eLAN-RF-003 API documentation

Obsah

1. Introduction

Api for ELAN RF based on HTTP, WebSocket and JSON. IP addresses urls in this document you should replace by actual IP address of ELAN. Examples are written using curl and BASH, with another shell there can be some minor changes (different parenthesis and escaping). Applications can be created by any tools and languages which supports HTTP and WebSockets (if notifications are required).

2. Error handling

Error handling is done in simplified REST-like fashion. Success is designated by `2xx` response codes depending on HTTP method used to obtain a resource, client errors (invalid data, improper action,…) are designated by `400 Bad Request`, server (device) errors are designated by `500 Internal Server Error`. On error, body of the response contains a JSON schema further describing the error. Format of JSON error schema is following:

```
{ "error": { "id": error_id } }
```

Error IDs

Client errors

```
RESOURCE_NOT_FOUND = 4 ("Requested resource not found"),

FILE_TOO_LARGE = 5 ("The file size exceeds the limit allowed and cannot be saved"),

IMAGE_TOO_LARGE = 6 ("The image dimensions exceed the limit allowed and cannot be saved"),

INVALID_FILE_FORMAT = 7 ("Invalid file format")
```

Server errors

```
DEVICE_NOT_RESPONDING = 3 ("Requested device is not responding"),

NO_SPACE_LEFT = 7 ("No space left on the device"),

LIMIT_REACHED = 8 ("System limit was reached"),

DEVICE_BUSY = 9 ("Device is busy")
```

GET

On success `200 OK`.

POST

On success `201 Created`.

PUT

On success `200 OK`, if response's body is empty `204 No Content`.

DELETE

On success `200 OK`, if response's body is empty `204 No Content`. If action cannot be taken immediatelly and client is obliged to send another request to make sure the resource was deleted, `202 Accepted`.

3. /api

```
curl -XGET http://10.10.5.185/api
```


```
{
    "info":{
        "ELAN-RF version": "1.6.108",
        "API version": "0.4",
        "MAC address": "8c:70:5a:f3:95:52"
    },
    "notifications": "ws://10.10.5.185/api/ws",
    "rooms": "http://10.10.5.185/api/rooms",
    "devices": "http://10.10.5.185/api/devices",
    "configuration": "http://10.10.5.185/api/configuration",
    "floorplans": "http://10.10.5.185/api/floorplans",
    "scenes": "http://10.10.5.185/api/scenes",
    "temperature": "http://10.10.5.185/api/temperature"
}
```

ELAN-RF version
is version of ELAN firmware. Newer versions contains bug fixes, optimization, etc. and usually does not change API.
API version
is version of API, make sure that API version is supported by your application
notifications
is WebSocket url (see Notifications)

Rest are URLs to other parts of API. These URLSs should be read from `/api` endpoint and not hard-coded to an application.

4. /api/configuration

```
curl -XGET http://10.10.5.185/api/configuration

{

    "configuration changes counter": 698,

    "networking": "http://10.10.5.185/api/configuration/networking",

    "counter": "http://10.10.5.185/api/configuration/counter",

    "firmware": "http://10.10.5.185/api/configuration/firmware",

    "data": "http://10.10.5.185/api/configuration/data",

    "product types": "http://10.10.5.185/api/configuration/product_types",

    "device types": "http://10.10.5.185/api/configuration/device_types",

    "room types": "http://10.10.5.185/api/configuration/room_types",

    "time": "http://10.10.5.185/api/configuration/time",

    "flash memory": "http://10.10.5.185/api/configuration/flash_memory",

    "limits": "http://10.10.5.185/api/configuration/limits"

}
```

configuration changes counter
increased if configuration of ELAN changed
networking
elan network settings

```
curl -XGET http://10.10.5.185/api/configuration/networking

{

    "IP": "10.10.5.185",

    "Mask": "255.255.254.0",

    "Gateway": "10.10.5.254",
```

```
    "DHCP": true
}
```

Change network settings using API:

```
curl -XPOST http://10.10.5.185/api/configuration/networking -d '
{
    "IP": "10.10.5.185",
    "Mask": "255.255.254.0",
    "Gateway": "10.10.5.254",
    "DHCP": false
}
```

firmware
is endpoint for uploading new firmware

```
curl -XPOST http://10.10.5.185/api/configuration/firmware --data-binary @ELANRF_firmware_file.bin
```

data
allows exporting and importing whole ELAN settings

Export:

```
curl -XGET http://10.10.5.185/api/configuration/data -o data.bin
```

Import:

```
curl -XPOST http://10.10.5.185/api/configuration/data --data-binary @data.bin
```

product types
list of RF products supported by ELAN and available when adding device (see /api/devices). In current firmware (and API) version are available following RF products:

- HeatCoolArea
- RFATV-1
- RFDA-11B
- RFDA-71B
- RFDA-73M/RGB
- RFDAC-71B
- RFDEL-71B
- RFDSC-11

- RFDSC-71
- RFJA-12B
- RF-RGB-LED-550
- RFSA-11B
- RFSA-61B
- RFSA-61M
- RFSA-62B
- RFSA-66M
- RFSAI-61B
- RFSC-11
- RFSC-61
- RFSTI-11B
- RFSTI-11G
- RFTI-10B
- RFUS-11
- RFUS-61
- RF-White-LED-675

```
curl -XGET http://10.10.5.185/api/configuration/product_types

["RFSA-62B", "RFSAI-61B", "RFDSC-11", "RF-RGB-LED-550", "RFSA-66M", "RFJA-12B", "RFSTI-11B", "RFDA-11B", "RF-White-LED-675", "RFATV-1", "RFSA-11B", "HeatCoolArea", "RFDA-71B", "RFSA-61B", "RFSA-61M", "RFDAC-71B", "RFUS-61", "RFDEL-71B", "RFSTI-11G", "RFSC-61", "RFDA-73M/RGB", "RFTI-10B"]
```

device types
available types of devices. Type is semantic information about purpose of device connected to RF product. This list can be changed using API. Type of created device has to be specified here. Default device types are (all default types has to be present if ELKO EP MARF or MIRF applications will be used):

- appliance
- blinds
- climatization
- dehumidifier
- detector
- dimmed light
- gate
- heating
- irrigation
- lamp
- light
- rgb light
- temperature regulation area
- thermometer
- ventilation

```
curl -XGET http://10.10.5.185/api/configuration/device_types
```

```
{

    "light": { "icons": { "default": "light.png" } },

    "lamp": { "icons": { "default": "lamp.png" } },

    "dimmed light": { "icons": { "default": "dimmed_light.png" } },

    "new device type": {}

}
```

If custom device type is necessary, GET JSON object with device types, add custom device type and POST**whole** new object:

```
curl -XPOST http://10.10.5.185/api/configuration/device_types -d '

{

    "light": { "icons": { "default": "light.png" } },

    "lamp": { "icons": { "default": "lamp.png" } },

    "dimmed light": { "icons": {"default": "dimmed_light.png"} },

    "new device type": {}

}'
```

room types
available types of rooms. Type is semantic information about purpose of room. This list can be changed using API. Type of created room has to be specified here. Default room types are (all default types has to be present if ELKO EP MARF or MIRF applications will be used):

- living room
- bedroom
- kitchen
- dining room
- child room
- guestroom
- hall
- study
- workroom
- bar
- garage
- garden
- bathroom
- toilet

- laundry
- misc
- pool
- cellar

```
curl -XGET http://10.10.5.185/api/configuration/room_types

{

    "living room":{},

    "bedroom":{},

    "kitchen":{},

    "dining room":{},

    "child room":{}

}
```

If custom device type is necessary, GET JSON object with device types, add custom device type and POST**whole** new object:

```
curl -POST http://10.10.5.185/api/configuration/room_types -d '

{

    "living room":{},

    "bedroom":{},

    "kitchen":{},

    "dining room":{},

    "child room":{},

    "new room type": {}

}'
```

time
settings of NTP server IP address and timezone

```
curl -XGET http://10.10.5.185/api/configuration/time
```

```
{
    "NTP": "81.0.246.244",
    "timezone": 2,
    "timestamp": 1415007869,
    "daylight saving time": true
}
```

Time settings can be set by

```
curl -POST http://10.10.5.185/api/configuration/time -d '
{
    "NTP": "81.0.246.244",
    "timezone": -2,
    "timestamp": 1415007869,
    "daylight saving time": true}'
```

flash memory
shows usage of ELAN flash memory in bytes, application should check available memory before uploading uploading files to ELAN.

```
curl -XGET http://10.10.5.185/api/configuration/flash_memory
{ "free": 3906560, "total": 4041728 }
```

limits
list of existing limits. These values should be taken into account by client applications.

```
curl -XGET http://10.10.5.185/api/configuration/limits
{
  "heatcoolarea": {
    "heating devices": {
      "max count": 12
```

```
    },
    "temperature offset": {
        "max": 20.0,
        "min": -20.0
    },
    "cooling devices": {
        "max count": 12
    }
},
"camera": {
    "max count": 10
},
"timeschedule": {
    "time interval": {
        "max count": 8
    },
    "mode temperature": {
        "max": 32.0,
        "min": 15.0
    },
    "hysteresis": {
        "max": 5.0,
        "min": 0.5
    }
```

```
    }
}
```

5. /api/devices

Device is abstraction of physical device connected to RF product. In /api/devices endpoint are configured devices indexed by their `id`. Every device has assigned url with its description.

```
curl -XGET http://10.10.5.185/api/devices

{
    "RFSA61B": {"url": "http://10.10.5.185/api/devices/RFSA61B"},
    "RFDA71B": {"url": "http://10.10.5.185/api/devices/RFDA71B"},
    "RFDAC71B": {"url": "http://10.10.5.185/api/devices/RFDAC71B"}
}
```

Configure devices

Add device

```
curl -XPOST http://10.10.5.185/api/devices -d '
{
    "device info": {
        "product type": "RFSA-61B",
        "type": "light",
        "label": "Device label in UTF-8",
        "address": 5585
    },
    "id": "RFSA61B_01"
}'
```

id

is unique identifier of device, if device with same id exists it is replaced by new definition. `id` is optional and ELAN generates one automatically if it is not present. `id` has to match regular expression `^[0-9A-Za-z_-]+`.

product type
specifies RF actuator type, it has to be one from product types list.

type
is type of device connected to RF actuator, it has to be one from device types list.

label
UTF-8 description of device, usually visible to user.

address
RF address of actuator as decimal number. User should enter this as hexadecimal number (same as it is printed on actuators box).

Delete device

```
curl -XDELETE http://10.10.5.185/api/devices/RFSA61B_01
```

ELAN keeps consistency of /api/rooms and /api/scenes, if device is removed, it is removed from rooms and scenes too.

Controlling devices

```
curl -XGET http://10.10.5.185/api/devices/RFSA61B_01

{
    "device info": {
        "product type": "RFSA-61B",
        "type": "light",
        "label": "Light in kitchen",
        "address": 5585
    },
    "id": "RFSA61B_01",
    "actions info": {
        "on": {
            "type": "bool"
```

```
        },
        "delayed on: set time": {
            "max": 3600,
            "step": 1,
            "type": "int",
            "min": 2
        },
        "delayed on": {
            "type": null
        },
        "delayed off: set time": {
            "max": 3600,
            "step": 1,
            "type": "int",
            "min": 2
        },
        "delayed off": {
            "type": null
        }
    },
    "primary actions": ["on"],
    "secondary actions": [["delayed on", "delayed on: set time"], ["delayed on", "delayed on: set time"]],
    "settings": {
        "delayed off: set time": 0,
```

```
        "delayed on: set time": 0
    }
}
```

Actions

actions info
contains description of available actions.
Actions are sent as `PUT` request to device url with action specified in body. Device can be turned on:

```
curl -XPUT http://10.10.5.185/api/devices/RFSA61B_01 -d '{"on": true}'
```

or off:

```
curl -XPUT http://10.10.5.185/api/devices/RFSA61B_01 -d '{"on": false}'
```

Devices accepts only one action in one request. Every action has specified type.
null
action without parameter: `{"delayed on": null}`
bool
parameter is boolean value `true` or `false`: `{"on": true}` `{"on": false}`
int
parameter is integer between `max` and `min` inclusive and can have only values $x=min+step*n$, where $n \in \mathbb{N}$
number
same as int, but represents floating point number
primary actions
actions which should be accessible to user immediately (for example after click
secondary actions
actions which should be accessible from some kind of hidden menu, nested lists groups actions which belongs together
settings
is handled internally by ELAN, it has no meaning for client applications

State

State of device can be retrieved by GET request to `/api/devices/[device id]/state`:

```
curl -XGET http://10.10.5.185/api/devices/RFSA61B_01/state

{
    "on": true,
```

```
    "delayed off: set time": 0,
    "delayed on: set time": 0
}
```

If action directly changes state it has same name as appropriate state. If device doesn't know its own state, it returns `null` as state:

```
{
    "on": null,
    "delayed off: set time": 0,
    "delayed on: set time": 0
}
```

If device doesn't communicate at all, it returns only `null`:

```
null
```

6. /api/rooms

Room is group of devices. It can be devices which are actually together in physical room (living room, kitchen) or devices which are supposed to be controlled together (for example room containing all lights). Managing rooms through API is similar to devices.

```
curl -XGET http://10.10.5.185/api/rooms
{
    "lights": { "url": "http://10.10.5.185/api/rooms/lights" },
    "all": { "url": "http://10.10.5.185/api/rooms/all" }
}
```

Configure rooms

Add room

```
curl -XPOST http://10.10.5.185/api/rooms -d '
```

```
{
    "id": "lights",
    "room info": {"label": "Lights", "type": "misc"},
    "floorplan": null,
    "devices":{
        "RFSA11B": { "coordinates": null },
        "RFDA11B": { "coordinates": null },
        "RFSA62B_01": { "coordinates": null },
        "RFRGB01": { "coordinates": null }
    }
}'
curl -XPOST http://10.10.5.185/api/rooms -d '
{
    "id": "kitchen",
    "room info": {"label": "Kitchen", "type": "kitchen"},
    "floorplan": {"image":"kitchen.jpg"},
    "devices":{
        "RFSA11B":{"coordinates":[0.1475, 0.325]},
        "RFDA11B":{"coordinates":[0.6167, 0.3313]},
        "RFSA62B_01":{"coordinates":[0.3975, 0.2775]},
        "RFRGB01":{"coordinates":[0.4308, 0.6413]}
    }
```

id
see devices
label

UTF-8 description of room

type

type (purpose of room). Has to be one of types specified in `/api/configuration/room_types`.

floorplan

is reference to one of floorplans in `/api/floorplans` or `null` if floorplan is not specified.

devices

devices which are in room (reference to device id in `/api/devices`)

coordinates

position of device in floorplan (or `null` if floorplan is null). Coordinate `[0, 0]` is in the top left corner and `[1.0, 1.0]` is in the bottom right corner of floorplan.

Delete room

```
curl -XDELETE http://10.10.5.185/api/rooms/lights
```

Controlling rooms

```
curl -XGET http://10.10.5.185/api/rooms/room031

{
    "room info": {
        "type": "study",
            "label": "jednotky"
    },
    "devices": {
        "RFSTI-11B00346": {
            "coordinates": [
                0, 0
            ]
        },
        "RFUS-6100349": {
            "coordinates": [
                0, 0
```

```
                ]
            },
            "RFDA-73MRGB00336": {
                "coordinates": [
                    0, 0
                ]
            }
        },
        "id": "room00376",
        "floorplan": null
}
```


Actions

Actions are sent as PUT request to a room URL with an action specified in body of the request. When action is sent to a room, ELAN will select only those devices in the room that support the action, ignoring the rest of them.

```
curl -XPUT http://10.10.5.185/api/rooms/room031 -d '{ "on": true }'
```

7. /api/floorplans

Floorplans are ordinary images with any format which used applications will support.

```
url -XGET http://10.10.5.185/api/floorplans

{

    "kitchen.jpg": "http://10.10.5.185/api/floorplans/kitchen.jpg"

}
```

Add floorplan

```
curl -XPOST http://10.10.5.185/api/floorplans/kitchen.jpg --data-binary @kitchen_picture.jpg
```

`kitchen.jpg` is id of floorplan which will be used in API (for example in rooms). It has to match regular expression `^[0-9A-Za-z_-.]+`.

Download floorplan

```
curl -XGET http://10.10.5.185/api/floorplans/kitchen.jpg
```

Delete floorplan

```
curl -XDELETE http://10.10.5.185/api/floorplans/kitchen.jpg
```

8. /api/scenes

```
curl -XGET http://10.10.5.185/api/scenes

{

    "on": "http://10.10.5.185/api/scenes/on",

    "off": "http://10.10.5.185/api/scenes/off"

}
```

Scene is sequence of actions on various devices.

Add scene

```
curl -XPOST http://10.10.5.185/api/scenes -d '

{

    "id": "lon",

    "actions": [

        { "RFSA61B": {"on": true} },

        { "RFDA71B": {"brightness": 80} },

        { "RFDAC71B": {"brightness": 80} },

        { "RFSA62B_02": {"on": true} },

        { "RFRGB01": {"blue": 0, "green": 0, "red": 255, "brightness": 255} }

    ],

    "label": "Lights on"

}'
```

id
see /api/rooms and /api/devices.
actions
list of actions in specified order. Each action is JSON object with one property which determines device id, value of this property is action in the same format as is used in /api/devices.
label
UTF-8 desription of scene.

Launch scene

```
curl -XPUT http://10.10.5.185/api/scenes/lon
```

Actions specified in scene are launched immediately one after another (delayed on/off and increase/decrease actions doesn't make pause in performing actions).

Delete scene

```
curl -XDELETE http://10.10.5.185/api/scenes/lon
```

9. Notifications

Url for WebSocket notifications is specified in response to /api endpoint. WebSocket message contains url which changed (/api/rooms, /api/devices, /api/floorplans etc.). If state of device is changed, url of device is sent and application should `GET` `/api/devices/[device id]/state` to retrieve new state of device.

10. RF actuators

RFSA-11B

Basic switching actuator, only `on` action available.

```
{
    "device info": {...},
    "actions info": {
        "on": {
            "type": "bool"
        }
    },
    "primary actions": ["on"],
    "secondary actions": [],
    "id": "RFSA11B"
}
```

```
{
    "on": false
}
```


RFSA-61B, RFSA-62B, RFSAI-61B, RFSA-61M, RFSA-66M, RFSC-61, RFUS-61


```
{
    "device info": {...},
    "id": "RFSA61B_01",
    "actions info": {
        "on": {
            "type": "bool"
        },
        "delayed on: set time": {
            "max": 3600,
            "step": 1,
            "type": "int",
            "min": 2
        },
        "delayed on": {
            "type": null
        },
        "delayed off: set time": {
            "max": 3600,
            "step": 1,
```

```
            "type": "int",
            "min": 2
        },
        "delayed off": {
            "type": null
        }
    },
    "primary actions": ["on"],
    "secondary actions": [["delayed on", "delayed on: set time"], ["delayed on", "delayed on: set time"]],
}
```


These switching actuators has `delayed on` and `delayed off` actions available.

delayed on
switches device off immediately and switches it on after amount of time specified by previous `delayed on: set time` action or default two seconds if time wasn't set.

delayed off
switches device on immediately and switches it off after amount of time specified by previous `delayed off: set time` action or default two seconds if time wasn't set.


```
{
    "on": true,
    "delayed off: set time": 2,
    "delayed on: set time": 2
}
```


RFDA-11B, RFDSC-11

Basic dimming actuators, instead `on` action have `brightness`. If `brightness` is set to `0`, light is off, brightness can be increase at steps of size 10 until maximum intensity 100.


```
{
```

```
    "device info": {..},
    "actions info": {
        "brightness": {
            "type": "int",
            "min": 0,
            "max": 100,
            "step": 10
        }
    },
    "primary actions": ["brightness"],
    "secondary actions": ["brightness"],
    "settings": {},
    "id": "RFDA11B"
}
{
    "brightness": 20
}
```


RFDA-71B, RFDAC-71B, RFDEL-71B, RF-White-LED-675


```
{
    "device info": {...},
    "actions info": {
        "brightness": {
            "type": "int",
```

```
            "min": 0,
            "max": 100,
            "step": 10
        },
        "increase": {
            "type": null
        },
        "decrease": {
            "type": null
        },
        "increase: set time": {
            "type": "int",
            "min": 2,
            "max": 1800,
            "step": 1
        },
        "decrease: set time": {
            "type": "int",
            "min": 2,
            "max": 1800,
            "step": 1
        }
    },
    "primary actions": ["brightness"],
```

```
    "secondary actions": ["brightness", ["increase", "increase: set time"], ["decrease", "decrease: set time"]],
    ...
}
```

These dimming actuators support `increase` and `decrease` actions.

increase
continuously increases brightness until maximum value is reached, time of increasing is defined by `increase: set time` action.

decrease
continuously decreases brightness until minimum value is reached, time of decreasing is defined by `decrease: set time` action.

```
{
    "brightness": 0,
    "increase: set time": 0,
    "decrease: set time": 0
}
```

RFDA-73M/RGB, RF-RGB-LED-550

```
{
    "device info": {...},
    "actions info": {
        "red": {
            "type": "int",
            "min": 0,
            "max": 255,
            "step": 1
        },
        "green": {
```

```
            "type": "int",
            "min": 0,
            "max": 255,
            "step": 1
        },
        "blue": {
            "type": "int",
            "min": 0,
            "max": 255,
            "step": 1
        },
        "brightness": {
            "type": "int",
            "min": 0,
            "max": 255,
            "step": 1
        },
        "demo": {
            "type": "bool"
        }
    },
    "primary actions": [["red", "green", "blue", "brightness"]],
    "secondary actions": [["red", "green", "blue", "brightness"], "demo"],
    "settings": {},
```

```
    "id": "RFRGB01"
}
```


Controlling of RGB actuators is slightly different than other actors. Values of red, green blue and brightness has to be set in one `PUT` request as following (order is not important): `{"red": 10, "green": 100, "blue": 200, "brightness": 255}`. Next available action is `{"demo": true/false}` which turns on/off demo mode. Demo is special effect which changes colors of light.


```
{
    "red": 0,
    "green": 0,
    "blue": 0,
    "brightness": 0,
    "demo": false
}
```


RFJA-12B

RFJA-12B is shutter actuator and can be controlled by actions `roll up`, `roll down` and `stop`. What time it takes to roll shutters up or down is set by `set time` action.


```
{
    "device info": {...},
    "actions info": {
        "roll up": {
            "type": null
        },
        "roll down": {
            "type": null
        },
```

```
        "stop": {
            "type": null
        },
        "set time": {
            "type": "int",
            "min": 2,
            "max": 240,
            "step": 1
        }
    },
    "primary actions": ["roll up", "roll down", "stop"],
    "secondary actions": ["roll up", "roll down", "stop", "set time"],
    "settings": { "set time": 0 },
    "id": "RFJA12B"
}
{
    "roll up": true,
    "set time": 2
}
```


State `roll up` changes to `true` if blinds are rolled up and to `false` if they are rolled down. Otherwise this state doesn't change.

Example

- blinds are down, state is `false`
- after sending `roll up` action blinds are going up, state is still `false`
- after sending `stop` blinds stops in the middle, state is still `false`
- after sending `roll up` blinds are going up, they reaches top position and the state changes to `true`

- after sending `roll down` state is still `true` until blinds are rolled down completely

RFTI-10B

```
{
    "device info": {...},
    "actions info": {},
    "primary actions": [],
    "secondary actions": [],
    "id": "RFTI10B"
}
{
    "temperature IN": 27.3,
    "temperature OUT": 18.5,
    "battery": true
}
```

If external sensor is not connected, its state is `null`.

```
{
    "temperature IN": 27.3,
    "temperature OUT": null,
    "battery": true
}
```

RFSTI-11B

```
{
    "device info": {...},
```

```
    "actions info": {
        "safe on": {
            "type": "bool"
        }
    },
    "primary actions": [ ],
    "secondary actions": [ ],
    "id": "RFSTI11B"
}
{
    "temperature": 27.3,
    "on": true
}
```


RFSTI-11G


```
{
    "device info": {...},
    "actions info": {
        "safe on": {
            "type": "bool"
        }
    },
    "primary actions": [ ],
    "secondary actions": [ ],
```

```
    "id": "RFSTI11G"
}
```

```
{
    "temperature IN": 27.3,
    "temperature OUT": 17.3,
    "on": true
}
```


RFTC-10G


```
{
    "device info": {...},
    "actions info": {},
    "primary actions": [],
    "secondary actions": [],
    "id": "RFTC10B"
}
{
    "temperature": 23.2,
    "battery": true
}
```


RFATV-1

This thermovalve behaves differently than other RF actuators. It initiates communications in certain time interval (default is 6 minutes) and expect response with instructions. Actions performed on this actuator takes effect when next response is send to the actuator. It regulates itself to `requested temperature` and sends actual temperature from internal thermometer. This actuator is also capable to detect open window by rapid temperature drops.
`open window sensitivity` action sets sensitivity to temperature drops:

- 0 - off
- 1 - 1.2°C
- 2 - 0.8°C
- 3 - 0.4°C

Temperature drop is measured in interval of one minute. If `open window sensitivity` is set to 1 and temperature drops by more than 1.2°C during one minute, valve will be turned off for `open window off time` minutes.
`open valve` is in %, 0% is closed and is 100% open. `open window` is `true` if timer after open window detection is running and `false` otherwise. `battery` is `false` if battery inside RFATV is discharged.

```
{
    "device info": {...},
    "actions info": {
        "requested temperature": {
            "type": "number",
            "min": 0,
            "max": 32,
            "step": 0.5
        },
        "open window sensitivity": {
            "type": "int",
            "min": 0,
            "max": 3,
            "step": 1
        },
        "open window off time": {
            "type": "int",
            "min": 0,
```

```
                "max": 60,
                "step": 10
            }
        },
        "primary actions": [],
        "secondary actions": ["requested temperature", "open window sensitivity", "open window off time"],
        "id": "RFATV1"
}
{
        "temperature": 23.3,
        "open window": true,
        "open valve": 9.5,
        "battery": true,
        "requested temperature": 27.5,
        "open window sensitivity": 1,
        "open window off time": 10
}
```


Heat Cool Area


```
{
        "device info": {...},
        "actions info": {
            "mode": {
                "type": "int",
                "min": 1,
```

```
            "max": 4,
            "step": 1
        },
        "correction": {
            "type": "number",
            "min": -5.0,
            "max": 5.0,
            "step": 0.5
        },
        "on": {
            "type": "bool"
        }
    },
    "primary actions": [ ],
    "secondary actions": [ "correction", "mode", "on" ],
    "heating devices": [
    {
        "id": "RFATV123",
        "central source": true
    },
    ...
    ],
    "cooling devices": [
    {
```

```
            "id": "RFSA61B02",
            "central source": true
        },
        ...
    ],
    "schedule": "tempschedule01",
    "temperature sensor": {"RFATV1231": "temperature"},
    "temperature offset": 0.0,
    "id": "HCA1"
}
{
    "temperature": 23.3,
    "mode": 1,
    "correction": -3.5,
    "on": true,
    "requested temperature": 24.5
}
```


mode
sets curent mode
correction
shifts current maximal and minimal temperature
on
sets state of heating/cooling in the area (false => turned off)
requested temperature
temperature that is sent to thermovalve. The value is calculated from mode temperature + correction.

For more information see Temperature schedule.

11./api/temperature

```
curl -XGET http://10.10.5.185/api/temperature
```

```
{
    "sources": "http://10.10.5.185/api/temperature/sources",
    "schedules":"http://10.10.5.185/api/temperature/schedules"
}
curl -XGET http://10.10.5.185/api/temperature/sources
{
    "HCA1": "http://10.10.5.185/api/temperature/sources/HCA1",
    "723648": "http://10.10.5.185/api/temperature/sources/723648",
    "HCA3": "http://10.10.5.185/api/temperature/sources/HCA3"
}
curl -XGET http://10.10.5.185/api/temperature/schedules
{
    "tempschedule01": "http://10.10.5.185/api/temperature/schedules/tempschedule01",
    "tempschedule02": "http://10.10.5.185/api/temperature/schedules/tempschedule01",
    "tempschedule03": "http://10.10.5.185/api/temperature/schedules/tempschedule01"
}
```

HeatCool area

HeatCool area is manipulated as device (it is possible to `POST` it to `/api/devices`) in following format:

```
{
    "device info": {
        "product type": "HeatCoolArea",
        "type": "temperature regulation area",
```

```
        "label": "HeatCoolArea label in UTF-8"
    },
    "schedule": "tempschedule01",
    "heating devices": [
    {
        "id": "RFATV123",
        "central source": true
    },
    ...
    ],
    "cooling devices": [
    {
        "id": "RFSA61B02",
        "central source": true
    },
    ...
    ],
    "temperature sensor": {"RFATV1231": "temperature"},
    "temperature offset": 0.0,
    "id": "HCA1"
}
```


schedule
id if existing schedule
heating/cooling devices
devices which controls heating/cooling in area (switching actuators or thermovalve)
temperature sensor
{id of temperature sensor: state with temperature} which will be used for temperature regulation

in the area

temperature offset

value in degrees of Celsius used as adjustment for temperature sensor that measures incorrectly.

Central source


```
{
    "label": "Boiler",
    "source type": "heating",
    "areas": [
        "HCA1", "HCA2", ...
    ],
    "device": "RFSA61B03",
    "id": "HCS1"
}
{
    "label": "Clima",
    "source type": "cooling",
    "areas": [
        "HCA1", "HCA2", ...
    ],
    "device": "RFSA61B04",
    "id": "HCS1"
}
```


source type

specifies if object is source for heating or cooling

areas

array of identifiers of HeatCool areas connected to this source

device

this device is turned on if there is any active actuator in areas with central source property set

to `true` otherwise it is turned off

Add/change

```
curl -POST http://10.10.5.185/api/temperature/source -d '...'
```

Delete

```
curl -DELETE http://10.10.5.185/api/temperature/source/HCS1
```

Temperature schedule

```
{
    "label": "Bedroom winter schedule",
    "hysteresis": 0.5,
    "modes": {
        "1": {"min": 15, "max": 35},
        "2": {"min": 18, "max": 30},
        "3": {"min": 20, "max": 28},
        "4": {"min": 22, "max": 24}
    },
    "schedule": {
        "monday": [
            {"duration": 400, "mode": 1},
            {"duration": 440, "mode": 2},
            {"duration": 300, "mode": 3},
            {"duration": 300, "mode": 1}
        ],
        "tuesday":[...],
```

```
            "wednesday":[...],
            "thursday":[...],
            "friday":[...],
            "saturday":[...],
            "sunday":[...]
        },
        "id": "tempschedule01"
}
```

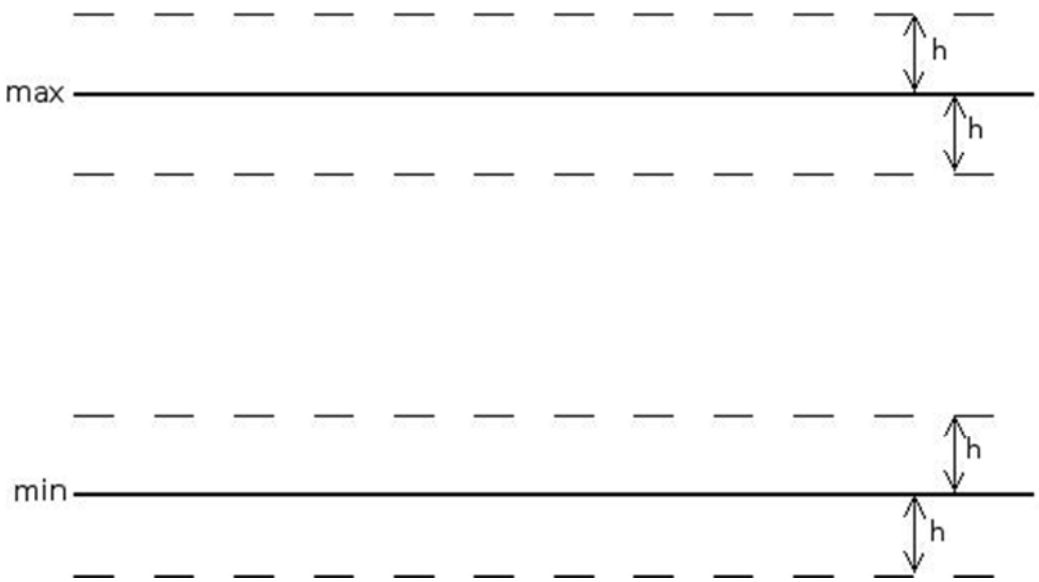

Figure 1: Maximal and minimal temperature and hysteresis.

Weekly schedule is divided to days, every day starts in 00:00 and mode from first item is activated for duration minutes. After duration is activated mode in second item. Mode from last item holds until end of the day independently from duration set in the last item.
max temperature, *min* temperature and hysteresis *h* are shown on the figure 1. If temperature drops below $min-h$ heating will be activated until temperature reaches $min+h$. If temperature raises over $max+h$ cooling will be activated until temperature drops below $max-h$.
Correction *c* shifts whole *max*-*min* interval:
$max_c=max+c$
$min_c=min+c$
Correction holds to the next scheduled item.

Add/change

```
curl -POST http://10.10.5.185/api/temperature/schedule -d '...'
```

Delete

```
curl -DELETE http://10.10.5.185/api/temperature/schedule/tempschedule01
```

12. Examples

There are some usage examples of ELAN RF API. Purpose of the examples is to show interaction with ELAN API. It is not idiomatic code and it is not intended for production use. All examples can be downloaded in zip archive.

JavaScript

These examples can be tried online, just enter IP address of your ELAN (it has to be accessible from your computer) and follow instruction on example pages. Interaction between JavaScript applications and ELAN API uses CORS, so it is possible to call ELAN API from application stored on local file or hosted at another domain.

webcontrol.html and webcontrol.js
shows control of switching, dimming, shutter and RGB actuators, scenes are also covered.

notifications.html and notifications.js
shows how to connect to notification WebSocket and read state of device. It can be used during development of ELAN RF API applications to watch notifications and state of devices.

Author: Michal Mrnuštík, Daniel Beseda
Created: 2014-11-19 St 16:01
Emacs 24.3.1 (Org mode 8.2.4)
Validate