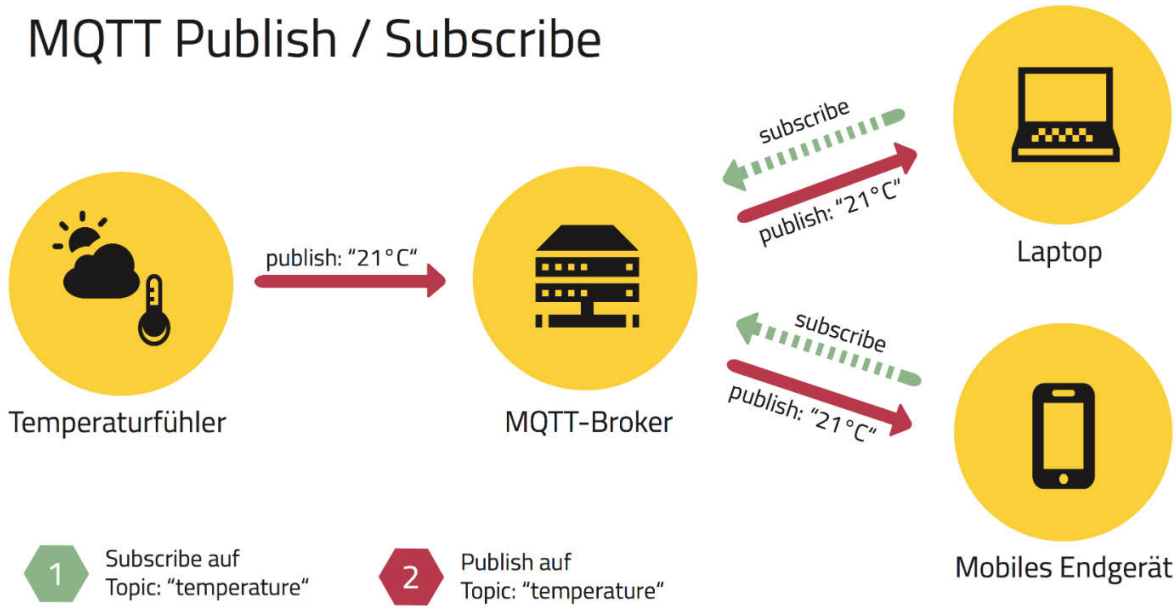
# Lynus Schnittstellenbeschreibung über Mqtt

# Inhalt

| Version | Datum | Bearbeiter/Ersteller | Äderung/Ergänzung | Firma |
|---|---|---|---|---|
| 0.9 | 23.06.2020 | Kai Ebensperger | Dokument erstellt | Lynus AG |
| 0.9 | 23.06.2020 | Stefanie Rinderer | Seite 3 ergänzt | Lynus AG |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |

# 1 Mqtt Grundlagen und Kommunikation

Für die Kommunikation zwischen den Clients (SPS, Microcontroller u.Ä.) und dem Lynus Server (Broker) wurde primär das Mqtt Netzwerkprotokoll verwendet. Über diesen Weg werden die Daten an den Broker geschickt und können dort verarbeitet werden und bei Bedarf wieder zurück an den Client geschickt werden. Mqtt eignet sich durch seinen geringen Overhead gut für solche Applikationen. Zum Schutz wurde der Port 8883 gewählt um den Inhalt der Message zusätzlich über TLS zu verschlüsseln. Das jeweilige Zertifikat muss dafür beim Client hinterlegt werden. Fehlt dieses Zertifikat, kann sich der Client nicht mit dem Broker verbinden. Das Zertifikat kann über die Lynus Webseite bei der Projekterstellung heruntergeladen werden. Die Nachrichten werden über ein spezielles Topic an den Broker geschickt. Dieser wiederum kann über ein zweites Topic Nachrichten an die Clients verschicken. Diese 2 Topics nennt man das Publish und Subscribe Topic. Ein Client Subscribed sich z.B. auf einen Broker mit dem jeweiligen Subscribe Topic. Von diesem kann er somit Daten empfangen, wenn dieser die selbigen published. Wenn die Verbindung beendet wird, muss der Client zuerst eine Unsubscribe Anforderung mit dem Subscribe Topic zum Broker schicken. Der Client schickt über das Publish Topic Daten zum Broker. Zum Anmelden an den Broker brauch es zusätzlich zum Host Name noch einen Benutzernamen und ein Passwort. Dieses Passwort inklusive dem Benutzernamen werden beim Projekterstellen im Lynus Service automatisch erstellt. Damit die Anmeldung am Broker reibungslos funktioniert brauch es zusätzlich zu allen vorherig erwähnten Angaben noch eine Client ID. Wir empfehlen diese Client ID aus dem aktuellen Datum inkl. Uhrzeit und Projektname zu generieren. So ist es fast ausgeschlossen dass eine Client ID 2 mal vorkommt und sich gleichzeitig anmelden versucht. Gleiche Client ID's führen zu Problemen. Als TLS Version wird die Version 1.2 verwendet. Es sollten nicht schneller wie alle 50MS Nachrichten zum Broker geschickt werden. Der Broker schickt alle 250MS einen Wert an den Client. Um einen Online Status des Clients auf Broker Seite realisieren zu können, kann alle 15 Sekunden eine Heart Beat Nachricht an den Client geschickt werden. Siehe Spezifikationen.



Folgend noch einmal alle Verbindungsinformationen im kurzen Überblick:

ClientID : String mit Maximal 255 Zeichen. Empfohlen wird ein Timestamp auf Millisekunden Genauigkeit mit Datum und Projektbezeichnung. (Beispiel = 01011970235959150Testprojekt)

Hostname : mqtt.lynus.io

Port : 8883

Benutzername : String. Wird bei Projekterstellung im Lynus Account erstellt und muss mit angegeben werden. (Beispiel = aaa15527-2a11-4482-8aa3-5fbedd089fe4)

Passwort : String. Wird bei Projekterstellung im Lynus Account erstellt und muss mit angegeben werden. (Beispiel = 071d4e98-50d5-4154-8afe-c2e452d58aec)

Zertifikate : Das CA Zertifikat (Certificate Authority) und das Client Zertifikat (Client Certificate) müssen bei der Verbindung mit angegeben werden und können ebenfalls im Lynus Account heruntergeladen werden.

TLS Version : 1.2

Topic Client-Broker (Publish) : projects/Benutzername/messages

Topic Broker-Client (Subscribe) : projects/Benutzername/messages2

# 2 Spezifikationen

Als Format wie die Daten über Mqtt übertragen werden, wurde die Spezifikation SenML verwendet. Diese Spezifikation eignet sich besonders zum Übertragen von Werten jeglichen Standartformats. Da in Lynus momentan alle Standartformate implementiert sind, kommen wir mit wenigen Labels von SenML zurecht. Anbei alle Lebels, die SenML unterstützt. Die Gelb Markierten sind momentan im Backend von Lynus umgesetzt. Damit ist es schon möglich alle Standarddatentypen umzusetzen. Alle anderen werden in dieser Version nicht unterstützt.

| Name | Labelbezeichnung | Format |
|---|---|---|
| Base Name | bn | String |
| Base Time | bt | Number |
| Base Unit | bu | String |
| Base Value | bv | Number |
| Base Sum | bs | Number |
| Base Version | bver | Number |
| Name | n | String |
| Unit | u | String |
| Value | v | Number |
| String Value | vs | String |
| Boolean Value | vb | Boolean |
| Data Value | vd | Base 64 encodet String with… |
| Sum | s | Number |
| Time | t | Number |
| Update Time | ut | Number |

SenML ist am Ende nichts anderes wie ein JSON String. Jedoch befindet sich der einzelne, oder auch mehrere Datenpunkte immer noch zusätzlich in einem Array. Sprich, vor den {} Klammern das JSON Strings, kommen immer noch die Klammern []. In den [] Klammern kann dann wie oben schon beschrieben, ein Datenpunkt enthalten sein oder auch mehrerer gleichzeitig.

## 2.1 Client-Broker Spezifikation

Um mit dem Lynus Service kommunizieren zu können, muss die Spezifikation vom Client zum Broker wie folgt aussehen :

### Einzelner Datenpunkt

```
[
        {"n":"TemperatureRoom1","v":23.1}
]
```

ODER

```
[
        {"n":"TemperatureRoom1","vs":"Hello Broker"}
]
```

### Mehrere Datenpunkte (max 20)

```
[
        {"n":"TemperatureRoom1","v":23.1},
        {"n":"TemperatureRoom2","v":24.1}
]
```

ODER

```
[
        {"n":"TemperatureRoom1","vs":"Hello Broker"},
        {"n":"TemperatureRoom2","vs":"Again!!!"}
]
```

## 2.2 Client Spezifikation

Der Broker kann bei Bedarf den jeweiligen Wert auch verändert zurück zum Client schicken. Dabei sieht die Spezifikation wie folgt aus :

### Einzelner Datenpunkt

```
[
        {"n":"TemperatureRoom","v":25.1}
]
```

ODER

```
[
        {"n":"TemperatureRoom","vs":"HelloClient"}
]
```

### Mehrere Datenpunkte (max 20)

```
[
        {"n":"TemperatureRoom1","v":24.1},
        {"n":"TemperatureRoom2","v":26.1}
]
```

ODER

```
[
        {"n":"TemperatureRoom1","vs":"Hello Client"},
        {"n":"TemperatureRoom2","vs":"Again!!!"}
]
```

Dadurch dass der Broker den Namen des Datenpunktes wieder zurück an den Client schickt, weiß der Entwickler der Software auf dem Client genau wo der Wert geschrieben werden muss. Somit kann er die Software passend dazu aufbauen.

# 3 Datenübertragung

Die Datenübertragung vom Client zum Broker sollte so ressourcenschonend wie möglich aufgebaut werden. Das heisst, an den Broker sollte auch nur dann etwas geschickt werden, wenn sich der jeweilige Wert auch geändert hat (Eventbasiert). Vom Zyklischen senden wird abgeraten da es enormen und unnötigen Traffic verursacht. Der Broker senden zum Client immer Eventbasiert. Der Variablennamen der nach "n" : folgt darf maximal 48 Zeichen betragen. Nach "v" : kann maximal eine 32 Bit Zahl gesendet werden. Auch Kommazahlen sind möglich. Nach "vs" : sind maximal 255 Zeichen erlaubt. Jedoch sind die Standard Geräte in Lynus für eine maximale Zeichenlänge von 30 Zeichen ausgelegt.

# 4 Heart Beat Nachricht

Um im Backend anzeigen zu können ob der Client noch mit dem Broker verbunden ist, gibt es die Möglichkeit eine Heart Beat Nachricht zu verschicken. Diese Nachricht muss mindesten alle 15 Sekunden 1 mal verschickt werden. Diese Nachricht brauch es, da wir im Idealfall ja immer Eventbasiert Nachrichten an den Broker verschicken. So kann es auch schon mal vorkommen dass über längere Zeit keine «normale» Nachricht an der Broker verschickt wird. Die Heart Beat Nachricht muss wie folgt über die SenML Spezifikation verschickt werden :

```
[
        {"n":"","vs":"hb"}
]
```

# 5 Mqtt Bibliothek Implementierung

Damit der Mqtt Client im Feld draußen mit dem Broker in der Cloud kommunizieren kann, muss das Mqtt Protokoll auch dort implementiert sein. Dazu verwenden wir die «Eclipse Paho» Bibliothek. Diese Bibliothek ist in den gängigsten Programmiersprachen verfügbar. Somit sollte eine Implementierung auf unterschiedlichsten Geräten problemlos möglich sein.

https://www.eclipse.org/paho/

Alternativ kann das Mqtt Protokoll auch zur Gänze selber implementiert werden.

https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html

# 6 Link

Für eine detailliertere Erklärung der Spezifikation klicken sie bitte auf folgenden Link :

https://tools.ietf.org/html/rfc8428

https://de.wikipedia.org/wiki/MQTT