

MQTT Protocol for Standard Interfaces

Revision history

Version	Date	Author	Description
1.0	12-03-2025	SML	Initial version of the MQTT protocol.
	24-06-2025	PDA	Release

Review history

Version	Date	Reviewer	Notes / description
1.0	12-03-2025	AEF	Review
	24-06-2025	PDA	Release

Table of Contents

1 Introduction	4
1.1 References	4
1.2 Glossary	4
1.3 Copyright and right to use under Apache License 2.0	5
1.4 Background / 5G-Robot	5
2 Communication	7
2.1 Communication Protocol	7
2.2 MQTT Configuration	7
2.2.1 Broker Configuration	7
2.2.2 Topic Naming Conventions	7
2.2.3 Authentication	8
2.2.4 Quality of Service (QoS)	8
2.2.5 Retention	8
2.2.6 Sessions	8
2.2.7 Connection Timeout to the Broker	8
2.2.8 Communication Monitoring and Last Will and Testament	8
2.3 Sequence Numbering and Acknowledgements	9
2.3.1 Sequence Number & Acknowledging Incoming Messages	9
2.4 Keep-alive Messages	9
3 Telegram Format	11
3.1 Formats	11
3.2 Header Information	11
4 Mapping Telegram Header to Topic	12
5 Common Telegrams	13
5.1 Acknowledgement (HLC <-> Client)	13
5.2 Heartbeat (HLC <-> Client)	15

1 Introduction

This document describes the specification for the MQTT protocol and how it can be used to implement any of Intelligent System's standard interfaces [1].

The document details a MQTT protocol designed for robust client-to-client communication, specifically between the High-level Control (HLC) and connected equipment/systems. While standard interfaces for equipment, fleet, and ERP systems are technology-neutral, the MQTT protocol outlined herein represents a viable technological implementation option. Determining to use this MQTT protocol for the standard interfaces, should be based on a comparison to the other technologies such as REST, socket, etc. to find the most appropriate for the use case.

1.1 References

ID	Document	Description
[1]	Title: Standard Interface	The base interface description used for all the open interfaces, to be found at Intelligent Systems website: https://www.intelligentsystems.dk/products-keep-customers-at-forefront-of-technology/
[2]	Title: TCP/IP Model vs OSI model	https://www.fortinet.com/resources/cyberglossary/tcp-ip-model-vs-osi-model
[3]	Title: MQTT	https://mqtt.org/
[4]	Title: VerneMQ	https://vernemq.com/
[5]	Title: Introducing JSON	https://www.json.org/json-en.html
[6]	Title: The JSON data interchange syntax 2nd edition, December 2017	https://www.ecma-international.org/publications-and-standards/standards/ecma-404/
[7]	Title: ISO 8601 Date and time format	https://www.iso.org/iso-8601-date-and-time-format.html

1.2 Glossary

Term	Description
Broker	A server that receives messages from publishers and distributes them to subscribers based on topics.
Message ID	A unique identifier for each telegram.
MQTT (Message Queuing Telemetry Transport)	A lightweight messaging protocol used for machine-to-machine (M2M) communication. It operates on a publish/subscribe model.
MQTTS	MQTT is used with TLS (Transport Layer Security) for secure communication.
Prefix	A part of the topic structure used for separating topics into specific topic spaces.

Publish/Subscribe	A messaging pattern where publishers send messages to topics, and subscribers receive messages from topics they are subscribed to.
System ID	A unique identifier for a specific system or piece of equipment.
Topic	A string that the broker uses to filter messages. Publishers send messages to topics, and subscribers subscribe to topics to receive messages.

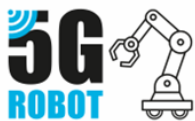
1.3 Copyright and right to use under Apache License 2.0

The copyright on this document and any contained specifications, designs, etc. belongs to the company: Intelligent Systems A/S, Havnevej 11, DK-9560 Hadsund, DENMARK.

Any referenced 3rd party designs, technologies and or other IP shall remain with the original owners.

The reference architecture, designs and the included open standard integration interfaces are open and free to use under Apache License 2.0. For details please see <https://www.apache.org/licenses/LICENSE-2.0>.

1.4 Background / 5G-Robot



Parts of this document / release was made in the **5G-Robot** project also known under the long name **5G-ENABLED AUTONOMOUS MOBILE ROBOTIC SYSTEMS** - the largest innovation project that has been launched under the Innovation Fund Denmark's (IFD) Grand Solutions program.

The groundbreaking project united Denmark's leading robot, automation and factory digitalization companies as technology vendors, research partners and industry-leading end-user companies.



Illustration: Project partner logos.

The aim of the project was to revolutionize manufacturing - paving the way to smart production and smart factories and the application of a number of new technologies in production and manufacturing including 5G wireless communication, cloud and edge computing and digital twin.

Intelligent Systems played a leading role in the project, providing the glue that ties the robotic solutions of the partners together making the work as one - i.e. one connected integrated intelligent manufacturing system.

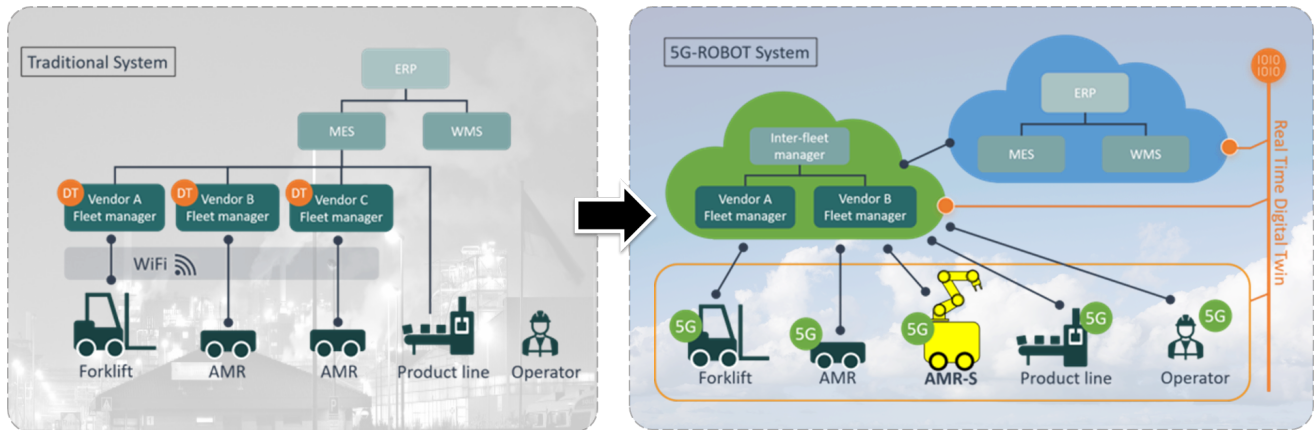


Illustration: The aim is to revolutionize manufacturing paving the way to smart production and smart factories.

Read more about the 5G-Robot project here: <https://www.5gsmartproduction.aau.dk/5g-robot>

2 Communication

This section contains the description of the protocol used for communication with the interface.

If considering the TCP/IP model [2], the aim of the protocol described in this section lies at the application layer, while handling some of the responsibilities of the network layer. It handles aspects such as authentication, reliability and routing for the communication between systems. The application data itself, i.e. the data transferred between the HLC and the system, will be through the telegrams described in the system's interface.

2.1 Communication Protocol

This interface specification utilizes MQTT v3.1.1 for communicating between the HLC and the equipment/system. To enable secure communication, it will be used with TLS, and thus can be referred to as MQTTS [3].

MQTT uses a publish/subscribe architecture, where end-systems can publish messages to specific topics and send these to a broker. Other systems can then subscribe to these topics and receive them by the broker. The messages are received in a push-configuration, opposite to pull-based, e.g. a traditional REST interface, where end-systems will periodically need to poll for new data. While designed for many-to-many communication, MQTT will in this specification be used for one-to-one communication.

2.2 MQTT Configuration

This section presents the configuration of MQTT-related aspects.

2.2.1 Broker Configuration

VerneMQ [4] is chosen as the broker of choice due to its focus on high performance and scalable design. However, any broker implementing the MQTT v3.1.1 specification may be used with this interface specification.

For actual setups with multiple systems and devices, care should be taken to avoid bottlenecks and single point-of-failures in the communication. If required, the communication can be segmented with multiple brokers. Otherwise, to improve scalability, deploying multiple brokers in a cluster-configuration allows for higher scalability and reliability. This is also supported by the selected broker [4].

2.2.2 Topic Naming Conventions

Telegrams related to the system's interface should use the following topic structure:

`{prefix}/{system-id}/{publisher}/...`

where

{prefix}: Optional prefix to help separating topics as needed into specific topic spaces. Note, the prefix can contain multiple topic levels. If a prefix should be used, it is specified in the system's specialized interface.

{system-id}: A unique identifier for the system, e.g. the name of the equipment or system.

{publisher}: Either **equipment** or **hlc**. Using MQTT, if the publisher is also subscribed to a topic they will receive the same message. To avoid this, the publisher uses their identifier as part of the topic. The HLC will only publish messages to the with the *hlc* identifier, and the

equipment likewise with the *equipment*. The specific direction identifier used for the publisher can also be overridden by any specialization in the equipment's interface. Note that the *equipment* does not necessarily mean a physical piece of equipment; it can also be e.g. a business system, however, the term *equipment* has been chosen to remain generic.

Example topic for a palletizing cell publishing its state: *palletizing-cells/cell-3821/client/system/state*

Note that the topic naming is **case-sensitive**. It is recommended to only use **lowercase ASCII letters**, **numbers** and **dashes (-)** for each topic level (between each slash (/)).

The specific values used for prefix, system-id, and publisher is part of the header information for the telegram. See [Header information](#) and [Mapping Telegram Header to Topic](#) for more information.

2.2.3 Authentication

To ensure that only intended clients have access to the broker, it is recommended to set up authentication and authorization on the broker. By doing so, it is possible to limit which clients can receive and transmit messages in a given topic space. This reduces the risk of malicious clients that may try to gain access to or impact the system.

It is recommended that all clients connecting to the broker must be authenticated and only allowed to do authorized actions. Thus, the broker must not permit anonymous access. On connecting to the broker, clients must provide their own unique credentials that authenticate their access to the broker. The broker must use the received credentials to enforce which topics the client can subscribe and publish to.

2.2.4 Quality of Service (QoS)

For both subscribing and publishing, a QoS level of 0 (at least once) is used. To handle the reliability-aspect of the connection, please see [Sequence Numbering and Acknowledgements](#).

2.2.5 Retention

It is recommended that messages are **not** retained. This is to avoid synchronization issues where new clients receive obsolete data when subscribing to the broker. Instead, this client's standard interface provides handles that a new or reconnected client must use to request data and get into a synchronized state.

2.2.6 Sessions

Both systems should connect with a clean session upon connection and reconnection. The sequence numbering and acknowledgement concept described in section [Sequence Numbering and Acknowledgements](#) are designed to handle the reliability aspects and using MQTT sessions are not required.

2.2.7 Connection Timeout to the Broker

A suitable connection timeout interval to the broker must be determined to ensure that the broker does not close the connection to the client in case no keep-alive message has been received in time.

2.2.8 Communication Monitoring and Last Will and Testament

It is advised **not** to use Last Will and Testament as this state is monitored by the broker on the basis of the keep-alive interval configured by the client when establishing its connection with the broker. Instead, clients should rely on the application level keep-alive concepts described in [Keep-alive Messages](#).

2.3 Sequence Numbering and Acknowledgements

To ensure the end-to-end communication and the order of the telegrams, the interface contains a sequence and acknowledgement concept. This is necessary to obtain the reliability and low-latency demands. Using MQTT's quality of service 1 (at most once) and 2 (exactly once) **only** ensures the communication between a client and the broker, but not from client to client, e.g. High-level control system to the equipment's PLC.

Furthermore, QoS 2 will result in a significant increase of control messages due to the four-part handshake between the sender and receiver for each telegram, and thus a higher latency in the communication.

The acknowledgement concept also ensures that out-of-order telegrams are handled in a reliable manner, which MQTT cannot guarantee.

2.3.1 Sequence Number & Acknowledging Incoming Messages

See Standard Interface [1] for information on sequence numbers and acknowledgment of incoming messages.

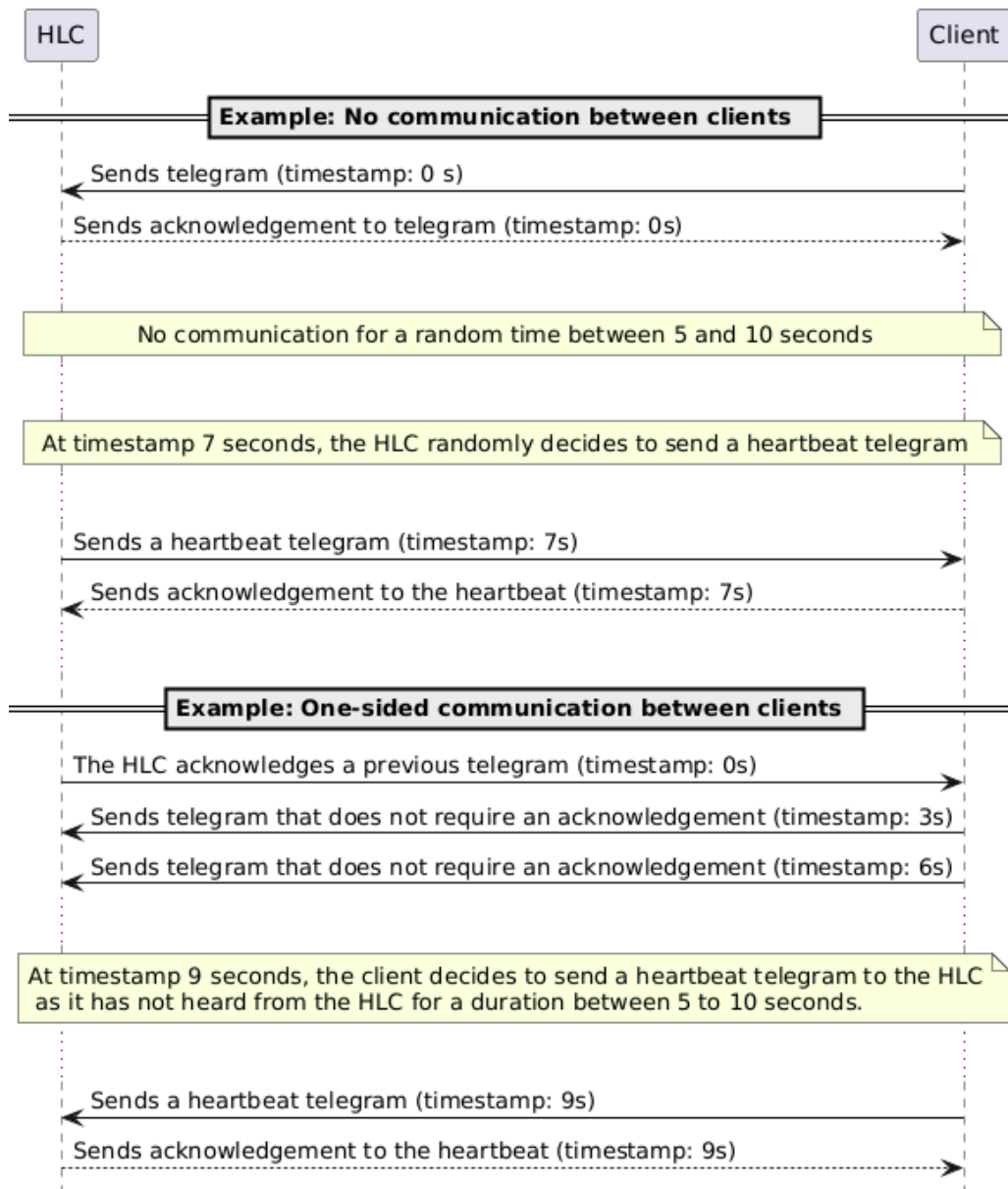
2.4 Keep-alive Messages

It is both sides' responsibility to monitor the receiving and transmission of telegrams to ensure that the connection is active. If there is a period of no or one-sided communication (e.g. communication with telegrams that do not require acknowledgements), clients must take action to verify that the other party is still connected.

In this protocol, it is the client that has not heard from the other client that emits a [Heartbeat](#) to determine if the connection is still alive. If the keep-alive telegram is acknowledged, the connection is alive. Opposite, if no acknowledgement is received after 3 retransmissions, the connection should be considered faulty.

The specific interval for when a keep-alive message must be sent should be random within a certain time frame. This is to reduce the risk that both clients send a [Heartbeat](#) in the same instance. The default configuration is that a client should send a [Heartbeat](#) if it has not heard from the other client within a random duration of 5 and 10 seconds.

An example of keep-alive messages can be seen below.



3 Telegram Format

This section describes the structure and format of the message telegrams.

3.1 Formats

All payloads will use the JSON format with UTF-8 encoding. See [5] or [6] for details on the JSON format.

The following data types are supported:

- Number (integer and decimal)
- String
- Boolean (lowercase true/false)
- Array
- Object
- Date (formatted as a String following the ISO 8601 standard [7])
- Null
 - The value for the property should be *null* without quotation marks. Alternatively, the key can be omitted entirely if the value is null.

Object types can consist of arrays of a data type or a collection of more properties.

Note, that the formats used in payloads may be extended in the client's specific interface.

3.2 Header Information

This MQTT protocol specifies a set of properties that should be included in every telegram unless otherwise noted in the specific telegram. These properties include interface version, sequence number, acknowledgement, retransmissions, message ID, and request ID.

The header also contains other information such as the telegram type ID, prefix, and system ID that can be mapped into the specific MQTT topic. Refer to [1] to see other information that should also be added to the header. Thus, the system-id and telegram-type-id fields are *omitted*.

For more information on mapping the header into a specific topic refer to [Mapping Telegram Header to Topic](#).

Example

```
{
  "header": {
    "version": "v1.0",
    "message-id": "504c198d-6140-4b69-b54d-c21eafb732de",
    "seq-no": 14,
    "need-ack": false,
    "retransmitted": 2,
    ... // Other header information
  },
  ... // telegram properties go here
}
```

4 Mapping Telegram Header to Topic

As described in [Header information](#), the header contains information such as telegram type ID, prefix, and system ID. This information must be mapped into the specific topic that the telegram is transceived on.

The telegram type ID is used to determine the MQTT topic a telegram is sent unless a specific topic is specified (e.g., for [Acknowledgement](#), [Heartbeat](#), etc.). Note, that the topic should be prefixed with additional information as described in [Topic naming conventions](#).

Compound words in the Telegram type IDs are joined with punctuations (i.e., system.state). To convert the telegram type ID to the specific topic where the telegram should be sent, every punctuation must be replaced with a slash (/). For instance, the telegram type ID 'system.state' can be mapped to the topic 'system/state'.

Other information required for the topic, such as system-id, can be derived directly from the properties in the header that have the identical names. The publisher is implicitly derived based on which system that is sending the message. See [Topic naming conventions](#) for more information on deriving the publisher. The specific values used for prefix and publisher may be overridden in the specific interface that utilizes the MQTT Protocol.

Example

Telegram Name To Topic	Entire Topic to Use
Telegram name: <i>system.state</i> Topic name: <i>system/state</i>	<p>In the example, the following values are used for prefix, system-id, and publisher.</p> <p>prefix: <i>palletizing-cells</i> system-id: <i>cell-3821</i> publisher: <i>cell</i></p> <p>Full topic the telegram should be sent to: <i>palletizing-cells/cell-3821/cell/system/state</i></p>
Telegram name: <i>traffic-mangement.zones.event</i> Topic name: <i>traffic-management/zones/event</i>	<p>In the example, the following values are used for prefix, system-id, and publisher.</p> <p>prefix: <i>interfleet</i> system-id: <i>fleet-1</i> publisher: <i>fleet</i></p> <p>Full topic the telegram should be sent to: <i>interfleet/fleet-1/fleet/traffic-management/zones/event</i></p>

5 Common Telegrams

5.1 Acknowledgement (HLC <-> Client)

Telegram Type ID: ack

When a data telegram is received, an acknowledge telegram must always be returned immediately. For more information, see [Sequence Numbering and Acknowledgements](#).

The acknowledgement is always sent on the topic specified below.

Note acknowledgements do not have sequence numbers in their header.

Topic: .../ack

Properties

Property	Type	Description
seq-no	Number	The same sequence number as in the telegram being acknowledged.
error-code	String or null	An error code indicating a faulty package. If omitted it corresponds to <i>NO_ERROR</i> .
reason	String or null	String containing a human-readable error message or null.

Error codes

Note, that other possible error codes exist. These will be defined in the specific client's interface.

Error code	Description
NO_ERROR	Telegram was accepted without any issues.
REPEATED_TELEGRAM	Sent, when a telegram is – based on sequence number - considered "repeated". Telegram will <u>not</u> be processed.
SEQ_TOO_HIGH	Sent when a sequence number is larger than current value is received. The telegram will be processed and the sequence counter will be adjusted by the receiver.
SEQ_TOO_LOW	Sent when the sequence is too small. Telegram is considered too small when deviating more than -1 (default) from the expected sequence number. Else it is considered as a repeat telegram. Telegram will <u>not</u> be processed.

Examples

Successful acknowledgement

```
// Successful acknowledgement (implicit no error)
{
  "header": {
    ... // Contents omitted for clarity
  },
  "seq-no": 2255
}

// Successful acknowledgement (explicit no error){
{
  "header": {
    ... // Contents omitted for clarity
  },
  "seq-no": 2255,
  "error-code": "NO_ERROR"
}

// Synchronous error telegram
{
  "header": {
    ... // Contents omitted for clarity
  },
  "seq-no": 2255,
  "error-code": "BAD_REQUEST" // Client interface specific error code
}
```

5.2 Heartbeat (HLC <-> Client)

Telegram Type ID: heartbeat

When a client has not sent any messages for a specified period of time, it should send a heartbeat to notify the other client that it is still alive. This message must be acknowledged by the receiver. For more information about this telegram and how it is used, see [Keep-alive Messages](#).

Topic: .../heartbeat

Properties

Property	Type	Description
timestamp	Date	Timestamp from when the heartbeat was sent.

Example

```
{
  "header": {
    ... // Contents omitted for clarity
  },
  "timestamp": "2025-03-11T09:01:50Z"
}
```