deCONZ REST API Documentation

Contents

Introduction

This documentation describes the REST API, which is provided by the deCONZ REST API Plugin from dresden elektronik that runs a lightweight HTTP server within the deCONZ application on the Raspberry Pi.

The REST API allows third party applications easy monitoring and control of a ZigBee network from local or remote operating clients.

One of the following devices is needed to get ZigBee support on the Raspberry Pi or PC.

- RaspBee ZigBee shield for Raspberry Pi
- ConBee USB radio stick for PC or Raspberry Pi

Features

- Support for ZigBee Home Automation (HA) and ZigBee Light Link (ZLL) based lights
- Add, remove and modify groups of lights
- Control single lights or groups
- Control colors and dimmlevels via hue, saturation, brightness, CIE xy color coordinates
- Smooth transitions of colors and dimming over time
- Save and recall individual scenes for a group
- Create rules to automate light control
- Trigger timed commands
- Reset ZLL lights to factory new state

Extensibility

The deCONZ REST API Plugin is a open source project licensed under the BSD license and available at GitHub. It could therefore be extended with further functionality, for example to support more devices.

About REST Documentation

Introduction

REST stands for Representational State Transfer and sets the ground for various modern web based APIs. The main idea behind REST is that everything is a resource and has a state.

Resources are represented by URLs like:

- `/lights` - a collection of lights
- `/lights/1` - a single light
- `/lights/1/state` - the current state of a light

API endpoints

All resources are provided by so called endpoints. The API endpoint documentation can be found in the menu on the left side.

Currently the following endpoints are available.

Endpoint	Description
/config	Interface to query and modify the gateway configuration.
/lights	Interface for single lights.
/groups	Interface for groups of lights.
/scenes	Interface to the scenes of a group.
/schedules	Interface for timed commands.
/touchlink	Interface for touchlink commands.

More endpoints and functionality will be added in future.

Methods

Resources can be queried and modified with standard HTTP methods. Where GET, PUT, POST and DELETE are only a subset of all possible methods, they are by far the most used ones.

Method	Description

GET	Query the content of a ressource.
PUT	Modifies a **existing** ressource.
POST	Creates a **new** ressource which did not exist before.
DELETE	Deletes a ressource.

JSON

The contents of ressources are often expressed in Javascript Object Notation better known as JSON. That's not a requirement of REST itself, in fact some APIs also use XML but JSON is by far more popular due to its simplicity.

The JSON format is a very simple but powerful notation to express structured objects and lists. The following example covers everything that can be expressed with JSON.

Example object

```
{
        "a_string": "this is a string",
        "a_number": 5,
        "a_list": [ 1, 2.0, 3, 4 ],
        "a_mixed_list": [ 2, {}, "name", 6, [ 1, 2 ,3 ] ],
        "a_nested_object": {
                "foo": "bar"
                }
}
```

- Strings are always double quoted `"like this"`
- Keys and values are separated by a colon `:` and keys are always strings like `"key"`
- Objects `{ }` and lists `[ ]` might be empty and can be nested
- Numbers can be integers `1` or fractional `0.5`

That's all about JSON.

URLs and the API key

When reading the API endpoint documentation URLs will look like `/api/<apikey>/lights`.

The `/api` prefix separates the API interface from the HTML5 web application which is reachable through the document root `/`.

Nearly every API request requires a so called **API key** which is a *mandatory* part of request URLs.

The API key has the only purpose to restrict access to the gateway. Remember the gateway is reachable through the whole local network and without the API key requirement anybody could control the lights.

Nevertheless all clients need to acquire API key by means of the configuration endpoint.

Benefits

- Clients might access the API local or remote via network
- Access from any desktop and mobile platform
- Access from any programming language
- All popular programming languages provide helper classes and functions to work with RESTful APIs
- The format of requests and responses is human readable
- Learning and using REST APIs is pretty straight forward

What’s next

Now you know the basics about REST. It’s time to move on to the Getting Started section which explains step by step how to acquire an API key and do some basic control of the lights.

Getting started Documentation

Introduction

This section describes the first steps needed in order to use the API. If you are new to REST APIs please read the About REST section first.

Requirements

The only tool needed in this section is a browser with a **REST client add-on** to access the API. This document doesn’t cover the API access through a programming language since everybody may have its favorite language.

Get a REST client

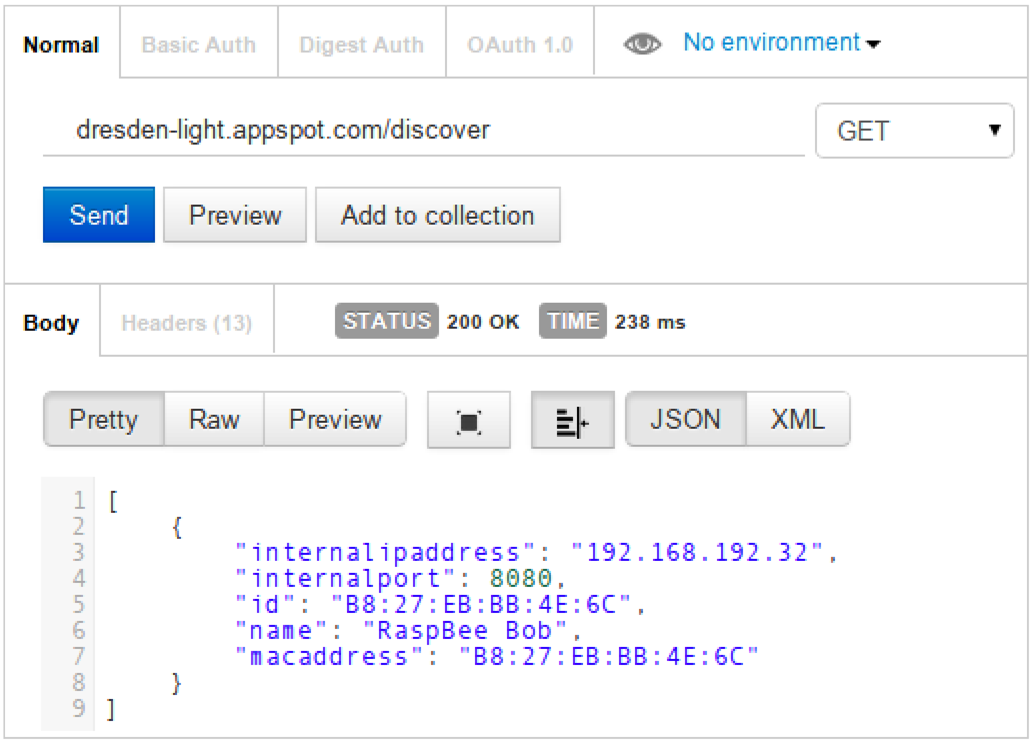
There are various free clients available; please pick one for your favourite browser in the browser add-on section.

In the following steps *Postman* for Chrome from the Google Webstore will be used. For Firefox the REST Client is another popular client.

Find your gateway

As first step the gateway IP address and port must be found.

This could be achieved by doing a `GET` request to `https://dresden-light.appspot.com/discover`.



dresden-light.appspot.com/discover

GET

STATUS 200 OK TIME 238 ms

```
[
    {
        "internalipaddress": "192.168.192.32",
        "internalport": 8080,
        "id": "B8:27:EB:BB:4E:6C",
        "name": "RaspBee Bob",
        "macaddress": "B8:27:EB:BB:4E:6C"
    }
]
```

The response body shows that the gateway has the IP address *192.168.192.32* and the API is reachable through port *8080*.

`Note` If the above request doesn't work, there are several other ways to find the gateway IP address as described in the Discovery section.

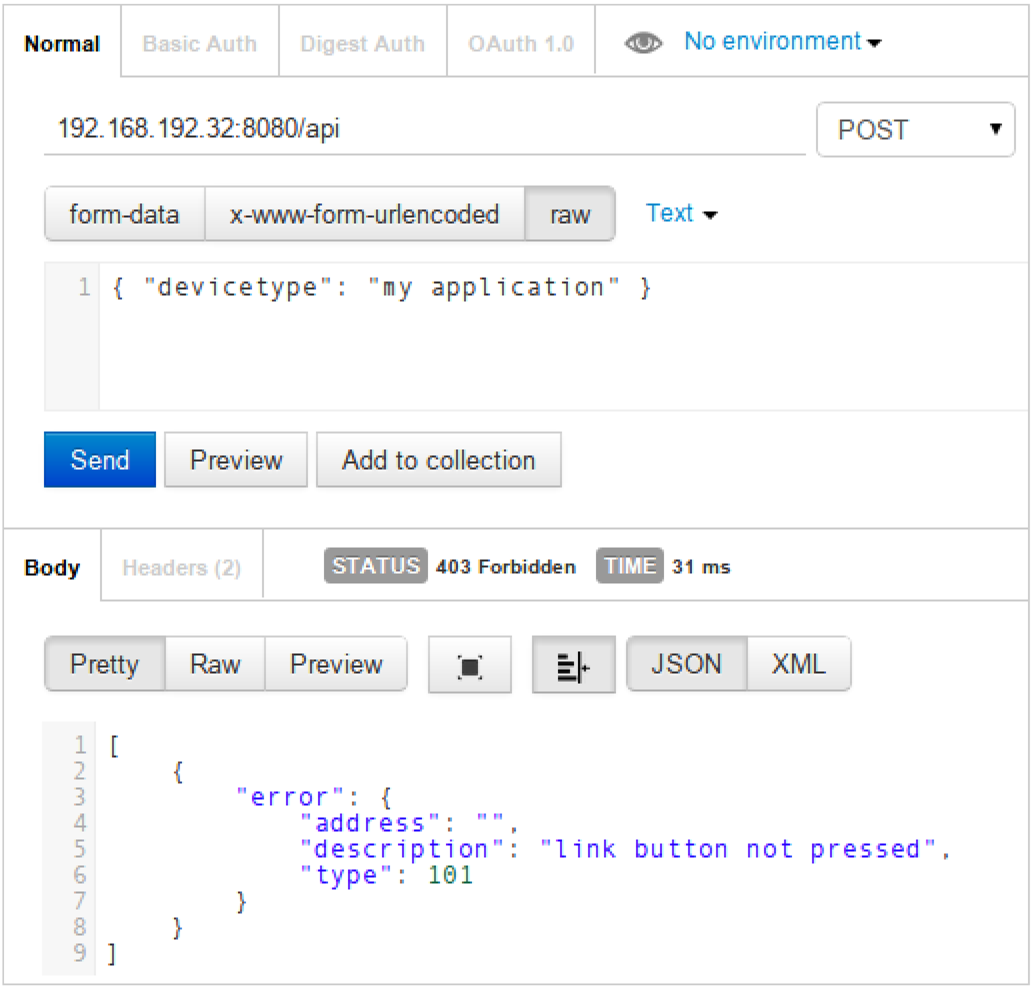
Acquire an API key

Any client that wants to access the API must provide a valid API key otherwise the access will fail.

To acquire an API key send a `POST` request to `/api` as follows. Use the IP address and the port of your gateway that you got during discovery.

`Note` The request must contain a JSON object with the required field *devicetype*.

`Note` In some Rest clients it is mandatory to put 'http://' in front of the IP address of the gateway.



192.168.192.32:8080/api

```
{ "devicetype": "my application" }
```

STATUS 403 Forbidden TIME 31 ms

```
[
    {
        "error": {
            "address": "",
            "description": "link button not pressed",
            "type": 101
        }
    }
]
```

`--->` **This didn't work!**

The STATUS says `403 Forbidden`.

The response body provides further information about the raised error in the JSON object.

Unlock the gateway

The reason why the request failed is that the gateway was not unlocked. This mechanism is needed to prevent anybody from access to the gateway without being permitted to do so.

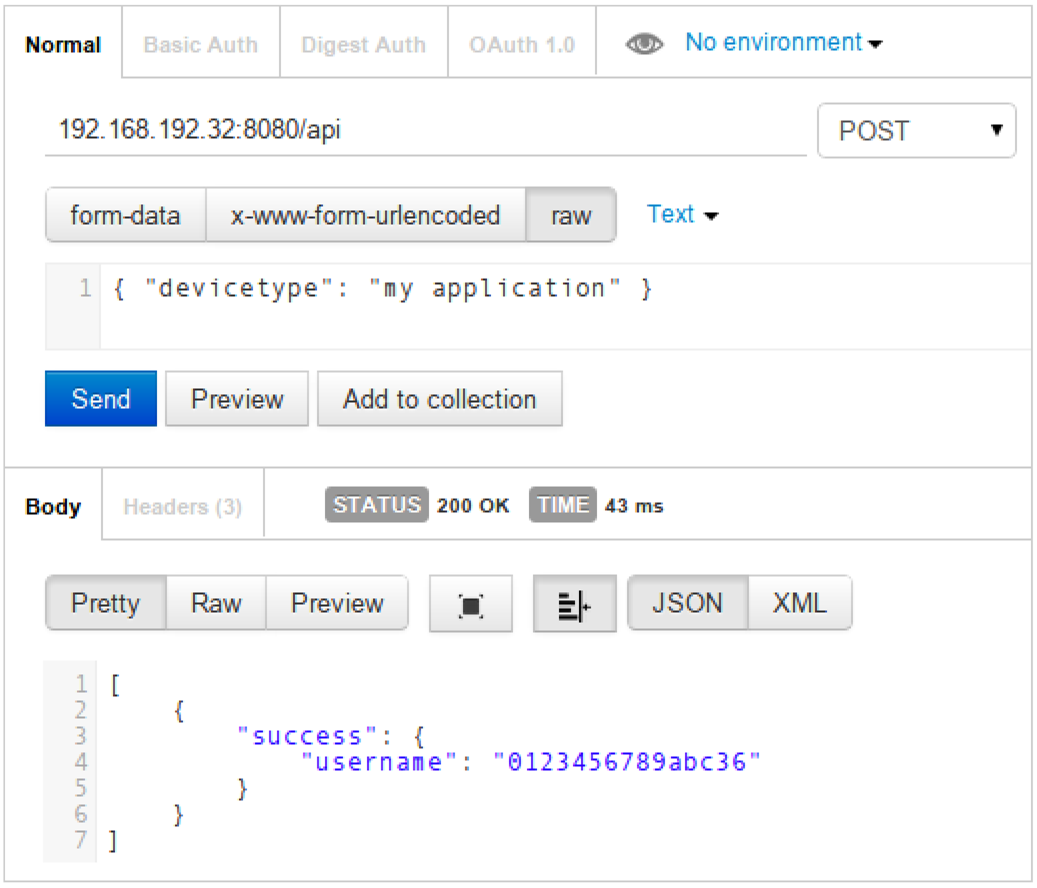
As described in the section Authorization unlock the gateway as follows:

- In a new browser tab open the webapp
- Click on `Menu/Settings` from the top menu
    - Click on the Unlock Gateway button

Now the gateway is unlocked for *60 seconds*.

Second attempt

Within 60 seconds after unlocking the gateway, go back to the REST client and repeat the acquire API key request as before. (just click on *Send* again)



192.168.192.32:8080/api

POST

form-data x-www-form-urlencoded raw Text

```
{ "devicetype": "my application" }
```

STATUS 200 OK TIME 43 ms

```
[
    {
        "success": {
            "username": "0123456789abc36"
        }
    }
]
```

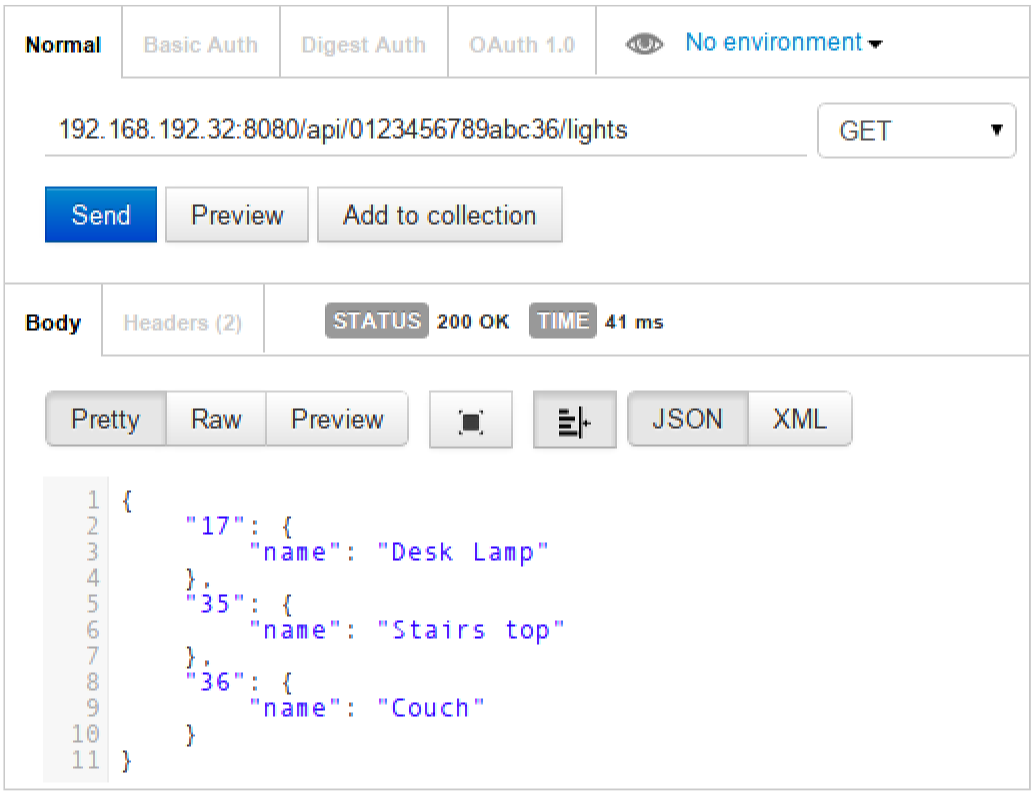
This time the request succeded with STATUS 200 OK.

In the response body the new API key is in the field `username`, from now on this key will be used in further API requests.

Get a list of all lights

With the API key from the last section it is now possible to access the full API.

To get a list of all available lights run a `GET` request to `/api/<apikey>/lights` as follows.



192.168.192.32:8080/api/0123456789abc36/lights GET

STATUS 200 OK TIME 41 ms

```
{
    "17": {
        "name": "Desk Lamp"
    },
    "35": {
        "name": "Stairs top"
    },
    "36": {
        "name": "Couch"
    }
}
```

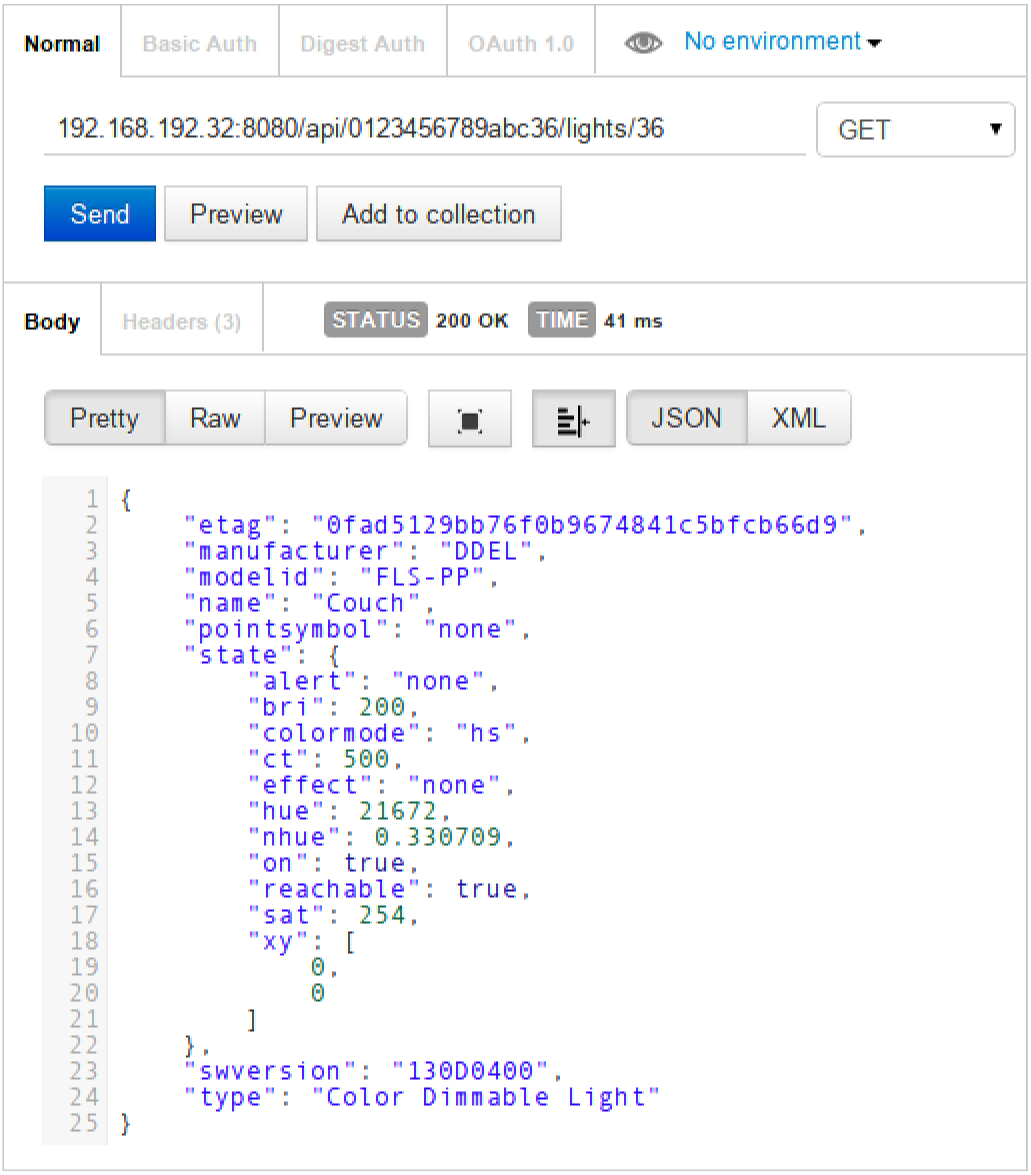
In the response 3 lights where returned. There are serval things to note here.

- The response contains not a list like `[ ]` of lights but a object `{ }` with key/value pairs
- Each light can be accessed by its id `"17"`
- The light id is a key in the response object and the related value is a further object

`Note` Ids are strings and even if they contain numbers **never** expect them to be “1”, “2”, “3”, … if the user removes light “2” the list will become “1”, “3”.

Get the details of a light

To get the detail of a light do a `GET` request to `/api/<apikey>/lights/<id>` as follows.



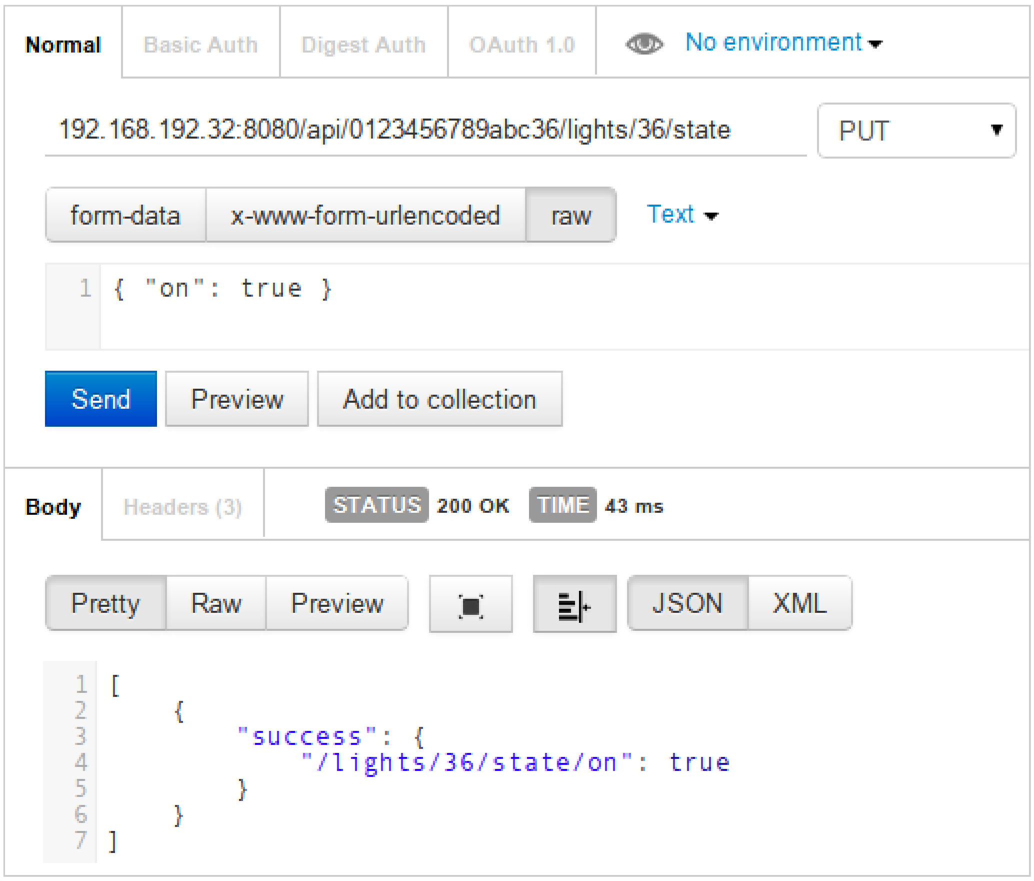
```
192.168.192.32:8080/api/0123456789abc36/lights/36    GET
```

STATUS 200 OK TIME 41 ms

```
{
    "etag": "0fad5129bb76f0b9674841c5bfcb66d9",
    "manufacturer": "DDEL",
    "modelid": "FLS-PP",
    "name": "Couch",
    "pointsymbol": "none",
    "state": {
        "alert": "none",
        "bri": 200,
        "colormode": "hs",
        "ct": 500,
        "effect": "none",
        "hue": 21672,
        "nhue": 0.330709,
        "on": true,
        "reachable": true,
        "sat": 254,
        "xy": [
            0,
            0
        ]
    },
    "swversion": "130D0400",
    "type": "Color Dimmable Light"
}
```

Turn light on/off

To turn a light on/off do a `PUT` request to `/api/<apikey>/lights/<id>/state` as follows.



192.168.192.32:8080/api/0123456789abc36/lights/36/state PUT

```
{ "on": true }
```

STATUS 200 OK TIME 43 ms

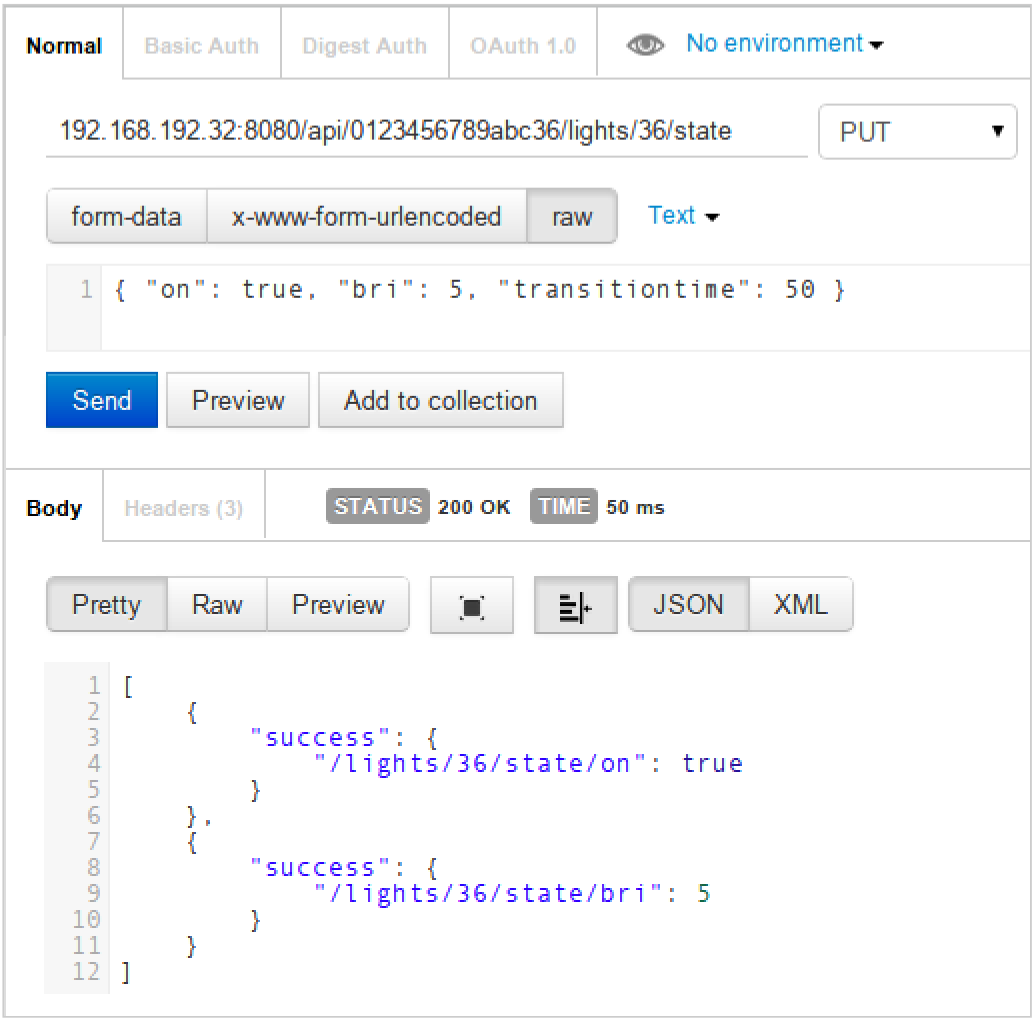
```
[
    {
        "success": {
            "/lights/36/state/on": true
        }
    }
]
```

In the request body set the `on` value to *true* or *false* to turn the light on and off.

Dim the light with transition time

Dimming is done the same way as sending on/off by using the `bri` parameter; additionally specify a transition time in 1/10 seconds.

The following example dims the light in 5 seconds down.



192.168.192.32:8080/api/0123456789abc36/lights/36/state

PUT

```
{ "on": true, "bri": 5, "transitiontime": 50 }
```

STATUS 200 OK TIME 50 ms

```
[
    {
        "success": {
            "/lights/36/state/on": true
        }
    },
    {
        "success": {
            "/lights/36/state/bri": 5
        }
    }
]
```

What’s next

To do some more advanced things with this API please refer to the *API endpoints* documentation on the left side menu.

Discovery Documentation

Finding the gateway

The gateway(s) in the local network can be discovered in various ways.

Discovery via internet

```
GET https://dresden-light.appspot.com/discover
```

This returns a JSON list of all known gateways in the local network.

If both the gateway and the application have access to the internet, discovery via the internet is the easiest way to find the gateway.

Response

```
[{
        "id": "E0:69:78:58:22:A4:32:CE",
        "internalipaddress": "192.168.192.34",
        "internalport": "8080",
        "macaddress": "E0:69:78:58:22:A4:32:CE",
        "name": "RaspBee GW"
}]
```

`Note` For webapps this is the only way to automatically find the gateway.

By visiting http://www.dresden-elektronik.de/discover a list of all gateways in the local network will be displayed. This is done by only using jQuery, Ajax and internet discovery.

Discovery via UPnP

Another method to find the gateway is UPnP discovery via UDP sockets.

The main advantage compared to the internet discovery is that no internet is needed at all.

`Note` The discovery might not work as expected if in the local network beside the main router also bridges are used, which might prevent UDP broadcasts to reach the whole network.

Discovery via nmap

Nmap is an open source command-line network scanner which is available for all major platforms. Since the gateway runs a SSH deamon at port 22 it is easy to find it in the local network.

```
$ nmap -p 22 -T5 -n -min-parallelism 100 --open 192.168.192.0/24
```

`Note` Replace the `192.168.192.0/24` with your subnetwork for example `192.168.0.0/24`.

Result

```
Starting Nmap 6.25 ( http://nmap.org ) at 2013-07-01 13:04 CEST
Nmap scan report for 192.168.192.34
Host is up (0.00081s latency).
PORT   STATE SERVICE
22/tcp open  ssh
```

Configuration

The configuration endpoint allows to retrieve and modify the current configuration of the gateway.

Acquire API key

```
POST /api
```

Creates a new API key which provides authorized access to the REST API.

`Note` The request will only succeed if the gateway is unlocked or valid HTTP basic authentification credentials are provided in the HTTP request header (see authorization).

Parameters

Field	Type	Description	Required
devicetype	String (0..40 chars)	Name of the client application.	required
username	String (10..40 chars)	Will be used as username. If not specified a random key will be generated.	optional

Example request data

```
{
    "username": "988112a4e198cc1211",
    "devicetype": "my application"
}
```

Response

```
HTTP/1.1 200 OK
```

```
[ { "success": { "username": "988112a4e198cc1211" } } ]
```

Possible errors

400 Bad Request

403 Forbidden

Delete API key

```
DELETE /api/<apikey>/config/whitelist/<apikey2>
```

Deletes an API key so it can no longer be used.

Parameters

None

Possible errors

403 Forbidden

404 Not Found

Get configuration

```
GET /api/<apikey>/config
```

Returns the current gateway configuration.

Parameters

None

Response

```
HTTP/1.1 200 OK
ETag: "203941fel3ds8ad61903224"
```

```
{
    "apiversion": "1.0.0",
    "dhcp": true,
    "gateway": "192.168.80.1",
    "ipaddress": "192.168.80.142",
    "linkbutton": false,
    "localtime": "2016-06-29T14:00:40",
    "mac": "74:46:a0:9e:92:c7",
    "name": "deCONZ-GW",
    "netmask": "255.255.255.0",
    "networkopenduration": 60,
    "panid": 56889,
    "portalservices": false,
    "proxyaddress": "",
    "proxyport": 0,
    "swupdate": {
        "notify": false,
        "text": "",
        "updatestate": 0,
        "url": "",
    },
    "swversion": "20405",
    "timeformat": "12h",
    "timezone": "Europe/Berlin",
    "utc": "2016-06-29T12:00:40",
    "uuid": "a65d80a1-975a-4598-8d5a-2547bc18d63b",
    "whitelist": {},
    "zigbeechannel": 20
}
```

Response fields

Field	Type	Description
apiversion	String	The version of the deCONZ Rest API

dhcp	Bool	Whether the IP address of the bridge is obtained with DHCP.
gateway	String	IPv4 address of the gateway.
ipaddress	String	IPv4 address of the gateway.
linkbutton	Bool	true if the gateway is unlocked.
localtime	String	The localtime of the gateway
mac	String	MAC address of the gateway.
name	String	Name of the gateway.
netmask	String	Network mask of the gateway.
networkopenduration	Number (0..65535)	Can be used to store the permitjoin (see Modify configuration) value permanently.
panid	Number (0..65535)	The ZigBee pan ID of the gateway.
portalservices	Bool	This indicates whether the bridge is registered to synchronize data with a portal account.
proxyaddress	String	Not supported
proxyport	Number	Not supported
softwareupdate	Object	Contains information related to software updates.

swversion	String	The software version of the gateway.
timeformat	String	Stores a value of the timeformat that can be used by other applications. "12h" or "24h"
timezone	String	Timezone used by the gateway (only available on Raspberry Pi Gateway). "None" if not further specified.
utc	String	Current UTC time of the gateway in ISO 8601 format.
uuid	String	UPNP Unique Id of the gateway
whitelist	Object	An array of whitelisted api keys.
zigbeechannel	Number	The current wireless frequency channel used by the Gateway. Supported channels: 11, 15, 20, 25.

Possible errors

304 Not Modified

403 Forbidden

Get full state

`GET /api/<apikey>`

Returns the full state of the gateway including all its lights, groups, scenes and schedules.

Parameters

None

Response

```
HTTP/1.1 200 OK
ETag: "203941fel3ds8ad61903224"
```

```
{
    "config": {
        "dhcp": true,
        "gateway": "192.168.178.1",
        "ipaddress": "192.168.192.237",
        "linkbutton": true,
        "mac": "E0:69:95:58:06:7F",
        "name": "RaspBee GW",
        "netmask": "255.255.255.0",
        "portalservices": false,
        "proxyaddress": "",
        "proxyport": 0,
        "swupdate": {
            "notify": false,
            "text": "",
            "updatestate": 0,
            "url": ""
        },
        "swversion": "1.12.3",
        "utc": "2013-05-22T12:02:30",
        "whitelist": {}
    },
    "groups": {
        "1": {
            "action": {
                "bri": 3945,
                "colormode": "hs",
                "ct": 500,
                "effect": "none",
                "hue": 0,
                "on": true,
                "sat": 17680,
                "xy": [0.0610457, 0.219979]
            },
            "devicemembership": [],
            "etag": "893f60b611274d1803207298cf26b1e1",
            "hidden": false,
            "lights": [ "1" ],
            "lightsequence": [ "1" ],
            "multideviceids": [],
            "name": "Office",
            "scenes": [
                "0": {
                    "id": "1",
                    "name": "blue moon"
                }
            ]
        }
    },
    "lights": {
        "1": {
            "etag": "030cf8c1c0025420f3a0659afab251f5",
            "name": "Desk Lamp",
            "modelid": "FLS-PP-01",
            "pointsymbol": {},
            "swversion": "14010400",
            "type": "Color Dimmable Light",
            "state": {
                "on": true,
                "bri": 190,
                "hue": 21672,
                "sat": 254,
                "ct": 500,
                "alert": "none",
                "colormode": "hs",
                "effect": "none",
                "reachable": true,
                "xy": [ 0.805343, 0.000612754 ]
            }
        }
    },
    "schedules": {
        "1": {
            "autodelete": false
            "command": {
            "address": "/api/AD4F14F244/groups/2/scenes/1/recall"
            "body": {}
            "method": "PUT"
            }
            "etag": "3dea322b33d34a9134e5632706448f8f"
            "name": "Good Morning"
            "status": "enabled"
            "time": "W124/T05:00:00"
        }
    },
    "sensors": {
        1: {
            "config": {
                "on": true
                "reachable": false
            }
            "etag": "01252de8b14f62a234a4680827cf1609"
            "manufacturername": "dresden elektronik"
            "mode": 2
            "modelid": "Lighting Switch"
            "name": "Lighting Switch 1"
            "state": {
                "lastupdated": "2016-06-29T13:16:41"
            }
            "swversion": "1.0"
            "type": "ZHASwitch"
            "uniqueid": "0x00212effff00a6bc"
            }
        }
       "rules": {}
    }
```

Response fields

Field	Type	Description
config	Object	Configuration of the gateway.
groups	Object	All groups of the gateway.
lights	Object	All lights of the gateway.
rules (as from deconz version > 2.04.12)	Object	All rules of the gateway.
schedules	Object	All schedules of the gateway.

Possible errors

304 Not Modified

403 Forbidden

Modify configuration

`PUT /api/<apikey>/config`

Modify configuration parameters.

Parameters

Field	Type	Description	Required
name	String (0..16 chars)	Name of the gateway.	optional
rfconnected	Bool	Set connected state of the gateway.	optional
updatechannel	String	Set update channel ("stable"|"alpha"|"beta").	optional
permitjoin	Number (0..255)	Open the network so that other zigbee devices can join. 0 = network closed, 255 = network open, 1..254 = time in seconds the network remains open. The value will decrement automatically.	optional
groupdelay	Number (0..5000)	Time between two group commands in milliseconds.	optional
otauactive	Bool	Set OTAU active or inactive.	optional

discovery	Bool	Set gateway discovery over the internet active or inactive.	optional
unlock	Number (0..600)	Unlock the gateway so that apps can register themselves to the gateway (time in seconds).	optional
zigbeechannel	Number (11|15|20|25)	Set the zigbeechannel of the gateway. Notify other ZigBee devices also to change their channel.	optional
timezone	String	Set the timezone of the gateway (only on Raspberry Pi). Format: tzdatabase e.g. “Europe/Berlin” https://en.wikipedia.org/wiki/List_of_tz_database_time_zones	optional
utc	String	Set the UTC time of the gateway (only on Raspbery Pi) in ISO 8601 format (yyyy-MM-ddTHH:mm:ss).	optional
timeformat	String ("12h"|"24h")	Can be used to store the timeformat permanently.	optional

Example request data

```
{
  "zigbeechannel": 25
}
```

Response

```
HTTP/1.1 200 OK
ETag: "203941fel3ds8ad61903224"
```

```
[
  {
    "success": {"/config/zigbeechannel": 25 }
  }
]
```

Possible errors

400 Bad Request

Update software

```
POST /api/<apikey>/config/update
```

Returns the newest software version available. Starts the update if available (only on raspberry pi).

Response

```
HTTP/1.1 200 OK
```

```
{
  "success": {
    "/config/update": "2.04.05"
  }
}
```

Update firmware

```
POST /api/<apikey>/config/updatefirmware
```

Starts the update firmware process if newer firmware is available.

Response

```
HTTP/1.1 200 OK
```

```
{
  "success": {
    "/config/updatefirmware": "26050500"
  }
}
```

Possible errors

503 Service Unavailable

Reset gateway

```
POST /api/<apikey>/config/reset
```

Reset the gateway network settings to factory new and/or delete the deCONZ database (config, lights, scenes, groups, schedules, devices, rules).

Parameters

Field	Type	Description	Required
resetGW	Bool	Set the network settings of the gateway to factory new.	optional
deleteDB	Bool	Delete the Database.	optional

At least one parameter is required!

Response

```
HTTP/1.1 200 OK
```

```
{
  "success": {
    "/config/reset": "success"
  }
}
```

Possible errors

400 Bad Request

503 Service Unavailable

Change password

```
PUT /api/<apikey>/config/password
```

Change the Password of the Gateway. The parameter must be a Base64 encoded combination of “<username>:<password>”.

Parameters

Field	Type	Description	Required
username	String	The user name (currently only “delight” is supported).	required
oldhash	String	The Base64 encoded combination of “username:old password”.	required

newhash	String	The Base64 encoded combination of “username:new password”.	required

Response

```
HTTP/1.1 200 OK
```

```
{
  "success": {
    "/config/password": "changed"
  }
}
```

Possible errors

400 Bad Request

401 Unauthorized

Reset password

```
DELETE /api/<apikey>/config/password
```

Resets the username and password to default (“delight”,”delight”). Only possible within 10 minutes after gateway start.

Response

```
    HTTP/1.1 200 OK
```

```
    {}
```

Possible errors

403 Forbidden

Groups

Groups are useful to control many lights at once and provide the base to use scenes.

Create group

`POST /api/<apikey>/groups`

Creates a new empty group.

Parameters

Field	Type	Description	Required
name	String	The name of the new group	required

Example request data

```
{ "name": "Garage" }
```

Response

```
HTTP/1.1 200 OK
```

```
[ { "success": { "id": "3" } } ]
```

Response fields

Field	Type	Description
id	String	The unique identifier of the group.

`Note` Creating a group with a name which already exists will not create a new group or fail. Such a call does only return the id of the existing group.

Possible errors

400 Bad Request

403 Forbidden

Get all groups

`GET /api/<apikey>/groups`

Returns a list of all groups.

Parameters

None

Response

```
HTTP/1.1 200 OK
```

```
{
    "1": {
        "devicemembership": [],
        "etag": "ab5272cfe11339202929259af22252ae",
        "hidden" : false,
        "name": "Living Room"
    },
    "2": {
        "devicemembership": ["3"],
        "etag": "030cf8c1c0025420f3a0659afab251f5",
        "hidden" : false,
        "name": "Kitchen"
    }
}
```

Response fields

Field	Type	Description
devicemembership	Array	If this group was created by a device (switch or sensor) this list contains the device ids.
name	String	Name of a group.
etag	String	HTTP etag which changes on any action to the group.
hidden	Bool	Indicates if this group is hidden.

Possible errors

403 Forbidden

Get group attributes

`GET /api/<apikey>/groups/<id>`

Returns the full state of a group.

Parameters

None

Response

```
HTTP/1.1 200 OK
ETag: "0b32030b31ef30a4446c9adff6a6f9e5"
```

```
{
    "action": {
        "bri": 0,
        "ct": 500,
        "effect": "none",
        "hue": 0,
        "on": false,
        "sat": 0,
        "xy": [ 0, 0 ]
    },
    "devicemembership": [],
    "etag": "0b32030b31ef30a4446c9adff6a6f9e5",
    "hidden": false,
    "id": "32772",
    "lights": [ "3","42","43" ],
    "lightsequence": [ "42","43","3" ],
    "multideviceids": ["2"],
    "name": "Livingroom",
    "scenes": [
        { "id": "1", "name": "warmlight" }
    ],
    "state": 0
}
```

Response fields

Field	Type	Description
action	Object	The last action which was send to the group.
action.on	Bool	true if the group was turned on.
action.bri	Number (0..255)	Brightness of the group. Depending on the lights 0 might not mean visible "off" but minimum brightness.
action.hue	Number (0..65535)	The hue parameter in the HSV color model is between 0°-360° and is mapped to 0..65535 to get 16-bit resolution.

action.sat	Number (0..255)	Color saturation there 0 means no color at all and 255 is the greatest saturation of the color.
action.ct	Number (153..500)	Mired color temperature. (2000K - 6500K)
action.xy	Array	CIE xy color space coordinates as array [x, y] of real values (0..1).
action.effect	String	Dynamic effect: ● none - no effect ● colorloop
devicemembership	Array	A list of device ids (sensors) if this group was created by a device.
etag	String	HTTP etag which changes on any action to the group.
hidden	Bool	Indicates the hidden status of the group. Has no effect at the gateway but apps can uses this to hide groups.
id	String	The id of the group.
lights	Array	A list of all light ids of this group. Sequence is defined by the gateway.
lightsequence	Array	A list of light ids of this group that can be sorted by the user. Need not to contain all light ids of this group.
mulitdeviceids	Array	A list of light ids of this group that are subsequent ids from multidvices with multiple endpoints like the FLS-PP.

name	String	Name of the group.
scenes	Array	A list of scenes of the group.
state	Number	Deprecated - will be removed in future.

Possible errors

304 Not Modified

403 Forbidden

404 Not Found

Set group attributes

`PUT /api/<apikey>/groups/<id>`

Sets attributes of a group which are not related to its state.

Parameters

Field	Type	Description	Required
name	String (0..32)	The name of the group	optional
lights	Array	IDs of the lights which are members of the group.	optional
hidden	Bool	Indicates the hidden status of the group. Has no effect at the gateway but apps can uses this to hide groups.	optional

lightsequence	Array	Specify a sorted list of light ids that can be used in apps.	optional
mulitdeviceids	Array	Append the subsequential light ids of multidevices like the FLS-PP if the app should handle that light differently.	optional

Example request data

```
{
    "name": "Living Room",
    "lights": [ "1", "4" ]
}
```

Response

```
HTTP/1.1 200 OK
ETag: "000bf36b51ef3324446c98hdf6a6ace6"
```

```
[
    { "success": { "/groups/1/name": "Living Room" } },
    { "success": { "/groups/1/lights": [ "1", "4" ] } }
]
```

`Note` In order to add or remove lights to the group the lights must be powered on.

Possible errors

400 Bad Request

403 Forbidden

404 Not Found

503 Service Unavailable

Set group state

`PUT /api/<apikey>/groups/<id>/action`

Sets the state of a group.

Parameters

Field	Type	Description	Required
on	Bool	Set to true to turn the lights on, false to turn them off.	optional
toggle	Bool	Set to true toggles the lights of that group from on to off or vice versa, false has no effect. **Notice:** This setting supersedes the `on` parameter!	optional
bri	Number (0..255)	Set the brightness of the group. Depending on the lights 0 might not mean visible "off" but minimum brightness. If the	optional

		lights are off and the value is greater 0 a on=true shall also be provided.	
hue	Number (0..65535)	Set the color hue of the group. The hue parameter in the HSV color model is between 0°-360° and is mapped to 0..65535 to get 16-bit resolution.	optional
sat	Number (0..255)	Set the color saturation of the group. There 0 means no color at all and 255 is the highest saturation of the color.	optional
ct	Number (153..500)	Set the Mired color temperature of the group. (2000K - 6500K)	optional
xy	Array	Set the CIE xy color space coordinates as array [x, y] of real values (0..1).	optional
alert	String	Trigger a temporary alert effect: ● none - lights are not performing an alert ● select - lights are blinking a short time ● lselect - lights are blinking a longer time	optional
effect	String	Trigger an effect of the group: ● none - no effect ● colorloop - the lights of the group will cycle continously through all colors with the speed specified by colorloopspeed	optional
colorloopspeed	Number (1..255)	Specifies the speed of a colorloop. 1 = very fast, 255 = very slow (default: 15). This parameter only has an effect when it is called together with effect colorloop.	optional

transitiontime	Number	Transition time in 1/10 seconds between two states.	optional

Example request data

```
{
  "on": true,
  "bri": 180,
  "hue": 43680,
  "sat": 255,
  "transitiontime": 10
}
```

Response

```
HTTP/1.1 200 OK
ETag: "030cf8c1c0025420f3a0659afab251f5"
```

```
[
    { "success": { "/groups/1/action/on": true   }},
    { "success": { "/groups/1/action/bri": 180   }},
    { "success": { "/groups/1/action/hue": 43680 }},
    { "success": { "/groups/1/action/sat": 255   }}
]
```

Possible errors

400 Bad Request

403 Forbidden

404 Not Found

503 Service Unavailable

Delete group

```
DELETE /api/<apikey>/groups/<id>
```

Deletes a group.

Parameters

None

Response

```
HTTP/1.1 200 OK
```

```
[ { "success": { "id": "1" } } ]
```

`Note` In order to delete the group and therefore remove all lights from the group the lights must be powered on.

Possible errors

403 Forbidden

404 Not Found

503 Service Unavailable

Lights

Monitor and control single lights.

Get all lights

```
GET /api/<apikey>/lights
```

Returns a list of all lights.

Parameters

None

Response

```
HTTP/1.1 200 OK
```

```
{
    "1": {
        "etag": "026bcfe544ad76c7534e5ca8ed39047c"
        "hascolor": true
        "manufacturer": "dresden elektronik"
        "modelid": "FLS-PP3"
        "name": "Light 1"
        "pointsymbol": {}
        "state": {
            "alert": "none"
            "bri": 111
            "colormode": "ct"
            "ct": 307
            "effect": "none"
            "hue": 7998
            "on": false
            "reachable": true
            "sat": 172
            "xy": [ 0.421253, 0.39921 ]
        }
        "swversion": "020C.201000A0"
        "type": "Extended color light"
        "uniqueid": "00:21:2E:FF:FF:00:73:9F-0A"
    }

    "2": {
        "etag": "026bcfe544ad76c7534e5ca8ed39047c"
        "hascolor": false
        "manufacturer": "dresden elektronik"
        "modelid": "FLS-PP3 White"
        "name": "Light 2"
        "pointsymbol": {}
        "state": {
            "alert": "none"
            "bri": 1
            "effect": "none"
            "on": false
            "reachable": true
        }
        "swversion": "020C.201000A0"
        "type": "Dimmable light"
        "uniqueid": "00:21:2E:FF:FF:00:73:9F-0B"
    }
}
```

Response fields

The whole light object as descripted in Get light state.

Possible errors

403 Forbidden

Get light state

```
GET /api/<apikey>/lights/<id>
```

Returns the full state of a light.

Parameters

None

Response

```
HTTP/1.1 200 OK
```

```
ETag: "030cf8c1c0025420f3a0659afab251f5"
```

```
{
        "etag": "026bcfe544ad76c7534e5ca8ed39047c"
        "hascolor": true
        "manufacturer": "dresden elektronik"
        "modelid": "FLS-PP3"
        "name": "Light 1"
        "pointsymbol": {}
        "state": {
            "alert": "none"
            "bri": 111
            "colormode": "ct"
            "ct": 307
            "effect": "none"
            "hue": 7998
            "on": false
            "reachable": true
            "sat": 172
            "xy": [ 0.421253, 0.39921 ]
        }
        "swversion": "020C.201000A0"
        "type": "Extended color light"
        "uniqueid": "00:21:2E:FF:FF:00:73:9F-0A"
    }
```

Response fields

Field	Type	Description
etag	String	HTTP etag which changes on any action to the light.
hascolor	bool	Indicates if the light can change color. Deprecated - use state instead: if light has no color colormode, hue and xy will not be shown.
manufacturer	String	The manufacturer of the light device.
name	String	Name of a light.
modelid	String	An identifier unique to the product.

pointsymbol	Object	Not used in the current version.
swversion	String	Firmware version.
type	String	Human readable type of the light.
state	Object	The current state of the light.
state.on	Bool	true if the light is on.
state.bri	Number (0..255)	Brightness of the light. Depending on the light type 0 might not mean visible "off" but minimum brightness.
state.hue	Number (0..65535)	Color hue of the light. The hue parameter in the HSV color model is between 0Â°-360Â° and is mapped to 0..65535 to get 16-bit resolution.
state.sat	Number (0..255)	Color saturation of the light. There 0 means no color at all and 255 is the greatest saturation of the color.
state.ct	Number (153..500)	Mired color temperature of the light. (2000K - 6500K)
state.xy	Array	CIE xy color space coordinates as array [x, y] of real values (0..1).
state.alert	String	Temporary alert effect. Following values are possible: ● none - light is not performing an alert ● select - light is blinking a short time ● lselect - light is blinking a longer time

state.colormode	String	The current color mode of the light: ● hs - hue and saturation ● xy - CIE xy values ● ct - color temperature
state.effect	String	Effect of the light: ● none - no effect ● colorloop
state.reachable	Bool	true if the light is reachable and accepts commands.
uniqueid	String	The unique id of the light. It consists of the MAC address of the light followed by a dash and an unique endpoint identifier in the range 01 to FF.

Possible errors

304 Not Modified

403 Forbidden

404 Not Found

Set light state

`PUT /api/<apikey>/lights/<id>/state`

Sets the state of a light.

Parameters

Field	**Type**	**Description**	**Required**
on	Bool	Set to true to turn the light on, false to turn it off.	optional
bri	Number (0..255)	Set the brightness of the light. Depending on the light type 0 might not mean visible "off" but minimum brightness. If the light is off and the value is greater 0 a on=true shall also be provided.	optional
hue	Number (0..65535)	Set the color hue of the light. The hue parameter in the HSV color model is between 0Â°-360Â° and is mapped to 0..65535 to get 16-bit resolution.	optional
sat	Number (0..255)	Set the color saturation of the light. There 0 means no color at all and 255 is the greatest saturation of the color.	optional
ct	Number (153..500)	Set the Mired color temperature of the light. (2000K - 6500K)	optional
xy	Array	Set the CIE xy color space coordinates as array [x, y] of real values (0..1).	optional
alert	String	Trigger a temporary alert effect: ● none - light is not performing an alert ● select - light is blinking a short time ● lselect - light is blinking a longer time	optional
effect	String	Trigger an effect of the light: ● none - no effect	optional

		• colorloop - the light will cycle continously through all colors with the speed specified by colorloopspeed	
colorloopspeed	Number (1..255)	Specifies the speed of a colorloop. 1 = very fast, 255 = very slow (default: 15). This parameter only has an effect when it is called together with effect colorloop.	optional
transitiontime	Number	Transition time in 1/10 seconds between two states.	optional

Example request data

```
{
  "on": true,
  "bri": 180,
  "hue": 43680,
  "sat": 255,
  "transitiontime": 10
}
```

Response

```
HTTP/1.1 200 OK
ETag: "030cf8c1c0025420f3a0659afab251f5"
```

```
[
    { "success": { "/lights/1/state/on": true   }},
    { "success": { "/lights/1/state/bri": 180   }},
    { "success": { "/lights/1/state/hue": 43680 }},
    { "success": { "/lights/1/state/sat": 255   }}
]
```

Possible errors

400 Bad Request

403 Forbidden

404 Not Found

Set light attributes

```
PUT /api/<apikey>/lights/<id>
```

Sets attributes of a light which are not related to its state.

Parameters

Field	Type	Description	Required
name	String (0..32)	Set the name of the light.	required

Example request data

```
{ "name": "Living Room 1" }
```

Response

```
HTTP/1.1 200 OK
ETag: "030cf8c1c0025420f3a0659afab251f5"
```

```
[{ "success": { "/lights/1/name": "Living Room 1"}}]
```

Possible errors

400 Bad Request

403 Forbidden

404 Not Found

Delete light

`DELETE /api/<apikey>/lights/<id>`

Removes the light from the gateway. It will not be shown in any rest api call. Also deletes all groups and scenes on the light device.

Parameters

Field	Type	Description	Required
reset	Bool	If true sends a network leave command to the light device (may not supported by each manufacturer).	optional

Response

```
HTTP/1.1 200 OK
ETag: "030cf8c1c0025420f3a0659afab251f5"
```

```
[{ "success": { "id": "1"}}]
```

Possible errors

400 Bad Request

404 Not Found

Remove all groups

`DELETE /api/<apikey>/lights/<id>/groups`

Remove the light from all groups it is a member of.

Response

```
HTTP/1.1 200 OK
ETag: "030cf8c1c0025420f3a0659afab251f5"
```

```
[{ "success": { "id": "1"}}]
```

Possible errors

404 Not Found

Remove all scenes

```
DELETE /api/<apikey>/lights/<id>/scenes
```

Remove the light from all scenes it is a member of.

Response

```
HTTP/1.1 200 OK
ETag: "030cf8c1c0025420f3a0659afab251f5"
```

```
[{ "success": { "id": "1"}}]
```

Possible errors

404 Not Found

Rules

Rules provide the ability to trigger actions of lights or groups when a specific sensor condition is met.

Create rule

`POST /api/<apikey>/rules`

Creates a new rule.

Parameters

Field	Type	Description	Required
actions	Array(action) (1..8)	An array of actions that will happen when the rule triggers.	required
action.address	String	path to a light, group or scene resource	required
action.body	Object	Parameters that will be send to the resource formated as JSON.	required
action.method	String	Can be **PUT**, **POST**, **DELETE** (currently only used for green power devices) or **BIND** which will create a ZigBee binding between a sensor and a light or group.	required

conditions	Array (condition) (1..8)	The conditions that must be met to trigger a rule.	required
condition.address	String	path to a sensor resource and the related state	required
condition.operator	String	**eq**, **gt**, **lt**, **dx** (equals, greater than, lower than, on change).	required
condition.value	String	The value the operator is compared with. Will be casted automatically to the corresponding data type.	required
name	String	The name of the rule.	required
periodic	Number	Specifies if the rule should trigger periodically. 0 = trigger on event; >0 = time in ms the rule will be triggered periodically. Default is 0.	optional
status	String ("enabled" | "disabled")	The status of the rule. Default is enabled.	optional

Notes for using the action method BIND

To create ZigBee bindings between a sensor and a light or group use the BIND method. The rules condition specifies which ZigBee cluster will be used.

Body	**Cluster**
"on": true	On/Off cluster

"bri": 1	Level cluster (brightness control)
"scene": "S1"	Scenes cluster

Currently creating a binding is only supported for the sensor type *ZHASwitch* and the state *buttonevent*. For the BIND method the condition value must be set to the ZigBee endpoint which contains the cluster. The endpoint number should be taken from the sensors *ep* field.

Example request data

```
{
    "actions": [
        {
            "address": "/groups/0/action",
            "body": {
                "on": true
            },
            "method": "BIND"
        }
    ],
    "conditions": [
        {
            "address": "/sensors/1/state/buttonevent",
            "operator": "eq",
            "value": "1"
        }
    ],
    "name": "Switch button 1 all lights On/Off"
}
```

This will create a binding between a switch and the On/Off Cluster of all Lights of the group 0.

Response

```
HTTP/1.1 200 OK
```

```
[ { "success": { "id": "1" } } ];
```

Response fields

Field	Type	Description
id	String	The unique identifier of the new rule.

Possible errors

400 Bad Request

403 Forbidden

Get all Rules

```
GET /api/<apikey>/rules
```

Returns a list of all rules. If there are no rules in the system then an empty object {} will be returned.

Parameters

None

Response

```
HTTP/1.1 200 OK
```

```
{
    "1": {
        "actions": [
            {
                "address": "/lights/1/state",
                "body": {
                    "on": true
                },
                "method": "BIND"
            }
        ],
        "conditions": [
            {
                "address": "/sensors/2/state/buttonevent",
                "operator": "eq",
                "value": "4"
            }
        ],
        "created": "2016-07-04T14:17:12",
        "etag": "9bd1fcc627001458ea88c8742e61c692",
        "lasttriggered": "none",
        "name": "Sensor: 2 EP:4 On/Off",
        "owner": "AD4F14F244",
        "periodic": 0,
        "status": "enabled",
        "timestriggered": 0
    },
    "2": {
        "actions": [
            {
                "address": "/groups/0/action",
                "body": {
                    "on": false
                },
                "method": "PUT"
            }
        ],
        "conditions": [
            {
                "address": "/sensors/5/state/buttonevent",
                "operator": "eq",
                "value": "34"
            },
            {
                "address": "/sensors/5/state/lastupdated",
                "operator": "dx"
            }
        ],
        "created": "2016-07-05T13:36:52",
        "etag": "0fb118418fa77116052f74fb129a648b",
        "lasttriggered": "none",
        "name": "0x0000000000402483[Rule1]",
        "owner": "AD4F14F244",
        "periodic": 0,
        "status": "enabled",
        "timestriggered": 0
    }
}
```

Response fields

The whole rule object as descripted in Get rule.

Possible errors

403 Forbidden

Get rule

```
GET /api/<apikey>/rules/<id>
```

Returns the rule with the specified id.

Parameters

None

Response

```
HTTP/1.1 200 OK
ETag: "030cf8c1c0025420f3a0659afab251f5"
```

```
{
    "actions": [
        {
            "address": "/lights/1/state",
            "body": {
                "on": true
            },
            "method": "BIND"
        }
    ],
    "conditions": [
        {
            "address": "/sensors/2/state/buttonevent",
            "operator": "eq",
            "value": "4"
        }
    ],
    "created": "2016-07-04T14:17:12",
    "etag": "9bd1fcc627001458ea88c8742e61c692",
    "lasttriggered": "none",
    "name": "Sensor: 2 EP:4 On/Off",
    "owner": "AD4F14F244",
    "periodic": 0,
    "status": "enabled",
    "timestriggered": 0
}
```

Response fields

Field	Type	Description
actions	Array (action)	An array of actions that will happen when the rule triggers.
action.address	String	path to a light, group or scene resource
action.body	Object	Parameters that will be send to the resource formated as JSON.
action.method	String	Can be "PUT","POST","DELETE" (currently only used for green power devices) or "BIND" which will create a binding between a sensor and a light or group.
conditions	Array (condition)	The conditions that must be met to trigger a rule.
condition.address	String	path to a sensor resource
condition.operator	String	eq, gt, lt, dx (equals, greater than, lower than, on change).
condition.value	String	The value the operator is compared with. Will be casted automatically to the corresponding data type.
created	String	Timestamp when the rule was created.
etag	String	HTTP etag which changes whenever the rule is changed.

lasttriggered	String	Timestamp when the rule was last triggered.
name	String	The name of the rule.
owner	String	The owner of the rule.
periodic	Number	Specifies if the rule should trigger periodically. 0 = trigger on event; >0 = time in ms the rule will be triggered periodically.
status	String ("enabled" | "disabled")	The status of the rule.
timestriggered	Number	Times the rule was triggered.

Possible errors

403 Forbidden

404 Not Found

Update rule

```
PUT /api/<apikey>/rules/<id>/
```

Update a rule with the specified parameters.

Parameters

Field	Type	Description	Required

actions	Array (action) (1..8)	An array of actions that will happen when the rule triggers.	optional
conditions	Array (condition) (1..8)	The conditions that must be met to trigger a rule.	optional
name	String	The name of the rule.	optional
periodic	Number	Specifies if the rule should trigger periodically. 0 = trigger on event; >0 = time in ms the rule will be triggered periodically. Default is 0.	optional
status	String ("enabled" | "disabled")	The status of the rule. Default is enabled.	optional

Example request data

```
{
  "actions": [
        {
            "address": "/lights/1/state",
            "body": {
                "bri": 1
            },
            "method": "BIND"
        }
    ]
}
```

Response

```
HTTP/1.1 200 OK
ETag: "030cf8c1c0025420f3a0659afab251f5"
```

```
[
    {
        "success": { "/rules/1/actions":
            {
                "address": "/lights/1/state",
                "body": {
                    "bri": 1
                },
                "method": "BIND"
            }
        }
    }
]
```

Possible errors

400 Bad Request

403 Forbidden

Delete rule

```
DELETE /api/<apikey>/ruless/<id>
```

Delete a rule.

Parameters

None

Response

```
HTTP/1.1 200 OK
ETag: "030cf8c1c0025420f3a0659afab251f5"
```

```
[{ "success": "1"}]
```

Possible errors

400 Bad Request

403 Forbidden

404 Not Found

Scenes

Scenes provide an easy and performant way to recall often used states to a group.

Create scene

```
POST /api/<apikey>/groups/<group_id>/scenes
```

Creates a new scene for a group. The actual state of each light will become the lights scene state.

Parameters

Field	Type	Description	Required
name	String	The name of the new scene	required

Example request data

```
{ "name": "Garage" }
```

Response

```
HTTP/1.1 200 OK
```

```
[ { "success": { "id": "3" } } ];
```

Response fields

Field	Type	Description

id	String	The unique identifier of the scene.

`Note` Creating a scene with a name which already exists will not create a new scene or fail. Such a call will only return the id of the existing scene and store the current state of all lights.

Possible errors

400 Bad Request

403 Forbidden

404 Not Found

503 Service Unavailable

Get all scenes

```
GET /api/<apikey>/groups/<group_id>/scenes
```

Returns a list of all scenes of a group.

Parameters

None

Response

```
HTTP/1.1 200 OK
Etag: 203941fel3ds8ad61903224
```

```
{
    "1": {
        "lights": ["1","2"],
        "name": "working"
    },
    "2": {
        "lights": ["3"],
        "name": "reading"
    }
}
```

Response fields

Field	Type	Description
lights	Array	Lights which are members of the scene.
name	String	Name of the scene.

Possible errors

403 Forbidden

404 Not Found

Get scene attributes

`GET /api/<apikey>/groups/<group_id>/scenes/<scene_id>`

Returns all attributes of a scene.

Parameters

None

Response

```
HTTP/1.1 200 OK
Etag: 0b32030b31ef30a4446c9adff6a6f9e5
```

```
{
    lights": [
        {
            "bri": 111
            "id": "3"
            "on": false
            "transitiontime": 0
            "x": 27499
            "y": 26060
        }
    ],
    "name": "reading"
    "state": 0
}
```

Response fields

Field	Type	Description
lights	Array	Contains objects which describe the state fof each light in the scene.
lights[].id	String	The id of the light.
lights[].on	Bool	True if the light is on.
lights[].bri	Number (0..255)	The brightness of the light.
lights[].transitiontime	Number	The scene fading transition time in 1/10 seconds.
lights[].x	Number (0..1)	The color x value of the light.
lights[].y	Number (0..1)	The color y value of the light.
lights[].ct	Number	The mired color temperature value of the light.
lights[].hue	Number (0.65535)	The hue value of the light.
lights[].sat	Number (0.255)	The saturation value of the light.

name	String	Name of the scene.
state	Number	Deprecated - will be removed in future.

Possible errors

403 Forbidden

404 Not Found

Set scene attributes

`PUT /api/<apikey>/groups/<group_id>/scenes/<scene_id>`

Sets attributes of a scene.

Parameters

Field	Type	Description	Required
name	String	Name of the scene.	optional

Example request data

```
{
  "name": "working"
}
```

Response

```
HTTP/1.1 200 OK
Etag: 030cf8c1c0025420f3a0659afab251f5
```

```
[ { "success": { "/groups/1/scenes/1/name": "working" } } ]
```

Possible errors

400 Bad Request

403 Forbidden

404 Not Found

Store scene

```
PUT /api/<apikey>/groups/<group_id>/scenes/<scene_id>/store
```

Stores the current group state in the scene. The actual state of each light in the group will become the lights scene state.

Parameters

None

Response

```
HTTP/1.1 200 OK
```

```
[ { "success": { "id": "3" } } ]
```

Response fields

Field	Type	Description
id	String	The unique identifier of the scene.

Possible errors

400 Bad Request

403 Forbidden

404 Not Found

503 Service Unavailable

Recall scene

```
PUT /api/<apikey>/groups/<group_id>/scenes/<scene_id>/recall
```

Recalls a scene. The actual state of each light in the group will become the lights scene state stored in each light.

`Note` Lights which are not reachable (turned off) won’t be affected!

Parameters

None

Response

```
HTTP/1.1 200 OK
```

```
[ { "success": { "id": "3" } } ]
```

Response fields

Field	Type	Description
id	String	The unique identifier of the scene.

Possible errors

400 Bad Request

403 Forbidden

404 Not Found

503 Service Unavailable

Modify scene

`PUT /api/<apikey>/groups/<group_id>/scenes/<scene_id>/lights/<light_id>/state`

Modifies the state of a light of the scene.

`Note` The light must be a member of the scene.

Example request data

```
{
    "bri": 111
    "on": true
    "transitiontime": 10
    "xy": [ 0.44, 0.98 ]
}
```

Parameters

Field	Type	Description	Required
bri	Number (0..255)	Brightness of the light	optional
on	Bool	On/off status of the light	optional

transitiontime	Number	Transitiontime of the light when the scene is called in 1/10 seconds	optional
xy	Array	Xy color values of the light mapped to [0..1]	optional

Response

```
HTTP/1.1 200 OK
```

```
[ { "success": { "id": "1" } } ]
```

Possible errors

400 Bad Request

403 Forbidden

404 Not Found

503 Service Unavailable

Delete scene

```
DELETE /api/<apikey>/groups/<group_id>/scenes/<scene_id>
```

Deletes a scene.

Parameters

None

Response

```
HTTP/1.1 200 OK
```

```
[ { "success": { "id": "3" } } ]
```

Response fields

Field	Type	Description
id	String	The unique identifier of the scene.

Possible errors

403 Forbidden

404 Not Found

503 Service Unavailable

Schedules

Schedules provide the ability to trigger timed commands to groups or lights.

Create schedule

`POST /api/<apikey>/schedules`

Creates a new schedule.

Parameters

Field	Type	Description	Required
name	String	The name of the new schedule. If the name already exists a number will be appended.	optional
description	String	The description of the schedule.	optional
command	Object	The command to execute when the schedule triggers.	required
command.address	String	The address of a light or group ressource	required
command.method	String	must be "PUT"	required
command.body	Object	The state that the light or group will activate when the schedule triggers	required

status	String ("enabled" | "disabled")	Whether the schedule is enabled or disabled. Default is enabled.	optional
autodelete	Bool	If true the schedule will be deleted after triggered. Else it will be disabled. Default is true.	optional
time	String	Time when the schedule shall trigger in UTC ISO 8601:2004 format. ● specific date: "yyyy-MM-ddThh:mm:ss" ● repeated days: "W[0..127]/Thh:mm:ss" ● timer: "PThh:mm:ss" ● recurring timer: "R[0..99]/PThh:mm:ss" Repeated days use a bitmap to determine on which day of the week the alarm should	required

		trigger. The Format is: 0MTWTFSS. Example: 01111100 = 124 is weekdays, 00000011 = 3 is weekend. The number after R of recurring timer determine the number of repetitions of the timer. Not specifying a number means infinity.	

Example request data

```
{
    "name": "blue moon",
    "description": "Turns all lights blue",
    "command": {
        "address": "/api/8918fbad2100nag17ca1/groups/5/action",
        "method": "PUT",
        "body": { "on": true, "hue": 43000, "sat": 255 }
    },
    "time": "2013-07-29T09:30:00"
}
```

`Note` The address in the command object must contain a valid API key.

Response

```
HTTP/1.1 200 OK
```

```
[ { "success": { "id": "3" } } ];
```

Response fields

Field	Type	Description
id	String	The unique identifier of the new schedule.

Possible errors

400 Bad Request

403 Forbidden

Get all schedules

```
GET /api/<apikey>/schedules
```

Returns a list of all schedules.

Parameters

None

Response

```
HTTP/1.1 200 OK
Etag: 203941fel3ds8ad61903224
```

```
{
    "1": {
        "autodelete": false,
        "command": {
            "address": "/api/8918fbad2100nag17ca1/groups/2/action",
            "method": "PUT",
            "body": { "on": false }
        },
        "description": "Turns all lights off",
        "etag": "4e100d1c4e3497154a77bc0865c89030",
        "name": "turn all off",
        "status": "enabled",
        "time": "2013-07-30T20:10:00"
    },
    "2": {
        "autodelete": false,
        "command": {
            "address": "/api/AD4F14F244/groups/4/scenes/1/recall"
            "body": {}
            "method": "PUT"
        },
        "description": "",
        "etag": "4e100d1c4e3497154a77bc0865c89030",
        "name": "call scene",
        "status": "enabled",
        "time": "W120/T10:00:00"
    }
}
```

Response fields

The full schedule object as in Get schedule attributs.

Possible errors

403 Forbidden

Get schedule attributes

```
GET /api/<apikey>/schedules/<id>
```

Returns all attributes of a schedule.

Parameters

None

Response

```
HTTP/1.1 200 OK
Etag: 0b32030b31ef30a4446c9adff6a6f9e5
```

```
    {
        "autodelete": false,
        "command": {
            "address": "/api/8918fbad2100nag17ca1/groups/2/action",
            "method": "PUT",
            "body": { "on": false }
        },
        "description": "Turns all lights off",
        "etag": "4e100d1c4e3497154a77bc0865c89030",
        "name": "turn all off",
        "status": "enabled",
        "time": "2013-07-30T20:10:00"
    }
```

Response fields

Field	Type	Description	
autodelete	Bool	If set to true the schedule will be deleted after trigger. Else it will be disabled.	
command	Object	The command to execute when the schedule triggers.	
command.address	String	The address of a light or group ressource	required
command.method	String	must be "PUT"	required
command.body	Object	The state that the light or group will activate when the schedule triggers	required
description	String	The description of the schedule.	
etag	String	The etag of the schedule.	
name	String	Name of the schedule.	
status	String	The status of the schedule (enabled or disabled).	

time	String	Time when the schedule shall trigger in UTC ISO 8601:2004 format. • specific date: "yyyy-MM-ddThh:mm:ss" • repeated days: "W[0..127]/Thh:mm:ss" • timer: "PThh:mm:ss" • recurring timer: "R[0..99]/PThh:mm:ss" Repeated days use a bitmap to determine on which day of the week the alarm should trigger. The Format is: 0MTWTFSS. Example: 01111100 = 124 is weekdays, 00000011 = 3 is weekend. The number after R of recurring timer determine the number of repetitions of the timer. Not specifying a number means infinity.

Possible errors

403 Forbidden

404 Not Found

Set schedule attributes

`PUT /api/<apikey>/schedules/<id>`

Sets attributes of a schedule.

Parameters

Field	Type	Description	Required

name	String	The name of the new schedule. If the name already exists a number will be appended.	optional
description	String	The description of the schedule.	optional
command	Object	The command to execute when the schedule triggers.	optional
command.address	String	The address of a light or group ressource	optional
command.method	String	must be "PUT"	optional
command.body	Object	The state that the light or group will activate when the schedule triggers	optional
status	String ("enabled"|"disabled")	Whether the schedule is enabled or disabled. Default is enabled.	optional
autodelete	Bool	If true the schedule will be deleted after triggered. Else it will be disabled. Default is true.	optional

time	String	Time when the schedule shall trigger in UTC ISO 8601:2004 format. The time must be in the future. ● specific date: "yyyy-MM-ddThh:mm:ss" ● repeated days: "W[0..127]/Thh:mm:ss" ● timer: "PThh:mm:ss" ● recurring timer: "R[0..99]/PThh:mm:ss" Repeated days use a bitmap to determine on which day of the week the alarm should trigger. The Format is: 0MTWTFSS. Example: 01111100 = 124 is weekdays, 00000011 = 3 is weekend. The number after R of recurring timer determine the number of repetitions of the timer. Not specifying a number means infinity.	optional

Example request data

```
{
    "name": "working"
}
```

Response

```
HTTP/1.1 200 OK
Etag: 030cf8c1c0025420f3a0659afab251f5
```

```
[ { "success": { "/schedules/1/name": "working" } } ]
```

Possible errors

400 Bad Request

403 Forbidden

404 Not Found

Delete schedule

`DELETE /api/<apikey>/schedules/<id>`
Deletes a schedule.

Parameters

None

Response

```
HTTP/1.1 200 OK
```

```
[ { "success": { "id": "3" } } ]
```

Response fields

Field	Type	Description
id	String	The unique identifier of the schedule.

Possible errors

403 Forbidden

404 Not Found

Sensors

Sensors can be used to measure environment parameters like brightness or activation of a switch. With a coressponding rule they can control lights and groups.

Create sensor

`POST /api/<apikey>/sensors`

Creates a new sensor.

Parameters

Field	Type	Description	Required
name	String	The name of the sensor.	required
modelid	String	The model identifier of the sensor.	required
swversion	String	The software version of the sensor.	required
type	String	The type of the sensor (see: allowed sensor types and its states).	required
uniqueid	String	The unique id of the sensor. Should be the MAC address of the device.	required
manufacturername	String	The manufacturer name of the sensor.	required
state	Object	The state of the sensor (see: supported sensor types and its states).	optional

config	Object	The config of the sensor. • on - Bool - default: true • reachable - Bool - default: true • battery - Number (0..100)	optional

Supported sensor types and its states

Sensor type	**Supported state**	**Type**
ZHASwitch	buttonevent	Number
ZHALight	lux	Number
ZHAPresence	presence	Bool

Example request data

```
{
    "config": {
        "on": true
        "reachable": true
    }
    "manufacturername": "Me"
    "modelid": "T1000"
    "name": "My Switch"
    "swversion": "1.0"
    "type": "CLIPSwitch"
    "uniqueid": "0x001fee00000008bb"
}
```

Response

```
HTTP/1.1 200 OK
```

```
[ { "success": { "id": "1" } } ];
```

Response fields

Field	Type	Description
id	String	The unique identifier of the new sensor.

Possible errors

400 Bad Request

403 Forbidden

Get all Sensors

```
GET /api/<apikey>/sensors
```

Returns a list of all Sensors. If there are no sensors in the system then an empty object {} will be returned.

Parameters

None

Response

```
HTTP/1.1 200 OK
```

```
{
    "1": {
        "config": {
            "on": true,
            "reachable": true
        },
        "ep": 1,
        "etag": "61eaee2477fc3d5c27932fefeef638bd",
        "manufacturername": "dresden elektronik",
        "modelid": "Lighting Switch",
        "name": "Lighting Switch 1",
        "state": {
            "lastupdated": "2016-07-06T09:39:53"
        },
        "swversion": "1.0",
        "type": "ZHASwitch",
        "uniqueid": "0x00212effff00a6bc"
    }
    "2": {
        "config": {
            "on": true,
            "reachable": true
        },
        "ep": 2,
        "etag": "61eaee2477fc3d5c27932fefeef638bd",
        "manufacturername": "dresden elektronik",
        "modelid": "Lighting Switch",
        "name": "Lighting Switch 2",
        "state": {
            "lastupdated": "2016-07-06T09:39:53"
        },
        "swversion": "1.0",
        "type": "ZHASwitch",
        "uniqueid": "0x00212effff00a6bc"
    }
}
```

Response fields

The whole sensor object as descripted in Get sensor.

Possible errors

403 Forbidden

Get sensor

```
GET /api/<apikey>/sensors/<id>
```

Returns the sensor with the specified id.

Parameters

None

Response

```
HTTP/1.1 200 OK
ETag: "030cf8c1c0025420f3a0659afab251f5"
```

```
{
    "config": {
        "on": true,
        "reachable": true
    },
    "ep": 1,
    "etag": "61eaee2477fc3d5c27932fefeef638bd",
    "manufacturername": "dresden elektronik",
    "mode": 2,
    "modelid": "Lighting Switch",
    "name": "Lighting Switch 1",
    "state": {
        "lastupdated": "2016-07-06T09:39:53"
    },
    "swversion": "1.0",
    "type": "ZHASwitch",
    "uniqueid": "0x00212effff00a6bc"
}
```

Response fields

Field	Type	Description
config	Object	The config of the sensor.
config.on	Bool	Specifies if the sensor is on or off.
config.reachable	Bool	Specifies if the sensor is reachable.
config.battery	Number (0..100)	The battery status of the sensor.
ep	Number	The Endpoint of the sensor.

etag	String	HTTP etag which changes whenever the sensor changes.
manufacturername	String	The manufacturer name of the sensor.
modelid	String	The model id of the sensor.
mode	Number (1|2|3)	The mode of the sensor (Only available for dresden elektroink Lighting Switch). ● 1 = Scenes mode ● 2 = Two groups mode ● 3 = Color temperature mode
name	String	The name of the sensor..
state	Object	The state of the sensor.
state.lastupdated	String	Timestamp when the sensor was last updated.
swversion	String	Software version of the sensor.
type	String	The type of the sensor.
uniqueid	String	The unique identifiers (MAC address) of the sensor.

Possible errors

403 Forbidden

404 Not Found

Update sensor

`PUT /api/<apikey>/sensors/<id>`

Update a sensor with the specified parameters.

Parameters

Field	Type	Description	Required
name	String	The name of the sensor.	optional
mode	Number (1|2|3)	Only availabe for dresden elektronik Lighting Switch. Set the mode of the switch. ● 1 = Scenes mode ● 2 = Two groups mode ● 3 = Color temperature mode	optional

Example request data

```
{
  "name": "a nice name"
}
```

Response

```
HTTP/1.1 200 OK
ETag: "030cf8c1c0025420f3a0659afab251f5"
```

```
    { "success": { "/sensors/1/name": "a nice name" } }
```

Possible errors

400 Bad Request

403 Forbidden

Change sensor config

```
PUT /api/<apikey>/sensors/<id>/config
```

Update a sensor config with the specified parameters.

Parameters

Field	Type	Description	Required
on	Bool	The on/off status of the sensor.	optional
reachable	Bool	The reachable status of the sensor.	optional
battery	Number (1..100)	The current battery state in percent, only for battery powered devices.	optional

Example request data

```
{
  "on": false,
  "reachable: false
}
```

Response

```
HTTP/1.1 200 OK
ETag: "030cf8c1c0025420f3a0659afab251f5"
```

```
    {
      "success": { "/sensors/1/config/on": false }
      "success": { "/sensors/1/config/reachable": false }
    }
```

Possible errors

400 Bad Request

403 Forbidden

Change sensor state

`PUT /api/<apikey>/sensors/<id>/state`

Update a sensor state with the specified parameters.

Parameters

Allowed sensor types and its states:

Sensor type	**Allowed state**	**type**
CLIPSwitch	buttonevent	Number
CLIPOpenClose	open	Bool
CLIPPresence	presence	Bool
CLIPTemperature	temperature	Number
CLIPGenericFlag	flag	Bool

CLIPGenericStatus	status	Number
CLIPHumidity	humidity	Number

Example request data

```
{
  "flag": false
}
```

Response

```
HTTP/1.1 200 OK
ETag: "030cf8c1c0025420f3a0659afab251f5"
```

```
    {
      "success": { "/sensors/1/state/flag": false }
    }
```

Possible errors

400 Bad Request

403 Forbidden

Delete sensor

```
DELETE /api/<apikey>/sensors/<id>
```

Delete a sensor.

Parameters

None

Response

```
HTTP/1.1 200 OK
ETag: "030cf8c1c0025420f3a0659afab251f5"
```

```
[{ "success": "1"}]
```

Possible errors

400 Bad Request

403 Forbidden

404 Not Found

Touchlink

The touchlink endpoint allows to communicate with near by located devices.

Scan for devices

`POST /api/<apikey>/touchlink/scan`

Starts scanning on all channels for devices which are located close to the gateway. The whole scan process will take about 10 seconds.

`Note` While scanning is in progress further API requests which require network access aren't allowed.

Parameters

None

Response

`HTTP/1.1 200 OK`

Possible errors

403 Forbidden

503 Service Unavailable

Get scan results

`GET /api/<apikey>/touchlink/scan`

Returns the results of a touchlink scan.

Parameters

None

Response

```
HTTP/1.1 200 OK
```

```
{
    "scanstate": "scanning",
    "lastscan": "2013-11-05T08:14:12",
    "result": {
        "1": {
            "factorynew": true,
            "address": "0x0017880100bfbfed"
        },
        "2": {
            "factorynew": false,
            "address": "0x001788010022b40a"
        }
     },
}
```

Response fields

Field	Type	Description
scanstate	String	State of a scan request: • idle - scan is finished or was not started • scanning - scan is in progress
lastscan	String	UTC time of the last scan in ISO 8601 format.
result	Object	A list of all devices which where found during the scan.

Possible errors

403 Forbidden

Identify a device

```
POST /api/<apikey>/touchlink/<id>/identify
```

Puts a device into identify mode for example a light will blink a few times.

`Note` *id* must be one of the indentifiers which are returned in the scan result.

Parameters

None

Response

```
HTTP/1.1 200 OK
```

Possible errors

403 Forbidden

404 Not Found

503 Service Unavailable

Reset a device

```
POST /api/<apikey>/touchlink/<id>/reset
```

Send a reset to factory new request to a device.

`Note` *id* must be one of the indentifiers which are returned in the scan result.

Parameters

None

Response

```
HTTP/1.1 200 OK
```

Possible errors

403 Forbidden

404 Not Found

503 Service Unavailable

Websocket

The embedded Websocket server provides push notifications to applications which require real-time feedback from devices like lights, groups, switches, and sensors.

`since` version 2.04.40

Websocket Configuration

The Websocket server is started on an unused proxy friendly port which, depending on the system, is either 443, 8080, 8088, 20877, or any other unused random port.

The Websocket server can be configured to include all `state` or `config` attributes in the message, or only the changed attributes.

The Websocket used port and setting are listed in the configuration API endpoint:

```
GET /api/<apikey>/config
```

Parameters

None

Response

```
HTTP/1.1 200 OK
```

```
{
    ...
    "websocketnotifyall": true,
    "websocketport": 8088,
    ...
}
```

Possible errors

403 Forbidden

503 Service Unavailable

Open Connection

How to establish a connection to a Websocket server depends on the underlying programming environment.

Javascript example

The following example demonstrates how to establish a connection with Javascript in a browser or NodeJS implementation.

```
const WebSocket = require('ws');

const host = '192.168.1.202';
const port = 8088;

const ws = new WebSocket('ws://' + host + ':' + port);

ws.onmessage = function(msg) {
    console.log(JSON.parse(msg.data));
}
```

Message Format

Messages received over a Websocket connection contain data in JSON format.

Light state change example

```
{
    "e": "changed",
    "id": "1",
    "r": "lights",
    "state": {
        "bri": 1,
        "on": true,
        "x": 65279,
        "xy": [
            0.9961,
            0.9961
        ],
        "y": 65279
    },
    "t": "event",
    "uniqueid": "00:0b:57:ff:fe:9a:46:ab-01"
}
```

Note that `x` and `y` are included in the `state` for backwards compatibility. New apps should use `xy` instead.

Group state change example

```
{
    "e": "changed",
    "id": "1",
    "r": "groups",
    "state": {
        "all_on": true,
        "any_on": true
    },
    "t": "event"
}
```

Sensor button event example

```
{
    "e": "changed",
    "id": "5",
    "r": "sensors",
    "state": {
        "buttonevent": 2002,
        "lastupdated": "2019-03-15T20:16:30"
    },
    "t": "event",
    "uniqueid": "00:0d:6f:00:10:65:8a:6e-01-1000"
}
```

Sensor name change example

```
{
    "e": "changed",
    "id": "10",
    "name": "Pulse 2",
    "r": "sensors",
    "t": "event",
    "uniqueid": "00:0d:6f:00:10:65:8a:6e-01-1000"
}
```

Sensor added example

```
{
    "e": "added",
    "id": "10",
    "r": "sensors",
    "sensor": {
        "config": {
            "battery": null,
            "on": true,
            "reachable": true
        },
        "ep": 1,
        "etag": "7088b28f8a8a2c786e6e48d95c547fa4",
        "id": "10",
        "manufacturername": "icasa",
        "mode": 1,
        "modelid": "ICZB-KPD12",
        "name": "ICZB-KPD12 10",
        "state": {
            "buttonevent": null,
            "lastupdated": "none"
        },
        "type": "ZHASwitch",
        "uniqueid": "00:0d:6f:00:10:65:8a:6e-01-1000"
    },
    "t": "event",
    "uniqueid": "00:0d:6f:00:10:65:8a:6e-01-1000"
}
```

Scene Recall example

```
{
    "e": "scene-called",
    "gid": "0",
    "r": "scenes",
    "scid": "2",
    "t": "event"
}
```

Message fields

Field	Type	Description
`t`	String	The **type** of the message:

		<ul><li>`event` - the message holds an event.</li></ul>
`e`	String	The **event type** of the message:<ul><li>`added` - resource has been added;</li><li>`changed` - resource attributes have changed;</li><li>`deleted` - resource has been deleted.</li><li>`scene-called` - a scene has been recalled.</li></ul>
`r`	String	The **resource type** to which the message belongs:<ul><li>`groups` - message relates to a group resource;</li><li>`lights` - message relates to a light resource;</li><li>`scenes` - message relates to a scene under a group resource;</li><li>`sensors` - message relates to a sensor resource.</li></ul>
`id`	String	The id of the resource to which the message relates, e.g. `5` for `/sensors/5`. Not for `scene-called` events.
`uniqueid`	String	The `uniqueid` of the resource to which the message relates, e.g. `00:0d:6f:00:10:65:8a:6e-01-1000`. Only for light and sensor resources.
`gid`	String	The group id of the resource to which the message relates. Only for `scene-called` events.
`scid`	String	The scene id of the resource to which the message relates. Only for `scene-called` events.

\`config\`	Map	Depending on the \`websocketnotifyall\` setting: a map containing all or only the changed \`config\` attributes of a sensor resource. Only for \`changed\` events.
\`name\`	String	The (new) name of a resource. Only for \`changed\` events.
\`state\`	Map	Depending on the \`websocketnotifyall\` setting: a map containing all or only the changed \`state\` attributes of a group, light, or sensor resource. Only for \`changed\` events.
\`group\`	Map	The full group resource. Only for \`added\` events of a group resource.
\`light\`	Map	The full light resource. Only for \`added\` events of a light resource.
\`sensor\`	Map	The full sensor resource. Only for \`added\` events of a sensor resource.

Note that only one of `config`, `name`, or `state` will be present per `changed` event.

Note that the Websocket functionality is still under development. Notably `added` and `deleted` notifications might not be issued under all circumstances.

Polling

Polling state

Since the state of lights and groups might be changed from various devices, client applications shall update their local cache regularly to provide the best user experience.

To keep the processing overhead low in the gateway as well as on the client low the API supports the common HTTP `ETag` and `If-None-Match` headers to prevent full state updates in each polling attempt.

ETag HTTP header

Many API calls return an ETag in the HTTP header. An ETag is a hash string which belongs to a resource and is changed every time the resource is modified.

Ressources are:

- Lights
- Groups
- Configuration

For example the first API call to get the state of light 1 returns an ETag. In a second call the client provides the HTTP header field `If-None-Match` with the latest known ETag of the light.

- If the light meanwhile has changed the request will return the new state and another ETag.
- If the light wasn't changed a HTTP status `304 Not Modified` will be returned with no body content, in this case the client doesn't need to update any data or UI.

Authorization

API keys

Apps which want to access the API must obtain an API key. There are two methods for doing so.

- Unlocking the gateway
- HTTP basic authentification

Unlocking the gateway

Unlocking the gateway for a short period of time allows any app to acquire an API key via configuration API.

To unlock the gateway for 60 seconds visit the gateway main page (see discovery) in the browser and choose `Settings/System` from the top menu. On the system page click on the `unlock` button in order to unlock the gateway.

In the next 60 seconds any app may acquire a new API key.

HTTP basic authentication

Apps might want to receive an API key without the need that the user must unlock the gateway. This could be achieved by asking the user for the gateway username and password and handover the credentials in the Acquire API key call via HTTP basic authentification.

The API call needs to be extended with HTTP header field `Authorization` as follows:

```
Authorization: Basic <credential-hash>
```

There <credential-hash> is the base64 encoded version of `username:password`.

Error handling

HTTP status codes

Errors might occur for various reasons. Robust applications shall handle them and not assume that each API call will succeed.

As usual in REST APIs errors are returned as HTTP status codes. The documentation for each API call lists all possible errors which might occur.

Error code	Name	Description
200	OK	Request succeded
201	Created	A new resource was created
202	Accepted	Request will be processed but isn't finished yet
304	Not Modified	Is returned if the request had the If-None-Match header and the ETag on the resource was the same.
400	Bad request	The request was not formated as expected or missing parameters
401	Unauthorized	Authorization failed
403	Forbidden	The caller has no rights to access the requested URI
404	Resource Not Found	The requested resource (light, group, ...) was not found
503	Service Unavailable	The device is not connected to the network or too busy to handle further requests

JSON error objects

Further details of the errors are available as JSON object in the response body.

```
{
	"error": {
		"type": <error code>,
		"address": <ressource/parameter>,
		"description": <description>
	}
}
```

Field	Type	Description
type	Number	One of the error codes listed below.
address	String	The url which refers to the resource/parameter which caused the error.
description	String	The error description contains details on what went wrong.

Errors

Error	Description	Details
1	unauthorized user	This will be returned if the request had no valid **apikey** or if the apikey has not the rights to access a resource.
2	body contains invalid JSON	This will be returned if the JSON in the body couldn't be parsed.
3	resource, `<resource>`, not available	This will be returned if the requestet resource like a light or a group does not exist.

4	method, `<method>`, not available for resource, `<resource>`	This will be returned if the requested method (GET, PUT, POST or DELETE) is not supported for the resource.
5	missing parameters in body	This will be returned if the request didn't contain all required parameters.
6	parameter, `<parameter>`, not available	This will be returned if a parameter sent in the request is not supported.
7	invalid value, `<value>`, for parameter, `<parameter>`	This will be returned if a parameter hasn't the expected format or is out of range.
8	parameter, `<parameter>`, is not modifiable	This will be returned in an attempt to change a read only parameter.