22 MQTT

Introduction

MQTT (short for MQ Telemetry Transport) is an open OASIS and ISO standard (ISO/IEC PRF 20922) lightweight, publish-subscribe network protocol that transports messages between devices. The protocol usually runs over TCP/IP, although its variant, MQTT-SN, is used over other transports such as UDP or Bluetooth. It is designed for connections with remote locations where a small code footprint is required or the network bandwidth is limited.

The broker acts as a post office, MQTT doesn't use the address of the intended recipient but uses the subject line called "Topic", and anyone who wants a copy of that message will subscribe to that topic. Multiple clients can receive the message from a single broker (one-to-many capability). Similarly, multiple publishers can publish topics to a single subscriber (many to one).

Each client can both produce and receive data by both publishing and subscribing, i.e. the devices can publish sensor data and still be able to receive the configuration information or control commands. This helps in both sharing data and managing and controlling devices.

With MQTT broker architecture the devices and applications become decoupled and more secure. MQTT might use Transport Layer Security (TLS) encryption with a user name, password-protected connections, and optional certifications that require clients to provide a certificate file that matches the server's. The clients are unaware of each other's IP address.

The broker can store the data in the form of retained messages so that new subscribers to the topic can get the last value straight away.

The main advantages of an MQTT broker are:

- Eliminates vulnerable and insecure client connections
- Can easily scale from a single device to thousands
- Manages and tracks all client connection states, including security credentials and certificates
- Reduced network strain without compromising the security (cellular or satellite network)

Each connection to the broker can specify a quality of service measure. These are classified in increasing order of overhead:

- At most once - the message is sent only once and the client and broker take no additional steps to acknowledge delivery (fire and forget).
- At least once - the message is re-tried by the sender multiple times until acknowledgement is received (acknowledged delivery).
- Exactly once - the sender and receiver engage in a two-level handshake to ensure only one copy of the message is received (assured delivery).

Using WCC Lite as an MQTT Client

MQTT serves as an alternative for protocols conforming to IEC standards, for example, to send data to a cloud infrastructure that supports MQTT instead of IEC-60870-5-104.

WCC Lite supports MQTT messaging compatible with MQTT v3.1 standard (starting from version**v1.4.0**). Such messaging is possible via mapping of Redis and MQTT data therefore data can be transmitted from any protocol that is supported by WCC Lite.

All standard functions, except for data encryption, are supported. Encrypted messages are not supported yet, therefore to ensure security a user would have to use a VPN service. A user can choose from three different Quality of Service levels, select if messages are to be retained, authenticate users and optionally send Last Will messages.

To configure WCC Lite a user can fill in the needed parameters in Excel configuration. These parameters are shown in the two tables below.

Table. MQTT parameters for the Devices tab

Parameter	**Type**	**Description**	**Required**	**Default value** (when not specified)	**Range**	
					Min	Max
name	string	User-friendly device name	Yes			
device_alias	string	Device alias to be used in the configuration	Yes			

enable	boolean	Enabling/disabling of a device	No	0	0	1
protocol	string	Selection of protocol	Yes		MQTT	
ip	string	MQTT broker IP address/Domain name selection	Yes			
port	integer	MQTT broker port selection	No	1883		
enable_threshold	boolean	A parameter to determine if identical values should not be sent multiple times in a row.	No	1	0	1
mqtt_qos	integer	MQTT Quality of Service for the message as in standard	No	0	0	2
mqtt_retain	boolean	Selecting if the MQTT broker should retain the last received messages	No	0	0	1
username	string	MQTT user name	Yes			
password	string	MQTT user password	Yes			
auth	string	Selecting if TLS should be used	Yes		none, password, tls	
ca_certificate	string	Certificate authority file for TLS connection	Yes (If auth = tls)			
client_certificate	string	Client certificate file for TLS connection	Yes (If auth = tls)			
client_key	string	The private key that corresponds to the client certificate for TLS connection	Yes (If auth = tls)			
use_last_will	boolean	Selecting if MQTT should use the Last Will and Testament functionality (Default: False)	No	0	0	1
last_will_topic	string	Topic to which an MQTT message would be sent if the device abruptly disconnected the message broker	Yes (If use_last_will = True)			
last_will_message	string	Message to be sent over MQTT if the device abruptly disconnected message broker	No			
last_will_qos	integer	MQTT Quality of Service selection as in standard	No	0	0	2
last_will_retain	boolean	Selecting if the MQTT broker should retain the last will message	No	0	0	1
client_id	string	User-friendly name for client ID	No			

To map the signal to send through the MQTT client, it should have its device_alias and signal_alias mapped to source_device_alias and source_signal_alias respectively.

Table. MQTT parameters for the Signals tab

Parameter	**Type**	**Description**	**Required**	**Default value** (when not specified)	**Range**	
					Min	Max

signal_name	string	User-friendly signal name	Yes			
device_alias	string	Device alias from a Devices tab	Yes			
signal_alias	string	Unique signal name to be used	Yes			
source_device_alias	string	device_alias of a source device	Yes			
source_signal_alias	string	signal_alias of a source signal	Yes			
enable	boolean	Enabling/disabling of an individual signal	No	1	0	1
log	integer	Allow signal to be logged. Log signal with 1 and no logging with 0.	No	0		
topic	string	Topic name to override the value built by default	No			
periodic_update_ms	integer	Signal value is published periodically according to the value set.	No	-	-	-

MQTT data format

The format of a MQTT message is a bit different than Redis messages. Redis messages are supported as CSV strings: value, timestamp, flags (where value can be float, integer or nan; timestamp - Unix timestamp in milliseconds; flags contain additional information about a measurement). MQTT messages are supported as value, timestamp, and quality (where value can be float, integer or nan; timestamp - Unix timestamp in milliseconds; quality shows if a value is to be considered valid). Quality parts of a string are always equal to 1 except for Redis messages containing invalid (IV), non-topic (NT) and/or overflow (OV) flags.

As mentioned, the MQTT client acts as an adapter between Redis and MQTT, therefore data from the topic in Redis is written to a topic in MQTT. Therefore mqtt-client has to know the mapping table before starting. This table is saved at /etc/elseta-mqtt.json. Every Redis topic name is constructed as tag/[device_alias]/[signal_alias]/[direction]. Prefix tag/ is always used before the rest of the argument. `device_alias` and `signal_alias` represent columns in Excel configuration. Direction can have one of four possible values - rout, out, in, rin; all of which depend on the direction data is sent or acquired protocol-wise. The same Redis topic structure is preserved in MQTT by default making it easier to find matching signals, however, as no recalculation is done by MQTT and only PUBLISH messages are now supported, only Redis signals within direction have their MQTT mappings.

A user can create and select his topic name in Excel configuration, in the topic column. As no recalculation is done by MQTT and only PUBLISH messages are now supported, only Redis signals within in direction have their MQTT mappings.

Debugging a MQTT protocol

If the configuration for MQTT is set up, a handler for the protocol will start automatically. If the configuration is missing parameters or contains errors, the protocol will not start. It is done intentionally to decrease unnecessary memory usage.

MQTT Client command line debugging options

- **Step 1**: Service must be stopped by entering the following command into the wcclite: **/etc/init.d/mqtt-client stop**
- **Step 2**: After the service is stopped it must be started with the preferred configuration file (JSON files found in the /etc/ folder) and a debug level 7: **mqtt-client -c /etc/mqtt-client/mqtt-client.json -d7** Additional output forming options described in the table below.
- **Step 3**: Once the problem is diagnosed normal operations can be resumed with the following command: **/etc/init.d/mqtt-client start**

```
-h [ –help ] Display help information
-c [ –config ] Configuration file location (default - /etc/elseta-mqtt.conf)
-V [ –version ] Show version
-d<debug level> [ –debug ] Set debugging level
-r [ –redis ] Show REDIS output
```

If the MQTT Client does not work properly (e.g. no communication between devices, data is corrupted, etc.), a user can launch a debug session from the command line interface and find out why the link is not functioning properly.

To launch a debugging session, a user should stop `mqtt-client` process and run `mqtt-client` command with respective flags as was shown above.

Revision #6
Created 22 November 2024 07:15:14 by Gabriele
Updated 14 February 2025 08:56:51 by Andrej