

# MQTT - Eine Einführung

## Inhaltsverzeichnis

- [I - Einleitung](#)
- [II - MQTT bei der Arbeit zusehen](#)
- [III - Grundlagen der MQTT-Syntax \(am Beispiel Tasmota\)](#)
- [IV - Weitere Beispiele – Spielwiese](#)
- [V - Quellen, Weiterführende Links](#)

Dieser Artikel führt in die Benutzung des Kommunikationsprotokolls MQTT - insbesondere in Verbindung mit [Tasmota](#)-Sonoffs - ein. Anhand von direkter Kommunikation mit einem MQTTBroker mithilfe eines MQTT-Publishers (mosquitto\_pub) und der Beobachtung des Nachrichtenverkehrs durch einen MQTT-Suscribers (mosquitto\_sub) soll Struktur und Arbeitsweise des Protokolls verständlich werden und zur weiteren Benutzung anregen.

## MQTT

### I - Einleitung

Wer mit der [Tasmota](#)-Firmware arbeitet, kommt kaum an dem Thema MQTT vorbei, außer wenn jemand ausschließlich über Alexa mit den [Tasmota](#) geflashten Geräten kommunizieren will. Da stört MQTT eher und sollte mit dem Konsolen-Befehl `setoption3 0` abgeschaltet werden.

Auf der Original-Commands-Webseite von [Tasmota](#) heißt es: "MQTT ist das bevorzugte Interaktions-Interfaces". Grund genug sich mit diesem Protokoll etwas ausführlicher zu beschäftigen. Bevor es in die direkte und konkrete Kommunikation/Interaktion mit den [Tasmota](#)-Geräten über MQTT geht, möchte ich versuchen, die Arbeitsweise von MQTT "sichtbar" und vielleicht verständlich zu machen.

Was sind die Voraussetzungen für MQTT? Es muss eine Software laufen (am besten 7 Tage, 24 Stunden), die die Funktion eines sogenannten MQTT-Brokers wahrnimmt. Unter Linux ist das in der Regel das Paket mosquitto, das in allen gängigen Distributionen zur Verfügung steht. Es wird mit `sudo apt install mosquitto` installiert. Zusätzlich sollte das Paket mosquitto-clients installiert werden. In diesem Paket stecken die beiden Terminal-Programme mosquitto\_pub und mosquitto\_sub. Also sollte man alle drei gleich in einem Rutsch installieren

```
sudo apt install mosquitto mosquitto-clients.
```

Wer mit dem IOBroker arbeitet und dort den Sonoff-Adapter installiert hat aktiviert auf derselben Maschine einen MQTTBroker, der unter der IP des IOBrokers und dem Port 1883 (Standard) erreichbar ist.

Das Paket mosquitto-clients kann allerdings auch auf einem anderen Rechner installiert werden, denn wie der Name schon sagt, sind das Client-Programme, die mit dem Server-Program mosquitto kommunizieren. Es gibt Versionen für Linux und für Windows.

Was genau ist ein MQTTBroker, also das Programm mosquitto? Im Deutschen würden wir zu einem Broker eher Makler sagen oder Vermittler. Ein Makler bringt Leute zusammen: Hausbesitzer und Hauskauf-Interessenten, Aktiengesellschaften und Investoren. Anders als menschliche Makler hat der MQTTBroker keine eigenen (wirtschaftlichen) Interessen, sondern vermittelt nur Nachrichten von den Geräten.

Und nun kommen unsere (mosquitto-)MQTTClients wieder ins Spiel. mosquitto\_pub ist ein Anbieter, ein Publisher von Nachrichten, mosquitto\_sub ist ein Empfänger, ein Abonnent, ein Empfänger oder ein Subskriptient von Nachrichten. Durch mosquitto\_sub können wir MQTT-Botschaften sichtbar machen, wir können den MQTT-Broker "abhören".

Mit der [Tasmota](#)-Firmware geflashte Sonoffs sind in der Regel beides: Sie verschicken Nachrichten, z. B. über ihren Status (An/Aus, Sensordaten) und sie empfangen Nachrichten bzw. Befehle wie „Schalte An“, „Zeige Sensordaten“, „Starte neu“ usw.

## II - MQTT bei der Arbeit zusehen

### 1. Unter Linux

#### 1. mosquitto\_sub und mosquitto\_pub

Im folgenden gehe ich von folgender Konstellation aus:

**a)** Es gibt einen Raspberry Pi, auf dem die Pakete mosquitto und mosquitto-clients installiert sind. In meinen Beispielen wird hat Raspberry Pi mit dem MQTTBroker mosquitto die IP 192.168.178.20. Diese IP muss beim Nachvollziehen der Beispiele natürlich immer durch die korrekte IP des eigene MQTTBrokers ersetzt werden.

**b)** Von Vorteil wäre nun ein zweiter Rechner mit einem größeren Bildschirm, um zwei Terminalfenster oder zwei SSH-Verbindungsfenster nebeneinander stellen zu können. Ist der zweite Rechner (Client-Rechner) ebenfalls ein Linux-Gerät, dann sollten darauf das Paket mosquitto-clients (ebenfalls) installiert werden. Ist der zweite Rechner ein Windows-Rechner, muss darauf entweder das Programm Putty (oder ein anderer SSH-Client) installiert sein oder die Windows-Versionen von mosquitto\_pub und mosquitto\_sub. Wie das unter Windows geht – leider etwas komplizierter als unter Linux – ist hier ganz gut beschrieben:

<https://www.msxfaq.de/sonst/iot/mqtt.htm> besonders im Abschnitt "Mosquitto unter Windows".

**c)** Ist der Client-Rechner ein Linux-Gerät dann sollten zwei Terminal-Fenster geöffnet werden. In das eine Fenster wird folgendes eingegeben:

```
mosquitto_sub [-u Benutzername -P Passwort] -v -h 192.168.178.20 -t '#'
```

Womöglich erscheinen sogleich viele Nachrichten. Das passiert, wenn der Broker in einem Netz bereits aktiver IoT-Geräte läuft, in unserem Fall sind das die [Tasmota](#)-Geräte. Wenn noch kein Gerät installiert ist, ist auch nichts zu sehen, weil es keine Nachrichten gibt.

**d)** Nun wechseln wir in das andere Terminalfenster und geben folgendes ein:

```
mosquitto_pub [-u Benutzername -P Passwort] -h 192.168.178.20 -t stat/ -m 'so ein Quatsch'
```

Im Fenster mit mosquitto\_sub soll nun stat So ein Quatsch erschienen sein. Das zeigt schon einmal zweierlei: Erstens die Kommunikation mit dem MQTTBroker hat funktioniert und zweitens ist es dem Broker völlig egal, was er überträgt, Hauptsache die Syntax stimmt.

### 2. MQTTfx

Wird später ergänzt

### 3. Unter Windows

#### 1. Variante 1: Mit Putty

Zwei SSH-Verbindungen zum Raspberry Pi aufbauen. Diese beiden Verbindungen den oben beschriebenen zwei Linux-Terminals.

In einem der beiden Fenster mosquitto und mosquitto-clients installieren mit

```
sudo apt install mosquitto mosquitto-clients
```

Dann den Schritten c) und d) folgen wie oben beschrieben. Hinweis: Die Angabe von „-h 192.168.178.20“, also die genaue Bezeichnung des Rechners auf dem mosquitto läuft kann entfallen, wenn mosquitto\_sub oder mosquitto\_pub auf demselben Rechner ausgeführt werden, auf denne mosquitto läuft. Standardmäßig gehen die clients von localhost aus, wenn keine IP

angegeben wird.

## 2. Variante 2: Installation von mosquitto unter Windows

Die Installation unter Windows finde ich offen gestanden etwas sperrig und für einen normalen Windows-User kaum realisierbar. Ich werde auch nicht versuchen, dies hier anders und besser zu erklären, als es auf dieser Webseite gemacht wurde:

<https://www.msxfaq.de/sonst/iot/mqtt.htm>

Alles andere läuft dann im Wesentlichen wie unter Linux beschrieben:

Zwei Konsolenfenster öffnen indem man cmd in die Suchleiste von Windows eingibt

in beiden Konsolenfenstern in das mosquitto-Programmverzeichnis wechseln mit `cd c:\Program Files\mosquitto`

Schritte c) und d) wie unter Linux

## III - Grundlagen der MQTT-Syntax (am Beispiel Tasmota)

### 1. MQTT-Syntax allgemein

Wie sieht die Syntax im allgemeinen aus? Für `mosquitto_sub`, also dem Empfänger oder dem Abhörprogramm, bedeutet der oben aufgeführte Befehl:

`-v` möglichst viele Informationen ausgeben

`-h` IP des MQTTBrokers

`-t` Topic, das angezeigt werden soll

das '#' steht für "alle Informationen"

Falls nur das Topic # verwendet wird, also kein zusätzliches wie `stat/` oder `cmdn/` dann muss dieses # unter Linux in Hochkommata eingeschlossen werden: '#'. Wenn Topics benannt werden, dann bleiben die Hochkommata weg.

Auch der Schalter `-v` ist besonders sinnvoll, wenn wir wissen wollen, welches Gerät eine Nachricht abgeschickt hat. Bei Nutzung von `mosquitto_sub` in Skripten oder bei der Auswertung der Nachrichten können diese Zusatzinformationen allerdings stören. Doch dazu später bzw. in einem separaten Artikel.

Für `mosquitto_pub`, also dem Programm für die Erzeugung von Nachrichten, sieht es so aus:

`-h` IP des MQTTBrokers

`-t` Topic, das übermittelt werden soll

`-m` Botschaft (message, payload), die übermittelt werden soll

### 2. Tasmota-Syntax: Prefix, Topic, Payload

In der [Tasmota](#)-(MQTT-)Firmware wird unterschieden zwischen Prefix, Topic und der Payload.

Mit dem Prefix geben wir an, ob wir einen *Befehl* schicken wollen (*cmdn*), ob wir einen Zustand auslesen wollen (*stat*) oder ob wir Sensor-Daten (*tele*) auslesen wollen. Diese Präfixe sind im Prinzip Bestandteile eines jeden Topics:

```
mosquitto_pub ... -t cmdn/
```

An zweiter Stelle folgt das *Gerätetopic*, das ist der MQTT-Name des Gerätes. Deshalb sollte jedes MQTT-Gerät in den MQTT-Einstellungen im Feld Topic immer einen eindeutigen Namen bekommen. Alternative: ([Tasmota](#))-Konsolenbefehl topic NAME.

```
mosquitto_pub ... -t cmdn/Sonoff01
```

An dritter Stelle folgt die konkrete *Funktion*, die wir benutzen wollen, bzw. über deren Status wir etwas erfahren wollen, z. B. Power (Schaltzustand) oder SSId1 (Name des ersten Wifi-Netzes).

```
mosquitto_pub ... -t cmdn/Sonoff01/Power
```

Als vierte Information wird der *Inhalt*, die Aufgabe (*Payload*, Message) für die Funktion übertragen.

```
mosquitto_pub ... -t cmdn/Sonoff01/Power -m 'ON'
```

#### 1. Beispiele:

Zunächst einmal in dem Fenster, in dem `mosquitto_sub` läuft, das Programm mit Strg-C beenden. Dann folgendes eingeben:

```
mosquitto_sub [-u Benutzer -P Passwort] -h 192.168.178.20 -v -t tele/#
```

Falls mosquitto in einem bereits bestehenden Netz von [Tasmota](#)-Geräten arbeitet, erscheint jetzt nur noch ein Teil der Meldungen, nämlich alle, die mit tele im Topic beginnen. Wir haben mit dem neuen Befehl nur noch Nachrichten abonniert, die Telemetrie-Informationen enthalten (Filterung).

Wir machen die Gegenprobe. Im Fenster mit mosquitto-pub geben wir erneut den bereits gesendeten Befehl ein:

```
mosquitto_pub [-u Benutzername -P Passwort] -h 192.168.178.20 -t stat/  
-m 'so ein Quatsch'
```

In dem mosquitto\_sub-Fenster ist nichts passiert? Gut so! Wir geben folgendes ein:

```
mosquitto_pub [-u Benutzername -P Passwort] -h 192.168.178.20 -t tele  
-m 'so ein Quatsch'
```

Nun erscheint der Text im mosquitto\_sub Fenster. Man kann unter MQTT also sehr gezielt Informationen beziehen (\_sub) bzw. veröffentlichen (\_pub). Dieser Vorgang wird auch filtern genannt.

Kommen wir nun zu zwei Sonderzeichen von MQTT, die keinesfalls in einem Topic auftauchen dürfen: # und +. Diese Zeichen sind Wildcards, Stellvertretersymbole für „alles“ bzw. für eine bestimmte Strukturebene. Sie werden zum Filtern beim Abonnieren (mosquitto\_sub) eingesetzt.

Weitere Beispiele:

```
mosquitto_sub [-u Benutzer -P Passwort] -h 192.168.178.20 -v -t ,#'
```

Bedeutet: Zeige alle Informationen jeden Typs (prefix) aller Geräte (topic) und aller Funktionen an

```
mosquitto_sub [-u Benutzer -P Passwort] -h 192.168.178.20 -v -t stat/#
```

Bedeutet: Zeige alle Informationen des Typs (prefix) stat aller Geräte und aller Funktionen an

```
mosquitto_sub [-u Benutzer -P Passwort] -h 192.168.178.20 -v -t  
stat/Sonoff01/#
```

Bedeutet: Zeige alle Informationen des Typs stat des Gerätes Sonoff01 an

```
mosquitto_sub [-u Benutzer -P Passwort] -h 192.168.178.20 -v -t  
stat/+ /POWER
```

Bedeutet: Zeige alle Power-Zustände aller Geräte an. Man kann auch sagen: Zeige nur die Power-Zustände aller Geräte an.

Leider kennt MQTT bzw. mosquitto keine Wildcards für topics, also folgendes ist nicht möglich:

```
mosquitto_sub [-u Benutzer -P Passwort] -h 192.168.178.20 -v -t  
stat/Sonoff*/POWER
```

Könnte bedeuten: Zeige mir nur die Power-Zustände von Geräten an, deren Namen mit Sonoff beginnt, geht aber leider nicht.

Desweiteren sind Filterungen der Art „Zeig mir alles bis auf XYZ“ auch nicht möglich.

Ein paar Beispiele, falls mosquitto in einem realen [Tasmota](#)-Gerätezoos bereits arbeitet. Am besten den Subscriber-Client starten mit einem Filter auf ein konkretes Gerät mit dem MQTT-Namen GERÄTETOPIC (das ist natürlich ein Stellvertretername für ein real existierendes Gerät):

```
mosquitto_sub [-u Benutzer -P Passwort] -h 192.168.178.20 -v -t  
+ /GERÄTETOPIC
```

Im Fenster des Publisher-Clients z. B. folgende Befehle absetzen und beobachten, was im mosquitto\_sub-Fenster erscheint bzw. wie das Gerät sich verhält:

```
mosquitto_pub [-u Benutzername -P Passwort] -h 192.168.178.20 -t  
cmdnd/GERÄTETOPIC/Power -m 1
```

```

mosquitto_pub [-u Benutzername -P Passwort] -h 192.168.178.20 -t
cmd/GERÄTETOPIC/Power -m 0
mosquitto_pub [-u Benutzername -P Passwort] -h 192.168.178.20 -t
cmd/GERÄTETOPIC/Power -m ON
mosquitto_pub [-u Benutzername -P Passwort] -h 192.168.178.20 -t
cmd/GERÄTETOPIC/Power -m OFF
mosquitto_pub [-u Benutzername -P Passwort] -h 192.168.178.20 -t
cmd/GERÄTETOPIC/Status -m 5
mosquitto_pub [-u Benutzername -P Passwort] -h 192.168.178.20 -t
cmd/GERÄTETOPIC/Status -m 0

```

Vielleicht ist nun schon die Phantasie angeregt, was man mit der direkten Kommunikation über MQTT mit den Geräten machen könnte: Alles. In Verbindung mit einem Bash-Skript (oder einer Batch-Datei, oder der Power-Shell) hat man im Prinzip alles zur Hand, um sich eine eigenen Hausautomation zu bauen !page not found or type unknown

### 3. LWT

Eine besondere Bewandnis hat ein MQTT-Topic, das auch von [Tasmota](#) unterstützt wird: Last Will and Testament (LWT). Das LWT-Topic wird von den Geräten im MQTTBroker hinterlegt und jedem Subscriber, der sich beim Broker anmeldet, gemeldet. Es sei denn, die Anmeldung benutzt Filter. Bei [Tasmota](#) enthält der LWT die Information, ob die Geräte im Wifi sind. Es lässt sich am besten mit einem Gerät überprüfen, dass man ohne großen Aufwand vom Stromnetz trennen kann, z. B. ein Test- und Erprobungs-Sonoff. Das Geräte sollte zunächst unter Strom stehen.

Dann mit folgenden Befehl die Meldungen filtern

```
mosquitto_sub [-u Benutzer -P Passwort] -h 192.168.178.20 -v -t tele/+/LWT
```

Es sollte eine Liste erscheinen mit vergleichbarem Inhalt wie diesem:

```

tele/POW/LWT online
tele/S20neu/LWT offline
tele/RadioStube/LWT online

```

Achte auf das 'online' bzw. 'offline' am Ende jeder Zeile. Nun nimm den Strom von dem Test-Sonoff weg. Nach spätestens 10 Sekunden erscheint das Gerät mit der Payload offline. Lege den Strom wieder an und nach ein paar Sekunden erscheint eine Zeile mit der Message 'online'.

Diese Zustandsmeldung bleibt dauerhaft im MQTTBroker gespeichert und wird jedem Gerät gemeldet, das sich bei diesem Broker anmeldet.

LWT meldet also, ob das Gerät grundsätzlich am MQTT-Broker angemeldet, also erreichbar ist

### 4. Retain (Zurückhalten, Aufbewahren)

Ein mit Retain-Option versehenes [Tasmota](#)-Topic (Powerretain, Buttonretain, Switchretain, Sensorretain) bewahrt MQTT-Meldungen auf.

Powerretain: Der Schaltzustand bleibt im MQTTBroker gespeichert (On/Off) und an Subscriber des Topics stat/# übermittelt.

Buttonretain: Meldungen von Button-Schaltvorgänge werden im Sonoff gespeichert

Sensorretain: (letzte) Sensordaten bleiben im MQTTBroker gespeichert und an Subscriber des Topics tele/+/SENSOR übermittelt

Switchretain: Reatain-Funktion konnte nicht ermittelt werden.

### 5. QoS

Mit QoS ist "Quality of Services" gemeint. Um gleich alle Illusionen zu rauben: Die [Tasmota](#) MQTT Integration unterstützt "nur" QoS 0. Die nachfolgenden Informationen sind nur der Information halber hier genannt:

QoS 0: Broker und Client schicken eine Meldung nur genau einmal raus. Es findet keien Bestätigung des Empfangs statt.

QoS 1: Broker und Client schicken eine Meldung nur genau einmal raus. Es wird eine Bestätigung des Empfangs erwartet.

QoS 2: Broker und Client schicken eine Meldung nur genau einmal raus. Dabei wird ein Handshake in vier Schritten vollzogen.

6. Verschlüsselung  
mosquitto (mosquitto\_pub, mosquitto\_sub) und die [Tasmota](#) MQTT Implementierung unterstützen laut Dokumentation die TLS-Verschlüsselung. Dabei handeln Broker und Gerät die Verschlüsselung beim Verbindungsaufbau aus. In der [Tasmota](#)-MQTT-Dokumentation wird auf den hohen Speicherbedarf bei Verwendung der Verschlüsselung hingewiesen. Das ließt sich wie eine Warnung davor dieses Feature zu verwenden. Auf den Seiten der mosquitto.org kann man mehr zum Einrichten der Verschlüsselung erfahren. Die Konfiguration des Tasmota-Gerätes für die Nutzung der verschlüsselten Kommunikation wird mit dem [Tasmota](#)-Konsolenbefehl `MqttFingerprint` vorgenommen.

## IV - Weitere Beispiele – Spielwiese

1. Weitere interessante MQTT-Optionen (Konsolenbefehle)
2. `SetOption4 0/1` (0 ist Vorgabe/default)  
Normalerweise meldet der das [Tasmota](#)-Gerät einen ausgeführten Befehl in der Form  
`.../.../RESULT {"POWER": "OFF"}`  
an den Broker. Es wird also das Topic `RESULT` benutzt und im JSON Ausdruck steht dann das Topic und der dazu gehörende Wert.  
Setzt man `SetOption4 1` und führt dann den Schaltvorgang durch, zeigt sich diese Darstellung:  
`.../.../POWER {"POWER": "OFF"}`  
Es wird also das allgemeine Topic `RESULT` mit `{"JSON": "AUSDRUCK"}` ersetzt durch das Topic, das angesprochen wurde:  
`.../POWER {"POWER": "OFF"}`.

Dies kann eine relevante Option für Skripte sein, da man das angesprochene Topic direkt mit seinem Namen zurück gemeldet bekommt. Da sich in einem Skript ein solcher Topic-Name in der Regel in einer Variable befindet, kann man denselben Variablenwert benutzen um z. B. die Rückmeldung besser auswerten zu können. Dies geht allerdings auch mit anderen Techniken

3. Grouptopic  
Auf der [Tasmota](#)-Konsole definiert man ein Grouptopic mit `grouptopic NAME`. Werden mehrere Geräte mit demselben Grouptopic ausgestattet, kann man alle Geräte zusammen steuern. Angenommen es gibt im Wohnzimmer dreiverschiedene Lampen, die jeweils einen eigenen Sonoff-Basic mit [Tasmota](#) vorgeschaltet haben. Diese Basics haben das Topic `WZ01`, `WZ02` und `WZ03`. Alle Basic bekommen jeweils das Grouptopic `WZ_Licht`. Dann kann ich mit folgenden Befehl alle Lampen gleichzeitig einschalten:  
`mosquitto_pub [-u Benutzername -P Passwort] -h 192.168.178.20 -t cmd/WZ_Licht/POWER -m 1`
4. Publish  
Das Befehlsword `publish` kann in den [Tasmota-Rules](#) verwendet werden, um mithilfe des MQTT-Protokolls andere Geräte zu steuern oder beliebige Nachrichten abzusetzen, die von einem entsprechend konfigurierten Subscriber ausgewertet werden können.
5. 2. Einbindung von MQTT in bash-Scripte  
Der Forum-User [HoerMirAuf](#) hat in einem Thread ein grundlegendes Skript zur Auswertung von MQTT-Meldungen gepostet. Das sieht so aus:  
Bash

```

#!/bin/bash
#CONFIG
#####
#MQTT-SERVER:
MQHOST=localhost
#Start_Client
#####
mosquitto_sub -i RASP -h $MQHOST -R -v -t '#' | while read RAW_DATA
do
#RF_Bridge
#####
#RF_Wandtaster                               Mitte                               047796
if [ "$(echo $RAW_DATA | grep 'tele/RF_Bridge/RESULT' | grep '047796')" != "" ]; then
mosquitto_pub -h $MQHOST -t cmd/SCHLAF_LED/power -m toggle
mosquitto_pub -h $MQHOST -t cmd/NachtTisch/power -m toggle
fi
#RF_Wandtaste                               Rechts                               047794
if [ "$(echo $RAW_DATA | grep 'tele/RF_Bridge/RESULT' | grep '047794')" != "" ]; then
mosquitto_pub -h $MQHOST -t cmd/SCHLAF_LED/dimmer -m 10
#mosquitto_pub -h $MQHOST -t cmd/SCHLAF_LED/color2 -m '#FFBF00'
mosquitto_pub -h $MQHOST -t cmd/SCHLAF_LED/power -m toggle
fi
done

```

Alles anzeigen

Diese Skript diente in dem konkreten Beispiel zur Auswertung von Signalen, die bei einem Schaltvorgang einer RF-Bridge auflaufen.

Erklärung:

Zeile 5: Definition der Variablen

Zeile 8: Start einer Dauerschleife, die laufend alle Daten vom MQTTBroker "abhört". Die Schleife endet erst in Zeile 23. Alles was dazwischen steht, sind Auswertungen der Nachrichten und je nach Nachrichteninhalt (if ... then) wird ein MQTT-Befehl rausgeschickt.

Zeilen 13 bis 16: Nach Auswertung einer Meldung des MQTTBrokers a) durch einen Filter tele/RF\_BRIDGE/RESULT und einer zweiten Filterung mithilfe des Programms grep ist die Bedingung wahr oder falsch. Wenn sie wahr ist, werden die Anweisungen hinter "then" ausgeführt.

Zeilen 18 bis 22: Eine andere Bedingung (grep ...) wird geprüft.

Es gehört nicht viel Phantasie dazu sich vorzustellen, dass zwischen dem do (Zeile 9) und dem done (Zeile 23) sich zig oder hunderte Prüfungen und Anweisungen befinden können, z. B. Temperatúrauswertungen (tele/sonoff/SENSOR), Helligkeitsauswertungen, Stromverbrauchsauswertungen usw. Die jeweiligen Aktionen, wenn eine Bedingung 'wahr' ist, können ganz verschiedene sein: Ein anderes Gerät steuern, eine E-Mail abschicken, einen Log-Eintrag schreiben und und und. Mit einer gesetzten 'SetOption4 1' könnte man eventuell noch treffsicherer auswerten. Eine Alternative zu grep stellt das Paket jq dar, das eigens für die Verarbeitung von JSON-Strings programmiert wurde. Denn die MQTT-Nachrichten haben ja das JSON-Format. Das wird in meinem Artikel [Tasmota-Meldungen mit jq verarbeiten](#) dargestellt.

#### 6. Filterung von Daten mit jq

Dieses Thema wird in einem eigenen Artikel behandelt.

## V - Quellen, Weiterführende Links

<https://www.msxfag.de/sonst/iot/mqtt.htm>

<https://www.heise.de/developer/arti...TT-3238975.html>

<https://mosquitto.org/>