DALI PRO 2 IoT

REST API

Light Control Quick Start Guide

Version 1.14.0

Table of Contents

1. Introduction

The present document is meant to help the adoption and usage of the REST API interface for light control. It provides some practical examples on how the APIs shall be utilized in common use cases and a description of the available REST API.

2. Prerequisites

The following hardware and software requirements need to be satisfied in order to reproduce the examples provided in the next pages.

2.1 Hardware

1. At least 1 DALI PRO 2 IoT.
2. At least 1 DALI ballast and 1 DALI input device (e.g. motion sensor, push button, etc.).
3. A computer connected to the DALI PRO 2 IoT via LAN cable or Wi-Fi dongle.

2.2 Software

1. *DALI Professional 3* software version 3.1.13 or higher (ask your sales representative to get the software installation package). This is the software needed to commission your lighting system.
2. *Postman* (https://www.postman.com/downloads/). This is the software needed to issue REST calls to the DALI PRO 2.

   N.B.: *Postman* is not the only software for issuing REST calls. There are many others available tools, feel free to choose the one you are more comfortable with.
3. Generic web browser e.g. *Chrome, Edge, Firefox* etc.

3. System setup

A DALI PRO 2 IoT can be setup by means of one of the following tools:

- OSRAM software *DALI Professional 3* version 3.1.13 or higher. This software shall be installed on a PC hence it is hereafter also referred as *PCTool*.
- DALI PRO 2 *Commissioning UI*, a product resident user interface that can be accessed by typing the IP-address of the device into a web browser e.g. https://192.168.2.109.

3.1 DALI PRO 2 IP address

The *PCTool* and the *Commissioning UI* make use of the IP address of DALI PRO 2 IoT to establish a connection with it.

The *PCTool* offers some network discovery capabilities i.e. it automatically provides an IP address list of all the DALI PRO 2 IoTs detected on the network. The user shall then identify which is the device he wants to connect to and eventually connect to it by typing the credentials. Once connected, the IP address of the DALI PRO 2 IoT is available on the *PCTool* too (see Figure 1). Due to the network configuration itself and other factors out of the *PCTool* control, the IP address list could be incomplete. In this case the user shall retrieve the IP address of the DALI PRO 2 IoT by means of a software network scanner (e.g. Advanced IP Scanner), or by looking up the network router page or by consulting the local area network administrator.

When the user plans to use the *Commissioning UI* instead, the DALI PRO 2 IoT IP address shall be known in advance. The user must then type the IP address on a web browser as described in the previous paragraph. At this point the *Commissioning UI* opens on the browser.

A third option, available for both tools, is to exploit the WiFi network created by the DALI PRO 2 IoT when the MODE button on the front case of the device is (short) pressed. Once the WIFI led is stable on, the user can connect to the WiFi network named with the serial number of the device at address 192.168.8.1 by providing the required info.

The DALI PRO 2 IP address used in the examples of this document is *192.168.2.109*.

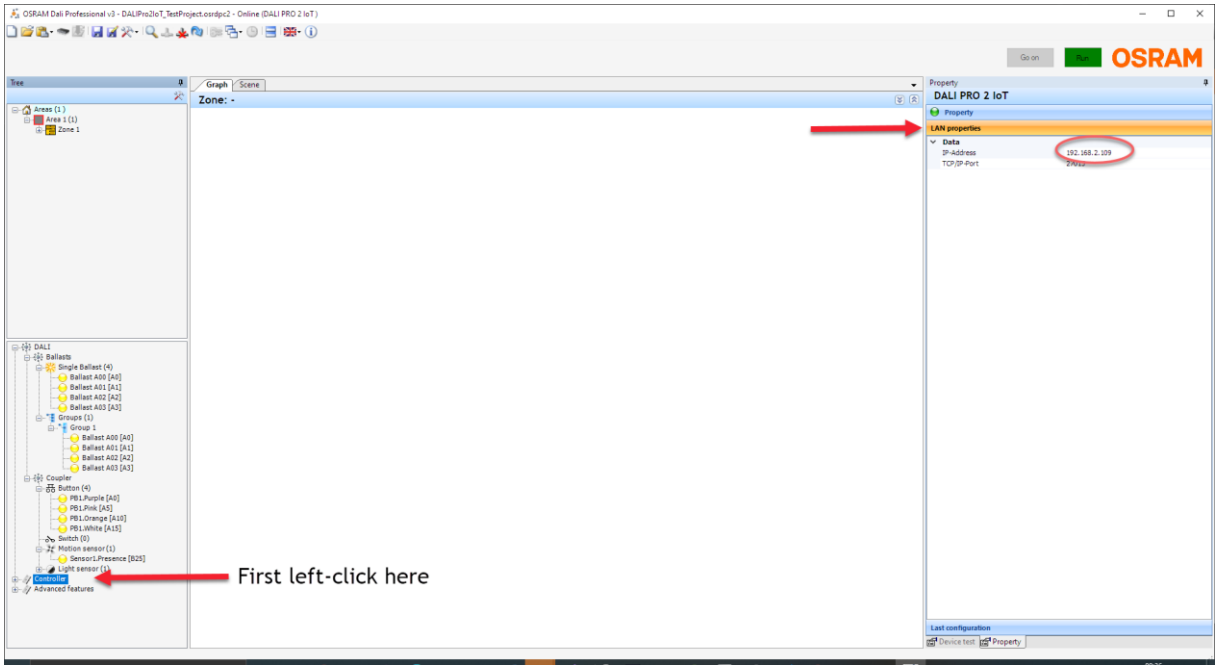

Figure 1

3.2 Commission the system

Commissioning the system means creating a basic configuration i.e. a group of ballasts with one or more input devices connected (e.g. presence sensors, buttons, etc.) as depicted in the Figure 2. In this example a group of four ballasts (Ballast A00 - A03) is controlled by a Push-Button Coupler (PB1) with four buttons. The configuration must then be uploaded on the DALI PRO 2 IoT by clicking on the *Upload* icon.

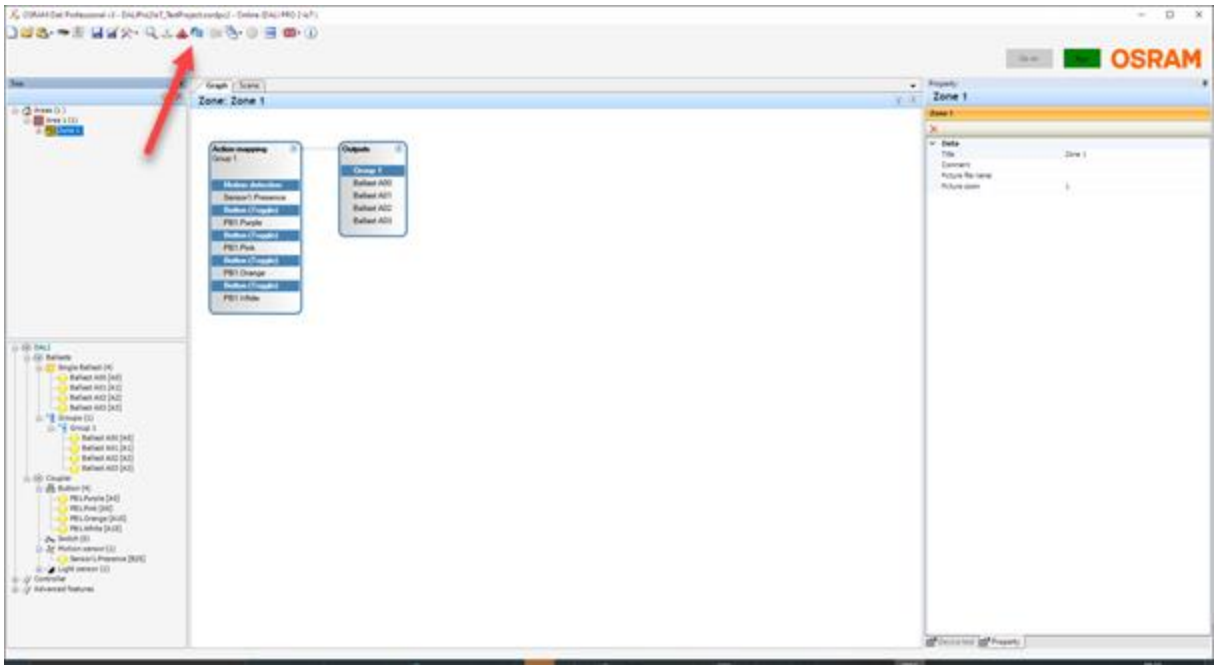

Figure 2

In Figure 3, a similar configuration is created with the *Commissioning UI*; here four buttons and a presence sensor control a group of four ballasts. In order to upload the configuration on the DALI PRO 2 IoT the *Sync* button must be pressed.

IMPORTANT NOTICE

Light control can only be performed at group level, not on a single ballast

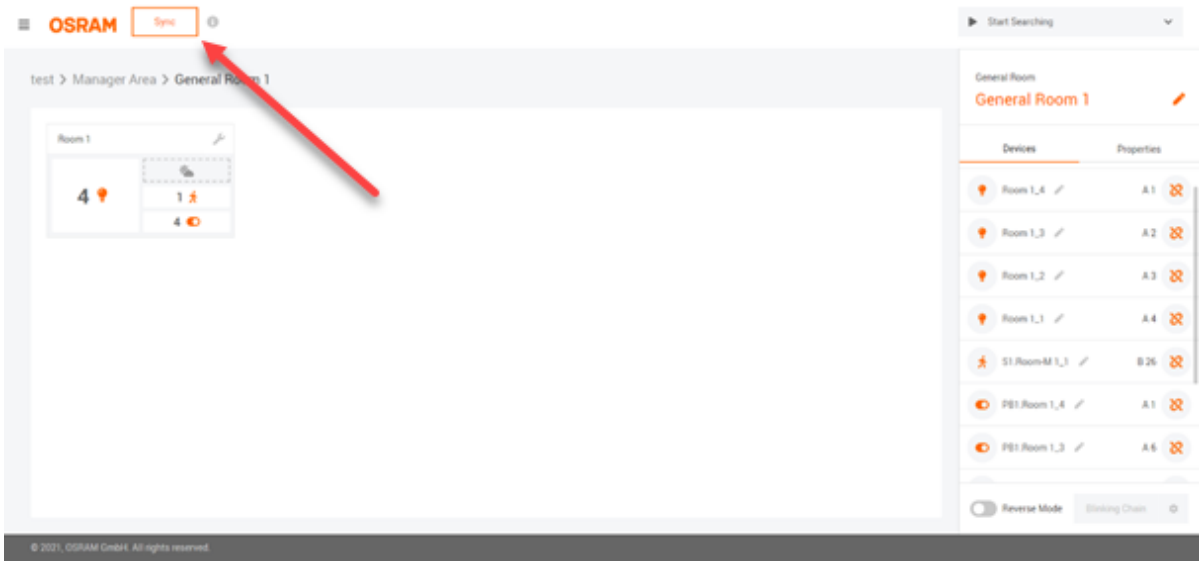

Figure 3

3.3 Demo REST interface in the PC-Tool

The demo REST interface built into the PC tool can be used to simply try out the REST calls. The authentication token is automatically added in the background.

After connection to the device with put in of the password and username, the demo REST interface can be used over the tool menu "REST Interface":

After connecting to the device by entering the password and username, the demo REST interface can be opened via the "REST interface" tool menu:

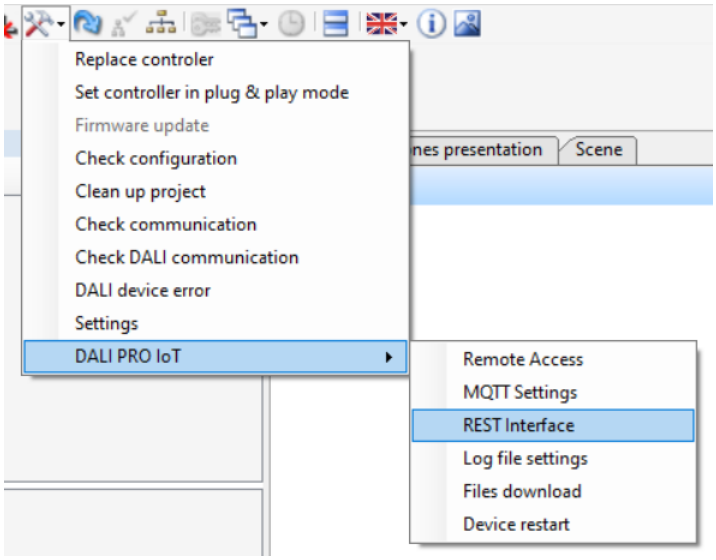

Figure 4

The REST interface provides the methods GET, PUT, POST and DELETE. The methods PUT and POST can have a payload.

The payload and result will be interpreted as json.

Over the button "Send" will be the REST request transmitted to the connected device.

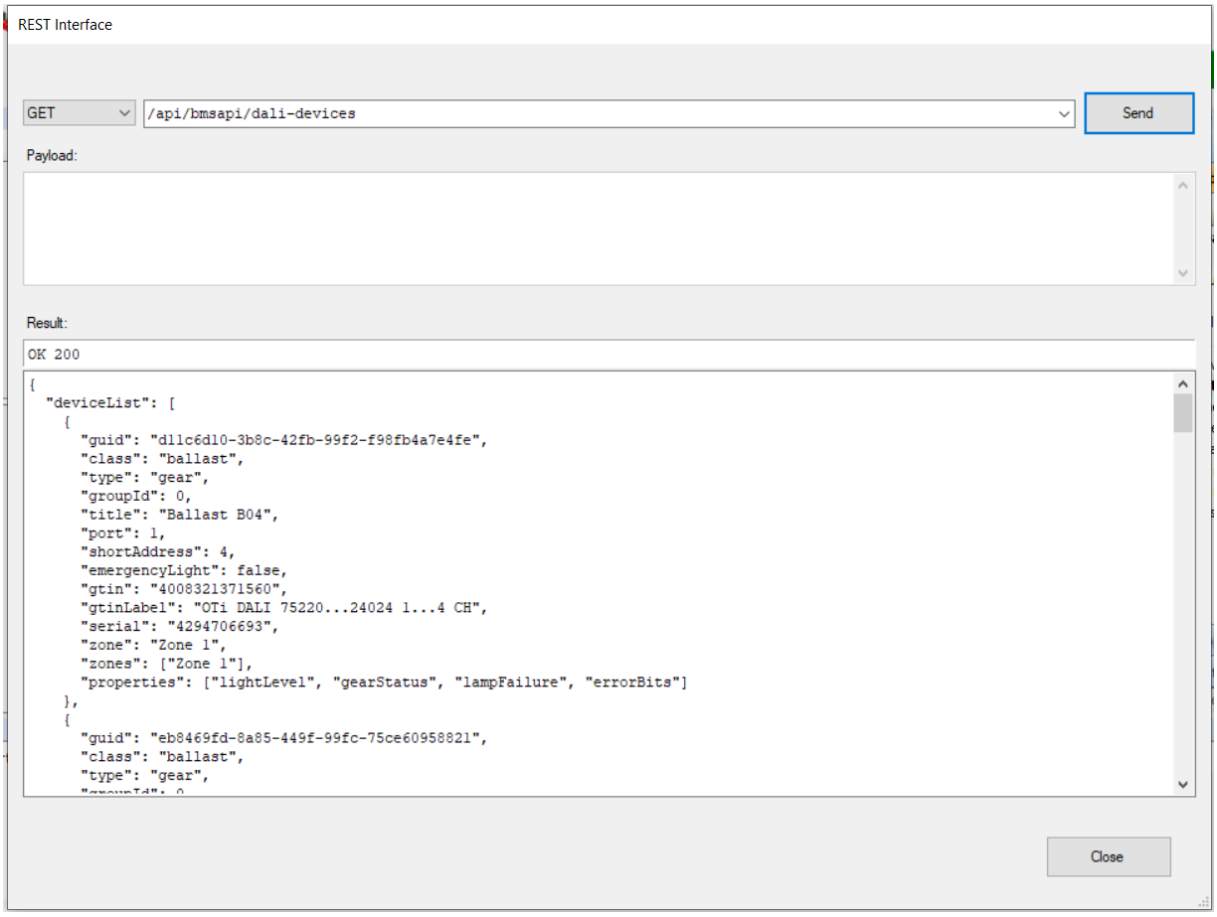

Figure 5

3.4 Authentication token

All REST API calls shall include an authentication token in the request header.

Notice that the following placeholders must be replaced, wherever present, with the DALI PRO 2 IoT administrator username and password used during registration and IP address:

- <ADMIN_USERNAME>
- <ADMIN_PASSWORD>
- <DALI_PRO2_IP>

In order to get the authentication token, the following steps shall be followed:

4. Open *Postman.*
5. In the address bar input the following URL:

```
https://<DALI_PRO2_IP>/auth/login
```

6. In the methods dropdown select *POST*.
7. Click on the Body tab and insert the following json payload:

```
{
  "username": "<ADMIN_USERNAME>",
  "password": "<ADMIN_PASSWORD>"
}
```

8. Select for the Body type *raw* and *JSON*.
9. Press the *Send* button.
10. The answer from the DALI PRO 2 appears on the bottom of the *Postman* window. If credentials are correct the response status is *200 OK* and the response payload contains a field with the authentication token (see Figure 6).

```
"authHeader": "Basic <token>"
```

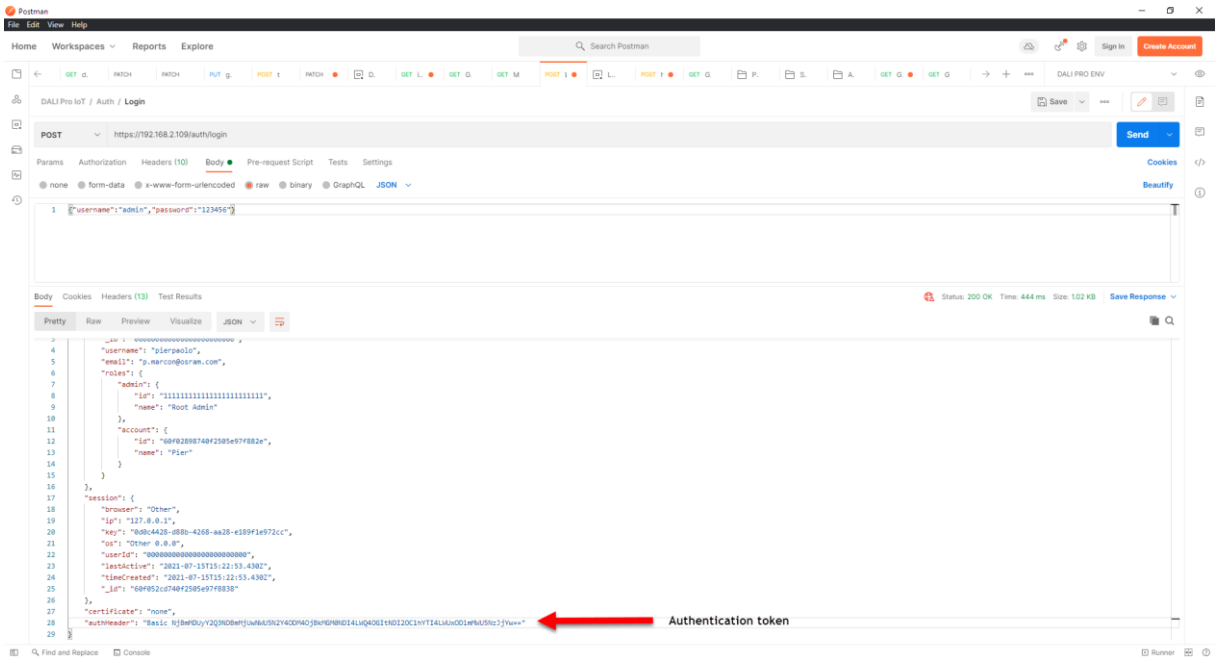

Figure 6

Add the authentication token in the header request

In order to issue a REST call the authentication token shall be included in command.

> All REST calls shall have the authentication token in the header's Authorization field (see *Figure 7*). The expiration time of the authentication token is 30 mins.

```
"Authorization": "Basic <token>"
```

Notice that in the example below (Figure 7) the response status is *401 Unauthorized* because the authentication token in the header is not valid.

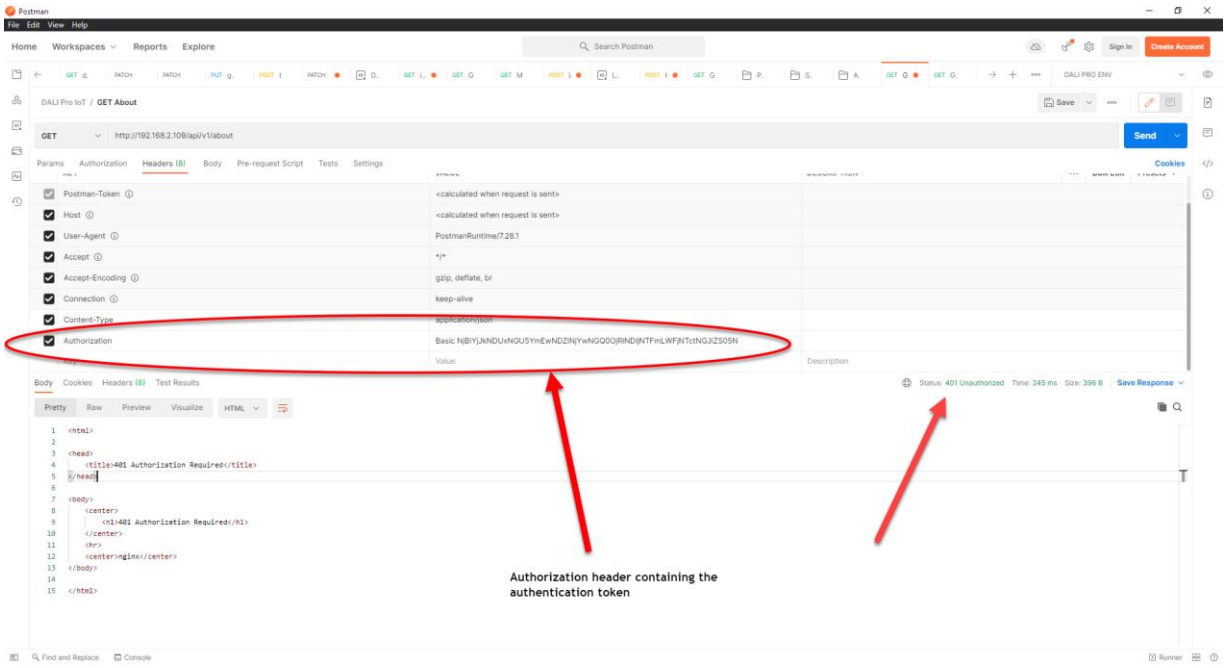
Authorization header containing the authentication token

Figure 7

4. Examples

At the end of the document in Appendix B snippets of Python code are reported. They represent a basic implementation of some of the commands used in the following examples.

4.1 Light control - with Demo REST interface

Access to lighting groups is via the groupId. the group Id is an up-counting number, starting with 0 and is part of the URL.

For example, to change the light of the first group to 10 % use the URL:

```
/api/bmsapi/groups/0/state
```

with the PUT method and the payload:

```
{
  "level": 10
}
```

Figure 8

The current state can be got with the same URL and the GET method.

Figure 9

The URL:

```
/api/bmsapi/groups
```

provides the overview about all existing groups.

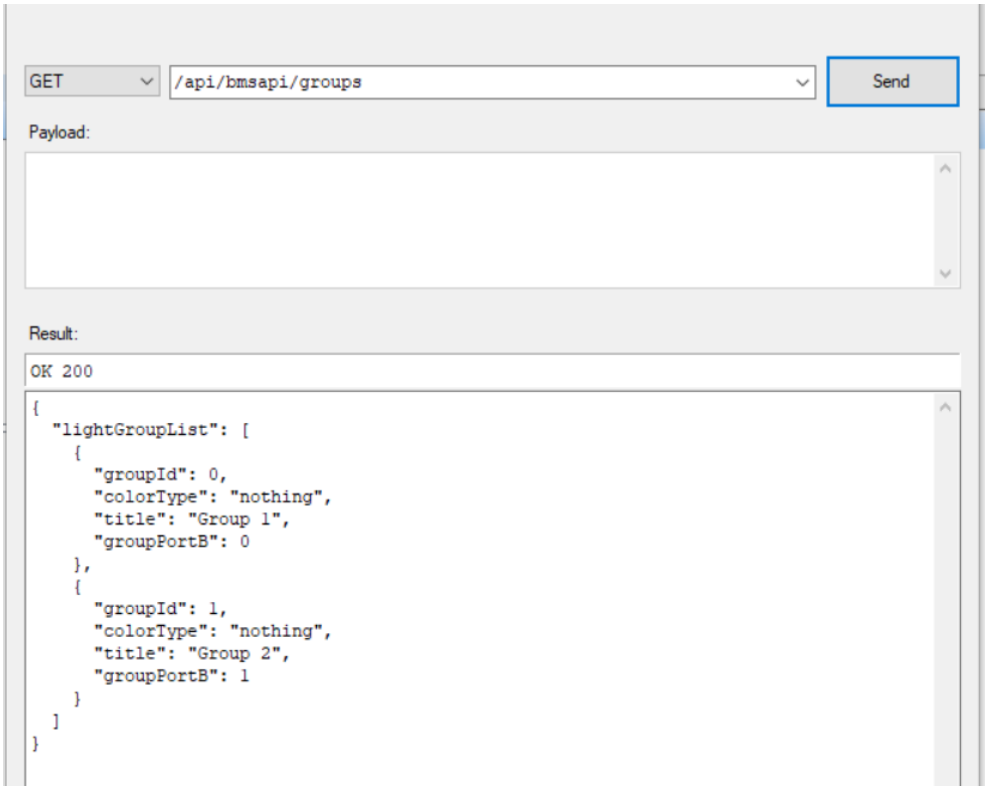

Figure 10

4.2 Light control - with Postman

The example of the next paragraphs shows how to control the lights of a system (e.g. change dimming level, color temperature, etc.).

4.2.1 Fetch light groups information

In order to control a group, its *groupId* (up counting number: 0, 1, 2, ...) must be known.

> Every group is identified with a unique *groupId*.

In order to get the *groupId* of the existing groups a REST call shall be issued following these steps:

1. Open *Postman.*
2. In the address bar input the following URL:

```
https://<DALI_PRO2_IP>/api/bmsapi/groups
```

3. In the methods dropdown select *GET*.
4. Press the *Send* button.
5. The answer from the DALI PRO 2 will appear in the bottom of the *Postman* window (see Figure 11).

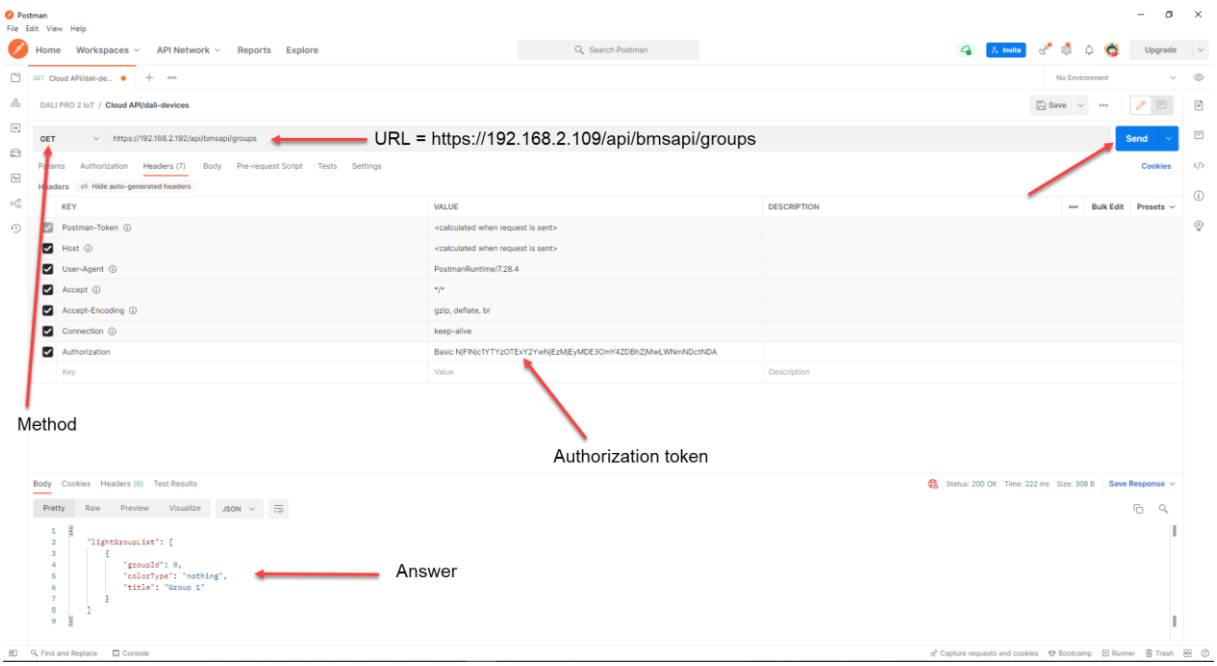

Figure 11

4.2.2 Take note of the *groupId*

The answer received in the previous call is a list named *lightGroupList.* Every element of the list represents a group of lights uniquely identified by the field *groupId* (in this example only one group is present). Each group has a name that can be found in the *title* field.

Note down the *groupId* of the group that shall be controlled.

In this example we want to control the only existing group, i.e.

```
"groupId": 0
```

4.2.3 Change the light level of the group

1. Open *Postman*
2. In the address bar input the following URL:

   ```
   https://<DALI_PRO2_IP>/api/bmsapi/groups/<GROUP_ID>/state
   ```

   where:

   - <GROUP_ID> shall be substituted with the *groupId* noted down in the previous step.
3. In the methods dropdown select *PATCH*.
4. Click on the Body tab and insert the following payload:

   ```
   {
   "level": 50
   }
   ```

   This brings the light dimming level to 50%.
5. Press the *Send* button.
6. The connected lights, belonging to the chosen group, should dim to 50% and response status should be *200 OK* (see Figure 12).

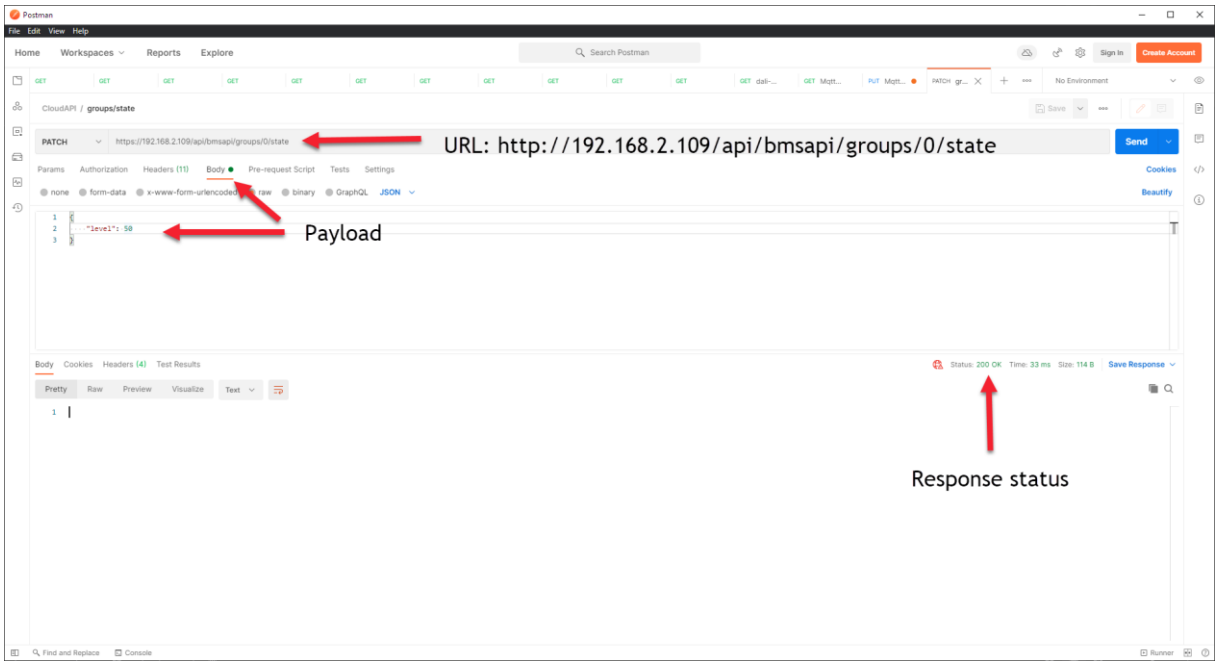

Figure 12

4.3 Configure a scene with *DALI Professional 3* software

A light scene is a set of light levels that a predefined group of ballasts takes whenever the scene is recalled. Scenes are useful to define different light levels that are set at the same time.

The first thing to do is creating a scene associated to a group.

> A scene is always associated to a ballasts group

In order to have a scene that can be recalled through the REST API, the following operations must be performed:

1. Configure a scene (see Figure 13). Take note of the scene name, it will be used to recall the scene.

   In this example the scene created is called *Scene 1*.
2. Program an input device (e.g. a button) to recall that scene (see Figure 14).
3. Upload the new configuration to DALI PRO 2 (see Figure 14).

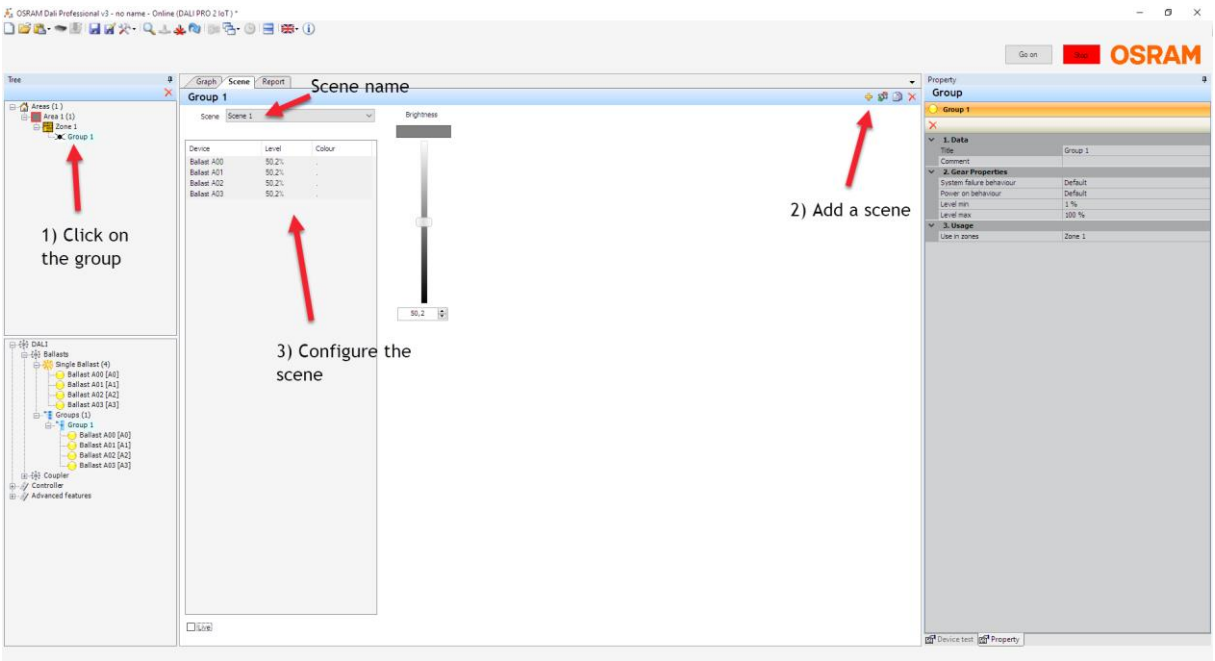

Figure 13

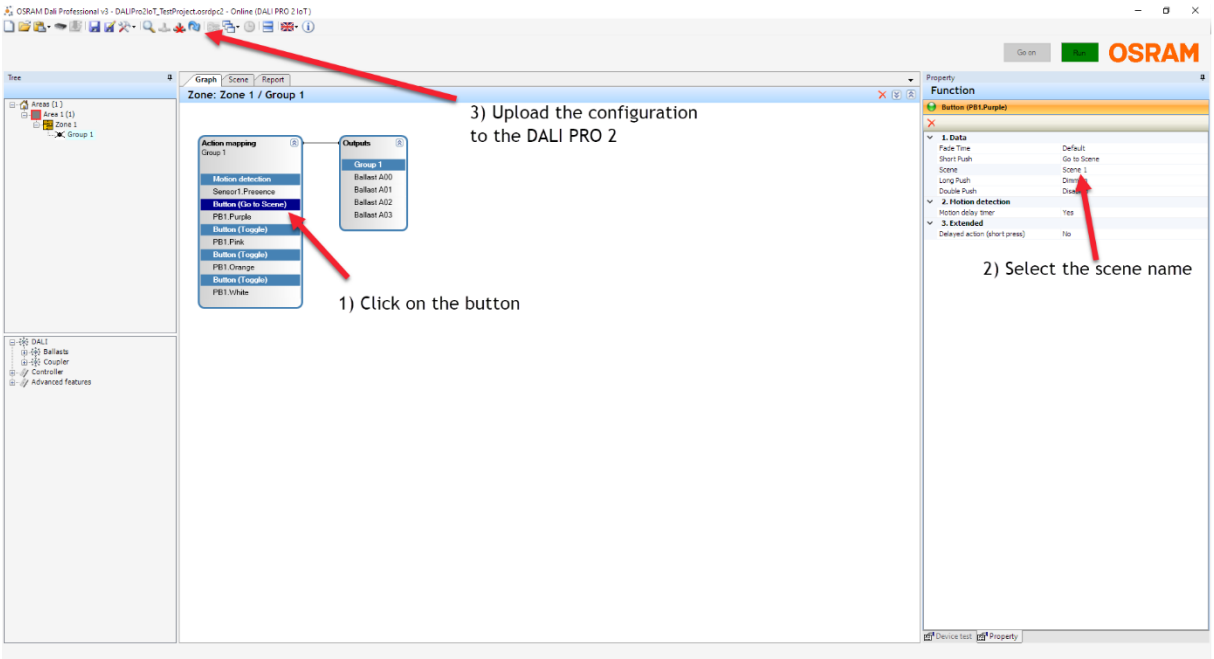

Figure 14

4.4 Scene recall - with Demo REST interface

The first step is to query the names of the existing light groups of the selected group, if the scene names not already known, by the URL for example for the first group with the groupId 0:

```
/api/bmsapi/groups/0
```

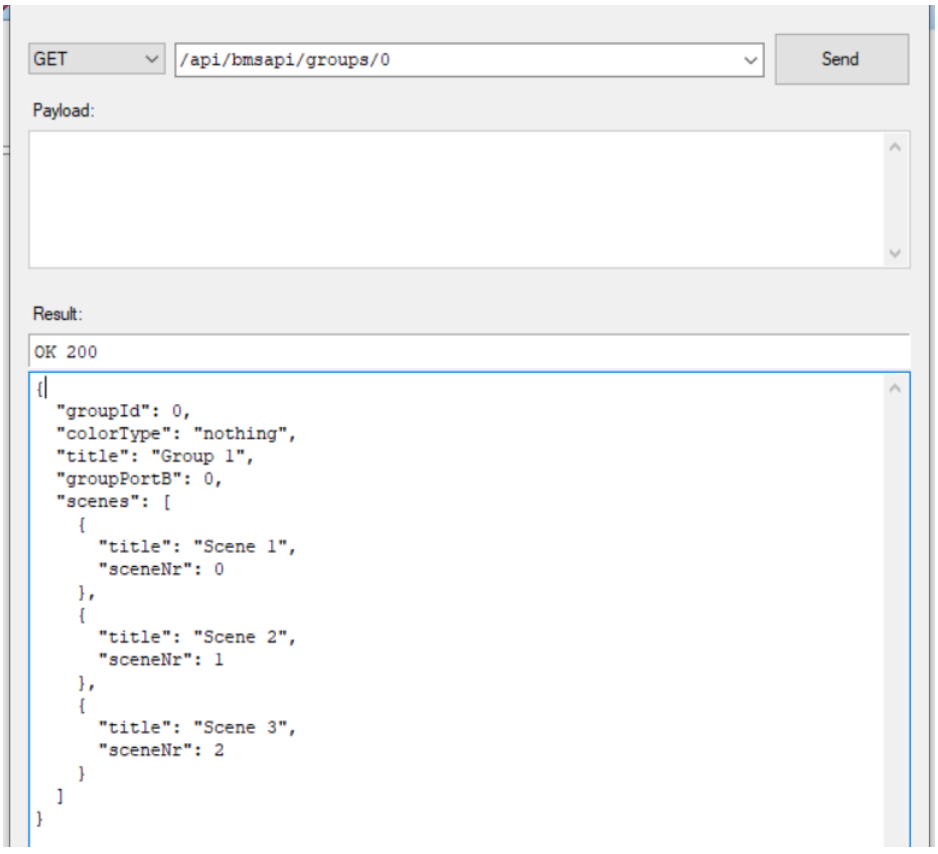

Figure 15

The scene can be recalled with the URL:

```
/api/bmsapi/groups/0/state
```

with the PUT method and the payload:

```
{
  "scene": Scene 2"
}
```

4.5 Scene recall - with Postman

4.5.1 Fetch light groups information

In order to verify that the scene has been correctly stored on the desired group, follow the steps described in paragraph 4.2.1 to fetch light group information. The answer should contain a list of light groups, one of these groups contains the programmed scenes (see Figure 16).

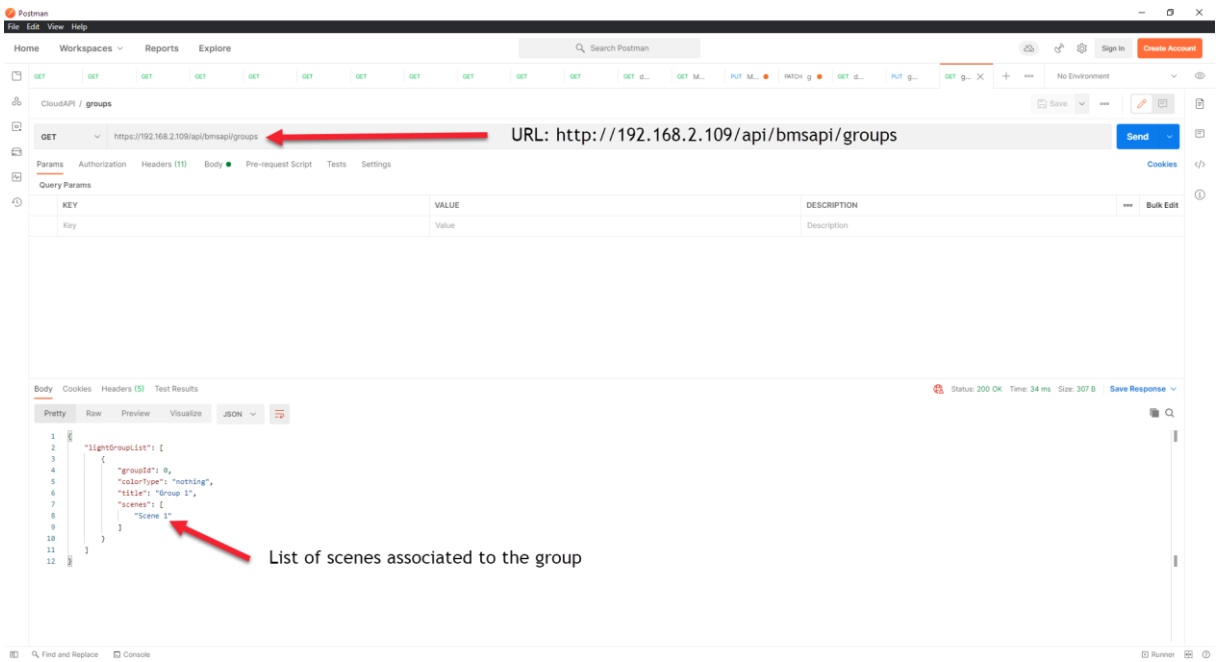

Figure 16

Extract from the response payload the *groupId* where the scene has been programmed. Take also note of the scene name.

In this example *groupId* and *scene* have the following values:

```
{
  "groupId": 0,
  "scene": "Scene 1"
}
```

4.5.2 Send the recall scene command

1. Open *Postman.*
2. In the address bar input the following URL:

```
https://<DALI_PRO2_IP>/api/bmsapi/groups/<GROUP_ID>/state
```

In the methods dropdown select *PATCH* or *PUT* (this is a second option available since device version 3.1.10.x).

3. Click on the Body tab and insert the following payload:

```
{
  "scene": "<SCENE_NAME>"
}
```

where:

- <SCENE_NAME> shall be replaced with the scene name noted down in the previous step.

4. Press the *Send* button.
5. The response status should be *200 OK* and the scene should be recalled by the related group of lights.

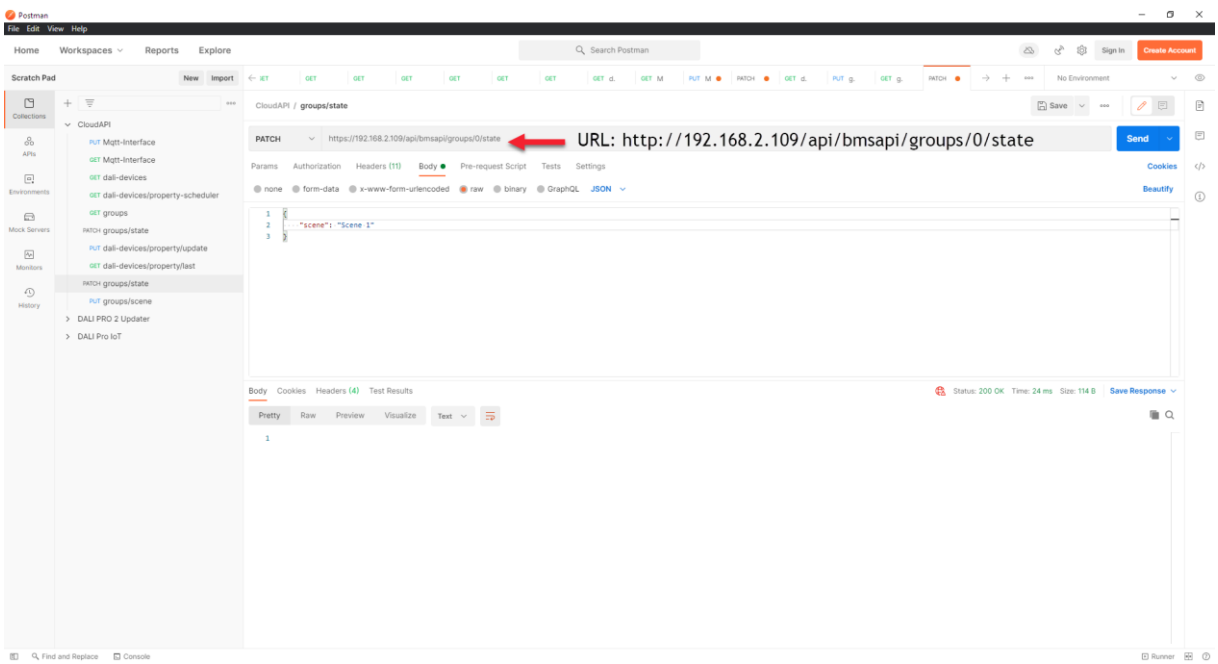

Figure 17

4.6 Ballast properties - with Demo REST interface

Access to the device data is via the device GTIN. The GTIN is generated with adding of a device to the project file and will not change between different uploads, based on the same project file.

The endpoint URL "/api/bmsapi/dali-devices" provides the list off all existing devices.

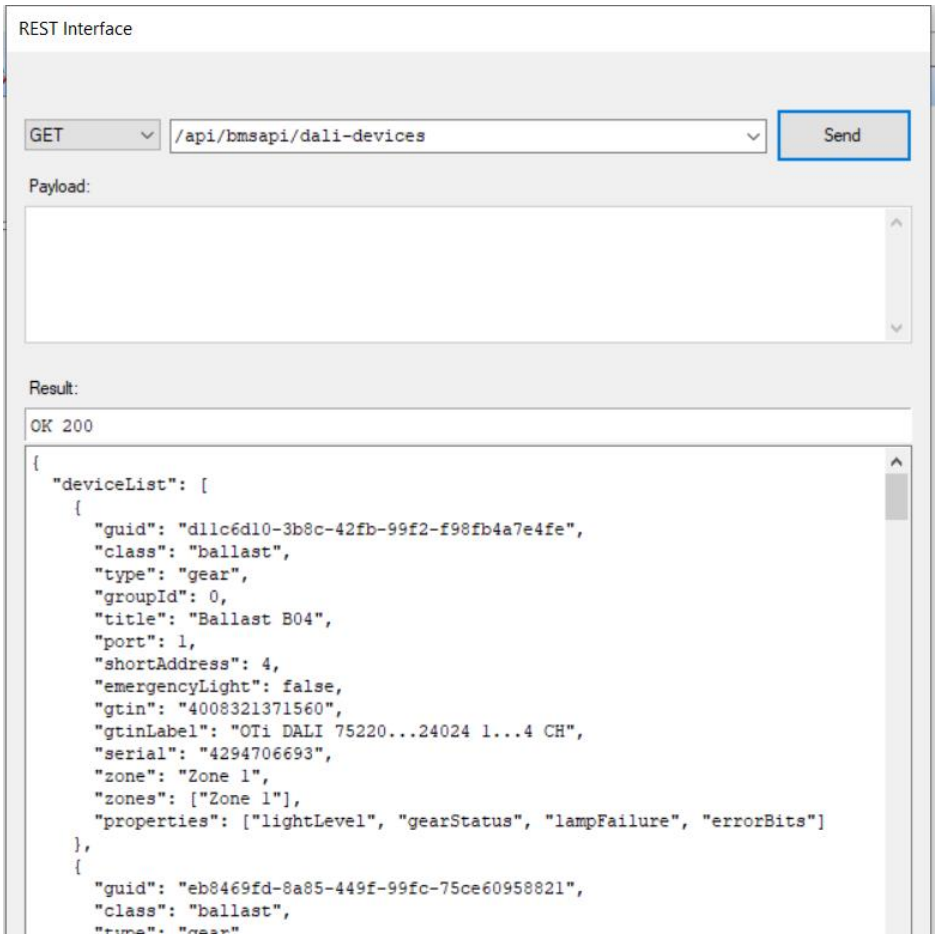

Figure 18

The devices with the property *type* equal to "gear" are the DALI ballasts.

The device data contains for each device also a list of the supported properties of the device, for example for a DALI data device:

Figure 19

With the GUID <BALLAST_GUID> and the property <Property_Name> in the URL

```
/api/bmsapi/dali-devices/<BALLAST_GUID>/property/<PROPERTY_NAME>/active
```

will be trigger the active query of the value from the device by DALI commands. This can be taken some time, depending on the current action on the DALI line. If the device not reachable the result is ERROR 503.

4.7 Ballast properties - with Postman

This example shows how get the information about a ballast connected to a DALI PRO 2 IoT.

> Every DALI device in the system has a unique *guid*

The ballasts information can be retrieved following these steps:

1. Open *Postman.*
2. In the address bar input the following URL:

```
https://<DALI_PRO2_IP>/api/bmsapi/dali-devices
```

3. In the methods dropdown select *GET*.

4. Press the *Send* button.
5. The answer from the DALI PRO 2 appears in the bottom of the *Postman* window.

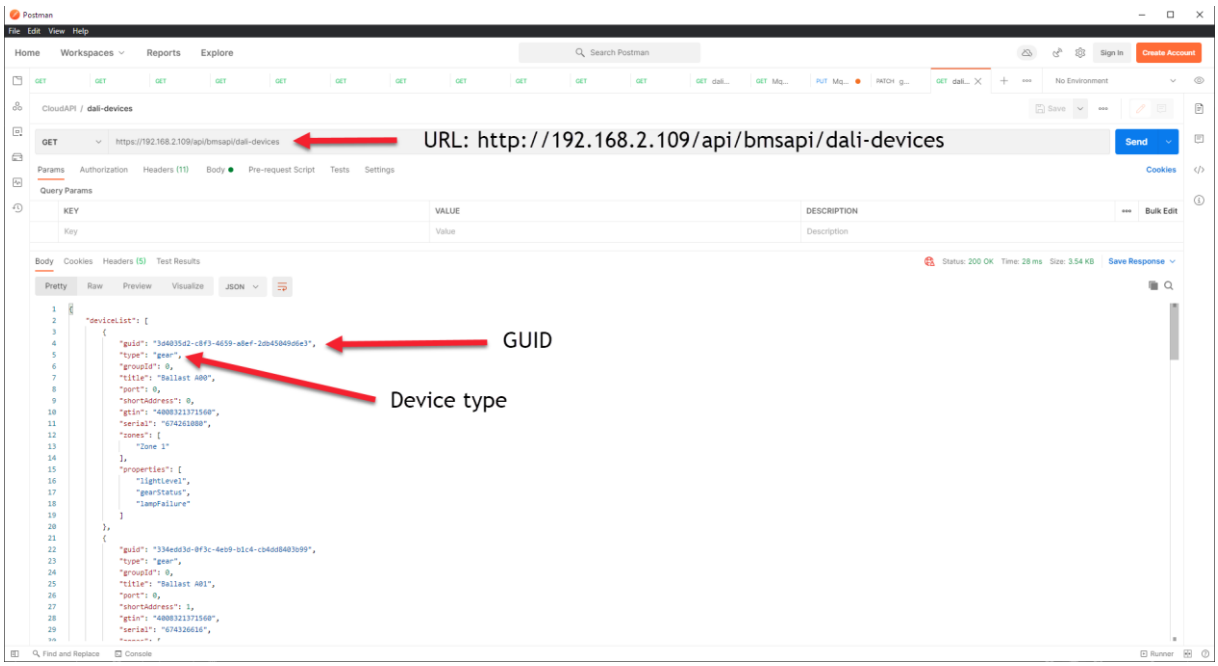

Figure 20

4.7.1 Take note of the ballast *guid*

The answer received from the previous command contains the list of all DALI devices connected to the DALI PRO 2 with the related information.

Every device is identified by:

- *guid*
- *type*
- a set of properties

The full list of available properties (depending on the control gear type) is reported in paragraph 6.1 in Appendix A.

In this example we consider a device of type *gear* (i.e. a ballast) and we query its *lightLevel* property.

E.g. the first device in the list has:

```
"guid": "3d4035d2-c8f3-4659-a8ef-2db45049d6e3"
```

4.7.2 Retrieve the *lightLevel* property value

In order to retrieve the value of a property value on a specific ballast:

1. In the methods dropdown select *GET*.

   ```
   https://<DALI_PRO2_IP>/api/bmsapi/dali-
   devices/<BALLAST_GUID>/property/<PROPERTY_NAME>/last
   ```

   Where:

   - <BALLAST_GUID> shall be substituted with the *guid* of the selected ballast.

- <PROPERTY_NAME> shall be substituted with the desired property name (e.g. *lightLevel*).

2. In the methods dropdown select *GET*.
3. Press the *Send* button.
4. The answer appearing in the bottom contains the actual value of the property as well as a timestamp of when the property was updated.

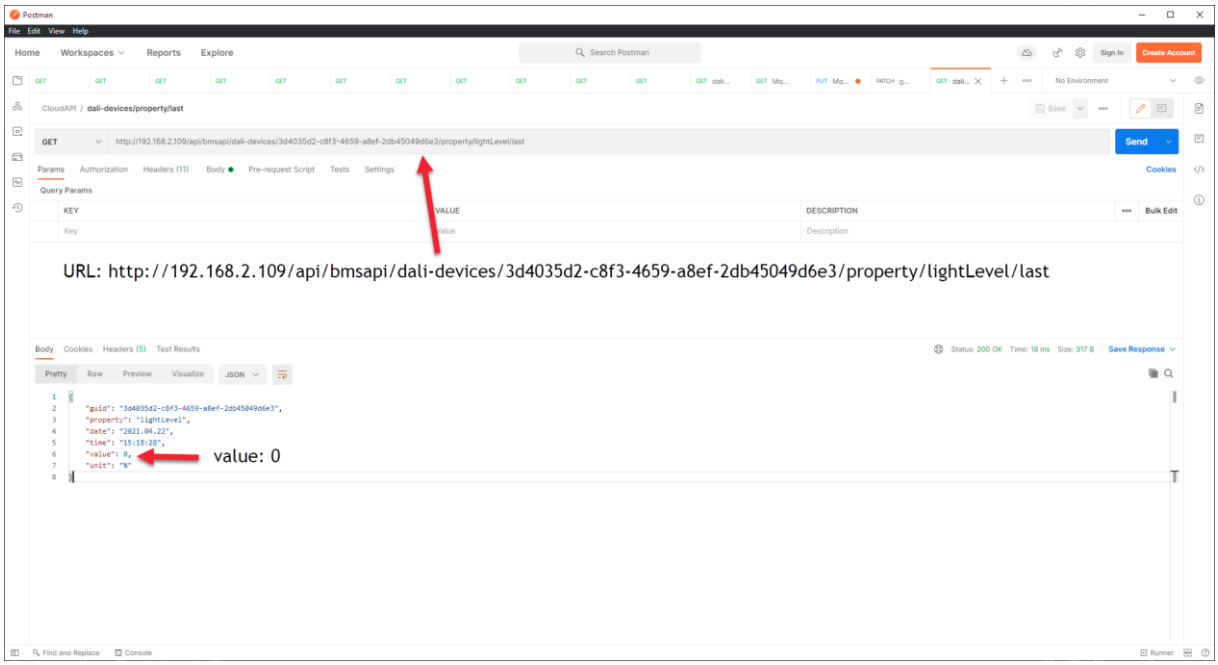

Figure 21

5. APIs

This part gives an overview of the REST APIs, grouped by functionality.

5.1 Application controller

These endpoints are useful to get basic information about the controller and to identify it.

5.1.1 GET /api/bmsapi/application-controller/info

Command:

```
https://<DALI_PRO2_IP>/api/bmsapi/application-controller/info
```

Gets information about the device. Since version 3.1.10.x this endpoint doesn't require authentication token in the REST call header.

Response payload

```
{
  "title”: "DaliPro_7000001A",
  "gtin”: "4062172114189",
  "serial”: "7000001A",
  "firmwareVersion”: "1.13.2"
  "timeUtc": "2022-09-15T12:24:33.000Z"
}
```

The *title* is equal to the host name of the DALI PRO 2 IoT controller in the LAN network.

The *timeUtc* is the current time on the controller.

5.1.2 GET /api/bmsapi/application-controller/state

Command:

```
https://<DALI_PRO2_IP>/api/bmsapi/application-controller/state
```

Gets information about the current configuration of the controller.

Response payload

```
{
  "running": true,
  "plugAndPlayMode": false,
  "applicationGuid": "30a96a5f-fc88-469e-ad99-f2468e759cdf",
  "configName": "Demo Application,
  "creator": "DALI Professional 3 v3.1.13 (Build 20504)",
  "dateUpload": "2022.09.8",
  "timeUpload": "12:29:58",
  "timeUtc": "2022-09-15T12:25:38.000Z"
}
```

The state *running* is false when a DALI device search is on-going, while uploading a new configuration or when a test function is in use.

The state *plugAndPlayMode* is true when no valid configuration is present in the controller (in this case *applicationGuid* does not exist).

The *applicationGuid* changes every time a new configuration is uploaded. The *applicationGuid* is therefore updated not only when a different functionality is introduced in the last valid configuration but also when a device is replaced/removed/added or even when a single parameter in the configuration is modified.

The properties *configName*, *creator*, *dateUpload*, *timeUpload* collects information about the last project upload (that changed the *applicationGuid*).

The *timeUtc* is the current time on the controller.

5.1.3 GET /api/bmsapi/api/bmsapi/application-controller/identify-start

Command:

```
https://<DALI_PRO2_IP>/api/bmsapi/application-controller/identify-start
```

The device "plug and play" led starts blinking for 30 s. There is no payload and no answer.

5.1.4 GET /api/bmsapi/application-controller/identify-stop

Command:

```
https://<DALI_PRO2_IP>/api/bmsapi/application-controller/identify-stop
```

The device "plug and play" led stops blinking. There is no payload and no answer.

5.1.5 GET /api/bmsapi/api/bmsapi/application-controller/time

Command:

```
https://<DALI_PRO2_IP>/api/bmsapi/application-controller/time
```

Gives the current time of the controller.

Response payload


```
{
  "date": "2021.11.18",
  "time": "09:26:47",
  "summer": false,
  "ntp": true
}
```

5.2 Light control

These endpoints are meant to control the lights and retrieve information on the installation status.

5.2.1 GET /api/bmsapi/groups

Command:

```
https://<DALI_PRO2_IP>/api/bmsapi/groups
```

Gets information about all existing light groups. To access the properties of a light group, select in the response payload the group of interest from the list *lightGroupList*. Notice that indexing starts from 0 and the index corresponds to the *groupId* itself.

Response payload


```
{
  "lightGroupList": [
    {
      "groupId": 0,
      "colorType": "nothing",
      "title": "Group Demo"
    },
    {
      "groupId": 1,
      "colorType": "tw",
      "title": "Tunable White Demo",
      "scenes": [
        {
          "title": "Scene A",
          "sceneNr": 0
        },
        {
          "title": "Scene B",
          "sceneNr": 1
        },
        {
          "title": "Scene C",
          "sceneNr": 2
        }
      ]
    }
  ]
}
```


For detailed information refer to the next GET function.

5.2.2 GET /api/bmsapi/groups/{groupId}

Command:

```
https://<DALI_PRO2_IP>/api/bmsapi/groups/{groupId}
```

Gets information about a specific group. The *groupId* is the Id of the group.

Response payload

```
{
  "groupId": 1,
  "colorType": "tw",
  "title": "Tunable White Demo",
  "scenes": [
    {
      "title": "Scene A",
      "sceneNr": 0
    },
    {
      "title": "Scene B",
      "sceneNr": 1
    },
    {
      "title": "Scene C",
      "sceneNr": 2
    }
  ]
}
```


The *scenes* array contains all configured scenes in the light group.

The *colorType* property informs about the supported color type of the light group:

colorType	Description
nothing	Only white light
tw	Tunable white: color temperature is changeable
rgb	Color is changeable by the red - green - blue components
rgbw	Color is changeable by the red - green - blue - white components

Depending on the *colorType*, the light group supports different parameters.

5.2.3 GET /api/bmsapi/groups/{groupId}/state

Command:

```
https://<DALI_PRO2_IP>/api/bmsapi/groups/{groupId}/state
```

Gets the light state of a specific group.

Response payload

The response payload contains the properties of the light group identified by the *groupId*, depending on the supported color type.

The basic parameters are:


```
{
  "groupId": 0,
  "lightState": false,
  "level": 0,
  "levelDali": 0
}
```

The *lightState* property informs, as Boolean value, if the light is on or off.

The *level* property is the current light level as percentage (0-100) of the full-scale range.

The *levelDali* property is the current level as DALI arc power level with values in the range 0…254.

When a light group contains scenes and one of them is recalled, the response payload does not have any information about the level because the scene typically embeds several light levels, not only one.

```
{
  "groupId": 0,
  "lightState": true,
  "scene": "Scene B",
  "sceneNr": 1
}
```

Scene are always interpreted as light on.

The *scene* property contains the name of the active scene.

The *sceneNr* property is the internal reference number of this scene.

When there is no active scene, the *sceneNr* value is (-1).

If no scene is recalled the different color types provide additional information.

Additional parameters of "colorType”:” tw"

```
{
  ...
  "twMired": 262,
  "twKelvin": 3817
}
```

The *twMired* property is the color temperature with values in mired: (x mired = y Kelvin / 1000000).

The *twKelvin* property is the color temperature with values in Kelvin.

Additional parameters for "colorType”: "rgb"

```
{
  ...
  "rgb”: 65535,
  "red”: 0,
  "green”: 255,
  "blue”: 255
}
```

The *rgb* property contains the numeric value of the color in hex format 0xRRGGBB with RR for red (0x00…0xFF), GG for green (0x00…0xFF) and BB for blue (0x00…0xFF). The properties *red, green* and *blue* show the single component values.

Additional parameters for "colorType”: "rgbw"

```
{
  ...
  "rgb”: 65535,
  "red”: 0,
  "green”: 255,
  "blue”: 255
  "white”: 30
}
```

The *rgb, red, green* and *blue* parameters represent the RGB color part equal to “colorType” = “rgb” as described in the previous paragraph.

The parameter *white* is the additional white part with values in the range 0…511. Values from 0 to 255 modulate the white component in the same way as red, green or blue are controlled. Values from 256 to 511 set the white component at the max value (0xFF) while the red, green and blue parts are gradually reduced until off when the value 511 is reached. In this case, the light will have only the white component.

Depending on the actual configuration, additional status properties can be provided:

```
{
  ...
  "effectState”: true,
  "effectNr”: 3,
  "effectTitle”: "Spring",
  "motionBlocked”: true,
  "regulationRunning”: true,
  ...
}
```

- Properties *effectState*, *effectNr*, *effectTitle* are present only if effect exists.
- Property *motionBlocked* is present if there is a function blocking motion detection.
- Property *regulationRunning* is present only if regulation exists.

5.2.4 PUT /api/bmsapi/groups/{groupId}/state

Command:

```
https://<DALI_PRO2_IP>/api/bmsapi/groups/{groupId}/state
```

Changes the light state of a certain group. Notice that the same functionality can be obtained with *PATCH*.

Request payload

A new light level can be specified as percentage of the full-scale range:

```
{
  "level": 80
}
```

A scene can be recalled by scene name:

```
{
  "scene": "Scene B"
}
```

or by the scene number (equal to the DALI scene number).

```
{
  "sceneNr": 2
}
```

Optional fade time

Default the Light change is as fast as possible (fade time 0). With the additional parameter

```
  "fadeTime": 6,
   ...
```

can be set the used fade time of the light change action of this rest call. The parameter is the DALI number of the fade time:

fade time value	time for the light change action
0	0 s
1	0.7 s
2	1 s
3	1.4 s
4	2 s
5	2.8 s
6	4 s
7	5.7 s
8	8 s
9	11.3 s
10	16 s
11	22.6 s
12	32 s
13	45.3 s
14	64.0 s
15	90.5 s

Optional parameters for "colorType" = "tw"

The *twMired* parameter is used to change the color temperature in unit mired (=1000000/colorTemperatur).

Important note: the three dots (...) in the below payloads must be replaced with the non-optional parameter *level* representing the desired level. This parameter is independent of the colors.

Examples: 2500 K - 400 mired, 3000 K - 333 mired, 4000 K - 250 mired, 5000 K – 200 mired

```
{
  ...
  "twMired": 250
}
```

When the temperature is specified in Kelvin, *twKelvin* shall be used:

```
{
```

```
  ...
  "twKelvin": 4000
}
```

Optional parameters for "colorType" = "rgb"

This parameter is meant to change the RGB colors using the format 0xRRGGBB (RR - red, GG - green, BB - blue) as decimal number.

Example:	
Only Red	0xFF0000 = 16711680
Only Green	0x00FF00 = 65280
Only Blue	0x0000FF = 255

Notice that the values 0xFEF700 and 0x040200 will mix red and green components with different ratios without blue component.

```
{
  ...
  "rgb”: 65280,
  "red”: 0,
  "green”: 255,
  "blue”: 0
}
```

The *rgb* parameter is first evaluated then the *red*, *green* and *blue* components are evaluated. In case of mismatch between *rgb* and the set *red*, *green*, *blue*, the *rgb* property is ignored and *red*, *green* and *blue* parameters are considered.

Optional parameters for the "colorType" = "rgbw"

The *rgb, red, green* and *blue* parameters represent the RGB color part equal to “colorType” = “rgb” as described in the previous paragraph.

The parameter *white* is the additional white part with values in the range 0…511. Values from 0 to 255 modulate the white component from zero to the max value (0xFF). Values from 256 to 511 set the white component at the max value (0xFF) while the red, green and blue parts are gradually reduced until off when the value 511 is reached.

```
{
  ...
  "rgb": 65280,
  "red": 0,
  "green": 255,
  "blue": 0,
  "white": 30,
}
```

5.3 DALI device properties

These endpoints can be used to get information about the connected DALI devices and their dynamic properties. The full list of device properties (depending on the control gear type) is reported in paragraph 6.1 of Appendix A.

5.3.1 GET /api/bmsapi/dali-devices

Command:

```
https://<DALI_PRO2_IP>/api/bmsapi/dali-devices
```

Gets information about all the connected devices.

Response payload


```
{
  "deviceList": [
    {
      "guid": "e516322a-d7c1-4d2a-b0a4-5c9e9457cf1c",
      "type": "gear",
      "lightGroupId": 0,
      "title": "EVG A63",
      "port": 0,
      "shortAddress": 63,
      "staticProperties": [
        {
          "property": "oemGtin",
          "text": "1234"
        },
        {
          "property": "nominalInputPower",
          "value": 120,
          "unit": "W"
        },
        {
          "property": "colourRenderingIndex",
          "value": 89
        },
        {
          "property": "colourTemperature",
          "value": 3000,
          "unit": "K"
        },
        {
          "property": "luminaireIdentification",
          "text": "Test Leuchte33"
        },
        {
          "property": "ratedMedianUsefullLifeOfLuminaire",
          "value": 100000,
          "unit": "h"
        }
      ],
      "properties": [
                    "lightLevel",
                    "gearStatus",
                    "lampFailure",
                    "driverEnergyConsumption",
                    "driverInputPower",
                    "driverApparentEnergyConsumption",
                    "driverInputApparentPower",
                    "loadEnergyConsumption",
                    "loadOutputPower",
                    "driverOperationTime",
                    "mainPowerUpCount",
                    "driverInputVoltage",
                    "inputFrequency",
                    "powerFactor",
                    "errorOverall",
                    "errorOverallCount",
                    "errorUndervoltage",
                    "errorUndervoltageCount",
```

```
                    "errorOvervoltage",
                    "errorOvervoltageCount",
                    "errorOutputPowerLimit",
                    "errorOutputPowerLimitCount",
                    "errorThermalDerating",
                    "errorThermalDeratingCount",
                    "errorThermalShutDown",
                    "errorThermalShutDownCount",
                    "driverTemperature",
                    "outputCurrentPercent",
                    "lampOnCountRelativ",
                    "lampOnCount",
                    "lampOperationTimeRelativ",
                    "lampOperationTime",
                    "outputVoltage",
                    "outputCurrent",
                    "errorLamp",
                    "errorLampCount",
                    "errorLampShortCircuit",
                    "errorLampShortCircuitCount",
                    "errorLampOpenCircuit",
                    "errorLampOpenCircuitCount",
                    "errorLampThermalDerating",
                    "errorLampThermalDeratingCount",
                    "errorLampThermalShutDown",
                    "errorLampThermalShutDownCount",
                    "lampTemperature",
                    "ratedMedianUsefullLifeOfLuminaire",
                    "ratedMedianUsefullLifeOfLuminaire",
                    "ratedMedianUsefullLifeOfLuminaire"
                  ]
  },
  {
    "guid": "72164943-5a77-4429-8ead-68dde5ba5430",
    "type": "gear",
    "lightGroupId": -1,
    "title": "Ballast A00",
    "port": 0,
    "shortAddress": 0,
    "zones": ["Zone 1"],
    "gtin": "4008321371560",
    "serial": "4294706693",
    "properties": ["lightLevel", "gearStatus", "lampFailure"]
  },
  {
    "guid": "6174ae79-b300-4bae-b23e-799d71e22fc4",
    "type": "gear",
    "lightGroupId": -1,
    "title": "Ballast A09",
    "port": 0,
    "shortAddress": 9,
    "zones": ["Zone 1","Zone 2"],
    "gtin": "4008321371560",
    "serial": "4294772229",
    "properties": ["lightLevel", "gearStatus", "lampFailure"]
  },
  {
    "guid": "5b38bddb-2fe9-4301-b0c9-6a8a019b829b",
    "type": "gear",
    "lightGroupId": -1,
    "title": "Ballast A10",
    "port": 0,
    "shortAddress": 10,
    "gtin": "4008321371560",
    "serial": "4294837765",
    "properties": ["lightLevel", "gearStatus", "lampFailure"]
  },
  {
    "guid": "5cfcee97-4428-413a-a8a5-f41ab93e1806",
```

```
      "type": "gear",
      "lightGroupId": -1,
      "title": "Ballast A20",
      "port": 0,
      "shortAddress": 20,
      "gtin": "4008321371560",
      "serial": "4294903301",
      "properties": ["lightLevel", "gearStatus", "lampFailure"]
    }
  ]
}
```

Here below an example of devices' combination implementing a mixture of lights i.e. tunable white with warm and cold components:

```
{
  "deviceList": [
    {
      "guid": "6db6439c-5307-42f6-8755-96dd68d6d165",
      "type": "gear",
      "title": "Ballast A00",
      "port": 0,
      "shortAddress": 0,
      "mixType": "TW",
      "mixTitle": "Tunable White 1",
      "mixPart": "could",
      "zones": [
        "Zone 1"
      ],
      "properties": [
        "lightLevel",
        "gearStatus",
        "lampFailure",
        "errorBits"
      ]
    },
    {
      "guid": "e8214b25-545d-4f11-8c89-65913f8f55d0",
      "type": "gear",
      "title": "Ballast A09",
      "port": 0,
      "shortAddress": 9,
      "mixType": "TW",
      "mixTitle": "Tunable White 1",
      "mixPart": "warm",
      "zones": [
        "Zone 1"
      ],
      "properties": [
        "lightLevel",
        "gearStatus",
        "lampFailure",
        "errorBits"
      ]
    },
  ]
}
```

The property *mixType* describes the type of light mixture i.e. TW, RGB, RGBW. The property *mixTitle* is the title of the devices' combination and the *mixPart* represents the light component implemented by the device i.e. warm, could, red, green, blue, white. Each part can contain more the one device.

For color device type (DT8) are additional information about the used color type.

```
      ...,
      "colorType": "tw",
      "twKelvinMin": 8000,
```

```
      "twKelvinMax": 2500,
      ...
```

The values can be "tw", "rgb" or "rgbw" and additional for tw the color temperature limits of the device.

5.3.2 GET /api/bmsapi/dali-devices/{guid}

Command:

```
https://<DALI_PRO2_IP>/api/bmsapi/dali-devices
```

Get information about the selected devices.

Response payload

```
{
  "guid": "77a4bf85-56b0-460b-96c0-68e248cec0cd",
  "type": "gear",
  "groupId": 0,
  "title": "Ballast B04",
  "port": 1,
  "shortAddress": 4,
  "emergencyLight": false,
  "gtin": "4008321371560",
  "gtinLabel": "OTi DALI 75220...24024 1...4 CH",
  "serial": "4294706693",
  "zone": "Zone 1",
  "zones": ["Zone 1"],
  "properties": ["lightLevel", "gearStatus", "lampFailure", "errorBits"]
}
```

5.3.3 GET /api/bmsapi/dali-devices/{guid}/property/{property}/active

Command:

```
https://<DALI_PRO2_IP>/api/bmsapi/device-devices/{guid}/property/{property}/active
```

A new request is sent on the DALI line to query the property of the specified device (*guid*). This can take some time and it is permitted only when the controller is in run state. In case of success, the value of the property is stored or updated in the DALI PRO 2 IoT.

For more details about error management please refer to paragraphs 6.3 and 6.5 in Appendix A.

Response payload

If the answer cannot be delivered within x seconds, the response is 503. If the device and the property exist and the query is successfully completed the response is 200, otherwise the response is 404. The response payload is:

```
{
  "guid": "182fd8e8-a6db-496d-931f-3d7d81ca36d7",
  "property": "driverInputVoltage",
  "date": "2021.01.27",
  "time": "16:37:19",
  "value": 225.7,
  "unit": "Vrms"
}
```

5.3.4 GET /api/bmsapi/dali-devices/{guid}/property/{property}/last

Command:

```
https://<DALI_PRO2_IP>/api/bmsapi/dali-devices/{guid}/property/{property}/last
```

Gets the most recent value of the property of the specified ballast (*guid*) stored in the DALI PRO 2 IoT.

Important note: this command does not issue a new query to the ballast on the DALI line. It just returns the most recent value, stored in the DALI PRO 2 IoT, that has been queried from the ballast in the past. In order to issue a new property query on the DALI line to the ballast, use *GET active* command (see paragraph 5.3.3). This means that subsequent calls of *GET last* command might in principle always return the same payload, at least until a new *GET active* command is sent.

Response payload

Error code 503 is given when the last value does not exist.


```
{
  "guid": "182fd8e8-a6db-496d-931f-3d7d81ca36d7",
  "property": "driverInputVoltage",
  "date": "2021.01.27",
  "time": "16:37:19",
  "value": 225.7,
  "unit": "Vrms"
}
```


5.3.5 GET /api/bmsapi/dali-devices/{guid}/property/{property}/history

Command:

```
https://<DALI_PRO2_IP>/api/bmsapi/device-devices/{guid}/property/{property}/history
```

All stored values for this property are returned as an array. Consider that for each property only a limited number of results are saved. A value is stored only when it changes or if the value type is an event (for example button input event).

Response payload

If there is no stored data, the array is empty. If the device (or property) does not exist, the response gives error code 404.


```
{
  "guid": "182fd8e8-a6db-496d-931f-3d7d81ca36d7",
  "property": "driverInputVoltage",
  "unit": "Vrms",
  "propertyData": [
    {
      "date": "2021.01.27",
      "time": "13:53:59",
      "value": 224.6
    },
    {
      "date": "2021.01.27",
      "time": "13:55:55",
      "value": 224.4
    },
    {
      "date": "2021.01.27",
      "time": "14:04:27",
      "value": 224.7
    }
  ]
```

```
}
```

5.3.6 GET /api/bmsapi/dali-devices/error

Command:

```
https://<DALI_PRO2_IP>/api/bmsapi/dali-devices/error
```

Gets information about all connected devices with errors. If an empty list is returned, use *GET active* command (see paragraph 5.3.3) to acquire values from the devices and repeat the query.

Response payload


```
{
  "deviceList": [
    {
      "guid": "6db6439c-5307-42f6-8755-96dd68d6d165",
      "type": "gear",
      "title": "Ballast A00",
      "port": 0,
      "shortAddress": 0,
      "error": true,
      "errorBits": 4,
      "errors": [
        "errorLamp"
   ]    },
    {
      "guid": "db0a2d4b-1a3e-49f3-9c74-1ef400e02603",
      "type": "gear",
      "title": "Ballast A20",
      "port": 0,
      "shortAddress": 20,
      "error": true,
      "emergencyErrorBits": 128,
      "emergencyErrors": [
        "errorDurationTestOutdated"
      ]
    }
  ]
}
```


5.3.7 GET /api/bmsapi/dali-devices/{guid}/error

Command:

```
https://<DALI_PRO2_IP>/api/bmsapi/dali-devices/{guid}/error
```

Gives the state and the error state of the specified device (*guid*). It is internally using the property *errorBits*.

For more details about error management please refer to paragraphs 6.3 and 6.5 in Appendix A.

Response payload

The payload gives the *errorBits* as single Boolean properties. If the device an emergency light device, it contains also the property *emergencyErrorBits*.

DALI input devices containing the property *errorBitsCoupler*.

```
{
  "error": true,
  "errorBits": 4,
  "errors": [
    "errorLamp"
  ],
  "emergencyErrorBits": 128,
  "emergencyErrors": [
    "errorDurationTestOutdated"
  ]
}
```

Example of no error:

```
{
  "error": false
}
```

5.4 DALI device property scheduler

The current values of the DALI devices must be actively queried over the DALI line. Depending on the number of properties monitored, several DALI commands might be needed. The limited bandwidth of the DALI line is a bottleneck in the accessing these data. For this reason, only a selected number of properties can be continuously queried in an application.

The property scheduler is managing the queries of the properties based on profiles. A profile contains a list of device property names and update intervals. If a profile is active, the scheduler queries each device that is supporting this property periodically according to the specified interval.

Additionally, it is possible to add for a single device a single property (including the update interval) to the scheduler.

The result of a query is pushed by MQTT and can be retrieved by the REST endpoints (for the last or a limited list of history data). Each result value is provided together with the data and time of the query action. Only value changes are considered (except for events, for example for motion sensors).

5.4.1 GET /api/bmsapi/property-scheduler-profiles/data

Command:

```
https://<DALI_PRO2_IP>/api/bmsapi/property-scheduler-profiles/data
```

Gets all existing scheduler profiles by *profile* name.

Response payload

List of profiles.

```
{
  "profiles": ["DeviceError", "Emma"]
}
```

5.4.2 GET /api/bmsapi/property-scheduler-profiles/data/{profile}

Command:

```
https://<DALI_PRO2_IP>/api/bmsapi/property-scheduler-profiles/data/{profile}
```

Gets the description of a single profile by *profile* name.

Response payload

Gets all the properties with the related interval times (in seconds with range 1…65535, max roughly 18.2 hours).

```
{
  "profilItems": [
    {
      "property": "driverOperationTime",
      "interval": 10
    },
    {
      "property": "mainPowerUpCount",
      "interval": 20
    },
    {
      "property": "lampOnCount",
      "interval": 30
    }
  ]
}
```

5.4.3 PUT /api/bmsapi/property-scheduler-profiles/data/{profile}

Command:

```
https://<DALI_PRO2_IP>/api/bmsapi/property-scheduler-profiles/data/{profile}
```

Adds a new profile named *{profile}*. The profile is permanently stored on the device and any existing profile with the same name is replaced. The format of the profile title must be host name compatible.

Request payload

The payload contains an array of properties names and interval times (in seconds, with range 1…65535, max roughly 18.2 hours).

```
{
  "profilItems": [
    {
      "property": "driverInputVoltage",
      "interval": 10
    },
    {
      "property": "driverEnergyConsumption",
      "interval": 20
    },
    {
      "property": "driverInputPower",
      "interval": 30
    }
  ]
}
```

5.4.4 DELETE /api/bmsapi/property-scheduler-profiles/data/{profile}

Command:

```
https://<DALI_PRO2_IP>/api/bmsapi/property-scheduler-profiles/data/{profile}
```

Deletes a profile by *profile* name.

5.4.5 GET /api/bmsapi/property-scheduler-profiles/active

Command:

```
https://<DALI_PRO2_IP>/api/bmsapi/property-scheduler-profiles/active
```

Gets all active profiles listed by *profile* name.

5.4.6 PUT /api/bmsapi/property-scheduler-profiles/active/{profile}

Command:

```
https://<DALI_PRO2_IP>/api/bmsapi/property-scheduler-profiles/active/{profile}
```

Activates a profile by *profile* name.

5.4.7 DELETE /api/bmsapi/property-scheduler-profiles/active/{profile}

Command:

```
https://<DALI_PRO2_IP>/api/bmsapi/property-scheduler-profiles/active/{profile}
```

Deactivates a profile by *profile* name.

5.5 Emergency light

These endpoints can be used with self-containing emergency lighting devices that implement DALI part 202 (Device type 1).

Emergency lighting devices support two types of tests called *function test* and *duration test*. These tests shall be automatically and periodically executed within a specified interval of time. The test results shall then be stored and made available to the user.

To use the emergency light functionality, the related feature must be activated on the device (please refer to paragraph 5.2).

5.5.1 GET /api/bmsapi/dali-devices/emergency-light

Command:

```
https://<DALI_PRO2_IP>/api/bmsapi/dali-devices/emergency-light
```

Gets information about all connected devices with self-contained emergency lighting.

Response payload

List of all emergency lighting devices.


```
{
  "deviceList": [
    {
      "guid": "a6d7d0ce-fbf1-4db3-80d7-93e3a246e7dd",
      "title": "Ballast A02",
      "port": 0,
      "shortAddress": 2,
      "forceInhibit": false,
      "state": "normalMode",
      "batteryCharge": "53 %",
      "functionTestDateTime": "2021.09.01T11:20:47",
      "functionTestResult": "ok",
      "nextFunctionTestDateTime": "2021.09.29T11:20:47",
      "durationTestDateTime": "2021.09.02T14:22:17",
      "durationTestResult": "ok",
      "durationTestResultTime": "3:00:00",
      "durationTestResultValue": 180,
      "nextDurationTestDateTime": "2021.12.02T14:22:17",
      "functionTestPending": false,
      "durationTestPending": false,
      "failureStatus": 32
    },
    {
      "guid": "a6d7d3ce-acf3-4db9-82d8-98e3a366e6ef",
      "title": "Ballast A03",
      "port": 0,
      "shortAddress": 3,
      "forceInhibit": false,
      "state": "normalMode",
      "batteryCharge": "100 %",
      "functionTestDateTime": "2021.09.02T11:21:47",
      "functionTestResult": "ok",
      "nextFunctionTestDateTime": "2021.09.28T11:21:47",
      "durationTestDateTime": "2021.09.03T14:22:17",
      "durationTestResult": "error",
      "durationTestResultTime": "2:50:00",
      "durationTestResultValue": 170,
      "nextDurationTestDateTime": "2021.12.02T14:22:17"
      "functionTestPending": false,
      "durationTestPending": false,
      "failureStatus": 200

    }
  ]
}
```


Please refer to paragraph 6.1.5 in Appendix A for the values that the variable *state* can take.

5.5.2 GET /api/bmsapi/dali-devices/emergency-light/config

Command:

```
https://<DALI_PRO2_IP>/api/bmsapi/dali-devices/emergency-light/config
```

Gets the configuration parameter of the emergency lighting functionality.

Response payload

```
{
  "functionTestInterval”: 28,
  "durationTestInterval”: 11,
  "functionTest": true,
  "durationTest": true,
  "maxRunTestInZone”: 3,
  "testWindowStartHour”: -1,
  "testWindowEndHour”: -1
}
```

The property *functionTestInterval* represents the interval in days between two function tests on the same device, while the property *durationTestInterval* is the number of months between two duration tests.

5.5.3 PUT /api/bmsapi/dali-devices/emergency-light/config

Command:

```
https://<DALI_PRO2_IP>/api/bmsapi/dali-devices/emergency-light/config
```

Changes one or more configuration parameters.

Request payload

The payload contains the parameters to change.

```
{
  "testWindowStartHour”: 22,
  "testWindowEndHour”: 3
}
```

In this example the duration test can only start within a time window between 22:00 and 3:00. The test can then run over the window end-time, until it is finished. A duration test typically takes 3 hours.

The *testWindowStartHour* and *testWindowEndHour* are affecting only duration test.

Response payload

The response is the new settings.

```
{
  "functionTestInterval”: 28,
  "durationTestInterval”: 11,
  "functionTest": true,
  "durationTest": true,
  "maxRunTestInZone”: 3,
  "testWindowStartHour”: -1,
  "testWindowEndHour”: -1
}
```

5.5.4 Get /api/bmsapi/dali-devices/emergency-light/report-file

Command:

```
https://<DALI_PRO2_IP>/api/bmsapi/dali-devices/emergency-light/report-file
```

Updates and retrieves the report file *Result.csv.*

Response payload

The report file.

```
Device guid;Device name;Area name;Zone name;port;short address;Current ELT
state;Function test result;Function test date;Duration test result;Duration
test data;Duration test length

f8cc4214-140e-4d09-b064-8cfa1662c3b3;Ballast
B08;;;B;8;normalMode;ok;2022.02.22 07:51:33;ok;2022.02.22
07:37:38;10800.0s;

160c796c-43ce-4600-8b1a-3d5266a26d5e;Ballast
B11;;;B;11;normalMode;error;2022.02.21 17:15:55;error;2022.02.21
18:34:58;0.0s;
```

5.5.5 GET /api/bmsapi/dali-devices/{guid}/emergency-light

Command:

```
https://<DALI_PRO2_IP>/api/bmsapi/dali-devices/{guid}/emergency-light
```

Gives the current state of the specified (*guid*) self-contained emergency lighting device.

Response payload

```
{
  "guid": "a6d7d0ce-fbf1-4db3-80d7-93e3a246e7dd",
  "title": "Ballast A02",
  "port": 0,
  "shortAddress": 2,
  "forceInhibit": false,
  "state": "normalMode",
  "batteryCharge": "53 %",
  "functionTestDateTime": "2021.09.01T11:20:47",
  "functionTestResult": "ok",
  "nextFunctionTestDateTime": "2021.09.29T11:20:47",
  "durationTestDateTime": "2021.09.02T14:22:17",
  "durationTestResult": "ok",
  "durationTestResultTime": "3:00:00",
  "durationTestResultValue": 180,
  "nextDurationTestDateTime": "2021.12.02T14:22:17"
}
```

The properties *nextFunctionTestDateTime* and *nextDurationTestDateTime* give the date-time of the next planned function and duration tests. The actual tests start might then be delayed for other reasons.

Please refer to paragraph 6.1.5 in Appendix A for the values that the variable *state* can take.

5.5.6 GET /api/bmsapi/dali-devices/{guid}/emergency-light/history

Command:

```
https://<DALI_PRO2_IP>/api/bmsapi/dali-devices/{guid}/emergency-light/history
```

Gives the list of tests already run on the specified (*guid*) self-contained emergency lighting device.

The test result is stored in the device in a text file. Any new test result is appended to this file. These data are persistent even to a new upload as long the *guid* of the device is not changed.

Response payload


```
{
  "testResultList": [
    {
      "type": "function",
      "dateTime": "2021.06.01T11:20:47",
      "testResult": "ok"
    },
    {
      "type": "duration",
      "dateTime": "2021.06.02T14:22:17",
      "testResult": "ok",
      "durationTestResultTime": "7:00:00"
    },
    {
      "type": "function",
      "dateTime": "2021.08.01T11:20:47",
      "testResult": "ok"
    },
    {
      "type": "duration",
      "dateTime": "2021.08.02T14:22:17",
      "testResult": "ok",
      "durationTestResultTime": "6:20:00"
    },
    {
      "type": "function",
      "dateTime": "2021.09.01T11:20:47",
      "testResult": "ok"
    },
    {
      "type": "duration",
      "dateTime": "2021.09.02T14:22:17",
      "testResult": "ok",
      "durationTestResultTime": "3:20:00"
    }
  ]
}
```


5.5.7 POST /api/bmsapi/dali-devices/{guid}/emergency-light/start-inhibit

Command:

```
https://<DALI_PRO2_IP>/api/bmsapi/dali-devices/{guid}/emergency-light/start-inhibit
```

Triggers the inhibit mode in the specified (*guid*) device by sending the command every 10 minutes. Upon the command arrival, the device enters and stays in inhibit mode for 15 minutes. If there is a mains interruption while the device is in inhibit state, the device does not switch into emergency mode.

The inhibit mode can only be triggered when the device is in normal mode. If not possible the answer is 503.

5.5.8 POST /api/bmsapi/dali-devices/{guid}/emergency-light/stop-inhibit

Command:

```
https://<DALI_PRO2_IP>/api/bmsapi/dali-devices/{guid}/emergency-light/stop-inhibit
```

Stops the inhibit mode for the specified (*guid*) device.

The inhibit mode can only be stopped when the device is in inhibit mode. If the device is not in inhibit mode, the answer is 503.

5.5.9 POST /api/bmsapi/dali-devices/{guid}/emergency-light/start-function-test

Command:

```
https://<DALI_PRO2_IP>/api/bmsapi/dali-devices/{guid}/emergency-light/start-function-test
```

Reschedules the next function test to start as soon as possible for the specified (*guid*) device.

5.5.10 POST /api/bmsapi/dali-devices/{guid}/emergency-light/start-duration-test

Command:

```
https://<DALI_PRO2_IP>/api/bmsapi/dali-devices/{guid}/emergency-light/start-duration-test
```

Reschedules the next duration test to start as soon as possible for the specified (*guid*) device.

5.5.11 POST /api/bmsapi/dali-devices/{guid}/emergency-light/stop-test

Command:

```
https://<DALI_PRO2_IP>/api/bmsapi/dali-devices/{guid}/emergency-light/stop-test
```

Stops any running function or duration test. If no test is running the answer is 503.

6. Appendix A

6.1 DALI control gear properties

Depending on the supported device types, some properties might not be available.

Property Id	Description	Mem Bank
lightLevel	DALI control gear light level	-
gearStatus	DALI control gear status byte	-
lampFailure	DALI control gear lamp error bit	-
emergencyMode	DALI part 201 emergency mode byte	-
emergencyStatus	DALI part 201 emergency status byte	-
emergencyBatteryCharge	DALI part 201 emergency battery charge in percentage	-
emergencyLampEmergencyTime	DALI part 201 emergency lamp operation time	-
emergencyLampTotalOperationTime	DALI part 201 emergency device operation time	-

6.1.1 Energy consumption, DALI-part 252, device type 51

Property Id	Description	Mem Bank
driverInputPower	Active input power [W]	202
driverEnergyConsumption	Active input energy [Wh]	202
driverInputApparentPower	Apparent input power [W]	203
driverApparentEnergyConsumption	Apparent input energy [Wh]	203
driverLoadsidePower	Output power [W]	204
driverLoadsideEnergy	Output energy [Wh]	204

6.1.2 Control gear diagnostics and maintenance, DALI-part 253, device type 52

Method	Description	Mem Bank
driverOperationTime	Driver operation time [s]	205
mainPowerUpCount	Driver power cycle counter	205

Method	Description	Mem Bank
driverInputVoltage	Driver input voltage [Vrms]	205
powerFactor	Driver power factor	205
inputFrequency	Driver input frequency [Hz]	205
errorOverall	Overall error flag	205
errorOverallCount	Overall-Error counter	205
errorUndervoltage	Driver input undervoltage detection flag	205
errorUndervoltageCount	Driver input undervoltage detection counter	205
errorOvervoltage	Driver input overvoltage error flag	205
errorOvervoltageCount	Driver input overvoltage error counter	205
errorOutputPowerLimit	Driver output power limitation flag	205
errorOutputPowerLimitCount	Driver output power limitation count	205
errorThermalDerating	Driver thermal derating flag	205
errorThermalDeratingCount	Driver thermal derating counter	205
errorThermalShutDown	Driver over temperature shutdown flag	205
errorThermalShutDownCount	Driver over temperature shutdown counter	205
driverTemperature	Driver temperature [°C]	205
outputCurrentPercent	Real output arc power [%]	205

6.1.3 Light source diagnostics and maintenance, DALI-part 253, device type 52

Method	Description	Mem Bank
lampOnCountRelativ	Lamp switch on counter, resettable	206
lampOnCount	Lamp switch on counter	206
lampOperationTimeRelativ	Lamp operation timer, resettable [s]	206
lampOperationTime	Lamp operation timer [s]	206
outputVoltage	Driver output voltage [V]	206
outputCurrent	Driver output current [A]	206
errorLamp	Lamp error flag	206
errorLampCount	Lamp error counter	206
errorLampShortCircuit	Lamp short circuit error flag	206

Method	Description	Mem Bank
errorLampShortCircuitCount	Lamp short circuit error counter	206
errorLampOpenCircuit	Lamp open circuit detection counter	206
errorLampOpenCircuitCount	Lamp open circuit detection counter	206
errorLampThermalDerating	Lamp thermal derating flag	206
errorLampThermalDeratingCount	Lamp thermal derating counter	206
errorLampThermalShutDown	Lamp over temperature shutdown flag	206
errorLampThermalShutDownCount	Lamp over temperature shutdown counter	206
lampTemperature	Lamp temperature [°C]	206

6.1.4 Device information data, DALI-part 251, device type 50 and DALI-part 253, device type 52

The device information are static data, generated during the commissioning phase. Base data are read out from the memory banks and set by the luminaire manufacturer. Some ECG specific data items can be changed in the ECG property page of the *PCTool*.

Method	Description	Mem Bank
oemGtin	GTIN of the luminary Property "OEM GTIN" in PC-Tool	1
oemSerial	Serial number of the luminary	1
luminaireYearOfManufacture	Year of the luminary manufacture	1
luminaireWeekOfManufacture	Week of the luminary manufacture	1
nominalInputPower	Nominal input power [W] Property "Nominal input power" in PC-Tool	1
powerAtMinDimLevel	Power a minimum dim level [W]	1
nominalMinAcMainsVoltage	Nominal minimum AC mains voltage [V]	1
nominalMaxAcMainsVoltage	Nominal maximum AC mains voltage [V]	1
nominalLightOutput	Nominal light output in lumen [Lm]	1
colourRenderingIndex	Color rendering index of the light Property "Color rendering index " in PC-Tool	1
colourTemperature	Color temperature of the light [K] Property "Color temperature" in PC-Tool	1
lightDistributionType	Light distribution type (type I…type V)	1

Method	Description	Mem Bank
luminaireColor	Color as string	1
luminaireIdentification	Luminaire identification string Property "Luminaire identification" in PC-Tool	1
ratedMedianUsefullLifeOfLuminaire	Rated median useful lifetime in hour Property "Expected lifetime" in PC-Tool	207
internalControlGearReferenceTemperature	Reference temperature on the device for 25 °C ambient temperature	207
ratedMedianUsefulLightSourceStarts	Rated median lamp switch on actions	207

6.1.5 Emergency lighting *state*

Overview about the *state* property of an emergency lighting device:

State	Description
unknown	State after power-on until the first state check
sendQueryState	First state check running (take a very short time)
notConnected	Device is not connected (or not main powered and cannot communicate with battery power)
restMode	Device is not main powered and emergency mode is suppressed because forced in inhibit state
normalMode	Device is main powered and in the normal state
emergencyMode	Device is not mains powered and in the emergency mode lighting
extendedEmergencyMode	Device is mains powered again but in the emergency mode lighting
functionTest	Functional test is running
durationTest	Duration test is running

6.2 DALI input device (control device) properties

Depending on the supported device types, some properties might not be available.

Property Id	Description	Mem Bank
inputStatus	The input device status byte	-
inputEvent	The last input event for button and motion sensor	-
lightLevel	The light level from light sensor	-

6.3 Device error manager, property *errorBits*

The property *errorBits* (supported only by ECGs, for input device please refer to *errorBitsCoupler*) shall be dynamically managed. That's because in normal state only the device state is periodically queried. That means that, when a supported error is detected, the extended DALI data error properties start to be cyclically queried, until the general error disappears. The combined result is provided in the property *errorBits* as bit map (24-bit number).

Interpretation of the property *errorBits*. If the device is working as expected, the value is 0.

bit	related single propertyId	Interpretation
0		Device is missing
1	errorControlGear	Device error
2	errorLamp	Lamp error
3	reserved	
4	reserved	
5	reserved	
6	reserved	
7		
8		Device error, the device supports detailed error information
9	errorUndervoltage	Line power undervoltage detected
10	errorOvervoltage	Line power overvoltage detected
11	errorOutputPowerLimit	The output power will be limited
12	errorThermalDerating	The light is reduced due to overheating
13	errorThermalShutDown	The light is switched off due to overheating
14	reserved	
15	reserved	
16		Lamp error, the device supports detailed error information

bit	related single propertyId	Interpretation
17	errorLampShortCircuit	Short circuit on the output line
18	errorLampOpenCircuit	No lamp connected
19	errorLampThermalDerating	Light level reduces because lamp overheating
20	errorLampThermalShutDown	Light switched off because lamp overheating
21	reserved	
22	reserved	
23	reserved	

Comments:

- In a fault condition, the first query *GET active* returns in the response payload only partial information based on the status byte. In parallel the *GET active* queries of the specific properties start in order to populate the bits of the *errorBit* property. At this points a second query of the *errorBit* property contains the additional information with the detailed properties.

- Lamp failures cannot be detected if the current arc power level is 0 (light off).

6.3.1 Dependences for the control gear general error (DALI -part 253)

If the property *controlGearFailure* (part 102) or *errorOverall* (part 253) shows an error, the following properties are enabled:

Property Id	Interval time [min]	"errorFlags" bits
errorUndervoltage	15	bit 9
errorOvervoltage	15	bit 10
errorOutputPowerLimit	15	bit 11
errorThermalDerating	15	bit 12
errorThermalShutDown	15	bit 13

The results of these errors are reflected in the change of the property *errorBits*.

If the property *lampFailure* (part 102) or *errorLamp* (part 253) shows an error, the following properties are enabled:

Property Id	Interval time [min]	"errorLampFlags" bits
errorLampShortCircuit	15	bit 17
errorLampOpenCircuit	15	bit 18
errorLampThermalDerating	15	bit 19

Property Id	Interval time [min]	"errorLampFlags" bits
errorLampThermalShutDown	15	bit 20

The results of this errors are reflected in the change of the property *errorBits*.

6.4 Device error manager, property "emergencyErrorBits"

The property "emergencyErrorBits" (supported only by ECG with part 202 "Self-contained emergency lighting, device type 1) will be active managed.

The combined result will be provided in the property "errorBits" as bit map (24 bit number).

This property can not be active queried and is only the combination of the result of different actions.

Interpretation of the property "emergencyErrorBits". If the device ok, the value is 0.

bit	related errorId	Interpretation
0	errorMissing	Device is missing
1	errorCommunication	Communication error
2	reserved	
3	reserved	
4	errorFunctionTestFailed	Updated after the function test run
5	errorDurationTestFailed	Updated after the duration test run
6	errorFunctionTestOutdated	Updated after power on end once the day Outdated if the last test older then the set interval + 3 days
7	errorDurationTestOutdated	Updated after power on end once the day Outdated if the last test older then the set interval + 5 days
8	errorCircuit	Circuit failure, from emergency status bit 0
9	errorBatteryDuration	Battery duration failure, from emergency status bit 1
10	errorBattery	Battery failure, from emergency status bit 2
11	errorLamp	Emergency lamp failure, from emergency status bit 3
12	reserved	
13	reserved	
14	reserved	
15	reserved	
16	errorBatteryUnknown	Will set if DALI battery value MAKS
17	reserved	

bit	related errorId	Interpretation
18	reserved	
19	reserved	
20	reserved	
21	reserved	
22	reserved	
23	reserved	

6.5 Device error manager, property "errorBitsCoupler"

The property *errorBitsCoupler* is shall be dynamically managed. Only the device state is periodically queried. Depending on the answer or if the answer is missing, different bits are set.

The reason for not using *errorBit* is to support different query intervals for ECG and coupler and save some DALI bus bandwidth.

Interpretation of the property *errorBitsCoupler*. If the device ok, the value is 0.

bit	related single propertyId	Interpretation
0		Device is missing
1	errorDevice	Device error
2	reserved	
3	reserved	
4	reserved	
5	reserved	
6	reserved	
7	reserved	

6.6 Default scheduler profile "DeviceError" for error monitoring

propertyId	Interval time [s]
errorBits	300
errorBitsCoupler	3600

6.7 Default scheduler profile *Emma* for EM/MA

Property Id	Interval time [s]
driverInputVoltage	600
driverEnergyConsumption	300
driverInputPower	60
driverOperationTime	300
driverTemperature	60
errorOverall	3600
lampOperationTime	300
lightLevel	60
errorBits	300
errorBitsCoupler	3600

7. Appendix B

7.1 Code samples

The below snippet of Python code is an adaptation of the code generated by Postman.

Notice that the first function does not need an authentication token. Its purpose in fact is to return the authentication token that must then be used in the subsequent REST commands.

When a REST command is executed with success, the return value is *<Response [200]>* i.e. status *OK*.

To extract from the return value of a function the response payload in json format, apply the .json() method.

```
import requests
import json

def REST_Login():

    url = 'https://192.168.2.109/auth/login'

    payload = json.dumps({
      "username": "admin",
      "password": "psw12345"
    })
    headers = {
      'Authorization': '',
      'Content-Type': 'application/json'
    }

    r = requests.request("POST", url, headers=headers, data=payload)

    return r


 def REST_Get_Groups(auth_token):

    url = 'https://192.168.2.109/api/bmsapi/groups'

    payload = ""
    headers = {
      'Authorization': auth_token
    }

    r = requests.request("GET", url, headers=headers, data=payload)

    return r


def REST_Get_Group_State(auth_token, group):

    url = 'https://192.168.2.109/api/bmsapi/groups/' + group + '/state'

    payload = {}

    headers = {
      'Authorization': auth_token,
      'Content-Type': 'application/json'
    }

    r = requests.request("GET", url, headers=headers, data=payload)

    return r
```

```
def REST_Set_Group_Level(auth_token, group, level):

    url = 'https://192.168.2.109/api/bmsapi/groups/' + group + '/state'

    payload = json.dumps({
      "level": level
    })

    headers = {
      'Authorization': auth_token,
      'Content-Type': 'application/json'
    }

    r = requests.request("PUT", url, headers=headers, data=payload)

    return r


def REST_Scene_Recall(auth_token, group, scene):

    url = 'https://192.168.2.109/api/bmsapi/groups/' + group + '/state'

    payload = json.dumps({
      "scene": scene
    })
    headers = {
      'Authorization': auth_token,
      'Content-Type': 'application/json'
    }

    r = requests.request("PATCH", url, headers=headers, data=payload)

    return r


#------------------------------------ Main ------------------------------------#


def main():

    print("Set group 0 at 50% level, read the level back and check it")

    desired_level = 50
    group = 0

    r = REST_Login()

    auth_token = r.json()['authHeader']

    # Set level
    REST_Set_Group_Level(auth_token, group, desired_level)

    # Get level
    r = REST_Get_Group_State(auth_token, group)

    level_set = r.json()['level']

    # Check level
    if level_set == desired_level:

        print('Level successfully set at 50%')

    return


if __name__ == "__main__":
    main()

 #------------------------------------------------------------------------------#
```