorer with a tree view containing folders like 'etc', 'home', 'Desktop', 'CoAp', 'Server', 'aioc', 'ai', 'cc', 'dk', 'python-f', 'Document', 'Download', 'MagPi', and 'my code'. The main editor window displays a Python script named 'server.py' with the following code:

```
79
80
81 def reschedule(self):
82     self.handle = asyncio.get_event_loop().call_later(5, self.notify)
83
84 def update_observation_count(self, count):
85     if count and self.handle is None:
86         print("Starting the clock")
87         self.reschedule()
88     if count == 0 and self.handle:
89         print("Stopping the clock")
90         self.handle.cancel()
91         self.handle = None
92
93 async def render_get(self, request):
94     sensor = '/sys/bus/w1/devices/28-00000b807cfa/w1_slave'
95     f = open(sensor, 'r')
96     lines = f.readlines()
97     f.close()
98     lines = lines[1]
99     lines = lines[29:]
100     temp = int(lines)/1000
101     payload = (str(temp)).encode('ascii')
102     return aiocap.Message(payload=payload)
103
104 # Logging setup
```

At the bottom, a shell window shows the Python version: Python 3.7.3 (/usr/bin/python3).

Benefits of CoAP:

Constrained Application Protocol is a lightweight protocol which allows reliable data transfer (CON, ACK) using UDP. It also has significantly less overhead than HTTP which allows efficient transfer for small packet sizes. Using the tutorial GitHub, it was really easy to setup, but there is way less information and examples easily accessible. The CoAP protocol was not implemented naturally into our browser yet, so additional software is required to test and debug it.

Task 4) MQTT

For the last task a messaging protocol out of {MQTT, AMQP and XMPP} should be implemented. After finding a useful tutorial to MQTT which looked tailored to our problem, we simply choose MQTT. The required ecosystem is only python interpreter and mosquitto which is described below.

Link: <https://tutorials-raspberrypi.com/raspberry-pi-mqtt-broker-client-wireless-communication/>

First, we installed mosquitto, which is a MQTT library. After installing it an MQTT server is already running and listens to the MQTT Port number.

```
sudo apt-get install -y mosquitto mosquitto-clients
```

Then we programmed, a publisher and subscriber clients. The publisher publishes data to the broker, which informs the clients which subscribed to the channel and forwards it.

The publish and subscriber programs in python:

```
1  #!/usr/bin/python
2  import paho.mqtt.publish as publish
3  import time
4
5  MQTT_SERVER = "192.168.0.179"
6  MQTT_PATH = "test_channel"
7
8
9  while(1):
10     sensor = '/sys/bus/wl/devices/28-00000b807cfa/wl_slave' #change to our device id
11     f = open(sensor, 'r')
12     lines = f.readlines()
13     f.close()
14     lines = lines[1]
15     lines = lines[29:]
16     temp = int(lines)/1000
17
18     publish.single(MQTT_PATH, "Current Temperature [" + str(temp) + "] Grad C" , hostname=MQTT_SERVER)
19     time.sleep(5)
20
```

```
1  #!/usr/bin/python
2  import paho.mqtt.client as mqtt
3
4  MQTT_SERVER = "localhost"
5  MQTT_PATH = "test_channel"
6
7
8  # The callback for when the client receives a CONNACK response from the server.
9  def on_connect(client, userdata, flags, rc):
10     print("Connected with result code "+str(rc))
11
12     # Subscribing in on_connect() means that if we lose the connection and
13     # reconnect then subscriptions will be renewed.
14     client.subscribe(MQTT_PATH)
15
16
17  # The callback for when a PUBLISH message is received from the server.
18  def on_message(client, userdata, msg):
19     print(msg.topic+" "+str(msg.payload))
20     # more callbacks, etc
21
22
23  client = mqtt.Client()
24  client.on_connect = on_connect
25  client.on_message = on_message
26  client.connect(MQTT_SERVER, 1883, 60)
27
28  # Blocking call that processes network traffic, dispatches callbacks and
29  # handles reconnecting.
30  # Other loop*() functions are available that give a threaded interface and a
31  # manual interface.
32  client.loop_forever()
33
```

Both of the clients were programmed quite simple, because they are only used to demonstrate the usage of MQTT with the RPi and temperature sensor. The publisher publishes the temperature sensors data every 5 seconds.