System Specification

enocean alliance
Building Smarter Connectivity

EnOcean over IP MQTT Implementation Example Specification

V 1.0

San Ramon, CA, USA
Original Release 1.0, 2021

Executive Summary

This document is owned by the Technical Working Group (TWG) of the EnOcean Alliance. It is maintained and will be progressed within the authority of the Chairman of the TWG.

REVISION HISTORY

Ver.	Editor	Change	Date
1.0	AP	Internal Release candidate	Mar 30, 2021

DISCLAIMER

This information within this document is the property of the EnOcean Alliance and its use and disclosure are restricted. Elements of the EnOcean Alliance specifications may also be subject to third party intellectual property rights, including without limitation, patent, copyright or trademark rights (such a third party may or may not be a member of the EnOcean Alliance.)

The EnOcean Alliance is not responsible and shall not be held responsible in any manner for identifying or failing to identify any or all such third party intellectual property rights. This document and the information contained herein are provided on an “as is” basis and the EnOcean Alliance disclaims all warranties express or implied, including but not limited to
(1) any warranty that the use of the information herein will not infringe any rights of third parties (including any intellectual property rights, patent, copyright or trademark rights, or SEP(2) any implied warranties of merchantability, fitness for a particular purpose, title or non-infringement.

In no event will the EnOcean Alliance be liable for any loss of profits, loss of business, loss of use of data, interruption of business, or for any other direct, indirect, special or exemplary, incidental, punitive or consequential damages of any kind, in contract or in tort, in connection with this document or the information contained herein, even if advised of the possibility of such loss or damage. All Company, brand and product names may be trademarks that are the sole property of their respective owners.
The above notice and this paragraph must be included on all copies of this document that are made.

The EnOcean Alliance “EnOcean over IP MQTT Implementation Example Specification” is available free of charge to companies, individuals and institutions for all non-commercial purposes (including educational research, technical evaluation and development of non-commercial tools or documentation.)

This specification includes intellectual property („IPR“) of the EnOcean Alliance and joint intellectual properties („joint IPR“) with contributing member companies. No part of this specification may be used in development of a product or service for sale without being a participant or promoter member of the EnOcean Alliance and/or joint owner of the appropriate joint IPR.

These errata may not have been subjected to an Intellectual Property review, and as such, may contain undeclared Necessary Claims.

EnOcean Alliance Inc.
2603 Camino Ramon, Suite 200

San Ramon, CA 94583
USA
Graham Martin
Chairman & CEO EnOcean Alliance

Table of Contents

Introduction

1.1. Scope and Purpose

This document is intended for users and producers of EAGs (EnOcean Alliance IP Gateways) to implement or use MQTT-Interface, which simplifies the EnOcean Alliance World to third parties. All techniques, explained here, are based on standardized IP and EnOcean Alliance technologies. One of the main aims of the specification is to remove the complexity of the EnOcean profiles for product & system developers and to allow them to use/integrate EnOcean technology simply in their products.

For manufacturers of an EAG, this document shows how different profiles and EnOcean technologies are mapped to an MQTT-Interface. This document should be understood as one possible - already implemented and field proven - example of how this could be implemented and is intended to assist developers in reducing development times in creating solutions around the EnOcean Standard. It is not compulsory to implement the MQTT interface exactly as described in this document and it is not planned that any future EnOcean Alliance certification will demand this.

This document is not intended to create a User Manual or Product Specification for an EAG, the implementer may add additional features to his gateway. The specification should be a base line for an EAG, which allows a basic usage for EnOcean Alliance Products. It also does not contain any information nor has the aim to generate a logic/rule machine for automatic interaction with EnOcean devices. The document is a baseline to allow other vendors/app developer or rule machines to connect to the EAG.

1.2. Definitions

API – **A**pplication **P**rogramming **I**nterface

BaseID – is a base Address which may be used to send telegrams to different actuators using different source IDs. It is possible to transmit a telegram with the base ID of an EnOcean device as the source address and any ID up to the base ID + 127.

Device-Identifier – Contains information about the EnOcean device, i.e. friendly name, EURID, location, etc.
The format is not defined by the EnOcean Alliance and can be customer, product or device specific. It has to be unique to the EAG.

Digital Twin - is a real time digital replica of a physical device.

EAG – An **E**nOcean **A**lliance **G**ateway is defined as a Gateway developed (typically) by an EnOcean Alliance member company which receives EnOcean communication (e.g. EEPs, GPs) and translates these into IP representations typically according to the EnOcean over IP specification for transportation to the internet via various possible transportation methods (e.g.

MQTT or HTTP). In the case of a bi-directional communication to the field device, the EAG can also accept communication from IP network and translate these commands into EnOcean communication to be sent to the EnOcean based field devices. Typically, an EAG will include a digital twin of each field device, storing the actual state of the device.

EAG-Identifier – Identifies the gateway connected to the broker, e.g. EAG name, location, controller, EURID, etc.
The format of the identifier is not defined by the EnOcean Alliance and can be customer or product specific. It has to be unique on the broker.

EEP - **E**nOcean **E**quipment **P**rofile; Specification to define the structure of over-the-air data bytes for EnOcean transmitters. Also see Generic Profiles.

EnOcean based field devices – are devices which use EnOcean radio to communicate.

EURID – **E**nOcean **U**nique **R**adio **Id**entifier, a unique and non-changeable identification number assigned every EnOcean transmitter during its production process.

GP - **G**eneric **P**rofiles; Specification to define the structure of over-the-air data bytes for EnOcean transmitters. Also see EEP.

HTTP – **H**yper**t**ext **T**ransfer **P**rotocol

IoT – **I**nternet **O**f **T**hings

JSON - **J**ava **S**cript **O**bject **N**otation

MQTT - MQTT is an OASIS standard for IoT connectivity. It is a publish/subscribe, extremely simple and lightweight messaging protocol, designed for constrained devices and low-bandwidth, high-latency or unreliable networks. The design principles are to minimize network bandwidth and device resource requirements whilst also attempting to ensure reliability and some degree of assurance of delivery. These principles also turn out to make the protocol ideal of the "Internet of Things" world of connected devices, and for mobile applications where bandwidth and battery power are at a premium.

REST - Representational State Transfer

TCP/IP - Transmission Control Protocol (TCP) and the Internet Protocol (IP). The Internet protocol suite provides end-to-end data communication specifying how data should be packetized, addressed, transmitted, routed, and received.

TLS – **T**ransport **L**ayer **S**ecurity, a cryptographic protocol designed to provide communications security over a computer network.

1.3. Documents

The EnOcean over IP Specification consists of a main document and several descriptions of example implementations with different transport protocols. You find them in the technical specifications section of the EnOcean Alliance homepage.

The EEP Viewer tool provides the IP representation description of the various EEPs.

1.4. References

1.4.1. EnOcean Alliance

[E1] EnOcean over IP Specification
https://www.enocean-alliance.org/ip/

[E2] EnOcean over IP REST API Implementation Example
https://www.enocean-alliance.org/ip/rest/api

[E3] EEP (EnOcean Equipment Profiles) Specification
https://www.enocean-alliance.org/eep/

[E4] EnOcean GP(Generic Profiles)
https://www.enocean-alliance.org/gp/

[E5] EnOcean Remote Management Specification.
https://www.enocean-alliance.org/reman/

[E6] EnOcean Wireless Standard
https://www.enocean-alliance.org/about-us/enocean-wireless-standard/

[E7] EEP Viewer
http://tools.enocean-alliance.org/EEPViewer/

1.4.2. Internet Engineering Task Force Documents

[RFC1] RFC 7159: The JavaScript Object Notation (JSON) Data Interchange Format
https://tools.ietf.org/html/rfc7159

[RFC2] RFC 7230: Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing
https://tools.ietf.org/html/rfc7230

1.4.3. Others

[O1] JSON website
http://www.json.org/

[O2] MQTT v 3.1.1
http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.html

[O3] MQTT v 5.0
https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html

2. EnOcean Over IP – MQTT Interface

2.1. Motivation

With the emergence of IoT technology it is necessary that EnOcean devices become a part of the IoT world. Due to the limitation of Energy Harvesting Devices and the EnOcean protocol, it is not possible to integrate the devices directly into the IP world with e.g. a 6LoWPAN Adaption Layer. This specification specifies a MQTT Interface to allow an easy integration of an EnOcean ECO-System into an IP Network, Intranet or Internet. Another main motivation is to hide the complexity of the EnOcean Radio and the different Application Protocols (e.g. EEP/GP/ReCom) from the user of the API

MQTT is a Client Server publish/subscribe messaging transport protocol. It is lightweight, open, simple, and designed to be easy to implement. These characteristics make it ideal for use in many situations, including constrained environments such as for communication in Machine to Machine (M2M) and Internet of Things (IoT) contexts where a small code footprint is required and/or network bandwidth is at a premium.

The MQTT protocol is based on TCP/IP and the client needs to have a TCP/IP stack.

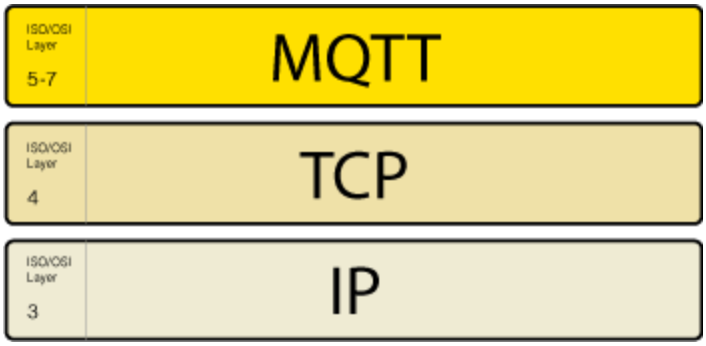

Figure 1: ISO/OSI layers used for MQTT

2.2. Security recommendations

The Security of the MQTT is not part of this spec and manufacturer specific, but it is recommended to use TLS to provide encryption, authentication and message integrity.

See OASIS references [O2] or [O3] chapter's security.

2.3. MQTT

2.3.1. Introduction to MQTT

The publish/subscribe pattern provides an alternative to traditional client-server architecture. In the client-server model, a client communicates directly with an endpoint. MQTT decouples the client that sends a message (the publisher) from the client or clients that receive the messages (the subscribers). The publishers and subscribers never contact each other directly. In fact, they are not even aware that the other exists. The connection between them is handled by a third component (the broker). The job of the broker is to filter all incoming messages and distribute them correctly to subscribers.

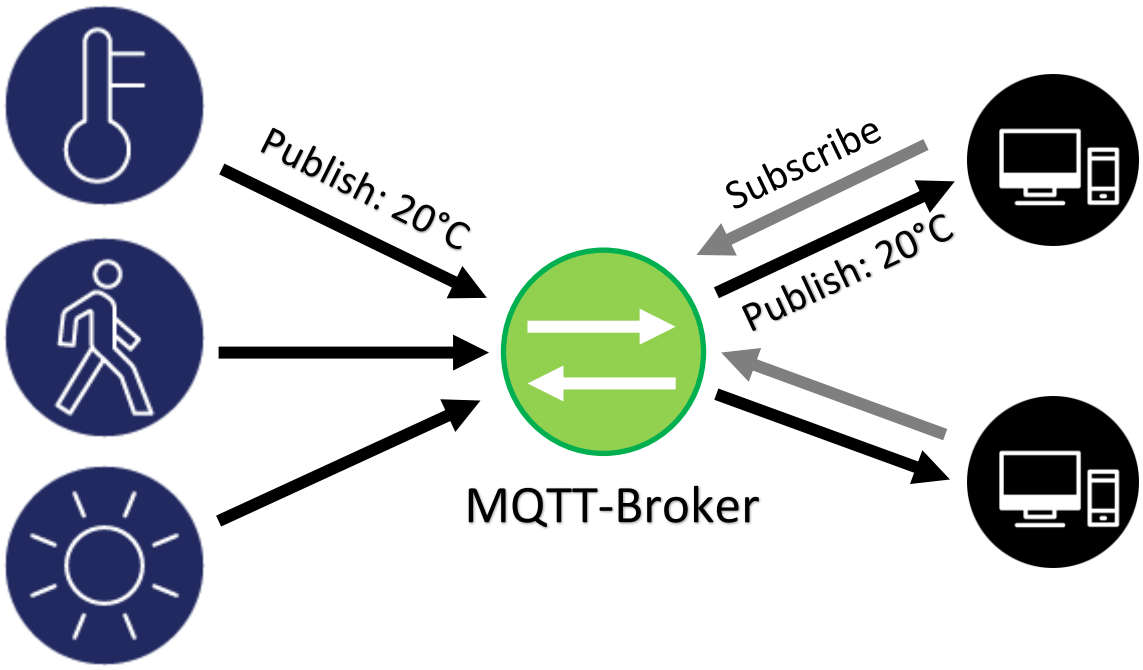

Figure 2: Principle MQTT architecture

To enable sending information from clients via MQTT Broker to the EAG, or to EnOcean sensors, the EAG itself is able to subscribe for messages from the broker. The sensors are connected via EnOcean Radio Protocol to the EAG and the EAG itself is connected to the broker via any network.

The EAG system MQTT architecture looks like this:

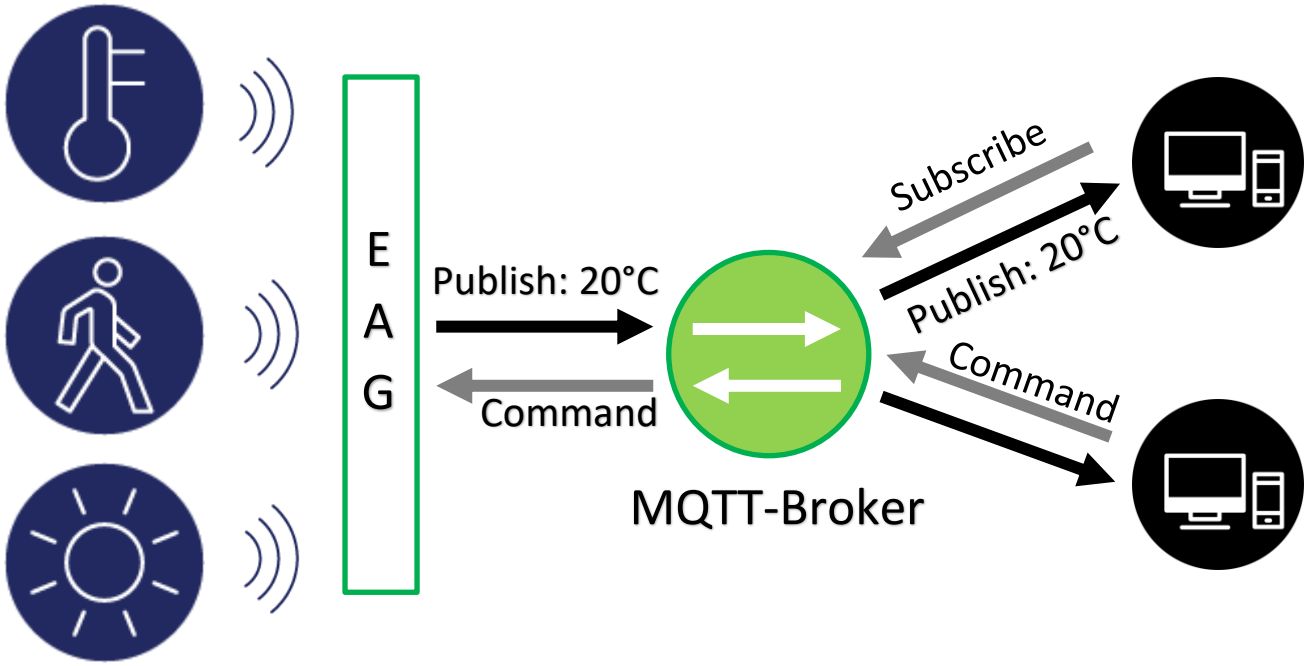

Figure 3: EAG system MQTT architecture

2.3.2. Topics

MQTT uses subject-based filtering of messages. Every message contains a topic (subject) that the broker can use to determine whether a subscribing client gets the message or not.

In MQTT, the word “topic” refers to an UTF-8 string that the broker uses to filter messages for each connected client. The topic consists of one or more topic levels. Each topic level is separated by a forward slash (topic level separator).

Example:
EnOcean/GW-Identifier/stream/Device-Identifier/from

EAG’s shall use the topics defined in chapter 3.

2.3.3. Payload

MQTT is data-agnostic. The use case of the client determines how the payload is structured. The sending client (publisher) decides whether it wants to send binary data, text data, or even full-fledged XML or JSON.

The EAG shall never use binary data. It shall use the JSON objects defined in [E1].

2.3.4. Quality of Service

The Quality of Service (QoS) level is an agreement between the sender of a message and the receiver of a message that defines the guarantee of delivery for a specific message. There are 3 QoS levels in MQTT:

- At most once (0)
  This service level offers a best-effort-delivery. There is no guarantee of delivery.
- At least once (1)
  Level 1 guarantees that a message is delivered at least one time to the receiver. QoS 1 is the most frequently used service level if the recipients are able to handle occasional duplicated messages, as it avoids the overheads associated with QoS 2.
- Exactly once (2)
  This is the highest level of service in MQTT. This level guarantees that each message is received only once by the intended recipients. QoS 2 is the safest and slowest quality of service level with the most overhead.

The EnOcean Alliance recommends using QoS level 1 or 2 depending on the use case.

2.3.5. Last Will and Testament

Because EAG's are sometimes used in mobile networks it is reasonable to assume that MQTT clients in these scenarios will occasionally disconnect ungracefully.

In MQTT the Last Will and Testament (LWT) feature is used to notify other clients about an ungracefully disconnected client. The last will message is a normal MQTT message with a topic, retained message flag, QoS and payload. The broker stores the message until it detects that the client has disconnected ungracefully. In response to the ungraceful disconnect, the broker sends the last-will message to all subscribed clients of the last-will message topic. If the client disconnects gracefully with a correct DISCONNECT message, the broker discards the stored LWT message.

EAC's shall use the LWT defined in chapter 3.6.

2.3.6. Keep alive

This feature is especially important for mobile networks. MQTT is based on TCP which ensures, that packets are transferred over the internet in a reliable, ordered and error checked way.

Nevertheless, from time to time, the transfer between communicating parties can get out of sync if one of the parties crashes or has transmission errors. In TCP, this state of incomplete connection is called a half-open connection.

MQTT includes a keep alive function that provides a workaround for the issue of half-open connections. When the client establishes a connection to the broker, the client communicates a time interval in seconds to the broker. This interval defines the maximum length of time that the broker and client may not communicate with each other.

As long as messages are exchanged frequently and the keep-alive interval is not exceeded, there is no need to send an extra message to establish whether the connection is still open.

For EAG's it is not ensured, that frequently messages are transferred. This depends on the amount and the type of the EnOcean devices connected.

If the EAG does not send a message during the keep-alive period, it has to send a PINGREQ packet to the broker to confirm that it is available and to make sure that the broker is also still available.

The broker must disconnect a client that does not send a message or a PINGREQ packet in one and a half times the keep alive interval.

The EAG can send a PINGREQ packet any time it wants to confirm that the network connection is still alive. The PINGREQ packet does not contain a payload. When the broker receives a PINGREQ it replies with a PINGRESP packet to show the client that it is still available. The PINGRESP packet also does not contain a payload.

If the broker does not receive a PINGREQ or any other packet from a client, the broker closes the connection and sends the last will and testament message.

The EAG has to set an appropriate keep alive value depending on the use case and environment.

2.3.7. Retained Messages

Retained messages help newly-subscribed clients get a status update immediately after they subscribe to a topic. The retained message eliminates the wait for the publishing clients to send the next update.

A retained message is a normal MQTT message with the retained flag set to true. The broker stores the last retained message. Each client that subscribes to a topic pattern that matches the topic of the retained message receives the retained message immediately after they subscribe. The broker stores only one retained message per topic.

The EAG may set the retained flag to true if required for the messages it publishes.

3. Topic definitions

This is the topic structure of an EAG:

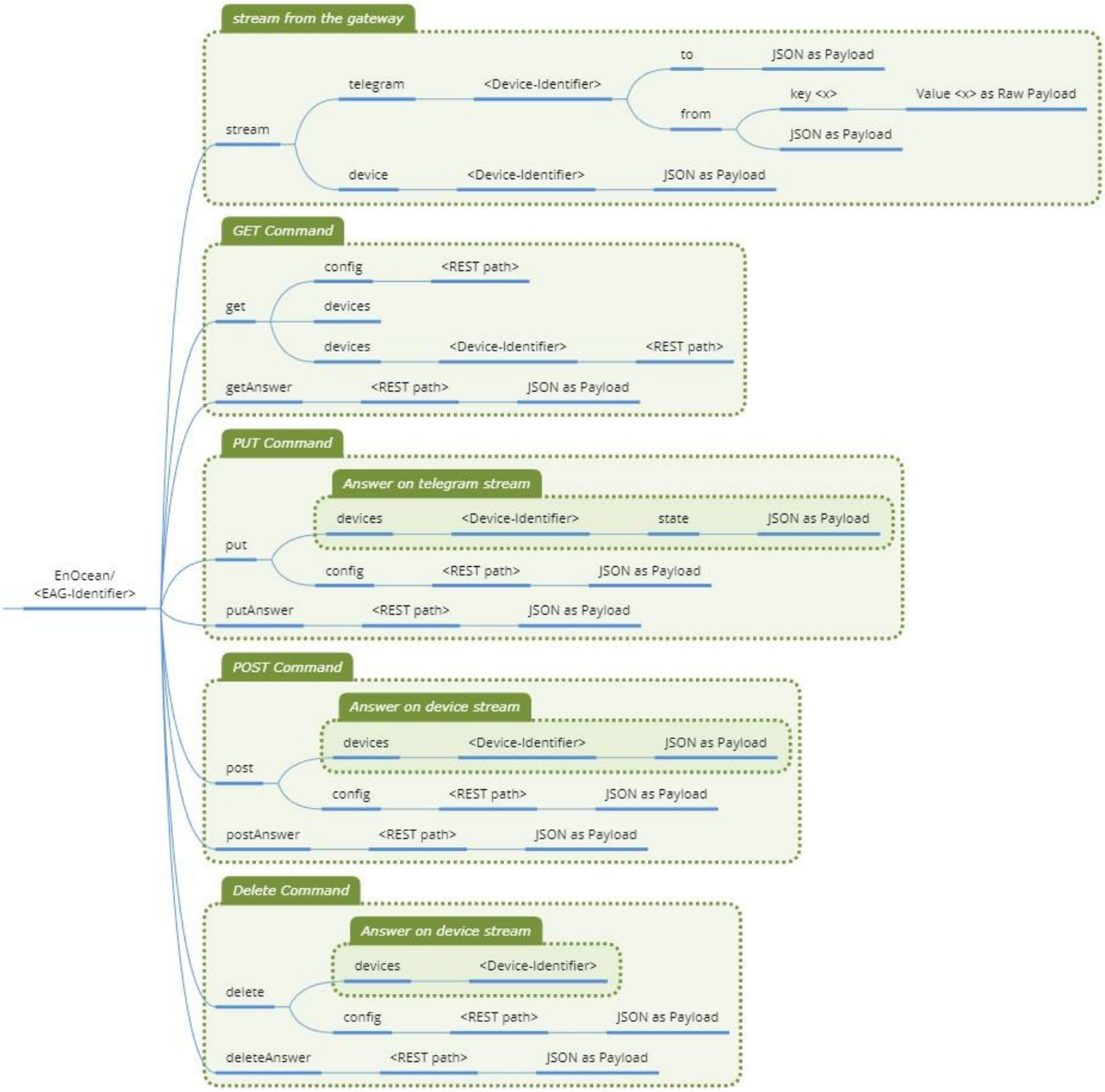

Figure 4: Topic structure of EAG

3.1. The stream topic

The stream/telegram topics are used by the EAG to publish copies of the telegram communication data that travel back and forth between devices and EAG. Each MQTT message contains a single telegram. The payload data is composed of IP-Representation information either with JSON, or the raw payload.
In any case a published entry contains only the data of one radio telegram received.

3.1.1. Unidirectional sensor device

Device-level topics

The following topic structure has to be supported by each EAG:

EnOcean/{EAG-Identifier}/stream/telegram/{Device-Identifier}/from
The payload is a telegram object.

Example:
EnOcean/0185408E/stream/telegram/SensorFriendlyName /from

JSON for telegram Object

```
"telegram" : {
    "deviceId" : "01843197",
    "friendlyId" : " SensorFriendlyName",
    "physicalDevice" : "testdevice",
    "timestamp" : "2020-11-16T18:12:16.134+0100",
    "direction" : "from",
    "functions" : [ {
      "key" : "humidity",
      "value" : "0",
      "unit" : "%"
    }, {
      "key" : "temperature",
      "value" : "0",
      "unit" : "°C"
    } ],
    "telegramInfo" : {
      "data" : "0000000A",
      "status" : "0",
      "dbm" : -65,
      "rorg" : "A5"
    }
  }
```

Figure 5: telegram JSON example

Key-level topics

As an alternative, the EAG may publish telegram data at the level of individual EEP keys. This may be configured by the user and is EAG specific. The EAG can be configured to report at device level for some devices and key level for others. In case this alternative is enabled, the EAG need not to publish data on the corresponding device-level topic.

EnOcean/{EAG-Identifier}/stream/telegram /{Device-Identifier}/from /{key}

The payload is a functions array containing just the entry for the particular key, even when the telegram for the device contains more than one key. The use case for this alternative topic form is a customer who is only interested in a special data value of a telegram.

If a telegram is received from the EnOcean field device which does not contain this key, nothing is published.

Example:

EnOcean/0185408E/stream/telegram/SensorFriendlyName /from/humidity

JSON for functions Object

```
{
    "key" : "humidity",
    "value" : "0",
    "unit" : "%"
}
```

Figure 6: telegram key payload example

3.1.2. Bidirectional sensor device

Whenever an EAG sends an EnOcean telegram to a field device, independent if this was triggered by a MQTT client or by the EAG itself, it publishes a copy of that telegram to a stream-telegram-level topic of the form

EnOcean/{EAG-Identifier}/stream/telegram/{Device-Identifier}/to
The payload is a telegram object (see chapter 3.1.1.).

This stream topic containing the direction “to” reports all EnOcean telegrams transmitted to the field device. To keep the complexity low, a key level topic with direction “to” is not supported. Each EAG manufacturer is free to implement a key level topic individually for this use case.

If an MQTT client wishes to send a telegram to an EnOcean field device it uses a put topic (see chapter 3.2.1.) of the form

EnOcean/{EAG-Identifier}/put/devices/{Device-Identifier}/state

to send it, via the MQTT Broker, to the EAG and the EAG then forwards it to the EnOcean field device. The payload is a telegram object (see chapter 3.1.1.).

The EAG is subscribed to this topic and if it receives a valid EnOcean telegram it sends it to the EnOcean field device according the EEP. This is an asynchronous process.

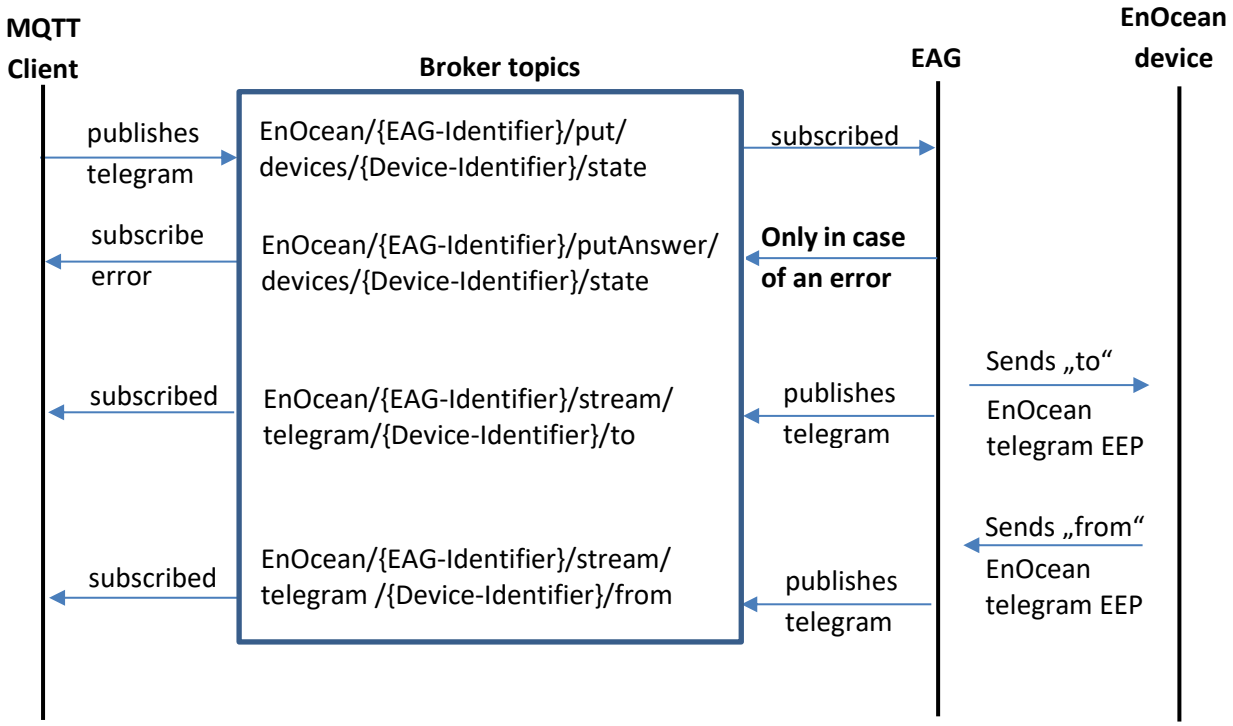

Figure 7: Bidirectional sensors message flow

In case an error occurs and the EAG is not able to send the telegram to the EnOcean field device, the EAG publishes the error on this topic:
EnOcean/{EAG-Identifier}/putAnswer/devices/{Device-Identifier}/state
The payload is an error object.

When it has sent the telegram to the EnOcean field device, the EAG publishes the telegram object to this topic (this is the confirmation that the telegram was sent from EAG):
EnOcean/{EAG-Identifier}/stream/telegram /{Device-Identifier}/to
The payload is a telegram object (see before).

If the EnOcean field device reacts by sending a telegram to the EAG, the EAG publishes it in the "from" direction of the telegram according chapter 3.1.1.
EnOcean/{EAG-Identifier}/stream/telegram /{Device-Identifier}/from

3.1.3. Examples

Example D2-01-00 Query the energy value and response

EnOcean/0185408E/put/devices/SensorFriendlyName/state (published from client)

JSON for telegram Object

```
"telegram" : {
    "deviceId" : "01843197",
    "friendlyId" : " SensorFriendlyName",
    "physicalDevice" : "testdevice",
    "timestamp" : "2020-11-16T18:12:16.134+0100",
    "direction" : "to",
    "functions" : [{
      "key" : "query",
      "value" : "energy"
    }]
 }
```

Figure 8: Query Event "to" EnOcean field device example

EnOcean/0185408E/stream/telegram /SensorFriendlyName/to (published from EAG)

JSON for telegram Object

```
"telegram" : {
    "deviceId" : "01843197",
    "friendlyId" : " SensorFriendlyName",
    "physicalDevice" : "testdevice",
    "timestamp" : "2020-11-16T18:12:16.134+0100",
    "direction" : "to",
    "functions" : {
      "key" : "query",
      "value" : "energy"
    }
    "telegramInfo" : {
    "data" : "0000000A",
    "status" : "0",
    "rorg" : "A5"
    }
 }
```

Figure 9: Query Event "to" EnOcean field device example

EnOcean/0185408E/stream/telegram /SensorFriendlyName/from (EAG publishes the response it received from the device)

JSON for telegram Object

```
"telegram" : {
    "deviceId" : "01843197",
    "friendlyId" : " SensorFriendlyName",
    "physicalDevice" : "testdevice",
    "timestamp" : "2020-11-16T18:12:17.134+0100",
    "direction" : "from",
    "functions" : {
      "key" : "energy",
      "value" : "78"
    }
    "telegramInfo" : {
    "data" : "0000000A",
    "status" : "0",
    "dbm" : -65,
    "rorg" : "A5"
    }
}
```

Figure 10: Response "from" EnOcean field device example

Example D2-01-00 Actuator Status Response only, triggered by EnOcean field device

EnOcean/0185408E/stream/telegram /SensorFriendlyName/from

JSON for telegram Object

```
"telegram" : {
    "deviceId" : "01843197",
    "friendlyId" : " SensorFriendlyName",
    "physicalDevice" : "testdevice",
    "timestamp" : "2020-11-16T18:12:16.134+0100",
    "direction" : "from",
    "functions" : { [ {
      "key" : "localControl",
      "value" : "off",
      }, {
      "key" : "switch",
      "value" : "on",
      } ]
    },
    "telegramInfo" : {
    "data" : "0000000A",
    "status" : "0",
    "dbm" : -65,
    "rorg" : "A5"
    }
 }
```

Figure 11: Actuator status response JSON payload example

Example D2-01-11 Actuator Set

EnOcean/0185408E/put/devices/SensorFriendlyName/state (published from client)

JSON for telegram Object

```
"telegram" : {
    "deviceId" : "01843197",
    "friendlyId" : " SensorFriendlyName",
    "physicalDevice" : "testdevice",
    "timestamp" : "2020-11-16T18:12:16.134+0100",
    "direction" : "to",
    "functions" : [ {
      "key" : "channel",
      "value" : "0"
    }, {
      "key" : "switch",
      "value" : "on"
    } ]
  }
```

Figure 12: Actuator Set JSON example

EnOcean/0185408E/stream/telegram/SensorFriendlyName/to (published from EAG)

JSON for telegram Object

```
"telegram" : {
    "deviceId" : "01843197",
    "friendlyId" : " SensorFriendlyName",
    "physicalDevice" : "testdevice",
    "timestamp" : "2020-11-16T18:12:17.134+0100",
    "direction" : "to",
    "functions" : [ {
      "key" : "channel",
      "value" : "0"
    }, {
      "key" : "switch",
      "value" : "on"
    } ]
    "telegramInfo" : {
    "data" : "0000000A",
    "status" : "0",
    "rorg" : "A5"
    }
 }
```

Figure 13: functions JSON example

EnOcean/0185408E/stream/telegram/SensorFriendlyName/from (reaction after actuator output is set)

JSON for telegram Object

```
"telegram" : {
    "deviceId" : "01843197",
    "friendlyId" : " SensorFriendlyName",
    "physicalDevice" : "testdevice",
    "timestamp" : "2020-11-16T18:12:19.134+0100",
    "direction" : "from",
    "functions" : { [ {
      "key" : "channel",
      "value" : "0",
    }, {
      "key" : "switch",
      "value" : "on",
    } ] },
    "telegramInfo" : {
    "data" : "0000000A",
    "status" : "0",
    "dbm" : -65,
    "rorg" : "A5"
    }
 }
```

Figure 14: functions JSON example

Example for error object:

EnOcean/0185408E/putAnswer/devices/SensorFriendlyName/state

JSON for error Object

```
"error" : 404
```

Figure 15: error object example

3.1.4. Device information

To see the device information an EAG manages the following topic structure has to be supported by each EAG:

EnOcean/{EAG-Identifier}/stream/device/{Device-Identifier}
The payload is a devices object.

In this topic all managed devices are published with their device information.

3.2. The put topic

The put topic will be used to send radio telegrams to EnOcean field devices or to configure the EAG system.

3.2.1. Send radio telegrams to field device

This use case is for sending data from an EAG or a MQTT client to an EnOcean field device. The telegram object will be published under the topic:
EnOcean/{EAG-Identifier}/put/devices/{Device-Identifier}/state
The payload is a telegram object (see chapter 3.1.1.).

More details are already described in chapter 3.1.2. and examples can be found in chapter 3.1.3.

3.2.2. Configure the EAG system

This use case is for configuring the system of the EAG via the following topic:
EnOcean/{EAG-Identifier}/put/config/{REST path}
The payload is a JSON object. This specification will not define the {REST path} and the payload because it is manufacturer specific.

The answer of this configuration will be posted in this topic:
EnOcean/{EAG-Identifier}/putAnswer/{REST path}
The payload is a JSON object. This specification will not define the {REST path} and the payload because it is manufacturer specific.

Configuration of the EAG system with MQTT is not mandatory to implement.

3.3. The get topic

The following topic examples are not mandatory to be implemented.

3.3.1. Use case get all devices from EAG

This use case is for new subscribers to get all the devices information of an EAG. The request will be published under the topic:
EnOcean/{EAG-Identifier}/get/devices
There is no payload needed.

The EAG then publishes all device object(s) under the topic(s):
EnOcean/{EAG-Identifier}/getAnswer/devices/{Device-Identifier}

Example:

EnOcean/0185408E/get/devices

```
No payload
```

Figure 16: No payload for requesting all devices objects from EAG

EnOcean/0185408E /getAnswer/devices/Opus-Bridge-2K

```
JSON for device object
"device" : {
    "deviceId" : "01910188",
    "friendlyId" : "Opus-Bridge-2K",
    "eeps" : [ {
      "eep" : "D2-01-11",
      "version" : 0.9,
      "direction" : "both"
    }, {
      "eep" : "F6-02-01",
      "version" : 0.9,
      "direction" : "from"
    }, {
      "eep" : "F6-03-01",
      "version" : 0.9,
      "direction" : "from"
    } ],
    "manufacturer" : "Jaeger Direkt",
    "firstSeen" : "2016-10-27T07:59:07.137+0200",
    "lastSeen" : "2016-11-30T12:51:33.350+0100",
    "dbm" : -70
  }
```

Figure 17: device JSON example

The EAG publishes device objects of all devices similar as below.

Only in case of an error, the device publishes an error object under the topic:
EnOcean/0185408E/getAnswer/devices/Opus-Bridge-2K

```
JSON for error object
"error" : 404
```

Figure 18: error JSON example

It is also possible to request the information of one device only and this will be published under the topic:
EnOcean/{EAG-Identifier}/get/devices/{Device-Identifier}
There is no payload needed.

The EAG then publishes the requested device object under the topic:
EnOcean/{EAG-Identifier}/getAnswer/devices/{Device-Identifier}

3.3.2. Request system information of the EAG

This use case is for requesting the system information's of the EAG via the following topic:
EnOcean/{EAG-Identifier}/get/config/{REST path}
This specification will not define the {REST path} and the payload because it is manufacturer specific.

The answer of this request will be posted in this topic:
EnOcean/{EAG-Identifier}/getAnswer/{REST path}
This specification will not define the {REST path} and the payload because it is manufacturer specific.

Examples:

EnOcean/{EAG-Identifier}/get/config/system/info
There is no payload needed.
EnOcean/{EAG-Identifier}/getAnswer/config/system/info
The payload are the http answers.

EnOcean/{EAG-Identifier}/get/config/system/processes
There is no payload needed.
EnOcean/{EAG-Identifier}/getAnswer/config/system/processes
The payload are the http answers.

3.4. The post topic

The following topic examples are not mandatory to be implemented.

3.4.1. Use case add a device to EAG

This use case is for adding a new device to an EAG. The device object will be published under the topic:
EnOcean/{EAG-Identifier}/post/devices/{Device-Identifier}
The payload is a device object.

Only in case of an error, the device publishes an error object under the topic
EnOcean/{EAG-Identifier}/postAnswer/devices/{Device-Identifier}
The payload is an error object.

The EAG then publishes the new device object under the topic:
EnOcean/{EAG-Identifier}/stream/devices/{Device-Identifier}
The payload is a device object.

Example:

EnOcean/0185408E/post/devices/Opus-Bridge-2K

JSON for device object

```
"device" : {
    "deviceId" : "01910188",
    "friendlyId" : "Opus-Bridge-2K",
    "eeps" : [ {
      "eep" : "D2-01-11",
      "version" : 0.9,
      "direction" : "both"
    }, {
      "eep" : "F6-02-01",
      "version" : 0.9,
      "direction" : "from"
    }, {
      "eep" : "F6-03-01",
      "version" : 0.9,
      "direction" : "from"
    } ],
    "manufacturer" : "Jaeger Direkt"
  }
```

Figure 19: adding a device JSON example

EnOcean/0185408E/stream/devices/Opus-Bridge-2K

JSON for device object

```
"device" : {
    "deviceId" : "01910188",
    "friendlyId" : "Opus-Bridge-2K",
    "eeps" : [ {
      "eep" : "D2-01-11",
      "version" : 0.9,
      "direction" : "both"
    }, {
      "eep" : "F6-02-01",
      "version" : 0.9,
      "direction" : "from"
    }, {
      "eep" : "F6-03-01",
      "version" : 0.9,
      "direction" : "from"
    } ],
    "manufacturer" : "Jaeger Direkt"
  }
```

Figure 20: adding a device JSON example

Only in case of an error, the device publishes an error object under the topic:

EnOcean/0185408E/postAnswer/devices/Opus-Bridge-2K

JSON for error object

```
"error" : 404
```

Figure 21: error JSON example

3.4.2. Use case add a system information parameter to EAG

This use case is for adding a new system information parameter to an EAG. It will be published under the topic:
EnOcean/{EAG-Identifier}/post/config/{REST path}
The payload is a manufacturer specific JSON object.

Only in case of an error, the device publishes an error object under the topic
EnOcean/{EAG-Identifier}/postAnswer/config/{REST path}
The payload is an error object.

3.5. The delete topic

The following topic examples are not mandatory to be implemented.

3.5.1. Use case delete a device from EAG

This use case is for deleting an existing device from an EAG. The command will be published under the topic:
EnOcean/{EAG-Identifier}/delete/devices/{Device-Identifier}
There is no payload needed.

To delete the device from the devices list, the EAG publishes with no payload under the topic:
EnOcean/{EAG-Identifier}/stream/devices/{Device-Identifier}
There is no payload needed.

Only in case of an error (e.g. deleting a device which is non-existent), the EAG publishes an error object under the topic:
EnOcean/{EAG-Identifier}/deleteAnswer/devices/{Device-Identifier}
The payload is an error object.

3.5.2. Use case delete a system information parameter from EAG

This use case is for deleting an existing system information parameter from the EAG. It will be published under the topic:
EnOcean/{EAG-Identifier}/delete/config/{REST path}
The payload is a manufacturer specific JSON object.

Only in case of an error, the device publishes an error object under the topic
EnOcean/{EAG-Identifier}/deleteAnswer/config/{REST path}
The payload is an error object.

3.6. The Last Will and Testament topic

In case of an unexpected exit the EAG has to configure the last will and testament option of the MQTT-Broker. It has to decide whether it wants to specify an LWT message or not. If it does then it needs to provide the topic name, the message payload, the QoS and the Retain flag.

All these decisions are entirely up to the EAG developer.