

Input paper for the following Committee(s):	check as appropriate	Purpose of paper:
<input type="checkbox"/> ARM	<input checked="" type="checkbox"/> ENG	<input checked="" type="checkbox"/> Input
<input checked="" type="checkbox"/> DTEC	<input type="checkbox"/> VTS	<input type="checkbox"/> Information
<input checked="" type="checkbox"/> PAP		

Agenda item ² 6.7

Technical Domain / Task Number ²

Author(s) / Submitter(s) Jonas Lindberg

Harmonized Visual AtoN IoT protocol

1 SUMMARY

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, GSM 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 communication sessions they can manage, so reporting frequency is not only limited by outstation constraints.

As a result, the owner of the asset always has outdated information and no real-time situational awareness.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.

In this paper, we will demonstrate how modern and true IoT (Internet of Things) technology can be implemented to overcome all the current limitations and issues.

We will demonstrate that we are able to resolve two of the main issues in existing remote monitoring technologies;

1. Implementing a new, open, secure and standardized non-proprietary communication protocol used by a huge number of existing IoT devices, and
2. Utilizing modern IoT platforms like LTE-M and LoRaWAN achieving communication close to real time without driving data costs and energy consumption

This new method enables the Marine Signal lanterns to enter the real IoT era we have seen moving quickly into other industrial fields.

1.1 Purpose of the document

This paper could be used as a starting point to create a harmonized communication protocol for internet connected visual aids to navigation

¹ Input document number, to be assigned by the Committee Secretary

² Leave open if uncertain

2 STANDARDIZED PROTOCOL

There are various communication protocols used in IoT and IIoT (Industrial IoT). The most appropriate protocol for a particular case depends on factors such as data rate, security, power consumption, compatibility, and complexity.

For the Visual AtoN IoT harmonization, the MQTT (Message Queuing Telemetry Transport) protocol is proposed as the best suited protocol. This protocol is optimal for small low-power applications and is one of the most commonly used communication protocols. It can be implemented in ultra low-power microcontroller applications, and on the server side, the MQTT broker is straightforward to establish and integrates well with an already established remote monitoring system.

2.1 MQTT

MQTT is a lightweight publish/subscribe messaging protocol designed for use in low-bandwidth, high-latency, or unreliable network environments. It was originally developed by IBM in the late 1990s and has since become an open standard maintained by the OASIS consortium.

The MQTT protocol operates on top of TCP/IP and uses a publish/subscribe messaging model, where publishers send messages to a broker, which then distributes those messages to interested subscribers. The broker acts as an intermediary between publishers and subscribers, allowing messages to be sent and received even if the publisher and subscriber are not connected at the same time.

Here is how the MQTT protocol works in more detail:

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 mediator 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 topic.
4. **Quality of Service (QoS):** MQTT supports three levels of Quality of Service (QoS) for message delivery: QoS 0, QoS 1, and QoS 2. QoS 0 provides at most once delivery, QoS 1 provides at least once delivery, and QoS 2 provides exactly once delivery.
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 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 client to specify a message that will be published by the broker in the event that the client becomes disconnected unexpectedly.

In summary, MQTT is a lightweight publish/subscribe messaging protocol that enables efficient communication between devices in low-bandwidth, high-latency, or unreliable network environments. It uses a broker to mediate communication between publishers and subscribers, and supports a range of QoS levels, Keep Alive mechanism, and LWT feature for reliable and fault-tolerant messaging.

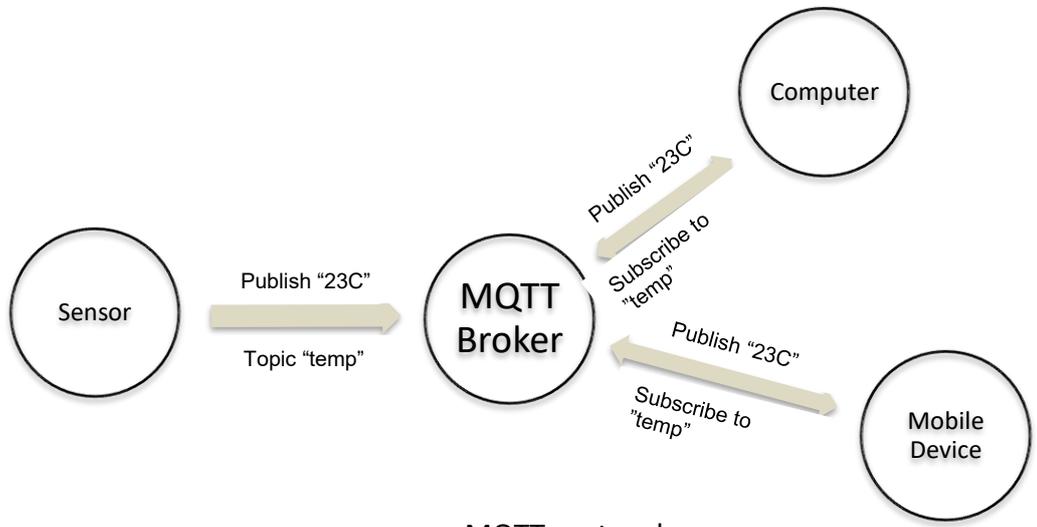


Figure 1: MQTT protocol

3 SECURITY

MQTT is a protocol that runs over TCP/IP and does not provide security measures itself, but it can be implemented with security measures to ensure secure communication between the client and the broker.

3.1 Encryption and authentication

MQTT does support the use of security measures such as SSL/TLS encryption, which provides a secure communication channel between the client and the broker. When SSL/TLS encryption is enabled, all data transmitted between the client and the broker is encrypted and cannot be intercepted by a third party.

Additionally, MQTT supports authentication mechanisms, such as username and password, to ensure that only authorized clients can connect to the broker. Access control lists (ACLs) can also be used to restrict the operations that clients can perform on the broker.

In this proposal, we recommend to restrict communication to only use TLS 1.2 or higher and the dedicated Secure-MQTT port 8883.

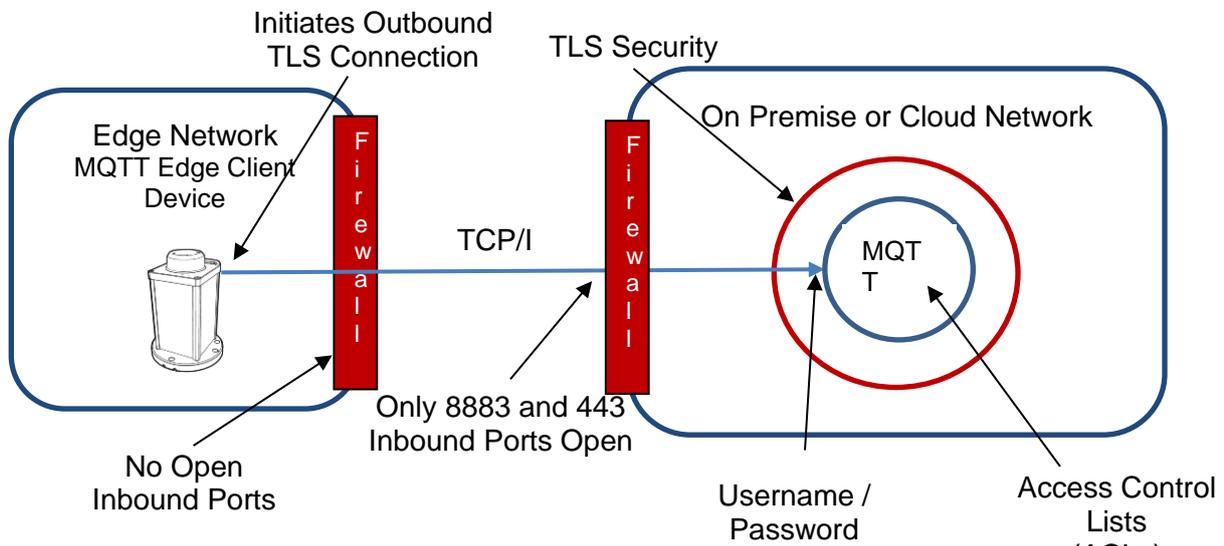


Figure 2: MQTT Security, Courtesy of Cirrus Link

3.2 One way or two way communication

For extremely critical infrastructure of Aids to Navigation (AtoN), or when remote commands are considered unnecessary, the implementation can be limited to only offer inbound messages from the remote AtoN. In such cases, the asset would not be capable of processing remote commands. This can be seen as analogous to the difference between AIS Type 1 and Type 3, where Type 1 does not include a receiver.

4 MESSAGE STRUCTURE

It is not enough to only standardize on using MQTT as the connectivity protocol. In order to reach full compatibility between different providers it is necessary to use a common message structure in the payload. The proposed structure is based upon JSON (JavaScript Object Notation).

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. The main reasons to use JSON, is that JSON is a platform-independent data format, which means it can be used with any programming language or platform.

JSON is a well-established format that is widely used for data-interchange, with support for parsing and generating JSON built into many programming languages and frameworks.

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

```
{
  "firstName": "John",
  "lastName": "Smith",
  "isAlive": true,
  "age": 27,
  "address": {
    "streetAddress": "21 2nd Street",
    "city": "New York",
    "state": "NY",
    "postalCode": "10021-3100"
  },
  "phoneNumbers": [
    {
      "type": "home",
      "number": "212 555-1234"
    },
    {
      "type": "office",
      "number": "646 555-4567"
    }
  ],
  "children": [
    "Catherine",
    "Thomas",
    "Trevor"
  ],
  "spouse": null
}
```

Figure 3: Example of JSON syntax, <https://en.wikipedia.org/wiki/JSON>

Please find the complete proposed communication protocol in Appendix A, Draft Visual AtoN MQTT Protocol.

5 WORKING WITH DIFFERENT COMMUNICATION PLATFORMS

In cases where the AtoN is connected using TCP/IP, MQTT can be implemented in the AtoN itself with direct MQTT communication to the Broker. The new GSM standard for IoT devices, LTE-M (Long Term Evolution for Machines), is perhaps the most suitable for this purpose. Low cost and low-power LTE-M modules capable of native encrypted MQTT are already widely available.

Offshore, or in other places where internet connection is not an option, there are many indirect possibilities using a proprietary network with an added MQTT gateway to relay the messages to the MQTT IoT platform. Examples of this are iridium, inmarsat, Globalstar and LoRaWAN / Sigfox to name a few.

It's not necessary to build such integration from scratch. There are both on premises and cloud based systems available for fast integration. An example of a solution with LoRaWAN to MQTT using HiveMQ broker is shown in the Figure 4 below.

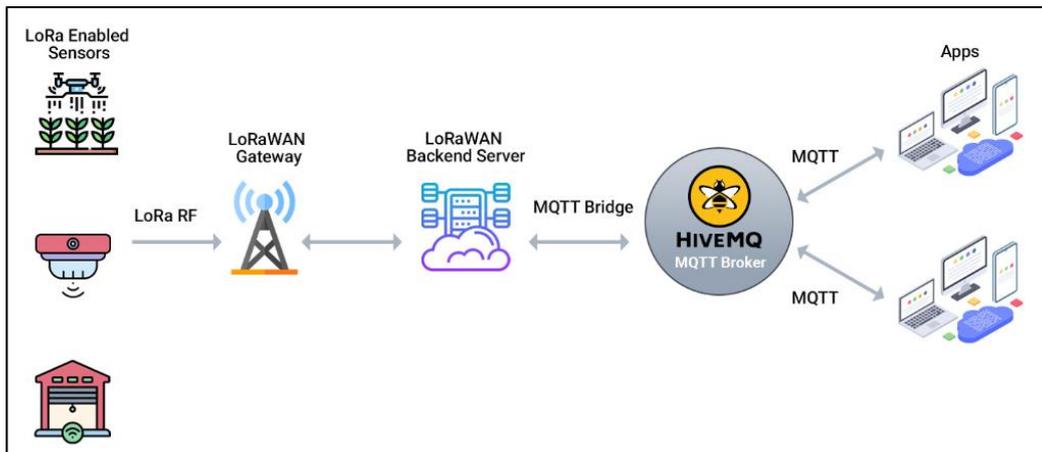


Figure 4: LoRaWAN MQTT Broker, HiveMQ GmbH

<https://www.hivemq.com/blog/lorawan-and-mqtt-integrations-for-iot-applications-design/>

6 CONCLUSIONS

MQTT is a lightweight publish/subscribe messaging protocol that enables efficient and frequent communication between devices in low-bandwidth, high-latency, or unreliable network environments. It supports SSL/TLS encryption enabling secure and authenticated communication. Used in conjunction with JSON, which is a platform-independent data format, it can be used with any programming language or platform.

Both MQTT and JSON are non-proprietary and royalty free. Many open source platforms and libraries are available to support easy integration/adaptation.

IALA is invited to develop this protocol and message structure as the standard IALA Visual AtoN Protocol to use so that members with a mixed network from various Industrial Members have the ability to monitor them in a single platform.

7 APPENDIX A, DRAFT VISUAL ATON MQTT PROTOCOL

7.1 Topic names

Each topic contain least four fields and optional device name:

1. First level topic name describe topic purpose, e.g. telematics topic, information topic, etc. First level topic names are described below.
2. Second and third level topic contain location or region, eg. estonia/tallinn
3. Fourth level topic contain site name, eg. soderskar-lighthouse
4. Fifth level contain device name or identificator

First level topic names:

1. 'tele' – telemetry information. Device issued automatic monitoring data.

Example:

```
tele/<location1>/<location2>/<site name>  
tele/<location1>/<location2>/<site name><device x>
```

2. 'info' – device information. Device ID, capabilities. Etc.

Example:

```
info/<location1>/<location2>/<site name>  
info/<location1>/<location2>/<site name>/res  
info/<location1>/<location2>/<site name><device x>  
info/<location1>/<location2>/<site name><device x>/res
```

3. 'cmd' – command, can trigger telemetry response

Example:

```
cmd/<location1>/<location2>/<site name>  
cmd/<location1>/<location2>/<site name>/res  
cmd/<location1>/<location2>/<site name><device x>  
cmd/<location1>/<location2>/<site name><device x>/res
```

4. 'parameter' – get or set parameter(s)

Example:

```
parameter/<location1>/<location2>/<site name>  
parameter/<location1>/<location2>/<site name>/res  
parameter/<location1>/<location2>/<site name><device x>  
parameter/<location1>/<location2>/<site name><device x>/res
```

5. 'direct' – device specific data that can be used for implement custom protocols inside device, like Modbus.

Example:

```
direct/<location1>/<location2>/<site name>  
direct/<location1>/<location2>/<site name>/res  
OR  
direct/<location1>/<location2>/<site name><device x>  
direct/<location1>/<location2>/<site name><device x>/res
```

Subscribe example:

1. Subscribe to all telemetry messages

tele/#

2. Subscribe to all telemetry messages in region named 'Country1'

tele/country1/#

3. Subscribe to all telemetry messages in region named 'country1/north-territory'

tele/country1/north-territory/#

4. Subscribe to telemetry messages which come from 'Lighthouse1' in 'country1/north-territory' region

tele/country1/north-territory/Lighthouse1/

5. Subscribe to telemetry messages which issued by telematics module, and flasher1 and flasher2

tele/country1/north-territory/Lighthouse1/telematics

tele/country1/north-territory/Lighthouse1/flasher1

tele/country1/north-territory/Lighthouse1/flasher2

7.2 Communication sequence

Every connection should subscribe to following topics:

- Command ('cmd' topic)
- Parameter

7.2.1 Telemetry information

- Device connect to broker and send:
 - Telemetry packet. Depending on configuration this may be repeated during session
- Device disconnect from broker

7.2.2 Info

- Device connect to broker, and send following data
 - Subscribe for topics
 - Telemetry packet
 - Server ask info
 - Device send info topic
 - Server send done
- Device disconnect from broker

7.2.3 Set parameters

- Device connect to broker, and send following data
 - Subscribe for topics
 - Telemetry packet
 - Server ask parameter
 - Device send parameter
 - Device update parameter
 - Server send done

- Device disconnect from broker

7.3 Payload format

Payload have two different formats: unencrypted JSON format, and encrypted format.

	Byte 0	Byte 1	Byte n
JSON	'{'	Unencrypted JSON data	
Encrypted data	'E'	Unused, should be 0	Encrypted binary data

When unencrypted JSON data is used in MQTT payload, only plain JSON messages transmitted, no data is added. JSON data should be follow format published in <https://www.json.org/json-en.html>.

When payload is encrypted, then first byte must be 'E' (ASCII 0x45) and second byte is reserved and should have value 0x00. All subsequent bytes are encrypted JSON data. Unencrypted payload must be multiple of 16 (AES requirement), it is recommended to fill unused bytes with random data after terminating 0x00 in JSON string.

7.4 Payload data

7.4.1 General payload data rules

1. Payload is in JSON format (<https://www.json.org/json-en.html>).
2. Property names are only ASCII.
3. Maximum property name length is 32 characters.
4. In property names are allowed only lower case letters ('a' – 'z') and numbers. '-' (minus) is used to separate words in property names.
5. Property values are UTF-8 encoded strings.
6. Not recommended property name is "class".
7. User defined properties are allowed, but must follow above listed limits.
8. 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.

7.4.2 Topic 'tele' – status information

Minimal JSON message for generic device:

```
{
  "session-id": "session-1",
  "status": "ready",
  "uptime": 20
}
```

6.4.2.1 Session ID

- This property is mandatory
- Property name: 'session-id'
- Description: can be monotonic counter, e.g. timer or session counter
- Example: "session-id": "session-820923084792"

6.4.2.2 Status

- This property is mandatory
- Property name: 'status'
- Allowed values: init, ready, alert, suspend
 - init – least one component to initialize, default value
 - ready – system is fully functional, 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

6.4.2.3 Uptime

- This property is mandatory
- Property name: 'uptime'
- Seconds from last boot. Default value is 0.
- Example: "uptime": 211

6.4.2.4 Time

- Required only on devices with RTC clock
- Property name: 'time'
- Description: device UTC time, default value is 0
- Allowed values: seconds from January 1st, 1970 at UTC (UNIX time). Only positive values allowed.
- Example: "time": 1673564596

6.4.2.5 Alert status

- Only required when device status is 'alert'
- Property name: 'alert'
- Description: list of alert statuses. If device operates normally and don't have any alerts, then this property may be omitted or set value to 'none'. Possible values:
 - none, default value
 - light-fail
 - low-battery
 - gnss-error
 - off-location
 - overheated
 - etc...
- Example, two alerts: "alert": ["low-battery", "off-location"]

6.4.2.6 Beacon status

- Mandatory only on beacons. 'type' field in info message should be 'beacon'.
- Property name: 'beacon-status'
- Allowed values:
 - on-main-character – main or night character
 - on-alternative-character – alternative or day character
 - off – not flashing, default value
- Example: "beacon-status": "on-main-character"

6.4.2.7 Device temperature

- Only on devices which have temperature sensor.
- Property name: 'temperature'
- Contains following sub properties:
 - 'last' – last read temperature
 - 'max' – maximum temperature in last 24 hours, optional
 - 'min' – minimum temperature in last 24 hours, optional
 - 'avg' – average temperature in last 24 hours, optional
- Allowed values: floating point value
- Example, valid temperature: "temperature": {"last": 21.0, "max": 25, "min": 19, "avg": 22}

6.4.2.8 Battery voltage

- Only on devices which can measure battery voltage
- Property name: 'voltage'
- Contains following sub properties:
 - 'average' – last battery voltage, averaged 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
- Allowed values: positive floating point value
- Example, valid battery voltage: "voltage": {"average": 12.3, "loaded": 12.1, "unloaded": 12.5, "max": 13.3, "min": 11.5}

6.4.2.9 Location

- Only on devices with GNSS receiver
- Property name: 'position'

- Description: JSON array of floating point values with last GNSS position, exact method how this value is computed is implementation defined
- Allowed values: degrees for latitude -90.0/90.0, and for longitude -180.0/180.0
- Example: "position": [60.0, 110.0]

6.4.2.10 Deviation

- Only on devices with GNSS receiver
- Property name: 'position-deviation'
- Description: Floating point value of deviation in meters from GNSS fixed position. Exact method how this value is computed is implementation defined
- Allowed values: positive floating point value
- Example: "position-deviation": 10.0

6.4.2.11 Last GNSS fix time

- Only on devices with GNSS receiver
- Property name: 'position-time'
- Description: Floating point value of UTC time derived from GNSS.
- Allowed values: seconds from January 1st, 1970 at UTC (UNIX time). Only positive values allowed.
- Example: "position-time": 1673564596

6.4.2.12 GNSS quality indicator

- Only on devices with GNSS receiver
- Property name: 'position-quality'
- Description: Floating point value of CSQ value
- Allowed values: only positive floating point values allowed
- Example: "position-quality": 1.1

6.4.2.13 Ambient light level measured by the light sensor

- Only on devices that can measure ambient light level.
- Property name: 'ambient-light-level'
- Description: Ambient light level measured in lux
- Allowed values: only positive floating point values allowed
- Example: "ambient-light-level": 30

6.4.2.14 Network statistics

- Optional. If present must contain all sub properties
- Property name: 'network-statistics'
- Description: contains nested statistics information
- Allowed values: positive value, default value is 0

- Example: "network-statistics": {"succeeded-server-connections": 2, "failed-server-connections": 0, "succeeded-network-logins": 2, "failed-network-logins": 2}

Amount of succeeded server connections

- Property name: 'succeeded-server-connections'
- Description: number connections between server and broker that have valid end.
- Allowed values: positive value, default value is 0
- Example: "succeeded-server-connections": 2

Amount of failed connections

- Optional. All network statistics data should be present in together: 'succeeded-server-connections', 'failed-server-connections', 'succeeded-network-logins' and 'failed-network-logins'.
- Property name: 'failed-server-connections'
- Description: number connections between server and broker that have no valid end.
- Allowed values: positive value, default value is 0
- Example: "failed-server-connections": 0

Amount of succeeded network logins

- Optional. All network statistics data should be present in together: 'succeeded-server-connections', 'failed-server-connections', 'succeeded-network-logins' and 'failed-network-logins'.
- Property name: 'succeeded-network-logins'
- Description: number of succeeded network logins
- Allowed values: positive value, default value is 0
- Example: "succeeded-network-logins": 2

Amount of failed network logins

- Optional. All network statistics data should be present in together: 'succeeded-server-connections', 'failed-server-connections', 'succeeded-network-logins' and 'failed-network-logins'.
- Property name: 'failed-network-logins'
- Description: number of failed network logins
- Allowed values: positive value, default value is 0
- Example: "failed-network-logins": 2

6.4.2.15 Last reset source

- Optional
- Property name: 'last-reset-source'
- Description: number of resets starting from production
- Allowed values:
 - por – power on reset
 - wdr – watchdog reset
 - rst – reset from external reset signal (HW signal)
 - bor – brown-out reset
 - usr – reset triggered by command, e.g. SMS reset command
 - other – all other reset sources
- Example: "last-reset-source": "wdt"

6.4.2.16 Reset count

- Optional and only when last reset source is present. This property have nested properties with reset names defined in Last Reset sources.
- Property name: 'reset-count'
- Allowed values: positive value, default value is 0
- Example: "reset-count": {"por":30, "wdr":1}

7.4.3 Topic 'info'

Info is triggered by 'cmd' topic "send": "info"

6.4.3.1 Protocol version

- This property is mandatory.
- Property name: 'protocol-version'
- Integer to describe protocol version. Currently supported value is 1.
- Example: "protocol-version": 1

6.4.3.2 Type

- This property is mandatory
- Property name: 'type'
- Description: device type class
 - 'group' – logical container for device group
 - 'beacon' – for beacons
- Example: "type": "beacon"

6.4.3.3 System information

- This property is mandatory. This information can be used to set up optimal set/get transmission packet sizes.
- Property name: 'sys-info'
- Description: system parameters
 - 'rx-buf' – size of rx buffer in bytes, -1 means infinite
 - 'tx-buf' – size of tx buffer in bytes, -1 means infinite
- Example: "sys-info": [{"rx-buf": 512}, {"tx-buf": 512}]

6.4.3.4 Serial number

- This property is mandatory for non group devices
- Property name: 'serial-nr'
- Description: device serial number, this number may contain product code also, if product code and serial number are not related then 'product-code' property show product code

6.4.3.5 Product code

- Required only when serial number does not have product information
- Property name: 'product-code'
- Description: device product code

6.4.3.6 Firmware version

- This property is mandatory for non-group devices
- Property name: 'firmware-version'
- Description: device firmware version

6.4.3.7 Component info

- Only for device which have components with own version
- Property name: 'component-info'
- Description: device component version list, like onboard GNSS receiver.
- Example: "component-info": [{"gnss-version":"1.0"}, {"gnss-type":"NEO M8N"}]

6.4.3.8 Limits

- Optional property. Not required to list all parameters.
- Property name: 'limits'
- Description: returns list of device limit values, for example maximum allowed light intensity, maximum battery voltage, etc. Limit values and value names must match with same configuration parameters.
- Example: "limits": [{"light-intensity":1000}, {"low-voltage-level":6.0}]

7.4.4 Topic 'cmd'

This topic is for server initiated actions.

6.4.4.1 Send

- Required
- Property name: 'send'
- Description: send requested topic
- Allowed values:
 - tele – for telematics
 - info – for information
- Example: "send": "info"

6.4.4.2 Reset

- Property name: 'reset'
- Description: reset device or parameter
- Allowed values:
 - null, empty string or 'reset' – reset device (required)
 - parameters – reset all parameters (optional)
 - <parameter-name> – reset parameter name to default (optional)
- Example: "reset": null

6.4.4.3 Done

- Required
- Property name: 'done'
- Description: this is hint from server, that server has been completed all tasks and controller is free to disconnect from broker. It depends on client configuration if it disconnects immediately or send telematics packets. If server send packet after 'done' message then for disconnect is needed re-send 'done' message. If server does not send 'done' message then controller can disconnect if last message from server was more than X seconds ago.
- Allowed values: any string
- Example: "done": "ok"

6.4.4.4 Light on demand

- Optional
- Property name: 'light-on-demand'
- 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.
- Contains following sub properties:
- 'timeout' – last battery voltage, averaged over one flash cycle
 - -1 – light on demand is active until switched off
 - 0 – light on demand switched off
 - 1...2147483648 – seconds active
- 'intensity' – effective intensity in cd, optional
- Example: "light-on-demand": {"timeout": 3600, "intensity": 300}
- Light activated for one hour with 300 cd effective intensity

6.4.4.5 Fix position

- Optional
- Property name: 'fix-position'
- Description: start or stop GNSS position fix
- Allowed values:
 - start – start position fix
 - stop – stop position fix
- Example: "fix-position": "start"

7.4.5 Topics 'parameter'

Set or get configuration parameter. All get commands must have 'res-topic' property. All topic queries have parameter value null.

Get

```
{
  "res-topic": "get/locationa/locationb/site/device1/res",
  "time": null
  "light-intensity": null
}
```

Response to get/locationa/locationb/site/device1/res

```
{
  "session-id": "session-1",
  "time": 1677677908
  "light-intensity": 34
}
```

Set

```
{  
  "res-topic": "get/locationa/locationb/site/device1/res",  
  "time": 1677679999  
  "light-intensity": 30  
}
```

6.4.5.1 Date and time

- Property name: 'time'
- Description: set or get time
- Allowed values:
 - null – query from server
 - any positive number – set or get result

6.4.5.2 Light intensity

- Property name: 'light-intensity'
- Description: set or get effective light intensity in candelas.
- Allowed values:
 - null – query from server
 - any positive number to max allowed value – light intensity in candelas

Note: Max allowed 'light-intensity' is retrieved with the 'info' Topic.

6.4.5.3 Ambient light threshold

- Property name: 'ambient-light-threshold'
- Description: set or get ambient light threshold levels in lux. Minimum ambient light level triggering beacon activation.
- Allowed values:
 - null – query from server
 - any positive number – light level in lux

6.4.5.4 Maximum allowed distance from fix position

- Property name: 'distance-from-fix'
- Description: set or get distance from fix position
- Allowed values:
 - null – query from server
 - any positive floating point number – distance from fix

6.4.5.5 Latitude and longitude of fix position

- Property name: 'fix-position'
- 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
 - degrees for latitude -90.0/90.0, and for longitude -180.0/180.0

6.4.5.6 Telemetry

- Property name: 'telemetry'
- Description: common property for telemetry
- Allowed sub properties
 - report-mode – telemetry report mode
 - report-period – telemetry report period

Telemetry report mode

- Property name: 'report-mode'
- Description: set or get telemetry report mode
- Allowed values:
 - null – query from server
 - off – telemetry data is sent only after query with 'cmd'
 - utc-fixed – UTC fixed mode, for example 00:00, 00:05, 00:10, To spread simultaneous sessions, can be added delay to this period. 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
 - on-failure – only when error condition is detected

Telemetry report period

- Property name: 'report-period'
- 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

APN

- Property name: 'apn'
- Description: set or get APN
- Allowed values
 - null – query from server
 - string – APN name

APN user

- Property name: 'apn-user'
- Description: set or get APN
- Allowed values
 - null – query from server
 - string – APN user name

APN password

- Property name: 'apn-password'
- Description: set or get APN
- Allowed values
 - null – query from server
 - string – APN password

Broker address

- Property name: 'broker-address'
- Description: set or get broker address. Currently used address will not be changed.
- Allowed values
 - null – query from server
 - list of addresses with port, may contain current address

6.4.5.7 Low voltage level

- Property name: 'low-voltage-level'
- Description: set or get low voltage level

- Allowed values:
 - null – query from server
 - any positive floating point number – voltage level

6.4.5.8 GNSS

- Property name: 'gnss'
- Description: common property for GNSS
- Allowed sub properties
 - base – GNSS wakeup base
 - interval – interval of GNSS time and position checkup
 - duration – duration of GNSS time and position checkup
 - sync – flash code synchronization base

GNSS base

- Property name: 'base'
- Description: select GNSS wakeup base
- Allowed values:
 - async – not synchronized with other threads (this is default)
 - pre-telematics – always before telematics (optional). In this mode GNSS start 'duration' seconds before telematics and after task is completed then start telematics. If interval is set then GNSS started before telematics and repeated with interval. If telemetry is disabled, then this option does not have any effect.
 - utc – synchronized with UTC (optional)

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

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

GNSS sync

- Property name: 'sync'
- Description: set or get GNSS-guided synchronization of flash code
- Allowed values:
 - off – GNSS sync disabled
 - on – GNSS sync is enabled, this is equal with 'utc'
 - utc – UTC based GNSS sync (start of flash adjusted to UTC 00:00:00)
 - gps – GPS time based GNSS sync (start of flash adjusted to GPS 00:00:00)

6.4.5.9 Beacon flash character

- Property name: 'flash-code'
- Description: set or get beacon flash character
- Allowed values:
 - null – flashing is disabled
 - "flash-code": {"main": [{"on": "ms"}, {"off": "ms"}, ...], "secondary": [{"on": "ms"}, {"off": "ms"}, ...]}

6.4.6 Topic 'direct'

Direct commands to device. For example Modbus packets.

- Only single device
- Must contain 'res-topic' and 'data' properties
 - 'res-topic' property describe response topic name
 - 'data' property contains data in device specific format
- Example: {"res-topic": "direct/locationa/locationb/site/device1/res", "data": "10AB"}

Useful references:

- <https://github.com/kokke/tiny-ECDH-c>
- <https://stackoverflow.com/questions/6032675/diffie-hellman-test-vectors>
- <https://www.techiedelight.com/c-program-demonstrate-diffie-hellman-algorithm/>
- https://www.oryx-embedded.com/doc/dh_8c_source.html
- <https://github.com/parthendo/thrain>
- <https://github.com/terlan98/Diffie-Hellman-and-AES>
- https://github.com/ojan2021/AES_and_DiffieHellman_Implementation
- <https://www.programmingboss.com/2015/11/diffie-hellman-key-exchange-algorithm.html>
- <https://datatracker.ietf.org/doc/html/draft-irtf-cfrg-augpake-08>