|  |
| --- |
| IALA Guideline |

Gnnnn

Harmonised IoT Protocol

for Visual AtoN

Edition x.x

Date (of approval by Council)

urn:mrn:iala:pub:gnnnn

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

|  |  |  |
| --- | --- | --- |
| Date | Details | Approval |
| 02/02/24 | Reviewed and updated. |  |
| 05/06/24 | Appendix changed |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

1. Introduction 6

2. Scope 6

3. IoT Infrastructure 6

3.1. Topology 6

3.1.1. 3rd Party platform. 6

3.1.2. An organisational developed platform. 7

3.2. IoT Communication Protocol 7

4. Security 9

4.1. Encryption and Authentication 9

5. Harmonized payload 10

5.1. Message structure. 10

5.2. Payload 11

6. Example of HEADING 1 STYLE 14

6.1. Example of Heading 2 style 14

6.1.1. Example of Heading 3 style 14

7. SECTION 2 – HEading 1 style 15

7.1. Section 2.1 – Heading 2 style 15

8. Section 3 – Heading 1 Style 16

8.1. Section 3.1 – Heading 2 style 16

8.2. Section 3.2 – Heading 2 style 16

8.3. Section 3.3 – Heading 2 style 16

8.3.1. Equations 16

9. DEFINITIONS 17

10. abbreviations 18

11. references 18

12. Further reading 18

13. Index 19

List of Tables

Table 1 Comparison of various IoT protocols 8

Table 2 Example of table with row headers 15

Table 3 Example of table with column headers 15

Table 4 Table 2 Example of retained values for a minimum QoS level 36

List of Figures

Figure 1 Topology of a 3rd party IoT solution 7

Figure 2 Topology of an organisation IoT platform 7

Figure 3 Example of publishing and subscribing 9

Figure 4 Example of MQTT Security, Courtesy of Cirrus Link need to recreate 10

Figure 5 An example of JSON syntax, https://en.wikipedia.org/wiki/JSON 11

Figure 6 Example of wrapping in line with text 14

Figure 7 Example of wrapped square 15

Figure 8 Example of how to achieve right justified equation number 17

Figure 9 Simple architecture 20

Figure 10 Complex architecture 20

Figure 11 Minimal system 21

Figure 12 Typical system 21

Figure 13 Telemetry session 24

Figure 14 Info session 25

Figure 15 Get parameter 26

Figure 16 Set parameter 27

Figure 17 Proprietary session 28

Figure 18 Persistent session 29

Figure 19 Failed persistent session 30

Figure 20 Device provisioning 32

Figure 21 General message 38

Figure 22 General message 38

# 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 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). Additionally, many current conventional communication systems have a limitation in the number of simultaneous communication sessions they can manage, so reporting frequency is not just limited by outstations.

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.

# Scope

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

# IoT Infrastructure

When considering the development of a protocol for interoperability of visual AtoN , we need to understand that the AtoN equipment can be considered as an IoT device with various sensors. This device needs to send its information to an IoT application to process and display the information for a user. 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 by a 3rd party e.g. Azure or can be delivered within the organisation existing infrastructure.

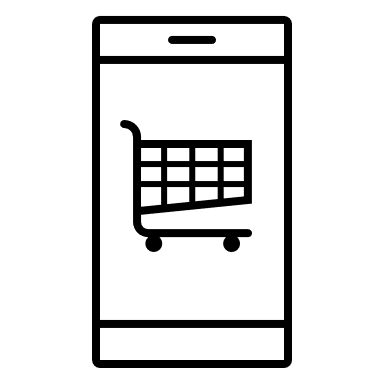
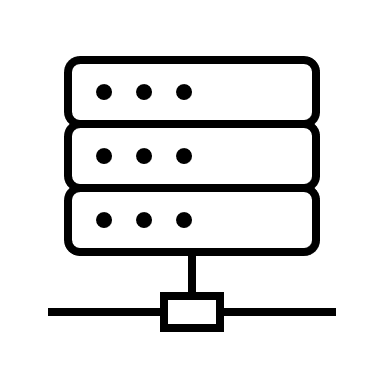
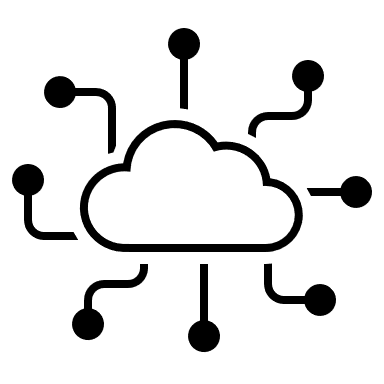
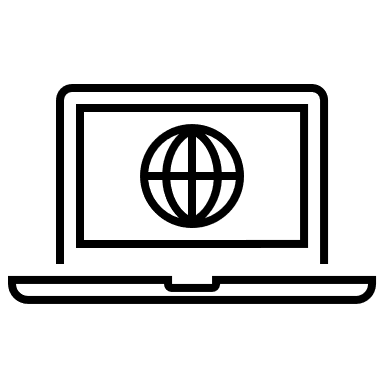
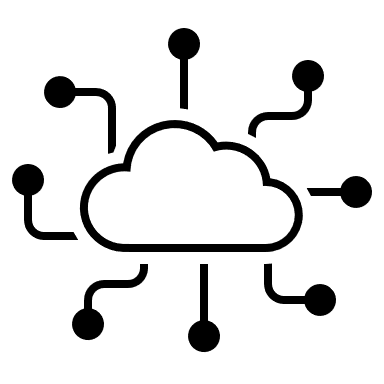
## Topology

The type of topology adopted, will be subject to the how the IoT service is to be provided. This section shows two examples.

### 3rd Party platform.

The following diagram shows how AtoN equipment can be connected using a commercially provided IoT platform and the security that would need to be in place.

ADD NEW PICTURE



IoT device (AtoN) Publisher

Internet

IoT service (Azure broker)

Internet

IoT application Subscriber

1. Topology of a 3rd party IoT solution

### An organisational developed platform.

If considering the development of an organisational IoT solution, then there are several infrastructure aspects that may need to be provided and these are shown in the diagram below.



1. Topology of an organisation IoT platform

This example can be considered as a hybrid arrangement which shows where there is a number of different protocol employed. In this case the Message Queuing Telemetry Transport (MQTT) protocol is integrated into the broker, but also into the back end server to allow data exchange to the broker.

Peter S to develop another example of topology. With AtoN example and VPN secure example.

## IoT Communication Protocol

EXPLAIN WHY YOU CHOOSE ONE OVER ANOTHER.

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:

MQTT: A lightweight publish/subscribe protocol designed for machine-to-machine (M2M) communication.

CoAP: A protocol designed for resource-constrained devices in IoT networks.

DDS: A real-time data sharing protocol for industrial control and monitoring systems.

AMQP: An open standard for message queuing and data transfer in IoT systems.

OPC UA: A vendor-independent communication protocol for industrial automation and control systems.

LwM2M: A protocol designed for managing and updating device firmware and configuration over the air (OTA).

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

1. Comparison of various IoT protocols

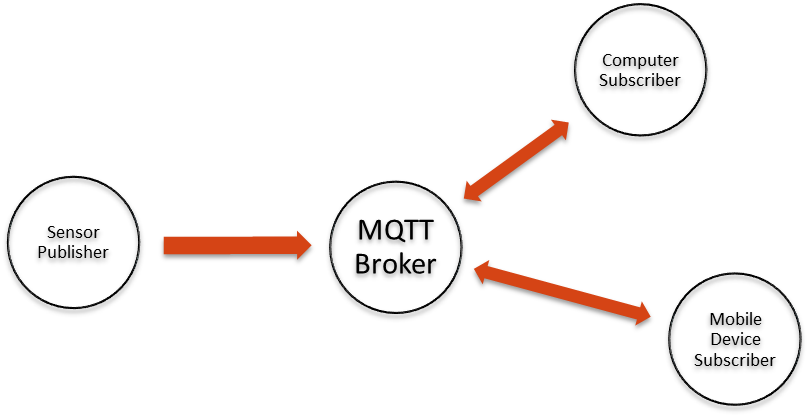
Add table from Jonas

Given the sample of various IoT protocols given in the table above, 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 detailed above and considering the demands for the Visual AtoN IoT harmonization, the MQTT protocol has been selected for this implementation This protocol is optimal for small low-power applications and is one of the most commonly used communication protocols. The implementation of the MQTT broker is straightforward to establish and integrates well with an already established remote monitoring system.

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

1. Clients: MQTT clients can be either publishers or subscribers. Publishers are responsible for sending messages to the broker, while subscribers are interested in receiving those messages.
2. Broker: The MQTT broker acts as a organizer or coordinator between publishers and subscribers. It receives messages from publishers and then distributes them to subscribers that have expressed an interest in those messages.
3. Topics: Messages are sent and received on topics, which act as channels for communication. 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 Quality of Service (QoS) for message delivery:
   1. QoS 0, just sends the data without acknowledgement. provides at most a single delivery;
   2. QoS 1, provides at least a single delivery with acknowledgement of receipt.
   3. QoS 2, ensures that exactly a single delivery is received with acknowledgement on 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. Clients send periodic ping request (PINGREQ) messages to the broker to indicate that they are still connected.
6. Last Will and Testament (LWT): MQTT supports a Last Will and Testament (LWT) feature that allows a publishing client to specify a message that will be published by the broker in the event that the client becomes disconnected unexpectedly.



Publish “23°C”

Publish “23°C”

Topic “temp”

Subscribe to ”temp”

Publish “23°C”

Subscribe to ”temp”

1. Example of publishing and subscribing

# Security

General guidance on Cyber security needs to be considered at the design stage and guidance can be obtained from the guideline ??? on Cyber Security Specifics in IALA Domains.

## Encryption and Authentication

With MQTT, it is a protocol that runs over TCP/IP and does not provide security measures itself, but it can be implemented with TLS and other security measures 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 one of the communication channels utilises the public internet, to avoid unencrypted data being transmitted.

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

For small systems with only a few publishing clients, the implementation of client side certification would be onerous. For such situations MQTT support the use of user names and passwords for each client. Additional security measures can include access control list (ACL) and IP whitelisting

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

Firewalls

Peter S to provide this intersessionaly.

When considering the levels of security to apply, it should be reflective of the importance of the data. Additionally such security requires additional processing power, which can impact on type, cost and power of the lantern, as well as the potential duration of the communication sessions, which again can impact on cost and power.

Only 8883 and 443 Inbound Ports Open

TLS Security

On Premise or Cloud Network

Initiates Outbound TLS Connection

Edge Network

MQTT Edge Client Device



F  
i  
r  
e  
w  
a  
l  
l

Access Control Lists   
(ACLs)

Username / Password Authentication

No Open Inbound Ports

F  
i  
r  
e  
w  
a  
l  
l

MQTT Server

1. Example of MQTT Security

# 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.

## Message structure.

When considering the information to be captured, stored and sent, the structure of such messages can be arranged 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 data small, can’t easily be read and understood. Alternatively, a much larger and easily to read structure could be achieve by the use of XML. Again these choices impact the amount of data sent and the power and communication cost to achieve this. 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.

A screenshot of a computer code

Description automatically generated

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

## Payload

The payload can be considered as all the dynamic information that is to be sent.

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”. A more typical topic for a visual AtoN might be battery voltage, position, etc.

These topics are then populated by values or parameters. These 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 for aspect associated with visual AtoN.

{

    "sessionId": "session-1",

    "status": "ready",

    "uptime": 20,

    "time": 1673564596

    "beaconStatus": "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

}

Possible key point to include from DTEC1 Liaison note at ENG17

• There would benefit in preparing a comparison table that looks at: the IoT typical use case; size of message; latency; frequency; availability of network; existing standards; IP (or royalty free); efficiency of protocol; reliability of information throughput.

• When looking to implement IoT in the IALA perspective, the availability and reliability of information were discussed, noting possible liability issues.

• Guiding principles on implementing IoT could include: efficient; affordable; scalable; with a focus on harmonised sharing of data.

| Title | Description | Expected outcome | Committee (\*leading) |
| --- | --- | --- | --- |
| S1010, 1.1  Guidance on the use of simple IOT sensors on physical aids | Establish requirement for IOT sensors. | Guideline | ARM |
| S1060, 6.2  Define user requirements for Maritime Connectivity, Maritime Internet of Things (IoT), and MRN addressing (may be three subtasks) | Revised Guideline G1143 to include aspects relevant to MRN | Revised guideline | ARM |

# Example of HEADING 1 STYLE

This guideline template should be used in conjunction with the *IALA Style Guide*. Utilising the styles provided in the **Styles Gallery** is key to using the document templates. Selecting the appropriate style from the Style Gallery will apply most text (and often layout) formatting required to comply with the *IALA Style Guide.* There should be no need to apply font formatting, numbering or bullets by selecting options from the **Font** or **Paragraph** dialog boxes.

The main text within a document is written in the **Body text** style, which is Calibri and 11 font size. Section titles can be inserted for up to four levels of text and should be created using the **Heading 1**, **Heading 2** etc. styles. Ensure the correct heading styles are selected as there are similar heading styles for the annexe and appendix entries, respectively. The blue colour used in the section headings and table texts is Red Green Blue (RGB) R0, G85, B140. The styles are referenced throughout this template and highlighted in bold.

**Heading 1 separation line style** follows the first carriage return after the first level heading title, and the style **Body text** follows the second carriage return after the separation line (if the line disappears, reposition the cursor at the end of the section heading text and press carriage return).

## Example of Heading 2 style

**Heading 2 separation line** style follows the second carriage return after the second level heading title, and the style **Body text** follows the second carriage return after the separation line.

### Example of Heading 3 style

**Body Text** style follows the first carriage return after the third level heading title; there is no separation line at this level.

#### Figures – Heading 4 style

**Body Text** style follows the first carriage return after the fourth level heading title; there is no separation line at this level.

Footnotes should be used sparingly but can be inserted and are found in **Footnote Reference** style at the bottom of the page[[1]](#footnote-1).



1. Example of wrapping in line with text

Figures should be centred with wrapping **In Line with Text** and labelled by writing the figure titles using the **Figure caption** style below the figure. It is important to note that figures and tables should be labelled in this manner with their respective styles to ensure that the tables in the contents section are updated correctly.

##### Alternative figure layout – Heading 5 style

Alternatively, figures can be offset with **Square** text wrapping so that the text does not overlap the figure but arranges the paragraph such that it continues onto the next line in an appropriately sized paragraph.

If no figures are included in the guideline, the respective table on the contents page should be deleted.

1. Example of wrapped square

# SECTION 2 – HEading 1 style

Sections should be typed continuously, and generally page breaks or section breaks should not be entered between main sections. It may be necessary sometimes to insert a page break to allow for aesthetic layout e.g., not breaking a list over two pages.

## Section 2.1 – Heading 2 style

Tables should be centred on the page. The table label should be created using the **Table caption** style and the caption should be positioned above the table.

Table text should be **Table heading** style for the column or row headings and **Table text** style for the content. The style **Table inset list** can be used for bulleted content within a table. The default table layout is for left justified and vertically centred table text but this can be amended using the table Layout menu to suit the content.

1. Example of table with row headers

| Table heading | Table text |
| --- | --- |
| Table heading | Table text |
| Table heading | Table text |

Space below the table should be maintained or inserted as necessary for clarity.

1. Example of table with column headers

| Table heading | Table heading | Table heading | Table heading |
| --- | --- | --- | --- |
| Table text | Table text | Table text | Table text |
| Table text | Table text | Table text | Table text |
| Table text | * Table inset list * Table inset list * Table inset list | Table text | Table text |

If no tables are included in the guideline, the respective table on the contents page should be deleted.

# Section 3 – Heading 1 Style

## Section 3.1 – Heading 2 style

The choice of numbered or bullet point lists depends on the context and content of the text and further guidance is given in the *IALA Style Guide*. Bullets are preferred unless it is important that the list is numbered e.g., for future reference or for a sequence.

Three levels of list styles are provided and these styles should be used rather than the default Microsoft Word numbering lists:

1. List 1 style example

**List 1 text** style example

* 1. **List a** style example

**List a text** style example

* + 1. **List i** style example

**List i text** style example

Each list style has a corresponding list text style that can be used for example, if the list requires more than one paragraph and the subsequent text needs to be aligned. If more than one list is used throughout the document it may be necessary to right click and select **Restart at 1** for subsequent lists.

## Section 3.2 – Heading 2 style

There are three levels of bullet point styles available:

* **Bullet 1** style

**Bullet 1 text** style

* **Bullet 2** style

**Bullet 2** **text** style

* **Bullet 3** style

**Bullet 3 text** style

Each bullet style has a corresponding bullet text style that can be used for example, if the bullet requires more than one paragraph and the subsequent text needs to be aligned.

## Section 3.3 – Heading 2 style

### Equations

#### Layout

If equations are included in the main body of the text, they should be explicitly referred to in the running text and centred on the page. Equations should be numbered consecutively with a right justified number in brackets e.g. (1) on the same line as the equation.

Any explanatory terms should be indented immediately below the equation starting with the non-capitalised term “where” and each term punctuated with a semi-colon until the penultimate term which should also include a semi-colon and the non-capitalised word “and”. For example:

The modified impulse response function is expressed by Equation (1):

where

*;*

*;*

*; and*

*.*

#### Numbering

The preferred method for including equations in the template documents is the Microsoft Word Equation Editor found in the **Insert** menu. The preferred layout and number reference described above can be achieved by typing #(x) where x is the number required immediately after the equation.



1. Example of how to achieve right justified equation number

For example, typing the formula followed by #(2) (as shown in Figure 3) and then pressing return will result in the following equation centred on the page and number being displayed on the same line to the right:

Note that equations do not automatically renumber using this method. If another equation is inserted between two existing equations the number must be adjusted manually.

Although the Microsoft Word Equation Editor is the preferred way of inserting equations, sometimes it is necessary to insert equations created elsewhere and copy those into the document as pictures. In the example below, the equation is included as a picture, wrapped **In Front of Text**:

“The area of a circle is shown in equation (3):

The picture can be copied directly into the document. To insert the corresponding number manually select **Equation number** paragraph style. The author may need to manually adjust the picture position to ensure it is centred and level with the number, achieving consistency with the automatically generated Microsoft Word Equation Editor layout described above.

# 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.

# abbreviations

This section should be typed with the **Abbreviations** style. The acronym or initialism is typed and then tab is pressed so that the style inserts the appropriate tabs and paragraph spacings e.g.:

NGO Non-governmental organization

VTS Vessel Traffic Services

The list should be typed in alphabetical order. The text automatically aligns as an indented paragraph until carriage return is hit and then the next term can be entered.

# references

References are sources directly referred to in the running text and should be given a sequential number, starting at 1. The reference number should be included as close to the referenced text as possible and included as a number within square brackets.

The reference should be listed in the References section in the following syntax using the **Reference** **list** style:

[Author surname,] <space> [initial.] <space> [year] <space> [title.]

For example:

“Hawking also suggests ways that quantum mechanics can be combined with the theory of special relativity [1]. This text builds on his discussion of the instability of black holes described in *A Brief History of Time* [2].”

should be included in the reference list as follows:

1. Hawking, S. (2001) The Universe in a Nutshell.
2. Hawking, S. (1988) A Brief History of Time.

The **Reference list** style will add a number for the reference as soon as you start typing the text and the paragraph will automatically align with the first line of text. Press return to enter a new reference in the list.

# Further reading

Any texts that are recommended to the reader without direct reference in the text should be listed within this section using the same syntax as the reference list. Sources should be listed using the **Further reading** style.

1. Einstein, A. (1905) Relativity: The Special and General Theory of Relativity
2. Idle, E. (1984) The Galaxy Song

# Index

**No index entries found.**

1. Visual AtoN MQTT Protocol Structure
   1. Logical architecture of MQTT enabled device

To support different legacy 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 (fig 1). For example, in a simple architecture case, telemetry messages can have the following topic name “tele/device123”, where the first part signifies the telemetry message, and the second part is the device unique ID.

A black background with a black square

Description automatically generated

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 (fig 2). For example, in a complex architecture case, telemetry messages can have the following topic name: "tele/device123/lantern1", where the first part signifies the telemetry message, the second part is the unique device ID, and the third part is the sub-device name.

A black background with a black square

Description automatically generated

1. 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 internal devices. Depending on the implementation, an MQTT gateway can be an aggregated gateway, translating data between the server and devices, and users can see only a single logical device.

* 1. Server-side implementations

The protocol described in this document allows 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.

* + 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.

A black background with a black square

Description automatically generated with medium confidence

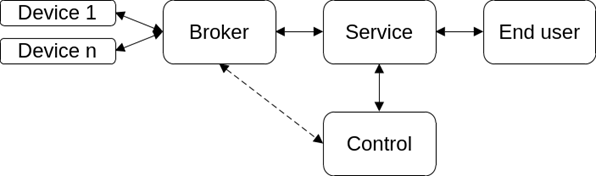
1. Minimal system

In this setup, both the device and 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 access list or private certificates. Without 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.

* + 1. Full implementation

In larger installations, it is recommended to use an additional server (service) that manages user roles and utilizes a database to store telemetry data and possibly providing some interface for users. It is not necessary for users to have specific programs to connect to the service; the service can use HTTP for user interactions. This setup requires additional access control to manage the server (service) and broker.



1. Typical system

In this setup, an additional server component is used between the end-user and the broker. This component contains a database and user control logic. 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 devices. The access control system manages the end user access to see or control devices.

* 1. Minimum compatibility and functional requirements

Devices that connect directly to the MQTT infrastructure should utilize a compliant 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 non-persistent mode, along with the following topics and properties:

1. Telemetry message: ‘tele’
   1. ‘sessionId’
   2. ‘status’
   3. ‘uptime’
2. Device information message: ‘info’
   1. ‘sessionId’
   2. ‘metadata’
   3. ‘protocolVersion’
   4. ‘sysInfo’
3. Command message: ‘cmd’
   1. ‘sessionId’
   2. ‘send’
4. Parameter message: ‘parameter’
   1. ‘sessionId’
   2. ‘brokerAddress’
   3. ‘mqttClientId’
   4. ‘mqttUser’
   5. ‘mqttPassword’
   6. ‘uid’
   7. 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 or 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. Sessions can be terminated by sending a 'done' session command from the server. Note that the 'done' command is only a hint, and the device is free to choose the time when the session is closed. In non-persistent mode, the session is interrupted when the server does not send any data during the receive wait period. The recommended receive wait period is 30 seconds. This value is device-specific and generally should be less than 300 seconds.

Every device should subscribe to at least the following topics:

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

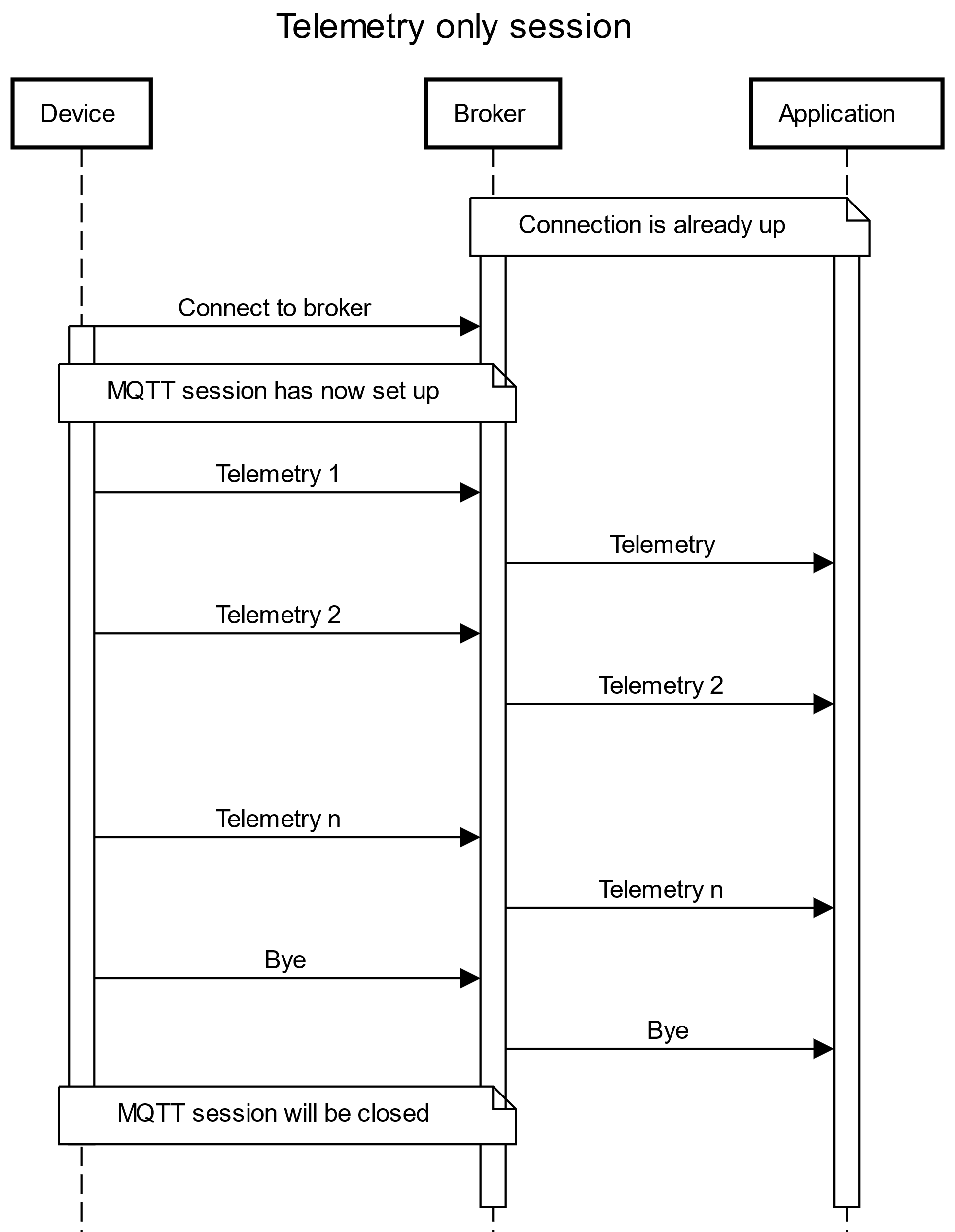
When a device expects to receive device-specific legacy commands, it should subscribe to the proprietary topic (with the topic name 'proprietary').

If a message has a response, it is not permissible to send a new message before receiving the response to the last message or until the message timeout occurs.

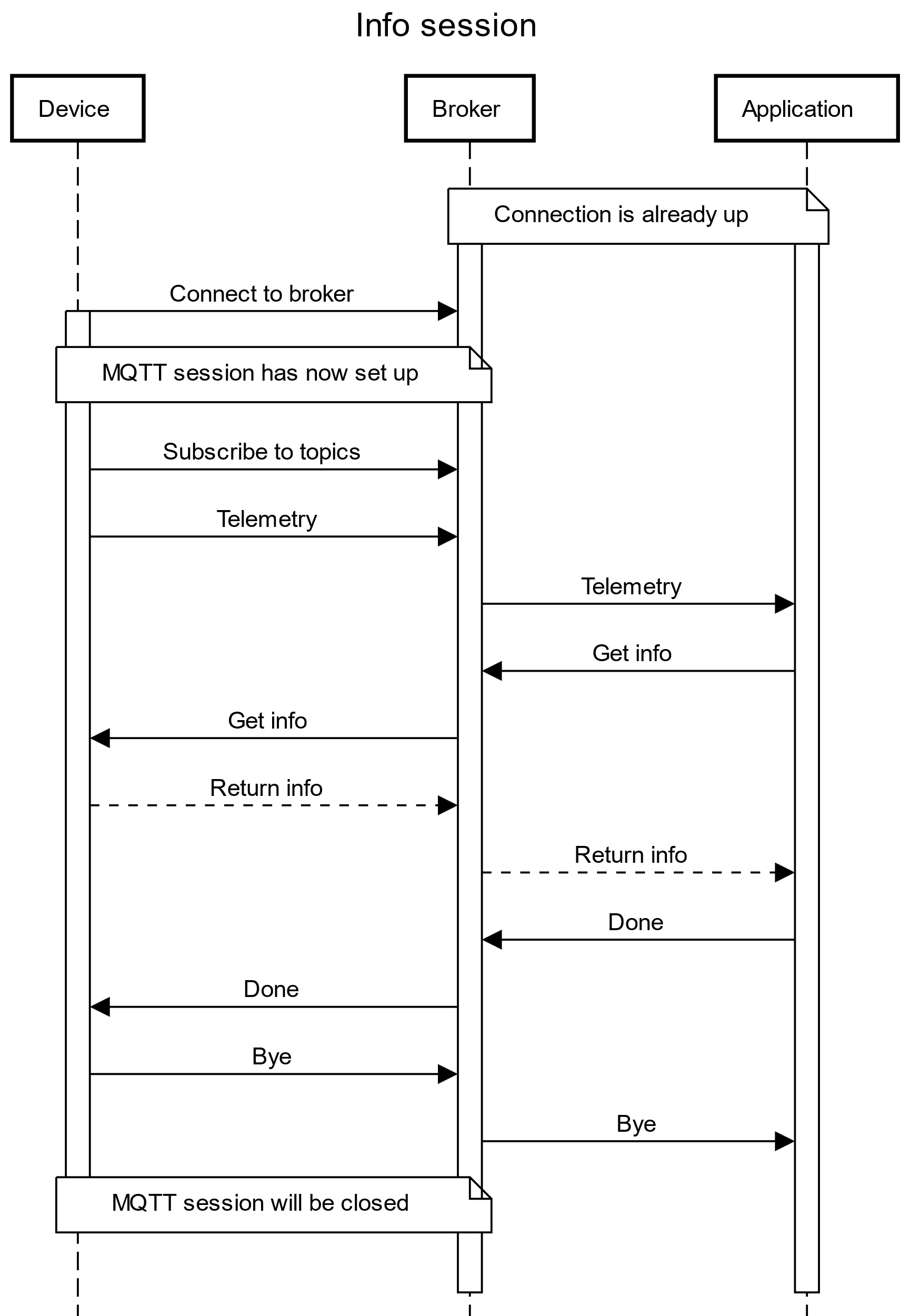
* + 1. Non-persistent session mode
       1. Telemetry information

Telemetry information is possible to transfer without additional data.

1. Device connects to broker and send telemetry packet. Depending on the configuration this may be repeated during the session.
2. Optionally, the device sends a ’bye’ message and disconnects from broker. The server should accept the connection closing without receiving a 'bye' message. The 'bye' message is solely for indicating that the device has gone offline and may not receive additional data.



1. Telemetry session
   * + 1. Info
2. The device connects to the broker and sends a telemetry packet.
3. Server ask info.
4. Device send data by using info topic.
5. Server send done. This means that server has no more commands queued.
6. Optionally, device send ’bye’ message and disconnects from broker. The server should accept the connection closing without receiving a 'bye' message. The 'bye' message is solely for indicating that the device has gone offline and may not receive additional data.
7. Device disconnects from broker.

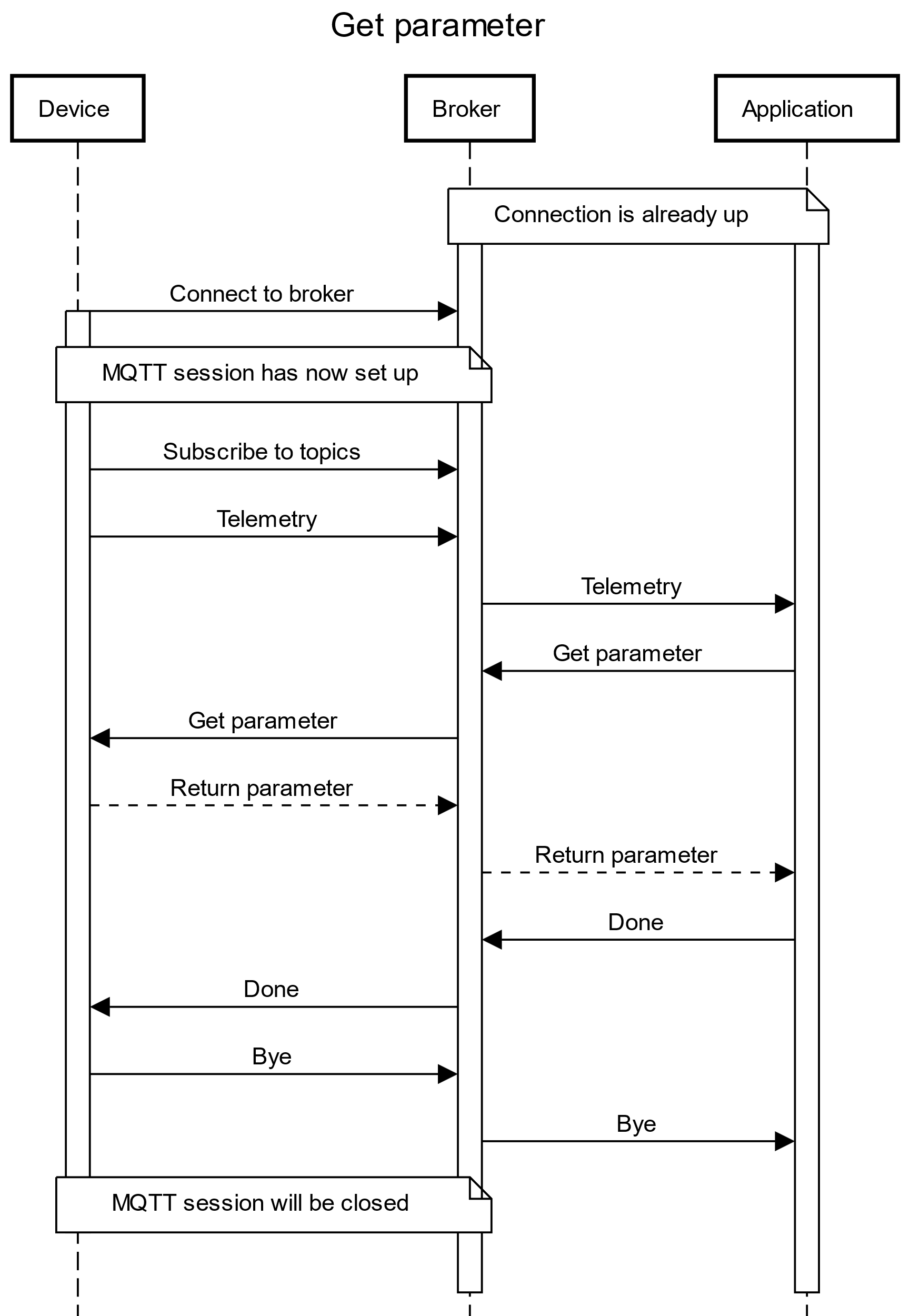


1. Info session
   * + 1. Get parameters

It is implementation-defined whether the device automatically sends configuration parameters after a change or only upon request. Automatically sent parameters are useful if a local user changes device parameters and the device sends the updated parameters to the server.

A typical sequence for retrieving parameters is as follows:

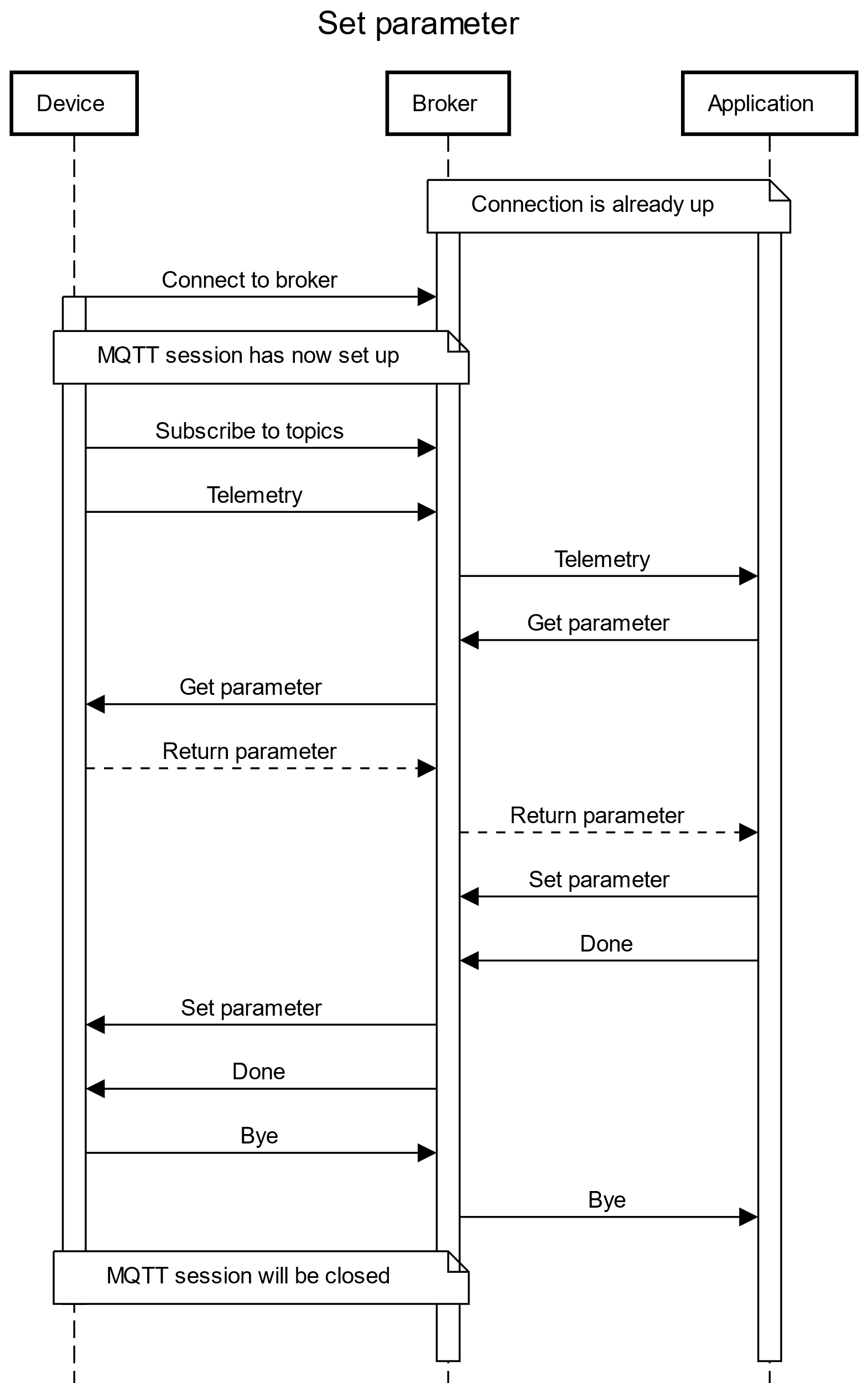
1. Device connects to broker and send telemetry packet.
2. Server ask parameter.
3. Device send parameter.
4. Server send done.
5. Device send optional ’bye’.
6. Device disconnects from broker.



1. Get parameter
   * + 1. Set parameters

Set parameters is used to change device configuration.

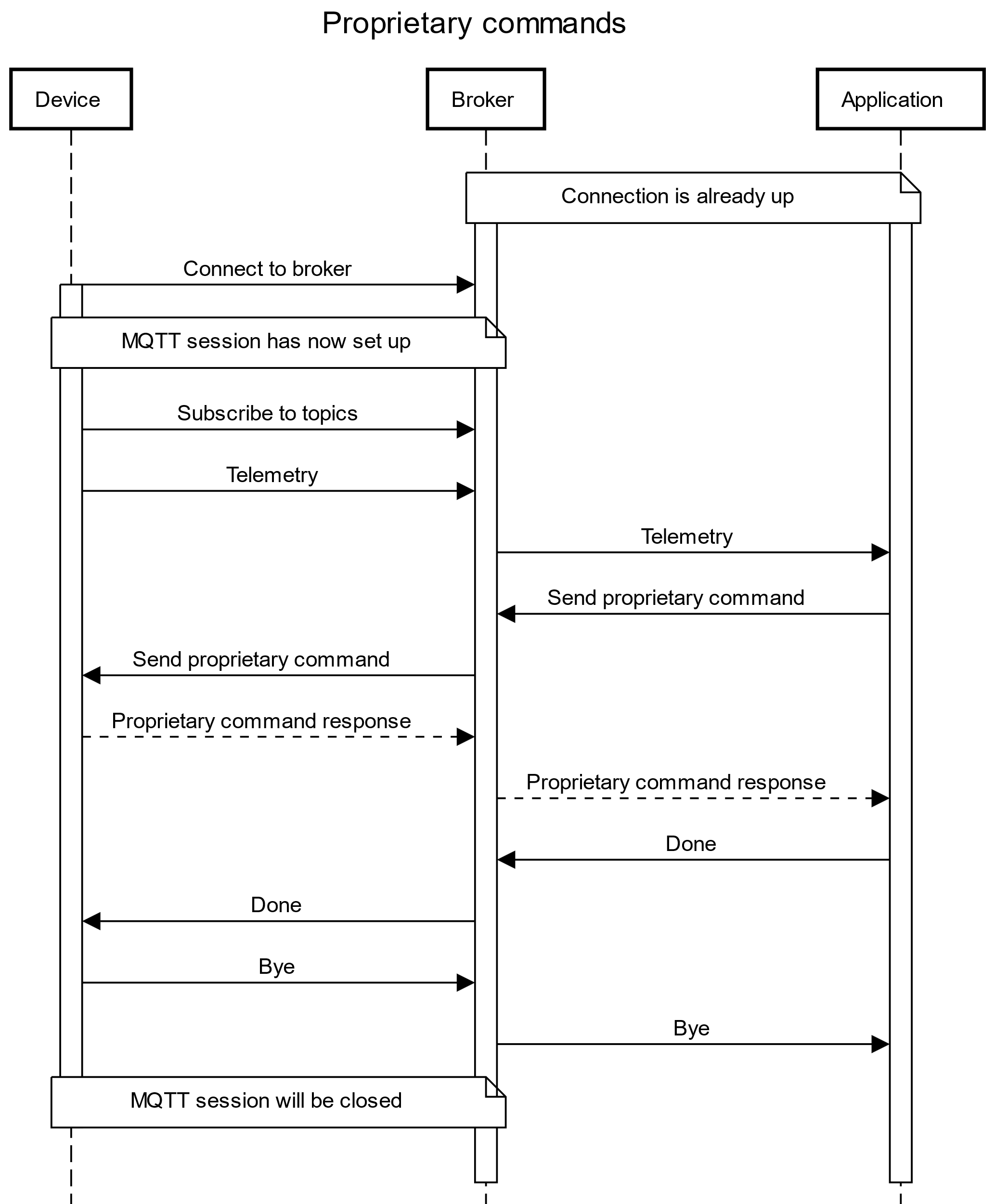
1. Device connects to broker and send telemetry packet.
2. Server ask parameter.
3. Device send parameter.
4. Server send updated parameter.
5. Server send done.
6. Device send optional ’bye’.
7. Device disconnects from broker.



1. Set parameter
   * + 1. Proprietary command

Set parameters is used to change device configuration.

1. Device connects to broker and send telemetry packet.
2. Server send proprietary command.
3. Device send response for proprietary command. Depending on command this may be acknowledgment or data response.
4. Server send done.
5. Device send optional ’bye’.
6. Device disconnects from broker.

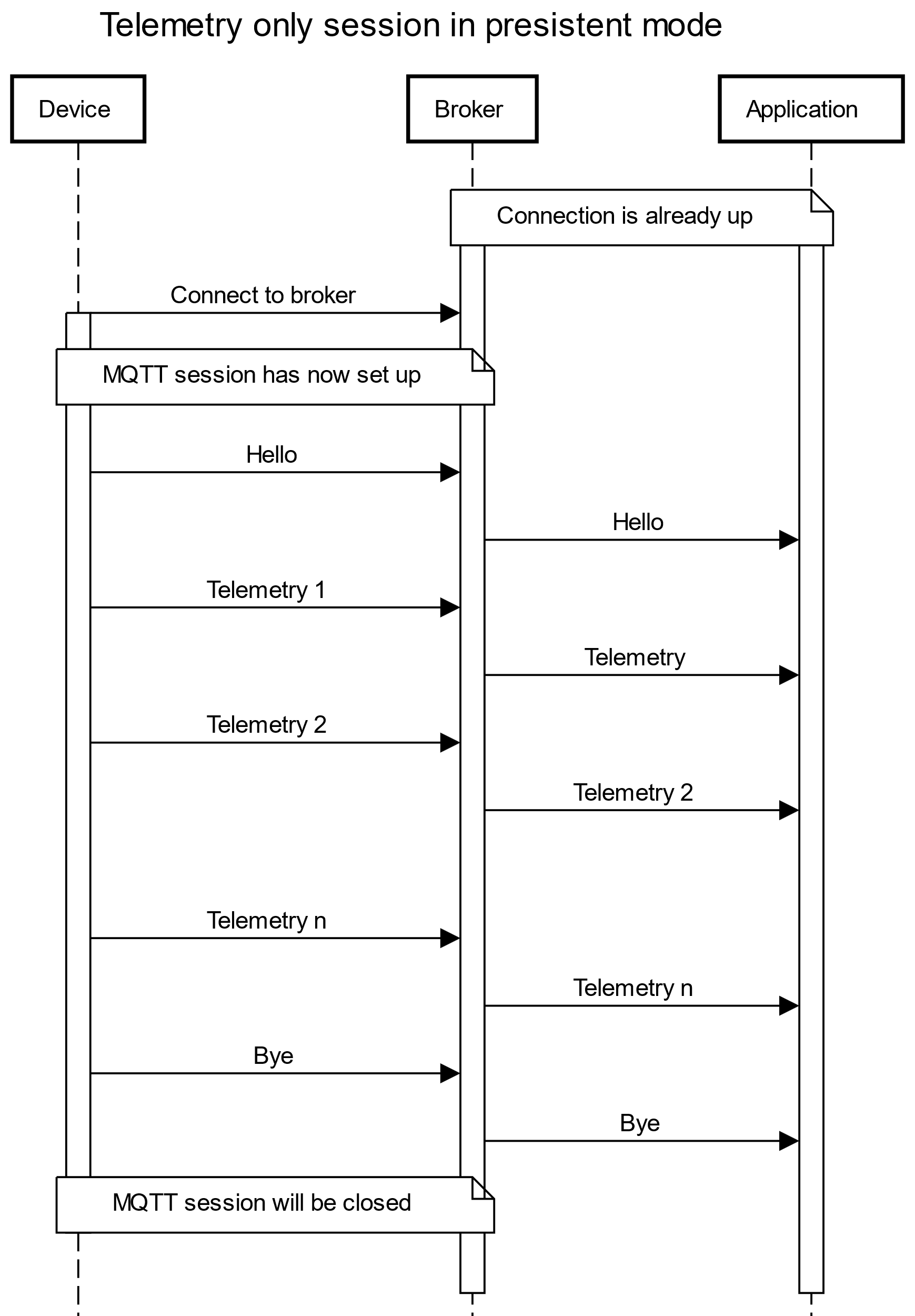


1. Proprietary session
   * 1. Persistent session mode

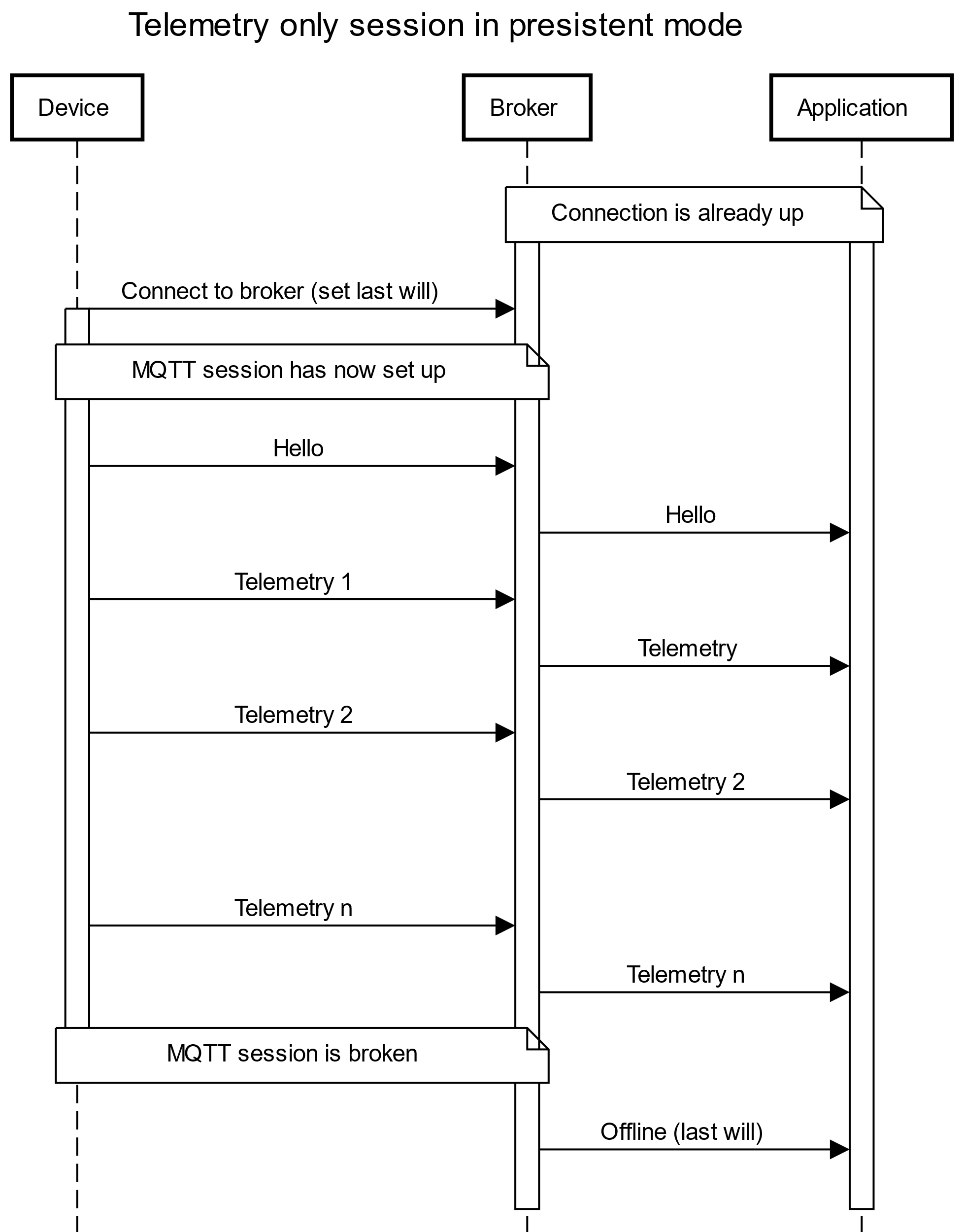
Persistent sessions differ from non-persistent ones only in terms of the beginning and end messages. Every persistent session must start with a 'hello' message and end with a 'bye' message, or in the case of link failure, an 'offline' message. This mode can be interrupted by sending a 'done' command.

* + - 1. Session with normal ending

1. Device connects to broker and send ’hello’ packet.
2. Device send telemetry packets. This is identical to non-persistent mode.
3. Device send required ’bye’.
4. Device disconnects from broker.



1. Persistent session
   * + 1. Failed session
2. Device connects to broker and send ’hello’ packet.
3. Device send telemetry packets. This is identical to non-persistent mode.
4. Link failure, broker detect timeout and send ’offline’ packet.



1. Failed persistent session
   * 1. Device registration and certificate renewal

Certificate renewal and device registration are optional processes. Depending on the capabilities of the device and the broker, there are two possible methods for device registration and certificate renewal:

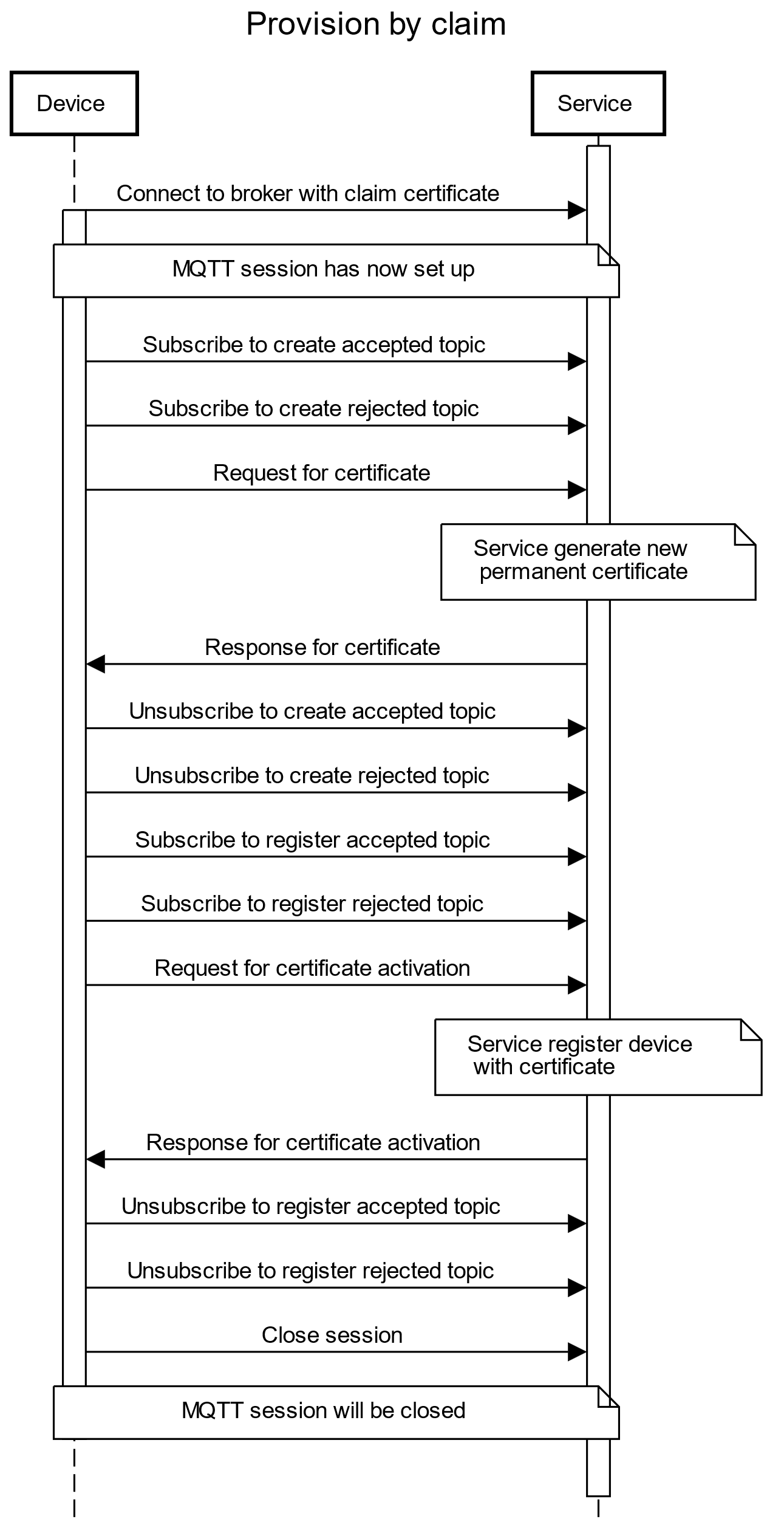
* 1. The device uses provisioning or already registered certificate for connection, and the server generates new certificate.
  2. An external device or service application is connected to the device, and the external application handles the registration process. After registration, a new certificate is loaded from the external application to the device.

If the device utilizes a TLS/SSL connection, it must support certificate renewal and be capable of using a third-party certificate.

* + - 1. Provisioning using a claim certificate

1. Device connects to the service using claim certificate and private key. Devices need to be manufactured with a provisioning claim certificate and private key.
2. Once connection established, device send request to create new permanent certificate.
3. Service generates a new certificate using and sends back following resources:
   1. New certificate.
   2. New private key.
   3. Ownership token, it is recommended to limit token lifetime up to 60 minutes.
   4. Checksum for certificate, private key and ownership token.
4. The device receives and stores the new certificate and private key, then acknowledges the service with the following information:
   1. Ownership token.
   2. Unique ID (7.8.9.10 – Unique ID).
5. Service receives a request to activate the certificate.
   1. If claim certificate was used, then validate claim certificate.
   2. Validate the device (using unique ID).
   3. Activate device certificate.
   4. Acknowledge new certificate.
6. Device receives acknowledgment and store device certificate.
7. Device disconnects current session and can start new session with new certificate.

After the first successful provisioning it may be necessary to send device info query. See section 9 – Info.



1. Device provisioning
   * + 1. Certificate renewal

Certificate renewal is similar to provisioning by claim, except that instead of connecting to the service with a claim certificate, the device uses its current certificate.

* 1. Topic Names

The Topic name must contain only characters (‘a’ – ’z’ and ‘A’ – ’Z’), numbers (0-9) and slash (’/’). Each topic contains at least two fields, command and unique ID. Unique ID is user defined and can be device serial number or site name. Preferred unique ID is site name.

Reserved topic name parts are:

* All first level topic names.
* All last level topic names.
* All topic names for certificate creation and provisioning.
  + 1. General structure of topic names

All topic names for messages that are sent from device to server or server to device have first level topic name and device unique identifier (1). To address local device, it is possible to add device name after unique identifier (2). Response messages have suffix res (3, 4). It is possible to use random ID in response topics (5), this is useful when it is needed to distinguish between different responses. Topic name may have suffix auto (6), this is used when telemetry data is something else than a regular telemetry packet (7.4 – Topic ’tele’ – status information), for example proprietary packet with telemetry data (7.9 – Topic ’proprietary’).

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

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

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

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

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

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

1. <first level topic name>/<uid>/<dev>/res – topic name for sub device response message
2. <random ID>/res – response topic with random ID

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

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

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

* + 1. First level topic names

First level topic name describes topic purpose, e.g. telematics topic, information topic, etc. First level topic name may have upper case suffix:

* ’B’ – binary payload, reserved for future use.
* ’S’ – encrypted payload, reserved for future use.
* no suffix – JSON payload.

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 user.
* ’parameter’ – get or set parameter(s).
* ’proprietary’ – device specific data that can be used for implementing custom protocols inside device, like Modbus.
  + 1. Second to n-1 level topic names

Second level topic name contain unique ID, it is also known as ‘thing’ name. It depends on the actual device configuration how many levels it has. For example, second level topic name can be one level *’lighthouseA’* or multilevel *’country/region/subregion/lighthouse’*.

* + - 1. Before last level topic name, n-1 level topic names

Topic names after second level topic name are optional and contains device name or identifier. This level is used only when 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.

* + 1. Last level topic name

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

* ‘auto’ – topics that are sent automatically, tele and proprietary topics, may have additional level named ’auto’. This level is used only when automatic message publish topic is same as subscribed topic name.
* ‘res’ – user for response message to query. When topic have response, then additional level named ’res’ will be added to the end. As response topic is issued from server, then it is possible to use any unique identifier before ’res’ part. But the format where are topic name and site is recommended.
  + 1. Topic names for provisioning

Creating a certificate and register a device have special topics. Device or product type is needed for device registration.

* ’certificates/create’ – create new certificates for SSL/TLS.
* ’certificates/create/accepted’ – topic for certificate acceptance response.
* ’certificates/create/rejected’ – topic for certificate rejection response.
* ’provision/<device type>’ – topic for register new device.
* ’provision/<device type>/accepted’ – topic for acceptance response.
* ’provision/<device type>/rejected’ – topic for rejection response.
  1. QoS values and retained messages

MQTT QoS (quality of service) defines level of delivery guarantee for message. QoS value 0 does not have any guarantee that message is received, value 1 means that acknowledgment that message is delivered is sent and value 3 means that both message and acknowledgment are delivered. QoS level 1 and 2 retransmit messages when no acknowledgment is received and are not suitable in cases when real-time data is transmitted.

Retained Messages is an MQTT feature that allows storing the “last known good” message for a particular topic on the broker and delivering the message to a client whenever the client subscribes to a matching topic. The best use case for retained messages are small systems that have few devices, broker and basic monitoring software that does not have any database, only real-time monitoring is allowed. It is possible to store last known status to broker and monitoring application can load last state from broker. For that case is needed that least session and tele topics have retention enabled. Parameter retention is used when device is most of the time off-line and monitoring server is not able to send new parameter during communication session.

Recommended minimum QoS levels and retained values are:

1. Table 2 Example of 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 |
| certificate and provision | 1 | No |

* 1. Messages

The payload utilizes JSON formatting. 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, and it is permitted to use null values. Arrays and JSON objects may contain either string or integer values.

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

* + 1. Rules for all messages

1. Payload is in JSON format [1, 2]
2. Ony ASCII characters are allowed in property names.
3. 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. ’.’ (dot) is used as decimal separator in floating-point numbers.
7. SI and derived units shall be used.
   1. Time units are in seconds.
   2. Distance or length is meters.
   3. Temperature in kelvin.
   4. Voltage in volts.
   5. Current in amperes.
8. All timestamps must be in Coordinated Universal Time (UTC).
9. Position coordinates should use the WGS 84 geographic coordinate system.
10. Not recommended property name is ”class”.
11. User defined properties are allowed but must follow payload data rules.
12. All optional properties can be omitted or have null value when data is invalid. Non-optional properties should have default value in case of invalid data.
13. All JSON messages from device should contain session ID information.
    * 1. General message

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

A black background with a black square

Description automatically generated with medium confidence

1. General message

TODO: decide which version to use, if type above is used then change message properties as well.

General message is a supertype for all messages and all messages must implement fields in general message. Only exception is certificate handling and provisioning, which are handled separately from other messages.

A black background with a black square

Description automatically generated with medium confidence

1. General message
   * + 1. Property session ID

|  |  |
| --- | --- |
| Property name: | ’sessionId’ |
| Description: | Session ID property is string which holds value, that is unique during MQTT session. This value can be monotonic counter which is started with MQTT session or timer. |
| Required: | Only on messages that sent from device to server |
| Example: | "sessionId": "session-820923084792" |

* + - 1. Time

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

* + - 1. Property response topic

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

* + - 1. Example

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

* + 1. Topic ’session’ – session information

Session topic is intended for notification session start and end. This topic is used on systems which use long-lived sessions. This topic is used in ’last will’ message also. It is assumed that device is offline when empty JSON payload is sent to server.

Device send following message in the beginning of persistent session:

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

Device send following message in the end of persistent session:

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

* + - 1. Done

|  |  |
| --- | --- |
| Property name: | ’done’ |
| Description: | Any string or null. This is hint from server, that server has been completed all tasks and device if free to disconnect from broker. It depends on client configuration if it disconnects immediately or send telematics packets. If controller send packets after ’done’ message, then this discard current active ’done’ hint. If server does not send ’done’ message, then controller can disconnect if last message from server was more than 60 seconds ago. This command should close MQTT session as well. |
| Required: | Optional |
| Example: | "done": "ok" |

* + - 1. State

|  |  |
| --- | --- |
| Property name: | ’state’ |
| Description: | String that describes session state.   * ‘hello’ – this value is sent in the beginning of persistent session. * ‘bye’ – this value is sent in the end of persistent session. If device supports, then it is reccomendad that ‘bye’ message have ‘time’ property. * ‘offline’ – this value is sent by the will message. An alternative is to use empty JSON payload. |
| Required: | Only on persistent sessions |
| Example: | "state": "hello" |

* + 1. Topic ’tele’ – status information

Minimal telematics JSON message for generic device:

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

* + - 1. Time info
         1. Uptime

|  |  |
| --- | --- |
| Property name: | ’uptime’ |
| Description: | Unsigned integer. Server should expect to receive at least 32-bit unsigned value. Seconds from last boot. Default value is 0. |
| Required: | Yes |
| Example: | "uptime": 211 |

* + - 1. Status info
         1. Status

|  |  |
| --- | --- |
| Property name: | ’status’ |
| Description: | This string property hold device status. Allowed values are: ’init’, ’ready’, ’alert’ and ’suspend’   * ’init’ – this status is set until all required components are not yet fully operational, e.g. GNSS is not acquired initial coordinates after booting. It is recommended that ’init’ state duration is limited by timeout. If ’init’ is not completed after timeout, then next state is alert. This is default value after booting. * ’ready’ – system is fully functional and does not have any active errors, e.g. lantern is switched on * ’alert’ – alarm condition detected, e.g. low battery * ’suspend’ – when system is switched off, but it is functional, e.g. storage state. This field is not required on devices which does not have suspend state. |
| Required: | Yes |
| Example: | "status": "ready" |

* + - * 1. Alert status

|  |  |
| --- | --- |
| Property name: | ’alert’ |
| Description: | This array of string holds list of alert statuses. If device operates normally and don’t have any alert status, then this property may be omitted or set value to ’none’. Any user defined values are allowed, but following status names are reserved:   * ’none’ – default value * ’lowBattery’ – for low battery alarm. Low battery level is configured by 7.8.8 – Low battery voltage level. * ’batteryWarning’ – for low battery warning. Battery warning level is cofigured by 7.8.7 – Low battery voltage warning level. * ’gnssError’ – any GNSS related errors * ’offLocation’ – if distance from expected site is more than configured * ’lampFail’ – only for beacons, if lantern does not operate as expected * ‘impactDetected’ – for collision detection * ‘lanternPowerWarning’ – for consuming too much power * ‘maintenance’ – for maintenance visit 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": ["lowBattery", "offLocation"] |

* + - * 1. Beacon status

|  |  |
| --- | --- |
| Property name: | ’beaconStatus’ |
| Description: | Status string. Shows beacon current status. Allowed values are:   * ’onMainCharacter’ – main or night character * ’onAlternativeCharacter’ – alternative or day character * ’off’ – not flashing, default value |
| Required: | Mandatory only on beacons. ’type’ field in info message should be ’beacon’. |
| Example: | "beaconStatus": "onMainCharacter" |

* + - * 1. Light sensor status

|  |  |
| --- | --- |
| Property name: | ’lightSensorStatus’ |
| Description: | Status string. Shows beacon light sensor status. Allowed values are:   * ’day’ – day mode * ’night’ – night mode * ’forceDay’ – always day mode, light sensor value is not used * ’forceNight’ – always night mode, light sensor value is not used |
| Required: | Optional and only on beacons. ’type’ field in info message should be ’beacon’. |
| Example: | "lightSensorStatus": "day" |

* + - 1. Environmental data
         1. Device temperature

|  |  |
| --- | --- |
| Property name: | ’temperature’ |
| Description: | Temperature in kelvins. JSON object, unordered name/value pairs, where value is floating-point value. This property contains 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 sensor. |
| Example: | "temperature":  {    "last": 294.15,    "max": 298.15,    "min": 292.15,    "avg": 295.15  } |

* + - * 1. Ambient light level measured by the photocell

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

* + - 1. Power and supply voltage data
         1. Supply voltage

|  |  |
| --- | --- |
| Property name: | ’voltage’ |
| Description: | Supply voltage in volts. JSON object, unordered name/value pairs, where value is positive floating-point value. This property contains following sub properties:   * ’avg’ – last battery voltage, this value is required. On beacons it is recommended to use average value over one flash cycle. * ’loaded’ – last measured battery voltage under load condition, optional. * ’unloaded’ – last measured battery voltage under no-load condition, optional. * ’max’ – maximum battery voltage under no-load condition in last 24 hour, optional. * ’min’ – minimum battery voltage under load condition in last 24 hours, optional. * ’aux’ – secondary battery voltage, optional. |
| Required: | Optional, only on devices which can measure supply voltage. |
| Example: | "voltage":  {    "avg": 12.3,    "loaded": 12.1,    "unloaded": 12.5,    "max":13.3,    "min":11.5  } |

* + - * 1. Beacon power

|  |  |
| --- | --- |
| Property name: | ’beaconPower’ |
| Description: | Beacon power consumption during flash cycle, measured in watts (W). Positive floating-point number. |
| Required: | Optional, only on devices which can measure beacon power consumption during flash cycle. |
| Example: | "beaconPower": 5.3 |

* + - * 1. Accumulated solar charge during last 24 hours

|  |  |
| --- | --- |
| Property name: | ’solarCharge’ |
| Description: | Solar charge value in watt-hours (Wh). The value is a floating-point number. Positive and negative numbers are allowed. A positive number indicates accumulated charge, while a negative number indicates energy consumed from the battery. The indication of consumed energy is optional. |
| Required: | Optional, only on devices which can measure solar charge voltage. |
| Example: | "solarCharge": 10.1 |

* + - 1. Beacon data
         1. Illumination counter

|  |  |
| --- | --- |
| Property name: | ’illuminationCounter’ |
| Description: | Counts time in seconds when the beacon is in the active state. Can be reset with the 'reset statistics' command." |
| Required: | Optional. |
| Example: | "illuminationCounter": 16695 |

* + - * 1. Astronomical dusk and down

TODO: Addition information can be found <https://science.howstuffworks.com/nature/climate-weather/atmospheric/twilight-dusk.html>

* + - 1. Position and GNSS data

|  |  |
| --- | --- |
| Property name: | ’position’ |
| Description: | Position property. JSON object, parameters described separately. This is available only on devices which have GNSS receiver and use GNSS receiver for positioning. 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 |
| Example: | "position":  {    "latLon": [60.0, 110.0],    "time": 1673564596,    "deviation": 10.0,    "hdop": 1.1  } |

* + - * 1. Location

|  |  |
| --- | --- |
| Property name: | ’latLon’ |
| Description: | JSON array of floating-point values with last GNSS position. Latitude have value between -90.0/90.0 degrees and longitude between -180.0/180.0 degrees. It is recommended to have least 7 decimal points. |
| Required: | Required on devices which have GNSS for positioning |
| Example: | "position":  {    "latLon": [60.0, 110.0],    "time": 1673564596  } |

* + - * 1. Last GNSS fix time

|  |  |
| --- | --- |
| Property name: | ’time’ |
| Description: | Seconds from January 1st, 1970, at UTC (UNIX time). Positive integer value. To avoid the year 2038 bug, it is recommended to use least 32-bit unsigned numbers in implementation. |
| Required: | Required on devices which have GNSS for positioning |
| Example: | "position":  {    "latLon": [60.0,110.0],   "time": 1673564596  } |

* + - * 1. Deviation

|  |  |
| --- | --- |
| Property name: | ’deviation’ |
| Description: | Floating-point value of deviation in meters from GNSS fixed position. Positive floating-point value. |
| Required: | Optional on devices which have GNSS for positioning |
| Example: | "position":  {    "latLon": [60.0,110.0],    "time": 1673564596,    "deviation": 10.0  } |

* + - * 1. GNSS HDOP indicator

|  |  |
| --- | --- |
| Property name: | ’hdop’ |
| Description: | GNSS HDOP value. Positive floating-point value. |
| Required: | Optional on devices which have GNSS for positioning |
| Example: | "position":  {    "latLon": [60.0,110.0],    "time": 1673564596,    "hdop": 1.1  } |

* + - 1. Network data

|  |  |
| --- | --- |
| Property name: | ’networkStatistics’ |
| Description: | Contains nested statistics information. JSON object, parameters described separately. |
| Required: | Optional. If present, must contain all sub properties (’succeededServerConnections’, ’failedServerConnections’, ’succeededNetworkLogins’ and ’failedNetworkLogins’) |
| Example: | "networkStatistics":  {    "succeededServerConnections": 2,    "failedServerConnections": 0,    "succeededNetworkLogins": 2,    "failedNetworkLogins":2  } |

* + - * 1. Amount of succeeded server connections

|  |  |
| --- | --- |
| Property name: | ’succeededServerConnections’ |
| Description: | Number of successful connections between server and broker. Positive integer, default value is 0. |
| Required: | Required only when ’networkStatistics’ is used. |
| Example: | "networkStatistics":  {    "succeededServerConnections": 2,    "failedServerConnections": 0,    "succeededNetworkLogins": 2,    "failedNetworkLogins": 2  } |

* + - * 1. Amount of failed connections

|  |  |
| --- | --- |
| Property name: | ’failedServerConnections’ |
| Description: | Number of failed connections between server and broker. Positive integer, default value is 0. |
| Required: | Required only when ’networkStatistics’ is used. |
| Example: | "networkStatistics":  {    "succeededServerConnections": 2,    "failedServerConnections": 0,    "succeededNetworkLogins": 2,    "failedNetworkLogins": 2  } |

* + - * 1. Amount of succeeded network logins

|  |  |
| --- | --- |
| Property name: | ’succeededNetworkLogins’ |
| Description: | Number of succeeded logins to network. Positive integer, default value is 0. |
| Required: | Required only when ’networkStatistics’ is used. |
| Example: | "networkStatistics":  {    "succeededServerConnections": 2,    "failedServerConnections": 0,    "succeededNetworkLogins": 2,    "failedNetworkLogins": 2  } |

* + - * 1. Amount of failed network logins

|  |  |
| --- | --- |
| Property name: | ’failedNetworkLogins’ |
| Description: | Number of failed logins to network. Positive integer, default value is 0. |
| Required: | Required only when ’networkStatistics’ is used. |
| Example: | "networkStatistics":  {    "succeededServerConnections": 2,    "failedServerConnections": 0,    "succeededNetworkLogins": 2,    "failedNetworkLogins": 2  } |

* + - 1. System data

System data is optional, but it is useful for problem solving. System data block can be transmitted once in first session after device restart.

* + - * 1. Last reset source

|  |  |
| --- | --- |
| Property name: | ’lastResetSource’ |
| Description: | String that contains last reset source. Allowed values are:   * ’por’ – power on reset * ’wdr’ – watchdog reset * ’rst’ – reset from external reset signal (HW 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" |

* + - * 1. Reset count

|  |  |
| --- | --- |
| Property name: | ’resetCount’ |
| Description: | Contains nested properties property names which are used in ’lastResetSource’. JSON object, unordered name/value pairs, where value is positive integer value. Default values are 0. |
| Required: | Optional, only when last reset source is present. |
| Example: | "resetCount":  {    "por": 30,    "wdr": 1  } |

* + - 1. Example

|  |
| --- |
| {    "sessionId": "session-1",    "time": 1673564596,    "status": "ready",    "uptime": 20,    "beaconStatus": "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,    "position":    {      "latLon": [60.0,110.0], "time": 1673564590,      "time": 1673564590,      "deviation": 22,      "hdop": 1.6    },    "networkStatistics":    {      "succeededServerConnections": 1,      "failedServerConnections": 2,      "succeededNetworkLogins": 0,      "failedNetworkLogins": 2    }  } |

* + 1. Topic ’info’

Information is triggered by info request command. Request topic is ’cmd/’ and message payload in JSON format is ’{"send": "info"}’.

* + - 1. Protocol version

|  |  |
| --- | --- |
| Property name: | ’protocolVersion’ |
| Description: | Integer value to describe protocol version. Currently supported value is 1. |
| Required: | Yes |
| Example: | "protocolVersion": 1 |

* + - 1. Metadata

|  |  |
| --- | --- |
| Property name: | ’metadata’ |
| Description: | Map of nested maps and strings with supported properties and limit values. This property contains only topics that can have data from device side which can be stored to database and does not contain commands. All values must match with parameters. Map key contains nested map name or properties name, and value contains nested map or value limits in interval notation. Limit value may contain limiting number values, list of allowed strings or empty string to allow all.   * The rectangular bracket symbols, "[" and "]", are used to describe 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 describe sets with a lower bound or upper bound, respectively. They correspond to the > and < symbols. Special string "inf" is used for infinity. * Value may contain comma separated list of allowed strings. * Value may contain map for nested parameters. |
| Required: | Yes |
| Example: | "metadata":  {  "tele":    {      "light-intensity": "[0, inf]",      "low-voltage-level": "[0,20)",      "time": "",      "last-reset-source": "wdr, por",      "network-statistics":      {        "succeeded-server-connections": "",        "failed-server-connections": "",        "succeeded-network-logins": "",        "failed-network-logins": ""      }    },    "parameter":    {      "mode": "photocell"    }  } |

* + - 1. System information

|  |  |
| --- | --- |
| Property name: | ’sysInfo’ |
| Description: | This information can be used to set up optimal set/get transmission packet sizes. JSON object, unordered name/value pairs, where value is 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  } |

* + - 1. Type

|  |  |
| --- | --- |
| Property name: | ’type’ |
| Description: | String, which contains device type class:   * ’group’ – logical container for device group, used for MQTT gateway * ’beacon’ – for beacons |
| Required: | Optional |
| Example: | "type": "beacon" |

* + - 1. Serial number

|  |  |
| --- | --- |
| Property name: | ’serialNr’ |
| Description: | String with device serial number. This string may contain product code also, if product code and serial number are not represented in this string then ’productCode’ property show product code. |
| Required: | Optional. Only on devices that have valid serial number. |
| Example: | "serialNr": "flasher1-001" |

* + - 1. Product code

|  |  |
| --- | --- |
| Property name: | ’productCode’ |
| Description: | String with device product code. |
| Required: | Optional. Required only when serial number does not have product information. |
| Example: | "productCode": "flasher1" |

* + - 1. Firmware version

|  |  |
| --- | --- |
| Property name: | ’firmwareVersion’ |
| Description: | String with firmware version. |
| Required: | Optional. Only on devices that have firmware version information. |
| Example: | "firmwareVersion": "version-1.0" |

* + - 1. Component info

|  |  |
| --- | --- |
| Property name: | ’componentInfo’ |
| Description: | Contains device component information. List of JSON objects, unordered name/value pairs, where values are strings. Allowed information parameters are:   * ’component’ – component description. * ’version’ – component version. * ’id’ – component ID or serial number. This parameter is optional and used in topics names to address sub-devices. |
| Required: | Optional. Only for devices or systems that have sub-components or sub-devicces. |
| Example: | "componentInfo":  [    {      "component": "gnss",      "version":"1.0"    },    {      "component": "flasher",      "version": "2.0",      "id": "flasher1"    }  ] |

* + 1. Example

|  |
| --- |
| {    "sessionId": "session-1",    "time": 1715169904,    "metadata":    {  "tele":      {        "light-intensity": "[0, inf]",        "low-voltage-level": "[0 ,20)",        "time": "",        "last-reset-source": "wdr, por",        "network-statistics":        {          "succeeded-server-connections": "",          "failed-server-connections": "",          "succeeded-network-logins" "",          "failed-network-logins": ""        }      },    "parameter":      {        "mode": "photocell"      }    },    "protocol-version": 1,    "type": "beacon",    "sys-info":    {      "rx-buf": 128,      "tx-buf": -1    },    "serial-nr": "flasher1-001",    "product-code": "flasher1",    "firmware-version": "1.0",    "component-info":    [      {        "component": "gnss",        "version": "1.0"      },      {        "component": "flasher",        "version": "2.0",        "id":"flasher1"      }    ]  } |

* + 1. Topic ’cmd’

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

* + - 1. Send

|  |  |
| --- | --- |
| Property name: | ’send’ |
| Description: | String with requested topic. Allowed values are:   * ’tele’ – for telematics * ’info’ – for information |
| Required: | Yes |
| Example: | "send": "info" |

* + - 1. Reconnect

|  |  |
| --- | --- |
| Property name: | ’reconnect’ |
| Description: | Disconnect and connect with broker. Any parameter value is accepted, but null is preferred. |
| Required: | Optional |
| Example: | "reconnect": null |

* + - 1. Reset

|  |  |
| --- | --- |
| Property name: | ’reset’ |
| Description: | String which describes device or parameter to reset. If reset device command is issued, then this command should close MQTT session as well. 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 |

* + - 1. Reload configuration

|  |  |
| --- | --- |
| Property name: | ’reloadConfig’ |
| Description: | Load configuration from configuration storage to device registers or RAM. This command is used after updating parameter value when device need additional steps to use new parameters. For example, load new GNSS acquisition period from configuration storage to GNSS handler. Allowed values are:   * null * all other parameters are reserved for future use |
| Required: | Optional |
| Example: | "reloadConfig": null |

* + - 1. Time

|  |  |
| --- | --- |
| Property name: | ’time’ |
| Description: | Set or get the time value in seconds, a complementary property for ‘time’ property in the ‘general’ message. Allowed values are:   * null – query from the server * any positive integer – sets a new time value |
| Required: | Optional |
| Example: | "time": 1673564596 |

* + - 1. Light on demand

|  |  |
| --- | --- |
| Property name: | ’lightOnDemand’ |
| Description: | Start light on demand, parameters describe how many seconds is light on demand mode is active. Device returns normal operation after this time is elapsed. This command allows to specify optional intensity for light on demand operation. Allowed values are:   * null – query from server, server send current status, timeout and intensity values are sent * sub-properties to describe on time and timeout   + ’timeout’ – integer, duration of light activation in seconds. If ’timeout’ property is missing, then light on demand will be switched off (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":  {    "timeout": 3600,    "intensity": 300  } |

* + - 1. Fix position

|  |  |
| --- | --- |
| Property name: | ’fixPosition’ |
| Description: | Any string or null. Start GNSS fix position updating. Allowed values are:  null – optional query from server, returns current status. Responses to query:  ‘started’ – position fix has already stated  ‘idle’ – position fix is not started or is already completed  number between 0 to 100 – precentage value of completed fix, can be used instead ‘started’  ‘start’ – start position fix |
| Required: | Optional |
| Example: | "fixPosition": "start" |

* + - 1. Examples

|  |
| --- |
| Command from server: start position fix and set time.  {    "time": 1715169904,    "fixPosition": "start"  }  Time query command from server:  {    "time": null    "resTopic": "630c8f35-148f-4961-8b56-96586b021ba0"  }  Response to time query:  {    "sessionId": "session-1",    "time": 1715169904  } |

* + 1. Topics ’parameter’

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

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": "photocell"    "lightIntensity": 34  } |

Set:

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

* + - 1. Automatic data

|  |  |
| --- | --- |
| Property name: | ’autoData’ |
| Description: | Set or get data which is sent as telemetry message on every connection. By default, telemetry packet is used. Reserved values:   * ’tele’ – telemetry packet is sent automatically, this is default. * ’proprietary’ – proprietary data packet is sent automatically. Data which is sent is device dependent.   It is allowed to use additional values by user. |
| Required: | Optional |
| Example: | "autoData": "proprietary" |

* + - 1. Beacon mode

|  |  |
| --- | --- |
| Property name: | ’mode’ |
| Description: | Set or get beacon operation mode. Allowed string values:   * null – query from server * ’photocell’ – flashing is controlled by photocell, this is default * ’alwaysNight’ – always flashing in night mode, it is same as photocell detected night (optional) * ’alwaysDay’ – always flashing in day mode, it is same as photocell has detected day (optional) * ’idle’ – idle mode, flashing is turned off, lowest power consumption |
| Required: | Optional |
| Example: | "mode": "photocell" |

* + - 1. Astronomical dusk and dawn

TODO: Addition information can be found <https://science.howstuffworks.com/nature/climate-weather/atmospheric/twilight-dusk.html>

* + - 1. Beacon flash character

|  |  |
| --- | --- |
| Property name: | ’flashCode’ |
| Description: | Common property for flash code. JSON object, see parameters descriptions below. 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]  } |

Main flash character

|  |  |
| --- | --- |
| Property name: | ’main’ |
| Description: | Set or get flash character. JSON array of integers pairs that describe flash character. Every odd element in array, describe flash duration in seconds. Every even element describes eclipse duration in seconds. Last eclipse character may be omitted to get fixed light. Allowed values:   * null – query from server * array of integers to describe flash character |
| Required: | Required for all flashers. |
| Example: | Flash code where flash is 500 ms and eclipse 500 ms:  "flashCode":  {  **"main": [0.5, 0.5]**  }  Flash code where eclipse is absent, light is always on i.e. fixed light.  "flashCode":  {  **"main": [1]**  } |

Secondary flash character

|  |  |
| --- | --- |
| Property name: | ’secondary’ |
| Description: | See description in 7.8.4.1 – Main flash character. |
| Required: | Required flashers that support secondary flash character. |
| Example: | "flashCode":  {    "secondary": [0.5, 0.5]  } |

* + - 1. Output intensity

|  |  |
| --- | --- |
| Property name: | ’intensity’ |
| Description: | Intensity values for flashers. Common property for flasher output intensity. JSON object, see parameters descriptions below. Allowed sub-properties are:   * null – query all from server. * defaultIntensity – intensity which is used during night. * alternativeIntensity – intensity which is used during day. |
| Required: | Optional |
| Example: | "intensity":  {    "default-intensity": 500,    "alternative-intensity": 1000  } |

Default output intensity

|  |  |
| --- | --- |
| Property name: | ’defaultIntensity’ |
| Description: | Set or get main light output intensity in candelas. Default intensity is mostly 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  } |

* + - * 1. Alternative output intensity

|  |  |
| --- | --- |
| Property name: | ’alternativeIntensity’ |
| Description: | Set or get secondary light output intensity in candelas. 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  } |

* + - 1. Light level

|  |  |
| --- | --- |
| Property name: | ’lightLevel’ |
| Description: | Set or get ambient light threshold levels in lux. Minimum light level triggering beacon activation. Allowed values:   * null – query from server * any positive number – light intensity in lux |
| Required: | Optional |
| Example: | "lightLevel": 50 |

* + - 1. Low battery voltage warning level

|  |  |
| --- | --- |
| Property name: | ’lowVoltageWarningLevel’ |
| Description: | Set or get low voltage warning level. An warning message ’batteryWarning’ (FIXME: nimi ei ole korrektne) is sent when device power supply voltage drop below low voltage warning level. This level must be higer or equal than low voltage level. It is allowed to change 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 |

* + - 1. Low battery voltage level

|  |  |
| --- | --- |
| Property name: | ’lowVoltageLevel’ |
| Description: | Set or get low voltage level. An warning message ’lowBattery’ (FIXME: nimi ei ole korrektne) is sent when device power supply voltage drop below low voltage level. This level must be lower or equal than low voltage warning level. It is allowed to change device 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 |

* + - 1. Telemetry

|  |  |
| --- | --- |
| Property name: | ’telemetry’ |
| Description: | Common property for telemetry. JSON object, see parameters descriptions below. It is allowed to send query with null parameter, this returns all available parameters with current values. Allowed sub-properties are:   * 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":  {    "report-mode": "interval",    "report-period": 300,    "apn": "internet",    "apn-user": "user",    "apn-password": "secret",    "broker-address": "example.org",    "mqtt-client-id": "client123",    "uid": "location1/location2/site"  } |

* + - * 1. Telemetry report mode

|  |  |
| --- | --- |
| Property name: | ’reportMode’ |
| Description: | Set or get telemetry report mode. Allowed values:   * null – query from server * ’off’ – telemetry data is sent only after query with ’cmd’ * ’utcFixed’ – UTC fixed mode, for example 00:00, 00:05, 00:10, etc. Delay can be added to report period to spread simultaneous sessions. Delay length is implementation defined, for example delay seconds can be calculated from device serial number. * ’interval’ – interval mode, for example every 3 minutes, not fixed to UTC * ’onEvent’ – only when state change and / or error condition is detected |
| Required: | Optional |
| Example: | "telemetry":  {    "reportMode": "interval"  } |

* + - * 1. Telemetry report period

|  |  |
| --- | --- |
| Property name: | ’reportPeriod’ |
| Description: | Set or get telemetry report period. Allowed values:   * null – query from server * 0 – disable telemetry period, telemetry is sent only after query * any positive number – telemetry period in seconds |
| Required: | Optional |
| Example: | "telemetry":  {    "reportPeriod": 300  } |

* + - * 1. APN

|  |  |
| --- | --- |
| Property name: | ’apn’ |
| Description: | Set or get APN. Allowed values:   * null – query from server * string – APN name |
| Required: | Optional |
| Example: | "telemetry":  {    "apn": "internet"  } |

* + - * 1. APN user

|  |  |
| --- | --- |
| Property name: | ’apnUser’ |
| Description: | Set or get APN username. Allowed values:   * null – query from server * string – A1715169904PN username |
| Required: | Optional |
| Example: | "telemetry":  {    "apnUser": "user"  } |

* + - * 1. APN password

|  |  |
| --- | --- |
| Property name: | ’apnPassword’ |
| Description: | Set or get password for APN. Allowed values:   * null – query from server * string – APN password |
| Required: | Optional |
| Example: | "telemetry":  {    "apnPassword": "secret"  } |

* + - * 1. Broker address

|  |  |
| --- | --- |
| Property name: | ’brokerAddress’ |
| Description: | Set or get broker address. List with broker addresses. Address may contain TCP port when non default port is used. IP addresses is also supported. Currently used address will not be changed. 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"]  } |

* + - * 1. MQTT client ID

|  |  |
| --- | --- |
| Property name: | ’mqttClientId’ |
| Description: | Set or get MQTT ClientId [3, 4]. While this parameter allows to write new ClientId value, it is still recommended that this parameter is read only, and ClientId value can be changed by using locally without using MQTT. This parameter may have same value as product serial number. Allowed values:   * null – query from server * string with client id |
| Required: | Yes. Device should respond to query but may ignore write. |
| Example: | "telemetry":  {    "mqttClientId": "mqtt client"  } |

* + - * 1. MQTT username

|  |  |
| --- | --- |
| Property name: | ’mqttUser’ |
| Description: | Set or get MQTT username. It is not required to implement query in device side. Generic topic query should not list username. Can be only used with encrypted payload. Allowed values:   * null – query from server * MQTT username string |
| Required: | Optional. Query may be omitted. |
| Example: | "telemetry":  {    "mqttUser": "MQTT User"  } |

MQTT password

|  |  |
| --- | --- |
| Property name: | ’mqttPassword’ |
| Description: | Set MQTT password. Supports only write command. Can be only used with encrypted payload. Allowed values:   * MQTT password string |
| Required: | Optional. Query may be omitted. |
| Example: | "telemetry":  {    "mqttPassword": "secret"  } |

* + - * 1. Unique ID

|  |  |
| --- | --- |
| Property name: | ’uid’ |
| Description: | Set or get device unique ID string. Unique ID is text that is used to describe device location or device serial number in MQTT topic name.  Allowed values:   * null – query from server * unique ID string. It is allowed to use forward slash for multilevel ID string. Device location string in format <location1>/</location2>/<site> or device serial number can be used for unique ID. |
| Required: | Yes |
| Example: | "telemetry":  {    "uid": "location1/location2/site"  } |

* + - * 1. Certificate selection

|  |  |
| --- | --- |
| Property name: | ’cert’ |
| Description: | Set or get certificate identification. This allows to select certificate from stored certificate list that is used for communication.  Allowed values:   * null – query from server * string with certificate identification |
| Required: | Optional. Only on devices that use TLS/SSL. |
| Example: | "telemetry":  {    "cert": "aws"  } |

* + - 1. GNSS and positioning

|  |  |
| --- | --- |
| Property name: | ’gnss’ |
| Description: | 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"  } |

* + - * 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 (optional). If interval is set, then GNSS started before telematics and repeated with interval. Completion of GNSS task triggers 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"  } |

* + - * 1. GNSS interval

|  |  |
| --- | --- |
| Property name: | ’interval’ |
| Description: | Set or get time interval for GNSS time and position checkup. Allowed values:   * null – query from server * 0 – disable GNSS time and position periodical checkup * any positive number – interval in seconds |
| Required: | Optional |
| Example: | "gnss":  {    "interval": 300  } |

* + - * 1. GNSS duration

|  |  |
| --- | --- |
| Property name: | ’duration’ |
| Description: | Set or get duration for GNSS time and position checkup. Allowed values:   * null – query from server * 0 – disable GNSS time and position checkup * any positive number – duration in seconds |
| Required: | Optional |
| Example: | "gnss":  {    "duration": 120  } |

* + - 1. Latitude and longitude of fixed position

|  |  |
| --- | --- |
| Property name: | ’fixPosition’ |
| Description: | Set or get latitude and longitude of fix position. Array, where first value is latitude and second value longitude. Positive values indicate Northern latitudes and Eastern longitudes. Allowed values:   * 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]  } |

* + - * 1. Maximum allowed distance from fix position

|  |  |
| --- | --- |
| Property name: | ’distanceFromFix’ |
| Description: | Set or get distance from fix position. Allowed values:   * null – query from server * any positive floating-point number – distance from fix in meters |
| Required: | Optional |
| Example: | "gnss":  {    "distanceFromFix": 30  } |

* + - * 1. GNSS sync

|  |  |
| --- | --- |
| Property name: | ’sync’ |
| Description: | Set or get GNSS guided synchronization of flash code. Only available when positioning is enabled. Allowed values:   * 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 |
| Required: | Optional |
| Example: | "gnss":  {    "sync":"utc"  } |

* + 1. 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 controller 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"  } |

* + - 1. Flags property

|  |  |
| --- | --- |
| Property name: | ’flags’ |
| Description: | Send direct command flags. If this property is omitted, then default values is used. Array of strings with flag description:   * noResponse – no response is expected, by default response is expected |
| Required: | Optional |
| Example: | "flags": ["noResponse"] |

* + - 1. Timeout property

|  |  |
| --- | --- |
| Property name: | ’timeout’ |
| Description: | Proprietary command timeout in seconds. This is timeout which is expected to have response from controller. If "noResponse" flags are active, then this command wait given timeout before processing next command. If this property is omitted, then no timeout is used, this is default. |
| Required: | Optional |
| Example: | "timeout": 2 |

* + - 1. Result property

|  |  |
| --- | --- |
| Property name: | ’result’ |
| Description: | Return result of operation. This property is sent only by controller. Allowed string values:   * 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, CRC errors, etc. |
| Required: | Optional, may be omitted when device respond with data property and no errors are set. |
| Example: | "result": "ok" |

* + - 1. Data property

|  |  |
| --- | --- |
| Property name: | ’data’ |
| Description: | Send and receive data from controller. String with hexadecimal coded bytes. If response does not contain any data, then empty string will be sent back. It is possible to send only one device at time, no multicast allowed. |
| Required: | Optional |
| Example: | "data": "0011" |

* + 1. Topic ’certificates’

This topic is intended for certificate download, device registration and certificate renewal. To obtain new certificate device connect to broker and send request for certificate. Certificate request is empty MQTT message. Server responds with new certificate message:

|  |
| --- |
| {    "certificateId": <ID string>,    "certificatePem": <certificate string>,    "privateKey": <private key string>,    "certificateOwnershipToken": <token string>,    "checksum": <checksum, integer>  } |

After successful response device send registration request:

|  |
| --- |
| {    "certificateOwnershipToken": <token string>,    "uid": <device unique ID string>,   "serialNr": <device serial nr>,    "productCode": <device product code>  } |

Server accept registration request:

|  |
| --- |
| {    "deviceName": <device name string>  } |

In case of any error server responds with error message:

|  |
| --- |
| {    "statusCode": <status code>,    "errorCode": <error code string>,    "errorMessage": <error message string>  } |

* + - 1. Certificate ID property

|  |  |
| --- | --- |
| Property name: | ’certificateId’ |
| Description: | The ID of the certificate. Value in this property can be used for certificate identification (7.8.9.11 – Certificate selection) |
| Required: | Optional |
| Example: | "certificateId": "certID1" |

* + - 1. Certificate PEM property

|  |  |
| --- | --- |
| Property name: | ’certificatePem’ |
| Description: | The certificate data, in PEM format. |
| Required: | Yes |
| Example: | "certificatePem": "---BEGIN ..." |

* + - 1. Private key property

|  |  |
| --- | --- |
| Property name: | ’privateKey’ |
| Description: | The private key. |
| Required: | Yes |
| Example: | "privateKey": "---BEGIN ..." |

* + - 1. Certificate ownership token property

|  |  |
| --- | --- |
| Property name: | ’certificateOwnershiptoken’ |
| Description: | The token to prove ownership of the certificate during provisioning. |
| Required: | Yes |
| Example: | "certificateOwnershiptoken": "token123" |

* + - 1. Checksum property

|  |  |
| --- | --- |
| Property name: | ’checksum’ |
| Description: | Integer value. CRC-32 calculated over all other properties data in this message. Used only to check data corruption. |
| Required: | Optional |
| Example: | "checksum": 3632233996 |

* + - 1. Unique ID property

|  |  |
| --- | --- |
| Property name: | ’uid’ |
| Description: | See description in 7.8.9.11 – Unique ID |
| Required: | Yes |
| Example: | "uid": "location1/location2/site" |

* + - 1. Serial number property

|  |  |
| --- | --- |
| Property name: | ’serialNr’ |
| Description: | See description in 7.5.6 – Serial number |
| Required: | Yes |
| Example: | "sserialNr": "flasher1-001" |

* + - 1. Product code property

|  |  |
| --- | --- |
| Property name: | ’productCode’ |
| Description: | See description in 5.4.6 – Product code |
| Required: | Yes |
| Example: | "productCode": "flasher1" |

* + - 1. Device name property

|  |  |
| --- | --- |
| Property name: | ’deviceName’ |
| Description: | The name of the device created during provisioning. |
| Required: | Yes |
| Example: | "deviceName": "device123" |

* + - 1. Status code property

|  |  |
| --- | --- |
| Property name: | ’statusCode’ |
| Description: | Integer to describe status of operation. |
| Required: | Yes |
| Example: | "statusCode": 200 |

* + - 1. Error code property

|  |  |
| --- | --- |
| Property name: | ’errorCode’ |
| Description: | String that describes error. |
| Required: | Yes |
| Example: | "errorCode": "AccessDenied" |

* + - 1. Error message property

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

* 1. Security and encryption

1. For minimum security, MQTT username and password authentication should be used [5]. Both values should not be accessible directly by 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 7.10.12 – Error message property.
   1. Changes

2024-04-22

1. 1In alert status: lanternCons renamed to lanternPowerWarning
2. General message data rules: changed name format rules (camel case only)
3. GNSS quality indicator: changed CSQ (wrong value), to HDOP
4. Firmware version: updated text in example
5. Maximum allowed distance from fix position: added more descripte text

2024-05-06

1. Added low voltage warning level configuration ’lowVoltageWarningLevel’
2. Added low voltage warning ’batteryWarning’
3. Changed GNSS quality indicator property name: in ’position’ property, changed ’quality’ to ’hdop’
4. Added “Accumulated solar charge during last 24 hours” (‘solarCharge’)
5. Added “Beacon power” (‘beaconPower’)
6. Added reset all statistics command (‘reset=statistics’)
7. Added “Illumination counter” (‘illuminationCounter’)

2024-05-07

1. Moved ‘time’ property from telematics to general topic
2. Added “Logical architecture of MQTT enabled device”

2024-05-08

1. Replaced ‘timestamp’ property in session message with general ‘time’ property.
2. Removed message rule: “It is not recommended to use the same property names on different sub- properties in the data from the server to the device. For example {a:{"sub1”:1},b:{"sub1":1}} is not recommended.”
3. Added query to light on demand
4. Added query to ‘fixPosition’
5. Updated description for ‘time’ property in commands topic

2024-05-10

1. Added reccomendations to server side implementations
2. Changed comma separated list to JSON list in “Broker address” property.
   1. Bibliography
3. Introducing JSON, 2023, [https://www.json.org/json-en.html](https://www.json.org/json-en.html" \t "_top)
4. The JavaScript Object Notation (JSON) Data Interchange Format, 2017, [https://www.rfc-editor.org/rfc/rfc8259](https://www.rfc-editor.org/rfc/rfc8259" \t "_top)
5. Client identifier, 2023-09-06, [https://www.ibm.com/docs/en/ibm-mq/9.3?topic=concepts-client-identifier](https://www.ibm.com/docs/en/ibm-mq/9.3?topic=concepts-client-identifier" \t "_top)
6. 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/](https://www.hivemq.com/blog/mqtt-essentials-part-3-client-broker-connection-establishment/" \t "_top)
7. Securing MQTT with Username & Password Authentication, 2023-03-30, [https://www.emqx.com/en/blog/securing-mqtt-with-username-and-password-authentication](https://www.emqx.com/en/blog/securing-mqtt-with-username-and-password-authentication" \t "_top)
8. Example of appendix Title (Head 1) style

Appendices should be started on a separate page and contain information that is directly relevant to the main body of the text at a certain point, but that would be too large or distracting to include at that particular point. There are four levels of appendix heading styles available in the **Style Gallery.**

* 1. Example of Appendix Head 1 style
     1. Example of Appendix Head 2 Style

At the end of the **Appendix head 2** style text press carriage return, the following paragraph is **the Heading 1 separation line** style, press carriage return again, and the following line will be in **Body text** style.

* + - 1. Example of Appendix head 3 style

The same following formatting applies to the **Appendix Head 3** style i.e., press carriage return, the following paragraph is the **Heading 2 separation line** style, press carriage return again, and you will be back to body text.

* + - * 1. Example of Appendix Head 4 style

The Appendix Head 4 style is followed by body text and does not have a separation line. Only the level 1 **Appendix Title** will appear in the TOC.

* + - * 1. Example of Appendix Head 5 style

The **Appendix Head 5 style** is followed by body text and does not have a separation line. Figure and tables should be labelled as a continuation from the main Guideline content.

1. Example of Annex title (Head 1) style

Annexes should include information that can exist in isolation e.g.

* a technical specification for a new piece of equipment;
* the content and structure of a new training module; or
* the detail associated with a new recommendation for an AIS.

Annexes can include appendices if required. There are also four levels of annex heading styles available in the **Style Gallery.** In addition to the **Annex title** (**Head 1)** style there is **Annexe Head 2**, **Annexe Head 3** and **Annexe Head 4**. These follow a similar format to the appendix heading styles. As many annexes can be included as needed and it is advisable to separate them with a page break. Only the level 1 **Annex title** style text will appear in the TOC.

* 1. Example of Annex Head 2 style
     1. Example of Annex Head 3 style
        1. Example of Annex Head 4 style

Annex figures and tables should be labelled with the **Annex Figure Caption** and **Annex Table Caption** styles respectively, rather than the main figure and table caption styles. This ensures the annex can be read logically in isolation and that annex figures and tables are not included in the List of Figures and Tables respectively on the main Guideline contents page.

1. Example of annex figure caption
   * + - 1. Example of Annex Head 5 style

1. Footnotes should be used sparingly. [↑](#footnote-ref-1)