myStrom

REST API

All myStrom devices offer a REST API (REST = representational State Transfer).

The interface allows you to access/control the device directly from your local network independently of myStrom. That means you don't need a myStrom account or the myStrom app. With this documentation you can integrate myStrom devices in almost any environment.

MAC address

Often in these requests you will see a number like this `200AD4074I68`. This is the MAC address of the device without any delimiters.

Important note for Buttons

Keep in mind that for the Button or Button+ to respond to any of these requests they have to be in the configuration mode:

- Button: (Re-)Connect the button to a power source with the provided USB cable. Press the button and after some time it should become visible in the network.
- Button+: Open the back of the button by rotating it clockwise. Remove the batteries and reinsert them. The Button+ should now be visible.

Detect myStrom devices

To discover a myStrom device in your network, you mus listen on UDP port 7979. Each myStrom device will broadcast a message (buttons only if they are in the configuration mode). The first 6 bytes contain the mac address of the device and the following two bytes are a number that corresponds to the device type. See below for the list of type Numbers.

Security

myStrom Switches & myStrom LED Strips have theire own web interface where a user can specify an API access token. If such a token is specified, any request must have header set accordingly. E.g. `curl -H "Token: XXXXX" 192.....`. In order to access the web interface simply visit the devcies ip address in your webbrowser. When the devcies are in wifi mode, you can also manually connect to these devices by connecting to their wifi network, visit the web interface at 192.168.254.1 and set the wifi the device should connect to and its password directly in the web interface.

For all other devices API is transparent and has no authentication. If someone has access to your local network, **they will be able to control your myStrom devices**. Please apply strong security mechanisms to protect your network.

The CORS protection

In order to increase security by preventing unwanted execution of malicious HTTP requests to devices from browsers, e-mail clients, etc. with specially prepared scripts and URLs, CORS protection was introduced.

This mechanism works in a simple way to minimize its impact on external integrating systems. If the HTTP request in the header contains the `Accept-Encoding` field then the same request should also have a `Referer` or `Origin` header field starting with `http://{target device ip}`. If the `Referer` or `Origin` field does not match the expected value or is missing in the case of a request with `Accept-Encoding` then the request is rejected. If the request is made without the `Accept-Encoding` header then the `Referer` and `Origin` fields do not need to be provided.

Some requests cannot be performed if the `Accept-Encoding` header is given even if the correct `Referer` and `Origin` values are given. This are mainly requests that allow you to change the device state by using the `GET` method.

It also means that the query from the browser by simply entering the IP address of the device and operations on the REST API will not work.

You can disable this mechanism if you are not a developer of a more integration system available to a larger number of users, otherwise you run the risk of unwanted REST API operations that may lead to data acquisition or operations on the user's device.

To disable protection using the curl utility, make the following request:

```
curl -i -X POST http://{deviceip}/api/v1/protection/disable
```

To re-enable protection, follow the command below or reset the device to factory settings:

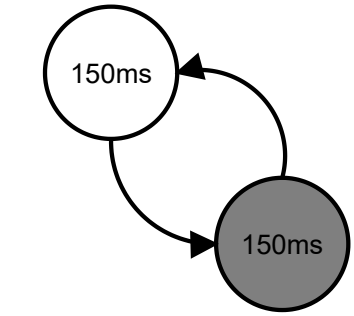
```
curl -i -X POST http://{deviceip}/api/v1/protection/enable
```

Security does not work in the self AP mode.

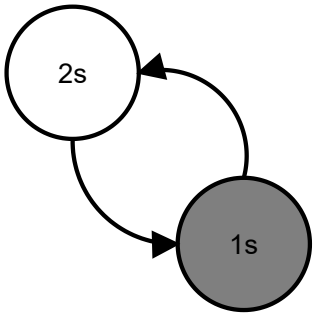
Protection has been implemented or will be introduced starting with the following firmwares:

- WS2/WSE/WRS/WLL 3.82.56
- WRB 2.59.32
- WBP/WBS 2.74.36

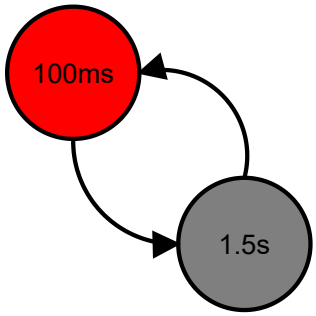
LEDs blinking patterns



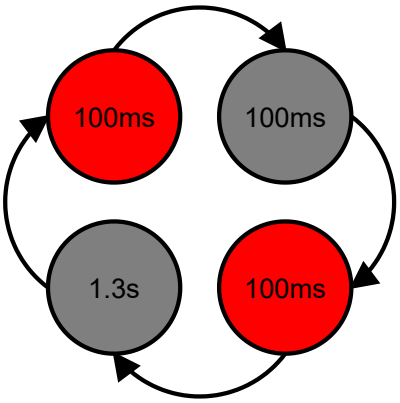
Factory reset



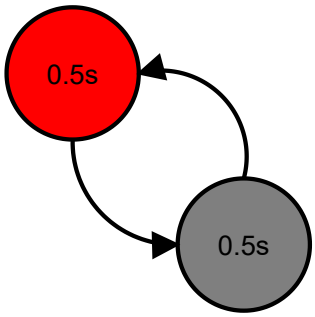
WPS is on



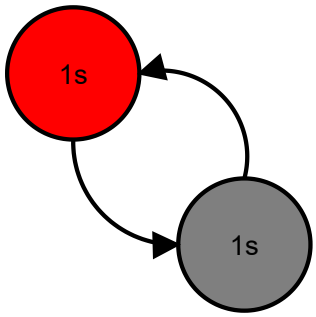
AP mode is on



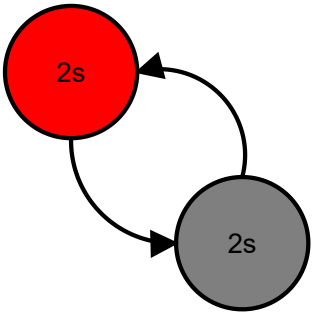
WAC



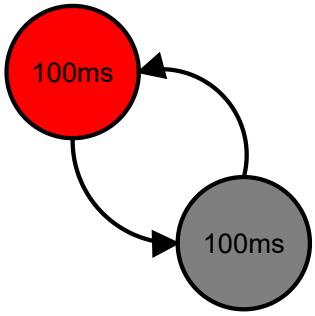
Connecting to a Wi-Fi network



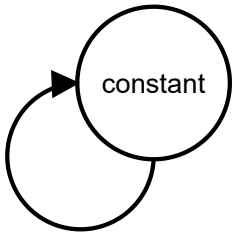
Connected to a Wi-Fi network



An IP address has been obtained



Upgrading the firmware



Connected to the cloud

Block Diagram

General Requests

These requests can be executed on any myStrom device.

GET Get general information

http://[device_ip]/api/v1/info

Gets general information about your myStrom device. Will return a JSON object with:

Field	Type	Description
version	STRING	Current firmware version
mac	STRING	MAC address, without any delimeters in capital characters
type	UINT	Numeric device type
ssid	STRING	The name of the WiFi network to which the device connects
ip	STRING	Current ip address
mask	STRING	The mask of the network the device connects to
gw	STRING	Gateway IP address
dns	STRING	DNS server address
static	BOOL	If the network IP addresses have been entered manually, the field is set to true, if the address is obtained from the DHCP server, the field is set to false
connected	BOOL	The device is set up to connect

The type list is as follows:

Device	Numeric type	**Literal type**
Switch CH v1	101	WSW
Bulb	102	WRS
Button+	103	WBP
Button	104	WBS
LED strip	105	WRS
Switch CH v2	106	WS2
Switch EU	107	WSE
Motion sensor	110	WMS
modulo® STECCO modulo® CUBO	113	WLL
Button Plus 2nd	118	BP2
Switch Zero	120	LCS

POST Connecting to a Wi-Fi network

http://[device_ip]/api/v1/connect

Configuration of connection to the WiFi network. The resource accepts a message with a JSON object in the body described below:

Field	Type	Required	Description
ssid	STRING 1..32	yes	The name of the WiFi network to which the device is to connect
passwd	STRING 0..64	no*	Password for the secured network, *some devices cannot connect to open networks
ip	IP	no	The field and the following may be omitted when the IP configuration is to be obtained from the DHCP server, otherwise this and all the fields below must be set to the correct IP addresses
mask	IP	no	
gw	IP	no	

The field of IP type is a string of characters representing the numeric representation of the IPv4 address: 192.168.1.100 or 8.8.8.8.

Mention: Before using this API, first call api/v1/scan and make sure that the network is within range (applies to firmware version below 4, not applicable to Wi-Fi bulb/button).

bash

```
curl -i -d '{"ssid":"WIFI", "passwd":"PASSWORD"}' http://192.168.254.1/api/v1/connect
```

Body raw (json)

json

```
{
    "ssid": "WIFI",
    "passwd": "PASSWORD"
}
```

GET Find nearby Wi-Fi networks

http://[device_ip]/api/v1/scan

Scans for nearby wifi networks. Will return a JSON array consisting alternatingly of a wifi name and its signal strength

Field	Type	Description
Even field	STRING	The name of the WiFi network found
Odd field	SINT	The signal strength of the Wi-Fi network mentioned in the previous field

POST Manual firmware upgrade

http://[Device IP]/load

Upgrade the firmware of the device to the firmwarefile provided.

Make sure that the firmware (.bin file) corresponds to your device type before uploading.

Returns an empty body.

Body formdata

file	@[FILE ON DISK]

GET Help

http://[device_ip]/help

Returns quick help of available HTTP API queries, result depends on device type.

Conventions

The first word specifies the type of method the query should be executed: GET or POST. Then the path and possible parameters appear. Square brackets indicate the optional part. Sharp brackets mean that value should be substituted for them. The sharp start parenthesis is followed by the type of the variable and its range. The pipe character specifies that one of the values separated by it should be set. If there is a space after the path and its parameters, then the query parameters should be sent as a JSON object, otherwise the parameters for POST queries should be sent in the format key=value[&key=value] application / x-www-form-urlencoded. This API is available for all device types with the latest firmware version. It may not be on older devices.

Settings

It allows you to read and write certain generic settings on the device, such as HTTP server access, device name and temperature offset.

The GET method returns a JSON object with the fields described in the following table. POST method takes the same object with changed values. In the object sent by the POST method, you do not need to send all the fields, you can also update individual fields.

Name	Type	Range	Default	Remarks
rest	bool		true	Set this parameter to false will disable access to device control requests (REST API).
panel	bool		true	Set this parameter to false will disable the WebUI, this API also will be blocked, so one way to reenable panel is factory reset.
name	string	0..32	""	Allows set the name of device.
token	string	0..256	""	This parameter is helpful for securing access to the HTTP

The code returned on successful value setting with POST is 200 OK. Code 400 is returned if there is a problem with parameters or their types and ranges.

GET Get device common settings

http://[device_ip]/api/v1/settings

POST Set settings

http://[device_ip]/api/v1/settings

Example of changing local temperature offset:

Under Linux:

bash

```
curl -d '{"temp_offset":"-1"}' -H "Content-Type: application/json" -X POST
http://192.168.1.121/api/v1/settings
```

HTTP response 200 OK with payload

json

```
{"rest":true,"panel":true,"hap_disable":false,"name":"","temp_offset":-1}
```

Under Windows CMD Shell:

bash

```
curl -d "{\"temp_offset\":\"1\"}" -H "Content-Type: application/json" -X POST
http://192.168.1.121/api/v1/settings
```

HTTP response 200 OK with payload

json

```
{"rest":true,"panel":true,"hap_disable":false,"name":"","temp_offset":1}
```

Body raw (json)

json

```
{
    "rest": true,
    "panel": true,
    "hap_disable": false,
    "name": "",
    "temp_offset": 0
}
```

myStrom Switch

Requests specific to myStrom Switch EU and CH devices.

Internet monitor

Turns the switch off and on again depending on whether or not a ping succeeds.

With this you can temporarily switch relay off if you loose internet connectivity. This can be very usefull if a router is attached to the switch

attached to the switch.

By default, this functionality is disabled.

Configuration parameters:

Field	Type	Description
enable	BOOL	Determines whether the functionality should be enabled
address	STRING	IPv4 address written numerically or in the form of a host name that will be periodically polled.
tryAt	UINT (unit s)	Interval until we check again
attempts	UINT(1..255)	How many times we retry to check the address if a request has failed
inhibitTIme	UINT (unit s)	Time for which to disable polling in the event of no response from the host and relay action was performed
pingTimeout	UINT (unit ms)	How long the ping waits for a response until it fails
relayOffTime	UINT (unit s)	The relay turn off time, after this time the relay is restored to previously state

The configuration is returned in the form of a JSON object, the object can be modified and sent back, it is also possible to send individual parts of the object separately

GET Get internet monitor settings

http://[switch_ip]/api/v1/monitor

POST Set internet monitor settings

http://[switch_ip]/api/v1/monitor

HEADERS

Content-Type application/json

Body raw (json)

json


```
{
    "address": "mystrom.ch",
    "tryAt": 15,
    "attempts": 3,
    "inhibitTime": 60,
    "pingTimeout": 3000,
    "relayOffTime": 5,
    "enabled": false
}
```


GET Turn on

http://[switch_ip]/relay?state=1

Turns the switch on. Does not return anything.

PARAMS

state 1

The value the relay/switch should be set to. 1 = turn on, 0 = turn off

GET Turn off

http://[switch_ip]/relay?state=0

Turns the switch off. Does not return anything.

PARAMS

state 0

The value the relay/switch should be set to. 1 = turn on, 0 = turn off

GET Toggle

http://[switch_ip]/toggle

Toggles the switch. Returns a JSON object with the following field:

- `relay` : meaning if the relay/switch has now been set to off (false) or on (true).

GET Get log

http://[Switch IP]/log

Gets the log of the myStrom switch. Returned as a raw string not as JSON.

GET Get report

http://[switch_ip]/report

Gets a report from the switch. Returns a JSON object with the following fields:

- `power`: The current power consumed by devices attached to the switch
- `relay`: The current state of the relay (wether or not the relay is currently turned on)
- `temperature`: The currently measured temperature by the switch. (Might initially be wrong, but will automatically correct itself over the span of a few hours)
- `Ws`: The Ws field represents the average power value since the last call. It is the energy consumed divided by the time elapsed since the last call. *Energy = Ws · ΔT. For continous consumption measurements (Ws values with timestamp stored in DB): Total Energy =* ∑Ws · (Tcurrent - Tbegin).

GET Get measured temperature (old path)

http://[switch_ip]/temp

Gets more detailed information about the temperature.
The compensation field might initially be wrong, but will automatically correct itself over the span of a few hours.
Returns a JSON object with the following fields:

- `measured`: The measured raw temperature by sensor
- `compensation`: How much we have to compensate the raw `measured` data for the cpu heat.
- `compensated`: The actual room temperature near the device (i.e. `measured - compensation`)
- `offset`*: Local temperature offset, value can be set using API api/v1/settings. This field is available in new firmware versions.

GET Get measured temperature

http://[switch_ip]/api/v1/temperature

Gets more detailed information about the temperature.
The compensation field might initially be wrong, but will automatically correct itself over the span of a few hours.
Returns a JSON object with the following fields:

- `measured` : The measured raw temperature by sensor
- `compensation` : How much we have to compensate the raw `measured` data for the cpu heat.
- `compensated` : The actual room temperature near the device (i.e. `measured - compensation`)
- `offset` *: Local temperature offset, value can be set using API api/v1/settings. This field is available in new firmware versions.

GET Scan Wifi (deprecated)

[switch_ip]/networks

Scans for nearby wifi networks in a detailed manner. Returns a JSON object where each detected wifi signal is a field. For each detected wifi it returns

- `name` : The name of the WiFi
- `signal` : The signal strenght
- `encryption-on` : Wether or not the wifi signal is encrypted
- `encryption` : The encryption standard used

GET Power cycle

http://[switch_ip]/power_cycle?time=10

Turns the switch off, waits for a specified amount of time (max 1h), then starts it again.

The switch has to be turned on in order for this call to work. We can send the following parameters:

- `time` : how long the switch should wait until it should restart [s] (max 3600).

PARAMS

time	10 The value in seconds determines how long to switch off the relay

---

POST Timer

http://[switch_ip]/timer?mode=on&time=5

It sets the relay state for a given time, after the time has elapsed the state of the relay is reversed.

The resource accepts the following parameters that can be passed as part of the query or as the body of the message

- `mode` :

- `on` : Turns the switch on (if it is not already), waits for specified time and turns the switch off again
- `off` : Turns the switch off (if it is not already), waits for specified time and turns the switch on again
- `toggle` : Toggles the switch, waits for specified time and toggles the switch on again
- `none` : Cancel previously planned action

- `time` : Time in seconds after which the action will be performed (max 3600).

PARAMS

mode	on The state of the relay that will be set after calling the resource. The set state will be maintained for the time specified by the parameter.
time	5 Time in seconds after which the action will be performed

---

myStrom Button & Button+

All request specific to the myStrom Button and Button+.

We interact with both types in the exact same way, however with the standard Button we can not execute a touch, thus the field `touch` is meaningless for the standard button.

Calling feedback:

When the button executes a http request, it will blink according the returned status code. For 200 it will blink green and for any other response code or if the request has timed out it will blink red.

Actions priorities:

Since the button can execute multiple action when it is pressed we need to specify an order in which these requests take place (e.g. It can call the myStrom server and additionally execute a http request that has been programmed manually). The order is as follows:

1. Call myStrom server (If the button is registered to a myStrom account)
2. Manually set action (i.e. requests shown in the single, double, long or touch field)
3. Generic action (i.e. the action shown in the generic field)

POST Set Button actions

http://[Button IP]/api/v1/action/single

General Actions

The URL has the form `/api/v1/action/<ENUM single|double|long|generic>`

We set each button action by simply providing the interaction type (single, double, long or touch) and what kind of request we want to happen upon that interaction.

We introduce an unconventional way to specify what kind of request the button should execute. A url of form `post://` will execute a post request and one of form `get://` will execute a get request. Requests of form

`put://` and `delete://` can also be used for PUT and DELETE requests respectively.

In the first postman example we set the single click action to send a post request to 192.168.1.153 on port 9090 with data "key=value" and "key2=value2"

In the second postman example we set the double click action to send a get request to 192.168.1.153 on port 9090

We will not get a return value.

Advanced Feautres

Implemented in Firmware version: 2.74.10

Multiple actions

The buttons can execute multiple actions after each other. Between every two actions we put "||" to show the end of the first action and the start of a new one. The number of actions we can set is limited! The number of characters of the entire string that we write to a specific action (i.e. the length of all actions including vertical delimiters) has to be smaller than 800 characters.

Example: post://192.168.1.42?key=value||get://192.168.1.173

Generic Action

Generic URL are called for all actions, the callback appends some additional parameters to the request URL which describe the action that just occured.

The Wheel action is triggered when touching the Button+ and performing a circular motion on the touch surface.

In addition, this action is called automatically every twelve hours, which allows you to receive the battery status of the button.

Name	Type	Range	Given	Comments
mac	MAC		yes	
action	ENUM	SINGLE = 1 DOUBLE=2 LONG=3 TOUCH=4 WHEEL=5 WHEEL_FINAL=11 BATTERY=6	yes	
wheel	INT	-127...127	no	Only in action WHEEL case
battery	INT	0..100	yes	In percent

Example:

If the generic field on the Button is set to: `get://192.168.1.144:8787/gen`

then the HTTP server running on 192.168.1.144:8787 will receive GET requests that looks similar to

`/gen?mac=18FE34CD9201&action=3&battery=60`

OR

`/gen?mac=18FE34CD9201&action=5&wheel=38&battery=60`

Body raw

```
post://192.168.1.192:9090?key=value=key2=value2
```

GET Get device specific information

http://[Button IP]/api/v1/device

Returns device specific information. Returns a JSON object with only the mac address (without delimiters) as its field. This field however contains the following data:

- `type` : The type of the device (buttonplus = wheel)
- `battery` : Wether or not the devices is using batteries
- `reachable` : Wether or not the device is connected to a myStrom account
- `meshroot` : Wether or not the device is in a mesh (?)
- `voltage` : Current voltage of the button
- `single` : http request executed when pressing the button once
- `double` : http request executed when pressing the button twice
- `long` : http request executed when pressing the button long
- `touch` : http request executed when touching the button (only used for Button+)
- `generic` : http request execute in case of any action on button. Button add additional fields to the assigned address to identify the action. In addition, this action is called automatically every twelve hours, which allows you to receive the battery status of the button. (Useful for bridges or servers)
- `fw_version` : The firmware version of the Button

GET Get request used for interaction

http://[Button IP]/api/v1/action/single

URL of form `http://<IP>/api/v1/action/<ENUM single|double|long|generic>`

Returns the currently assigned action for the specified interaction with the button.

myStrom Bulb

All request specific to the myStrom Bulb. All these request also work on the LED strip.

POST Turn on

http://[Bulb IP]/api/v1/device/[Bulb Mac]

Turns the bulb on.

Returns a JSON object with the mac address (without delimiters) as its only field. The mac field however has the following fields:

- `on` : wether or not the bulb is now turned on (after the request has been executed)
- `color` : The current color
- `mode` : The color mode the bulb is currently set to
- `ramp` : Transition time from the light's current state to the new state. [ms]
- `notifyurl` : Once defined the bulb will send POST request to that URL whenever its state changes

Body urlencoded

action	on The action we want to take (`on` , `off` or `toggle`)

POST Toggle

http://[Bulb IP]/api/v1/device/[Bulb Mac]

Toggles the bulb.

Returns a JSON object with the mac address (without delimiters) as its only field. The mac field however has the following fields:

- `on` : wether or not the bulb is now turned on (after the request has been executed)
- `color` : The current color
- `mode` : The color mode the bulb is currently set to
- `ramp` : Transition time from the light's current state to the new state. [ms]
- `notifyurl` : Once defined the bulb will send POST request to that URL whenever its state changes

Body urlencoded

action	toggle The action we want to take (`on` , `off` or `toggle`)

POST Turn off

http://[Bulb IP]/api/v1/device/[Bulb Mac]

Turns the bulb off.

Returns a JSON object with the mac address (without delimiters) as its only field. The mac field however has the following fields:

- `on` : wether or not the bulb is now turned on (after the request has been executed)
- `color` : The current color
- `mode` : The color mode the bulb is currently set to
- `ramp` : Transition time from the light's current state to the new state. [ms]
- `notifyurl` : Once defined the bulb will send POST request to that URL whenever its state changes

Body urlencoded

action	off The action we want to take (`on` , `off` or `toggle`)

POST Set color

http://[Bulb IP]/api/v1/device/[Bulb Mac]

Set the bulb to a specified color. Color can be either in hsv or in RGBW. We can send the following parameters:

- `action` : If we want to turn the bulb on or off
- `color` : The color we set the bulb to (When using RGBW mode the first two hex numbers are used for the white channel! hsv is of form `<UINT 0..360>;<UINT 0..100>;<UINT 0..100>`)
- `ramp` : Transition time from the light's current state to the new state. [ms]
- `mode` : The color mode we want the Bulb to set to (rgb or hsv)
- `notifyurl` : Once defined the bulb will send POST request to that URL whenever its state changes

HEADERS

Content-Type	application/x-www-form-urlencoded

Body raw

```
color=0;100;50&mode=hsv&action=on&ramp=20
```

GET Get device specific information

http://[Bulb IP]/api/v1/device

Returns device specific information. Returns a JSON object with only the mac address (without delimiters) as its field.

This field however contains the following data:

- `type` : The type of the device
- `battery` : Wether or not the devices is using batteries
- `reachable` : Wether or not the device is connected to a myStrom account
- `meshroot` : DEPRECATED
- `on` : Wether or not the bulb is currently turned on
- `color` : The current color
- `mode` : The color mode the bulb is currently set to
- `ramp` : How quickly the bulb changes its its color
- `power` : The power consumed by the bulb
- `fw_version` : The firmware version of the bulb

---

myStrom LED strip

All request specific to the myStrom LED strip.

All the Bulbs request also work on the LED strip. See the documentation of the bulb to see further requests.

POST Set Light Effects

http://[Strip IP]/api/v1/effect/set/0

You can specify a list of light effects the LED strip will go through. In this way you can make the LED strip blink or loop through different colors.

The last number of the URL specifies as which effect we want to save our new effect. To specify the effect pass a JSON array where every object in this array consists of 3 value namely:

- `color` : The color we want to set the light strip to can be either in hsv or RGBW format
- `ramp` : How quickly to set the led strip to the specified color [ms]
- `time` : How long we wait until we start the next request [s]
  - If you do not want to interrupt the color ramp up this time has to be bigger or equal than ramp value i.e. bigger or equal than `ramp` /1000

Body raw

```
[
    {
        "color": "0;100;50",
        "ramp": 1000,
        "time": 1
    },
        {
        "color": "FF000000",
        "ramp": 1000,
        "time": 1
    }
```

```
]
```

POST Start Light Effect

http://[Strip IP]/api/v1/effect/start/0

Starts the light effect. You need to set the light effect first by using the request listed above. Does not return anything.

The last number indicates which effect we are starting.

POST Stop Light effect

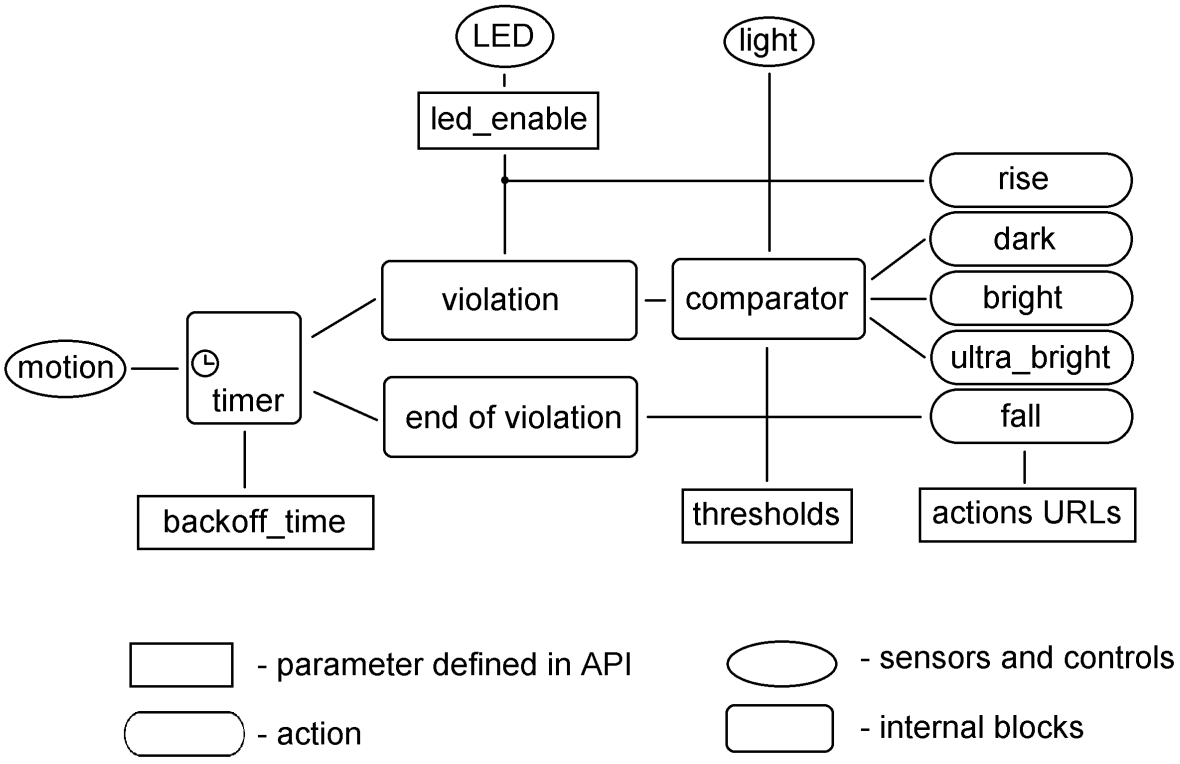
http://[Strip IP]/api/v1/effect/stop

Stops any currently ongoing light effect.

myStrom PIR

General API conventions

when changing settings using the POST operation, the meaning of the content type in the query does not matter. The device checks the body of the message for syntax, validates data types and ranges. In most cases of POST queries, only the HTTP status code OK 200 is returned with content type text/plain, in the case of setting failure due to bad parameters Bad Request 400 is returned. For GET queries, the device in most cases returns a JSON response with the application/json content type header set.



Block Diagram

Actions

Actions allow you to make an HTTP request on a remote host. The action will be performed after motion detection. PIR allows you to define different URLs for specific lighting levels, motion detection start and end. It is also possible to set the general action performed for all events.

The HTTP client on the device allows you to perform 4 methods of HTTP requests: GET, POST, PUT and DELETE.

In the definition of the address of the remote host (to which the request is to be sent) instead of the http scheme set get, post, put or delete depending on what HTTP method you want to perform the query.

Generic action URL syntax:

```
Plain Text

action_url := <get|post|put|delete>://[user:password@]host[:port][/path[?query]]
```

In the case of a generic action, additional query parameters are added in addition to those defined in the action address.

```
Plain Text

query := query<?|&>mac=<device_mac>&action=<numeric>&value=<light_value>
```

Name	Type	Range	Comments
mac	MAC		MAC address of PIR device
action	enum	RISE=8 FALL=9 NIGHT=14 TWILIGHT=15 DAY=16	action type given as number
value	uint	0..65535	light intensity

Get/Set actions URLs generic syntax:

```
Plain Text

GET api/v1/action

GET api/v1/action/pir/<ENUM generic|night|twilight|day|rise|fall>

POST api/v1/action/pir/<ENUM generic|night|twilight|day|rise|fall>
```

Suppose we want to perform a POST action on a remote host at the start of violation event:

```
Plain Text

target_ip := IP address or hostname of remote host (HTTP server)

device_ip := local IP address of PIR device
```

Suppose that on a remote host we define the path where events are to be sent:

```
Plain Text

/event/pir?id=1
```

Then action URL set on PIR device should look like:

```
Plain Text

post://<target_ip>/event/pir?id=1
```

taget_ip variable should be replace before action will be set.

If we have action URL then we can attach it:

```
Plain Text

curl -d "post://<target_ip>/event/pir?id=1" http://<device_ip>/api/v1/action/pir/rise
```

Now if motion will be detected PIR device send HTTP request to remote host. Request will be sent as POST with content type application/x-www-form-urlencoded. So, body include id=1.

In same way other action can be attached.

Description in which cases the action will be performed

Name	Description
generic	Call in any motion event independent of light intensity.
night	Call only if light value is bellow night threshold
twilight	Call only if light value is bellow day threshold and above night threshold
day	Call only if light value is above day threshold
rise	Call always on start motion violation independent of light value. This action
fall	Call always on end of motion violation independent of light value. This action will be called if backoff_time elapse

Actions will not be performed more often than every backoff_time. If the next violation occurs during backoff then the time will be extended, and the action will not be performed.

Features from version 3.82.45

- the actions support the http:// schema. Use of the http:// schema will execute the POST method,
- the actions support HTTPS requests. Add the letter s at the end of the protocol prefix (posts, gets, puts, deletes, https),
- the actions support multi requests for action, separate the actions URLs with ||,
- the actions recognize the type of content of request from URL syntax: end URL with ?a=1 then content type will application/x-www-form-urlencoded, end with ?{"a":1} then content type will application/json, end with ?some_string then content type will text/plain,
- the action definition limit is extended to 767 chars.

Example

Turn on the myStrom WiFi bulb if motion is detected.

Assumptions:

- the PIR IP address is 192.168.1.110 (please replace)
- the Bulb IP address is 192.168.1.120 (please replace)

The actions configuration:

Plain Text

```
# for turn on the bulb if motion detected
curl -d "post://192.168.1.120/api/v1/device/self?action=on" http://192.168.1.110/api/v1/act
# for turn off the bulb on end of motion
curl -d "post://192.168.1.120/api/v1/device/self?action=off" http://192.168.1.110/api/v1/ac
```

Important mention. The motion fall event will only be executed when there is no motion and the backoff_time has expired. So you can adjust the light bulb on time with the backoff_time parameter.

GET Get all assigned actions

http://{{pir_ip}}/api/v1/action

GET Get assignment for a selected action

http://{{pir_ip}}/api/v1/action/pir/<generic|night|twilight|day|rise|fall>

It allows you to get the address of a specific action

The last path parameter should take one of the following values: <ENUM generic | dark | bright | ultra_bright | rise | fall>

POST Setting the assignment for the selected action

http://{{pir_ip}}/api/v1/action/pir/<generic|night|twilight|day|rise|fall>

Allows you to set the address of the query to be performed for a specific action. The last path parameter should take one of the following values: <ENUM generic | dark | bright | ultra_bright | rise | fall>

Sensors

GET Get values from all sensors

http://{{pir_ip}}/api/v1/sensors

Returns measurements from all sensors in a simplified formation. This API is recommended if you plan to read data periodically from the device.

Name	Type	Description
motion	bool	true if motion is detected now
light	uint	value of light, this field must not be presented in response, if light was not measured at least once because LED is turn on this value will not be present in the result
temperature	real	value of ambient temperature

GET Get light value

http://{{pir_ip}}/api/v1/light

It allows reading data from a light sensor. In addition to lighting intensity, information is also provided about whether the reader recognizes the lighting level as night or day. In addition, the inquiry also provides information about the amount of infrared light. The success field in response will be set to false if no measurements have been taken yet (the device has been restarted and the LED is on).

Name	Type	Description
success	bool	read light value operation status, if true then light sensor response properly
intensity	uint	value of light intensity
day	bool	day/night information based on set thresholds
raw.adc0	uint	raw value of visible light from ADC
raw.adc1	uint	raw value of infrared light from ADC

GET Get motion status

http://{{pir_ip}}/api/v1/motion

The motion field will be set to true if motion is detected is still present or the backoff time has not expired.

Name	Type	Description
success	bool	status of read motion operation
motion	bool	true if motion is detected, false if not

GET Get temperature

http://{{pir_ip}}/api/v1/temperature

It allows you to read detailed information about the temperature measured and compensated by the device. Because the temperature sensor is inside the device, a static offset and a special algorithm simulating the heat capacity and thermal resistance of the device is used.

Name	Type	Description
measured	real	raw value of temperature measured by internal sensor
compensation	real	temperature compensation coefficient
compensated	real	compensated value of temperature the approximation of real ambient temperature

GET Get temperature (old API form)

http://{{pir_ip}}/temp

It allows you to read detailed information about the temperature measured and compensated by the device. Because the temperature sensor is inside the device, a static offset and a special algorithm simulating the heat capacity and thermal resistance of the device is used.

Name	Type	Description
measured	real	raw value of temperature measured by internal sensor
compensation	real	temperature compensation coefficient
compensated	real	compensated value of temperature the approximation of real ambient temperature

Motion Sensor specific settings

Settings for light thresholds

It allows you to read and change the brightness thresholds. They are mainly used in the decision-making process when reporting actions.

Name	Type	Range	Default	Required	Description
night	uint	0..65535	30	yes	sets the threshold below which motion detection will be reported for the night event
day	uint	0..65535	300	yes	sets the threshold above which motion detection will be reported for the day event

If light value is between day and night level, then motion event is reported as twilight event. The day value should be

If light value is between day and night level, then motion event is reported as twilight event. The day value should be greater than night value.

GET Get light thresholds

http://{{pir_ip}}/api/v1/settings/pir/thresholds

Returns the illumination thresholds in JSON format. The response content type is application/json.

POST Set light thresholds

http://{{pir_ip}}/api/v1/settings/pir/thresholds

It allows you to change the threshold of illuminance. The body of the message should contain the JSON object in the format as in the reply. In case of success HTTP OK 200 is returned.

Settings for motion

It allows you to update and change the parameters specific to a PIR device.

Name	Type	Range	Default	Required	Description
backoff_time	uint	1..84600	10	no	defines the minimum frequency between successive motion detections. If motion is detected during this period, this time is extended by the defined value without another motion event notification. Units are seconds
led_enable	bool		true	no	whether the LED should

GET Get settings for motion

http://{{pir_ip}}/api/v1/settings/pir

Returns the motion settings in JSON format. The response content type is application/json.

POST Set settings for motion

http://{{pir_ip}}/api/v1/settings/pir

It allows you to change the motion settings. The body of the message should contain the JSON object in the format as in the reply. In case of success HTTP OK 200 is returned.

Settings for HTTP server

It allows you to read and change mainly the parameters of the internal HTTP server

Name	Type	Range	Default	Required	Description
rest	bool		true	no	determines if representational state transfer (HTTP API) is enabled. This API includes some methods on the device not related to the WebUI interface
panel	bool		true	no	determines if the device's webui is active
referer	bool		false	no	if this option is enabled then HTTP server on

GET Get common settings

http://{{pir_ip}}/api/v1/settings

Returns the HTTP server settings in JSON format. The response content type is application/json

Returns the HTTP server settings in JSON format. The response content type is application/json.

POST Set common settings

http://{{pir_ip}}/api/v1/settings

It allows you to change the HTTP server settings. The body of the message should contain the JSON object in the format as in the reply. In case of success HTTP OK 200 is returned.

myStrom New Button Plus

This device has four buttons, a temperature and humidity sensor. The internal battery allows operation up to 2 months without recharging.
Each button has three types of presses: single press, double press, and long press.
The internal temperature and humidity sensor detects the change in value and on its basis allows you to take action, and the measurements are sent to the server.
Additional possibilities such as local storage of measurements or real-time measurements can be used while the device is permanently connected to the power supply.

The local API is available after connecting the button to the power supply.

Settings

api/v1/settings

The device has an extensive temperature and humidity compensation mechanism. In the case of temperature, cases where the button is disconnected from the charger, connected to the charger and charging are considered and separate compensation factors are assigned. The time of propagation of the change (the temperature-time curve) is also provided. There are also statically displacement values, e.g. when you know that in a given environment the temperature is always higher or lower by a given value and it does not depend on the condition of the device.

Accurate correction of these values is important especially when the button works as a thermostat, because its temperature increases during charging. Significant load of the button with requests to its local API increases the demand for processor power and this also translates into an increase in temperature measured by the internal sensor. For easier correction, the RAW values of the measurements are provided - that is, the values measured directly by the sensor inside the housing without any offsets.

Path	api/v1/settings
Method	GET
Body format	application/json, JSON object
Description	Allow to read and write general settings of device

Set/Response JSON object

Field	Format	Range	Default	Description
rest	bool			whether local HTTP REST API should be enabled
panel	bool			whether local user interface (Web UI) should be enabled
hap_disable	bool			read only (this device does not support HomeKit yet)
name	string	0..50	0 len	device name
prot_off	bool		true	HTTP API protection off
token	string	0..256		a token securing access to the local API
temp_offset	real	-30..30	0	generic

This method supports patch operation.

GET Settings get

http://{{dev_ip}}/api/v1/settings

POST Settings set

http://{{dev_ip}}/api/v1/settings

Body raw (json)

```
json

{"temp_offset":1}
```

Sensors

api/v1/sensors

Path	api/v1/sensors
Method	GET
Body format	application/json, JSON object
Description	Return status of sensors, battery, bus

JSON object

Field	Format	Description
temperature	real	Compensated temperature
humidity	real	Compensated humidity
battery.voltage	real	Battery voltage
battery.charging	bool	Is battery charging
charger.voltage	real	Bus/USB/charger voltage
charger.connected	bool	Is charger connected
temperature_compensation.raw	real	Temperature read from sensor
temperature_compensation.magnitude	real	Current magnitue of compensation
temperature_compensation.compensated	real	Compensated temperature (same value as in temperature field)
humidity_compensation.raw	real	Humidity read from sensor
humidity_compensation.magnitude	real	Current magnitude of compensation
humidity_compensation.compens	real	Compensated value

GET Read sensors

http://{{dev_ip}}/api/v1/sensors

Measurements

api/v1/meas

Path	api/v1/meas
Method	GET
Body format	application/json, array of JSON objects
Description	Returns measurements collected every measurement period (> 0)

JSON object

Field	Format	Description
time	YYYY-MM-DDTHH:MM:SSZ	
temp	real	Compensated temperature
humi	real	Compensated humidity

Up to 180 measurements are collected, then the eldest measurment are removed and new added. Calling of this API don't clear measurements.

GET Measurements read

http://{{dev_ip}}/api/v1/meas

Actions (Web hooks)

Local actions URIs

The device supports many types of actions to which you can assign webhooks. When the given action takes place, the given address of the request is executed. Using the action, the button can be set so that it controls individual devices, e.g. a single press turns on a switch relay, or sends actions to the server (e.g. you have a server that can handle the action and take further actions based on parameters). For each action, e.g. single pressing of the first button, many webhooks can be assigned, e.g. in case you want to control two devices at the same time.
The button also supports the execution of actions related to exceeding the temperature or humidity limit. When the temperature is too low or too high, the action is triggered, e.g. the case of a thermostat, when the button measures the temperature and with the use of an action it turns on or off the switch output. Same for humidity.
In the case of integrating the button with the server, after setting the generic / generic action, it periodically sends measurements to the server (default value is 12h).

Path	api/v1/actions(s)
Method	GET
Body format	application/json, JSON object
Description	Return action URI

Response as JSON object

Field	Format	Description
url	string	

Set (POST) request should be performed with plain text body containing action URI (see request syntax in PIR buttons actions).

The action address can have up to 767 characters long.

Send of empty body (string) cause erase/disable specific action.

```json
{
  "generic": {
    "generic": "",
    "single": "",
    "double": "",
    "long": "",
    "over": "",
    "under": ""
  },
  "btn1": {
    "generic": "",
```

Description of parameters added when calling general actions.

In addition to the parameters that are given in the address to be performed, in the case of generic actions, additional parameters are added to distinguish by receiving server between what the action was source. There also are passed additionally measurements.

For example we set generic action for button 2, like this: `url -i -d get://192.168.0.37:7002/test/btn2/generic' http://192.168.0.23/api/v1/action/btn2/generic`

then after press the button 2 in any pattern the bellow request are send:

```Plain Text
GET /test/btn2/generic?mac=E0E2E650AD78&action=1&bat=4.04&temp=26.23&rh=51.86 HTTP/1.1
Host: 192.168.0.37:7002
Connection: close
```

As you see additional parameters are added in request: mac, action, bat, temp, rh.

Description of parameters added in generic actions

Name	Type	Description
mac	string	device MAC address
index	uint	the ID of component triggered the action. The components IDs will be described follow.
action	uint	action what triggered the request, for example: single press. The actions numbers will be described follow.
bat	real	the current battery voltage in volts
temp	real	the current temperature in Celsius
rh	real	the current relative humidity in percents

Components IDs

Value	Component	Mentions
0	generic	if the generic/generic action is set, the battery, temp, humi values are send periodically (at 12h) at address defined in action
1	button 1	
2	button 2	
3	button 3	
4	button 4	
5	temperature sensor	the over/under temperature event caused action request
6	humidity sensor	the over/under humidity event caused action request

Actions IDs

Value	Source	Mentions
1	short/single press	
2	double press	
3	long press	
6	battery	
13	generic	case of periodically reports
28	over value	the over value caused request, temperature or humidity can be distinguished by component ID
29	under value	the under value caused request, temperature or humidity can be distinguished by component ID

GET Read all actions

http://{{dev_ip}}/api/v1/actions

GET Read all actions for component

http://{{dev_ip}}/api/v1/actions/btn2

Generic syntax

StartFragment

`http://{{dev_ip}}/api/v1/actions/`

where the component is one of: btn1, btn2, btn3, btn4, temp, humi, or generic

EndFragment

GET Read specific action

http://{{dev_ip}}/api/v1/action/temp/over

Genetic syntax

StartFragment

`http://{{dev_ip}}/api/v1/action//`

EndFragment

where action for buttons can be one of: `generic|single|double|long`

the action for temp or humi can be one of: `generic|over|under`

POST Set specific action

http://{{dev_ip}}/api/v1/action/btn2/single

Body raw

```
get://192.168.0.37/test/btn2/single
```

Realtime reports (experimental API)

api/v1/reports (experimental API)

Path	api/v1/reports
Method	GET
Body format	text/event-stream, SSE
Description	Send temperature and humidity event every 5s

Plain Text

```
event: meas
data: {"temp":25.31,"humi":24.84}
```

every 5s (no configuration parameters, filters etc.), the temperature and humidity are compensated

---

GET Measurements stream

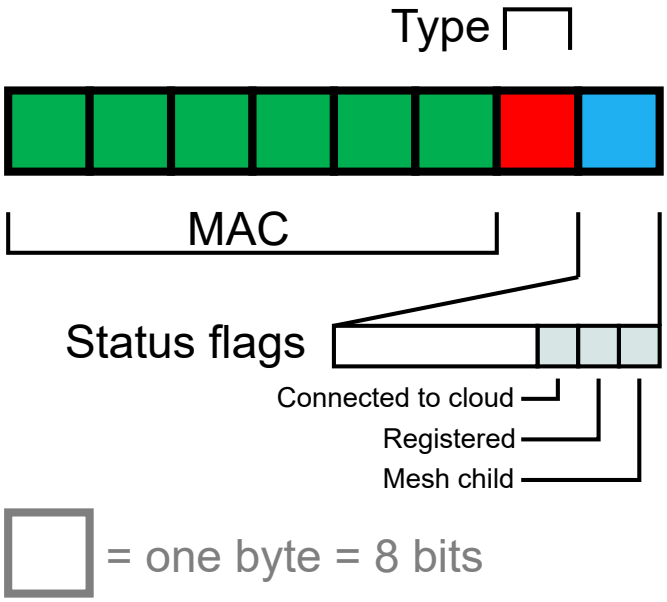
http://{{dev_ip}}/api/v1/reports

Discovery devices

Each device cyclically (every 5 seconds) sends a broadcast packet using the UDP protocol to the address 255.255.255.255 and port 7979

255.255.255.255 and port 7979.

The broadcast packet is 8 bytes and is described by the image below:



Device types

Numerical type	Device
102	Bulb
103	Button plus 1st generation
104	Button small/simple
105	LED Strip
106	Switch CH
107	Switch EU
110	Motion Sensor
112	Gateway
113	STECCO/CUBO
118	Button Plus 2nd generation
120	Switch Zero