Martin Weissenböck

Was ist ... MQTT?

MQTT ist ein Protokoll, das zur Übertragung geringer Datenmengen, zum Beispiel von Messstationen, entwickelt wurde. Ursprünglich war *MQTT* die Abkürzung für *Message Queueing Telemetry Transport*, seit der Version 3.1.1 gilt *MQTT* als Eigenname. Die aktuellen Versionen sind 3.1.1 und 5. Geringe Datenmengen, keine extremen Anforderungen an die Übertragungsgeschwindigkeit, einfach zu implementieren: deshalb ist MQTT für das *Smart Home* (die Haus-Automatisierung) oder allgemeiner für das *Internet-of-Things* (*IoT*) sehr gut geeignet. MQTT kann als inoffizieller Standard für das IoT angesehen werden.

Wie die Signale übermittelt werden, ist für das MQTT-Protokoll unwichtig. Nachrichten werden mittels TCP/IP - das Protokoll, über das auch unsere Internetanfragen laufen - übertragen. Dafür sind die Ports 1883 (unverschlüsselt) und 8883 (verschlüsselt) reserviert.

Auf der Webseite der MQTT-Organisation sind alle technischen Details dokumentiert. Von HiveMQ gibt es das kostenlose eBuch MQTT Essentials (englisch).

Was erwartet Sie in diesem Beitrag?

Sie werden...

- den Einsatz von MQTT kennenlernen
- Aufbau und Übertragung einer Nachricht kennenlernen
- den Ablauf einer MQTT-Übertragung verstehen
- eigene Versuche mit MQTT durchführen
- die Verwendung von MQTT in ZigBee und Node-RED kennenlernen
- die Unterschiede zwischen MQTT und HTML verstehen
- die wichtigsten Fachausdrücken von MQTT verstehen

Ein ☝ im Text weist darauf hin, dass in der Langversion an der Stelle mehr Text, ein Programmcode oder eine detaillierte Anleitung zu finden ist. Die Langversion kann über den Link und den QR-Code am Ende des Beitrags abgerufen werden.

Die Grundlagen

Jedes MQTT-Netzwerk hat genau einen *Server*, auch *Broker* oder *central hub* genannt. Häufig wird im Smart Home dazu ein Raspberry oder ein kleiner Linux-Desktoprechner verwendet. Da dieser Rechner ständig eingeschaltet sein muss, sollte auf den Energiebedarf geachtet werden.

Alle anderen Mitglieder des Netzwerks sind *Clients*. Jeder *Client* muss vorerst beim Server registriert werden. Die Registrierung heißt *pairing,* oft übersetzt mit *anlernen*. Clients tauschen keine Nachrichten direkt untereinander aus, vielmehr geht alles immer über den Server (Broker):

- Clients senden Nachrichten an den Broker und geben an, von welchem *Thema* (Topic) die Nachricht handelt.
  Beispiel: *Lufttemperatur im Badezimmer*.
- Clients können Nachrichten zu Themen abonnieren, auf die sie reagieren sollen.
  Beispiel: ein Client fragt nach der *Lufttemperatur im Badezimmer* und entscheidet, ob er zum Thema *Badezimmerheizung* den *Nachrichteninhalt* "Heizung ein" oder "Heizung aus" senden soll --- abhängig von der Temperatur und bei Bedarf auch von anderen Faktoren, wie etwa der Tageszeit. Ein weiterer Client hat das Thema *Badezimmerheizung* abonniert und schaltet über ein Relais die Heizung ein oder aus.

Aufbau einer Nachricht

Nachrichten bestehen aus einem *Thema* (*Topic*) und einem *Nachrichteninhalt* (*Payload*).

- Ein Topic ist eine Zeichenkette, die (im *Smart Home*) ein konkretes Gerät oder eine Funktion benennt.

  Beispiel: der Temperatursensor wurde als "Temp" benannt. Durch Schrägstriche können Hierarchieebenen eingerichtet werden, Beispiele: "Bad/Temp" oder "Wohnzimmer/Temp" oder "Bad/Heizung".
- Der Nachrichteninhalt wird oft in JSON-Form angegeben. JSON steht für "JavaScript Object Notation". Wie der Name schon sagt, ist dieses Format zur Speicherung und Übertragung von strukturierten Daten (das heißt von Objekten) für die Programmiersprache JavaScript entwickelt worden. JSON ist programmiersprachenunabhängig und wird auch gerne auf Webseiten zur Übertragung von Daten zwischen einem Client (zum Beispiel einem Webbrowser) und dem Server verwendet.
- Der Aufbau eines Objekts in JSON-Notation:
  - Jeder JSON-Eintrag ist ein Paar, bestehend aus einem *Schlüsselbegriff* (*Key*) und einem *Wert* (*Value*). Der Schlüsselbegriff ist in gerade Doppelapostrophe (" ") eingeschlossen. Zulässige Datentypen für Werte sind null, true, false (alle drei ohne Apostrophe!), eine Zeichenkette, eine Zahl, ein Array oder wieder ein Objekt, das selbst in JSON-Form angegeben wird.
  - Mehrere Key-Value-Paare werden durch Beistriche getrennt.
  - Das Ganze wird von geschwungenen Klammern umgeben.
  - Zeichen werden in der Regel mit UTF-8 kodiert.

  Beispiele:

  - Befehl zum Einschalten eines Verbrauchers:

    ```
    { "state":"on"
    }
    ```

    Meldung eines Temperatursensors:

    ```
    {  "temperature": 24.00
    }
    ```

Meldung eines Zwischensteckers über den aktuellen Zustanf (state) und Strom-, Energie-, Wirkleistungs- und Spannungsmesswerte (vereinfacht):

```
{ "current":0.27,
  "energy":3.97,
  "power":60,
  "state":"ON",
  "voltage":229
}
```

Bei der Ausgabe werden die Einträge nach dem Schlüsselbegriff oft alphabetisch sortiert.

Übertragen einer Nachricht

Ein Client kann Nachrichten an den Server senden, etwa wenn sich eine Messgröße (wie etwa die Raumtemperatur) ändert oder wenn der Client aufgefordert wird, etwas zu melden. In den Tasmotaeinstellungen kann eingestellt werden, in welchen Abständen der Client seinen Status meldet.

Dieses Senden von Nachrichten heißt *publish* (*veröffentlichen*).

Beispiel einer Nachricht (vereinfacht):

```
`{
  `"topic":"Bad/Temp",
    "payload": {
        "humidity":42.63,
        "temperature":24.00
    }
}`
```

Das bedeutet: in einem Smart Home ist im Bad der Temperatursensor *Temp* installiert und der meldet eine relative Feuchtigkeit von 42.63% und eine Temperatur von 24.00°C

Beispiel einer echten Nachricht :

Die tatsächliche Meldung wurde mit dem Programm JSON Beautifier "verschönert" und sieht so aus

```
{
  "topic": "zigbee2mqtt/EG/SR/TempS02",
  "changed": {
    "old": {
      "battery": 100,
      "humidity": 43.05,
      "linkquality": 32,
      "temperature": 20.99,
      "voltage": 3000
    },
    "new": {
      "battery": 100,
      "humidity": 42.63,
      "linkquality": 36,
      "temperature": 20.99,
      "voltage": 3000
```

```
    }
  },
  "payload": {
    "name": "SR",
    "time": "2024-05-29T17:12:35.366Z",
    "timestamp": 1717002755366,
    "temp": true,
    "humidity": 42.63
  },
  "payload_raw": {
    "battery": 100,
    "humidity": 42.63,
    "linkquality": 36,
    "temperature": 20.99,
    "voltage": 3000
  },
  "homekit": {
    "TemperatureSensor": {
      "CurrentTemperature": 20.99
    },
    "HumiditySensor": {
      "CurrentRelativeHumidity": 42.63
    },
    "Battery": {
      "BatteryLevel": 100,
      "StatusLowBattery": 0
    }
  },
  "format": {},
  "selector": "zigbee2mqtt_EG_SR_TempSO2",
  "item": {
    "date_code": "",
    "definition": {
      "description": "Temperature & humidity sensor",
      "exposes": [
        {
          "access": 1,
          "description": "Remaining battery in %",
          "name": "battery",
          "property": "battery",
          "type": "numeric",
          "unit": "%",
          "value_max": 100,
          "value_min": 0
        },
        {
          "access": 1,
          "description": "Measured temperature value",
          "name": "temperature",
          "property": "temperature",
          "type": "numeric",
          "unit": "°C"
        },
        {
          "access": 1,
          "description": "Measured relative humidity",
```

```
          "name": "humidity",
          "property": "humidity",
          "type": "numeric",
          "unit": "%"
        },
        {
          "access": 1,
          "description": "Voltage of the battery in millivolts",
          "name": "voltage",
          "property": "voltage",
          "type": "numeric",
          "unit": "mV"
        },
        {
          "access": 1,
          "description": "Link quality (signal strength)",
          "name": "linkquality",
          "property": "linkquality",
          "type": "numeric",
          "unit": "lqi",
          "value_max": 255,
          "value_min": 0
        }
      ],
      "model": "WSD500A",
      "options": [
        {
          "access": 2,
          "description": "Number of digits after decimal point for
temperature, takes into effect on next report of device.",
          "name": "temperature_precision",
          "property": "temperature_precision",
          "type": "numeric",
          "value_max": 3,
          "value_min": 0
        },
        {
          "access": 2,
          "description": "Calibrates the temperature value (absolute
offset), takes into effect on next report of device.",
          "name": "temperature_calibration",
          "property": "temperature_calibration",
          "type": "numeric"
        },
        {
          "access": 2,
          "description": "Number of digits after decimal point for
humidity, takes into effect on next report of device.",
          "name": "humidity_precision",
          "property": "humidity_precision",
          "type": "numeric",
          "value_max": 3,
          "value_min": 0
        },
        {
          "access": 2,
```

```
          "description": "Calibrates the humidity value (absolute offset), takes into effect on next report of device.",
          "name": "humidity_calibration",
          "property": "humidity_calibration",
          "type": "numeric"
        }
      ],
      "supports_ota": false,
      "vendor": "TuYa"
    },
    "disabled": false,
    "endpoints": {
      "1": {
        "bindings": [],
        "clusters": {
          "input": [
            "genPowerCfg",
            "genIdentify",
            "msTemperatureMeasurement",
            "msRelativeHumidity",
            "genBasic"
          ],
          "output": [
            "genIdentify",
            "genOta",
            "genTime"
          ]
        },
        "configured_reportings": [],
        "scenes": []
      }
    },
    "friendly_name": "EG/SR/TempSO2",
    "ieee_address": "0xa4c13890d5649310",
    "interview_completed": true,
    "interviewing": false,
    "manufacturer": "_TZ3000_xr3htd96",
    "model_id": "TS0201",
    "network_address": 61298,
    "power_source": "Battery",
    "supported": true,
    "type": "EndDevice",
    "current_values": {
      "battery": 100,
      "humidity": 42.63,
      "linkquality": 36,
      "temperature": 20.99,
      "voltage": 3000
    },
    "homekit": {
      "TemperatureSensor": {
        "CurrentTemperature": 20.99
      },
      "HumiditySensor": {
        "CurrentRelativeHumidity": 42.63
      },
```

```
      "Battery": {
        "BatteryLevel": 100,
        "StatusLowBattery": 0
      }
    },
    "format": {}
  },
  "_msgid": "0d5f3b18305b3dc5",
  "_event": "node:c7742d2455f00b41"
}
```

Das bedeutet: in dieser Smart Home Steuerung wird zigbee2mqtt verwendet und deshalb beginnen alle Topics mit zigbee2mqtt Im Erdgeschoß (EG) im Raum SR ist der Temperatursensor TempS02 und der meldet eine relative Feuchtigkeit von 42.63% und eine Temperatur von 20,99°C

Da war doch die Rede von kurzen Nachrichten. Ganz schön gesprächig, so ein einfacher Temperatursensor!

Erklärungen zu einigen ausgewählten Zeilen:

Zeile	Erklärung
2	Der Temperatursensor TempS02 im Raum SR im Erdgeschoß (EG) wurde unter zigbee2mqtt angemeldet. Jedes Topic beginnt daher mit zigbee2mqtt
3	Bei jeder neuen Übertragung werden bei diesem Sensor auch die alten Werte übermittelt.
5	Der Ladezustand der Batterie in Prozent, siehe Zeilen 54 bis 60
6	Die relative Luftfeuchtigkeit in Prozent, siehe Zeilen 72 bis 76
7	Die Qualität der Funksignale: ein Wert zwischen 0 und 255 , siehe Zeilen 88 bis 94
8	Die Temperatur in Grad Celsius. siehe Zeilen 64 bis 68
9	Der Ladezustand der Batterie in Millivolt, siehe Zeilen 80 bis 84
21	Jahr, Monat, Tag, Stunde, Minute, Sekunde, Millisekunde, Zeitzone
51	Exposes: welche Daten stellt das Gerät zur Verfügung? Zeilen 53 bis 94
97	Modellbezeichnung
98	Beschreibung des Geräts, bis Zeile 130
132	supports ota = Kann über das WLAN eine neue Firmware eingespielt werden? OTA = Over The Air
133	Vendor = Verkäufer
135	Wenn disabled auf true gestellt wird, nimmt das Gerät nicht an Datenaustausch teil
136	Endpoints = Liste aller Parameter, die für die Ein- und Ausgabe verwendet werden können.
157	Friendly name: ein Gerät kann über die IEEE-Adresse oder über die frei wählbare "freundliche" Bezeichnung angesprochen werden.

Zeile	Erklärung
158	Die IEEE-Adresse: eindeutig und unveränderlich
161	Bezeichnung, die der Hersteller vergibt
162	Modell-Bezeichnung
163	Kurzadresse, Verwendung statt der IEEE-Adresse
166	Gerätetyp
188	_msgid = Jede Mitteilung hat eine eindeutige Nummer, die vom Broker vergeben wird

Abonnieren

Ein (anderer) Client kann Nachrichten abonnieren. Das heißt, der Client teilt dem Server über die Mitteilung *subscribe* (*anmelden*) mit, dass er Nachrichten zu *einem* Topic (im Beispiel oben zum Topic "Bad/Temp") bekommen möchte, etwa um den Temperaturwert für die Steuerung der Heizung zu verwenden.

Auch mehrere Topic können auf einmal abonniert werden:

Mit "#" werden alle Topics einer Hierarchieebene und aller folgenden abonniert. mit "+" nur die Topics der jeweiligen Ebene.

Beispiel: "Erdgeschoß/Wohnzimmer/#" abonniert alle Geräte im Wohnzimmer, "Erdgeschoß/#" alle Geräte im Erdgeschoß.. Mit "Erdgeschoß/+/Temp" können alle Temperatursensoren im Erdgeschoß ausgelesen werden.

> **Note**
>
> Über "Erdgeschoß/Deckenlampen/#" können die Deckenlampen *nicht* ein- oder ausgeschaltet werden: dazu müsste eine *Gruppe* eingerichtet werden.

Eine Nachrichtenübertragung im Detail

Erklärung	Client 1	Richtung	Server = Broker	Richtung	Client 2
Client 1 meldet sich beim Server an	CONNECT	→			
Server bestätigt die Anmeldung (Acknowledge)	CONNACK	←			
Client 2 hat sich schon früher angemeldet und schickt über das Topic "Bad/Temp" den Wert 24°C an den Server				←	PUBLISH Bad/Temp 24°C

Erklärung	Client 1	Richtung	Server = Broker	Richtung	Client 2
Dabei wird auch *retain* gesetzt. Der Server merkt sich die Temperatur			24°C		retain
Client 1 abonniert die Temperatur im Bad	SUBSCRIBE Bad/Temp	→			
Server schickt den letzten (über retain) bekannten Temperaturwert	PUBLISH Bad/Temp 24°C	←	24°C		
Client 2 veröffentlicht einen neuen Temperaturwert: 27°C			27°C	←	PUBLISH Bad/Temp 27°C
Der Server merkts sich den aktuellen Wert und schickt ihn an alle Abonnenten weiter.	PUBLISH Bad/Temp 27°C	←	27°C		
Client 1 möchte nicht mehr an der Datenübertragung teilnehmen und meldet sich ab	DISCONNECT	→			

Experimente mit MQTT

Dazu benötigen wir einen MQTT-Broker, mindestens einen Client und ein Programm, mit dem wir die PUBLISH- und SUBSCRIBE-Befehle an den Broker senden können

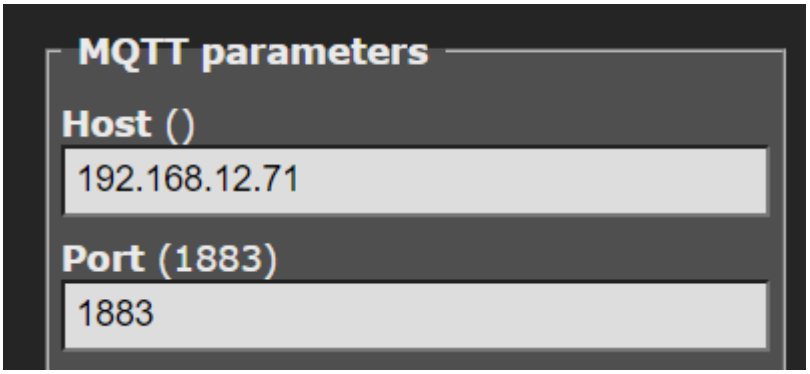
Der Broker: Mosquitto

Häufig wird Mosquitto eingesetzt: es gibt Versionen für Linux, Mac, Windows und andere. Die Installation wird auf der Mosquitto-Webseite sehr detailliert beschrieben.

Bei mir ist der Mosquitto-Broker unter der Adresse 192.168.12.71 zu finden, die Beispiele sind darauf ausgerichtet. Diese IP-Adresse bitte entsprechend dem eigenen Netz ändern.

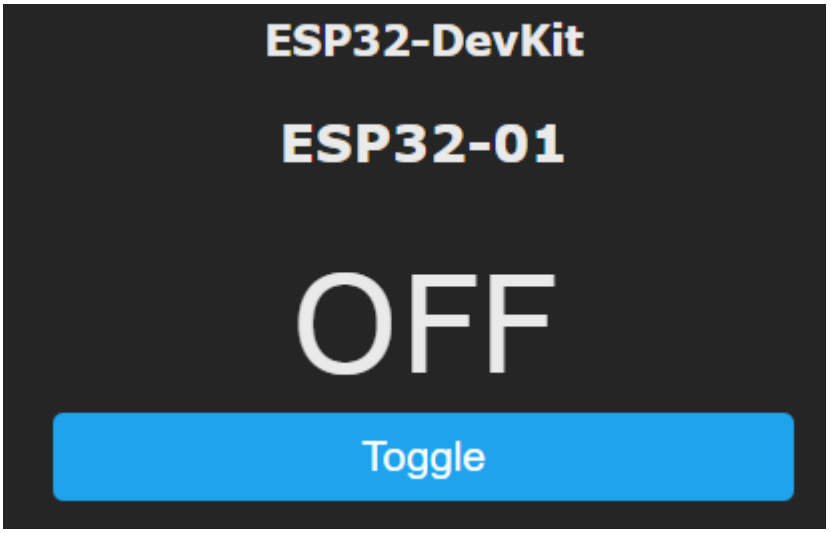
Der Client: ESP32

Dann brauchen wir wenigstens ein Gerät, das auf MQTT-Befehle reagiert. Wir können dazu den ESP32 verwenden, der in den PCNEWS 180 beschrieben worden ist. Die Adresse des Brokers muss in `Configuration` → `Configure MQTT` eingetragen sein:



> **Note**
>
> Wer ein Smart Home mit ZigBee und/oder Node-RED aufgebaut hat, findet in der Langversion dieses Beitrags Hinweise, wie damit MQTT-Experimente möglich sind.

Der ESP32 hat (bei mir) die Adresse 192.168.50.1, GPIO2 ist (wie in den PCNEWS 180 beschrieben) als Relais konfiguriert. Um den Zustand des Ausgangs 2 anzuzeigen, wird entweder eine LED angeschlossen oder der Ausgang in der Benutzeroberfläche kontrolliert:



Das Programm: MQTTX CLI

MQTTX ist nützliches Programm, mit dem alle MQTT-Befehle erprobt werden können und das ohne zusätzlich installierte Programme auskommt. Es wird für verschiedene Betriebssysteme angeboten.

Hier zeige ich den Aufruf über die Windows-Kommandozeile (CLI). Warum über die CLI? Weil damit am besten zu sehen ist, worauf es ankommt!

- Mit einem Klick in der Download-Seite auf v1.9.10.cli.win64.exe wird die Datei `mqttx-cli-win-x64.exe` in den Download-Ordner gestellt.
- Im Startmenü im Suchfenster `PowerShell` eingeben und damit die Windows PowerShell öffnen.
- Ins Download-Verzeichnis navigieren, üblicherweise mit
  `cd \users\EigenerBenutzername\Downloads`

- Dort sollte `mqttx-cli-win-x64.exe` zu finden sein — mit `dir` überprüfen.

Jetzt kann es losgehen!

> **Tip**
>
> Die PowerShell verlangt für Zeichenketten in der Kommandozeile einfache Apostrophe - sonst sind in Windows doppelte Apostrophe üblich.

Ein erster Test

Mosquitto bietet einen Testserver an. Können wir den erreichen? Wir geben ein:

```
mqttx-cli-win-x64.exe pub -h test.mosquitto.org  -t 'test/topic' -m 'hallo martin'
```

und erhalten:

```
PS C:\users\martin\Downloads> .\mqttx-cli-win-x64.exe pub -h
test.mosquitto.org  -t 'test/topic' -m 'hello martin'``
[2024-6-2] [14:26:11] » ...  Connecting...
[2024-6-2] [14:26:11] » √  Connected
[2024-6-2] [14:26:11] » ...  Subscribing to test/topic...
[2024-6-2] [14:26:11] » √  Subscribed to test/topic
[2024-6-2] [14:26:11] » mqtt-packet: Packet {
  cmd: 'publish',
  retain: true,
  qos: 0,
  dup: false,
  length: 26,
  topic: 'test/topic',
  payload: <Buffer 48 49 7e 0a 6d 71 74 74 20 74 65 73 74>
}
topic: test/topic
qos: 0
retain: true
payload: HI~
mqtt test
```

Die Verbindung klappt!

Publish

Beim ESP32 mit der Adresse http://192.168.50.1 ist der Ausgang GPIO2 im Zustand `OFF` .

Ein Klick auf `Information` zeigt das Topic dieses ESP32:

Das Topic zur Steuerung von *Tasmota*-Komponenten beginnt immer mit `cmnd/tasmota_`, gefolgt von den letzten 6 Stellen der MAC-Adresse, hier F8:29:14.

Um den Power-(Relay-)Ausgang umzuschalten, ist ein MQTT-Befehl mit dem Topic `cmnd/tasmota_F82914/power` und dem Payload `toggle` zu senden:

```
PS C:\users\martin\Downloads> .\mqttx-cli-win-x64.exe pub -h 192.168.12.71  -p 1883
-t 'cmnd/tasmota_F82914/power' -m 'toggle'
```

Der Reihe nach:

- `mqttx-cli-win-x64.exe` ist der Name des Programms. Wer will, kann die Datei auf `m.exe` umgenennen -- spart Tipparbeit!
- `pub` steht für `publish` - wir wollen etwas *senden*.
- `-h 192.168.12.71` Der Befehl geht an den Host 192,.168.12.71 - da ist der Mosquitto-Server installiert
- `-p 1883` Die Postnummer ist 1883. Da das der Standard ist, kann sie auch weggelassen werden
- `-t cmnd/tasmota_F82914/power` Das Topic
- `-m toggle` Die Mitteilung (Message = Payload) besteht aus einem einzigen Wort: toggle (umschalten)

Absenden! Die Antwort von Mosquitto in der PowerShell

```
[2024-6-2] [17:07:39] » ...  Connecting...
[2024-6-2] [17:07:39] » √  Connected
[2024-6-2] [17:07:39] » ...  Message publishing...
[2024-6-2] [17:07:39] » √  Message published
```

Im Browser wird auf der ESP32-Benutzeroberfläche die Änderung angezeigt. Wer eine LED angeschlossen hat, sieht die Zustandsänderung auch dort.

Details zu den möglichen Formaten des Message (Payload):

https://www.emqx.com/en/blog/how-to-process-json-hex-and-binary-data-in-mqtt

Subscribe

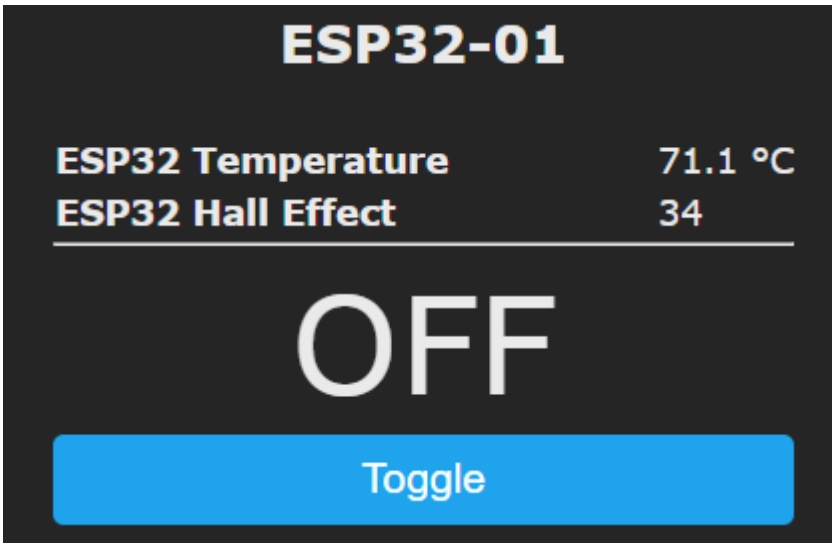
Nun wollen wir etwas von unserem ESP32 empfangen. Der Prozessor bietet viele Anschlussmöglichkeiten für Peripheriekomponenten. Programme müssen nicht geschrieben werden, alles kann über die `Configuration` eingestellt werden.

Der ESP32 hat aber auch eine Reihe von Sensoren eingebaut: so kann das Magnetfeld über eine Hallsonde oder die Prozessortemperatur direkt ausgelesen werden. (Das Handbuch weist darauf hin, dass diese Messwerte nicht sehr genau sind.)

Um das Magnetfeld anzeigen zu können, ist über `Configuration` → `Configute Module` GPIO36 auf `HallEffect` `1` zu setzen. Nicht vergessen: danach `Save` anklicken!

Der interne Temperatursensor wird über `Tools` → `Console` und den Befehl `SetOption146 1` eingeschaltet. Es folgt wieder `Tools` und `Main Menu`.

In der Benutzeroberfläche erscheint:



Mit subscribe teilen wir dem Broker mit, dass wir diese Daten gerne (regelmäßig, bei jeder Änderung) hätten:

`PS C:\users\martin\Downloads\> .\mqttx-cli-win-x64.exe sub -h 192.168.12.71 -t 'tele/tasmota_F82914/SENSOR' -p 1883`

Zu den Teilen des Befehls:

- `sub` steht für subscribe
- `-t 'tele/tasmota_F82914/SENSOR'` ist das Topic zum Anfordern der Messwerte
- `-h` und `-p` sind schon bekannt.

Und schon treffen die Antworten ein:

```
[2024-6-2] [18:42:39] » ...  Connecting...
[2024-6-2] [18:42:39] » √  Connected
[2024-6-2] [18:42:39] » ...  Subscribing to tele/tasmota_F82914/SENSOR...
[2024-6-2] [18:42:39] » √  Subscribed to tele/tasmota_F82914/SENSOR
[2024-6-2] [18:42:40] » topic: tele/tasmota_F82914/SENSOR
qos: 0
payload: {"Time":"2024-06-02T17:42:39","ESP32":
{"Temperature":72.2,"HallEffect":36},"TempUnit":"C"}

[2024-6-2] [18:42:50] » topic: tele/tasmota_F82914/SENSOR
qos: 0
payload: {"Time":"2024-06-02T17:42:49","ESP32":
{"Temperature":72.8,"HallEffect":31},"TempUnit":"C"}

[2024-6-2] [18:43:00] » topic: tele/tasmota_F82914/SENSOR
qos: 0
payload: {"Time":"2024-06-02T17:42:59","ESP32":
{"Temperature":72.2,"HallEffect":22},"TempUnit":"C"}
```

Hier noch einmal die erste Payload etwas übersichtlicher:

```
{
  "Time": "2024-06-02T17:42:39",
  "ESP32": {
    "Temperature": 72.2,
    "HallEffect": 36
  },
  "TempUnit": "C"
}
```

Zusammenfassung

MQTT kommuniziert vor allem mit den beiden Befehlen PUBLISH und SUBSCRIBE. Mit MQTTX CLI ist deren Funktion einfach zu erproben.

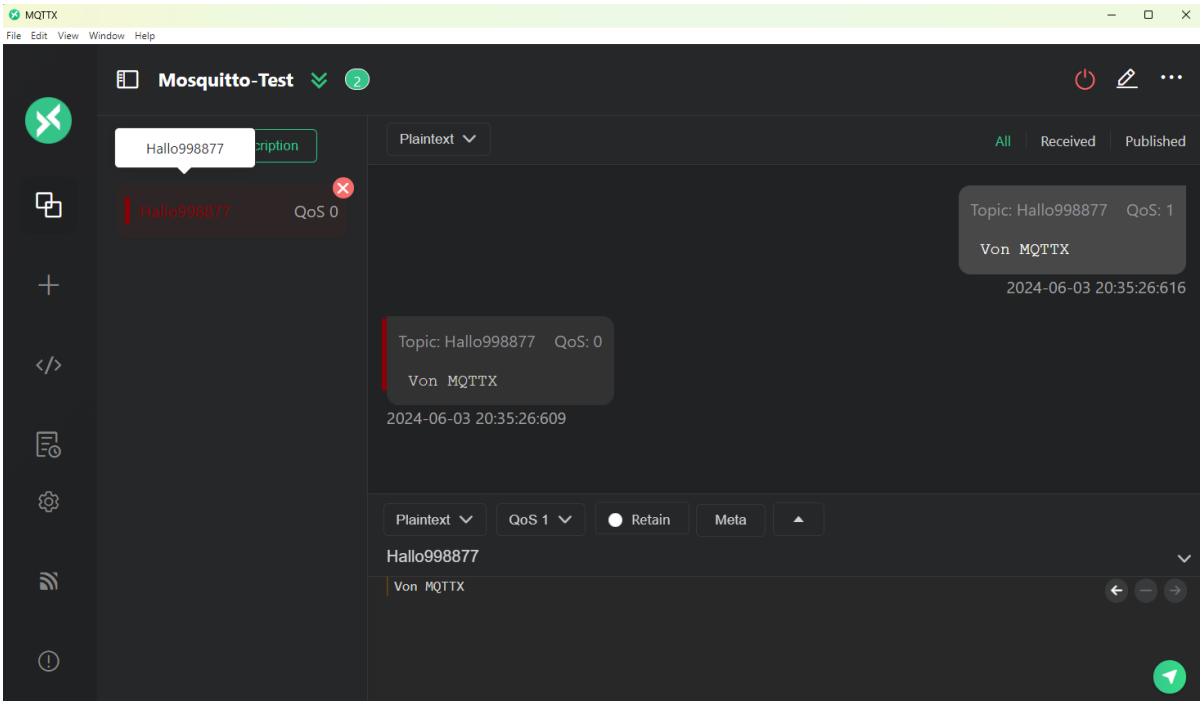
Für alle wichtigen Programmiersprachen gibt es Programmbibliotheken, um MQTT-Befehle direkt aus Programmen absetzen zu können.

Darüber hinaus gibt es eine große Zahl anderer Programme zur Kommunikation. Hier ein paar Beispiele.

Weitere Programme...

MQTTX

MQTTX (ohne CLI) liefert dieselben Informationen, allerdings mit einer grafischen Benutzeroberfläche. ~~



MQTT.js

MQTT.js ist eine Programmbibliothek für JavaScript bzw. ab Version 5.0 in TypeScript. Die Bibliothek kann im Browser und in Node.js-Projekten verwendet werden. Dadurch werden auch Anwendungen in JavaScript-basierenden Apps, zum Beispiel Svelte, möglich.

Ferner kann MQTT.js auch über die Kommandozeile verwendet werden. Dazu reicht die Installation mit

```
1 npm install mqtt -g
```

Zur Demonstration wird in einem Terminalfenster in dem öffentlich zugänglichen Testserver `test.mosquitto.org` ein `subscript` gestartet:

```
1 mqtt sub -t 'hallo998877' -h 'test.mosquitto.org' -v
```

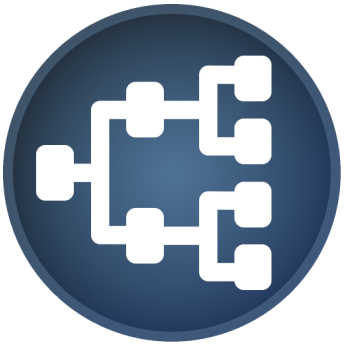
Von einem anderen Fenster wird der Text "`Von MQTT.js`" mit `publish` gesendet.

```
1 mqtt pub -t 'hallo998877' -h 'test.mosquitto.org' -m 'Von MQTT.js'
```

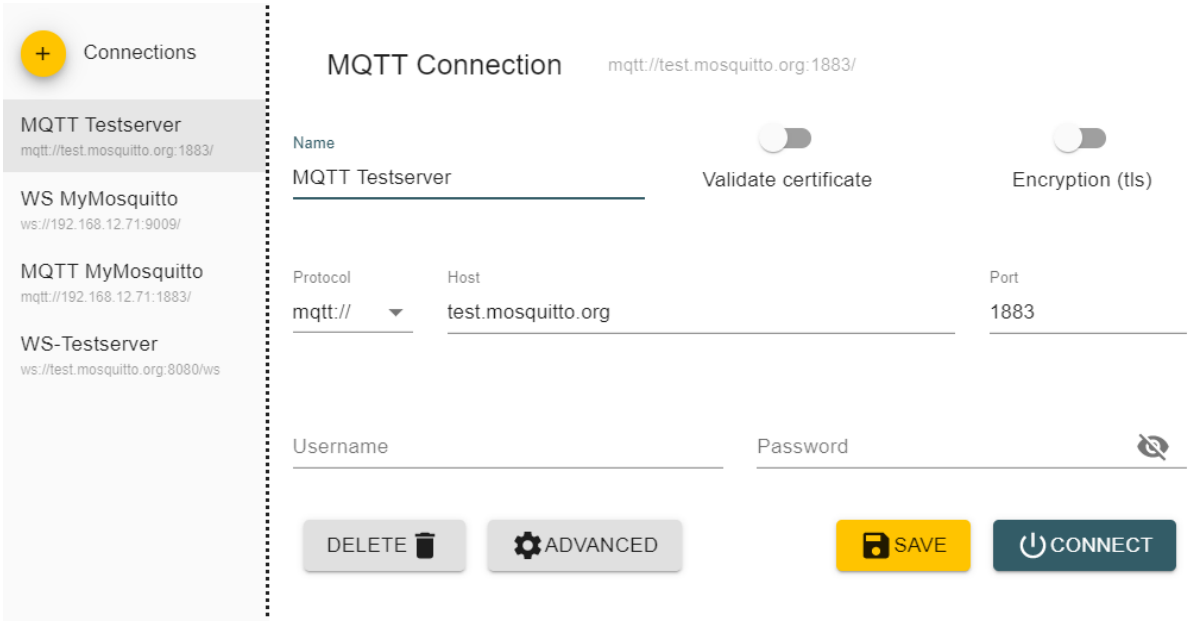
Erfolg — der Text kommt im ersten Fenster an!

Zum Topic `hallo998877`: Der Testserver kann aus dem gesamten Internet verwendet werden. Um nicht von anderen Nutzern gestört zu werden, sollte ein Topic verwenden, das höchstwahrscheinlich sonst niemand verwendet.

MQTT Explorer



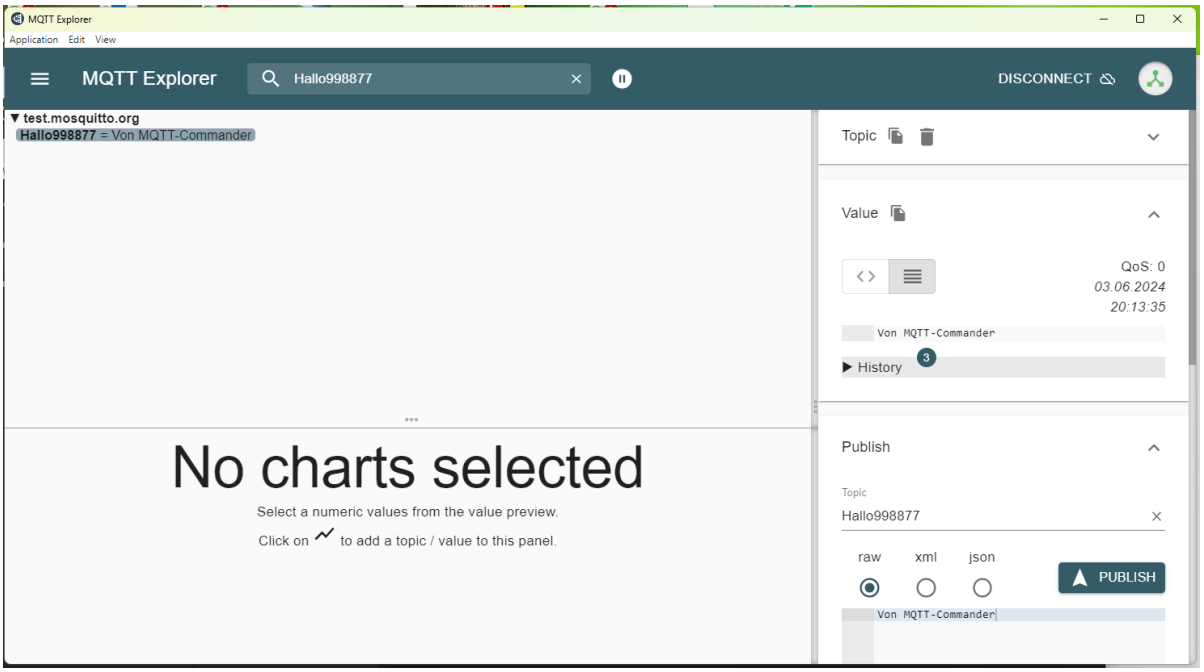
Der MQTT Explorer ist ein weiteres Programm, um Mitteilungen an einen Server zu senden (über publish) oder von dem Server zu empfangen (über subscribe), Die Benutzeroberfläche ist etwas gewöhnungsbedürftig, aber nach kurzer Einarbeitung leicht zu bedienen.



Die Anmeldeseite zeigt hier den Testserver `test.mosquitto.org` und meine eigenen Testserver.

Bei den bisher besprochenen Programmen ist es notwendig, schon beim Schreiben eines MQTT-Befehls zu wissen, wie die zu verwendenden Topic heißen. Der MQTT Explorer zeigt dagegen *alle* Topics, die im Netzwerk erreichbar sind.

In dem Beispiel wird an das Topic `Hallo998877` der Text "Von MQTT-Commander" gesendet. Der gesendete Text ist rechts unten zu senden, der empfangene links oben und rechts in der Mitte. Einfach einmal ausprobieren!



MQTT und Node-RED

Node-RED ist ein grafischer Werkzeugkasten für die Programmierung von ereignisgesteuerten Abläufen. In einer Smart Home Steuerung treten Signale (Ereignisse) ohne vorhersehbaren Ablauf auf und erfordern daher spezielle Programmiertechniken.

Node-RED kann durch Paletten (Programmpakete) erweitert werden, daher kann die MQTT-Kommunikation zwischen Broker (Server) und Clients in Node-RED implementiert werden. Die einzelnen Funktionen werden in Nodes implementiert und mit einander verknüpft. Das Node-RED-Programm kann gemeinsam mit einem MQTT-Broker (wie etwa Mosquitto) und Zigbee2MQTT auf einem Rechner installiert werden. Noide-RED wird als Webseite über einen beliebigen Browser bedient. Übliche Adresse: http://<Rechner-Adresse>:1880. Bei mir also http://192.168.12.71:1880.

MQTT und Zigbee2MQTT

ZigBee ist ein eigener Funkstandard, der wegen der geringeren Datenmenge weniger störungsanfällig ist. ZigBee-Komponenten bauen darüber hinaus dazu über ZigBee-Router ihr eigenes Mesh-Netzwerk auf und können damit auch große Flächen abdecken. ZigBee-Router sind, vereinfacht gesprochen, jene ZigBee-Komponenten, die ständig (über das Stromnetz) mit Energie versorgt werden und deshalb Signale an andere Komponenten weitergeben können. Das Mesh-Netzwerk wird von ZigBee selbstständig aufgebaut und bei Bedarf, zum Beispiel beim Ausfall einer Komponenten, selbstständig reorganisiert. Viele Komponenten für Smart Homes werden in zwei Varianten, nämlich für ZigBee und WLAN, angeboten. Zigbee2MQTT erlaubt die Verbindung von ZigBee-Geräten über MQTT.

> **Tip**
>
> Ein Topic für ein Gerät, das unter ZigBee2MQTT angemeldet ist, beginnt mit `zigbee2mqtt`, gefolgt vom *friedly name*, gefolgt von `set` oder `get` -- je nachdem, ob ein Befehl an das Gerät abgesetzt wird oder Daten von dem Gerät angefordert werden.
>
> Beispiel: `zigbee2mqtt/Erdgeschoß/Deckenlampe/set`

MQTT und HTTP

Eine Gegenüberstellung zeigt die wichtigsten Unterschiede.

Erklärung	**MQTT**	**HTTP**
Bezeichnung	MQTT - inzwischen ein "Eigenname"	Hyper Text Transfer Protocol
Datenübertragung mit ...	publish, subscribe	request, response
Die Objekte haben...	ein Topic	einen URI
Beide übertragen über ...	TCP	TCP
Beide erlauben eine gesicherte Übertragung über ...	TLS mit Benutzernamen und Passwort	TLS mit Benutzernamen und Passwort
Der Status der Verbindung ist ...	bekannt	nicht bekannt
Die Übertragung erfolgt...	asynchron, ausgelöst von Ereignissen	synchron

Werden Daten in einer Warteschlange zwischengespeichert?	Ja, der Broker kann Daten zwischenspeichern, wenn der Client nicht erreichbar ist	Nur über Zusatzprogramme
Länge einer Mitteilung	maximal 256 MByte	kein Limit
Verteilung der Mitteilungen	vom Broker an viele	eins zu eins
Sicherheit	Drei Qualitätsstufen	muss bei Bedarf zusätzlich implementiert werden

Quelle: basierend auf Unterlagen von HIVEMQ

Wichtige Begriffe

Connection:

Eine Verbindung zwischen Broker und Client (nach dem Pairing) zum Zweck des Datenaustauschs. Zwei Clients tauschen Daten nie direkt aus, immer nur über den Broker. Da der Broker ja den Überblick hat, entscheidet er selbst, wann Daten gesendet werden. Wenn ein Client etwas senden will, sendet er ein `CONNECT`-Ansuchen an der Broker. Dieser antwortet mit CONNACK (*Connection acknowledge*, Verbindung bewilligt).

Friendly name:

Jedes Gerät hat eine eindeutige MAC-Adresse, bestehend aus 12 Hexadezimalziffern und geschrieben beispielsweise als 94:B5:55:F8:29:14 oder 0x94b555f82914. Die Geräte können immer darüber angesprochen werden. Da das aber wenig benutzerfreundlich ist, kann zusätzlich ein friendly name, ein freundlicher Name, angegeben und in der Konfigurationsdatei des Brokers eingetragen werden. Beispiel: "Erdgeschoß/Küche/Deckenlampe".

ISO-Schichtenmodell:

MQTT ist in den Schichten 5-7 angelegt. Darunter (in der Schicht 4) ist TCP, das *transmission control protocol* und (in der Schicht 3) IP (das *internet protocol*).

Last will and testament (LWT):

Wenn ein Client eine Verbindung aufbaut, kann er zusätzlich eine Nachricht im Broker hinterlegen, die gesendet wird, wenn die Verbindung unterbrochen wird. Genannt *last will and testament*, (Letzter Wille und Testament) oder kurz nur *will*

Pairing:

Bevor eine Verbindung mit CONNECT angefordert werden kann, muss ein neues Gerät angemeldet werden. Der Vorgang heißt *pairing*, übersetzt oft mit *anlernen*. Dazu muss einerseits am MQTT Broker das Anlernen aktiviert werden und andererseits das neue Gerät selbst in den pairing-Modus versetzt werden.

- Beim Broker geschieht dies mit einem Software-Befehl. zigbee2mqtt verwendet die Konfigurationsdatei *configuration.yaml*. Dort kann das *pairing* mit *permit_join: true* aktiviert werden. Nicht vergessen: nach dem *paiting* wieder auf *false* setzen.

- Ein neues Gerät wird meistens bereits mit dem ersten Anschließen automatisch in den Pairing-Modus versetzt. Aber natürlich geht das auch nachträglich:
  - Oft wird dazu eine Taste solange gedrückt, bis eine LED blinkt.
  - LED-Lampen haben keine Tasten, daher werden sie vier- oder fünfmal kurz hintereinander ein- und ausgeschaltet. (Nicht vergessen: die Lampe muss zum Pairing natürlich zuletzt eingeschaltet bleiben.) Die Lampe beginnt dann im Zwei-Sekunden-Rhythmus zu "atmen", wird also langsam heller und wieder dunkler.
  - Gute Anleitungen zum Pairen sehr vieler Geräte sind in den ZigBee2MQTT-Gerätebeschreibungen zu finden.
- Empfehlung: beim Pairen sollten Broker und neues Gerät möglich nahe beisammen sein. Das dürfte ein Sicherheitsmaßnahme sein, um unbeabsichtigtes Pairing zu verhindern.

Payload:

Der aktuelle Nachrichteninhalt; anders formuliert die Daten, die gesendet oder empfangen werden. Payloads werden häufig als JSON-Zeichenkette geschrieben, da diese Schreibweise sehr leicht (für Menschen) lesbar und (für Programme) analysierbar ist.

Publish:

Der Mechanismus zum Senden von Daten

Publisher, *Subscriber:*

Clients können als *Publisher* oder *Subscriber* verwendet werden. Ein *Publisher* sendet Daten aus (Beispiel: Temperaturwerte), ein *Subscriber* empfängt Daten und wertet sie aus (Beispiel: ein Microcontroller, der einen Verbraucher schaltet). Viele Clients sind Publisher und Subscriber zugleich.

Quality of Service (QoS):

Welche Qualität der Datenübertragung ist in einem konkreter Anwendungsfall erforderlich, welche Maßnahmen zur Sicherung werden gefordert? Es gibt die Stufen 0, 1 oder 2.

- Stufe 0: Client und Server hängen über eine direkte Drahtverbindung zusammen, daher ist es unwahrscheinlich, dass Daten verloren gehen. Oder der Verlust einzelner Datenpakete ist zu verschmerzen
- Stufe 1: Jedes Datenpaket muss tatsächlich ankommen, doppelte Pakete werden erkannt und aussortiert. Oder Stufe 2 ist zu aufwändig.
- Stufe 2: Jedes Datenpaket muss genau einmal verarbeitet werden.

Retain:

Wenn der aktuelle Zustand eines Clients auf Anforderung nicht übertragen werden kann, wird der letzte Zustand übertragen. Dazu ist natürlich eine Zwischenspeicherung im Broker notwendig. Beispiel: ein batteriebetriebener Temperatursensor schickt -- um Energie zu sparen -- nur dann Daten, wenn sich ein Wert ändert, .

Subscribe:

Der Mechanismus für das Anmelden zum Empfangen von Daten

Subscriber:

Siehe *Publisher*

Topic:

Übersetzt: Thema. Die Adresse eines Geräts oder einer Funktion eines Geräts als Zeichenkette

WebSocket:

WebSocket baut auf TCP auf und erlaubt eine bidirektionale Verbindung. MQTT-Befehle können in WebSockets "verpackt" werden und damit über HTTP/HTTPS übertragen werden. Das gestattet, MQTT-Befehle über Webbrowser zu senden und zu empfangen.

Zur Langversion

Link und QR-Code: https://pcnews.at/markdown/n181/index.html oder https://t.ly/lblE0

Copyright