Silvair APIs

Application note

20 July 2023	SN-221 rev. 2.4

www.silvair.com

agree to submit to the exclusive jurisdiction of, and venue in, the courts of Krakow, in any dispute arising out of or relating to this agreement. The application of the "United Nations Convention on Contracts for the International Sale of Goods" is hereby excluded. All required or permitted notices to Silvair under this document will be made in writing, make reference to this document, and be delivered by hand, or dispatched by prepaid air courier or by registered or certified airmail, postage prepaid, addressed as follows:

SILVAIR Sp. z o.o.
ul. Jasnogórska 44
31-358 Kraków
Poland

Table of contents

1. Introduction **5**
2. Accessing the APIs **6**
3. Authentication and authorization API **7**
4. Project data API **8**
4.1 Network topology 8
5. Scheduling API **9**
5.1 Retrieve a scheduled event 9
5.2 Delete a scheduled event 9
5.3 Modify a scheduled event 9
5.4 Create a scheduled event 10
5.5 Setting the fade time 10
5.5.1 Fade time for scheduling of scenes 10
5.5.2 Fade time for setting "Lightness" in a list of zones 10
6. Energy consumption API **11**
7. Occupancy API **12**
8. Monitoring and remote control APIs **13**
8.1 Framing 13
8.2 Message format 14
8.3 Error handling 15
8.4 Monitoring 16
8.5 Subscription patterns 16
8.6 Topic hierarchy 16
8.7 CapNProto format 17
8.7.1 CapNProto description file 17
8.7.2 Obtaining the latest CapNProto description file 17
8.8 Binary messages 18
8.9 Receiving messages 18
8.10 Sending remote control messages 19
8.11 Unicast control 21
9. Versioning **22**
10. Definition of a breaking change **23**
11. Document revisions **24**
Contact information **25**
Appendix A: Websocket client example **26**
Appendix B: Message details and samples **28**
B.1 Light lightness 28
LIGHT_LIGHTNESS_GET 28
LIGHT_LIGHTNESS_SET / LIGHT_LIGHTNESS_SET_UNACKNOWLEDGED 28
LIGHT_LIGHTNESS_STATUS 29
B.2 Scenes 29
SCENE_GET 29
SCENE_RECALL / SCENE_RECALL_UNACKNOWLEDGED 29

SCENE_STATUS 30
SCENE_REGISTER_GET 30
SCENE_REGISTER_STATUS 31
B.3 Generic on/off 31
GENERIC_ONOFF_GET 31
GENERIC_ONOFF_SET / GENERIC_ONOFF_SET_UNACKNOWLEDGED 31
GENERIC_ONOFF_STATUS 32
B.4 Generic level 32
GENERIC_LEVEL_GET 32
GENERIC_LEVEL_SET / GENERIC_LEVEL_SET_UNACKNOWLEDGED 33
GENERIC_LEVEL_STATUS 33
B.5 Sensor 34
SENSOR_STATUS 34
B.6 Health status 35
HEALTH_CURRENT_STATUS 35
HEALTH_FAULT_CLEAR / HEALTH_FAULT_CLEAR_UNACKNOWLEDGED 36
HEALTH_FAULT_GET 36
HEALTH_FAULT_STATUS 36
B.6.1 Faults 37
Appendix C: Project data details **39**

1. Introduction

The ability to use data generated by building services such as lighting systems is driving the development of smart buildings and the effective management of the resources they consume, as well as the comfort of the people using them.

Integration with external systems is one of the key features required by managers of large commercial buildings and multi-site operators. To this end, Silvair is developing and releasing a series of APIs that will allow our customers to access the data from their Bluetooth mesh lighting networks, and connect these networks with other building automation and management (BMS) systems.

The Silvair APIs are as follows:

- **Project data API** – for retrieving the project structure, including the network/application keys, etc.
- **Scheduling API** – for managing the scheduling of lighting events
- **Energy consumption API –** calculated from rated power, dimming level over time, and energy profiles that are entered during commissioning
- **Occupancy API** – based on sensor status
- **Monitoring API** – for low-latency light level and sensor status monitoring
- **Remote control API** – for low-latency remote control

All API methods are exposed as publicly-available HTTP endpoints or websockets.

2. Accessing the APIs

The APIs are made available on a per project basis. This must be activated by Silvair – contact support@silvair.com or your account manager for more details.

The following are required for the APIs to be available:

- The lighting installation must be commissioned using the Silvair Commissioning tools (that is Silvair mobile/web apps or authorized branded partner apps).
- A Silvair-enabled gateway must be added to the project and have an active internet connection.
- Access to the APIs is available only to users with “Owner” or “Manager” roles in the Silvair platform.[1]

The technical API documentation is available at: https://api.platform-prod.silvair.com/public/docs/. It should be read together with this document.

To access the APIs:

- Make sure that the APIs have been enabled for your project.
- Log in to your project on the web app.
- Use the /public/projects endpoint to obtain the list of your projects along with their IDs and names. The project ID is needed by all other endpoints.
- *For scheduling/monitoring/remote control:* use the project data API to retrieve the project topology, including the network and application keys, area and zone IDs, and device names and UUIDs.

See Appendix A for sample code for a websocket client.
See Appendix B for examples of remote control and monitoring messages.
See Appendix C for details of the data available via the project data API.

[1] Whoever creates the project is automatically made the owner and can invite other users to collaborate, assigning them the role of manager, installer, or end user. Ownership can be transferred to another user, but there can only be one owner of a project at any one time. Only the owner can delete a project. See the Silvair Commissioning user manual for more details.

3. Authentication and authorization API

To use the APIs, the client must be logged in to the project. The first step is to obtain a JSON authentication token (JWT) and then add the contents of the token to the authorization header of each subsequent request.

Use the same credentials that you use to log in to the Silvair Commissioning platform to obtain the token. If you are using the Silvair-branded tools, the partnerID is silvair. For branded tools, request the partnerID from the brand owner or your account manager at Silvair.

The API returns the JWT token. Use it in the authorization header of each subsequent request.

The token is valid for 10 hours. Reauthentication is required to refresh it when it expires.

4. Project data API

The project data API provides the topology of a Bluetooth mesh network that has been commissioned using the Silvair Commissioning tools. It is needed for all subsequent requests.

It also returns the list of projects to which you have access, and the collaborator roles assigned to you by the project owners. Calls can be tailored to a specified depth (the default is 'zones'), and values can be explicitly included or excluded.

4.1 Network topology

Returns a tree of the project structure (areas/zones/nodes). This can include:

- For areas:
  - Application and network keys
  - Floor plan image
- For zones:
  - Group addresses
  - Energy profile ID & settings (if one has been configured)
  - x/y coordinates of the zone as arranged on the floor plan
  - The scene/scenario parameters that have been commissioned for the zone (including profile settings, high/low end trim levels, timeouts, etc.)
- For nodes:
  - Node description including its UUID, the names assigned to it, certain protocol parameters, and the mesh features that can be supported by the node
  - The features that are active (that is, have been commissioned), such as if the node is functioning as a relay or has been set up as an EnOcean proxy, etc.
  - The list of elements[2] for each node and their unicast addresses

See Appendix C for more detail on the commissioned settings available via the project data API.

[2] See the Bluetooth Mesh Profile Specification to find out more about elements and how they are handled in the protocol.

5. Scheduling API

The scheduling API allows lighting event schedules to be retrieved, modified, and deleted.

Events can be triggered by the gateway (gateway scheduling) or by the node (in-node scheduling).

For in-node scheduling, the following conditions must be met:

- The device must be running firmware version 2.20 or later.
- The project must be updated to 20211 or later.

Events can be recalled as scenes as defined in the “multiple scenes” scenario (that is a constant brightness level, or automatic control based on occupancy with or without daylight harvesting) stored in the node or as a light output level set by the gateway.

Events can be scheduled for any day of the week, together with specified fade-in periods, and offsets (for example 30 minutes before sunrise).

There are three categories of time values (TIME, SUNSET, and SUNRISE) and two types of actions that can be executed at a specified time (scene or brightness).

At the moment we do not support holiday exceptions.

ℹ	For gateway-based scheduling events, TIME must be given in UTC. For in-node scheduling events, TIME is in the project local time.
ℹ	If the firmware and project version conditions for in-node scheduling are not met, the schedule will be created but the nodes will not be configured.
❗	Changes to in-node scheduling events require reconfiguration of the device on site. This can only be done via the mobile app. It cannot be done via the gateway.

5.1 Retrieve a scheduled event

Retrieve a list of scheduled events, or details of a single event.

5.2 Delete a scheduled event

Remove a single scheduled event.

5.3 Modify a scheduled event

Modify a single event, that is changing the event time, setting an offset (for example 30 minutes before sunrise/sunset), or extending a fade time, etc.

5.4 Create a scheduled event

Create a new scheduled event for selected zones.
There are two ways to specify zones that participate in the scheduled event:

- By providing a profile ID and the scene number to be recalled by nodes, in which case *all* zones in the project using that profile will be controlled by the event.

  This type of event is supported by **both** gateway-based and in-node scheduling.

- By providing the list of zones as zone IDs and the exact 'Lightness' output value to be set in nodes.

  This type of event is supported **only** by gateway-based scheduling.

Selecting zones by profile ID means that zones can be configured later to use a certain profile, and they will be affected by the previously created schedule. This will *not* be the case for explicitly specified zone IDs.

Only scheduled events created via the API where a profile ID is assigned will appear in the web app, although all scheduled events can be retrieved via the API regardless of how they are created.

The two methods can be distinguished by checking for the profile ID key in the object describing the schedule – if it's absent, the list of affected zones is static.

5.5 Setting the fade time

Fade time and transition time are handled differently in the Bluetooth Mesh Specification, depending on their length, and therefore our API also has to handle them differently.

5.5.1 Fade time for scheduling of scenes

Fade time can be set in 1 second increments within a range of 0-60 s. Any other values are ignored by the API and are not implemented.

5.5.2 Fade time for setting "Lightness" in a list of zones

Fade time in this case, according to the Bluetooth Mesh Specification, can be set between 0-37,800 seconds (10.5 hours) but the step resolution depends on the length of the chosen period. The values are discrete, due to the limitations of the specification (technically, 1 byte is used in this message: 6 bits for determining value (0-63) and 2 bits for determining the multiplier which can be 100 ms, 1 s, 10 s, or 10 minutes). At the moment, if the step resolution is not aligned with aforementioned values, an error message will be displayed.

If the value entered is not a whole number of increments the API response will be a 400 Bad Request Error that includes the closest possible values.

6. Energy consumption API

The energy consumption API returns the energy use of a project, area, or zone (group of devices) in kilowatt-hours (kWh), as a total or 15-minute aggregates, for a selected period. It includes a measure of the completeness of the data (sometimes also referred to as "reliability").

Energy data is stored for a maximum of 24 months as 15-minute aggregates. Aggregation intervals start on the hour and every 15 minutes (for example 10:00-10:15, 10:15-10:30, etc.).

The energy use is calculated from the energy profile of the lighting devices, which is created by measuring the energy used (in watts) at several lighting levels and entering these values into the platform manually. See *SN-207 Energy monitoring – energy profiles* for information about how to create the energy profile.

The created energy profile and the actual light levels recorded over time are used for calculation. We record the light level of each node every one minute, calculate its energy use, and then aggregate these values over 15 minutes. See *SN-206 Energy monitoring – calculation* for information about how we calculate energy use.

The completeness rate (reliability) is calculated as the number of minutes in which we receive information about lightness divided by the time period (in minutes).

The energy profiles set up for the project can also be downloaded via the API.

See our application notes for more details on energy profiles, how we calculate energy use, and how we handle missing data.

If more granular data is required, then use the monitoring API, which provides low-latency (real-time) brightness levels for each node, and perform the calculation based on the energy profile.

7. Occupancy API

The occupancy API returns the occupancy per zone (group of nodes) as 15-minute aggregates for a defined period (date/time).

Occupancy is recorded for each zone every 60 s and these values are aggregated over 15 minutes to give the average occupancy for that period, that is if occupancy detectors in a zone detect occupancy for ten 1-minute periods and do not detect occupancy for five 1-minute periods, then the reported occupancy for that 15-minute period is 66%.

Currently occupancy data is stored for a maximum of 3 months as 15-minute aggregates.

Aggregation intervals start on the hour and every 15 minutes past the hour (for example 10:00-10:15, 10:15-10:30, etc.).

We do not provide a measure of data completeness (reliability) for occupancy as there is no way of knowing whether a sensor is not reporting occupancy because it has failed or because the space is empty.

8. Monitoring and remote control APIs

The monitoring and remote control APIs allow low-latency monitoring of traffic in the network by subscribing to specific messages, as well as remote control of nodes/zones/areas in the network, that is:

- switch on/off,
- control dim level,
- recall a scene.

Note that the monitoring and remote control APIs have the following conditions:

- a gateway can only monitor/control nodes in the same area,
- one area can contain multiple gateways.

The Silvair platform allows web clients to send and receive mesh messages over a websocket connection. We provide a websocket endpoint for monitoring (subscriptions) and remote control.

To access the endpoint, authenticate using the same token as all the other endpoints (see Authentication and authorization API), and immediately upgrade the connection to a websocket.

The websocket handshake also includes versioning information and has the same meaning as in a regular REST API (see Versioning).

8.1 Framing

After authentication, a simple ZeroMQ-like framing over the websocket is used, that is:

- A single message consists of one or more parts.
- Each part is sent as a separate BINARY frame (we do not use other frame types).
- Each part starts with a 1-byte FLAG, followed by a PAYLOAD.
- FLAG is either 0x01 (MORE) if there are parts to follow, or 0x00 (LAST) if this is the last part of the message.
- FLAG is stripped from the payload.

Any protocol violations (for example sending multi-part messages where a single-part is expected, or using an unexpected payload) will cause the server to immediately terminate the connection with an appropriate code (see https://developer.mozilla.org/en-US/docs/Web/API/CloseEvent for the full list):

Code	Reason
1002	● Client sent the wrong number of frames. ● FLAG value is neither 0x00 nor 0x01.
1003	● Client sent a non-BINARY frame.
1007	Malformed message part: ● The first part contains an unknown message type. ● The first part contains more data than expected. ● The second part is not a valid subscription pattern.
1008	● Client sent an unsupported message type to the server. ● Client triggered too many errors.
1011	● Service internal error.

The termination may also contain a reason field where appropriate (see https://tools.ietf.org/html/rfc6455#section-5.5.1). If present, it contains a UTF-8 encoded JSON document with the error details.

For example, when the client sends a frame with the first byte set to 0x02 (instead of 0x00 or 0x01), the reason will appear as follows:

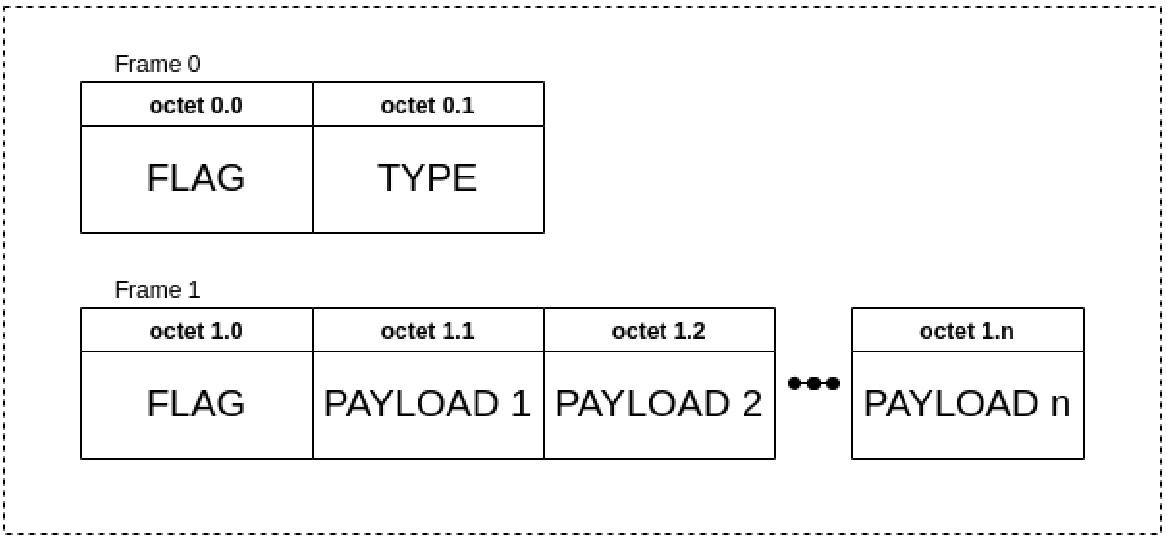
```
{
  "message": "Malformed FLAG"
  "extra": {
    "value": 0x02
  }
}
```

and it will be followed by all frames of the message that triggered the error, including its TYPE.

8.2 Message format

Each message consists of at least two parts: the header, and one (or more) body frames. The header is just a single byte denoting the message TYPE, as defined in this table below:

Value	Type	Direction	Body
0x00	SUBSCRIBE	client → server	UTF-8 string with subscription pattern (see Subscription patterns)
0x01	UNSUBSCRIBE	client → server	
0x02	RECEIVE	server → client	Binary encoded data
0x03	SEND	client → server	
0x04	ERROR	server → client	UTF-8 encoded JSON, optionally followed by a copy of message



8.3 Error handling

Non-terminal errors are reported as ERROR frames with the body containing a UTF-8 encoded JSON document describing the error name, message and (optionally) details:

```
{
  "code": "INVALID_ZONE",
  "message": "Zone does not exist",
  "extra": {
    "zoneId": "123abc"
  }
}
```

If the error has been triggered by the content of a message sent by the client, this message is appended after the error description. For example, if the client sends a malformed TOPIC in a SUBSCRIBE message, the ERROR message will be:

```
Frame 0:
   Flag:        01
   Payload:     04 (ERROR)
Frame 1:
   Flag:        01
   Payload:     {...} (binary encoded JSON error message)
Frame 2:
   Flag:        01
   Payload:     00 (SUBSCRIBE)
Frame 3:
   Flag:        00
   Payload:     25242423404021 (“%$$#@@!”)
```

And the error body will appear as follows:

```
{
  "code": "INVALID_TOPIC",
  "message": "Unexpected character in subscription topic",
  "extra": {
    "topic": "%$$#@@!"
  }
}
```

8.4 Monitoring

Connect to *wss://api.platform-prod.silvair.com/public/projects/{projectId}/mesh*

This endpoint allows a client to receive publications sent by mesh nodes to group addresses configured by the Silvair Commissioning platform but in a more friendly way, as we translate the raw mesh addresses to logical ones, denoting the area, zone, and publication group.

Subscriptions are not persistent, so the client needs to set them up on every (re)connection.

8.5 Subscription patterns

In order to start receiving publications, the client needs to configure the subscription. This is done by sending a two-part message, where the first part is `0x00` for SUBSCRIBE or `0x01` for UNSUBSCRIBE, and the second part is an UTF-8 encoded subscription TOPIC. See below for the topic hierarchy.

For example, to subscribe to PIR statuses in the whole project, the client sends two BINARY websocket frames with the following content:

```
Frame 0:
    Flag:          01
    Payload:       00 (SUBSCRIBE)
Frame 1:
    Flag:          00
    Payload:       232e53454e534f525f535441545553 (“#.SENSOR_STATUS”)
```

To unsubscribe, the client sends the same frame, but uses 0x00 as the single octet in the first part.

The client may subscribe to an unlimited number of topics.

8.6 Topic hierarchy

Messages are published to topic `<area>.<zone>.<group>.<opcode>`.

When subscribing, the client may substitute parts of the topic using either “*” (star) for a single word, or “#” (sharp) for multiple words, for example:

Topic	Description
#	Subscribe to everything.
#.LIGHT_LIGHTNESS_STATUS	Subscribe to Light Lightness statuses sent to any group in any zone.
5fbe343dcaef1700073a293c.#	Subscribe to all publications sent to any zone within the area 5fbe343dcaef1700073a293c.
`*.5fbe344ccaef1700073a2965.#`	Subscribe to all publications sent to zone 5fbe344ccaef1700073a2965.
`#.DEFAULT.*`	Subscribe to all publications sent to the DEFAULT group anywhere in the project.
`*.5fbe344ccaef1700073a2965.DEFAULT.LIGHT_LIGHTNESS_STATUS`	Subscribe to Light Lightness statuses sent to the DEFAULT group in zone 5fbe344ccaef1700073a2965.

Publication groups are:

- `DEFAULT`
- `SCENE_TRANSLATOR_LEFT`
- `SCENE_TRANSLATOR_RIGHT`
- `LIGHT_LIGHTNESS`
- `CCT`
- `LIGHT CONTROLLER`
- `PIR`
- `SCENE`

This hierarchy reflects information the client receives with each message's "destination" field.

`SCENE_TRANSLATOR_LEFT` and `SCENE_TRANSLATOR_RIGHT` are group addresses used by Silvair to assign scenes to left and right rockers of a non-mesh wall switch (that is EnOcean switch). We strongly recommend using `SCENE` to recall a scene instead.

8.7 CapNProto format

CapNProto is a fast serialization format which allows for effectively data packing to the binary form. Data structures are described in the special description file, which can be loaded with a dedicated library. This allows for easily packing and unpacking message payloads to and from objects format. CapNProto webpage: https://capnproto.org/.

8.7.1 CapNProto description file

The CapNProto description file contains high level data structure descriptions of all supported message payloads. It is provided in the text file format. It can be used with the most programming languages, but it requires an additional library. Most popular libraries are listed on the CapNProto webpage.

8.7.2 Obtaining the latest CapNProto description file

To ensure that the latest version of CapNProto description file is used, we recommend periodically polling the endpoint:

```
GET: https://api.platform-prod.silvair.com/public/control/message_definitions
```

This endpoint supports the ETag caching mechanism. This means that if you send a GET message with "If-None-Match" header set to the ETag value corresponding to your actual CapNProto description file version, you will get:

- HTTP status code 304 if the file is up-to-date.
- The latest file version with its contents, if there is a newer file.

8.8 Binary messages

The binary message consists of a header and the CapNProto packed payload added just after the header. The message header consists of fields listed in the table below. Fields are in the “little-endian” byte order.

Field name	Type	Description
timestamp	uint64_t	The Time UTC, milliseconds since epoch format.
source	uint16_t	In case of monitoring messages this field contains the originator address. For remote control messages this field is ignored (should be set to 0).
destination	uint16_t	Destination address for which message has been sent. It can be a node or group destination address.

A packed binary message example:

```
Raw values:
   timestamp: 1622553516863 (2021-06-01T13:18:36.863000)
   source: 1251
   destination: 1502
   payload={“opcode”: 33356, “lightLightnessSet”: {“minimal”: {“lightness”: 65535,
   “tid”: 34}}}

Packed binary header:
   “3fdbb9c779010000e304de05”

Packed payload (can be unpacked by the CapNProto library):
   “1004500101534c825801100107ffff22”

Packed binary message (in hex string format):
   “3fdbb9c779010000e304de051004500101534c825801100107ffff22” (header + payload)
```

8.9 Receiving messages

After subscribing, the client will start receiving RECEIVE messages. Each is a two-part message, where the first part contains a single byte 0x02 for RECEIVE, and the second is a binary encoded data with the message payload.

For example, subscribe to “*#.LIGHT_LIGHTNESS_STATUS*” to start receiving websocket frames that look like this:

```
Frame 0:

   Flag:        01

   Payload:     02 (RECEIVE)

Frame 1:

   Flag:        00

   Payload:     {...} (binary encoded data)
```

Where the binary encoded data will resemble:

```
    Binary encoded data:

      Data: c0b9c3ea79010000b369aa021004500101534e825a0110010000 (26 bytes)

      Data splitted:
             Header:  c0b9c3ea79010000b369aa02 (12 bytes)
             Payload: 1004500101534e825a0110010000 (14 bytes)

    Raw values:
      Header:
             Timestamp:   2021-06-08 08:38:37.215000
             Source:      27063
             Destination: 679

      Payload:
             {“lightLightnessStatus”: {“minimal”: {“presentLightness”: 0}}, “opcode”: 33358}
```

The available opcodes for remote monitoring are:

- `LIGHT_LIGHTNESS_STATUS (0x824E)`
- `SCENE_STATUS (0x5E)`
- `SCENE_REGISTER_STATUS (0x8245)`
- `GENERIC_ONOFF_STATUS (0x8204)`
- `GENERIC_LEVEL_STATUS (0x8208)`
- `SENSOR_STATUS (0x52)`
- `HEALTH_CURRENT_STATUS (0x04)`
- `HEALTH_FAULT_STATUS (0x05)`

8.10 Sending remote control messages

The client is able to send remote control messages just after opening the WebSocket. For example, to send a Lightness control message, send two frames over the websocket as below:

```
   Frame 0:

      Flag:          01

      Payload:       03 (SEND)

   Frame 1:

      Flag:          00

      Payload:       {...} (binary encoded data)
```

Where the binary encoded data will resemble:

```
    Binary encoded data:


       Data: 000000000000000000005bc11004500101534c82580110010415 (26 bytes)


       Data splitted:
             Header:  000000000000000000005bc1 (12 bytes)
             Payload: 1004500101534c82580110010415 (14 bytes)

    Raw values:
       Header:
             Timestamp:   0
             Source:      0
             Destination: 49499 (group address)


       Payload:
             {"lightLightnessSet": {"minimal": {"lightness": 0, "tid": 21}}, "opcode": 33356}
```

Where the TID field is a transaction identifier indicating whether the message is a new message or a retransmission of a previously sent message, as described in Section 3.4.1.2.2. of the Bluetooth Mesh Model Specification.

The available opcodes for remote control are:

- `LIGHT_LIGHTNESS_SET (0x824C)`
- `LIGHT_LIGHTNESS_SET_UNACKNOWLEDGED (0x824D)`
- `LIGHT_LIGHTNESS_GET (0x824B)`
- `SCENE_RECALL (0x8242)`
- `SCENE_RECALL_UNACKNOWLEDGED (0x8243)`
- `SCENE_GET (0x8241)`
- `GENERIC_ONOFF_SET (0x8202)`
- `GENERIC_ONOFF_SET_UNACKNOWLEDGED (0x8203)`
- `GENERIC_ONOFF_GET (0x8201)`
- `GENERIC_LEVEL_SET (0x8206)`
- `GENERIC_LEVEL_SET_UNACKNOWLEDGED (0x8207)`
- `HEALTH_FAULT_CLEAR (0x802F)`
- `HEALTH_FAULT_CLEAR_UNACKNOWLEDGED (0x8030)`
- `HEALTH_FAULT_GET (0x8031)`

See Appendix B for message examples.

8.11 Unicast control

It is possible to control individual devices via the API by directing a command to the element address of a selected node, instead of a predefined group address. Node addresses can be retrieved via the `/public/projects/{projectId}/tree` endpoint by setting 'depth' to 'nodes' and 'include' to 'nodesElements' – this returns the list of elements of each node together with their unicast addresses. See the Bluetooth Mesh Profile Specification for more information on elements and addressing.

It is important to note that both the number of elements and their function may differ across nodes, as they depend on a number of factors, such as the firmware version, enabled licenses, and in the case of dual-chip firmware – the way the host MCU registers Bluetooth mesh models. For details, please consult the device documentation, and/or the Bluetooth Mesh Profile Specification.

Most nodes based on Silvair firmware have a maximum of five (5) elements, with Light Lightness Server present on either element #0 or #1.

An example payload for getting Light Lightness level from a single node is as follows:

```
    Binary encoded data:

        Data: 000000000000000000000af691004500101534c82580110010728017e (28 bytes)

        Data splitted:
                Header:  0000000000000000000af69 (12 bytes)
                Payload: 1004500101534c82580110010728017e (16 bytes)

    Raw values:
        Header:
                Timestamp:   0
                Source:      0
                Destination: 27055 (unicast address)

        Payload:
                {"lightLightnessSet": {"minimal": {"lightness": 296, "tid": 126}}, "opcode": 33356}
```

9. Versioning

The Silvair APIs use versioning to check that the API client understands the response.

Additions to the API are not considered to be a new version, but API clients must take into account that responses may include additional information.

Silvair reserves the right to make changes to the API that create a new version. In this event, API users will be given a long notice period ahead of the change requiring a new version, and the old version will be supported for at least 6 months after the change.

10. Definition of a breaking change

A breaking change is a change in the information that has already been exposed – for example, the renaming of a property or the deletion of an endpoint.

Adding endpoints or individual properties at the resource level is **not considered** a breaking change and will not cause an API version change. API clients are expected to continue to work properly when new properties are added to the existing resources.

11. Document revisions

Revision	Date	Editor	Changes
2.4	20 July 2023	GM	Added B.6 Health status and B.6.1 Faults. Added units, resolutions, and values to sensor properties. Minor corrections.
2.3	22 March 2023	ES, GM	Added a table with sensor properties. Minor edits.
2.2	17 November 2022	GM	Changed the token expiration time from 24 h to 10 h. Added a note about the visibility of events created via API. Implemented template rev. 1.2.
2.1	26 January 2022	AS, PH	Correction in definition of completeness rate.
2.0	10 January 2022	ES	Updated retention policy for energy and occupancy APIs, introduced changes to project data API (topology). General editing.
1.0	29 July 2021	RO, ZZ	Updated the 4.1 Network Topology section.
0.9.7	29 June 2021	PF, ZZ	Added the 8.7.2 Obtaining the Latest CapNProto Description File section.
0.9.6	11 June 2021	PF, ZZ	Changing the message format to binary, impacted sections: ● 8. Monitoring and remote control API ● Appendix B: Message details and samples
0.9.5	24 May 2021	MLZ, PF	Added missing fields to message samples ("source" and "timestamp"). Added unicast control description.
0.9.4	20 May 2021	MLZ, PF	Changes to message details and samples, reference to the Bluetooth mesh specification.
0.9.3	12 April 2021	ES	Minor editing – added description of TID.
0.9.2	23 March 2021	ES, KJ, PF	Updated on limited release of remote control API.
0.9.1	26 January 2021	ES	Updated on limited release of monitoring API.
0.9	25 January 2021	ES	Beta release for external review and testing.

Contact information

Support:	**support@silvair.com**
Business development:	**business@silvair.com**
For more information please visit:	**www.silvair.com**

Our offices:

Europe	**North America**
ul. Jasnogórska 44	717 Market Street, Suite 100
31-358, Kraków	San Francisco, CA 94103
POLAND	USA

Appendix A: Websocket client example

```
#!python3
import asyncio
import getpass
import requests
import websockets
import capnp
import tempfile
import struct
from datetime import datetime

capnp.remove_import_hook()

DOMAIN = "api.platform-prod.silvair.com"
PARTNER_ID = "silvair"
PROJECT_ID = …
EMAIL = …
PASSWORD = getpass.getpass("Password:")

async def send_multipart(ws, *frames):
    *head, tail = frames
    for i in head:
        await ws.send(b"\x01" + i)
    await ws.send(b"\x00" + tail)

async def recv_multipart(ws):
    more, *frame = await ws.recv()
    frames = [bytes(frame)]
    while more:
        more, *frame = await ws.recv()
        frames.append(bytes(frame))
    return frames

response = requests.post(
    f"https://{DOMAIN}/public/auth/login",
    json=dict(partnerId=PARTNER_ID, email=EMAIL, password=PASSWORD)
)

token = response.json()["token"]

async def run():
    capnproto = requests.get(f"https://{DOMAIN}/public/control/message_definitions")
    with tempfile.NamedTemporaryFile(suffix='.capnp') as f:
        f.write(capnproto.text.encode())
```

```
            messages = capnp.load(f.name)

        async with websockets.connect(
            f"wss://{DOMAIN}/public/projects/{PROJECT_ID}/mesh",
            extra_headers=dict(Authorization=token)
        ) as connection:
            await send_multipart(connection, b"\x00", b"#.LIGHT_LIGHTNESS_STATUS")

            while True:
                frames = await recv_multipart(connection)

                if frames[0] != bytes([2]):
                    continue

                payload = frames[1]
                header, body = payload[:12], payload[12:]
                timestamp, src, dst = struct.unpack("<QHH", header)
                dt = datetime.fromtimestamp(timestamp / 1000)

                msg = messages.AccessMessage.from_bytes_packed(body)

                print(f"{dt}: {src:04x} -> {dst:04x} {msg.to_dict()}")

    asyncio.get_event_loop().run_until_complete(run())
```

Appendix B: Message details and samples

B.1 Light lightness

LIGHT_LIGHTNESS_GET

Fields	Description
-	-

Sample:

1. **`00000000000000000000af691003500101134b82570ffcffffff`**
   `{"timestamp": 0, "source": 0, "destination": 27055, "payload": {"lightLightnessGet": {}, "opcode": 33355}}`

LIGHT_LIGHTNESS_SET / LIGHT_LIGHTNESS_SET_UNACKNOWLEDGED

Fields	Description
`lightness`	integer, range 0-65535, target light lightness actual value
`tid`	integer, range 0-255, transaction identifier, **must be unique within 6 second range**
`transitionTime`	float, range 0-37200, time to reach the target lightness [seconds]
`delay`	float, range 0.0-1.276, command delay [seconds]

Samples:

1. **`00000000000000000000af691004500101534c825801100107ffff22`**
   `{"timestamp": 0, "source": 0, "destination": 27055, "payload": {"lightLightnessSet": {"minimal": {"lightness": 65535, "tid": 34}}, "opcode": 33356}}`
2. **`00000000000000000000af691005500101134c82581002c7ffff22f041083f`**
   `{"timestamp": 0, "source": 0, "destination": 27055, "payload": {"lightLightnessSet": {"optional": {"lightness": 65535, "tid": 34, "transitionTime": 30.0, "delay": 0.5}}, "opcode": 33356}}`
3. **`00000000000000000000af691004500101534d825901100107ffff22`**
   `{"timestamp": 0, "source": 0, "destination": 27055, "payload": {"lightLightnessSetUnacknowledged": {"minimal": {"lightness": 65535, "tid": 34}}, "opcode": 33357}}`
4. **`00000000000000000000af691005500101134d82591002c7ffff22f041083f`**
   `{"timestamp": 0, "source": 0, "destination": 27055, "payload": {"lightLightnessSetUnacknowledged": {"optional": {"lightness": 65535, "tid": 34, "transitionTime": 30.0, "delay": 0.5}}, "opcode": 33357}}`

LIGHT_LIGHTNESS_STATUS

Fields	Description
`presentLightness`	integer, range 0-65535, current light lightness actual
`targetLightness`	integer, range 0-65556, target light lightness actual during a transition
`remainingTime`	float, range 0-37200, remaining time to reach target lightness actual [seconds]

Samples:

1. **`d28785eb79010000af69a7021004500101534e825a01100103f8e4`**
   `{"timestamp": 1623154067410, "source": 27055, "destination": 679, "payload": {"lightLightnessStatus": {"minimal": {"presentLightness": 58616}}, "opcode": 33358}}`
2. **`8dc886eb79010000af69a7021004500101134e825a1001c32995f041`**
   `{"timestamp": 1623154149517, "source": 27055, "destination": 679, "payload": {"lightLightnessStatus": {"optional": {"presentLightness": 38185, "targetLightness": 0, "remainingTime": 30.0}}, "opcode": 33358}}`

B.2 Scenes

SCENE_GET

Fields	Description
-	-

Sample:

1. **`00000000000000000000af6910035001011341828a0ffcffffff`**
   `{"timestamp": 0, "source": 0, "destination": 27055, "payload": {"sceneGet": {}, "opcode": 33345}}`

SCENE_RECALL / SCENE_RECALL_UNACKNOWLEDGED

Fields	Description
`sceneNumber`	integer, range 1-15, scene to be recalled
`tid`	integer, range 0-255, transaction identifier, **must be unique within 6 second range**
`transitionTime`	float, range 0-37200, time to reach the target scene [seconds]
`delay`	float, range 0.0-1.276, command delay [seconds]

Samples:

1. **`00000000000000000000af691004500101534282 8b011001050122`**
   `{"timestamp": 0, "source": 0, "destination": 27055, "payload": {"sceneRecall": {"minimal": {"sceneNumber": 1, "tid": 34}}, "opcode": 33346}}`
2. **`00000000000000000000af6910045001015343828c011001050122`**
   `{"timestamp": 0, "source": 0, "destination": 27055, "payload": {"sceneRecallUnacknowledged": {"minimal": {"sceneNumber": 1, "tid": 34}}, "opcode": 33347}}`
3. **`00000000000000000000af6910055001011342828b1002c50122f041083f`**
   `{"timestamp": 0, "source": 0, "destination": 27055, "payload": {"sceneRecall": {"optional": {"sceneNumber": 1, "tid": 34, "transitionTime": 30.0, "delay": 0.5}}, "opcode": 33346}}`
4. **`00000000000000000000af6910055001011343828c1002c50122f041083f`**
   `{"timestamp": 0, "source": 0, "destination": 27055, "payload": {"sceneRecallUnacknowledged": {"optional": {"sceneNumber": 1, "tid": 34, "transitionTime": 30.0, "delay": 0.5}}, "opcode": 33347}}`

SCENE_STATUS

Fields	Description
`statusCode`	integer, 0=Success, 1=Scene Register Full, 2=Scene Not Found, 3>=RFU
`currentScene`	integer, range 0-15, current scene number or 0 if node is not in a scene
`targetScene`	integer, range 1-15, target scene during transition
`remainingTime`	float, range 0-37200, time remaining to reach target scene [seconds]

Samples:

1. **`a65689eb79010000af69a7021004500101515e8d0110010000`**
   `{"timestamp": 1623154316966, "source": 27055, "destination": 679, "payload": {"sceneStatus": {"minimal": {"statusCode": 0, "currentScene": 0}}, "opcode": 94}}`
2. **`84318ceb79010000af69aa021005500101115e8d100210010cf041`**
   `{"timestamp": 1623154504068, "source": 27055, "destination": 682, "payload": {"sceneStatus": {"optional": {"statusCode": 0, "currentScene": 0, "targetScene": 1, "remainingTime": 30.0}}, "opcode": 94}}`

SCENE_REGISTER_GET

Fields	Description
-	-

Sample:

1. **0000000000000000000af6910035001011344828e0ffcffffff**
   `{"timestamp": 0, "source": 0, "destination": 27055, "payload": {"sceneRegisterGet": {}, "opcode": 33348}}`

SCENE_REGISTER_STATUS

Fields	Description
`statusCode`	integer, 0=Success, 1=Scene Register Full, 2=Scene Not Found, 3>=RFU
`currentScene`	integer, range 0-15, current scene number or 0 if node is not in a scene
`scenes`	list[integer], list of numbers of configured scenes

Sample:

1. **899980ef79010000af69a702100650010113458 28f500101000011012355 03040102**
   `{"timestamp": 1623220853129, "source": 27055, "destination": 679, "payload": {"sceneRegisterStatus": {"statusCode": 0, "currentScene": 0, "scenes": [3, 4, 1, 2]}, "opcode": 33349}}`

B.3 Generic on/off

GENERIC_ONOFF_GET

Fields	Description
-	-

Sample:

1. **0000000000000000000af691003500101130182490ffcffffff**
   `{"timestamp": 0, "source": 0, "destination": 27055, "payload": {"genericOnoffGet": {}, "opcode": 33281}}`

GENERIC_ONOFF_SET / GENERIC_ONOFF_SET_UNACKNOWLEDGED

Fields	Description
`onoff`	integer, 0=on state, 1=off state
`tid`	integer, range 0-255, transaction identifier, **must be unique within 6 second range**
`transitionTime`	float, range 0-37200, time to reach the target state [seconds]
`delay`	float, range 0.0-1.276, command delay [seconds]

Samples:

1. **`00000000000000000000af6910045001015302824a011001030122`**
   `{"timestamp": 0, "source": 0, "destination": 27055, "payload": {"genericOnoffSet": {"minimal": {"onoff": 1, "tid": 34}}, "opcode": 33282}}`
2. **`00000000000000000000af6910045001015303824b011001030122`**
   `{"timestamp": 0, "source": 0, "destination": 27055, "payload": {"genericOnoffSetUnacknowledged": {"minimal": {"onoff": 1, "tid": 34}}, "opcode": 33283}}`
3. **`00000000000000000000af69100550010113028 24a1002c30122f041083f`**
   `{"timestamp": 0, "source": 0, "destination": 27055, "payload": {"genericOnoffSet": {"optional": {"onoff": 1, "tid": 34, "transitionTime": 30.0, "delay": 0.5}}, "opcode": 33282}}`
4. **`00000000000000000000af6910055001011303824b1002c30122f041083f`**
   `{"timestamp": 0, "source": 0, "destination": 27055, "payload": {"genericOnoffSetUnacknowledged": {"optional": {"onoff": 1, "tid": 34, "transitionTime": 30.0, "delay": 0.5}}, "opcode": 33283}}`

GENERIC_ONOFF_STATUS

Fields	Description
`presentOnoff`	integer, 0=on state, 1=off state
`targetOnoff`	integer, 0=on state during a transition, 1=off state during a transition
`remainingTime`	float, range 0-37200, time remaining to reach target on/off state [seconds]

Samples:

1. **`8ccb8deb79010000af69a7021004500101530 4824c0110010000`**
   `{"timestamp": 1623154609036, "source": 27055, "destination": 679, "payload": {"genericOnoffStatus": {"minimal": {"presentOnoff": 0}}, "opcode": 33284}}`
2. **`aad391eb79010000af69a702100450010113 04824c1001c30101f041`**
   `{"timestamp": 1623154873258, "source": 27055, "destination": 679, "payload": {"genericOnoffStatus": {"optional": {"presentOnoff": 1, "targetOnoff": 1, "remainingTime": 30.0}}, "opcode": 33284}}`

B.4 Generic level

GENERIC_LEVEL_GET

Fields	Description
-	-

Sample:

1. **`00000000000000000000af6910035001011305824f0ffcffffff`**
   `{"timestamp": 0, "source": 0, "destination": 27055, "payload": {"genericLevelGet": {}, "opcode": 33285}}`

GENERIC_LEVEL_SET / GENERIC_LEVEL_SET_UNACKNOWLEDGED

Fields	Description
`level`	integer, range -32767-32767, target level to set
`tid`	integer, range 0-255, transaction identifier, **must be unique within 6 second range**
`transitionTime`	float, range 0-37200, time to reach the target level [seconds]
`delay`	float, range 0.0-1.276, command delay [seconds]

Samples:

1. **`00000000000000000000af691004500101530682500110 0106c022`**
   `{"timestamp": 0, "source": 0, "destination": 27055, "payload": {"genericLevelSet": {"minimal": {"level": -16384, "tid": 34}}, "opcode": 33286}}`
2. **`00000000000000000000af69100450010153078251011000106c022`**
   `{"timestamp": 0, "source": 0, "destination": 27055, "payload": {"genericLevelSetUnacknowledged": {"minimal": {"level": -16384, "tid": 34}}, "opcode": 33287}}`
3. **`00000000000000000000af691005500101130682501002c6c022f041083f`**
   `{"timestamp": 0, "source": 0, "destination": 27055, "payload": {"genericLevelSet": {"optional": {"level": -16384, "tid": 34, "transitionTime": 30.0, "delay": 0.5}}, "opcode": 33286}}`
4. **`00000000000000000000af691005500101130782511002c6c022f041083f`**
   `{"timestamp": 0, "source": 0, "destination": 27055, "payload": {"genericLevelSetUnacknowledged": {"optional": {"level": -16384, "tid": 34, "transitionTime": 30.0, "delay": 0.5}}, "opcode": 33287}}`

GENERIC_LEVEL_STATUS

Fields	Description
`presentLevel`	integer, range -32767-32767, current level state
`targetLevel`	integer, range -32767-32767, target level state during a transition
`remainingTime`	float, range 0-37200, time remaining to reach target level state [seconds]

Samples:

1. **`aa7e9aeb79010000af69aa02100450010153088252011001 02c0`**
   `{"timestamp": 1623155441322, "source": 27055, "destination": 682, "payload": {"genericLevelStatus": {"minimal": {"presentLevel": -16384}}, "opcode": 33288}}`
2. **`fcef98eb79010000af69a70210045001011308825210 01cb6641c0f041`**
   `{"timestamp": 1623155339260, "source": 27055, "destination": 679, "payload": {"genericLevelStatus": {"optional": {"presentLevel": 16742, "targetLevel": -16384, "remainingTime": 30.0}}, "opcode": 33288}}`

B.5 Sensor

SENSOR_STATUS

Fields	Description
`format`	integer, range 0-1, format selector
`length`	integer, range 0-127, length of the sensor reading
`sensorSettingPropertyId`	integer, type of the measured value
`sensorPropertyName`	object, measurement returned by the sensor

Samples:

1. **`e0af93eb79010000a96792c0100750010111529711011751040101540 14d4c10010101`**
   `{"timestamp": 1623154995168, "source": 26537, "destination": 49298, "payload": {"sensorStatus": [{"presenceDetected": {"presenceDetected": true}, "format": 0, "length": 1, "sensorSettingPropertyId": 77}], "opcode": 82}}`
2. **`ffaf93eb790100008a6753c010075001011152971101175104010154034e4d1001ffc3f5285c8fea694000`**
   `{"timestamp": 1623154995199, "source": 26506, "destination": 49235, "payload": {"sensorStatus": [{"presentAmbientLightLevel": {"illuminance": 207.33}, "format": 0, "length": 3, "sensorSettingPropertyId": 78}], "opcode": 82}}`

See the Bluetooth Mesh Model Specification for a full description of `SENSOR_STATUS`.

Silvair firmware supports the following sensor properties:

Sensor type	Property name	Unit	Resolution	Values	Property ID (hex)	Supported by
Occupancy	`presenceDetected`	–	–	0 or 1	77 (0x004D)	Dimming (all), UART
Ambient light	`presentAmbientLightLevel`	lux	0.01	0–167772.14	78 (0x004E)	Dimming (all), UART
Energy	`presentDeviceInputPower`	W	0.1	0–1677721.4	82 (0x0052)	Dimming (DALI), UART
Energy	`preciseTotalDeviceEnergyUse`	kWh	0.001	0–4294967.293	114 (0x0072)	Dimming (DALI), UART
Energy	`totalDeviceEnergyUse`	kWh	1	0–16777214	106 (0x006A)	UART
Energy	`presentDeviceInputCurrent`	A	0.01	0–655.34	87 (0x0057)	UART
Energy	`presentInputVoltage`	V	1/64	0.0–1022.0	89 (0x0059)	UART

B.6 Health status

The opcodes for obtaining and clearing health faults are:

- `HEALTH_CURRENT_STATUS (0x04)`
- `HEALTH_FAULT_CLEAR (0x80 0x2F)`
- `HEALTH_FAULT_CLEAR_UNACKNOWLEDGED (0x80 0x30)`
- `HEALTH_FAULT_GET (0x80 0x31)`
- `HEALTH_FAULT_STATUS (0x05)`

HEALTH_CURRENT_STATUS

Fields	Description
`testId`	integer, range 0-255, identifier of the most recently performed test
`companyId`	integer, range 0-65535, Bluetooth assigned Company Identifier
`faultArray`	List[integers], an array containing a sequence of up to the last eight 8-bit current fault identifiers from the faults list

Samples:

1. **`23645ebd87010000a967c703100550010111047d5001010c3601110102`**
   ```
   {'timestamp': 1682509292579, 'source': 26537, 'destination': 967, 'payload':
   {'healthCurrentStatus': {'testId': 0, 'companyId': 310, 'faultArray': []},
   'opcode': 4}}
   ```
2. **`23645ebd87010000a967c7031006500101111047d5001010c360111010a0104`**
   ```
   {'timestamp': 1682509292579, 'source': 26537, 'destination': 967, 'payload':
   {'healthCurrentStatus': {'testId': 0, 'companyId': 310, 'faultArray': [4]},
   'opcode': 4}}
   ```

HEALTH_FAULT_CLEAR / HEALTH_FAULT_CLEAR_UNACKNOWLEDGED

Fields	Description
`companyId`	integer, range 0-65535, Bluetooth assigned Company Identifier

Samples:

1. **`000000000000000000a96710045001011132f807e1001033601`**
   `{'timestamp': 0, 'source': 0, 'destination': 26537, 'payload': {'healthFaultClear': {'companyId': 310}, 'opcode': 32815}}`
2. **`000000000000000000a967100450010113308 07f1001033601`**
   `{'timestamp': 0, 'source': 0, 'destination': 26537, 'payload': {'healthFaultClearUnacknowledged': {'companyId': 310}, 'opcode': 32816}}`

HEALTH_FAULT_GET

Fields	Description
`companyId`	integer, range 0-65535, Bluetooth assigned Company Identifier

Samples:

1. **`000000000000000000a967100450010113318080 1001033601`**
   `{'timestamp': 0, 'source': 0, 'destination': 26537, 'payload': {'healthFaultGet': {'companyId': 310}, 'opcode': 32817}}`

HEALTH_FAULT_STATUS

Fields	Description
`testId`	integer, range 0-255, identifier of a most recently performed test
`companyId`	integer, range 0-65535, Bluetooth assigned Company Identifier
`faultArray`	List[integers], an array containing a sequence of up to last sixteen 8-bit registered fault identifiers from the faults list

Samples:

1. **`23645ebd87010000a967c70310055001011105815001010c3601110102`**
   `{'timestamp': 1682509292579, 'source': 26537, 'destination': 967, 'payload': {'healthFaultStatus': {'testId': 0, 'companyId': 310, 'faultArray': []}, 'opcode': 5}}`
2. **`23645ebd87010000a967c70310065001011105815001010c360111010a0104`**
   `{'timestamp': 1682509292579, 'source': 26537, 'destination': 967, 'payload': {'healthFaultStatus': {'testId': 0, 'companyId': 310, 'faultArray': [4]}, 'opcode': 5}}`

B.6.1 Faults

Fault	Description	Supported by
8	Power supply was interrupted. Non-volatile memory items were not saved successfully. Reported with Company ID of 310.	Dimming (all), UART
19	Configuration warning. Firmware version 2.19.0 and later: Not all connected drivers are of the same type. ● DT51 (includes newer Dexal and Sensor Ready drivers) ● Sensor Ready (1.x) ● Old DEXAL ● Koizumi XE91990 ● Other (if none of above) In firmware version earlier than 2.22.0, reported with Company ID of 310. In firmware version 2.22.0 and later, reported with Company ID of 2788. Firmware version 2.21.0 and later: Not all connected drivers are of the same type, or some connected drivers support CCT while others do not. In firmware version earlier than 2.22.0, reported with Company ID of 310. In firmware version 2.22.0 and later, reported with Company ID of 2788. Firmware version 2.22.0 and later: Reported with fault 137. Reported with Company ID of 2788.	Dimming (DALI)
20	The firmware license is for CCT, but the output type is configured as closed loop. Reported with Company ID of 310.	Dimming (CCT)
47	UART communication warning (Modem ↔ MCU). Reported with Company ID of 310.	UART
	I2C communication warning (nrf52 ↔ ALS Sensor). Reported with Company ID of 310.	Dimming (ALS: BH1726 or OPT3004)
	Firmware version 2.22.0 and later: DALI bus warning. Reported with fault 133 or 134. Reported with Company ID of 2788.	Dimming (DALI)
48	UART communication error (Modem ↔ MCU). Reported with Company ID of 310.	UART
	I2C communication error (nrf52 ↔ ALS Sensor). Reported with Company ID of 310.	Dimming (ALS: BH1726 or OPT3004)
128	Firmware version 2.22.0 and later: DALI driver failure. Reported with Company ID of 2788.	Dimming (DALI)
129	Firmware version earlier than 2.22.0: DALI driver failure (received in response to Query Status). Reported with Company ID of 310. Firmware version 2.22.0 and later: Lamp failure detected by DALI driver. Reported with Company ID of 2788.	Dimming (DALI)
130	Firmware version earlier than 2.22.0: No light source is connected to the DALI driver (received in response to Query Status). Reported with Company ID of 310.	Dimming (DALI)

	Firmware version 2.22.0 and later: The DALI driver did not respond to 2 consecutive queries. Reported with Company ID of 2788.	
131	Firmware version 2.22.0 and later: The DALI driver did not respond to 5 consecutive queries. Reported with Company ID of 2788.	Dimming (DALI)
132	Firmware version earlier than 2.22.0: The DALI driver did not respond to 2 consecutive queries. Reported with Company ID of 310. Firmware version 2.22.0 and later: DALI bus low for at least the system failure time. Reported with Company ID of 2788.	Dimming (DALI)
133	Firmware version earlier than 2.22.0: The DALI driver did not respond to 5 consecutive queries. Reported with Company ID of 310. Firmware version 2.22.0 and later: No driver has been found on the bus. Reported with fault 47. Reported with Company ID of 2788.	Dimming (DALI)
134	Firmware version earlier than 2.22.0: DALI bus short. The DALI circuit is not working correctly. The bus short is detected by a lack of loopback. Reported with Company ID of 310. Firmware version 2.22.0 and later: The number of drivers on the bus exceeds the gateway capacity. Reported with fault 47. Reported with Company ID of 2788.	Dimming (DALI)
135	Firmware version 2.19.0 and earlier than 2.22.0: No drivers connected during the DALI Addressing procedure. Reported with Company ID of 310. Firmware version 2.22.0 and later: The configuration of the DALI bus has changed. For example, a new bus unit has been added or removed. Reported with Company ID of 2788.	Dimming (DALI)
136	Firmware version 2.19.0 and earlier than 2.22.0: Too many drivers connected during the DALI Addressing procedure. Reported with Company ID of 310. Firmware version 2.22.0 and later: Unexpected event on the DALI bus occurred. Reported with Company ID of 2788.	Dimming (DALI)
137	There is more than one driver connected and they have different capabilities. Reported with fault 19. Reported with Company ID of 2788.	Dimming (DALI)
144	Closed loop calibration error. Reported with Company ID of 310.	Dimming (closed loop)
160	The external MCU connected to the UARTModem did not respond to 5 consecutive queries for the RTC time, or it responded with a “Time not known” value. Reported with Company ID of 310.	UART
161	Error of the RTC connected to the external MCU. The external MCU sets this error using the SetFaultRequest UART command. Reported with Company ID of 310.	UART

Appendix C: Project data details

The following parameters are available via the project data API:

Project:

- Project ID (this is required to access all the APIs)
- Project name
- The role assigned to the user
- Features enabled for the project

Area:

- Application and network keys for the area
- The floor plan image uploaded to the project
- The x,y coordinates of the zones as seen in the web/mobile application

Zone:

- Zone ID
- Group address
- Energy profile ID, including the name assigned to it
  Note: the energy profile parameters are returned by the /public/projects/{projectId}/energyProfiles endpoint

Node:

- The node advertising name (how it appears in the app)
- The factory name – the name assigned when it was manufactured
- A custom name (if assigned)
- UUID
- The network features that are enabled for the node (for example whether it is a relay, a proxy, whether it is linked to an EnOcean switch, etc.)
- The capabilities of the node, for example whether it is a motion sensor, ambient light sensor, emergency support, etc.)

Scene/scenario parameters (require the ZoneID):

- Scene/scenario name
- Scene number and whether it is enabled
- Profile name, type, and number
- Scenario type (for example occupancy, daylight harvesting, etc.)
- Light level on power up
- Run, prolong, and standby levels and fade times (for automated scenarios)
- Lightness value set for the scene (for static scene)
- High/low end trim values