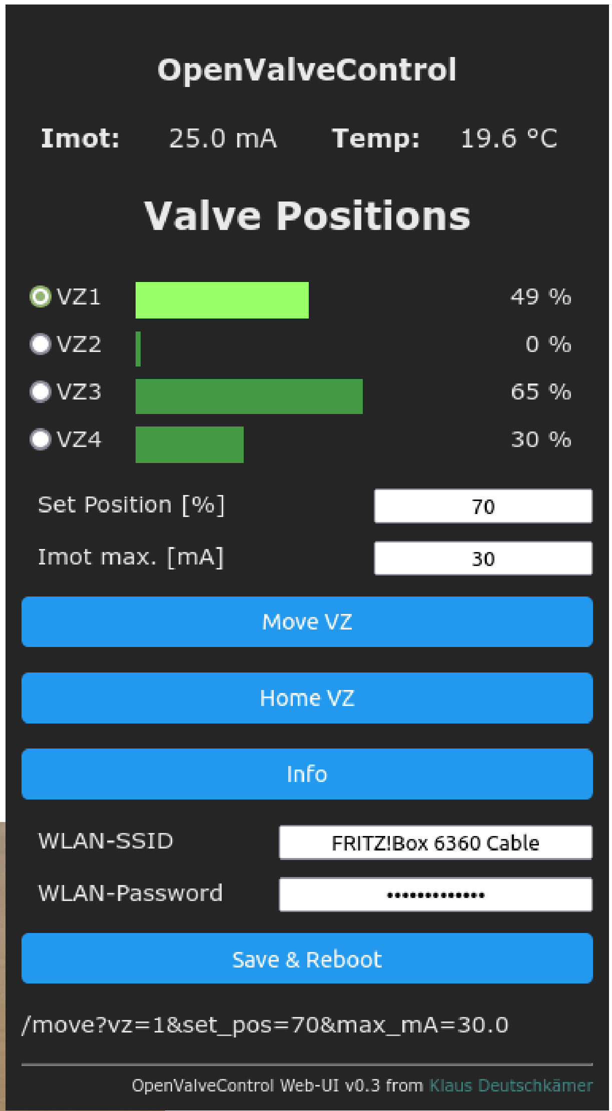
OpenValveControl

User Manual (DRAFT)



1 Initial Setup

Prerequisites

- The board is assembled with all the required components and
- The PIC controller has been flashed (e.g. with a PICkit4) with the latest firmware.

1.1 Flashing the "ESP8266 D1-mini" for the first time

There are many ways to flash the ESP for the first time. They all involve connecting the board via the USB interface. The appropriate driver for the USB interface IC installed on the D1 mini may need to be installed first.

Under Linux, the command 'lsusb' can be used to check this. The name of the USB port can be found in the /dev/ directory:

```
~$ lsusb
...
Bus 002 Device 006: ID 1a86:7523 QinHeng Electronics CH340 serial converter
...

~$ ls -l /dev/ttyUSB*
crw-rw---- 1 root dialout 188, 0 Dez 15 14:00 /dev/ttyUSB0
~$
```

1.1.1 Arduino IDE (2.2.1)

Board: LOLIN(WEMOS) D1 mini
FlashSize: 4 MB (FS: 1MB, OTA: ~1019KB)

1.1.2 tasmotizer

The "Tasmotizer" is a Python application and must therefore be started via Python (currently python3). The program *tasmotizer.py* can be downloaded from github, the path must be adapted to the location:

```
~$ python3 .local/bin/tasmotizer.py
esptool.py v2.8
Serial port /dev/ttyUSB0
Connecting....
Chip is ESP8266EX
Features: WiFi
Crystal is 26MHz
Uploading stub...
Running stub...
Stub running...
Configuring flash size...
Auto-detected Flash size: 4MB
Erasing flash (this may take a while)...
Chip erase completed successfully in 3.1s
Compressed 383456 bytes to 276588...
Wrote 383456 bytes (276588 compressed) at
0x00000000 in 24.4 seconds (effective 126.0
kbit/s)...
Hash of data verified.

Leaving...
Hard resetting via RTS pin...
```

In the graphical user interface, simply select the USB port and the binary program (ESP-ValveControl.ino.bin) with "Open" (navigate to the storage location) and press "Tasmotize!". The rest happens automatically.

1.1.3 Via command line using "esptool.py"

The Python tool "esptool.py" is usually part of the Arduino IDE, but can also be downloaded and installed from "Espressiv" if Phyton is not already installed on the computer. The path information for esptool.py and the ESP-ValveControl binary file must of course be adapted again:

```
~$ python3 ~/.arduino15/packages/esp8266/hardware/esp8266/3.1.2/tools/esptool/esptool.py -p /dev/ttyUSB0
write_flash 0x00000 ~/.git/OpenValveControl/ESP-ValveControl/build/esp8266.esp8266.d1_mini_clone/
ESP-ValveControl.ino.bin
esptool.py v3.0
Serial port /dev/ttyUSB0
Connecting....
Detecting chip type... ESP8266
Chip is ESP8266EX
Features: WiFi
Crystal is 26MHz
MAC: 24:a1:60:3a:bb:83
Uploading stub...
Running stub...
Stub running...
Configuring flash size...
Compressed 383456 bytes to 276588...
Wrote 383456 bytes (276588 compressed) at 0x00000000 in 24.4 seconds (effective 125.9
kbit/s)...
Hash of data verified.

Leaving...
Hard resetting via RTS pin...
~$
```

1.2 Configuration of the WiFi

If the flashing was successful, the following message[1] should appear in the first two lines of the **OLED display** (as an indication that the ESP is working as an access point):

```
OVC-Access-Point
OVC-password
```

Shortly afterwards, the message changes and the set IP address is displayed:

```
IP: 192.168.4.1
21.1 C         0.0 mA
```

1 If the OLED display does not work, the hardware and especially the connection (and polarity) of the OLED must be checked.

Now you can connect via WiFi to this access point "OVC-Access-Point" and the password "OVC-password" (case sensitive!) and enter the displayed IP address in a browser.

The following message should appear in your browser:

http://192.168.4.1

The file "fs.html" must be selected from the ESP directory with "Browse" and then copied to the ESP file system by clicking the button "Upload".

The page then updates itself and shows the ESP8266 file manager with the content "fs.html", the file that has just been uploaded:

http://192.168.4.1/fs.html

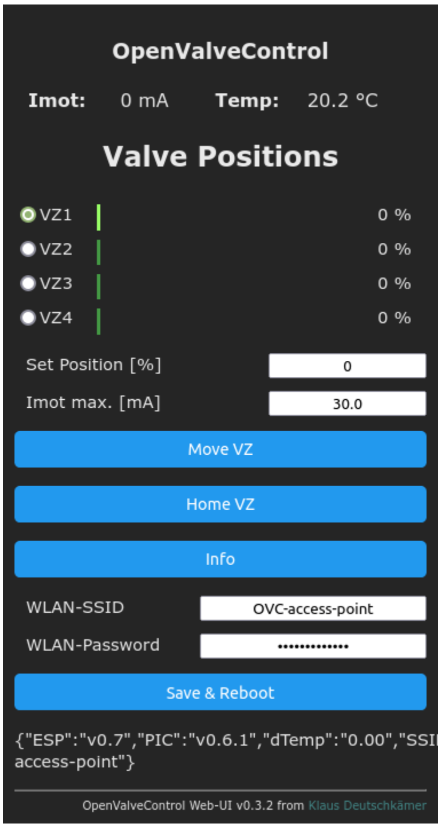
In the same way, we then load the files

"**style.css**" (css formatting for fs.html) and
"**index.html**" (the ESP Valve Control user interface)

to the ESP file system:

http://192.168.4.1/fs.html
(with style.css and index.html)

Now we can display the graphical user interface of ESP-ValveControl (it is loaded from the ESP file system):



← Enter here the WiFi SSID of your home network,
← and the WiFi password.
← Press this button to save the settings.

1.3 Reboot and Login to the home network

After the reboot, the ESP attempts to log into the home network with the data entered. If this is not successful within about 2 minutes, the access point is activated again.

i If problems occur, please check whether your router is configured so that only known devices are allowed to connect. Change this setting temporarily if necessary.

When restarting, the SSID you entered for your home network is displayed in the top line of the OLED. This line flashes while the connection is being established.

The version of the ESP8266 software is displayed in the second line:

Example:

```
FRITZ!Box 6390 Cable
ESP v0.7
```

After successful login, the display changes and the specified IP address is displayed. The motor current and temperature[2] are displayed in the second line.

Example:

```
IP: 192.168.2.131
19.5 C         0.0 mA
```

You can now connect to the home network and enter the IP address displayed. The ESP-ValveControl graphical user interface should then be displayed again, but now with the SSID of your home network.

1.4 Additional settings (ovc.ini)

After entering and saving the WiFi parameters, an "ovc.ini" file is created in which the settings are saved permanently. Even after a firmware update, the data is normally still available unless you specify that the flash is to be completely erased (e.g. with Tasmotizer: "Erase before flashing").

However, a loss is not a problem, the installation described can be repeated at any time.

As a further alternative, you can also download the „ovc.ini“ file to your computer and save it or change it with a text editor and add further information (e.g. for MQTT).

You can also duplicate the settings on several devices or skip entering the WiFi settings in the access point and upload a prepared „ovc.ini“ file using the Filesystem Manager. After the reboot, all settings are then immediately active.

2 If no DS18B20 is connected, -128.2 C is displayed.

An example file **ovc.ini** is available in github in the folder "/ESP-ValveControl/ovc.ini":

```
## ovc.ini
# As default (or when it cannot connect within 2 minutes to the WiFi using the current
# settings), the ESP creates an access point with the credentials shown below.
# Please change the SSID and PSK to meet the settings of your WiFi, then upload the file into
# the ESP's file system, which can be accessed in your browser with the URI <local IP>/fs.html

# WLAN credentials
#SSID = Your SSID
#PSK = Your WiFi password
SSID = OVC-access-point
PSK = OVC-password

# MQTT: server (IP) and token to publish data every period (seconds) as "<mqtt_prefix>/status"
MQTT_HOST   = 192.168.2.72
MQTT_PREFIX = OVC-1
MQTT_PERIOD = 900

# ValveZone aliases (not used yet)
VZ1 = ESS
VZ2 = KÜ
VZ3 = BAD
VZ4 = WZ1

# Adjust temperature sensor
dTemp = -0.5
```

It means:

SSID	The name of your home network
PSK	Password of your home network
MQTT_HOST	IP-Adresse of your MQTT server.
MQTT_PREFIX	Prefix for the sent data (to differentiate between several identical devices (all data is sent with the MQTT token "<mqtt_prefix>/status").
MQTT_PERIOD	Time interval in seconds for sending the MQTT data ("publishing").
VZ1 VZ2 VZ3 VZ4	Alternative identifiers for the valve zones (not yet used)
dTemp	Adjustment value for the DS18B20 temperature sensor

1.5 First driving trials

i A connected motor without a valve can move beyond its end position if it does not experience any resistance when moving. This may result in damage to the mechanics. The control unit is operated at your own risk.

1.5.1 Delivery status of the drives

The following photo shows an actuator in its as-delivered condition. You can see that the round plunger (in the center) is offset a few millimeters inwards. This corresponds to the OPEN valve position on my Rotex heating circuit manifold and makes it easier to install the motor.

When referencing (homing), the actuator moves the plunger out, so when installed it pushes the valve pin in, which corresponds to the CLOSED valve position (on my Rotex heating circuit manifold). When the valve is closed, the plunger presses against the spring force of the valve pin, which is why installing the motor in motor position CLOSED is not recommended.

i This may be different with other devices!
OpenValveControl does not currently allow the direction of rotation to be changed!

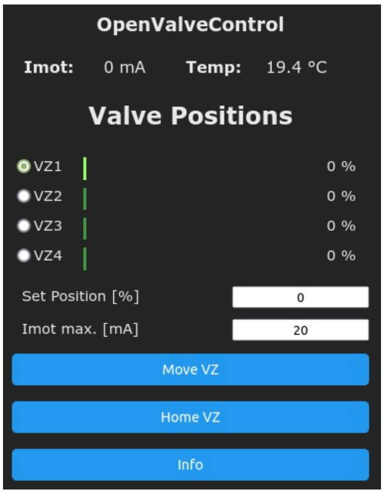
As-delivered condition

1.5.2 Checking the reference run

First of all, you should carry out a reference run - WITHOUT installing the motor in the heating circuit manifold. The easiest way to do this is on the desk. Connect the motor to the left-hand socket, for example, as shown in the next picture. This corresponds to valve zone VZ1 in the user interface.

Connection of a motor to VZ1
to test the reference run.

- Now select „VZ1“ in the user interface (by clicking with the mouse)
- Then enter the (small) value 20 for „Imot max [mA]“
- Now click on the "HOME VZ" button.



The motor should then move the plunger "out" and display the motor current during this process. The LED also lights up as an indicator.

The reference run is terminated when the motor current exceeds the entered limit value. Valve position "CLOSED" is then assumed and the actual position is set to 0 %.

You can easily simulate this with such a low limit value by pressing against the plunger with a suitable pin or with your thumb while it is being moved out.

Caution! The plunger must not move out so far that the guide lug comes completely out of the groove, otherwise the plunger can no longer be moved in without manual support.

The reference run is therefore aborted after approx. 2 minutes, even if the current limit value is not exceeded. The LED goes out.

In case of problems, simply unplug the drive!

If this "reference run" has worked, you can specify a target position of e.g. 20 % for the motor and move to this position using the "Move VZ" button.

If the motor stops prematurely, you can increase the current limit value in small steps. Proceed in a similar way once you have installed the motor. For comparison: I have entered a current limit value of 50 mA for my Rotex heating manifold.

2 Controlling with node-RED[3]

Node-RED is the "Swiss army knife" of the Internet of Things (IoT) and the ideal tool when it comes to exchanging data between sensors and actuators. In this chapter, we want to control *OpenValveControl* with a node-RED „flow“.

In node-RED, "flows" refer to the graphical representations and data connections of "nodes". A node can, for example, represent an interface to MQTT and is represented as a block (node) with inputs and outputs. The properties of these nodes are defined via parameters (e.g. the IP address of the MQTT server).

You can install node-RED on a RaspberryPi, for example, so that it is permanently available (24/7). On the same RaspberryPi, a mosquitto server can run in parallel as an MQTT host and other applications (e.g. grafana).

I have been running this configuration on a 3rd generation Pi for years without any problems.

2.1 Adjusting the valves using node-RED

2.1.1 Task definition

As we have seen, we can adjust our valves via the web interface. To do this, we need to know the IP address of the web UI.

Then we can select the ValveZone (the actuator or valve), enter the target position (0 to 100%), enter the maximum motor current in milliamperes and press the "Move" button.

This sends an "http GET" command to the ESP, which contains the parameters mentioned. The command is displayed in plain text at the bottom of the Web UI screen.

Example

Valve 1 (vz=1) should be moved to 26%. If the motor current exceeds the value 30 mA, the process should be aborted (blocking protection):

```
/move?vz=1&set_pos=26&max_mA=30.0
```

We can achieve exactly the same thing by sending the GET command via a web browser on our LAN. We just need to prefix it with the IP address of our control device (the IP address will be displayed on the OLED after the ESP has successfully logged into the home network):

```
http://192.168.2.131/move?vz=1&set_pos=26&max_mA=30.0
```

3 **node-RED** is a flow-based programming tool, originally developed by IBM’s Emerging Technology Services team (https://emerging-technology.co.uk/) and now a part of the OpenJS Foundation (https://openjsf.org/).

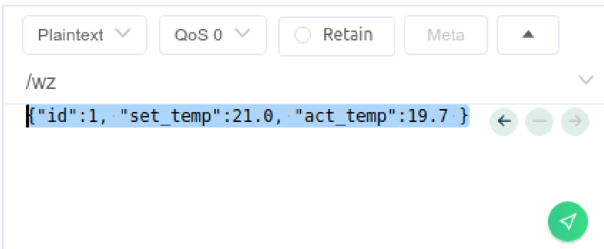
2.1.2 Specifying a setpoint

In order to solve the task in a practical way, we first want to solve the setpoint specification, as we don't want to make all the settings every time with the smartphone.

We assume that we have a temperature sensor with display in our living room for a reasonable individual room control as well as a possibility for a setpoint specification. This could be realized with a small ESP or with a commercially available industrial product (display of actual and setpoint temperature, 2 buttons ⇅ for the setpoint).

This ESP could now send the described GET command directly to our controller via WiFi and, for example, control the valve to 80% if it is too cold in the room or to 20% if it is too warm (simple "two-point" control). In addition, each thermostat must of course be informed of the valve assignment, the maximum current and possibly the control algorithm. However, there is other information that is useful for control, e.g. outdoor temperature, solar radiation, night setback, extended absence (vacation) and others. It therefore makes sense to only report the actual and setpoint temperatures of the respective room and to manage all rooms centrally, e.g. with node-RED.

To do this, we send a sensor ID, the actual temperature and setpoint value to an MQTT server (also known as a broker, e.g. our mosquitto). Until we have programmed the sensor, we can simulate this with the MQTTX[4] tool:



If we are connected to the MQTT broker (e.g. as test@raspi3:1880), we can define a **topic** (e.g. "/wz") and send the entered **payload** to the MQTT broker by clicking on the green arrow. The data can be changed as required.

2.1.3 flow: "Subscribe" thermostat values from our MQTT broker

In node-RED, we now place an "mqtt in" node in our "flow". Double-clicking on the node opens the "Properties" window, in which we enter the following data:

I have entered "localhost:1883" as the **server** because both node-RED and mosquitto (the MQTT broker) run on the same RaspberryPi. Otherwise, the corresponding IP address of the MQTT server must be entered instead of "localhost".

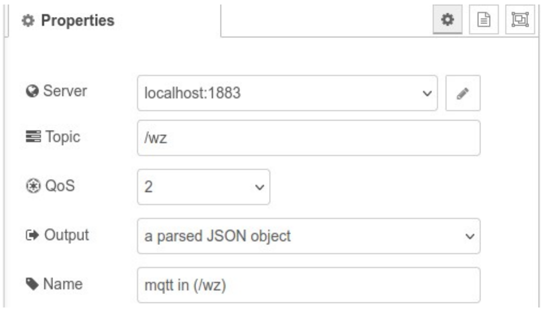
4 „Your All-in-One MQTT Client Toolbox“ (https://mqttx.app/). MQTTX is a free open-source project available under the Apache-2.0 LICENSE.

We can assign the **topic** "/wz" (deutsch: WohnZimmer = Living Room) in the thermostat (or temperature sensor). We have entered this in MQTTX as a test and it is used to differentiate between the various rooms or temperature sensors.

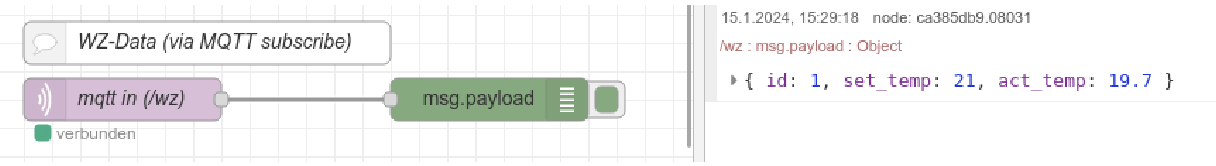
Quality of Service "**QoS**" describes the increasing degree of reliability in the delivery of messages, the highest value is 2.

As **output**, we want a "parsed JSON object", i.e. the same message that was sent from the sensor to the MQTT broker.

The **name** can be chosen at will and is primarily used to make the flow clearer.



We now add a "debug" node so that we can check the output of the "mqtt in" node. As soon as we send a data record to the MQTT broker via MQTTX, the subscribed data is sent to node-RED and is available at the output of the "mqtt in" node:



2.1.4 flow: Convert temperature difference into valve position

We now know the room and the setpoint and actual temperature and can now make a decision for the valve position.

To simplify matters, we take the room ID as the valve zone (vz) and program the two-point controller described above, which we want to formulate in JavaScript as follows:

If the deviation is more than 0.5 degrees, we switch the valve closed (20%) or open (80%), otherwise we leave it as it is.

Note: The simplified assumption is that the flow temperature is weather-compensated and we only make a fine adjustment with a valve. Of course, we can also program an arbitrarily complicated control algorithm here!

For this purpose, node-RED has the "function" node, in which JavaScript can be executed almost arbitrarily. A "message" is accepted as input, which can be modified (almost) at will by the program and output at one or more outputs. The instruction `return (x)` is used for this purpose, whereby x can also be `null`, in which case no output is generated. We use this to avoid executing an adjustment process if the temperature is reasonably correct.

At the start of the JavaScript program, our message has the following values, for example:

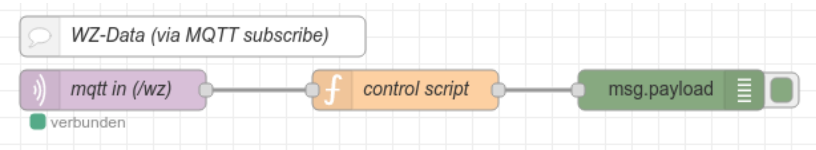
```
msg.payload = {"id":1,"set_temp":21,"act_temp":19.7}
```

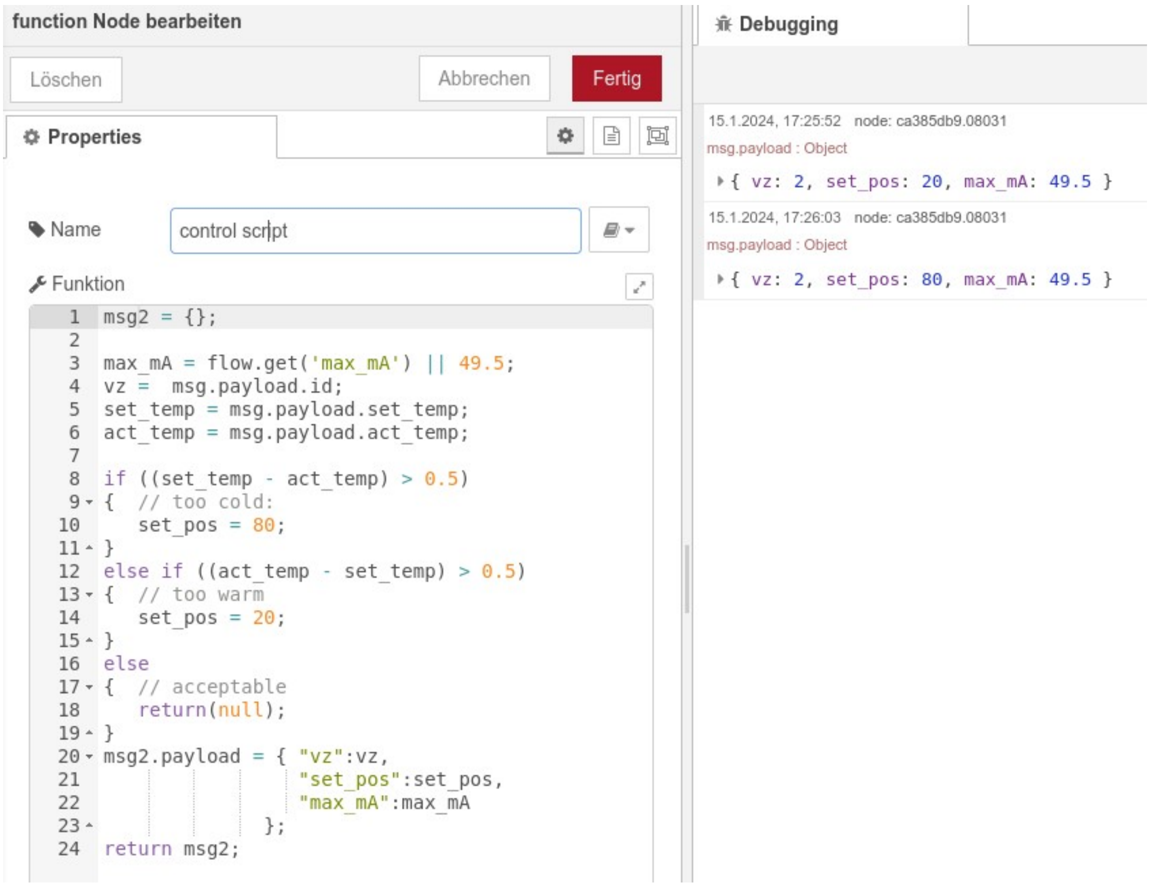
At the output, we want to have the appropriate parameters for the "http GET" request:

```
msg2.payload = {"id":1, "set_pos":80, "max_mA":50.0}
```

All we have to do is generate a new message msg2. We temporarily take the missing value for "max_mA" from a variable that was somehow saved in the flow with the instruction `flow.set("max_mA", <value>)`, for example.

Our JavaScript program could then look like this:





The 2 outputs in the debugger were the responses to the following 3 messages:

```
{"id":2, "set_temp":18.0, "act_temp":19.7 }         // too warm
{"id":2, "set_temp":22.0, "act_temp":19.7 }         // too cold
{"id":2, "set_temp":20.0, "act_temp":19.7 }         // acceptable
```

This is a very good way of testing that no output is produced on the third input as intended.

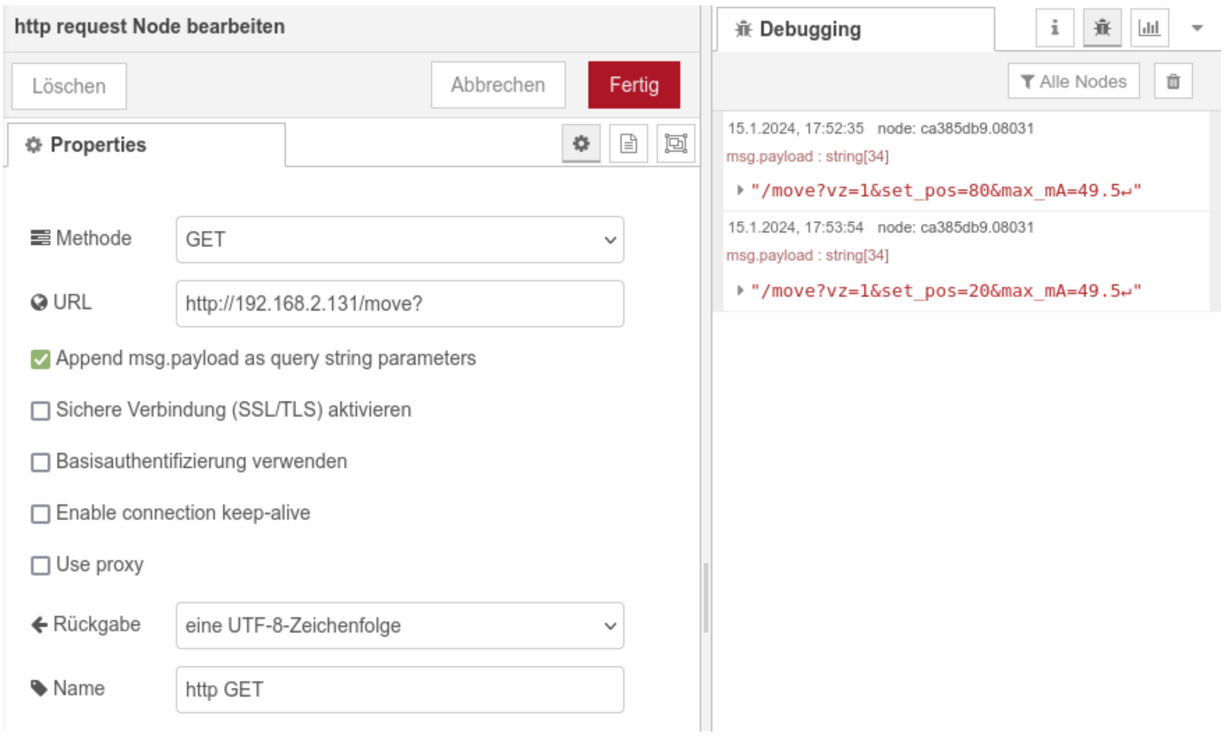
Note

To prevent an error message from occurring when reading the max_mA variable or an invalid value being used if no value has (yet) been saved, a logical AND has been appended, which assigns a default value in this case:

```
max_mA = flow.get('max_mA') || 49.5;
```

2.1.5 flow: Output the query parameters as http GET request

There is another ("network") node for sending the "http request". We place it at the output of the JavaScript "function" node and connect both nodes. We move the debug node to the output of the http request. Then we fill in the properties as follows:



The debug output when publishing our thermostat values shows that all the necessary parameters from the message were appended to the URL with the addition "/move?" for the move command.

Accordingly, the commands "home" or "info" or "status" can also be implemented and further processed in node-RED.

Of course, you can delete or deactivate the debug node if you no longer need it.

This completes the task and we have our finished flow: