

DALI-2 IoT

Manual

API Documentation

Application Programming
Interface Description
of the DALI-2 IoT Module

Art.Nr. 89453886
GTIN: 9010342013607



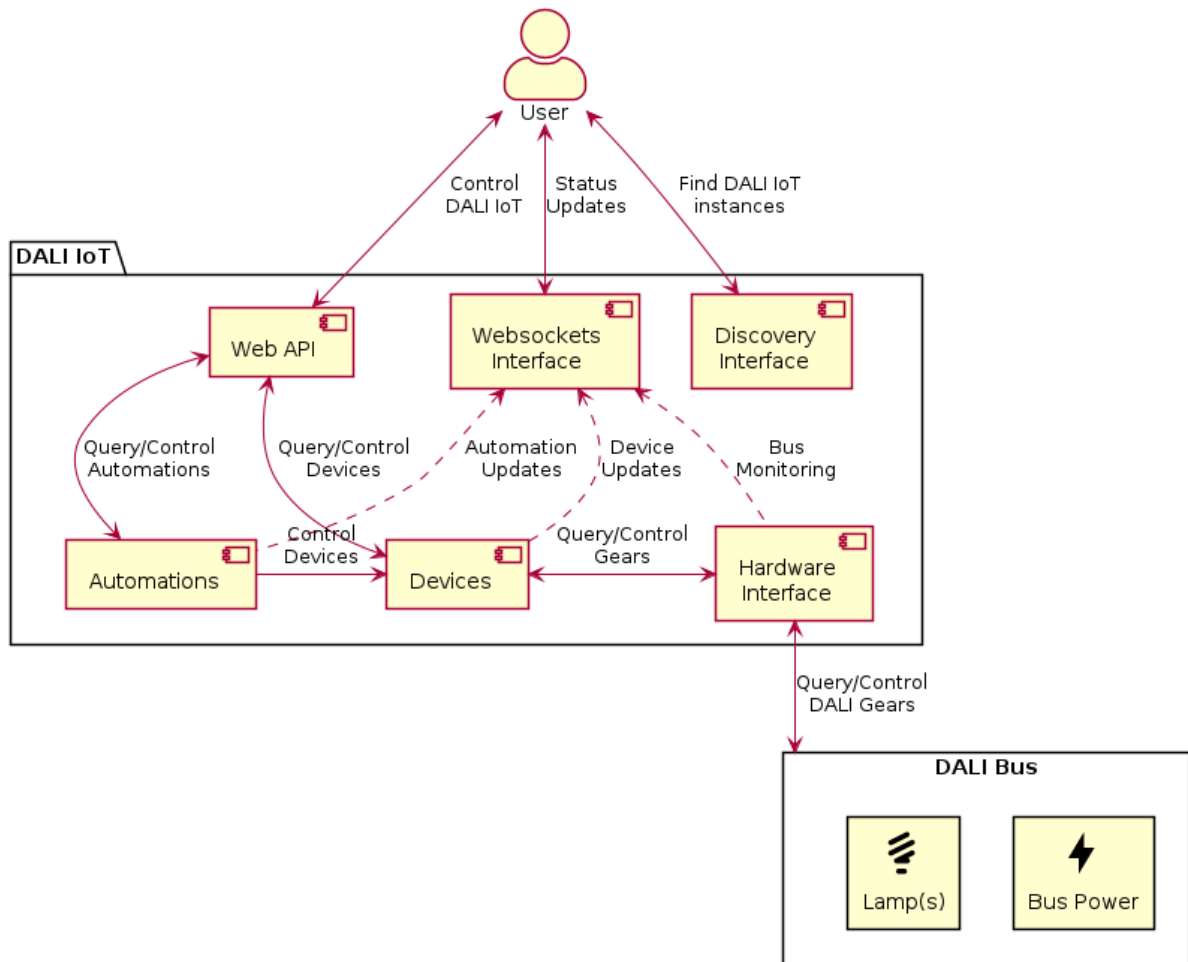


Table of Contents

1 Determining the IP Address.....	5
1.1 Direct Connection to a Computer.....	5
1.2 Determining the DALI-2 IoT Devices in a Network.....	5
2 Opening the DALI-2 IoT API Documentation.....	6
2.1 Categories and Endpoints.....	6
2.2 List of Schemata.....	9
2.3 Direct Testing of Endpoints via the API Documentation.....	10
3 Controlling DALI Devices.....	12
3.1 The ControlData-Object.....	12
3.1.1 Switching Devices On or Off.....	12
3.1.2 Dimming Devices.....	12
3.1.3 Recalling the Last Active Level.....	12
3.1.4 Recalling a Scene.....	12
3.1.5 Storing a Scene Level.....	12
3.1.6 Controlling Colour with Red, Green and Blue (RGB) Values.....	13
3.1.7 Controlling Colour with White, Amber and Free Colour (WAF) Values.....	13
3.1.8 Controlling Colour Temperatures.....	13
3.1.9 Controlling Colour with Colour Coordinates (XY).....	13
3.2 Targets: the DeviceModel Object.....	14
3.3 Controlling All Devices on the DALI Bus.....	14
3.4 Controlling Individual Devices.....	14
3.4.1 Device Scan.....	15
3.4.2 Requesting Registered Devices.....	16
3.4.3 Sending Control Commands to Individual Devices.....	16
3.5 Configuring and Controlling Groups.....	16
3.5.1 Assigning Groups.....	17
3.5.2 Sending Control Commands to Groups.....	17
3.6 Configuring and Controlling Zones.....	17
3.6.1 Adding Zones.....	18
3.6.2 Querying Zones.....	18
3.6.3 Updating Zones.....	19
3.6.4 Sending Control Commands to Zones.....	19
4 Automations.....	19
4.1 Sequences.....	19
4.1.1 Adding Sequences.....	20
4.1.2 Querying Sequences.....	21
4.1.3 Starting and Stopping Sequences.....	21
4.1.4 Updating Sequences.....	21
4.2 Time and Weekday Guided Commands (Schedules).....	22
4.2.1 Adding Schedules.....	23
4.2.2 Querying Schedules.....	23
4.2.3 Updating Schedules.....	24
4.3 Circadian Daylight Progressions.....	24
4.3.1 Adding Circadian Daylight Progressions.....	24
4.3.2 Querying Circadian Daylight Progressions.....	25
4.3.3 Updating Circadian Daylight Progressions.....	26
4.4 Trigger Actions.....	26
4.4.1 Adding Trigger Actions.....	27

4.4.2 Querying Trigger Actions.....	27
4.4.3 Updating Trigger Actions.....	28
5 General Settings of the DALI-2 IoT.....	28
5.1 General Information of the DALI-2 IoT.....	28
5.2 Time zones and time.....	29
5.3 Location.....	29
5.4 Network settings.....	30
5.5 Email.....	30
6 Websocket Interface.....	31
6.1 General Communication.....	31
6.1.1 Greeting message (info).....	32
6.1.2 Dismissing Event Types (filtering).....	33
6.2 Direct DALI access.....	33
6.2.1 DALI Bus Status Events (daliStatus).....	33
6.2.2 DALI Bus Monitor Events (daliMonitor).....	34
6.2.3 Sending DALI commands (daliFrame and daliAnswer).....	35
6.3 Events.....	36
6.3.1 DALI Bus Scan Progress Events (scanProgress).....	36
6.3.2 Device Events (devices and devicesDeleted).....	37
6.3.3 Zone Events (zones and zonesDeleted).....	39
6.3.4 Sequence Events (sequences and sequencesDeleted).....	40
6.3.5 Scheduler Events (schedulers and schedulersDeleted).....	41
6.3.6 Circadian Events (circadians and circadiansDeleted).....	42
6.3.7 Trigger-Actions (triggerActions and triggerActionsDeleted).....	43
6.3.8 Changes of System Time Settings (datetime).....	45
6.3.9 Display of Status Message Events (messageFlash).....	45
6.3.10 Events for Testing Connections (ping).....	45
7 Document History.....	46
8 Bibliography.....	46

1 Determining the IP Address

To make use of the API documentation, it is necessary to know the IP address of the DALI-2 IoT device. The determination of the IP address depends on whether the DALI-2 IoT is directly connected to a personal computer (section 1.1) or to a network (section 1.2).

1.1 Direct Connection to a Computer

A newly delivered DALI-2 IoT is configured to automatically obtain an IP address using the DHCP protocol. When the DALI-2 IoT is unable to reach a DHCP server, it instead falls back to the static IP address 169.254.0.1, and the subnet mask 255.255.0.0. This address can be used when the DALI-2 IoT has a direct Ethernet connection to a computer.

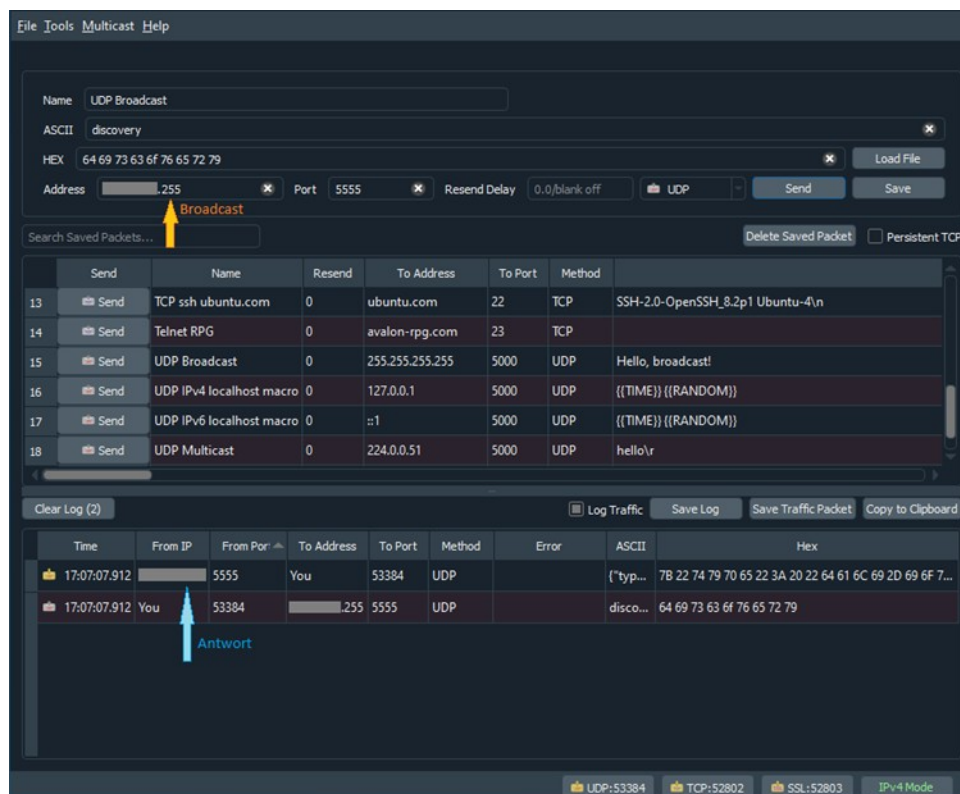
1.2 Determining the DALI-2 IoT Devices in a Network

The IP address can also be determined by means of a “discovery” protocol that is provided by the DALI-2 IoT. The DALI-2 IoT executes a service that listens to UDP packets on port 5555, and reacts to datagrams containing discovery by sending back a datagram with `{"type": "dali-2-iot"}`. Therefore, it is possible to identify all running DALI-2 IoT instances in a network, either by sending a single UDP broadcast packet or individual packets to all addresses within the network.

Beginning with version 1.2, the IoT answers contain an additional user-defined name (refer to section 5.1).

```
{  
  "type": "dali-2-iot",  
  "name": "user defined name"  
}
```

A simple tool to send discovery broadcasts is “Packet Sender”.



To find DALI-2 IoT instances in a network with Packet Sender, it is necessary to set the “ASCII”-input field to discovery, the address to either the broadcast address of the network (which ends in .255) or a single address that should be queried, and the port to 5555.

With these settings a click on the “Send” button will send out a UDP packet in the network. Answers to this packet will appear in the log below the packet templates. Here, the IP addresses of DALI-2 IoT instances can be found.

2 Opening the DALI-2 IoT API Documentation

The API documentation (“docs”-page) of the DALI-2 IoT can be requested with a web browser, by entering `http://<IP_ADDRESS>/docs`, where `<IP_ADDRESS>` should be replaced by the IP address of the device. This page lists API endpoints, sorted by their categories (`control`, `devices`, `dali`, etc.). Each category contains so called endpoints, that allow interactions with the DALI-2 IoT system. An endpoint consists of a request method (`GET`, `POST`, `PUT` or `DELETE`) and an endpoint name (e.g. `/broadcast/control`). The same endpoint name can be used for several endpoints with different request methods and functionality.

Dali IoT 0.5.1 OAS3

</openapi.json>

This API documentation is **work in progress**. In addition to the REST API, there is also a **websocket API** which provides asynchronous events for status updates etc. Documentation for this websocket API will be available soon.

Endpoints to control devices

control ^

- **switchable**: Turns the device on or off
- **dimnable**: Dims the device to a given percentage
- **gotoLastActive**: Turns the device on to it's last active level
- **scene**: Set the device to a given scene
- **saveToScene**: Store the current configuration to scene
- **colorRGB**: Set the device to a given RGB value from 0.0-1.0
- **colorKelvin**: Set the device to a given color temperature in Kelvin
- **colorXY**: Set the device to a given XY color coordinate

POST /device/{_id}/control Control Device

POST /group/{_id}/control Control Group

POST /broadcast/control Control Broadcast

devices Endpoints for information about devices ^

GET /devices Get Devices

DELETE /devices Delete Devices

2.1 Categories and Endpoints

Upon opening the API documentation, all endpoint descriptions are collapsed. Each endpoint can be opened with a click on it, to reveal more details about it: a short description of the endpoint, the parameters that must be transferred and its return value. The endpoint `GET /info` is explained below as an example.

GET /info Info

Load and return the device information.

Parameters Try it out

No parameters

Responses

Code	Description	Links
200	Successful Response	No links

Media type:

Controls Accept header.

Example Value | Schema

```
{
  "name": "dali-iot",
  "version": "vU.V.W/X.Y.Z",
  "tier": "basic",
  "errors": {
    "errorCode": "details"
  }
}
```

A request to this endpoint does not need any input parameters, and the API returns an `InfoModel` as response. An example value of a successful response is displayed in the documentation of the endpoint. Among other things, `GET /info` returns the name of the device and its software and firmware versions. More details about the structure of the return value can be obtained by opening the schema through the “Schema” button, which is located above the example value. This reveals the model's data items, with their according data types and short descriptions.

Example Value | Schema

```
InfoModel {
  description: Model for querying general information about the device.

  name: string
  title: Name
  default: dali-iot
  The user definable device name.

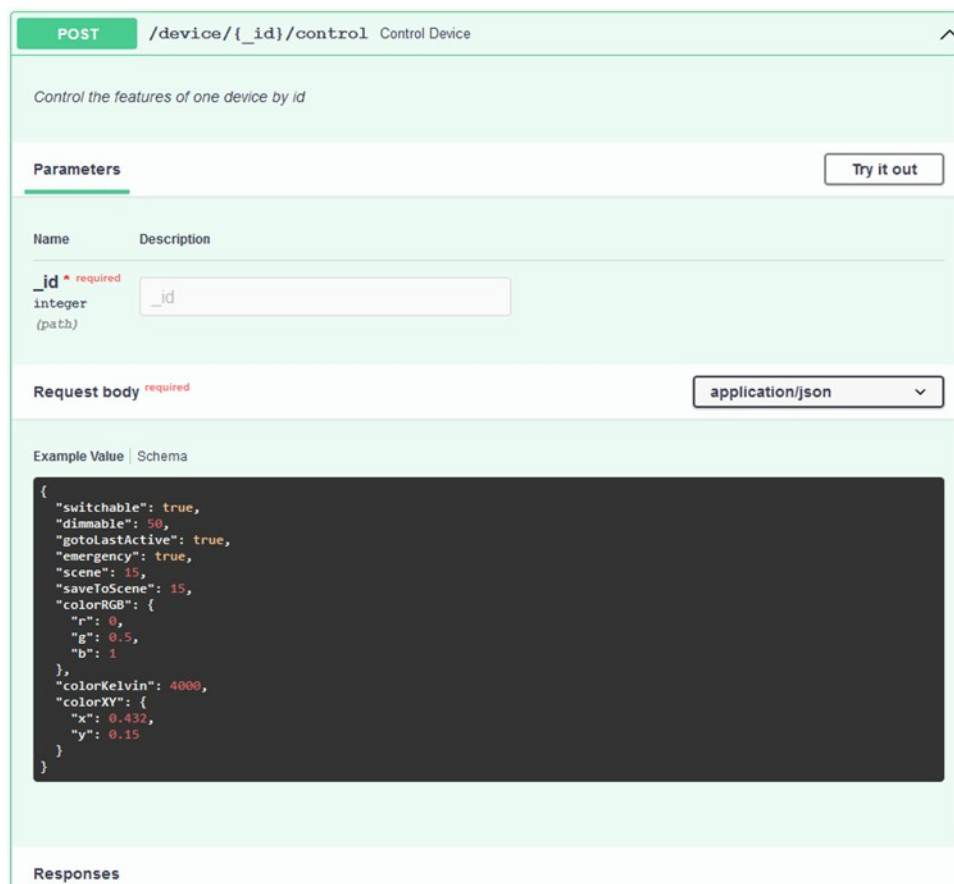
  version: string
  title: Version
  default: vU.V.W/X.Y.Z
  The version string consisting of application version/firmware version.

  tier: string
  title: Tier
  default: basic
  The tier of the device ∈ [basic, plus]. Plus devices support additional features, such as automations and addressing.

  errors: Errors > {...}
}
```

Some endpoints require additional input parameters to properly process a request. For example, the endpoint `POST /device/{_id}/control`, which allows to control individual devices, requires an identifying number of the device and a `ControlData` object. The identifying number determines which device is addressed, whereas `ControlData` defines the desired state of the device. An example value for the `ControlData` object is displayed in the documentation of the endpoint.

In the endpoint `POST /device/{_id}/control`, the identifying number is part of the endpoint address, whereas the `ControlData` object is in the request body. In the API documentation, the identifying number of the device can be entered through a separate text input. However, it should be noted that this number changes the endpoint of the request, which is indicated by the curly braces in the endpoint name. For example, requests to control a device with an identifying number "id": 1 are actually sent to the `POST /device/1/control`.



POST /device/{_id}/control Control Device

Control the features of one device by id

Parameters Try it out

Name	Description
_id * required integer (path)	<input type="text" value="_id"/>

Request body required application/json

Example Value | Schema

```
{
  "switchable": true,
  "dimmable": 50,
  "gotoLastActive": true,
  "emergency": true,
  "scene": 15,
  "saveToScene": 15,
  "colorRGB": {
    "r": 0,
    "g": 0.5,
    "b": 1
  },
  "colorKelvin": 4000,
  "colorXY": {
    "x": 0.432,
    "y": 0.15
  }
}
```

Responses

The schema of the data type can be opened in the same way as schemas for return types, by clicking on the "Schema" button above the example, next to the button "Example Value".


```

Example Value | Schema

ControlData ▾ {
  switchable          boolean
                      title: Switchable
                      example: true
  dimmable            number
                      title: Dimmable
                      example: 50
  gotoLastActive     boolean
                      title: Gotolastactive
                      example: true
  emergency           boolean
                      title: Emergency
                      example: true
  scene              integer
                      title: Scene
                      example: 15
  saveToScene        integer
                      title: Savetoscene
                      example: 15
  colorRGB           Colorrgb > {...}
                      example: OrderedMap { "r": 0, "g": 0.5, "b": 1 }
  colorKelvin        number
                      title: Colorkelvin
                      example: 4000
  colorXY            Colorxy > {...}
                      example: OrderedMap { "x": 0.432, "y": 0.15 }
}
    
```

2.2 List of Schemata

A list of schemata that are transmitted by endpoints is displayed in the API documentation, after the categories and endpoints.



Individual schemas are initially collapsed and can be opened with a click, to reveal their details.

```

InfoModel {
  description:
    Model for querying general information about the device.

  name
    string
    title: Name
    default: dali-iot
    The user definable device name.

  version
    string
    title: Version
    default: vU.V.W/X.Y.Z
    The version string consisting of application version/firmware version.

  tier
    string
    title: Tier
    default: basic
    The tier of the device = [basic, plus]. Plus devices support additional features, such
    as automations and addressing.

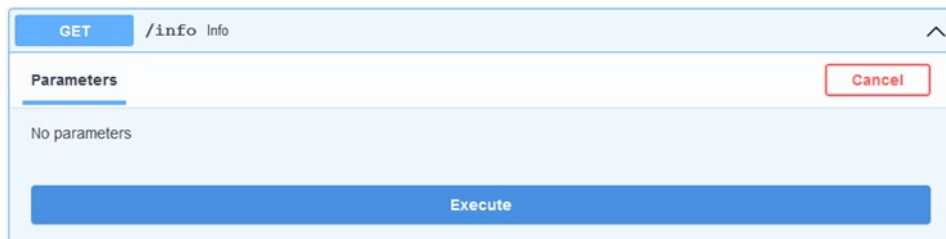
  errors
    Errors {
      description:
        A dictionary mapping error codes to their details.

      < * >:
        string
    }
}

```

2.3 Direct Testing of Endpoints via the API Documentation

API endpoints can be directly tested in the documentation, using the “Try it out” Button. Clicking this button opens additional input fields (the available properties depend on the endpoints), and an “Execute” button. No additional input fields appear for the [GET /info](#) endpoint, since it requires no additional parameters.



When the “Execute” button is clicked, a browser will send a request to the corresponding endpoint. Afterwards, an example of how to send the request with the program “curl”, the complete address of the request, and the answer to the request are displayed. Additionally, a “Clear” button to remove the request is added next to “Execute”.

Execute
Clear

Responses

Curl

```
curl -X 'GET' \
'http://192.168.0.62/info' \
-H 'accept: application/json'
```

Request URL

```
http://192.168.0.62/info
```

Server response

Code	Details
200	<p>Response body</p> <pre>{ "name": "dali-iot", "version": "v0.5.1/0.0.184", "tier": "basic", "errors": { "errorCode": "details" } }</pre> <p style="text-align: right;">Download</p> <p>Response headers</p> <pre>content-length: 94 content-type: application/json date: Wed, 07 Jul 2021 14:39:56 GMT server: uvicorn</pre>

Other endpoints may require additional input to properly process a request, in which case additional input fields are opened by “Try it out”. For example, the endpoint `POST /device/{_id}/control` requires an identifying number to specify the device, and a `ControlData` object, to control it. Mandatory input fields are labelled with “required”, fields that are not mandatory can be left empty.

Cancel

Parameters

Name	Description
<p>_id * required</p> <p>integer</p> <p>(path)</p>	<input style="width: 100%; height: 20px;" type="text" value="_id"/>

Request body required

application/json

```
{
  "switchable": true,
  "dimnable": 50,
  "gotolastActive": true,
  "emergency": true,
  "scene": 15,
  "saveToScene": 15,
  "colorRGB": {
    "r": 0,
    "g": 0.5,
    "b": 1
  },
  "colorKelvin": 4000,
  "colorXY": {
    "x": 0.432,
    "y": 0.15
  }
}
```

Execute

3 Controlling DALI Devices

3.1 The ControlData-Object

Devices are controlled by specifying their features, such as for example “dimmable” for devices that can be dimmed. To control a device, it is necessary to declare their features and the corresponding parameter values through a ControlData object. The ControlData schema is a mapping that matches the names of features to their desired values. Individual features are optional; therefore, it is sufficient to only state those features that should be controlled in a request. It is possible to control several features at once, by separating them with commas.

3.1.1 Switching Devices On or Off

The “switchable” feature can be used to switch devices on or off. It requires a boolean value to determine whether to switch the device on (`true`) or off (`false`). The “Request body” for switching devices consists of the name of the feature and its value.

```
{
  "switchable": true
}
```

3.1.2 Dimming Devices

The “dimmable” feature can be used to dim devices. It requires a dim percentage from 0 to 100. If a “dimmable” value of 0 is entered, the device is switched off. For any other value the device is switched on and set to the desired light level in percent.

```
{
  "dimmable": 50
}
```

3.1.3 Recalling the Last Active Level

The “gotoLastActive” feature can be used to recall the last active level. It requires a `true` value.

```
{
  "gotoLastActive": true
}
```

3.1.4 Recalling a Scene

Scenes can be recalled with the “scene” feature. This requires the respective scene number (0 to 15) of the scene that should be recalled.

Scenes are configured and stored in DALI devices. Many DALI devices are delivered with no initial scene values and must therefore be configured prior to recalling the respective scene (section 3.1.5).

```
{
  "scene": 15
}
```

3.1.5 Storing a Scene Level

The “saveToScene” feature can be used to store the current level as a scene value for a scene. This requires a scene number (0 to 15) of the scene for which the current level should be stored.

```
{
  "saveToScene": 15
}
```

3.1.6 Controlling Colour with Red, Green and Blue (RGB) Values

The "colorRGB" feature can be used to control the colour of a red-, green- and blue- (RGB / RGBW) light. This requires a mapping of the keys "r", "g" and "b" to relative colour values in the range from 0 to 1.

```
{
  "colorRGB": {
    "r": 0,
    "g": 0.5,
    "b": 1
  }
}
```

3.1.7 Controlling Colour with White, Amber and Free Colour (WAF) Values

The "colorWAF" feature can be used to control the colour of a white-, amber- and free colour (RGBWAF) light. This requires a mapping of the keys "w", "a" and "f" to relative colour values in the range from 0 to 1.

```
{
  "colorWAF": {
    "w": 0,
    "a": 0.5,
    "f": 1
  }
}
```

3.1.8 Controlling Colour Temperatures

The "colorKelvin" feature can be used to control the colour of a colour temperature (CW-WW) light. This requires a colour temperature in Kelvin (typical range of Lunatone CW-WW dimmers: 100 to 20000 K).

```
{
  "colorKelvin": 4000
}
```

3.1.9 Controlling Colour with Colour Coordinates (XY)

The "colorXY" feature can be used to control the colour of a light with colour coordinates in the colour space. This requires a mapping of the keys "x" and "y" to colour coordinates in the range of 0 to 1.

```
{
  "colorXY": {
    "x": 0.432,
    "y": 0.15
  }
}
```

3.2 Targets: the DeviceModel Object

Zones (section Error: Reference source not found) and automations (section 4) share that they refer to target devices (DeviceModel), the features of which are controlled together. Targets are a list of devices, groups, zones and broadcasts. Each entry in the list is a DeviceModel object that consists of a target type (type) and an optional identifying number (id). Broadcasts (broadcast), groups (group) and devices (device) are supported for target types. The identifying number is only relevant for groups, devices and zones; it can be omitted for broadcasts.

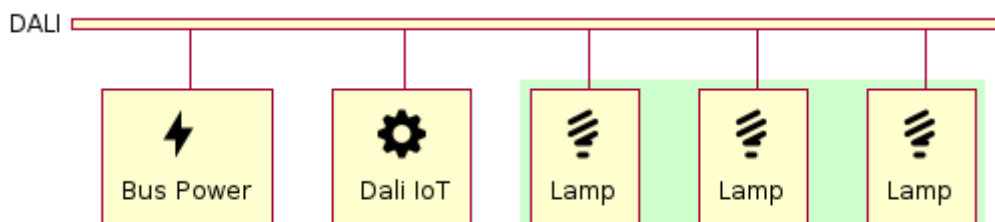
```
{  
  "type": "broadcast"  
}
```

```
{  
  "type": "device",  
  "id": 1  
}
```

```
{  
  "type": "group",  
  "id": 1  
}
```

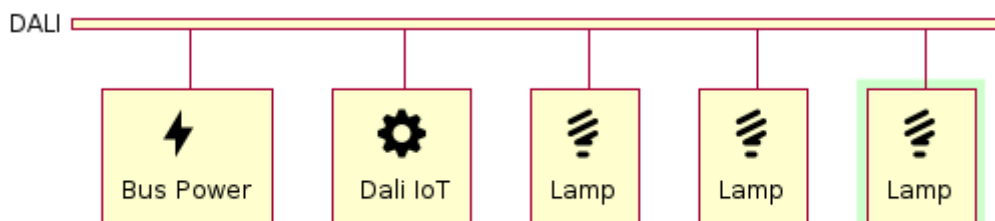
```
{  
  "type": "zone",  
  "id": 1  
}
```

3.3 Controlling All Devices on the DALI Bus

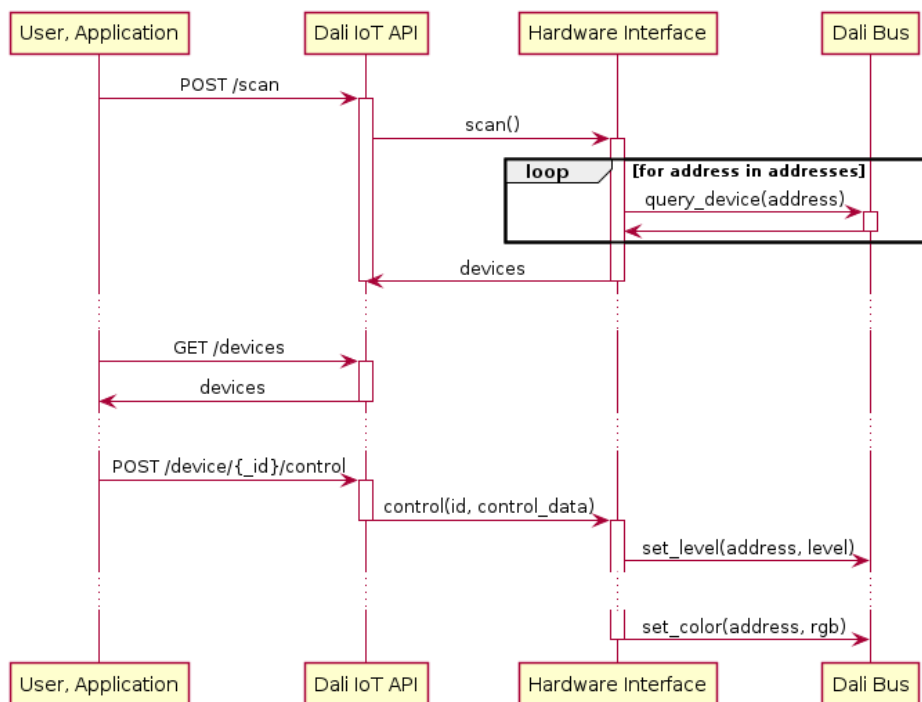


No additional set-up is required to control all devices on a DALI bus. The devices can be controlled directly with the endpoint `POST /broadcast/control`, which needs a `ControlData` object to process the request.

3.4 Controlling Individual Devices



To control individual devices on a DALI bus, it is initially required to register and save the devices on the bus. Each registered device receives an identifying number, with which it is differentiated from other devices during requests and responses. After carrying out a device scan once, it is possible to query the registered devices to obtain their state and identifying numbers. These numbers are used by the DALI-2 IoT to control individual devices and differ from the DALI addresses of the devices.



3.4.1 Device Scan

A so called device scan can be used to address, register and save devices on a DALI bus. The endpoint `POST /dali/scan`, which requires two inputs: `newInstallation` and `noAddressing`, can be used to start a new scan.

```
{
  "newInstallation": false,
  "noAddressing": false
}
```

`"newInstallation": true` only needs to be used for a reset and new set up of the installation: the already registered devices are deleted and subsequently re-registered. `"new Installation": false` is commonly used; here, new devices are added to the already registered ones.

Furthermore it is possible to carry out addressing prior to registering devices, which allocates unique bus addresses to the DALI devices. If this step was already performed through other means (such as a second DALI-2 IoT instance), addressing is not necessary and the DALI-2 IoT only needs to register devices. Addressing can then be skipped by setting the value of `noAddressing` to `true`.

Because the device scan queries all addresses on a DALI bus, it can last for approximately a minute. During this time it is possible to query the progress of the scan with the endpoint `GET /dali/scan`. This endpoint returns a `ScanModel` in the response, which contains a unique identifier of the current scan, its progress (in percent), the number of registered devices and the state of the scan.

```
{
  "id": "e9160f03-0982-4cd7-88ab-00a4755bd17d",
  "progress": 38.671875,
  "found": 1,
  "status": "in progress"
}
```

No second scan can be started while a scan is in progress. It is however possible to cancel a scan by sending a request to the endpoint `POST /dali/scan/cancel`.

3.4.2 Requesting Registered Devices

Registered devices of the DALI-2 IoT can be queried with the `GET /devices` endpoint. This endpoint does not need additional inputs and returns a list of known devices, their identifying numbers, names, types, the states of their features, as well as their scene values and group affiliations. Additionally, the response contains a signature consisting of a timestamp and a counter, to differentiate responses.

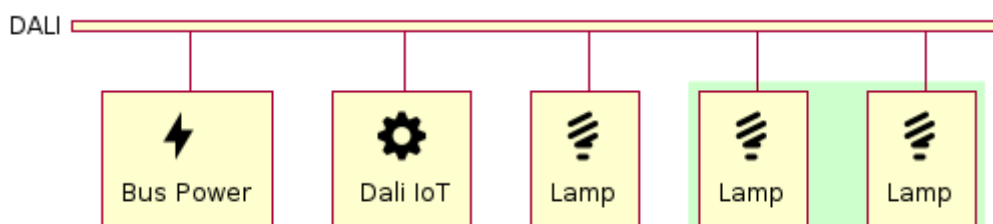
```
{
  "devices": [
    {
      "id": 1,
      "name": "DALI #0",
      "address": 0,
      "line": 0,
      "type": "default",
      "features": {
        "switchable": {
          "status": false
        },
        "dimnable": {
          "status": 0
        },
        "scene": true,
        "colorRGB": {
          "status": {
            "r": 1,
            "g": 1,
            "b": 1
          }
        },
        "colorKelvin": {
          "status": 2700
        },
        "colorXY": {
          "status": {
            "x": 0,
            "y": 0
          }
        },
        "saveToScene": true,
        "gotoLastActive": {}
      },
      "scenes": [],
      "groups": [],
      "daliTypes": [8]
    },
    { "id": 2, ... }, ...
  ],
  "timeSignature": {
    "timestamp": 1625747234.3620,
    "counter": 4
  }
}
```

Note that the devices had no address and line attributes in versions prior to 1.2, which instead used an `info` attribute with a string value. The attribute `daliTypes` was added in version 1.4.

3.4.3 Sending Control Commands to Individual Devices

Devices can be controlled individually with the `POST /device/{_id}/control` endpoint. As was the case with controlling all devices, this requires a `ControlData` object in the “Request body”. An additional required input is the identifying number of the device that should be controlled.

3.5 Configuring and Controlling Groups



Before groups can be controlled, they must first be assigned to the DALI devices. This step can be skipped if the devices were already assigned to groups. After assigning groups to devices, they can be controlled together, with only a single request.

3.5.1 Assigning Groups

The endpoint `PUT /device/{_id}` can be used to assign groups to devices. It requires an identifying number of a device, as well as a `DeviceUpdateModel` consisting of an (optional) name and an (also optional) list of group numbers (0 to 15). For example, to assign the group 0 to a device, the "Request body" must contain the key `groups`, mapped to a list of values that contains the group numbers (0 in this example).

```
{
  "groups": [
    0
  ]
}
```

The return value from the API contains the state of the devices with the amended group affiliations. The response is similar to responses from the `GET /device/{_id}` endpoint - although limited to the updated device - and therefore also contains the states of the device's features and scene values.

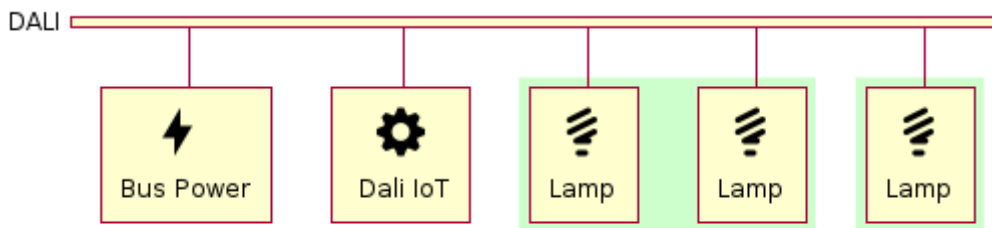
```
{
  "id": 1,
  "name": "DALI #0",
  "address": 0,
  "line": 0,
  "type": "default",
  "features": {
    "switchable": {
      "status": true
    },
    "dimmable": {
      "status": 100
    },
    "scene": true,
    "colorRGB": {
      "status": {
        "r": 1,
        "g": 1,
        "b": 1
      }
    },
    "colorKelvin": {
      "status": 2700
    },
    "colorXY": {
      "status": {
        "x": 0,
        "y": 0
      }
    },
    "saveToScene": true,
    "gotoLastActive": {}
  },
  "scenes": [],
  "groups": [0],
  "daliTypes": [8],
  "timeSignature": {
    "timestamp": 1625754320.503,
    "counter": 159
  }
},
}
```

Note that the devices had no address and line attributes in versions prior to 1.2, which instead used an `info` attribute with a string value. The attribute `daliTypes` was added in version 1.4.

3.5.2 Sending Control Commands to Groups

The endpoint `POST /group/{_id}/control` can be used to control groups. As was the case with controlling all devices, this requires a `ControlData` object in the "Request body". An additional required input is the group number of the group that should be controlled.

3.6 Configuring and Controlling Zones



Zones are logical collections of groups, devices and other zones within the DALI-2 IoT, that can be controlled together. Zones must be created and configured before they can be controlled. They are characterised by:

- an identifying number (`id`),
- an optional name (`name`), and
- a list of targets that are controlled together (`targets`).

Targets are defined by the `DeviceModel` (section 3.2).

3.6.1 Adding Zones

The endpoint `POST /zone` can be used to add a new zone. Only the targets of the zone are required in the request. A zone that consists of group 0 and device 1 is displayed as an example.

```
{
  "targets": [
    {
      "type": "group",
      "id": 0
    },
    {
      "type": "device",
      "id": 1
    }
  ]
}
```

3.6.2 Querying Zones

Zones created in the DALI-2 IoT can be queried with the `GET /zones` endpoint. This endpoint does not need additional inputs and returns a list of known zones, their identifying numbers, names and targets. Additionally, the response contains a signature consisting of a timestamp and a counter, to differentiate responses.

```
{
  "zones": [
    {
      "id": 1,
      "name": null,
      "targets": [
        {
          "type": "group",
          "id": 0
        },
        {
          "type": "device",
          "id": 1
        }
      ]
    }
  ],
  "signature": {
    "timestamp": 1611111111,
    "counter": 1
  }
}
```

```
"timeSignature": {
  "timestamp": 1644507239.4869525,
  "counter": 342
}
```

3.6.3 Updating Zones

The endpoint `PUT /zone/{_id}` can be used to update a zone. It requires an identifying number of a device, as well as a zone's attributes that should be updated. Only values that are passed in the request body are updated. For example, to change the name of a zone, the request body must contain the name key.

```
{
  "name": "Group 0 and device 1"
}
```

The return value from the API contains the state of the zone with the updated values. The response is equivalent to responses from the `GET /zone/{_id}` endpoint.

```
{
  "id": 1,
  "name": "Group 0 and device 1",
  "targets": [
    {
      "type": "group",
      "id": 0
    },
    {
      "type": "device",
      "id": 1
    }
  ],
  "timeSignature": {
    "timestamp": 1644507239.4869525,
    "counter": 342
  }
}
```

3.6.4 Sending Control Commands to Zones

The endpoint `POST /zone/{_id}/control` can be used to control zones. As was the case with controlling all devices, this requires a `ControlData` object in the "Request body". An additional required input is the identifying number of the zone that should be controlled.

4 Automations

Automations can be utilised to edit, store and execute complex sequences of control commands. Several different automations are available: sequences, time and weekday guided commands (schedules) and circadian daylight progressions.

4.1 Sequences

Sequences are automatic playbacks of a list of commands, and are characterised by

- an identifying number (`id`),

- a number of "sequence steps" (steps),
- a parameter that activates/deactivates the sequence (enabled),
- an optional name (name),
- an optional parameter for endless sequences (loop), and
- an optional parameter for playing a sequence for a finite number of times (repeat).

Each step in a sequence consists of a parameter that activates/deactivates the step (enabled), a type (type), an action to carry out (data) and an optional delay in seconds (delay) prior to the execution of the step. Currently only the value features is supported for step types. The action of a step consists of the action targets (targets) and the features that should be controlled (features). The enabled parameter can be omitted because it defaults to true.

An example of a single step that turns off all devices after a two second delay is displayed below.

```
{
  "type": "features",
  "data": {
    "targets": [
      {
        "type": "broadcast"
      }
    ],
    "features": {
      "switchable": false
    }
  },
  "delay": 2
}
```

4.1.1 Adding Sequences

The endpoint [POST /automations/sequence](#) can be used to add a new sequence. It requires the entire sequence with all of its steps, where enabled fields default to true. An endless sequence of on- and off-switches that are sent as broadcasts is displayed as an example. Between each of the switch actions is a delay of 2 s.

```
{
  "name": "blink_slowly",
  "loop": true,
  "steps": [
    {
      "type": "features",
      "data": {
        "targets": [
          {
            "type": "broadcast"
          }
        ],
        "features": {
          "switchable": true
        }
      },
      "delay": 2
    },
    →
```

```
←
    {
      "type": "features",
      "data": {
        "targets": [
          {
            "type": "broadcast"
          }
        ],
        "features": {
          "switchable": false
        }
      },
      "delay": 2
    },
  ],
}
```

4.1.2 Querying Sequences

All stored sequences can be queried with the `GET /automations/sequences` endpoint. The response to this endpoint transmits an additional attribute `active`, which indicated whether the sequence is currently being executed (`true`) or not (`false`).

```
{
  "sequences": [
    {
      "name": "blink_slowly",
      "enabled": true,
      "loop": true,
      "repeat": 0,
      "steps": [
        {
          "enabled": true,
          "type": "features",
          "data": {
            "targets": [
              {
                "type": "broadcast"
              }
            ],
            "features": {
              "switchable": true
            }
          }
        },
        {
          "delay": 2
        }
      ],
      "id": 1,
      "active": false
    }
  ]
}
```

Individual sequences can be queried with the `GET /automations/sequence/{_id}` endpoint. This requires an identifying number of the sequence as part of the endpoint address.

4.1.3 Starting and Stopping Sequences

Sequences must be started manually, by sending a request to the `POST /automations/sequence/{_id}/start` endpoint. This requires an identifying number of the sequence as part of the endpoint address. No additional data is necessary for the request body. The endpoint `POST /automations/sequence/{_id}/stop` can be used to stop a running sequence.

4.1.4 Updating Sequences

The endpoint `PUT /automations/sequence/{_id}` can be used to update an existing sequence. This requires an identifying number of the sequence as part of the endpoint address. The structure of the "Request body" is identical to the structure for adding new sequences (section 4.1.1), however all parameters are optional during updates; only those parameters that should be changed are required for an update.

If, for example, an endless sequence should be changed to only repeat 5 times, then it is sufficient to update the `loop` and `repeat` attributes of the sequence.

```
{
  "loop": false,
  "repeat": 5
}
```

Important: the steps of a sequence are regarded as a single parameter. Therefore, all steps must be sent in full even when only one of the steps is changing.

4.2 Time and Weekday Guided Commands (*Schedules*)

Schedules are time and weekday guided commands, and are characterised by:

- an identifying number (`id`),
- an action that will be executed (`action`),
- a list of targets that the action is applied to (`targets`),
- a time at which the action is scheduled (`recallTime`),
- a mode that defines how to interpret the defined time (`recallMode`),
- a parameter that activates/deactivates the schedule (`enabled`),
- an optional name (`name`),
- optional active periods (`activePeriod`), active months (`activeMonths`), active weekdays (`activeWeekdays`) or days in a month (`activeDays`).

The recall time (`recallTime`) and the recall mode (`recallMode`) of a schedule determine the time at which the time-controlled commands are to be used. The recall time is a time specification in hours, minutes and seconds; and is evaluated differently depending on the recall mode.

```
"recallTime": {  
  "hour": 6,  
  "minute": 30,  
  "second": 0  
}
```

The following recall modes are supported:

- `timeOfDay`: the schedule action is applied at time as time of the day (hh:mm:ss)
- `beforeSunrise`: the schedule action is applied with the recall time as an offset before sunrise
- `afterSunrise`: the schedule action is applied with the recall time as an offset after sunrise
- `beforeSunset`: the schedule action is applied with the recall time as an offset before sunset
- `afterSunset`: the schedule action is applied with the recall time as an offset after sunset

Important: Sunrise and sunset are calculated based on the configured location. In addition, all recall modes depend on the date time settings of the DALI-2 IoT. The time and location of the device need to be set for correct operation (section 5.2 and 5.3).

The parameters for active periods, months, days of the week and days of the month can be used to restrict the days on which the time-controlled commands are carried out. An active period consists of a start and end month and day. The schedule is active on all days between the start and end. On days that are outside the specified period the schedule is not carried out. For active months (and days of the week), the month names (or names of the days of the week) are assigned boolean values that indicate whether a month (or day of the week) is active (`true`) or inactive (`false`). Active days within a month can also be specified using the number of the day (from 1 to 31). On days that are not specified as active the schedule is not carried out.

```
"activePeriod": {  
  "startDay": 24,  
  "startMonth": 5,  
  "endDay": 24,  
  "endMonth": 9  
}
```

```
"activeMonths": {  
  "january": false,  
  "february": false,  
  "march": true,  
  ...  
}
```

```
"activeWeekDays": {
  "monday": false,
  "tuesday": false,
  "wednesday": true,
  ...
}
```

```
"activeDays": {
  "days": [
    1,
    2, ...
  ]
}
```

4.2.1 Adding Schedules

The endpoint [POST /automations/scheduler](#) can be used to add a new schedule. It requires the complete schedule, with its time of day (recallTime), the action targets (targets), the scheduling mode (recallMode) and the scheduled action (action), where enabled fields default to true.

A schedule that switches on group 0 on each weekday is displayed as an example. The parameters activePeriod, activeMonths, activeWeekdays and activeDays are optional and can therefore be omitted. Weekend days are deactivated in the example; all other days default to be active and can therefore be omitted too.

```
{
  "name": "lights-on",
  "targets": [
    {
      "type": "group",
      "id": 0
    }
  ],
  "activeWeekdays": {
    "saturday": false,
    "sunday": false
  },
  "recallMode": "timeOfDay",
}
```

```
←
  "recallTime": {
    "hour": 6
    "minute": 30
  },
  "action": {
    "type": "features",
    "data": {
      "features": {
        "switchable": true
      }
    }
  }
}
```

4.2.2 Querying Schedules

All stored schedules can be queried with a request of the [GET /automations/schedules](#) endpoint.

```
{
  "schedulers": [
    {
      "name": "lights-on",
      "targets": [
        {
          "type": "group",
          "id": 0
        }
      ],
      "enabled": true,
      "activePeriod": {},
      "activeMonths": {},
      "activeWeekdays": {
        "monday": true,
        "tuesday": true,
        "wednesday": true,
        "thursday": true,
        "friday": true,
        "saturday": false,

```

```
←
    "sunday": false
  },
  "activeDays": {},
  "recallMode": "timeOfDay",
  "recallTime": {
    "hour": 6,
    "minute": 30,
    "second": null
  },
  "action": {
    "type": "features",
    "data": {
      "features": {
        "switchable": true
      }
    }
  },
  "id": 1
}
]
```

Individual schedules can be queried with the `GET /automations/scheduler/{_id}` endpoint. This requires an identifying number of the schedule as part of the endpoint address.

4.2.3 Updating Schedules

The endpoint `PUT /automations/scheduler/{_id}` can be used to update an existing schedule. This requires an identifying number of the schedule as part of the address. The structure of the "Request body" is identical to the structure for adding a new schedule (section 4.2.1), however all parameters are optional during updates; only those parameters that should be changed are required for an update.

If, for example, a schedule should be changed to only activate on Saturdays, then it is enough to update the "activeWeekdays" attribute.

```
{
  "activeWeekdays": {
    "saturday": true
  }
}
```

4.3 Circadian Daylight Progressions

Circadian daylight progressions simulate the progression of natural daylight by continuously updating colour temperatures and brightnesses, and are characterised by

- an identifying number (`id`),
- two daylight curves for the longest (`longest`) and shortest (`shortest`) day of the year,
- a list of targets to which the progression is applied (`targets`),
- a parameter that activates/deactivates the progression (`enabled`), and
- an optional name (`name`).

Objects that describe a daylight curve consist of a day (`day`), a month (`month`) and a list of steps that make up the curve itself. Each of the steps in the curve contains an hour (`hour`), a dim percentage (`dimable`) and an optional colour temperature (`colorKelvin`). Dim percentages and colour values are interpolated between minutes of the curves and between the longest and shortest day curves over a year.

```
{
  "hour": 12,
  "colorKelvin": 5800,
  "dimmable": 20
}
```

Important: The system time must be set up correctly for circadian progressions to work (refer to section 5.2).

4.3.1 Adding Circadian Daylight Progressions

The endpoint `POST /automations/circadian` can be used to add a new circadian automation. It requires the complete progression, with its targets (`targets`) and the longest (`longest`) and shortest (`shortest`) day in the year, where `enabled` fields default to `true`.


```

{
  "targets": [
    {
      "type": "device",
      "id": 2
    }
  ],
  "longest": {
    "day": 21,
    "month": 6,
    "steps": [
      {
        "hour": 0,
        "colorKelvin": 2700
      },
      ...
      {
        "hour": 23,
        "colorKelvin": 2700
      }
    ]
  }
}

```

```

},
"shortest": {
  "day": 21,
  "month": 12,
  "steps": [
    {
      "hour": 0,
      "colorKelvin": 2700
    },
    {
      "hour": 1,
      "colorKelvin": 2700
    },
    ...
    {
      "hour": 23,
      "colorKelvin": 2700
    }
  ]
}
}

```

Values in the code examples for circadian progressions were abbreviated for an improved legibility. A complete example progression for controlling colour temperatures is displayed in table 1.

Table 1: An example curve for colour temperature progressions.

Hour	Temperature [K]	Hour	Temperature [K]	Hour	Temperature [K]
0	2700	8	4301	16	4101
1	2700	9	4767	17	3412
2	2700	10	5318	18	2700
3	2700	11	5685	19	2700
4	2700	12	5800	20	2700
5	2700	13	5685	21	2700
6	2700	14	5218	22	2700
7	3412	15	4767	23	2700

4.3.2 Querying Circadian Daylight Progressions

All stored circadian progressions can be queried with a request of the [GET /automations/circadians](#) endpoint.

```

{
  "circadians": [
    {
      "name": "New",
      "targets": [
        {
          "type": "device",
          "id": 2
        }
      ],
      "longest": {
        "day": 21,

```

```

"shortest": {
  "day": 21,
  "month": 12,
  "steps": [
    {
      "hour": 0,
      "colorKelvin": 2700
    },
    {
      "hour": 1,
      "colorKelvin": 2700

```

```

"month": 6,
"steps": [
  {
    "hour": 0,
    "colorKelvin": 2700
  },
  {
    "hour": 1, ...
  },
  ...
]
},
→
},
{
  "hour": 2,
  "colorKelvin": 2700
},
...
]
},
"enabled": true,
"id": 1
}
]
}

```

Individual circadian progressions can be queried with the [GET /automations/circadian/{_id}](#) endpoint. This requires an identifying number of the circadian progression as part of the endpoint address.

4.3.3 Updating Circadian Daylight Progressions

The endpoint [PUT /automations/circadian/{_id}](#) can be used to update an existing circadian progression. This requires an identifying number of the circadian progression as part of the endpoint address. The structure of the "Request body" is identical to the structure for adding a new circadian progression (section 4.3.1), however all parameters are optional during updates; only those parameters that should be changed are required for an update.

It should be noted that the entire parameter must be specified during an update. If, for example, the date of the shortest day should be changed, then it is required to also send all steps of the shortest curve.

```

"shortest": {
  "day": 22,
  "month": 12,
  "steps": [
    {
      "hour": 0,
      "colorKelvin": 2700
    },
    ...
  ]
}

```

4.4 Trigger Actions

Trigger actions are automatic forwards of DALI commands, characterised by:

- an identifying number (`id`),
- an optional name (`name`),
- a list of source addresses from which commands are forwarded (`sources`),
- a list of target addresses to which commands are forwarded (`targets`), and
- a parameter that activates/deactivates the automation (`enabled`).

Targets are defined by the `DeviceModel` (section 3.2). Sources are defined by the `TriggerActionSource`, which has a `type`, and optional parameters that depend on the type. The

type of a trigger action source can be a registered device (device), a group (group), an DALI address (d16gear) and a DALI group (d16group). Devices and groups require an additional identifying number (id). DALI addresses and DALI groups require an additional "address" and a "line" parameter. Because the DALI-2 IoT supports one DALI line, the line number must be zero (0).

Note that the d16gear type allows to use addresses that are not occupied by a DALI gear as a source for trigger action automations. Zones and broadcasts are not currently supported as sources.

```
{
  "type": "device",
  "id": 1
}
```

```
{
  "type": "group",
  "id": 1
}
```

```
{
  "type": "d16gear",
  "address": 1,
  "line": 0
}
```

```
{
  "type": "d16group",
  "address": 1,
  "line": 0
}
```

4.4.1 Adding Trigger Actions

The endpoint [POST /automations/triggerAction](#) can be used to add a new automatic forwarding. It requires at least one source and at least one target. An trigger action that forwards DALI commands from device 1 to zone 1 is displayed as an example.

```
{
  "enabled": true,
  "name": "",
  "sources": [
    {
      "type": "device",
      "id": 1
    }
  ],
  "targets": [
    {
      "type": "zone",
      "id": 1
    }
  ]
}
```

4.4.2 Querying Trigger Actions

Trigger actions can be queried with a request of the [GET /automations/triggerActions](#) endpoint.

```
{
  "triggerActions": [
    {
      "id": 1,
      "enabled": true,
      "name": "",
      "sources": [
        {
          "type": "device",
          "id": 1,
          "address": null,
          "line": null
        }
      ],
      "targets": [
```

```
{
  {
    "type": "zone",
    "id": 1
  }
]
```

Individual trigger actions can be queried with the [GET /automations/triggerAction/{_id}](#) endpoint. This requires an identifying number of the trigger action as part of the endpoint address.

4.4.3 Updating Trigger Actions

The endpoint [PUT /automations/triggerAction/{_id}](#) can be used to update an existing trigger action. This requires an identifying number of the trigger action as part of the endpoint address. The structure of the "Request body" is identical to the structure for adding a new trigger action (section 4.3.1), however all parameters are optional during updates; only those parameters that should be changed are required for an update.

```
{
  "name": "new name for the automation"
}
```

5 General Settings of the DALI-2 IoT

5.1 General Information of the DALI-2 IoT

General information about the DALI-2 IoT can be requested with the [GET /info](#) endpoint. It returns

- the user defined name of the DALI-2 IoT (name),
- the software and firmware version of the device (version),
- the unlocked features of the device (tier and emergencyLight),
- error states, such as problems with the DALI bus (errors),
- the descriptor of the hardware interface (descriptor), and
- information about the device itself, such as serial number and article numbers (device).

```
{
  "name": "dali-iot",
  "version": "v1.2.0/1.0.9",
  "tier": "plus",
  "emergencyLight": true,
  "errors": {},
  "descriptor": {
    "lines": 1,
    "bufferSize": 32,
    "tickResolution": 1978,
    "maxYnFrameSize": 32,
    "deviceListSpecifier": true,
    "protocolVersion": "1.0"
  },
  "device": {
    "serial": 1234567890,
    "gtin": 1234567890,
    "pcb": "9a",

```

```
"articleNumber": 1234567890,  
"articleInfo": "",  
"productionYear": 2021,  
"productionWeek": 31  
}
```

The **PUT** `/info` endpoint can be used to set the user-defined name, with the new name as input.

```
{  
  "name": "new name of the device"  
}
```

5.2 Time zones and time

The time zone and time need to be set for correct operation of time-controlled automations. If the DALI-2 IoT is operated in a network with internet access, the time can be updated automatically. For the automatic time settings, the correct time zone has to be set.

The **POST** `/datetime` endpoint can be used to set the time and time zone of the DALI-2 IoT. The Time zone, automatic update of the time via the Internet (`automatic_time`), date (`date`) and time (`time`) can be specified. All input parameters are optional, only the inputs specified will be changed.

```
{  
  "timezone": "Europe/Vienna",  
  "automatic_time": true,  
  "date": "28. October 2021",  
  "time": "15:26"  
}
```

Time zones are specified as a region / location. The end point **GET** `/datetime/timezones` can be used to output a list of known time zones.

```
{  
  "timezones": [  
    "Africa/Abidjan",  
    "Africa/Accra",  
    ...  
  ]  
}
```

If automatic reference of the time is activated, setting date and time can be omitted. Otherwise, both values have to be set. The set time zone and the current time can be queried via the end point **GET** `/datetime`.

5.3 Location

The approximate operating location has to be set for correct operation of time controlled automations with references to sunrise or sunset. If the DALI-2 IoT is operated in a network with internet access, the location can be recognized automatically. In order to automatically detect and configure the location, a query must be sent to the endpoint **POST** `/location/detect`. This end point does not require any additional information and returns the approximate coordinates of the recognized location.

```
{  
  "lat": 48.20849,  
  "lon": 16.37208  
}
```

If the DALI-2 IoT does not have internet access, the coordinates need to be set manually via the endpoint **POST** `/location`. The set coordinates can be queried via the endpoint **GET** `/location`.

5.4 Network settings

The network settings can be changed via the endpoint `POST /ethernet`. It can be set whether or not a DHCP server is used (`dhcp`). If no DHCP server is used to assign the IP, a static IP address (`ip_address`) and subnet mask (`subnet_mask`) have to be set. Additionally the gateway address (`gateway`) and the DNS name server (`nameservers`) can be specified.

```
{
  "dhcp": false,
  "ip_address": "10.0.0.73",
  "subnet_mask": "255.255.255.0",
  "gateway": "10.0.0.1",
  "nameservers": [
    "10.0.0.1"
  ]
}
```

Important: Changes to the network settings should be noted beforehand and made with particular care. If a static IP address is set, the DALI-2 IoT only reacts to this address. If no automatic address can be obtained via DHCP, the DALI-2 IoT uses the address 169.254.0.1 with the subnet mask 255.255.0.0 as a fallback (approx. 20Sec after powerup).

The network settings can be queried via the endpoint `GET /ethernet`. In addition to the parameters above, the MAC address of the DALI-2 IoT and, if DHCP is used, the expiry of the current DHCP lease object (`dhcp_lease`) are returned.

```
{
  "mac_address": "AA:BB:CC:DD:EE:FF",
  "settings": {
    "dhcp": false,
    "ip_address": "10.0.0.73",
    "subnet_mask": "255.255.255.0",
    "gateway": "10.0.0.1",
    "nameservers": [
      "10.0.0.1"
    ]
  },
  "dhcp_lease": null
}
```

5.5 Email

Email settings became available in version 1.3 and will be used for notifications for automatic emergency tests in an upcoming release.

Email settings can be changed via the endpoint `PUT /email`. To enable sending email reports, it is required to set up a mail configuration for sending (`mailConfig`), which contains the details of an SMTP server (`server`, `port`, `security`), login data (`username`, `password`) and the sender's name and email address. Three security modes are supported for the SMTP communication: plain text (`"none"`), SSL (`"sslTls"`) and opportunistic TLS (`"startTls"`). It is recommended to create a dedicated email account for the DALI-2 IoT, and to use a separate "app password" for sending email.

```
{
  "mailConfig": {
    "server": "smtp.example.com",
    "port": 25,
```

```
"security": "startTls",
"username": "username",
"password": "password",
"senderName": "Sender Name",
"senderEmail": "sender@example.com"
},
"notifications": {...}
}
```

In addition to the mail configuration, the email settings configure notifications for automatic reporting of emergency test results. Each available test can send a report when scheduled tests finish, when the test is successful (`sendOnSuccess`) or fails (`sendOnFailure`). Messages will be delivered to a preconfigured list of mail receivers.

```
{
  "mailConfig": {...},
  "notifications": {
    "functionTest": {
      "sendOnSuccess": true,
      "sendOnFailure": true
    },
    "functionTest": {
      "sendOnSuccess": true,
      "sendOnFailure": true
    },
    "functionTest": {
      "sendOnSuccess": true,
      "sendOnFailure": true
    },
    "mailReceivers": [
      "receiver@example.com"
    ]
  }
}
```

Email settings can be tested, by sending a request to the endpoint `POST /email`, which does not require a request body. The endpoint `GET /email` can be used to request the mail configuration.

6 Websocket Interface

6.1 General Communication

Events, such as a change of the light level of a device, are signalled over websocket connections by the DALI IoT. This allows multiple clients to sustain a permanent connection to the DALI IoT, to receive status updates from the DALI IoT and to react to them.

An example for a simple receiver that prints these events to the command line can be created with only a few lines of Python code (<IP_ADDRESS> must be replaced by the IP address of the DALI IoT).

```
import asyncio
import websockets

async def receive_message(websocket):
    async for message in websocket:
        print(message)

async def run_client(url):
    try:
        async with websockets.connect(url) as websocket:
            wait_task = asyncio.create_task(
                receive_message(websocket)
            )
            await asyncio.wait([wait_task])
            wait_task.result()
    except:
        pass

# insert ip address or host name
asyncio.run(run_client("ws://<IP_ADDRESS>"))
```

Every websocket event transmits a packet in JSON format, with fields for the type of event (type), a data field that depends on the type of event (data) and a signature consisting of a timestamp and a counter, to differentiate events (timeSignature). Currently, the following event types are supported: DALI bus scan events (scanProgress), device events (devices and devicesDeleted), DALI bus monitoring events (daliMonitor), display events of status messages (messageFlash) and events for testing connections (ping). Specifics about the particular event types and their corresponding data fields are explained in the following sections.

6.1.1 Greeting message (*info*)

When a websocket connection is first established with the DALI-2 IoT, it will send out an info message that describes some of its properties. The content of the data field of this message is equivalent to the response of a `GET /info` request (section 5.1).

```
{
  "type": "info",
  "data": {
    "name": "dali-iot",
    "version": "v1.2.0/1.0.9",
    "tier": "plus",
    "emergencyLight": true,
    "errors": {},
    "descriptor": {
      "lines": 1,
      "bufferSize": 32,
      "tickResolution": 1978,
      "maxYnFrameSize": 32,
      "deviceListSpecifier": true,
      "protocolVersion": "1.0"
    }
  },
}
```



```
"device": {
  "serial": 1234567890,
  "gtin": 1234567890,
  "pcb": "9a",
  "articleNumber": 1234567890,
  "articleInfo": "",
  "productionYear": 2021,
  "productionWeek": 31
},
"timeSignature": {
  "timestamp": 1644577937.2377043,
  "counter": 17
}
```

6.1.2 Dismissing Event Types (*filtering*)

For each websocket connection to the DALI-2 IoT, a websocket event of the type "filtering" can be used to define which event types the DALI-2 IoT should filter out. To set up the filter, a filtering event needs to be sent to the DALI-2 IoT by the client. Each event type is assigned a boolean value that indicates whether the type should be filtered (true) or not filtered (false). Filtered events are no longer sent over the connection until the filter is removed from the client. By default, no filters are configured for a new connection.

```
{
  "type": "filtering",
  "data": {
    "daliMonitor": true,
    "fileUpload": false
  }
}
```

6.2 Direct DALI access

6.2.1 DALI Bus Status Events (*daliStatus*)

DALI bus status events (`daliStatus`) are sent out by the hardware interface, when the state of the DALI bus changes. The status code is an integer number, explained in table 2.

```
{
  "type": "daliStatus",
  "data": {
    "status": 0,
    "line": 0
  }
}
```

Table 2: Status codes in `daliStatus` events.

Status Value	Description
0	The DALI bus is no longer powered.
1	System failure in the hardware interface.
2	The DALI bus is powered.
3	The send buffer of the hardware interface is full.
4	The send buffer of the hardware interface is empty.
5	The DALI bus power is low.

60	An macro was stopped due to a timeout or aborted.
61	An macro sent an intermediate event.
62	An macro has failed.
63	An macro has succeeded.

6.2.2 DALI Bus Monitor Events (*daliMonitor*)

DALI bus monitoring events (*daliMonitor*) are sent out by the hardware interface, when commands or answers to commands are registered on the DALI bus. They contain

- the time tick of the hardware event (`tick_us`),
- a timestamp of the registration in the software (`timestamp`),
- the length of the DALI command (`bits`),
- the command data (`data.data`) are transmitted in these events, and
- the line on which the event occurred (`data.line`).

Monitoring events are close to the hardware and therefore contain DALI commands as integers. Their command data depends on whether a command or an answer to a command was registered.

```
{
  "type": "daliMonitor",
  "data": {
    "tick_us": 86454,
    "timestamp": 1627631911.299144,
    "bits": 16,
    "data": [
      0,
      145
    ],
    "line": 0
  },
  "timeSignature": {
    "timestamp": 1627631911.3002446,
    "counter": 481
  }
}
```

Moreover, the command data differs depending on the type of command: DALI, DALI-2 or eDALI.

- DALI commands have a length of 16 bits and two values in the data field: address and opcode [IEC62386-102].
- DALI-2 commands have a length of 24 bits and three values in the data field: address, instance byte and opcode [IEC62386-103].
- eDALI commands have a length of 25 bits and three values in the data field.
- Answers have a length of 8 bits, a single value in the data field: the answer of the device [IEC62386-102], and an additional error field (`data.framingError`) to indicate erroneous frames. In the event of an error, this field contains the value `true` and the data field contains a single value `null`. In the general case, the error and the data field contain `false` and the answer, respectively.

```
{
  "type": "daliMonitor",
  "data": {
    "tick_us": 86468,
    "timestamp": 1627631911.3125844,
    "bits": 8,
    "data": [
      255
    ],
    "line": 0,
    "framingError": false
  },
  "timeSignature": {
    "timestamp": 1627631911.3135417,
    "counter": 482
  }
}
```

6.2.3 Sending DALI commands (*daliFrame* and *daliAnswer*)

The message type `daliFrame` can be used to directly send DALI commands to the DALI bus. Answers to direct DALI commands use the `daliFrame` type for send confirmations and errors, and the `daliAnswer` type for answers from the DALI Bus. Answers will only be returned to the websocket connection that originally sent a `daliFrame` message.

A `daliFrame` message includes the line on which to send the command, the number of bits of the command, the send mode, and the DALI data to send to the bus. The send mode contains whether to submit the command twice, whether to wait for an answer and the DALI priority of the command. An example for a `QUERY CONTROL GEAR PRESENT` command sent to address 0 is displayed below.

```
{
  "type": "daliFrame",
  "data": {
    "line": 0,
    "numberOfBits": 16,
    "mode": {
      "sendTwice": false,
      "waitForAnswer": true,
      "priority": 3
    },
    "daliData": [
      1, 145
    ]
  }
}
```

One `daliFrame` message is returned to the websocket client for each submitted DALI command. A `daliFrame` request with send mode `"twice": true` will therefore return two such answers.

```
{
  "type": "daliFrame",
  "data": {
    "line": 0,
    "result": 0
  }
}
```

These answers contain a result code that describes whether the DALI command was successfully submitted to the DALI bus or an error state. The concrete error states are described in table 3.

Table 3: Returned results in `daliFrame` answer messages.

Result	Description
0	The DALI command was sent to the DALI bus.
1	The command could not be sent due to a DALI bus voltage error.
2	The command could not be sent due to using the DALI initialize mode.
3	The command could not be sent due to using the DALI quiescent mode.
4	The send buffer of the DALI interface is full.
5	The DALI interface does not support the requested line number.
6	The command contains a syntax error.
7	The command could not be sent due to an active macro.
61	The command could not be sent due to a collision on the DALI bus.
62	The command could not be sent due to a DALI bus error.
63	The command could not be sent due to a timeout in the DALI interface.
100	The DALI interface did not return an answer.

Messages with a `daliAnswer` type will be returned for DALI commands with a `send modes "waitForAnswer": true`. They contain the line on which the answer was received, a result and the data that was returned on the DALI bus. Answer results are either 0 for no answer, 8 for an answer with an 8-bit value, or 63 for a framing error on the DALI bus. Framing errors may occur when a query is sent to several gears at once and each responds with a different value.

```
{
  "type": "daliAnswer",
  "data": {
    "line": 0,
    "result": 8,
    "daliData": 255
  }
}
```

6.3 Events

6.3.1 DALI Bus Scan Progress Events (*scanProgress*)

Scan progress events (`scanProgress`) are continuously sent out during an ongoing DALI bus scan. The data field of such events is identical to the `ScanMode1`, which can be queried through the web API (refer to section 3.4.1)

```
{
  "type": "scanProgress",
  "data": {
    "id": "8e2fdf64-bde7-4114-ad7c-ee4128ac986e",
    "progress": 0.78125,
    "found": 0,
    "status": "addressing"
  },
  "timeSignature": {
    "timestamp": 1627565631.0883,
  }
}
```

```
"counter": 11
}
```

6.3.2 Device Events (*devices* and *devicesDeleted*)

Device events (*devices*) are sent out when new devices are added to the known devices, such as during a device scan (section 3.4.1). These events contain a list of new devices and their states as a data field, which are identical to the return values of device queries in the web API (section 3.4.2).

```

{
  "type": "devices",
  "data": {
    "devices": [
      {
        "id": 1,
        "name": "Dali #0",
        "address": 0,
        "line": 0,
        "type": "default",
        "features": {
          "switchable": {
            "status": false
          },
          "dimmable": {
            "status": 0.0
          },
          "scene": true,
          "colorRGB": {
            "status": {
              "r": 1.0,
              "g": 1.0,
              "b": 1.0
            }
          }
        }
      }
    ]
  },
  "timeSignature": {
    "timestamp": 1627639253.1174,
    "counter": 217
  }
}

```

Note that the devices had no address and line attributes in versions prior to 1.2, which instead used an info attribute with a string value. The attribute daliTypes was added in version 1.4.

Additional device events (devices) are sent out when the states of devices are changed, such as when a new light level is configured. This event differs from a device event for newly added devices in that only the changed properties are transmitted in the data field. If, for example, a device with identifying number 1 is added to a group 1, then the data field only contains the list of groups. When a dimmer that was previously switched off is dimmed to 100 %, then the event will contain the new states of the “switchable” and “dimmable” features.

```

{
  "type": "devices",
  "data": {
    "devices": [
      {
        "id": 1,
        "groups": [
          0,
          1
        ]
      }
    ]
  },
  "timeSignature": {
    "timestamp": 1627639378.9165,
    "counter": 333
  }
}

```

```

{
  "type": "devices",
  "data": {
    "devices": [
      {
        "id": 1,
        "features": {
          "switchable": {
            "status": true
          },
          "dimmable": {
            "status": 100.0
          }
        }
      }
    ]
  },
  "timeSignature": ...
}

```

Device deleted events (devicesDeleted) are sent out when devices are deleted, such as with the endpoint **DELETE** /devices. The data field then contains a deleted key that is mapped to a list of identifying numbers of all devices that were deleted.

```
{
  "type": "devicesDeleted",
  "data": {
    "deleted": [
      1,
      2
    ]
  },
  "timeSignature": {
    "timestamp": 1627639216.5944345,
    "counter": 169
  }
}
```

6.3.3 Zone Events (*zones* and *zonesDeleted*)

Zone events (*zones*) are sent out when new zones are added or existing zones are updated. These events contain a list of new or updated zones, which are identical to the return values of zone queries in the web API (section 3.6.2).

```
{
  "type": "zones",
  "data": {
    "zones": [
      {
        "name": null,
        "targets": [
          {
            "type": "group",
            "id": 0
          },
          {
            "type": "device",
            "id": 1
          }
        ]
      }
    ]
  },
  "timeSignature": {
    "timestamp": 1627639216.5944345,
    "counter": 169
  }
}
```

Zone deleted events (*zonesDeleted*) are sent out when zones are deleted, such as with the endpoints `DELETE /devices` or `DELETE /zone/{_id}`. The data field then contains a deleted key that is mapped to a list of identifying numbers of all zones that were deleted.

```
{
  "type": "zonesDeleted",
  "data": {
    "deleted": [
      1,

```

```
    2
  ]
},
"timeSignature": {
  "timestamp": 1627639216.5944345,
  "counter": 169
}
}
```

6.3.4 Sequence Events (*sequences* and *sequencesDeleted*)

Sequence events (*sequences*) are sent out when new sequences are added or existing sequences are updated. These events contain a list of new or updated sequences, which are identical to the return values of sequence queries in the web API (section 4.1.2).

```
{
  "type": "sequences",
  "data": {
    "sequences": [
      {
        "name": "turn on after 2 seconds",
        "enabled": true,
        "loop": true,
        "repeat": 0,
        "steps": [
          {
            "enabled": true,
            "type": "features",
            "data": {
              "targets": [
                {
                  "type": "broadcast"
                }
              ],
              "features": {
                "switchable": true
              },
              "delay": 2
            }
          }
        ],
        "id": 1,
        "active": true
      }
    ]
  },
  "timeSignature": {
    "timestamp": 1627639216.5944345,
    "counter": 169
  }
}
```


Sequence deleted events (`sequencesDeleted`) are sent out when sequences are deleted, such as with the endpoint `DELETE /automations/sequence/{_id}`. The data field then contains a `deleted` key that is mapped to a list of identifying numbers of all sequences that were deleted.

```
{
  "type": "sequencesDeleted",
  "data": {
    "deleted": [
      1,
      2
    ]
  },
  "timeSignature": {
    "timestamp": 1627639216.5944345,
    "counter": 169
  }
}
```

6.3.5 Scheduler Events (*schedulers* and *schedulersDeleted*)

Scheduler events (`schedulers`) are sent out when new schedulers are added or existing schedulers are updated. These events contain a list of new or updated schedulers, which are identical to the return values of scheduler queries in the web API (section 4.2.2).

```
{
  "type": "schedulers",
  "data": {
    "schedulers": [
      {
        "id": 1,
        "name": "Turn off at 18:00",
        "enabled": true,
        "targets": [
          {
            "type": "device",
            "id": 1
          }
        ],
        "activePeriod": {},
        "activeMonths": {},
        "activeWeekdays": {},
        "activeDays": {},
        "recallMode": "timeOfDay",
        "recallTime": {
          "hour": 18,
          "minute": 0,
          "second": 0
        },
        "action": {
          "type": "features",
          "data": {

```

```

        "features": {
            "switchable": false
        }
    }
}
],
"timeSignature": {
    "timestamp": 1644836407.57024,
    "counter": 21
}
}

```

Scheduler deleted events (`schedulersDeleted`) are sent out when schedulers are deleted, such as with the endpoint `DELETE /automations/scheduler/{_id}`. The data field then contains a deleted key that is mapped to a list of identifying numbers of all schedulers that were deleted.

```

{
  "type": "schedulersDeleted",
  "data": {
    "deleted": [
      1,
      2
    ]
  },
  "timeSignature": {
    "timestamp": 1627639216.5944345,
    "counter": 169
  }
}

```

6.3.6 Circadian Events (*circadians* and *circadiansDeleted*)

Circadian events (`circadians`) are sent out when new circadian automations are added or existing automations are updated. These events contain a list of new or updated circadian automations, which are identical to the return values of circadian automation queries in the web API (section 4.3.2).

```

{
  "type": "circadians",
  "data": {
    "circadians": [
      {
        "name": "New",
        "id": 1,
        "targets": [],
        "enabled": true,
        "longest": {
          "day": 22,
          "month": 6,
          "steps": [

```

```

        {
            "hour": 0,
            "colorKelvin": 2700.0
        },
        ...
    ]
},
"shortest": {
    "day": 21,
    "month": 12,
    "steps": [
        {
            "hour": 0,
            "colorKelvin": 2700.0
        },
        ...
    ]
}
}
]
},
"timeSignature": {
    "timestamp": 1644836898.3646882,
    "counter": 28
}
}

```

Circadian deleted events (`circadiansDeleted`) are sent out when circadian automations are deleted, such as with the endpoint `DELETE /automations/circadian/{_id}`. The data field then contains a `deleted` key that is mapped to a list of identifying numbers of all circadian automations that were deleted.

```

{
  "type": "circadiansDeleted",
  "data": {
    "deleted": [
      1,
      2
    ]
  },
  "timeSignature": {
    "timestamp": 1627639216.5944345,
    "counter": 169
  }
}

```

6.3.7 Trigger-Actions (*triggerActions* and *triggerActionsDeleted*)

Trigger action events (`triggerActions`) are sent out when new trigger actions are added or existing trigger actions are updated. These events contain a list of new or updated trigger actions, which are identical to the return values of trigger action automation queries in the web API (section 4.4.2).

```
{
  "type": "triggerActions",
  "data": {
    "triggerActions": [
      {
        "id": 1,
        "enabled": true,
        "name": "New Link",
        "sources": [
          {
            "type": "device",
            "id": 1
          }
        ],
        "targets": [
          {
            "type": "zone",
            "id": 1
          }
        ]
      }
    ]
  },
  "timeSignature": {
    "timestamp": 1644837187.7067018,
    "counter": 35
  }
}
```

Trigger actions deleted events (`triggerActionsDeleted`) are sent out when trigger actions are deleted, such as with the endpoint `DELETE /automations/triggerActions/{_id}`. The data field then contains a `deleted` key that is mapped to a list of identifying numbers of all trigger actions that were deleted.

```
{
  "type": "triggerActionsDeleted",
  "data": {
    "deleted": [
      1,
      2
    ]
  },
  "timeSignature": {
    "timestamp": 1627639216.5944345,
    "counter": 169
  }
}
```

6.3.8 Changes of System Time Settings (*datetime*)

If the system time is changed via the settings of the DALI-2 IoT, the new configuration is sent out as an event. The date time configuration consists of the time zone, the date and time, and the setting whether the time should be automatically obtained from the Internet (*automatic_time*).

```
{
  "type": "datetime",
  "data": {
    "timezone": "Europe/Vienna",
    "automatic_time": true,
    "date": "28. October 2021",
    "time": "10:42"
  },
  "timeSignature": {
    "timestamp": 1635410542.061734,
    "counter": 2
  }
}
```

6.3.9 Display of Status Message Events (*messageFlash*)

The event type *messageFlash* is used to transfer general status messages. These events address users and inform in the event of errors, such as a disconnected DALI bus.

```
{
  "type": "messageFlash",
  "data": {
    "message": "Flash message text",
    "seconds": 30,
    "userDismissible": true
  },
  "timeSignature": {
    "timestamp": 1627565751.8057,
    "counter": 12
  }
}
```

6.3.10 Events for Testing Connections (*ping*)

If the API endpoint `POST /ping/echo` is queried, a ping event is sent to all websocket connections. This mechanism can be used for connection tests.

Request

```
{
  "echo": "test",
  "timeSignature": {
    "timestamp": 1635411825.11234,
    "counter": 4
  }
}
```

Event

```
{
  "type": "ping",
  "data": {
    "echo": "test"
  },
  "timeSignature": {
    "timestamp": 1635411825.11206,
    "counter": 3
  }
}
```

} 

7 Document History

Revision	Changes	Date
1.0	Initial release	20.09.2021
2.0	Added: <ul style="list-style-type: none"> • colorWAF Feature • Zones • Trigger Action Automation • Email Settings • Additional information in General Information of the DALI-2 IoT Info, Bus Status, Zone, Email Settings and Automation Websocket Events Changed: <ul style="list-style-type: none"> • Moved Websocket Interface after General Settings 	14.02.2022
3.0	Added: <ul style="list-style-type: none"> • Additional name attribute in the discovery answers. • Added release version 1.3 to email settings. • Added daliTypes attribute to DALI devices, since version 1.4. • Added daliFrame and daliAnswer websocket messages for direct DALI bus access, since version 1.2. Changed: <ul style="list-style-type: none"> • Changed info attribute in DALI devices to explicit address and line attributes, since version 1.2. • Separated Websocket Interface into subsections for general communication, DALI bus access and events. 	06.04.2022

8 Bibliography

IEC62386-102: International Electrotechnical Commission, Digital addressable lighting interface - Part 102: General requirements - Control gear, 2014
 IEC62386-103: International Electrotechnical Commission, Digital addressable lighting interface - Part 103: General requirements - Control devices, 2014