IALA GUIDELINE

GNNNN
HARMONISED IOT PROTOCOL FOR VISUAL ATON

Edition 1.0
June 2025

urn:mrn:iala:pub:gnnnn:ed1.0

10, rue des Gaudines - 78100 Saint Germain en Laye, France
Tel. +33 (0)1 34 51 70 01 - contact@iala.int
www.iala.int
International Organization for Marine Aids to Navigation

DOCUMENT REVISION

Revisions to this document are to be noted in the table prior to the issue of a revised document.

Date	Details	Approval
June 2025	First issue	Council 02

CONTENTS

List of Tables

List of Figures

1. INTRODUCTION

Marine Aids to Navigation (AtoN) have often been early adopters of new technologies. Since the 1980s, remote monitoring of marine signal lanterns has been available as a tool to track the availability of AtoN and predict maintenance needs. Remote Control has also been implemented in some applications. Today, there are various solutions available on the market based on Satellite Communication, mobile networks, Point-to-Point short-range radio communication, as well as Automatic Identification System (AIS) transponders.

However, current communication topologies often have a low reporting frequency due to the limitations of data communication costs or energy constraints. Status reports are typically only transmitted when lights turn on in the evening and turn off in the morning, with additional ad hoc reports transmitted when an issue is detected by the station (e.g., position, energy or light operation related). Also, many older systems have a limitation in the number of simultaneous sessions they can manage through bandwidth and processing.

As a result, the owner of the asset may have outdated information and no real-time situational awareness. They may also not be able to detect a malfunction of an AtoN in a timely manner. Due to the lack of industrial standards, each vendor operates a proprietary protocol and system, making it difficult for the owner of assets to mix devices in the field.

2. SCOPE

This guideline outlines how an established Internet of Things (IoT) system can be utilised to provide a harmonised efficient, affordable protocol for connected visual AtoN. This protocol will allow commonality of data exchange, providing interoperability of visual AtoN equipment from a mix of suppliers who adopt this approach.

This guideline explores how to achieve a standard interoperable solution, outlining the benefits achieved, with a detailed implementation in the appendix. This being a prerequisite to a common platform.

The guideline does not explore how to develop the infrastructure to achieve an operational system, but does refer to some aspects as part of a means of informing the reader.

3. SUITABLE PROTOCOLS

There are several communication protocols that are commonly used in an IoT ecosystem to support the transfer of data between devices and systems. Some of the most widely used protocols include:

- Message Queuing Telemetry Transport (MQTT): A lightweight publish/subscribe protocol designed for machine-to-machine (M2M) communication.
- Constrained Application Protocol (CoAP): A protocol designed for resource-constrained devices in an Internet of Things (IoT) networks.
- Advanced Message Queuing Protocol (AMQP): An open standard for message queuing and data transfer in an IoT system.
- Open Platform Communications Unified Architecture (OPC UA): A vendor-independent communication protocol for industrial automation and control systems.
- Light weight Machine to Machine (LwM2M): A protocol designed for managing and updating device firmware and configuration over the air.

These protocols are used in various IoT applications such as industrial control, monitoring, automation, real-time data collection and analysis, and M2M communications.

Feature / Protocol	MQTT	CoAP	AMQP	OPC UA	LwM2M
Purpose	Lightweight messaging for IoT	Lightweight RESTful protocol	Robust messaging for enterprise systems	Industrial automation and interoperability	Device management and telemetry
Bi-directional	Yes	Limited	Yes	Yes	Yes
Transport Layer	TCP	UDP	TCP	TCP (or HTTPS, OPC UA over UDP/MQTT)	UDP (typically with CoAP), TCP possible
Architecture	Publish/Subscribe	Request/Response, RESTful	Publish/Subscribe, Point-to-Point	Client/Server, Publish/Subscribe	Client/Server
Message Overhead	Low	Very Low	Moderate to High	Moderate	Very Low
Security	TLS/SSL	DTLS	TLS/SSL, SASL	TLS, X.509 certificates, encryption	DTLS, TLS
Data Format	Binary (protocol), Payload: JSON, text, etc.	Binary	Binary (AMQP-specific)	Binary (UA-specific), JSON	TLV, JSON, CBOR
QoS Levels	Yes (0, 1, 2)	Confirmable/Non-confirmable	Yes (settled/unsettled delivery)	No (relies on transport)	No (relies on underlying protocol)
Installed Base (2025 Estimate)	~2-3 billion devices	~300-500 million devices	~500-800 million devices	~50-100 million devices	~100-200 million devices
Use Case	Real-time data streaming, telemetry	Resource-constrained devices	Enterprise messaging, complex workflows	Industrial IoT, machine-to-machine communication	IoT device management

Table 1 Comparison of various IoT protocols

For all acronyms in the table, please refer to section 8 – Acronyms.

3.1. PROTOCOL RATIONALE

The most appropriate protocol for a particular case depends on factors such as data rate, latency, security, power consumption, compatibility, scalability, and complexity.

Given the factors and considering the demands for the Visual AtoN IoT harmonization, the MQTT protocol has been selected for this implementation. This protocol is considered the most suitable for small low-power applications and is one of the most commonly used communication protocols. The implementation of a MQTT broker is straightforward to establish and integrates well with an already established remote monitoring system.

This protocol was developed by IBM in 1999 for bandwidth efficiency, lightweight and low power for battery powered devices. In 2014 the protocol was developed into an industrial standard by the Organization for the Advancement of Structured Information Standards (OASIS) as a royalty free open standard

3.2. KEY FEATURES

Below are some of the key features associated with the MQTT protocol:

1 Clients: An MQTT client can be either a publisher or a subscriber. Publishers are responsible for sending messages to the broker, while subscribers are interested in receiving those messages.

2 Broker: The MQTT broker acts as an organizer or coordinator of data between publishers and subscribers. It receives messages from the publishers and then distributes them to the subscribers.

3 Topics: These are subject areas on which messages are sent and received, which act as channels for communications. Topics are hierarchical in nature, allowing for the creation of a tree-like structure that can be used to organize messages by topics.

4 Quality of Service (QoS): MQTT supports three levels of QoS for message delivery:

   a QoS 0, just sends the data without an acknowledgement, which provides at most a single delivery.

   b QoS 1, provides at least a single delivery with an acknowledgement of receipt.

   c QoS 2, ensures that exactly a single delivery is received with an acknowledgement at both ends.

5 Keep Alive: MQTT uses a “Keep Alive” mechanism to ensure that clients remain connected to the broker even if there is no data to transmit. The clients will send a periodic ping request (PINGREQ) message to the broker to indicate that they are still connected.

6 Last Will and Testament (LWT): MQTT supports a LWT feature that allows a publishing client to specify a message that will be published by the broker should the client becomes disconnected unexpectedly.

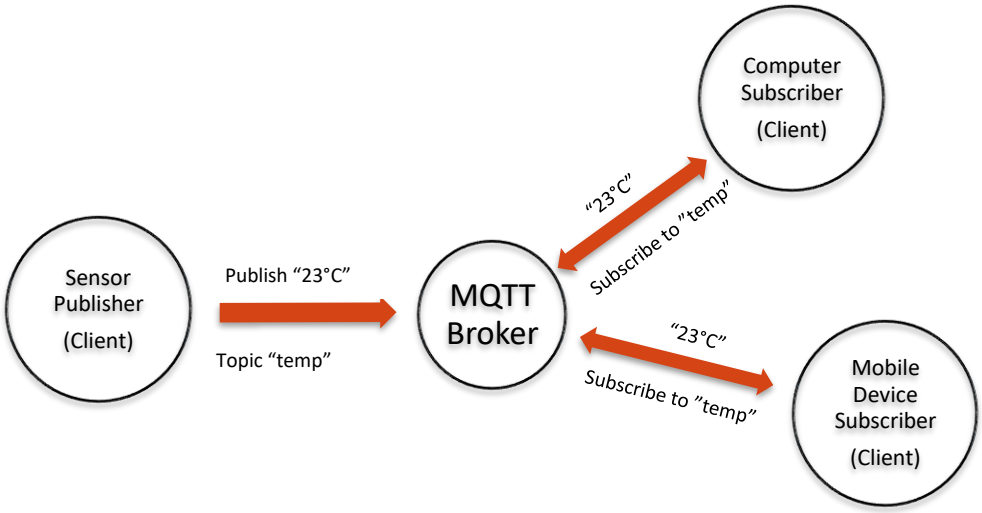

Figure 1 An example of publishing and subscribing

4. IOT INFRASTRUCTURE

When considering the development of a protocol for interoperability of visual AtoN, the AtoN equipment can be considered as an IoT device with various sensors. This device needs to send its data to an IoT application to process and display to users. An IoT platform provides the infrastructure to pass this information between the IoT device and the IoT application and can be considered as in integrated service. Such services can be provided (hosted) by a commercial service providers or can be delivered within the organisation existing infrastructure.

4.1. TOPOLOGY

There are many topological arrangements that can be adopted using various infrastructure option. The type of topology adopted, will be subject to the how the IoT service is to be provided. Weather this is a simple system or one integrating into an existing organisational Information Technology (IT) platform. This section shows two examples.

4.1.1. A SIMPLE HOSTED IOT PLATFORM.

The following diagram shows how AtoN equipment can be connected using a commercially provided IoT platform. This hosted platform will then be able to provide the various services needed to deliver a successful MQTT implementation including the necessary security. Below identifies the key components.

- An AtoN using Transmission Control Protocol/Internet Protocol (TCP/IP) with native MQTT interface. This could be one of many AtoN.
- The transmission channel can be wired or mobile
- Platform provides various services such as broker, access control, security, and application.
- Users access the platform via an application.

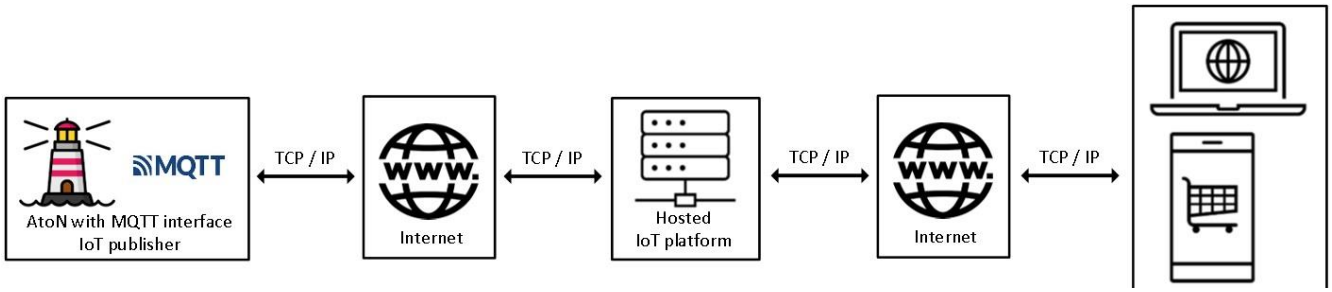

Figure 2 Topology of a hosted IoT solution

4.1.2. A HYBRID IMPLEMENTATION.

It is also possible to develop an MQTT system that operate over various industrial protocol to deliver the data to the user. An AtoN with native MQTT connection can operate as in the first case but now alternative communication can be used to create a hybrid system. An example of this is shown in figure 3 below.

Below identifies the key components:

- An AtoN that uses a standard Modicon Bus (MODBUS) interface and a MODBUS to Long Range Wide Area Network (LoRaWAN) adaptor. Alternatively, the AtoN can be native LoRaWAN.
- The transmission channel is LoRaWAN to LoRaWAN gateways.
- One or more LoRaWAN gateways are connected via TCP/IP to a LoRaWAN platform that manages the gateways and devices, and provides filtered data
- An MQTT bridge then translates the data for onward transmission.
- The data is then sent to the MQTT broker on the platform
- The platform provides various services such as access control, security, and application
- Users access the platform via an application.

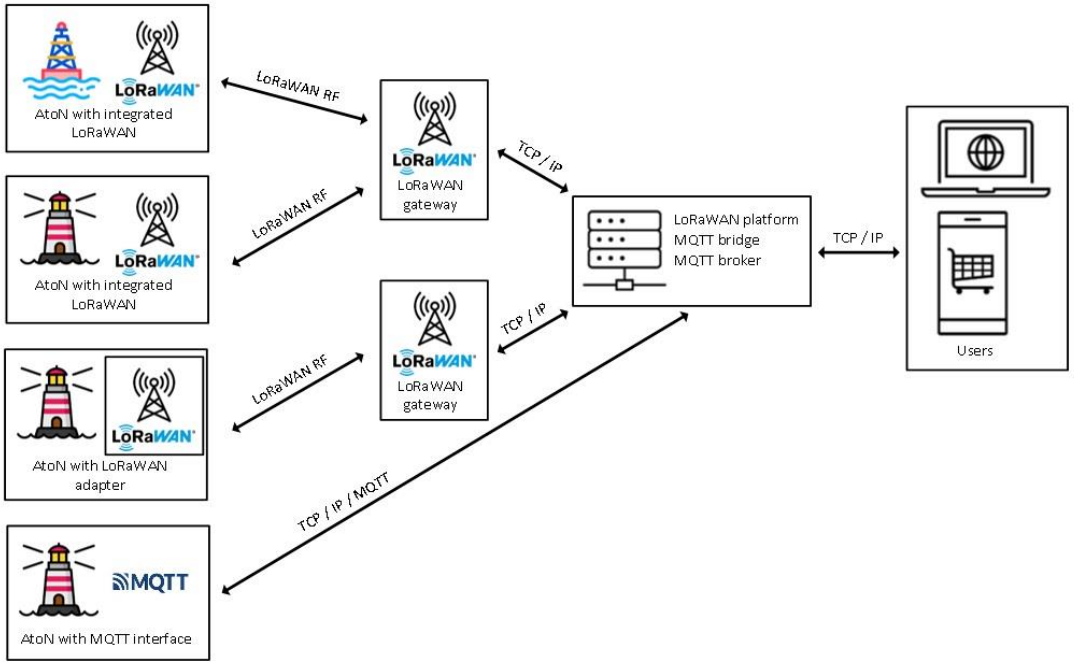

Figure 3 Topology of a hybrid platform

5. SECURITY

Cyber security needs to be considered at the design stage and guidance can be obtained from the Guideline 1182 on Cyber Security Specifics from an IALA perspective.

When considering the levels of security to be adopted, it needs to be reflective of the importance of the data. All security measures used require additional processing power, which can impact on type, cost and power of the solution, as well as the potential duration of the communication sessions, which again can impact on cost and power.

Some of the key area to consider regarding security at the design stage are:

- Firewalls & Selected Ports: These limit the access the server and mitigate risk of unauthorised communication.
- Virtual Private Network (VPN) tunnels: These provide end to end encryption.
- Data Encryption and Certificates: Used for encryption and authentication.
- Username & Password: These provide access control.
- Access Control Lists (ACL): These limit access to the broker.

5.1. ENCRYPTION AND AUTHENTICATION

MQTT is a protocol that runs over Transmission Control Protocol / Internet Protocol (TCP/IP and does not provide any security measures, but it can be implemented with various security controls, such as Transport Layer Security (TLS to ensure secure communication between the client and the broker.

When TLS is enabled or enforced, all data transmitted between the client and the broker is encrypted and authenticated. Such security measure must be implemented when the public internet communication channel is utilises, to avoid unencrypted data being transmitted.

The use of client-side certification as a security measure should be adopted to manage the connectivity of the publishers. This ensures that each publishing client possesses a dedicated certificate for authenticated access control. This also allows simple publishing client access management without the use of individual usernames and passwords.

For small systems with only a few publishing clients, the implementation of client-side certification would be onerous. For such situations MQTT supports the use of usernames and passwords for each client. Additional security measures can include ACL and Internet Protocol (IP) whitelisting

Alternative measures to the implementation of TLS encryption, is the application of a VPN tunnel for the communication section of a system that is within the public domain. The adoption of this approach ensures privacy, with smaller data loads and simplicity on the client side.

5.2. FIREWALLS

Firewalls are used as a protective mechanism to restrict access between different networks, ensure only the authorised data can pass. It is important that all unused ports are disabled, with only the MQTT port 8883 and 443 open fir this application. Below in Figure 4, is an example of an MQTT device connected with firewalls active.

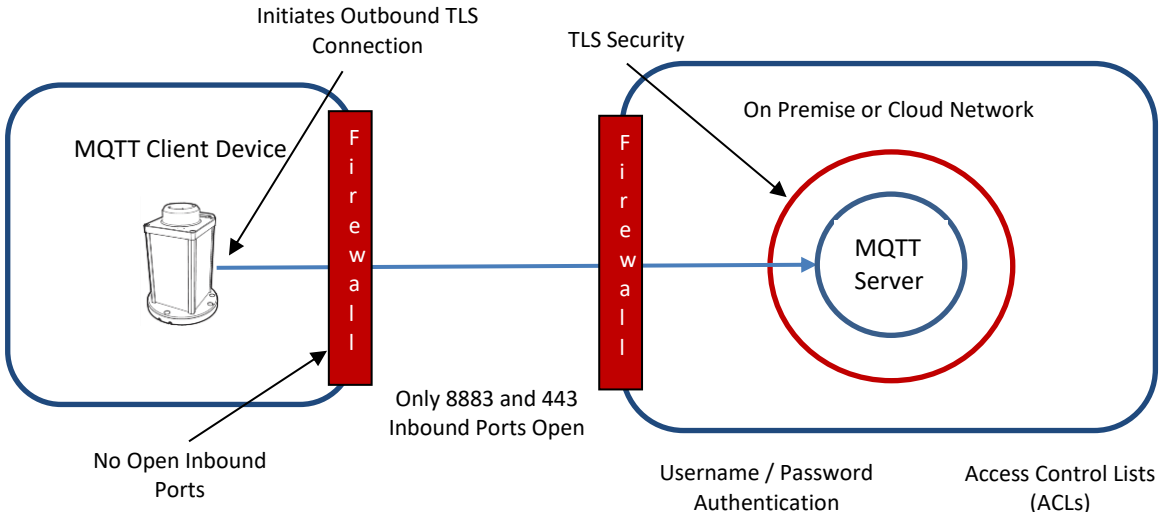

Figure 4 Example of MQTT Security

6. HARMONIZED PAYLOAD

To achieve the goal of a common protocol for visual AtoN, it is not enough to only standardize on using MQTT as the connectivity protocol, there is a need to define and harmonise the payload structure as well as the type of payloads. This section discusses how the payload is structured.

6.1. MESSAGE STRUCTURE

When considering the information to be captured, sent and stored, the structure of such messages can be organised by adopting any number of existing industrial approaches, but there is a conflicting trade off to be had. For example, a solution that adopts a byte-by-byte structure, keeping the size of the message data small, can't easily be read and understood. Alternatively, a much larger and easily to read message structure could be achieve by the use of Extensible Markup Language (XML). Again, these choices impact the amount of data sent, the power used, and communication costs incurred. Therefore, a balanced solution is needed, as both a low data size and readability are desirable.

To achieve this desired balance, a structure based upon JavaScript Object Notation (JSON) has been adopted. JSON is a lightweight data interchange format that is easy to read and write for humans, and relatively easy to parse and generate for machines. JSON is also a platform-independent data format, which means it can be used with any programming language or platform.

The following example shows a possible JSON representation describing a person.


```
{
  "first_name": "John",
  "last_name": "Smith",
  "is_alive": true,
  "age": 27,
  "address": {
    "street_address": "21 2nd Street",
    "city": "New York",
    "state": "NY",
    "postal_code": "10021-3100"
  },
  "phone_numbers": [
    {
      "type": "home",
      "number": "212 555-1234"
    },
    {
      "type": "office",
      "number": "646 555-4567"
    }
  ],
  "children": [
    "Catherine",
    "Thomas",
    "Trevor"
  ],
  "spouse": null
}
```


Figure 5 An example of JSON syntax, https://en.wikipedia.org/wiki/JSON

6.2. PAYLOAD

The payload can be considered as the dynamic information that is to be sent from the publisher as part of the message, structured in topics. From the example above (figure 5) using the JSON syntax, there are some areas of information, shown in green, such as children, which are known as “topics”. The payload for this topic a can be seen in red as names. A typical topic for a visual AtoN might be battery voltage or position.

These topics, which are populated by values or parameters have a defined data type based on the topic. So, for the children in the example above, the data type is a string. Given this, such topics and data types have been captured in appendix 1 as a minimum requirement associated with visual AtoN.

7. DEFINITIONS

The definitions of terms used in this Guideline can be found in the *International Dictionary of Marine Aids to Navigation* (IALA dictionary) at http://www.iala-aism.org/wiki/dictionary and were checked as correct at the time of going to print. Where conflict arises, the IALA Dictionary should be considered as the authoritative source of definitions used in IALA documents.

8. ABBREVIATIONS

ACL	Access Control List
Ah	Ampere-hours
AIS	Automatic Identification System
AMQP	Advanced Message Queuing Protocol
OPC UA	Open Platform Communications Unified Architecture
APN	Access Point Name
CBOR	Concise Binary Object Representation
CN	Common Name
CoAP	Constrained Application Protocol
CRC	Cyclic Redundancy Check
CSR	Certificate Signing Request
DTLS	Datagram Transport Layer Security
GNSS	Global Navigation Satellite System
HDOP	Horizontal Dilution of Precision
HTTP	HyperText Transfer Protocol
HTTPS	HyperText Transfer Protocol Secure
ID	Identifier
IoT	Internet of Things
IP	Internet Protocol
IT	Information Technology
JSON	JavaScript Object Notation
LoRaWAN	Long Range Wide Area Network
LwM2M	Light weight Machine to Machine
LWT	Last Will and Testament
M2M	Machine-to-Machine
Modbus	Modicon Bus
MQTT	Message Queuing Telemetry Transport
OASIS	Organization for the Advancement of Structured Information Standards
PEM	Privacy-Enhanced Mail
QoS	Quality of Service
RAM	Random Access Memory
RESTful	Representational State Transfer-ful
RTC	Real-Time Clock (hardware timekeeping)
SASL	Simple Authentication and Security Layer
SI	International System of Units
SMS	Short Message Service (text messaging)
SSL	Secure Sockets Layer (predecessor to TLS)
TCP/IP	Transmission Control Protocol/Internet Protocol
TLS	Transport Layer Security
TLV	Type-Length-Value
UDP	User Datagram Protocol

UTC	Coordinated Universal Time
UTF-8	Unicode Transformation Format - 8-bit
VPN	Virtual Private Network
WGS	World Geodetic System
XML	Extensible Markup Language

ANNEX A VISUAL ATON MQTT PROTOCOL

1. LOGICAL ARCHITECTURE OF MQTT ENABLED DEVICE

To support different devices, two different architectures are employed. In the simplest scenario, a lantern is equipped with a communication device capable of MQTT, such as a modem with an MQTT stack. From both the user's and server's perspectives, the lantern appears as a single device, allowing the user to control the lantern as one device (figure 1). For example, in a simple architecture case, telemetry messages can have the following topic name “tele/device123”, where the first part signifies a telemetry message, and the second part is the device unique ID.

Figure 1 Simple architecture

Alternatively, some systems have well-separated internal devices, such as multiple lanterns and communication modules connected via an internal network. From both the user's and server's perspectives, the lanterns and all sub-devices are seen as separate devices, allowing the user to control each device independently (figure 2). For example, in a complex architecture case, telemetry messages can have the following topic name: "tele/device123/lantern1", where the first part is the message type (in this case, a telemetry message), the second part is the unique device ID, and the third part is the sub-device name.

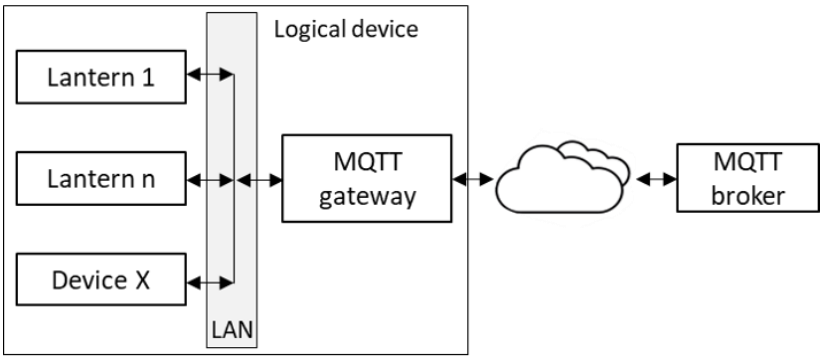

Figure 2 Complex architecture

In the case of a simple architecture, topic names contain only master device information, while in a complex device setup, topic names with sub-device designations are used to control each sub-device. For compatibility reasons, complex architectures should support a logical device, which groups all the internal devices. Depending on the implementation, an MQTT gateway device can be an aggregated gateway, translating data between the broker and the devices, while users see only a single logical device.

2. SERVER-SIDE IMPLEMENTATIONS

The protocol described in this document accommodates the use of several different server-side implementations. It is possible to use a very simple server implementation without a database, while at the same time, it also allows for the use of a full-scale enterprise system.

2.1. MINIMAL IMPLEMENTATION

In smaller systems with only a few devices, it's possible to use only an end-user control application and a broker. In this scenario, no database is necessary; the end-user program connects directly to the broker, enabling control and monitoring of each device.

Figure 3 Minimal system

In this setup, both the device and the end-user should use message retention as much as possible. This enables the storage of the latest data on the broker, allowing the end-user to retrieve the most recent valid data with each new connection. Since there's no database for user management, the broker can only control access through an access list or private certificates. Without any user access control, the end-user retains full control over all devices.

This implementation can be quite cost-effective, as only a broker is needed, and the end-user application can be relatively simple.

2.2. FULL IMPLEMENTATION

In larger installations, it is recommended to use an additional service, that manages user roles and utilizes a database to store telemetry data and possibly providing some interface for users. Users do not need specific programs to connect to the service, as it can use HyperText Transfer Protocol (HTTP) for interactions.

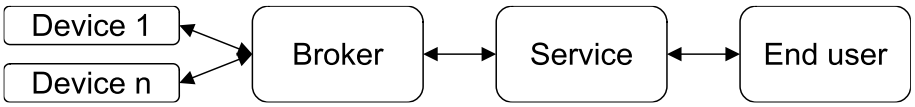

Figure 4 Typical system

In this setup, an additional component is used between the end-user and the broker. This component, called a service, contains a database and other supplementary programs for device control and management. Unlike minimal implementations, this implementation does not require the use of message retention, as the database is always available and stores all required data from the devices. The access control system allows restricting the end user's ability to see or control the devices.

The rest of this document uses this full implementation as an example. In the context of the MQTT protocol, the term 'broker' refers to the receiving part. However, when referring to this protocol, the term 'server' is used instead.

3. MINIMUM COMPATIBILITY AND FUNCTIONAL REQUIREMENTS

Devices that connect directly to the MQTT infrastructure should use MQTT version 3.1.1 (or later) connection, following the payload and topic notation specified in this document.

The device should also be capable of supporting the non-persistent mode, along with adopting the following topics and properties:

1 Telemetry message: ‘tele’
   a ‘sessionId’
   b ‘status’
   c ‘uptime’
2 Device information message: ‘info’
   a ‘sessionId’
   b ‘metadata’
   c ‘protocolVersion’
   d ‘sysInfo’
3 Command message: ‘cmd’
   a ‘sessionId’
   b ‘send’
4 Parameter message: ‘parameter’
   a ‘sessionId’
   b ‘brokerAddress’
   c ‘mqttClientId’
   d ‘mqttUser’
   e ‘mqttPassword’
   f ‘uid’

4. COMMUNICATION MODES

An MQTT device should support at least one of the communication modes: persistent session mode or non-persistent mode.

In persistent mode, the device sends a greeting message at the beginning of the session, a disconnect message at the end of the session, and sets up a last will and testament message before starting the session. This mode is beneficial when data packets are sent frequently, at least one packet within a 5-minute period. It can be used when the telemetry data sending process is triggered by an event. However, it is not recommended to use persistent mode when telemetry data is sent periodically, with a period greater than an hour.

In non-persistent mode, the device does not use greeting nor will and testament message, and the disconnect message is optional. Non-persistent mode is suitable for cases where data is not sent very often, for example, once per hour or at larger intervals.

All communication sessions are initiated and terminated by the device, session termination from broker is not allowed. Sessions can be requested to be terminated by sending a 'done' session command from the server. Note that the 'done' command is only a request, and the device is free to choose the time when the session is closed. In non-persistent mode, the device always waits for additional data or a command from the server, the session is interrupted when the server does not send any data during the receive wait period. It is recommended that the receive wait period is at least 30 seconds. This value is device-specific and generally should be less than 300 seconds.

If the message type requires a response, a new message must not be sent before either receiving the response to the last message or until the receive wait period expires. Every device should subscribe to at least the following topics:

- Commands topic (’cmd’)
- Parameters topic (’parameter’)

A device that is required to receive legacy commands specific to it must subscribe to the topic named 'proprietary'.

Optional topics that are used:

- Session topic (’session’): This is used when a device utilizes session control capability.
- Proprietary topic (’proprietary’): This is used when a device expects to receive device-specific legacy commands.

4.1. NON-PERSISTENT SESSION MODE

4.1.1. TELEMETRY INFORMATION

It is possible to transfer telemetry information without additional data.

1 A Device connects to a broker and sends a telemetry packet. Depending on the configuration this may be repeated during the session.

2 Optionally, the device sends a ’bye’ message and disconnects from the broker. The server should accept the connection closing without receiving a 'bye' message. The 'bye' message is solely for indicating to the server that the device has gone offline and so no additional data will be received.

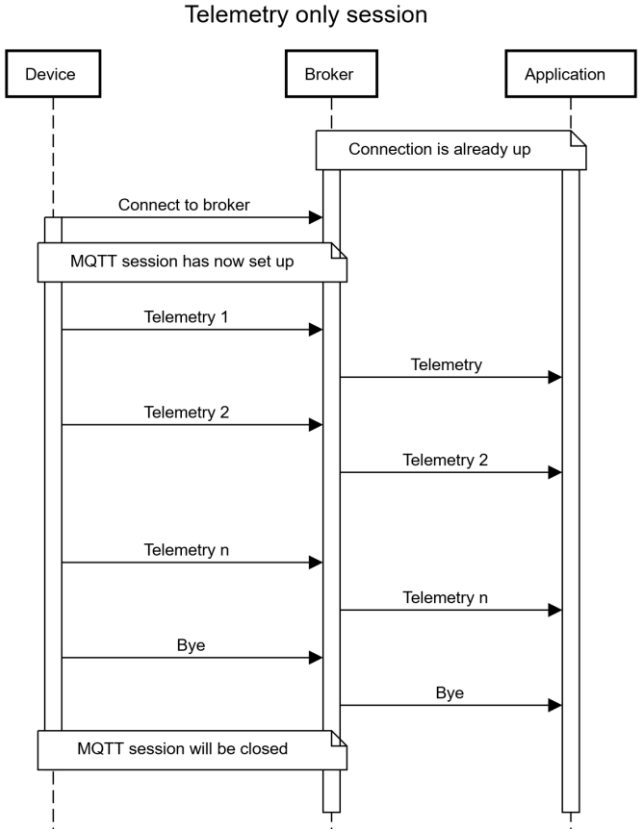

Figure 5 Telemetry session

4.1.2. INFO

1 The device connects to the broker and sends a telemetry packet.

2 The server asks ‘info’.

3 The device sends data by using the info topic.

4 The server sends done. This means that the server has no more commands queued.

5 Optionally, the device sends a ’bye’ message and disconnects from the broker. The server should accept the connection closing without receiving a 'bye' message. The 'bye' message is solely for indicating to the server that the device has gone offline and so no additional data will be received.

6 Device disconnects from the broker.

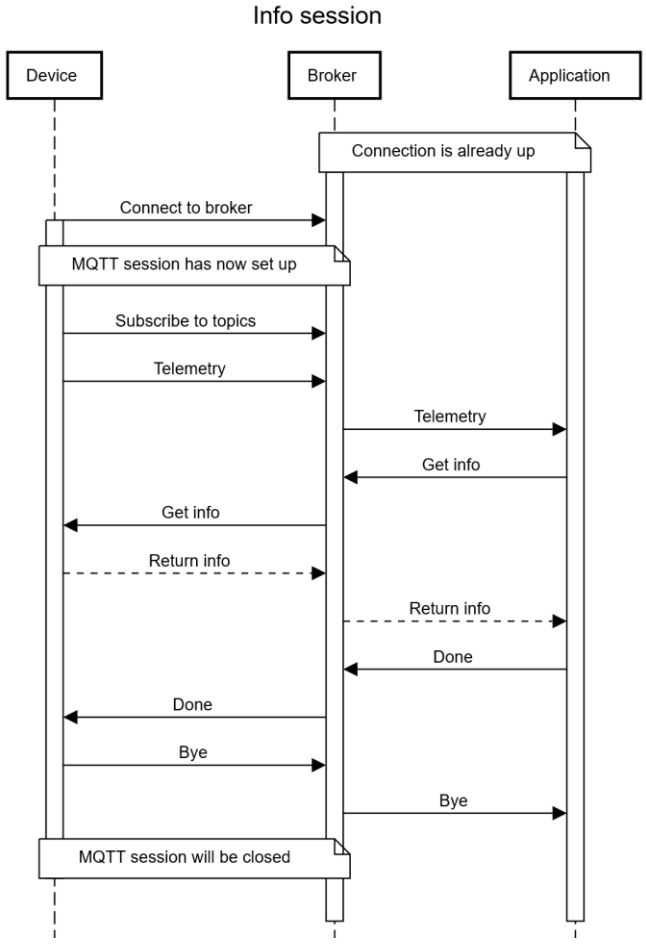

Figure 6 Info session

4.1.3. GET PARAMETER

It is implementation-defined as to whether the device automatically sends configuration parameters after a change or only upon request. Automatically sent parameters are useful, if a local user can change the device's parameters, as the device will send the updated parameters to the server.

A typical sequence for retrieving the parameters is as follows:

1 The device connects to the broker and send a telemetry packet.

2 The server asks 'parameter'.

3 The device sends the parameters.

4 The server sends 'done'.

5 The device sends an optional 'bye'.

6 The device disconnects from the broker.

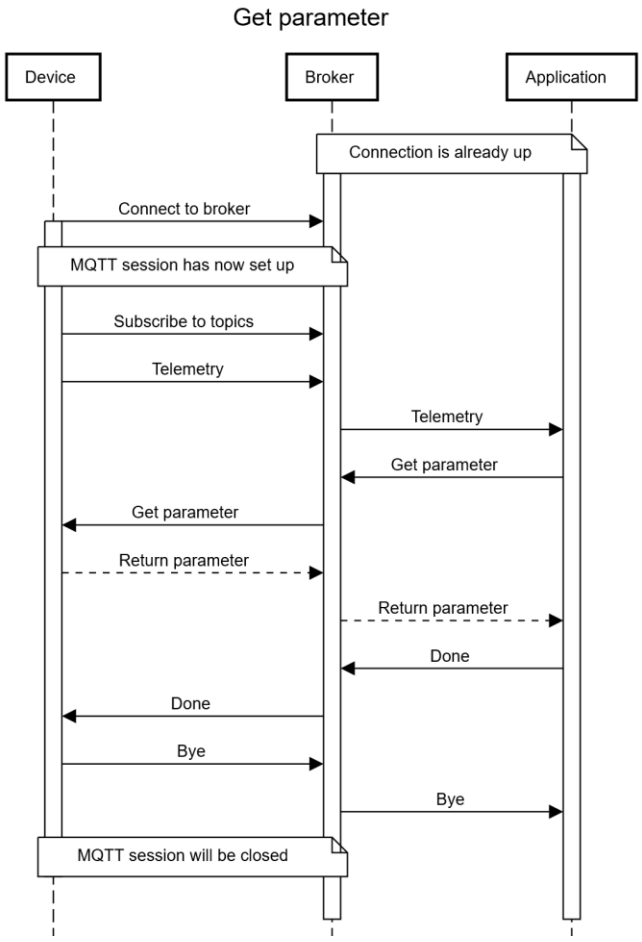

Figure 7 Get parameter

4.1.4. SET PARAMETER

Set parameter is used to change the device's configuration.

1 The device connects to the broker and sends a telemetry packet.

2 The server asks 'parameter'.

3 The device sends the current parameters.

4 The server sends the updated parameters.

5 The server sends 'done'.

6 The device sends an optional 'bye'.

7 The device disconnects from the broker.

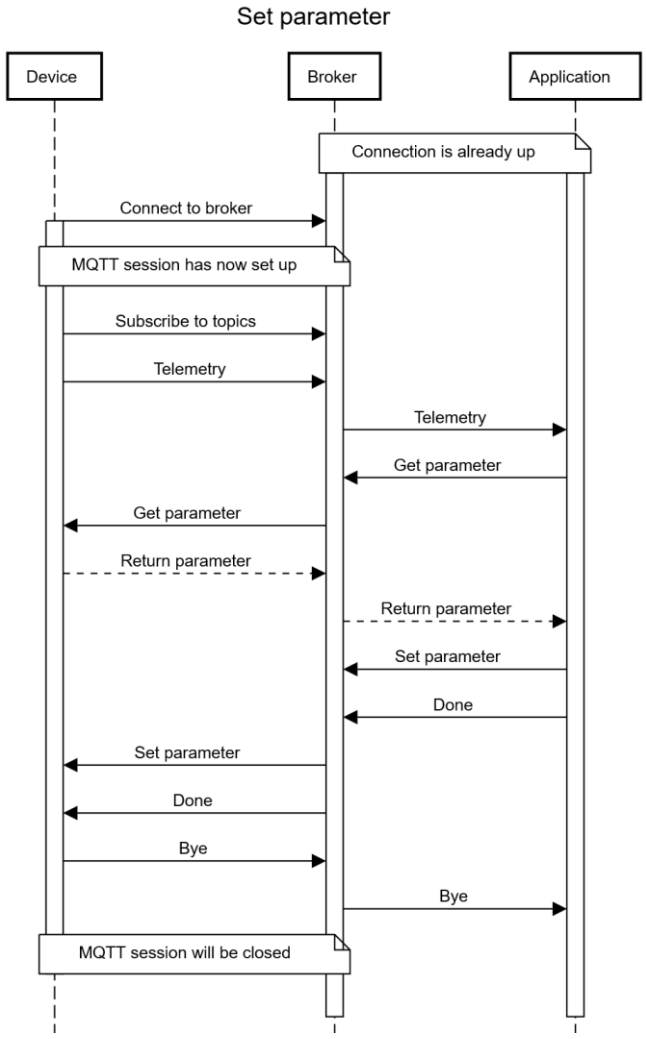

Figure 8 Set parameter

4.1.5. PROPRIETARY COMMAND

1 This command is used to communicate with the device by using non-standard commands or accessing non-standard features. The device connects to the broker and sends a telemetry packet.

2 The server sends a ’proprietary’ command.

3 The device sends a response to the proprietary command. Depending on the command, this may be an acknowledgment or a data response.

4 The server sends ’done’.

5 The device sends an optional ’bye’.

6 The device disconnects from the broker.

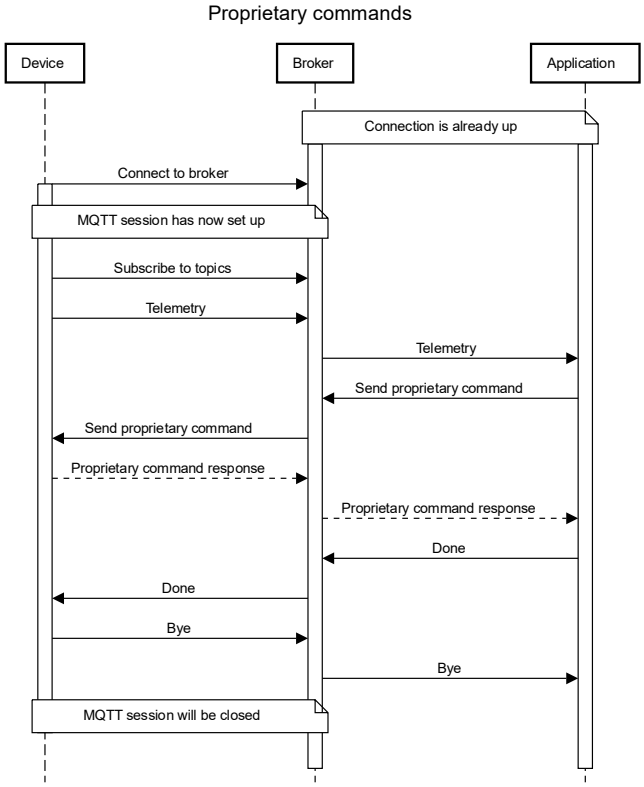

Figure 9 Proprietary session

4.2. PERSISTENT SESSION MODE

Persistent sessions differ from non-persistent ones only in terms of the beginning and end messages. Every persistent session must start with a 'hello' message and end with a 'bye' message, or in the case of link failure, the broker sends an 'offline' message. This mode can be interrupted by sending a 'done' command.

4.2.1. SESSION WITH NORMAL ENDING

1 The device connects to the broker and sends a ’hello’ packet.

2 The device sends telemetry packets. This is identical to the non-persistent mode.

3 The device sends the required ’bye’.

4 The device disconnects from the broker.

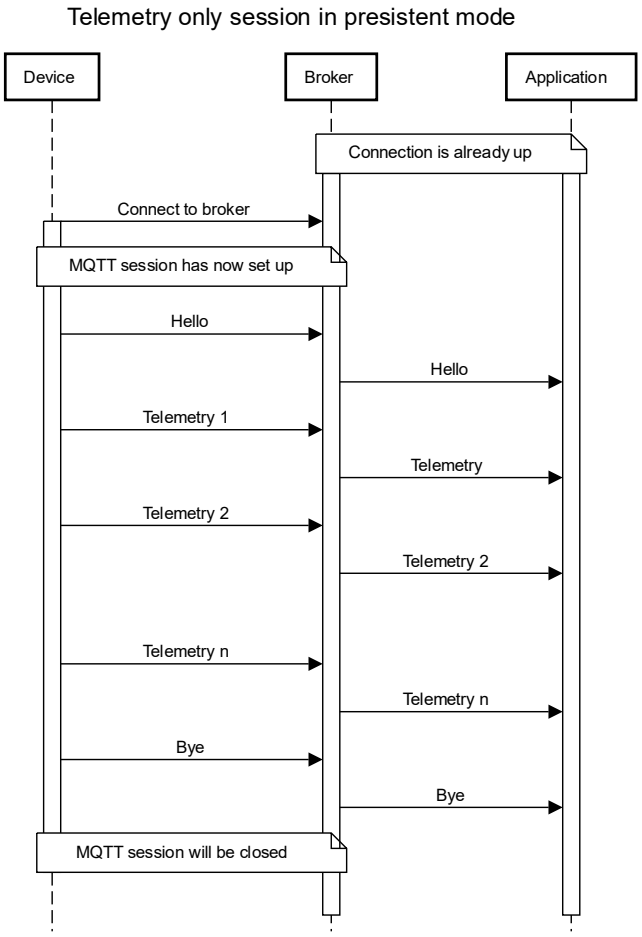

Figure 10 Persistent session

4.2.2. FAILED SESSION

1 The device connects to the broker and sends a 'hello' packet.

2 The device when operating normally sends packets until a link failure occurs. This is identical to the non-persistent mode.

3 On link failure, the broker detects a timeout and sends a last will packet with 'offline' message.

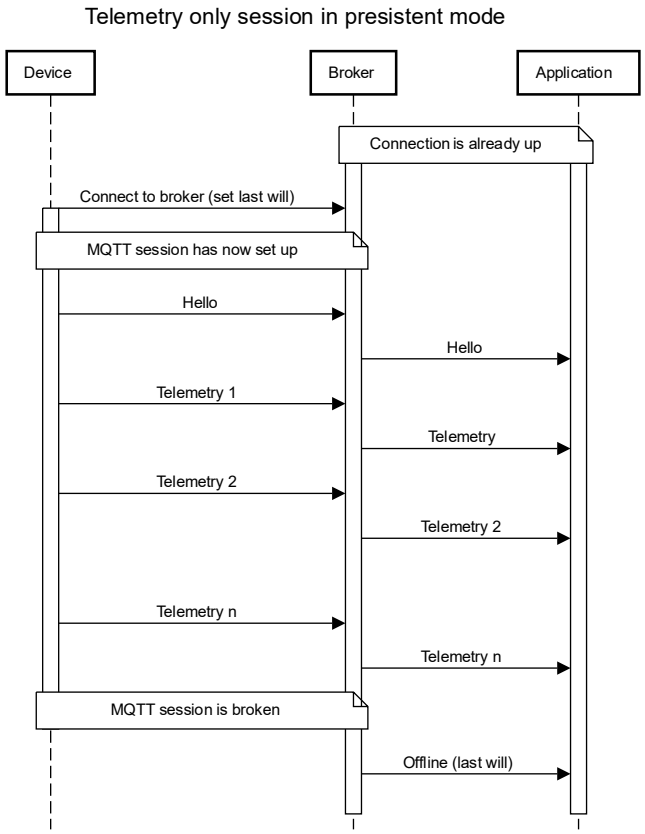

Figure 11 Failed persistent session

4.3. DEVICE REGISTRATION, DEREGISTRATION AND CERTIFICATE RENEWAL

- Certificate renewal and device registration are optional processes. Depending on the capabilities of the device and the broker, there are several options for registering a device: Registering without a certificate: the device registers without a certificate and does not use TLS/SSL capabilities. This is the recommended solution for private networks.
- Registering with a Certificate Signing Request (CSR): the device uses a CSR for registration and the server then generates a new valid certificate based on the CSR.
- Registering via an external device or application: an external device or service application is connected to the device, and the external application handles the registration process. After registration, a new certificate is loaded from the external application onto the device.
- Transfer generated keys and certificates using a device-specific transport method. The device then uses the received credentials to connect to the server and register with the service.

If the device utilizes a TLS/SSL connection, it must support certificate renewal and be capable of using a third-party certificate. All registration commands and properties can be sent from the server after starting the registration process with the 'register' command.

4.3.1. REGISTRATION WITHOUT CERTIFICATE

1 The device sets up a response topic for communication.

2 The device connects to the server and sends the response topic, MQTT device ID, and optionally a registration key. The registration key is previously sent via a separate communication channel, such as SMS.

3 The server sends a ‘registered’ response message to confirm the initial registration.

4 The device provides its information as outlined in section 7.5 – Info.

5 The server replies with a ‘completed’ response message. The device then marks its configuration to indicate that the registration is finalized.

6 The device can now disconnect and reconnect to the server as needed. The registration is now complete.

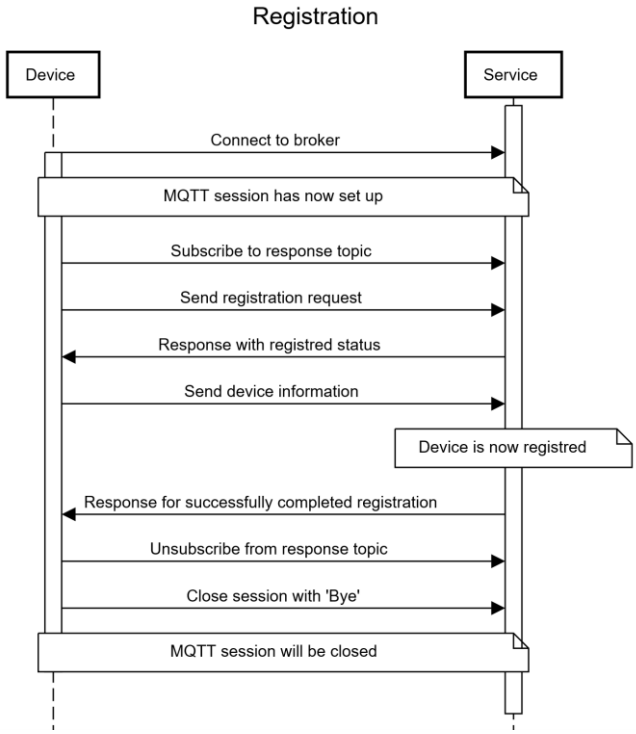

Figure 12 Device registration without certificate

4.3.2. REGISTRATION WITH CERTIFICATE SIGNING REQUEST (CSR)

1. The device sets up a response topic for communication.
2. The device connects to the service, sends response topic, MQTT device ID, CSR and optionally provides a registration key. The registration key is sent previously by a separate communication channel, for example SMS. The CSR and private key should be generated by the device.
3. The service sends a 'registered' response message to confirm the initial registration.
4. The device provides its information as outlined in section 7.5 – **Erreur ! Source du renvoi introuvable.**.
5. The service replies with a 'completed' response message. The device then marks its configuration that registration is finalized.
6. The device can now disconnect and reconnect to the service as needed. Registration is complete.

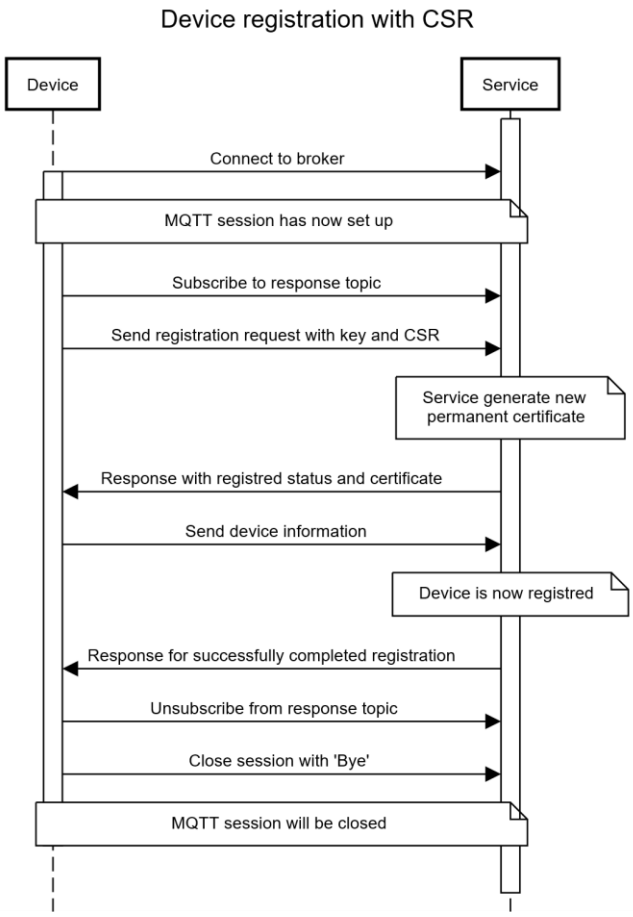

Figure 13 Device registration with CSR

4.3.3. DEREGISTRATION, INITIATED BY THE DEVICE

1. The device sets up a response topic for communication.
2. The device connects to the service, sends the response topic, and optionally provides the registration key. The registration key is previously sent via a separate communication channel, such as Short Message Service (SMS).
3. The service sends a ‘completed’ response message. The device then marks its configuration to indicate that deregistration is finalized.
4. The device can safely disconnect from the service.

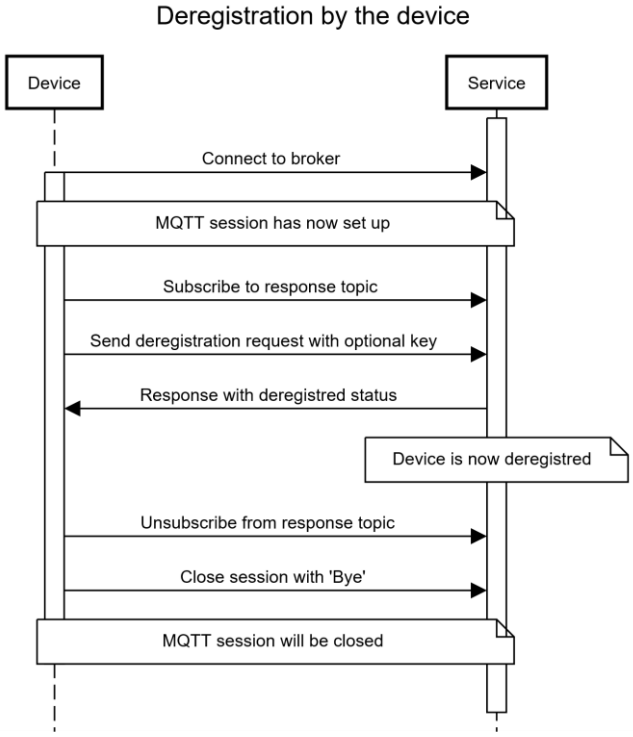

Figure 14 Device deregistration by the device

4.3.4. DEREGISTRATION, INITIATED BY THE SERVICE

1. The service sends the ‘register’ command. This command starts the registration-deregistration process.
2. The service sends the ‘deregister’ command.
3. The device sets up a response topic for communication.
4. The device connects to the service and sends the response topic.
5. The service sends a ‘completed’ response message. The device then marks its configuration to indicate that the deregistration is finalized.
6. The device can safely disconnect from the service.
7. The device connects immediately back to the server that was sent the deregistration command. It is up to the server if to re-register the device or ignore the device. After this connection attempt the device should continue communication attempts on its regular schedule.
8. The device disconnects from the service.

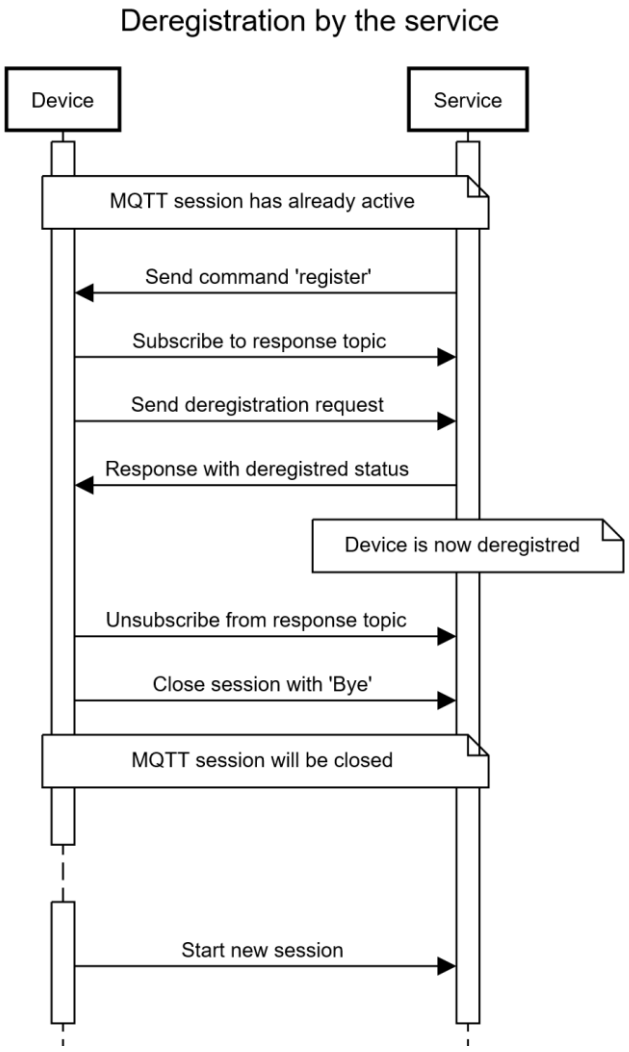

Figure 15 Device deregistration by the service

4.3.5. CERTIFICATE RENEWAL

Certificate renewal is similar to registering with a CSR, except that the device must deregister before initiating the next registration process.

5. TOPIC NAMES

The Topic name must contain only letters (‘a’ – ’z’ and ‘A’ – ’Z’), numbers (0-9) and slashes (’/’). Each topic name consists of two required parts: a first-level topic name indicating the purpose and the payload type of the topic, a second-level topic name as a unique ID, and an optional third part for responses or automatic messages. The unique ID is user defined and can be the device’s serial number or the site name. The preferred unique ID is the site name.

Some topic names are reserved:

- All first level topic names.
- All last level topic names.
- All topic names for certificate creation and provisioning.

5.1. GENERAL STRUCTURE OF TOPIC NAMES

All topic names for messages that are sent from the device to the server, or from the server to the device, shall have the first level topic name and the device unique identifier (see 1 below). To address a local device, it is possible to add a device name after the unique identifier (see 2 below). The response messages have suffix ’res’ (see 3, 4 below). It is possible to use a random identifier **(**ID) in the response topics (see 5 below). The topic name may have suffix ’auto’ (see 6 below). This is used when the telemetry data is something other than a regular telemetry packet (see 7.4 – Topic ’tele’ – status information), for example a proprietary packet with telemetry data (see 7.8 – Topic ’proprietary’).

1 <first level topic name>/<uid> – topic name for messages

Example: ’tele/lighthouse’ – telemetry topic from lighthouse

2 <first level topic name>/<uid>/<dev> – topic name for sub device messages

Example: ’cmd/lighthouse/lantern’ – control topic for lighthouse device lantern

3 <first level topic name>/<uid>/res – topic name for response message

Example: ’parameter/lighthouse/res’ – response for parameter message from lighthouse

4 <first level topic name>/<uid>/<dev>/res – topic name for sub device response message

Example: ‘parameter/lighthouse/lantern1/res’ – response for the parameter message from lighthouse device 1

5 <random ID> – response topic with random ID

Example: ’abcdxsd32/res’ – response for parameter message from lighthouse, using random ID

6 <first level topic name>/<uid>/auto – automatic topic, used for automatic telemetry data

Example: ’proprietary/lighthouse/auto’ – automatic proprietary message

5.2. FIRST LEVEL TOPIC NAMES

The first level topic name describes the topic's purpose, e.g. telematics topic, information topic, etc.

All first level topic names are:

- 'session' – session information.
- 'tele' – telemetry information. Device issued automatic monitoring data.
- 'info' – device information. Device ID, capabilities, etc.
- 'cmd' – commands. Actions that can be triggered by a user.
- 'parameter' – get or set parameter(s).
- 'proprietary' – device specific data that can be used for implementing custom protocols inside the device, like Modbus.

To allow for future development the first level topic names have a reserved upper-case suffix:

- 'B' – binary payload, reserved for future use. A binary payload can be used when a more compact data encoding other than JSON is needed. The only requirement is that the binary payload must be convertible without loss between its binary representation and JSON. The binary format can be Concise Binary Object Representation (CBOR).

Example: 'teleB/lighthouse/device1' – telematics message with a binary payload

- 'S' – encrypted payload, reserved for future use. An encrypted payload can be used when full end-to-end encryption is required. In the current implementation, when using a TLS/SSL encrypted communication channel, attackers can read or modify data in the broker or before it reaches the end user. End-to-end encryption ensures that data is encrypted from the device to the end user, preventing any possibility of modification by an attacker. This feature can be used for firmware updates.

Example: 'teleS/lighthouse/device1' – telematics message with encrypted payload

- no suffix – JSON payload.

Example: 'tele/lighthouse/device1' – telematics message with JSON

5.3. SECOND TO N-1 LEVEL TOPIC NAMES

Second level topic names contain a unique ID, also known as 'thing' name. The number of levels the unique ID has is dependent on the actual device's configuration. For example, a second level topic name can be one level *'lighthouseA'* or multilevel *'country/region/subregion/lighthouse'*.

5.3.1. BEFORE LAST LEVEL TOPIC NAME, N-1 LEVEL TOPIC NAMES

Topic names after the second level topic name are optional and shall contain the device name or identifier. This level is used only when a master device has sub-devices, which are not able to connect directly to the broker itself. This level can be considered as part of second level topic name.

5.4. LAST LEVEL TOPIC NAME

Last level topic names are used for responses and automatic messages. All possible last level topic names are:

- 'auto' – topics that are sent automatically, tele and proprietary topics, may have an additional level named 'auto'. This level is used only when an automatic message is published for a topic of the same name as the subscribed topic name (see example 4 in 5.1 General structure of topic names).

- ‘res’ – used for a response message to a query. When topics include a response, an additional level named ‘res’ is added to the end. Since the response topic is issued by the server, any unique identifier can be used before the ‘res’ part. However, it is recommended to use a response topic name that includes both the original topic name and the site identifier (see example 5 in 5.1 General structure of topic names).

6. QOS VALUES AND RETAINED MESSAGES

MQTT QoS (quality of service) defines a level of delivery performance for all messages. Below are the various levels available.

- QoS value 0: There is no guarantee that a message is received.
- QoS value 1: An acknowledgment is sent to confirm that the message has been delivered.
- QoS value 2: Ensures that both the message has been received and its acknowledgment are successfully delivered back to the sender.

With QoS level 1 and 2, a message is retransmitted if an acknowledgment is not received and this is not suitable in cases where real-time data is transmitted without any additional measures to exclude delayed packets.

Retained Messages is an MQTT feature that allows the storing of the “last known good” message., The typical case for using retained messages are on small systems that have only a few devices, such as a broker and a basic monitoring software without a database. It is also possible to store the last known status to a broker, so that the monitoring application can load the last state from the broker. For this case, it is necessary that the topics, ‘session’ and ‘tele’, have retention enabled. Parameter retention is used when a device is offline for most of the time and the monitoring server is unable to send new parameters during the communication session.

Recommended minimum QoS levels and retained values are:

Table 2 Recommended retained values for a minimum QoS level

Topic	Minimum QoS	Retained messages allowed
session	1	Yes
tele	0	Yes
info	1	Yes
cmd	0	No
parameter	1	Yes
proprietary	0	No
registration	1	No

7. MESSAGES

The message payload uses a JSON format. JSON data should adhere to the format described by the JSON standard [1]. All messages are presented in a single-level JSON structure. Values can be represented by strings, integers, arrays, JSON objects. It is permitted to use null values. Arrays and JSON objects may contain either a string or an integer value.

```
{
  "<key1>": "<value string>",
  "<key2>": <integer>,
  "<key3>": ["<value1>", "<value2>"]
  "<key4>": {"<subkey1>":"<subvalue1>", "<subkey2>": "<subvalue2>"}
  "<key5>": null
}
```

Figure 16 An example showing JSON message format.

7.1. RULES FOR ALL MESSAGES

1 The payload is coded in the JSON format [1][2]

2 Only Unicode Transformation Format - 8-bit (UTF-8) characters are allowed in property names.

3 The maximum property name length is 32 characters.

4 The first character must be a lower-case letter (’a’ – ’z’). Subsequent characters can be a letter (’a’ – ’z’ or ’A’ – ’Z’), or a digit. An upper-case (’A’ – ’Z’) letter is used to separate words in property names.

5 Property values are UTF-8 encoded JSON strings.

6 A ’.’ (dot) is used as a decimal separator in floating-point numbers.

7 The International System of Units (SI) and derived units shall be used. Some examples are shown below.

- a Time units are in seconds (s).
- b Distance or length is in meters (m).
- c Temperature in Kelvin (K).
- d Voltage in Volts (V).
- e Current in Amperes (A).

8 All timestamps must be in Coordinated Universal Time (UTC+00:00).

9 Position coordinates should use the World Geodetic System 84 (WGS 84) geographic coordinate system, using decimal degrees.

10 The property name “class” is not allowed.

11 User defined properties are allowed, but must follow the message payload data rules.

12 All optional properties can be omitted or set to a null value if the data is invalid. Non-optional properties should have a default value, as specified in this document, in case of invalid data.

13 All JSON messages from a device should contain session ID information.

7.2. GENERAL MESSAGE

A General message is a supertype for all messages and all messages must implement the fields in the general message.

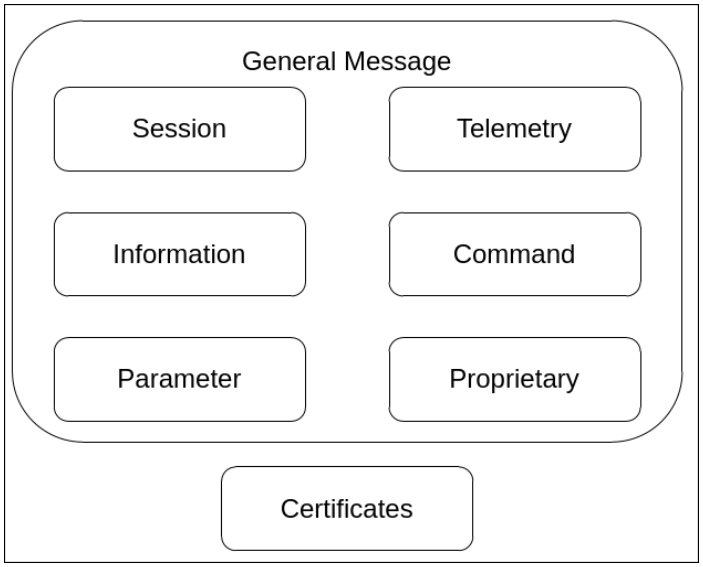

Figure 17 General message

This message supertype has the following two mandatory fields: ‘sessionId’ and ‘time’, and one optional field, 'resTopic'.

7.2.1. PROPERTY SESSION ID

Property name:	’sessionId’
Description:	Session ID is a string that holds a value unique to the MQTT session. This value can be a monotonic counter that starts with each MQTT session or a timer.
Required:	Only for messages that are sent from a device to the server
Example:	"sessionId": "session-820923084792"

7.2.2. TIME

Property name:	’time’
Description:	An unsigned integer. The server should expect to receive at least a 32-bit unsigned value. Possible values are:’ • Seconds from January 1st, 1970, at UTC (UNIX time). Default value is 0. • null – used only for ‘command’ message to query time
Required:	Optional, only if a real-time clock (RTC) timer is available.
Example:	"time": 1673564596

7.2.3. PROPERTY RESPONSE TOPIC

Property name:	’resTopic’
Description:	The value of the ’resTopic’ is a string, which is used as a topic for a response to query. This value can be generated dynamically from the server or can be a fixed topic name with ‘res’ suffix.
Required:	The server adds this property to messages that require a response from a device.
Example:	Dynamic response topic name: "resTopic": "62694cbf-9a7a-4af4-abf6-11dee1253001" Fixed response topic name: "resTopic": "parameter/location1/location1/site/device1/res"

7.2.4. EXAMPLE

```
{
  "sessionId": "session-1"
  "resTopic": "abcd123/res"
}
```

7.3. TOPIC 'SESSION' – SESSION INFORMATION

The session topic is intended as a notification of the session start and end. This topic is used on systems which use long-lived sessions. This topic is also used in 'last will' messages.

A device sends the following message at the beginning of a persistent session:

```
{
 "sessionId": "session-1",
 "state": "hello"
}
```

A device sends the following message at the end of a persistent session:

```
{
 "sessionId": "session-1",
 "state": "bye",
 "time": 1473564596
}
```

7.3.1. DONE

Property name:	'done'
Description:	Property value can be any arbitrary string or null; the server should not expect for any specific property value. This is a notification from the server that the server has completed all pending tasks, and that the device is free to close the MQTT session and disconnect from the broker. It depends on the device configuration if it disconnects immediately or sends further telematic packets. If the device sends further packets after the 'done' message, then the device discards the current active 'done' notification. If the server fails to send a 'done' message, the device may disconnect in non-persistent mode once the message receiving timeout has elapsed (see 4 – Communication modes).
Required:	Optional
Example:	"done": "ok"

7.3.2. STATE

Property name:	'state'
Description:	A string that describes the session's state. • 'hello' – this value is sent at the beginning of a persistent session. • 'bye' – this value is sent at the end of a persistent session. If the device supports real time information, then it is recommended that the 'bye' message includes the 'time' property. • 'offline' – this value is sent by the last will & testament message. An alternative is to use an empty JSON payload.
Required:	Only on persistent sessions
Example:	"state": "hello"

7.4. TOPIC 'TELE' – STATUS INFORMATION

7.4.1. TIME INFO

7.4.1.1. Uptime

Property name:	'uptime'
Description:	This is an unsigned integer. The server should expect to receive at least a 32-bit unsigned value, representing seconds from last boot. Default value is 0.
Required:	Yes
Example:	"uptime": 211

7.4.2. STATUS INFO

7.4.2.1. Status

Property name:	'status'
Description:	The property of this string represents the device's operational status. The allowed values are: 'init', 'ready', 'alert', 'suspend' and 'test'. • 'init' – this status is set until all required components within a device are fully operational, e.g. Global Navigation Satellite System (GNSS) has not acquired initial coordinates after booting. It is recommended that the 'init' state duration is limited by a timeout. If 'init' is not completed within the timeout period, then the device enters an 'alert' state. 'init' state is the default value after booting. • 'ready' – the system is fully functional and does not have any active errors, e.g. lantern is switched on • 'alert' – when an alarm condition is detected, e.g. low voltage • 'suspend' – when the system is switched off, but is still functional, e.g. storage state. If the device does not have this state, then this field is not required. • 'test' – when the lantern is in test mode.
Required:	Yes
Example:	"status": "ready"

7.4.2.2. Alert status

Property name:	'alert'
Description:	This string contains a list of active alerts. Include this property in each session, if alerts are present. When all alerts have cleared it is recommended that this property with the value 'none' is sent in the next session. Any user defined values are allowed, but the following status names are reserved: • 'none' – default value • 'lowVoltage' – for low battery alarm. Low battery disconnect level is configured by 7.7.10 – Low battery voltage level. • 'lowVoltageWarning' – for low battery warning. Battery warning level is configured by 7.7.8 – Minimum battery voltage warning level. • 'highVoltageWarning' – for a high battery warning. Battery warning level is configured by 7.7.9 – Maximum battery voltage warning level • 'gnssError' – any GNSS related errors • 'offLocation' – if distance from the appointed position is more than configured • 'lampFail' – if lantern does not operate as expected • 'motorFail' – if the rotation of a rotating beacon has failed • 'impactDetected' – for collision detection • 'lanternPowerWarning' – for consuming too much power • 'maintenance' – for when a maintenance visit is required • 'cfgChanged' – configuration has been externally changed, and not yet synchronized with server
Required:	Optional, only required when device status is 'alert'
Example:	Two alerts active: "alert": ["lowVoltage", "offLocation"]

7.4.2.3. Lantern status

Property name:	'lanternStatus'
Description:	An array of status strings. Shows the lantern's status. The allowed main status values are: • 'onMainCharacter' – main or night character • 'onAlternativeCharacter' – alternative or day character • 'off' – not flashing, this is the default value The optional secondary status values are (see 7.7.7 – Lantern trigger): • 'byLightSensor' – lantern is controlled by the light sensor, default • value'byCalculation' – lantern is controlled by a calculated situation. • 'byConfiguration' – lantern is controlled by configuration
Required:	Mandatory only on lanterns. 'type' field in info message should be 'lantern'.
Example:	"lanternStatus": ["onMainCharacter", "byLightSensor"]

7.4.2.4. Light sensor status

Property name:	'lightSensorStatus'
Description:	A status string, that shows the lantern light sensor status. Allowed values are: • 'day' – day mode • 'night' – night mode
Required:	Optional. The 'type' field in info message should be 'lantern'.
Example:	"lightSensorStatus": "day"

7.4.3. ENVIRONMENTAL DATA

7.4.3.1. Device temperature

Property name:	'temperature'
Description:	Temperature in Kelvin. This is a JSON object, which contains unordered name/value pairs, where the values are floating-point values. This property contains the following sub properties: • 'last' – last read temperature, this is required • 'max' – maximum temperature in last 24 hours, optional • 'min' – minimum temperature in last 24 hours, optional • 'avg' – average temperature in last 24 hours, optional
Required:	Optional, only on devices which have temperature sensors.

Example:	"temperature":
	{
	"last": 294.15,
	"max": 298.15,
	"min": 292.15,
	"avg": 295.15
	}

7.4.3.2. Ambient light level measured by the light sensor

Property name:	'lightLevel'
Description:	A Luminous flux value (lux) measured by light sensor. Positive floating-point value.
Required:	Optional, only on devices which can measure ambient luminous light levels.
Example:	"lightLevel": 30.0

7.4.4. POWER AND SUPPLY VOLTAGE DATA

7.4.4.1. Supply voltage

Property name:	'voltage'
Description:	Supply voltage in Volts. this is a JSON object, which contains unordered name/value pairs, where the values are a positive floating-point. This property contains the following sub properties: • 'avg' – last battery voltage, this value is required. On an active lantern it is recommended to use average value over at least one cycle of the flash character. • 'loaded' – last measured battery voltage under load conditions, optional. • 'unloaded' – last measured battery voltage under no-load conditions, optional. • 'max' – maximum battery voltage under no-load conditions in the last 24 hour, optional. • 'min' – minimum battery voltage under load conditions in the last 24 hours, optional. • 'aux' – auxiliary battery voltage, optional.
Required:	Optional, only on devices which can measure the supply voltage.

Example:	"voltage":
	{
	"avg": 12.3,
	"loaded": 12.1,
	"unloaded": 12.5,
	"max":13.3,
	"min":11.5
	}

7.4.4.2. Lantern power

Property name:	'lanternPower'
Description:	Lantern power consumption during the flash, measured in Watts (W) as an average over the flash. Positive floating-point number.
Required:	Optional, only on devices which can measure lantern power consumption during the flash.
Example:	"lanternPower": 5.3

7.4.4.3. Accumulated battery charge during last 24 hours

Property name:	'batteryCharge'
Description:	Battery charge value in Ampere-hours (Ah). The value is a floating-point number. A positive number indicates a cumulated charge over a rolling 24-hour period.
Required:	Optional, only on devices which can measure battery charge
Example:	"batteryCharge": 10.1

7.4.5. LANTERN DATA

7.4.5.1. Illumination counter

Property name:	’illuminationCounter’
Description:	This counts the time in seconds when the lantern is in the active state. This can be reset with the 'reset statistics' command."
Required:	Optional.
Example:	"illuminationCounter": 16695

7.4.5.2. Motor run time

Property name:	‘motorRunTime’
Description:	This counts the time in seconds when the motor is in the active state. This can include information from multiple devices. This can be reset with the 'reset statistics' command."
Required:	Optional.
Example:	"motorRunTime": 289431

7.4.6. GNSS AND POSITION DATA

Property name:	’ gnssData’
Description:	GNSS data is a JSON object, with parameters described separately. This is available only on devices which have GNSS receiver and use the GNSS receiver for positioning and time. Minimum required fields are location (’latLon’) and time (’time’). If position data is unavailable, then this property is omitted.
Required:	Required on devices which have GNSS for positioning and time

Example:	"gnssData":
	{
	"latLon": [60.1234567, 110.1234567],
	"time": 1673564596,
	"deviation": 10.0,
	"hdop": 1.1
	}

7.4.6.1. Location

Property name:	'latLon'
Description:	JSON array of floating-point values with last GNSS position. The latitude has a value between -90.0/90.0 decimal degrees and longitude between -180.0/180.0 decimal degrees. It is recommended to have least 7 decimal points.
Required:	Required on devices which have GNSS for positioning

Example:	"gnssData":
	{
	"latLon": [60.1234567, 110.1234567],
	"time": 1673564596
	}

7.4.6.2. Time of GNSS fix

Property name:	'time'
Description:	In seconds from January 1st, 1970, at UTC (UNIX time), As a positive integer value. To avoid the year 2038 issue, it is recommended to use at least a 32-bit unsigned value in the implementation.
Required:	Required on devices which have GNSS for positioning

Example:	"gnssData":
	{
	"latLon": [60.1234567,110.1234567],
	"time": 1673564596
	}

7.4.6.3. Deviation

Property name:	'deviation'
Description:	A positive floating-point value of distance in meters from a GNSS appointed position.
Required:	Optional on devices which have GNSS for positioning

Example:	"gnssData":
	{
	"latLon": [60.1234567,110.1234567],
	"time": 1673564596,
	"deviation": 10.0
	}

7.4.6.4. GNSS Horizontal dilution of precision (HDOP) indicator

Property name:	'hdop'
Description:	GNSS HDOP value. Positive floating-point value.
Required:	Optional on devices which have GNSS for positioning

Example:	"gnssData":
	{
	"latLon": [60.1234567,110.1234567],
	"time": 1673564596,
	"hdop": 1.1
	}

7.4.7. NETWORK DATA

Property name:	'networkStatistics'
Description:	Contains network statistical information, as a JSON object, containing the nested properties of the property names which are used in 'networkStatistics'. This is a JSON object, with unordered name/value pairs, where the value is a positive integer value. The default values are 0, with parameters described separately.
Required:	Optional. If present, it must contain all sub properties ('succeededNetworkConnections', 'failedNetworkConnections', 'succeededBrokerLogins' and 'failedBrokerLogins')

Example:	"networkStatistics":
	{
	"succeededNetworkConnections": 2,
	"failedNetworkConnections": 0,
	"succeededBrokerLogins": 2,
	"failedBrokerLogins":2
	}

7.4.7.1. Amount of succeeded network connections

Property name:	'succeededNetworkConnections'
Description:	This is the number of succeeded connections to the network as a positive integer, the default value is 0
Required:	Required only when 'networkStatistics' are used.

Example:	"networkStatistics":
	{
	"succeededNetworkConnections": 2,
	"failedNetworkConnections": 0,
	"succeededBrokerLogins": 2,
	"failedBrokerLogins":0
	}

7.4.7.2. Amount of failed network connections

Property name:	'failedNetworkConnections'
Description:	This is the number of failed connections to network as a positive integer, the default value is 0.
Required:	Required only when 'networkStatistics' are used.

Example:	"networkStatistics":
	{
	"succeededNetworkConnections": 0,
	"failedNetworkConnections": 2,
	"succeededBrokerLogins": 0,
	"failedBrokerLogins":0
	}

7.4.7.3. Amount of succeeded broker logins

Property name:	'succeededBrokerLogins'
Description:	This is the number of successful logins to the broker as a positive integer, the default value is 0.
Required:	Required only when 'networkStatistics' are used.

Example:	"networkStatistics":
	{
	"succeededNetworkConnections": 2,
	"failedNetworkConnections": 0,
	"succeededBrokerLogins": 2,
	"failedBrokerLogins":0
	}

7.4.7.4. Amount of failed broker logins

Property name:	'failedBrokerLogins'
Description:	This is the number of failed logins to the broker As a positive integer, the default value is 0.
Required:	Required only when 'networkStatistics' are used.

Example:	"networkStatistics":
	{
	"succeededNetworkConnections": 2,
	"failedNetworkConnections": 0,
	"succeededBrokerLogins": 0,
	"failedBrokerLogins":2
	}

7.4.8. SYSTEM DATA

System data is optional, but it is useful for problem solving. The system data block should only be transmitted during the first session following a device restart.

7.4.8.1. Cause of last reset

Property name:	'lastResetSource'
Description:	String that contains the last reset source. Allowed values are: • 'por' – power on reset • 'wdr' – watchdog reset • 'rst' – reset from an external reset signal (hardware reset signal) • 'bor' – brown-out reset • 'user' – user or software triggered reset, e.g. reset triggered by user command • 'other' – all other reset sources
Required:	Optional
Example:	"lastResetSource": "wdr"

7.4.8.2. Reset count

Property name:	'resetCount'
Description:	Contains the nested properties of the property names which are used in 'lastResetSource'. This is a JSON object, with unordered name/value pairs, where the value is a positive integer value. The default value is 0.
Required:	Optional, only when last reset source is present.

Example:	"resetCount":
	{
	"por": 30,
	"wdr": 1
	}

7.4.9. EVENT LOG

Property name:	'eventLog'
Description:	An array of objects that contains the device's recent events. The array size is device-dependent, but it is recommended that the array holds at least two history objects. Each object contains the following properties: 'time', 'lanternStatus', and an optional 'alert' property. Time represents the number of seconds since January 1st, 1970, at UTC (UNIX time). The 'alert' array and 'lanternStatus' properties contain values from their respective properties, except for the alert value 'none', which can be omitted.
Required:	Optional

Example:	"eventLog":
	[
	{
	"time": 1715156630,
	"alert": ["lampFail"],
	"lanterntatus": "off"
	},
	{
	"time": 1715156730,
	"lanternStatus": "onMainCharacter"
	}
	]

7.4.10. EXAMPLES OF JSON MASSAGES

(1) Minimal telematic JSON message for generic device:


```
{
 "sessionId": "session-1",
 "time": 1715156630,
 "status": "ready",
 "uptime": 20
}
```


(2) An example of a typical lantern telematics JSON message:


```
{
 "sessionId": "session-1",
 "time": 1673564596,
 "status": "ready",
 "uptime": 20,
 "lanternStatus": "onMainCharacter",
 "lightSensorStatus": "night",
 "temperature":
 {
  "last": 24.5,
  "max": 27.0,
  "min": 21.5,
  "avg": 23.0
 },
 "voltage":
 {
  "avg": 13.10,
  "max": 13.44,
  "min": 12.73
 },
 "lightLevel": 41,
 "gnssData":
 {
  "latLon": [60.0, 110.0],
  "time": 1673564590,
  "deviation": 22,
  "hdop": 1.6
 },
 "networkStatistics":
 {
  "succeededBrokerLogins": 1,
  "failedBrokerLogins": 2,
  "succeededNetworkConnections": 0,
  "failedNetworkConnections": 2
 },
 "eventLog":
```

```
 [
  {
   "time": 1615156630,
   "alert": ["lampFail"],
   "lanternStatus": "off"
  },
  {
   "time": 1715156730,
   "lanternStatus": "onMainCharacter"
  }
 ]
}
```


7.5. TOPIC 'INFO'

This set of topics outlines the device's capabilities and restrictions to the server.

The Information is triggered by the info request command. The request topic is 'cmd/' and its message payload format is '{"send": "info"}'.

7.5.1. PROTOCOL VERSION DATA

Property name:	'protocolVersion'
Description:	This is an integer value that describes the protocol version. The present value is 1.
Required:	Yes
Example:	"protocolVersion": 1

7.5.2. METADATA

Property name:	'metadata'
Description:	This is a map of nested maps and strings with supported properties and limit values. This property contains only topics that can have data from the device side which can be stored to the database and does not contain commands. All values must match with the parameters. The map key contains the nested map name or the properties name, and the value contains the nested map or value limits in interval notation. The limit value may contain a number of limiting number values, a list of allowed strings or an empty string. • The rectangular bracket symbols, "[" and "]", are used to represent sets with a "less than or equal to" or a "greater than or equal to" element, respectively. They correspond to the ≥ and ≤ symbols. The parentheses symbols, "(" and ")", are used to represent sets with a lower bound or upper bound, respectively. They correspond to the > and < symbols. Special string "inf" is used for infinity. • The value may contain a comma separated list of allowed strings. • The value may contain a map for nested parameters.
Required:	Yes

Example:	"metadata":
	{
	"tele":
	{
	"lightIntensity": "[0, inf]",
	"lowVoltageLevel": "[0,20)",
	"time": "",
	"lastResetSource": "wdr, por",
	"networkStatistics":
	{
	"succeededBrokerLogins": "",
	"failedBrokerLogins": "",
	"succeededNetworkConnections": "",
	"failedNetworkConnections": ""
	}
	},
	"parameter":
	{
	"mode": " byLight "
	}
	}

7.5.3. SYSTEM INFORMATION

Property name:	'sysInfo'
Description:	This information can be used to set up an optimal set/get transmission packet sizes as a JSON object, in an unordered name/value pairs, where the value is an integer value. Allowed system parameters are: • 'rxBuf' – size of receive buffer in bytes, -1 means infinite • 'txBuf' – size of transmit buffer in bytes, -1 means infinite
Required:	Yes

Example:	"sysInfo":
	{
	"rxBuf": 128,
	"txBuf": -1
	}

7.5.4. TYPE

Property name:	'type'
Description:	This is a string, which contains the type of device: • 'group' – logical container for device group, used for MQTT gateways • 'lantern' – for lanterns The type 'group' can be used where devices are not connected via TCP/IP directly. but via another means e.g. LoRaWAN.
Required:	Optional
Example:	"type": "lantern"

7.5.5. SERIAL NUMBER

Property name:	'serialNr'
Description:	A string containing the device serial number. It is recommended that the string also include the product code to ensure a unique identifier.
Required:	Optional. Only on devices that have valid serial number.
Example:	"serialNr": "product123-001"

7.5.6. PRODUCT CODE

Property name:	'productCode'
Description:	A string with the device product code.
Required:	Optional. Required only when the serial number does not have the product information.
Example:	"productCode": "product123"

7.5.7. FIRMWARE VERSION

Property name:	'firmwareVersion'
Description:	A string containing the firmware version.
Required:	Optional. Only on devices that have firmware version information.
Example:	"firmwareVersion": "version-1.0"

7.5.8. COMPONENT INFO

Property name:	'componentInfo'
Description:	Contains device component information. A list of JSON objects, as unordered name/value pairs, where the values are strings. The allowed information parameters are: • 'component' – component description. • 'version' – component version. • 'id' – component ID or serial number. This parameter is optional and can be used in the topics names to address sub-devices.
Required:	Optional. Only for devices or systems that have sub-components or sub-devices.

Example:	"componentInfo":
	[
	{
	"component": "gnss",
	"version":"1.0"
	},
	{
	"component": "flasher",
	"version": "2.0",
	"id": "flasher1"
	}
	]

7.6. TOPIC 'CMD'

Command topics are intended for server-initiated actions. Commands do not store data directly to non-volatile memory, although it is possible that data is stored as result of a command action. It is allowed that the command may lock the device for a short time. MQTT retention must be disabled for commands.

7.6.1. SEND

Property name:	'send'
Description:	A string with a requested topic. Allowed values are: • 'tele' – for telematics • 'info' – for information
Required:	Yes
Example:	"send": "info"

7.6.2. RECONNECT

Property name:	'reconnect'
Description:	This command disconnects and reconnects the device from the broker. Any parameter value is accepted, but null is preferred.
Required:	Optional
Example:	"reconnect": null

7.6.3. RESET

Property name:	'reset'
Description:	This is a string which describes a device or parameter to reset. When a reset device command is issued, this command must close the MQTT session. Allowed values are: • null, empty string or 'reset'– reset device (required) • 'parameters' – reset all parameters (optional) • 'statistics' – reset all statistics (optional) • <parameterName> – reset parameter name to default (optional)
Required:	Optional
Example:	"reset": null

7.6.4. RELOAD CONFIGURATION

Property name:	’reloadConfig’
Description:	Loads a new configuration from the configuration storage to the device registers or Random Access Memory (RAM). This command is used after updating the parameter values when the device needs additional steps to apply new parameters. For example, load a new GNSS acquisition period from the configuration storage to the GNSS handler. Allowed values are: • null • all other parameters are reserved for future use
Required:	Optional
Example:	"reloadConfig": null

7.6.5. TIME

Property name:	’time’
Description:	This sets or gets the time value in seconds, this is a complementary property to the ‘time’ property in the ‘general’ message, and follows the same format in seconds from January 1st, 1970, at UTC (UNIX time). Allowed values are: • null – query from the server • any positive integer – sets a new time value
Required:	Optional
Example:	"time": 1673564596

7.6.6. LIGHT ON DEMAND

Property name:	'lightOnDemand'

Description:	This parameter describes how many seconds the light on demand mode is to be active. The device returns to a normal operation after this time has elapsed. Optionally, this command also allows the intensity to be specify for the duration of the light on demand operation.
	Allowed values are:
	• null – query from server, the server sends the current status, including duration and intensity values
	• sub-properties to describe the operation
	  ○ 'duration' – a signed 32 bit integer for the duration of light activation in seconds. If the 'duration' property is missing, then the light on demand function will be cancelled (same as value 0).
	  ▪ -1 – light on demand is active until switched off
	  ▪ 0 – light on demand switched off
	  ▪ 1...2147483648 – seconds active
	  ○ 'intensity' – integer, effective intensity in candelas, optional

Required:	Optional, only when device supports it.

Example:	Light activated for one hour with 300 cd effective intensity:
	"lightOnDemand":
	{
	"duration": 3600,
	"intensity": 300
	}

7.6.7. FIXED POSITION

Property name:	'fixPosition'
Description:	This is used for the device to calculate and store its appointed position. Allowed values are: • 'start' – start position fix • null – optional query from server, returns the current position. Responses to query: • 'started' – position fix has already stated • 'idle' – position fix has not started or is already completed • number between 0 to 100 – percentage value of completed fix, can be used instead 'started'
Required:	Optional
Example:	"fixPosition": "start"

7.6.8. EXAMPLE OF THE TOPIC ‘ CMD’

(1) Command from server: start position fix and set time.

```
{
  "time": 1715169904,
  "fixPosition": "start"
}
```

(2) Time query command from server:

```
{
  "time": null
  "resTopic": "630c8f35-148f-4961-8b56-96586b021ba0"
}
```

(3) Response to time query:

```
{
  "sessionId": "session-1",
  "time": 1715169904
}
```

7.7. TOPICS 'PARAMETER'

The 'parameter' topic is used to store or retrieve device configuration parameters. Values that are stored in non-volatile memory are all considered as parameters. All get commands must include a 'resTopic' property. The value of the 'resTopic' is a string, which is used for sending a response from the device. This value is generated from the server and can be any unique string. It is allowed to use random characters or topic name with 'res' suffix. All topic queries have a parameter value null. If the device supports the requested parameter, but has an invalid value stored for the parameter, then null should be returned. It is allowed to use MQTT retention on these commands. Below are some examples.

Get and set in one message:

```
{
 "resTopic": "parameter/locationa/locationb/site/device1/res",
 "mode": null,
 "lightIntensity": null,
 "lightLevel": 50
}
```

Response to 'parameter/locationa/locationb/site/device1/res':

```
{
 "sessionId": "session-1",
 "mode": " byLight "
 "lightIntensity": 34
}
```

Set:

```
{
 "resTopic": "parameter/locationa/locationb/site/device1/res",
 "mode": "idle"
 "lightIntensity": 30
}
```

7.7.1. AUTOMATIC DATA

Property name:	'autoData'
Description:	This configures the device for what information is to be sent on each occasion the device connects. This is over and above what the device sends due to a change or other reasons. This can be considered as a default set of information but note the impact this may have on the data volume transmitted. By default, a telemetry packet is used. Reserved values are: • 'tele' – a telemetry packet is sent automatically, this is the default. • 'proprietary' – a proprietary data packet is sent automatically. The data which is sent is device dependent. It is allowed to use additional values by user.
Required:	Optional
Example:	"autoData": "proprietary"

7.7.2. LANTERN MODE

Property name:	’mode’
Description:	Set or get lantern operation mode. Allowed string values: • null – query from server • ’ byLight’ – operation is controlled by light sensor, this is the default • ’alwaysNight’ – always operating in night mode (forced), as if the light sensor detected night (optional) • ’alwaysDay’ – always operating in day mode (forced), as if the light sensor has detected day (optional) • ‘scheduled’ - operations is controlled based on a fixed template • ‘astroClock’ - operations is controlled based on the calculated dusk and dawn times. • ’idle’ – idle mode, normal operation is turned off, lowest power consumption
Required:	Optional
Example:	"mode": " byLight "

7.7.3. ASTRONOMICAL DUSK AND DAWN

Property name:	‘astroClock’
Description:	This is a common property for astronomical clock function in a JSON object format, see parameter descriptions below. The allowed sub properties are: • null – query all from server • ‘duskOffset’ – This is a signed integer value in minutes where a positive signed value increases the time when the dawn to dusk transition occurs. A negative signed value decreases the time when the dawn to dusk transition occurs. • ‘dawnOffset’ – This is a signed integer value in minutes where a positive signed value increases the time when the dusk to dawn transition occurs. A negative signed value decreases the time when the dusk to dawn transition occurs. • ‘astroPosition’ – These are values in latitude and longitude of a position that is used to calculate the dusk and dawn times. If set to zero the actual GNSS position will be used.
Required:	Optional.

Example:	"AstroClock":
	{
	"duskOffset":-50,
	"dawnOffset":+30,
	"fixPosition": [30.0, 100.0]
	}

7.7.4. LANTERN FLASH CHARACTER

Property name:	'flashCode'
Description:	This is a common property for flash code in a JSON object format, see parameters descriptions below. The allowed sub properties are: • null – query all from server • 'main' – main flash character • 'secondary' – secondary flash character
Required:	Optional

Example:	"flashCode":
	{
	"main": [0.5,0.5],
	"secondary": [1,1]
	}

7.7.4.1. Main flash character

Property name:	'main'
Description:	Set or get flash character. JSON array of floating-point pairs that describes the flash character. Every odd element in array, describes the flash duration in seconds. Every even element describes the eclipse duration in seconds. The last eclipse character may be omitted to get a fixed light. Allowed values: • null – query from server • array of floating-point values to describe the flash character
Required:	Required for all flashers.

Example:	VQ(6)+LFL 10s south cardinal flash character:
	"flashCode":
	{
	"main": [0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 3, 3.4]
	}
	
	Flash code where eclipse is absent, light is always on i.e. fixed light.
	"flashCode":
	{
	"main": [1]
	}

7.7.4.2. Secondary flash character

Property name:	'secondary'
Description:	See description in 7.7.4.1 - **Erreur ! Source du renvoi introuvable.**.
Required:	Required for flashers that support a secondary flash character.

Example:	"flashCode":
	{
	"secondary": [0.5, 0.5]
	}

7.7.5. OUTPUT INTENSITY

Property name:	'intensity'
Description:	The Intensity values for the lantern. A common property for luminous effective intensity in candelas. A JSON object, see parameters descriptions below. Allowed sub-properties are: • null – query all values from server. • 'defaultIntensity' – intensity which is used during the night. • 'alternativeIntensity' – intensity which is used during the day.
Required:	Optional

Example:	"intensity":
	{
	"defaultIntensity": 500,
	"alternativeIntensity": 1000
	}

7.7.5.1. Default output intensity

Property name:	'defaultIntensity'
Description:	This sets or gets the main light output intensity in candelas. The default intensity is normally used as light output intensity during night. Allowed values: • null – query from server • any positive number – light intensity in candelas
Required:	Only on flashers

Example:	"intensity":
	{
	"defaultIntensity": 500
	}

7.7.5.2. Alternative output intensity

Property name:	'alternativeIntensity'
Description:	This set or gets the secondary light output intensity in candelas. The secondary intensity is often used as light output intensity during day.
Required:	Only on flashers that supports secondary intensity
Parameter format:	Allowed values: • null – query from server • any positive number – light intensity in candelas

Example:	"intensity":
	{
	"alternativeIntensity": 1000
	}

7.7.6. TWILIGHT SWITCH

Property name:	'twilightSwich'
Description:	This sets or gets the twilight switching function values. 'dusk' and 'dawn' represent the light levels that triggers the operation of the lantern. 'timeDusk' and 'timeDawn' defines the duration of time in which the dusk or dawn state must remain until the function is triggered. A JSON object, see parameters descriptions below. Allowed sub-properties are: • null – query all values from server. • 'dusk' - unsigned integer value lower than Dawn in lux • 'dawn' - unsigned integer value higher than Dusk in lux • 'timeDusk' – unsigned integer in seconds • 'timeDawn' – unsigned integer in seconds
Required:	Optional

Example:	"twilightSwich"
	{
	"dusk": 90,
	"dawn": 120
	"timeDusk": 0,
	"timeDawn": 120
	}

7.7.7. LANTERN TRIGGER

Property name:	'lanternTrigger'
Description:	This set or get the lantern trigger configuration. Allowed values: • null – query from server • 'byLight' – the lantern is triggered by the light sensor. This is the default value. • 'calculated' – the lantern is triggered by a calculated or predefined condition e.g. astronomical clock. • 'off' – lantern is always off. • 'on' – lantern is always on.
Required:	Optional
Example:	"lanternTrigger": "byLight"

7.7.8. LOW BATTERY VOLTAGE WARNING LEVEL

Property name:	'lowVoltageWarningLevel'
Description:	This set or get the low voltage warning level. A warning message 'lowBatteryWarning' is sent when device power supply voltage drops below a low voltage warning level. This level must be higher or equal to the low voltage level. It is allowed to change the device operational mode to consume less power. Allowed values: • null – query from server • any positive floating-point number – voltage level
Required:	Optional
Example:	"lowVoltageWarningLevel": 11.1

7.7.9. HIGH BATTERY VOLTAGE WARNING LEVEL

Property name:	'highVoltageWarningLevel'
Description:	This set or get the high voltage warning level. A warning message 'highBatteryWarning' is sent when device power supply voltage exceeds a voltage warning level. Allowed values: • null – query from server • any positive floating-point number – voltage level
Required:	Optional
Example:	"highVoltageWarningLevel": 16.1

7.7.10. LOW BATTERY VOLTAGE LEVEL

Property name:	'lowVoltageLevel'
Description:	This set or get the low voltage level. A warning message 'lowBattery' is sent when the device power supply voltage drops below the low voltage level. This level must be lower or equal to the low voltage warning level. It is allowed to change the device's operational mode to consume less power. Allowed values: • null – query from server • any positive floating-point number – voltage level
Required:	Optional
Example:	"lowVoltageLevel": 10.1

7.7.11. TELEMETRY

Property name:	'telemetry'
Description:	A common property for the telemetry. A JSON object, see parameters descriptions below. It is allowed to send a query with null parameter, this will return all available parameters with current values. Allowed sub-properties are but not limited to: • null – query all from server • 'reportMode' – telemetry report mode • 'reportPeriod' – telemetry report period • 'apn' – access point name • 'apnUser' – APN username • 'apnPassword' – password for APN • 'brokerAddress' – broker address • 'mqttClientId' – MQTT client ID • 'mqttUser' – MQTT username for broker • 'mqttPassword' – password for username • 'uid' – unique ID for device or location string for MQTT topic
Required:	Optional

Example:	"telemetry":
	{
	"reportMode": "interval",
	"reportPeriod": 300,
	"apn": "internet",
	"apnUser": "user",
	"apnPassword": "secret",
	"brokerAddress": "example.org",
	"mqttClient-id": "client123",
	"uid": "location1/location2/site"
	}

7.7.11.1. Telemetry report mode

Property name:	'reportMode'
Description:	• A set or get telemetry report mode that is an array where the first value is the main report mode and the second value is the supplementary mode. Allowed values are: • null – query from server • 'off' – telemetry data is sent only after a request using 'cmd'. • 'onEvent' – only when a state change and / or an error condition is detected. This is the default mode. • 'utcFixed' – UTC fixed mode, for example 00:00, 00:05, 00:10, etc. A delay can be added to the report period to spread simultaneous sessions. The delay length is implementation defined, for example a delay in seconds can be calculated from device's serial number. • 'interval' – interval mode, for example can be every 3 minutes, so not fixed to UTC. The start of the interval session need to be randomized in a similar manner to the 'utcFixed' parameter. • The supplementary mode can only be 'onEvent', and it is only allowed with the 'utcFixed' and 'interval' modes.
Required:	Optional

Example:	(1) Interval mode only
	"telemetry":
	{
	"reportMode": ["interval"]
	}
	
	(2) Interval and onEvent mode
	"telemetry":
	{
	"reportMode": ["interval", "onEvent"]
	}

7.7.11.2. Telemetry report period

Property name:	'reportPeriod'
Description:	A set or get telemetry report period for the utcFixed and interval modes. Allowed values are: • null – query from server • 0 – disable the telemetry period, telemetry is sent only after a query request • any positive number – for the telemetry period in seconds
Required:	Optional

Example:	"telemetry":
	{
	"reportPeriod": 300
	}

7.7.11.3. Access point name - APN

Property name:	’apn’
Description:	A set or get APN. Allowed values are: • null – query from server • string – APN name
Required:	Optional

Example:	"telemetry":
	{
	"apn": "internet"
	}

7.7.11.4. APN user

Property name:	’apnUser’
Description:	A set or get APN username. Allowed values are: • null – query from server • string – A1715169904PN username
Required:	Optional

Example:	"telemetry":
	{
	"apnUser": "user"
	}

7.7.11.5. APN password

Property name:	’apnPassword’
Description:	A set or get password for APN. Allowed values are: • null – query from server • string – APN password
Required:	Optional

Example:	"telemetry":
	{
	"apnPassword": "secret"
	}

7.7.11.6. Broker address

Property name:	'brokerAddress'
Description:	A set or get the broker address. This is a list of broker addresses. The addresses may contain a TCP/IP port if a non-default port is used. IP addresses are also supported. Changing the active address is not allowed. To change the active address, the device should add another address and use the second address to make the change. Allowed values: • null – query from server • list of addresses with port, may contain current address
Required:	Yes

Example:	Single address:
	"telemetry":
	{
	"brokerAddress": ["example.org"]
	}
	
	Address list with fallback:
	"telemetry":
	{
	"brokerAddress": ["example1.org", "example-fallback.org"]
	}

7.7.11.7. MQTT client ID

Property name:	'mqttClientId'
Description:	A set or get MQTT ClientId [3][4]. While this parameter allows the writing of a new ClientId value, it is still recommended that this parameter is read only, and ClientId value should only be changed locally without the use of MQTT. This parameter may have same value as product serial number. Allowed values are: • null – query from server • string with client id
Required:	Yes. Device should respond to query but may ignore a write.

Example:	"telemetry":
	{
	"mqttClientId": "mqtt client"
	}

7.7.11.8. MQTT username

Property name:	'mqttUser'
Description:	A set or get MQTT username. It is not required to implement this query on the device side. Generic topic queries should not list usernames. Can be only used with encrypted payloads. Allowed values are: • null – query from server • MQTT username string
Required:	Optional. Query may be omitted.

Example:	"telemetry":
	{
	"mqttUser": "MQTT User"
	}

7.7.11.9. MQTT password

Property name:	'mqttPassword'
Description:	A set MQTT password. This supports only write commands. Can be only used with encrypted payloads. Allowed values: • MQTT password string
Required:	Optional. Query may be omitted.

Example:	"telemetry":
	{
	"mqttPassword": "secret"
	}

7.7.11.10. Unique ID

Property name:	'uid'
Description:	A set or get device unique ID string. Unique ID is a string that is used to describe the device's location or the device's serial number in MQTT topic name. Allowed values: • null – query from server • unique ID string. It is allowed to use forward slashes for a multilevel ID string. The device location string is formatted <location1>/<location2>/<site> or the device serial number can be used for unique ID.
Required:	Yes

Example:	"telemetry":
	{
	"uid": "location1/location2/site"
	}

7.7.11.11. Certificate selection

Property name:	'cert'
Description:	A set or get certificate identification. This allows for the selection of the certificate from stored certificate list that is used for communication. Allowed values are: • null – query from server • string with certificate identification
Required:	Optional. Only on devices that use TLS/SSL.

Example:	"telemetry":
	{
	"cert": "aws"
	}

7.7.12. GNSS AND POSITION

Property name:	'gnss'
Description:	A common property for GNSS and positioning. See parameters descriptions below. Allowed sub-properties are: • null – query all from server • 'base' – GNSS wake-up base • 'interval' – interval of GNSS time and position checkup • 'duration' – duration of GNSS time and position checkup • 'fixPosition' – latitude and longitude of fix position • 'distanceFromFix' – distance from fixed position • 'sync' – flash code synchronization base
Required:	Optional. Only for devices which have GNSS

Example:	"gnss":
	{
	"base": "async",
	"interval": 300,
	"duration": 120,
	"fixPosition": [60.0, 110.0],
	"distanceFromFix": 30,
	"sync": "off"
	}

7.7.12.1. GNSS base

Property name:	'base'
Description:	Select GNSS wake-up base. Allowed values: • null – query from server • 'async' – not synchronized with other tasks (optional) • 'preTelematics' – always before telematics session (optional). If interval is set, then GNSS is started before the telematics and repeated at an interval. Completion of GNSS task triggers the telematics task. • 'utc' – synchronized with UTC (optional) • If device supports only one wake-up base, then this command returns only supported base.
Required:	Optional

Example:	"gnss":
	{
	"base": "async"
	}

7.7.12.2. GNSS interval

Property name:	'interval'
Description:	A set or get time interval for GNSS time and position check. Allowed values are: • null – query from server • 0 – disable GNSS time and position periodical check • any positive number – interval in seconds
Required:	Optional

Example:	"gnss":
	{
	"interval": 300
	}

7.7.12.3. GNSS duration

Property name:	’duration’
Description:	A set or get duration for GNSS time and position check. Allowed values are: • null – query from server • 0 – disable GNSS time and position check • any positive number – duration in seconds
Required:	Optional

Example:	"gnss":
	{
	"duration": 120
	}

7.7.12.4. Latitude and longitude of fixed position

Property name:	’fixPosition’
Description:	A set or get latitude and longitude of fix position. An array, where first value is latitude and second value is longitude. Positive values indicate Northern latitudes and Eastern longitudes. Allowed values are: • null – query from server • array of degrees for latitude -90.0/90.0, and for longitude -180.0/180.0
Required:	Optional

Example:	"gnss":
	{
	"fixPosition": [60.0, 110.0]
	}

7.7.12.5. Maximum allowed distance from fix position

Property name:	’distanceFromFix’
Description:	A set or get distance from the fix position. Allowed values are: • null – query from server • any positive floating-point number – distance from the fixed position in meters
Required:	Optional

Example:	"gnss":
	{
	"distanceFromFix": 30
	}

7.7.12.6. GNSS sync

Property name:	'sync'
Description:	A set or get GNSS synchronization for the flash character. Only available when positioning is enabled. Allowed values are: • null – query from server • 'off' – GNSS sync disabled • 'utc' – UTC based GNSS sync (start of flash adjusted to UTC 00:00:00) • 'gps' – GPS time based GNSS sync (start of flash adjusted to TOW 00:00:00)
Required:	Optional

Example:	"gnss":
	{
	"sync":"utc"
	}

7.8. TOPIC 'PROPRIETARY'

This topic is intended to pass raw data to sub-modules, for example Modbus packets. All get commands must have 'resTopic' property. Proprietary topic to a device contains "resTopic" and "data" properties and to server contains session "sessionId" and "data" properties.

JSON string to from server to controller:

```
{
 "resTopic": "proprietary/locationa/locationb/site/device1/res"
 "flags": ["noResponse"],
 "timeout": 1000,
 "data": "0011"
}
```

JSON string to from controller server to server:

```
{
 "sessionId": "session-1",
 "data": "0011"
}
```

When "no response" flag is set:

```
{
 "sessionId: "session-1",
 "result": "ok"
}
```

When device does not respond in requested time:

```
{
 "sessionId": "session-1",
 "result": "timeout"
}
```

7.8.1. FLAGS PROPERTY

Property name:	'flags'
Description:	A send direct command flags. If this property is omitted, then the default values is used. Array of strings with flag description: • noResponse – no response is expected
Required:	Optional
Example:	"flags": ["noResponse"]

7.8.2. TIMEOUT PROPERTY

Property name:	'timeout'
Description:	A property command timeout in seconds. This is a timeout period by which time it is expected to have response from device. If the "noResponse" flags are active, then this command waits for the given timeout before processing the next command. If this property is omitted, then no timeout is used, this is default.
Required:	Optional
Example:	"timeout": 2

7.8.3. RESULT PROPERTY

Property name:	'result'
Description:	This returns a result of operation. This property is sent only by the device. Allowed string values are: • ok – operation was successful, this value is not needed on successful operation and when 'data' property is set. It is used when 'noResponse' command returns without errors. • timeout – timeout occurs. • error – any generic error: read errors, write errors, Cyclic Redundancy Check (CRC) errors, etc.
Required:	Optional, may be omitted when device respond with data property and no errors are set.
Example:	"result": "ok"

7.8.4. DATA PROPERTY

Property name:	'data'
Description:	A send and receive data from device as a string with hexadecimal coded bytes. If response does not contain any data, then an empty string will be sent back. It is possible to send to only one device at time, no multicast is allowed.
Required:	Optional

Example:	"data": "A011"

7.8.5. REQUEST PROPERTY

Property name:	'requestId'
Description:	The server uses this property to keep track of the sessions. If this property is present, then the device adds it to the response messages that result from server's actions. The message ID should not be longer than 32 bytes.
Required:	Optional
Example:	"requestId": "uniqueMsgID1"

7.9. TOPIC 'REGISTRATION'

This topic covers device registration, certificate download, and certificate renewal. To register a device with the service, a new device connects to the broker and sends a registration request. In the simplest case, the registration request includes the following fields: 'registration', 'mqttClientId' and 'resTopic'. Unlike other topics, the registration process uses the same 'resTopic' value for the entire session.

```
{
  "sessionId": "session-1",
  "time": 1715169904,
  "resTopic": "registration/locationa/locationb/site/device1/res",
  "command": "register",
  "mqttClientId": "loca-locb-dev1"
}
```

Response for registration:

```
{
  "result": "registered"
}
```

After a successful response, the device sends its information (7.5 - **Erreur ! Source du renvoi introuvable.**).

In case of error, the server responds with error message:

```
{
  "result": "failed",
  "errorMessage": "Access denied"
}
```

7.9.1. COMMAND PROPERTY

Property name:	'command'
Description:	Registration action. Allowed values are: • 'register' – registers the device without any security measures (no TLS) • 'deregister' – deregisters the device from the service. This command can be sent by either the server or the device. In both cases, the device and server disable or remove TLS certificates. After this command, a new registration is required to reconnect to the service. This command is optional. • 'registerWithCSR' – registers the device using a CSR. The CSR and private keys should be generated by the device, and if possible, the keys should be securely stored in the device's non-volatile memory. This command is optional.
Required:	Yes
Example:	"registration": "register"

7.9.2. MQTT CLIENT ID PROPERTY

Property name:	'mqttClientId'
Description:	**Erreur ! Source du renvoi introuvable.**
Required:	Yes
Example:	"mqttClientId": "loca-locb-dev1"

7.9.3. KEY PROPERTY

Property name:	'key'
Description:	Used as an additional measure for device authentication. During registration, the server can send a one-time authentication key to local service personnel, who enter the key before starting the device registration.
Required:	Optional
Example:	"key": "abc123"

7.9.4. CERTIFICATE SIGNING REQUEST PROPERTY

Property name:	'certificateSigningRequest'
Description:	Contains a CSR in Privacy-Enhanced Mail (PEM) format. It is only used when the registration property has the value 'registerWithCSR'. The CSR should include the subject Common Name (CN) with the 'mqttClientId' which can later be used by load balancers. The CSR uses a private key that is generated inside the device and preferably stored in a secure non-volatile memory location.
Required:	Optional
Example:	"certificateSigningRequest": "-----BEGIN NEW CERTIFICATE ..."

7.9.5. CERTIFICATE PEM PROPERTY

Property name:	'certificatePem'
Description:	The certificate data in PEM format. This is the response for a successful 'registerWithCSR' request.
Required:	Optional
Example:	"certificatePem": "---BEGIN ..."

7.9.6. ERROR MESSAGE PROPERTY

Property name:	'errorMessage'
Description:	String with error message.
Required:	Yes
Example:	"errorMessage": "Access Denied"

8. SECURITY AND ENCRYPTION

1 For minimum security, MQTT username and password authentication should be used [5]. Both values should not be accessible directly by an unencrypted communication channel. It is recommended to use other communication method, like local access, to change username and password.

2 Client-to-broker encryption. It is possible to encrypt MQTT session by using common session encryption methods such as TLS. Client to broker encryption can be used in places where broker is configured to block connections between devices, and full control is only from end-server. Operations performed with certificates are described in chapters 4.3 – Device registration and certificate renewal and 7.9.6 – Error message property.

9. BIBLIOGRAPHY

1 Introducing JSON, 2023, https://www.json.org/json-en.html

2 The JavaScript Object Notation (JSON) Data Interchange Format, 2017, https://www.rfc-editor.org/rfc/rfc8259

3 Client identifier, 2023-09-06, https://www.ibm.com/docs/en/ibm-mq/9.3?topic=concepts-client-identifier

4 MQTT Client, MQTT Broker, and MQTT Server Connection Establishment Explained – MQTT Essentials: Part 3, 2023-06-06, https://www.hivemq.com/blog/mqtt-essentials-part-3-client-broker-connection-establishment/

5 Securing MQTT with Username & Password Authentication, 2023-03-30, https://www.emqx.com/en/blog/securing-mqtt-with-username-and-password-authentication