HELIOS Processor

Public API

v24.07.0.22791
2024-07-11

HELIOS Processor - Public API

Contact:

https://www.megapixelvr.com

Document History

Date	Author	Notes
2016/11/28	CO	Initial version
2016/12/5	CO	Updated with correct JSON
2016/12/21	CB	Described preview retrieval
2017/1/4	CO	Reviewed layout
2017/2/1	CO	Refactoring of some display properties, added IP gateway settings & more info on alerts.
2017/3/11	CO	Updated after release of Sprint #20: added authentication.
2017/4/4	CO	Added more elaborate information about alerts
2017/5/31	CB	Further elaborated on alerts
2018/1/29	CB	Reformatted document
2020/2/24	SB	Updated product/company names and deprecating dev.Distro
2020/4/26	CB	dev.ingest.input is R/W
2020/7/31	CB	Correct preview URL
2021/1/5	CB	Saved configurations
2021/9/27	CB	Test patterns, additional preview sources
2022/1/29	CB	dev.display.out.adjust, dev.groups
2022/2/25	CB	dev.display.redundancy, removed dev.distros
2022/8/11	CB	Still store
2022/9/17	CB	Basic auth
2022/12/6	CB	Uploading to still store
2023/3/16	CB	Stop showing a still
2023/8/24	CB	dev.display.cctDuv
2023/9/19	CB	Blackout example
2023/12/16	CB	List of saved configurations /api/v1/presets/list
2024/5/3	CB	Brief description of HTTP

Table of Contents

1. Introduction

Megapixel's HELIOS Processors have been equipped with a ‘public API’ which allows third party systems to interface with a Processor, connected downstream distribution switches, and display devices. This document will explain how you can use this public API and will supply an overview of the possibilities to query and control a HELIOS Processor system, from an external system.

This document requires knowledge of common web technologies like URL, HTTP, JSON and REST interfaces.

2. Accessibility

The public API of a Processor can be reached using the following URL:

> http://IP_ADDRESS/api/v1/public

Where IP_ADDRESS is the IP address of the Processor (same address used to access the processor's web application).

The public API can be accessed using two protocols:

1. HTTP REST web service
2. HTTP WebSocket JSON-RPC

2.1. HTTP - Hypertext Transfer Protocol

HTTP is a request/response protocol over a TCP connection.

Port: TCP 80 (HTTP - non-encrypted)
Port: TCP 443 (HTTPS - encrypted)

- Requests are composed of a request line, followed by header fields, a blank line, and content body.
- Lines are terminated with a carriage return + line feed (\r\n).
- Refer to RFC 1945 (HTTP/1.0), RFC 9112 (HTTP/1.1), and related RFCs for more details

NOTE: Remainder of this document only specifies the METHOD and URL for API requests. While all requests are expected to comply with HTTP.

2.1.1. GET request

Request (using a GET method, public API endpoint, HTTP version 1.0, without header fields, and no body):

```
GET /api/v1/public HTTP/1.0


```

(blank line after the headers is required, and GET requests have no content body)

Response:

```
HTTP/1.1 200 OK
X-Content-Type-Options: nosniff
Cache-Control: private, no-cache, no-store, must-revalidate
Pragma: no-cache
X-RD-Uptime: 7204.7
Content-Type: application/json; charset=utf-8
Content-Length: 47545
Date: Fri, 03 May 2024 03:23:50 GMT
Connection: close

{"dev":{"display":{"blackClip  ...
```

2.1.2. POST request

```
POST /api/v1/public HTTP/1.0
Content-Type: application/json
Content-Length: 37

{"dev":{"display":{"blackout":true}}}

HTTP/1.1 200 OK
X-Content-Type-Options: nosniff
Cache-Control: private, no-cache, no-store, must-revalidate
Pragma: no-cache
Content-Type: application/json; charset=utf-8
Content-Length: 37
Date: Fri, 03 May 2024 03:30:40 GMT
Connection: close

{"dev":{"display":{"blackout":true}}}
```

2.1.3. Malformed requests

HELIOS will typically respond with a 400 status when the request is malformed.

```
malformed request

HTTP/1.1 400 Bad Request
Connection: close
```

2.1.4. Missing content-type

POST requests will respond with a 406 status when a JSON encoded content body is expected but Content-Type hasn't be set to `application/json`.

```
POST /api/v1/public HTTP/1.0
Content-Length: 37

{"dev":{"display":{"blackout":true}}}

HTTP/1.1 406 Not Acceptable
Content-Type: text/plain; charset=utf-8
Content-Length: 38
Date: Fri, 03 May 2024 03:45:29 GMT
Connection: close

Expected application/json Content-Type
```

3. REST Protocol

The REST web service uses JSON to represent the data and uses the standard HTTP methods like GET & PATCH to read or write properties.

3.1. Property interface `/api/v1/public`

3.1.1. Querying properties

```
GET /api/v1/public?[property[&...]]
```

Query	Description
_	(Optional) Ignored parameter for cache busting (e.g,. `GET /api/v1/public?_=1480466838`)
property	(Optional) Property path to query (e.g., `dev.display.brightness`)
...	(Optional) Additional property paths (e.g., `dev.display.brightness&dev.display.gamma`)

3.1.1.1. Retrieve all properties

Request:

```
GET /api/v1/public
```

Response:

```
{
    "dev": {
        "display": {
            "brightness": 50,
            "gamma": 1.5,
            "...": "..."
        }
    },
    "...": "..."
}
```

3.1.1.2. Retrieve specific properties

Request brightness & gamma:

```
GET /api/v1/public?dev.display.brightness&dev.display.gamma
```

Response:

```
{
    "dev": {
        "display": {
            "brightness": 50,
            "gamma": 2.4
        }
    }
}
```

3.1.2. Changing properties

```
PATCH /api/v1/public

{
    "path": "new value"
}
```

Query	Description
_	(Optional) Ignored parameter for cache busting (e.g,. `GET /api/v1/public?_=1480466838`)

3.1.3. Setting specific properties

Request to set brightness & gamma:

```
PATCH /api/v1/public

{"dev": {"display": {"brightness": 42, "gamma": 2.2}}}
```

Response:

```
{
    "dev": {
        "display": {
            "brightness": 42,
            "gamma": 2.200000047683716
        }
    }
}
```

3.1.4. Error conditions

Values outside of the supported range will result in the property retaining its current value

Request:

```
PATCH /api/v1/public

{"dev": {"display": {"brightness": 1000}}}
```

Response:

```
{
    "dev": {
        "display": {
            "brightness": 42
        }
    }
}
```

Unsupported or unknown properties are ignored

Request:

```
PATCH /api/v1/public

{"dev": {"display": {"invalid": true}}}
```

Response:

```
{
    "dev": {
        "display": {}
    }
}
```

3.1.5. Method not allowed

Occurs when the requested URL is malformed. When PATCHing the public API check that the URL in the request is set correctly (`/api/v1/public`).

3.1.6. Expected application/json Content-Type

Most HELIOS' API endpoints expect JSON formatted bodies. HTTP headers should include a 'content-type' header with the value 'application/json'.

3.2. Preview interface `/api/v1/preview`

A preview of the currently selected input is available using the following URL:

> http://{IP_ADDRESS}/api/v1/preview/preview

Where IP_ADDRESS is the IP address of the Processor (same address used to access the web application of the Processor).

A GET request for `preview` will return a scaled JPEG image of the currently selected input.

- Preview images are updated periodically (e.g., every half a second)
- If there is no input or the current input is invalid the preview image will be black
- It is possible for the server to respond with a 404 in cases where there is an internal error retrieving a preview. This typically happens in certain debugging scenarios, or when one or more system components have just booted. It is recommended to retry the preview request after a five second timeout in such cases.
- Previews will return a 403 in cases where the remote preview security settings are enabled (from HELIOS' front panel)

3.2.1. Sources

A array of available preview sources can be retrieved via:

> http://{IP_ADDRESS}/api/v1/preview

Example response:

```
["preview","hdmi","sdi1"]
```

Where a specific source can be retrievable via:

> http://{IP_ADDRESS}/api/v1/preview/{SOURCE}

For example:
`http://{IP_ADDRESS}/api/v1/preview/sdi1`

3.2.2. MJPEG

Clients may request a motion JPEG stream:

> http://{IP_ADDRESS}/api/v1/preview/{SOURCE}?mjpeg

JPEG frames are then delivered to the requesting client every half a second.

It is recommended to use the MJPEG stream when possible. As an example; the Web Application will always attempt to use the MJPEG stream. If the MJPEG stream fails, then the (static) previews will be retrieved every three seconds.

3.3. Saved configurations `/api/v1/presets`

3.3.1. List of available configurations (names)

Available: **HELIOS 23.12.1**

A list of the currently saved configurations (AKA presets) is retrievable via:

```
GET /api/v1/presets/list
```

Which responds with a collection of records, for example:

```
{
    "presets": [
        {
            "id": 3,
            "presetName": "Global Settings",
            "createdAt": "2021-01-05T20:36:42.227Z",
            "updatedAt":"2021-01-05T20:36:42.227Z"
        }
    ]
}
```

This collection does not include the underlying configuration data ("filterSettings" & "settings"), making this much more efficient when only the available configuration names is desired.

3.3.2. List of available configurations (full)

A complete list of the currently saved configurations (AKA presets) along with all of the associated data is retrievable via:

```
GET /api/v1/presets
```

Which responds with a collection of records, for example:

```
{
    "presets": [
        {
            "id": 3,
            "presetName": "Global Settings",
            "filterSettings": "[\"brightness\",\"cct\",\"gamma\",\"test-pattern\",\"blackout\"]",
            "settings":"{ ... complete state ... }",
            "createdAt": "2021-01-05T20:36:42.227Z",
            "updatedAt":"2021-01-05T20:36:42.227Z"
        }
    ]
}
```

Where each record contains:

Name	Type	Information
id	Number	Record's id
presetName	String	Saved configuration's name
filterSettings	String	JSON encoded array of settings types that were saved
settings	String	JSON dumped collection of complete system state
createdAt	String	Time when this record was created (in server time)
updatedAt	String	Time when this record was last updated (in server time)

3.3.3. Applying a saved configuration (by id)

Saved configurations may be applied via their numerical id:

```
POST /api/v1/presets/:id/apply
```

Where a successful response will return the applied state.

3.3.4. Applying a saved configuration (by presetName)

Saved configurations may also be applied by presetName:

```
POST /api/v1/presets/apply

{
    "presetName": "{presetName}"
}
```

Where a successful response will return the applied state.

Note: That the first matching preset will be used in the event that there are multiple saved configurations sharing the same presetName.

3.3.5. Example - Applying the saved configuration by id (id = 3)

```
POST /api/v1/presets/3/apply
```

Which responded with:

```
{
    "dev": {
        "display": {
            "brightness": 5.7,
            "cct": 6504,
            "gamma": 2.4,
            "blackout": false
        },
        "ingest": {
            "testPattern": {
                "enabled": false
            }
        }
    }
}
```

Applying an invalid saved configuration will return 404:

```
POST /api/v1/presets/4/apply
```

```
Not Found
```

3.3.6. Example - Apply saved configuration by name (presetName = "Global Settings")

```
POST /api/v1/presets/apply

{
    "presetName": "Global Settings"
}
```

Which responded with:


```
{
    "dev": {
        "display": {
            "brightness": 5.7,
            "cct": 6504,
            "gamma": 2.4,
            "blackout": false
        },
        "ingest": {
            "testPattern": {
                "enabled": false
            }
        }
    }
}
```

3.4. Still store `/api/v1/media`

3.4.1. List available stills

A list of the currently stored stills is retrievable via:

```
GET /api/v1/media
```

Which responds with a collection of records. For example:

```
{
    "media": [
        {
            "id": 26,
            "name": "VGA-no-signal-image.jpeg",
            "media": "525ee11685561e223d65018d170820184b4e284a0c128be5919f043dd117eac1.jpeg",
            "details": "{}",
            "createdAt": "2022-07-15T04:12:44.156Z",
            "updatedAt": "2022-07-15T04:12:44.156Z"
        }
    ]
}
```

Where each record contains:

Name	Type	Information
id	Number	Record's id
name	String	Displayed name for this still
media	String	Internal filename
details	Object / null	String encoded JSON with additional details (e.g., starred)
createdAt	String	Time when this record was created (in server time)
updatedAt	String	Time when this record was last updated (in server time)

3.4.2. Show a still (by id)

Stills may be shown via their numerical id:

```
POST /api/v1/media/:id/show
```

3.4.3. Show a still (by name)

Stills may also be shown by name:

```
POST /api/v1/media/show

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

Note: That the first matching still will be used in the event that there are multiple stills sharing the same name.

3.4.4. Stop showing a still

```
POST /api/v1/media/hide
```

or show a null name:

```
POST /api/v1/media/show

{
    "name": null
}
```

3.4.5. Uploading a still

Stills can be uploaded via POSTing a multipart/form-data with a single "file" value containing the image. For example:

```
POST /api/v1/media
...
Content-Type: multipart/form-data; boundary=------------------------7ae41882d286e32b

--------------------------7ae41882d286e32b
Content-Disposition: form-data; name="file"; filename="103.png"
Content-Type: image/png
...
```

Curl equivalent:

```
curl http://helios-ip/api/v1/media -F file=@103.png
```

Which responds with a record:

```
{
    "type": "media",
    "id": 75,
    "attributes": {
        "id": 75,
        "name": "103.png",
        "media": "3678f3fbc7cd7c3ba83da5e33848a43912aef6affbc9255bffd287d40d32b839.png",
        "updatedAt": "2022-11-28T19:46:48.329Z",
        "createdAt": "2022-11-28T19:46:48.329Z"
    }
}
```

3.4.6. Example - Showing a still by id (id = 26)

```
POST /api/v1/media/26/show
```

Which responded with a 204 (No Content)

Attempts at showing an invalid still will return 404:

```
POST /api/v1/media/9999/show
```

```
Not Found
```

3.4.7. Example - Show still by name (name = "VGA-no-signal-image.jpeg")

```
POST /api/v1/media/show

{
    "name": "VGA-no-signal-image.jpeg"
}
```

Which responded with 204 (No Content)

3.5. More info

REST clients for testing:

- Postman - https://www.getpostman.com

More information on REST web services:

- http://www.restapitutorial.com
- https://en.wikipedia.org/wiki/Representational_state_transfer

4. JSON-RPC Protocol

The JSON-RPC service also uses HTTP web sockets to call methods on the web server. A big advantage of the Web Sockets is that the web server can notify clients when a value has changed.

Direct WebSocket URL for the JSON-RPC protocol is:

> ws://IP_ADDRESS/api/v1/public/rpc/websocket

The calls are described in JSON like below:

4.1. Retrieve all properties

Request all properties:

```
{
    "jsonrpc": "2.0",
    "id": 123,
    "method": "state"
}
```

Response:

```
{
    "jsonrpc": "2.0",
    "id": 123,
    "result": {
        "dev": {
            "display": {
                "brightness": 50,
                "gamma": 1.5,
                "...": "..."
            }
        },
        "...": "..."
    }
}
```

4.2. Retrieve a specific property

With the JSON-RPC protocol it is current not possible to query individual properties. Rather, you can use the state method to get all properties at once, and you will then automatically get notified every time a property changes.

4.3. Set a specific property

Request:

```
{
    "jsonrpc": "2.0",
    "id": 123,
    "method": "set",
    "params": {
        "dev": {"display": {"brightness": 42}}
    }
}
```

Response:

```
{
    "jsonrpc": "2.0",
    "id": 123,
    "result": {
        "dev": {"display": {"brightness": 42}}
    }
}
```

4.3.1. Error conditions

Values outside of the supported range will result in the property retaining its current value

Request:

```
{
    "jsonrpc": "2.0",
    "id": 123,
    "method": "set",
    "params": {
        "dev": {"display": {"brightness": 1000}}
    }
}
```

Response:

```
{
    "jsonrpc": "2.0",
    "id": 123,
    "result": {
        "dev": {"display": {"brightness": 42}}
    }
}
```

Unsupported/Unknown properties are ignored

Request:

```
{
    "jsonrpc": "2.0",
    "id": 123,
    "method": "set",
    "params": {
        "dev": {"invalid": true}
    }
}
```

Response:

```
{
    "jsonrpc": "2.0",
    "id": 123,
    "result": {}
}
```

4.4. Property change notification

Once you are connected to the web service, you will automatically get notifications for all property changes, even when you have not (yet) retrieved the current state of the system. There is no need to subscribe to receive the updates.

Notifications can be for individual properties (like display brightness in case the brightness is changed), or for multiple properties at once (for instance receivercards' x & y properties when all tiles have moved).

Response:

```
{
    "jsonrpc": "2.0",
    "method": "update",
    "params": {
        "dev": {"display": {"brightness": 100}}
    }
}
```

4.5. Ping method

The ping method can be used to test the connection with the web service.

Request:

```
{
    "jsonrpc": "2.0",
    "id": 123,
    "method": "ping"
}
```

Response:

```
{
    "jsonrpc": "2.0",
    "id": 123,
    "result": "pong"
}
```

4.6. More info

More information on the JSON-RPC protocol:

- http://json-rpc.org/
- https://en.wikipedia.org/wiki/JSON-RPC

5. Authentication

HELIOS supports basic access authentication & digest access authentication to authenticate access to its web server. When authentication is enabled, it is enabled for all API access, including the public API.

For authentication you'll need:

- Username: user name as defined in the HELIOS web application
- Password: password as defined in the HELIOS web application

with digest based authentication additionally requiring (at a minimum):

- Realm: always "M8 Processor"
- Nonce: nonce value supplied by the server (in the HTTP header)

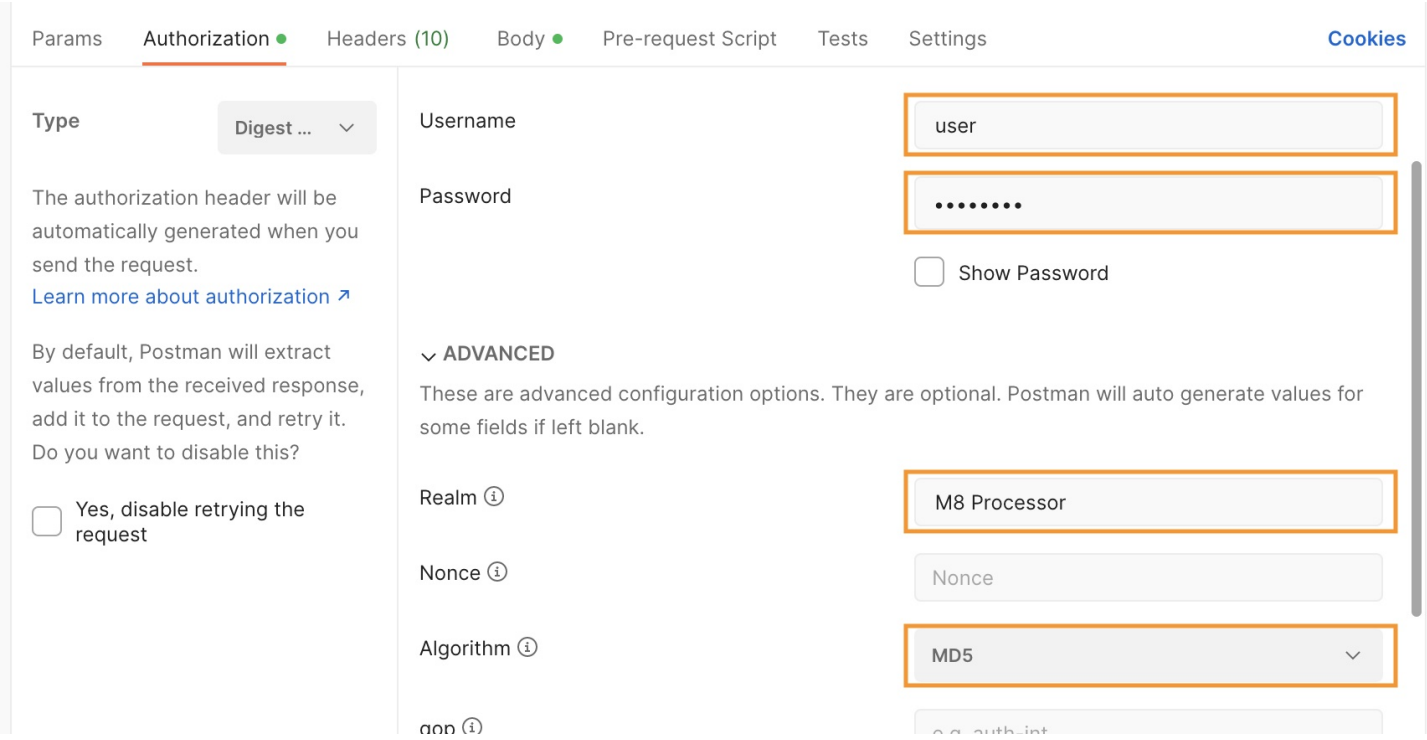
Please read RFC 2617 for more information about the digest access based authentication and how it works.

In cases where network security is a concern; please use https with a trusted certificate to protect credentials.

5.1. Example of a GET request with basic access authentication, using Curl

```
curl user:password@http://{IP_ADDRESS}/api/v1/public
```

5.2. Example of a GET request with digest access authentication, using Postman



5.3. More info

- Basic Access Authentication (Wikipedia): https://en.wikipedia.org/wiki/Basic_access_authentication
- Digest Access Authentication (Wikipedia): https://en.wikipedia.org/wiki/Digest_access_authentication
- RFC 2617: Basic and Digest Access Authentication; https://tools.ietf.org/html/rfc2617

6. Data model (JSON)

The JSON model contains 2 root objects:

- `dev` : device specific properties of the processor and display devices
- `sys` : overall system settings that are specific to a system configuration.

Each of these root objects will be explained in following sections.

6.1. `dev` object (devices)

The dev object keeps track of the device settings and properties.

6.1.1. `dev.display`

This object contains the output or display settings of the HELIOS Processor, including:

Name	Type	R/W	Information
blackout	Property	R/W	Black-out status of the display (as boolean)
brightness	Property	R/W	The brightness level of the display (in percentage)
cct	Property	R/W	Color temperature (in Kelvin)
cctDuv	Property	R/W	Color temperature Duv
freeze	Property	R/W	Freeze status of the display (as boolean)
gamma	Property	R/W	The gamma correction value
gamut	Object	R	Display gamut details (ideal = widest, target = desired)
out	Object	R/W	Output adjustments
redundancy	Object	R/W	Redundancy information + controls
tilesCount	Property	R	The total number of tiles in the display
tilesInfo	Property	R	Information about the tiles in the display
width	Property	R/W	width of the output window
height	Property	R/W	height of the output window
x	Property	R/W	x position of the output window
y	Property	R/W	y position of the output window

6.1.1.1. `dev.display.out`

Name	Type	R/W	Information
adjust	Object	R/W	Output adjustments

6.1.1.2. `dev.display.out.adjust`

Name	Type	R/W	Information
enabled	Property	R/W	Output adjustments enabled
gain	Object	R/W	RGB gain
gamma	Object	R/W	RGB gamma
lift	Object	R/W	RGB lift
offset	Object	R/W	RGB offset
saturation	Number	R/W	Output saturation

6.1.1.3. `dev.display.redundancy`

Name	Type	R/W	Information
info	Property	R	Processor's overall redundancy state (e.g., "Fail over - Main (Active)")
mode	Property	R/W	Redundancy mode (none/failover/seamless)
processors	Object	R	Detected processors organized by UUID
role	Property	R/W	Redundancy role (main/backup/offline)
state	Property	R/W	Redundancy state (active/mixed/standby). Set to "main"/"backup" to initiate appropriate flip

6.1.2. `dev.groups`

dev.groups object is a dictionary containing all the defined groups within the system. Groups are sequentially indexed but may contain gaps if groups have been removed.

For example: `dev.groups.0`

These group indices are referenced by receivers `.groupId` which defines which group a receiver is a member of.

Name	Type	R/W	Information
blackout	Property	R/W	Group's black-out status
gains	Object	R/W	RGBI gains
mask.enabled	Boolean	R/W	Enable/disable group's mask
mask.l/t/b/r	Number	R/W	Mask edge offset
name	Property	R/W	User defined name for this group
testPattern.enabled	Boolean	R/W	Enable group's test pattern
testPattern.r/g/b/a	Number	R/W	Group's test pattern color

6.1.3. `dev.ingest`

The dev.ingest object contains the properties for the Processor.

Name	Type	R/W	Information
alerts	Object	R	List of alerts
alertsCount	Property	R	Number of alerts (in alerts)
alertsSeverity	Property	R	Lowest alert severity (-1 when there are no alerts)
info	Object	R	General information like serial number and software/firmware version
input	Property	R/W	The selected input
testPattern.enabled	Boolean	R/W	Enable/Disable test pattern
testPattern.motion	Boolean	R/W	Enable/Disable test pattern motion
testPattern.type	String	R/W	Test pattern type (e.g., red, grid, etc)

6.1.4. `dev.receivers`

dev.receivers object contains a dictionary with all the discovered receivers connected to the processor. Receivers are a general term to describe the receiving end of the video data, so these are the display devices or LED panels.

The keys in the dictionary are the MAC addresses of the receivers (display devices).

For example an object for the LED panel with MAC address '58:98:6f:00:01:7d':

> `dev.receivers.58:98:6f:00:01:7d`

Name	Type	R/W	Information
alerts	Object	R	List of alerts
alertsCount	Property	R	Number of alerts (in alerts)
alertsSeverity	Property	R	Lowest alert severity (-1 when there are no alerts)
distroBoxMac	Property	R	MAC address of the HELIOS output port to which this receiver is connected
distroBoxPort	Property	R	HELIOS output port to which this receiver is connected
groupId	Property	R/W	Index of group receiver is a member of (-1 = no group)
info	Object	R	General information like name, type, serial number and software/firmware version
switchId	Property	R	Key of associated parent switch
switchPort	Property	R	Switch output port to which this receiver is connected
x	Property	R/W	x position of this receiver, on the processor's canvas
y	Property	R/W	y position of this receiver, on the processor's canvas

6.2. `sys` object (system)

The system object contains general system settings that are not specific to a device.

Name	Type	R/W	Information
alerts	Object	R	List of alerts
alertsCount	Property	R	Number of alerts (in alerts)
alertsSeverity	Property	R	Lowest alert severity (-1 when there are no alerts)
description	Property	R/W	The name of the Processor (as displayed on the LCD screen and on the web application)
info	Object	R	General information like name, type, serial number and software/firmware version
ipAddress	Property	R	Current IP address of the Processor
ipDns	Property	R	Current primary DNS address
ipGateway	Property	R	Current IP gateway
ipSubnet	Property	R	Current IP subnet mask

7. System health

To get the overall health status, look at the alerts properties on the `sys` object.

More detailed alerts are also available at the different device levels (`dev.ingest` , `dev.display` , `dev.switches.[SWITCH]` , and `dev.receivers.[RECEIVER]`).

7.1. Overall health status

An indication of the overall system's health is available via `sys.alertsCount` & `sys.alertsSeverity` .

REST Example

Request health overview:

```
GET /api/v1/public?sys.alertsCount&sys.alertsSeverity
```

Response:

```
{
    "sys":{
        "alertsCount": 2,
        "alertsSeverity": 3
    }
}
```

Informs that the system currently has two alerts, with at least one of those alerts having a severity of Error.

7.1.1. System level alerts

Request to gather system level alerts:

```
GET /api/v1/public?sys.alerts
```

Response:


```
{
    "sys":{
        "alerts":{
            "fanSlow":{
                "brief": "Slow fan",
                "count": 2,
                "desc": "Below normal operating speed",
                "devices":{
                    "receivers":{
                        "58:98:6f:00:0b:7a": ""
                    }
                },
                "severity": 4,
                "since": 6922
            },
            "tempHigh":{
                "brief": "Over temp",
                "count": 1,
                "desc": "Exceeds normal operating temperature",
                "devices": {
                    "receivers":{
                        "58:98:6f:00:0b:7a": ""
                    },
                },
                "severity": 3,
                "since": 9874
            }
        }
    }
}
```


Shows our two alerts are a slow fan and a device is over temperature. Our slow fan has a severity of warning, while the over temperature is considered an error.

7.2. Alerts

Alerts are (temporary) messages to notify about a certain unexpected condition of the system like failing fans, over temperature, exceeded port capacity, ...

Alerts will occur on two levels:

- Device level (`dev.*.alert`): specific alerts for each individual device. This can occur for ingests, switches, receivers, and display.
- System level (`sys.alert`): overview of all the current alerts, system wide.

One or more alerts on one or more devices will be aggregated by the Processor into system level alerts. Because of this, it is best to look for the alerts on the system level and if more information or detail is needed to refer to the alert details of each specific device.

7.2.1. Severity

Each alert type has an associated severity level. These levels correspond to the well-known syslog (RFC5424) severity levels which range from (0) Emergency to (7) Debug. To reduce confusion the Processor uses a subset of these severity levels, while maintaining the severity level numbering.

Value	Severity	Description
2	Critical	Critical conditions
3	Error	Error conditions
4	Warning	Warning conditions
5	Notice	Normal but significant condition

7.2.2. Possible alert sources

Overview of possible alerts and their meaning:

Alert Key	Source	Severity	Description
invalidCal	Receiver	Notice	A panel's calibration data is missing or the data is invalid
fanSlow	Device	Warning	One or more fans inside the panel's cabinet is not working properly
frameError	Receiver	Notice	Video frame error occured (frame drop)
pixelCap	Receiver	Error	Pixel capacity exceeded; a receiver card could not be added to the chain because the total pixel capacity of the processor/switch has been exceeded, although the capacity of the switch port on which the receiver card is connected has not been exceeded
pixelClamp	Receiver	Notice	Pixel clipping or clamping has occurred because of bandwidth issues
portCap	Device	Error	Port capacity exceeded; in case of a receiver card alert, it means the receiver card could not be added to the chain. In case of a switch alert, it means one or more receiver cards could not be added to a switch port because the capacity has been exceeded.
safeMode	Receiver	Notice	Receiver card is in safe mode. In safe mode, the panel's functionality has been limited; the panel can be mapped but no video will be displayed.
tempHigh	Device	Error	The temperature of the device or one of its component has exceeded the maximum value

7.2.3. Since

`since` property reports the time at which an alert was first asserted. This time is measured in seconds since the Processor was started. It is up to the requester to convert this time point to a more meaningful representation.

To aid with the uptime conversion, each HTTP API response from the Processor includes the current Processor uptime in an 'X-RD-Uptime' HTTP response header.

Request:

```
GET /api/v1/public?sys.alerts
```

Response:

```
HTTP/1.1 200 OK
Cache-Control: private, no-cache, must-revalidate
Pragma: no-cache
X-RD-Uptime: 14232
Content-Type: application/json; charset=utf-8
Content-Length: 1948
Vary: Accept-Encoding
Date: Thu, 01 Jun 2017 00:28:49 GMT

{
    "sys": {
        "alerts": {
            "tempHigh":{
                "brief": "Over temp",
                "count": 1,
                "desc": "Exceeds normal operating temperature",
                "devices": {
                    "receivers": {
                        "58:98:6f:00:0b:7a": ""
                    },
                },
                "severity": 3,
                "since": 9874
            }
        }
    }
}
```

At the time of this example request our Processor has been running for nearly four hours (X-RD-Uptime).

> X-RD-Uptime: 14232

Further down in the response there's an over temperature condition which was asserted at 9874 (since).

> "since": 9874

Time of response reception was roughly 5:28pm May 31st 2017 PDT.

> X-RD-Uptime – since = 14232 – 9874 = 4358 seconds ago

Meaning our alert first occurred at:

> 2017-05-31 17:28:00 – 4358 seconds => 4:15:22pm May 31st 2017 PDT (an hour ago from local time)

Note: Using uptime for the time base allows for events to be tied to a monotonic clock. This removes reliance on an accurate server clock, as there are many situations where the Processor may report time incorrectly (lack of Internet connectivity, flat battery, etc). As such it is discouraged to rely on the server Date response header.

7.2.4. Example - Over temperature alert (receiver card)

Condition: PSU and HUB board temperatures of an LED panel are over the maximum ratings

```
{
    "dev": {
        "receivers": {
            "RECEIVER_MAC_ADDRESS": {
                "alerts": {
                    "tempHigh": {
                        "desc":"",
                        "devices": {
                            "temps": { "CPU": "" }
                        }
                    }
                },
                "alertsCount": 1
            }
        }
    },
    "sys": {
        "alerts": {
            "tempHigh": {
                "desc":"",
                "devices": {
                    "receivers": {"RECEIVER_MAC_ADDRESS": ""}
                }
            }
        },
        "alertsCount": 1
    }
}
```

7.2.5. Example - Over temperature alert (Processor)

Condition: PSU temperature of the processor is over its maximum rating


```
{
    "dev": {
        "ingest": {
            "alerts": {
                "overTemp": {
                "desc":"",
                    "devices": {
                        "PSU": ""
                    }
                }
            }
        }
    },
    "sys": {
        "alerts": {
            "overTemp": {
                "desc": "",
                "devices": {
                    "ingest": ""
                }
            }
        },
        "alertsCount": 1
    }
}
```

7.2.6. Example - Port capacity (switch + receiver)

Condition: an LED panel could not be mapped because the Switch port capacity has been exceeded (~ port full).


```
{
    "dev": {
        "switches": {
            "SWITCH_MAC_ADDRESS": {
                "alerts": {
                    "portCap": {
                        "desc": "",
                        "devices": {
                            "ports": { "3": "" }
                        }
                    }
                },
                "alertsCount": 1
            }
        },
        "receivers": {
            "RECEIVER_MAC_ADDRESS": {
                "alerts": {
                    "portCap": {
                        "desc":""
                    }
                },
                "alertsCount": 1
            }
        }
    }
    "sys": {
        "alerts": {
            "portCap": {
                "desc": "",
                "devices": {
                    "receivers": {"RECEIVER_MAC_ADDRESS": ""},
                    "switches": {"SWITCH_MAC_ADDRESS": ""}
                }
            }
        },
        "alertsCount": 2
    }
}
```

8. Appendix

8.1. Data operations overview

Operation	REST	JSON-RPC
Reading property values	GET	"method": "state"
Writing property values	PATCH	"method": "set"
Performing actions	n/a	"method": "ping"
Alert / notifications	n/a	"method": "update"

8.2. Example - Blackout

8.2.1. Blackout

REST

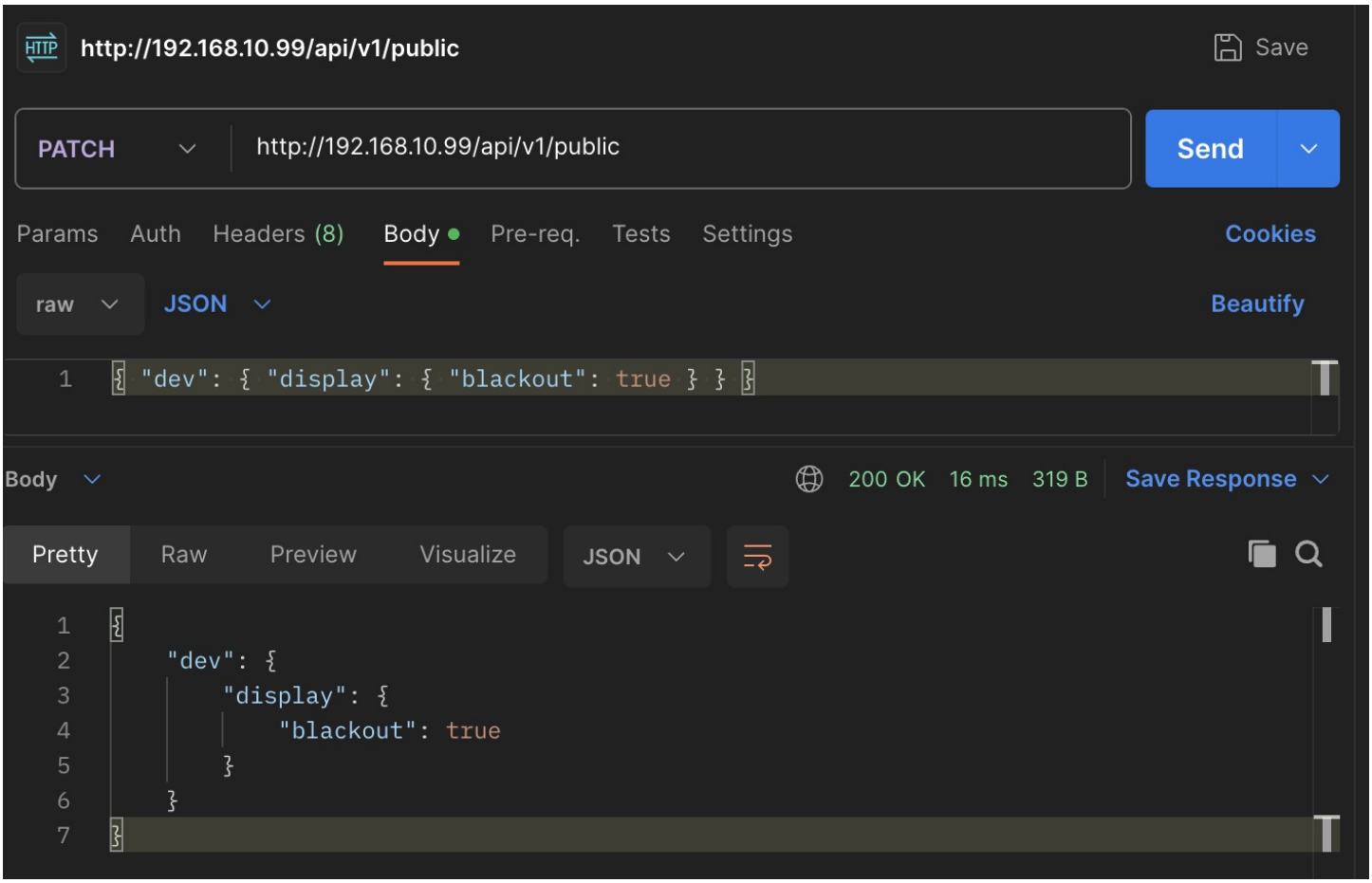
Request:

```
PATCH /api/v1/public
content-type: application/json

{ "dev": { "display": { "blackout": true } } }
```

Response:

```
{"dev": {"display": {"blackout": true}}}
```



8.2.2. Un-blackout

REST

Request:

```
PATCH /api/v1/public
content-type: application/json

{ "dev": { "display": { "blackout": false } } }
```

Response:

```
{"dev": {"display": {"blackout": false}}}
```

8.3. Example - Control screen brightness

8.3.1. Get screen brightness

REST

Request:

```
GET /api/v1/public?dev.display.brightness
```

Response:

```
{"dev": {"display": {"brightness": 28.200000762939453}}}
```

JSON-RPC

It is not possible to get an individual or specific property with JSON-RPC. You should get the current state using the 'state' method and then listen for updates every time the value changes.

Request:

```
{
    "jsonrpc": "2.0",
    "id": 100,
    "method": "state"
}
```

Response:

```
{
    "jsonrpc":"2.0",
    "id":100,
    "result": {"dev": {"display": {"brightness":28.200000762939453, "gamma":1.5}}, ... }}
}
```

Notification (async):

```
{
    "jsonrpc": "2.0",
    "method": "update",
    "params": {"dev": {"display": {"brightness": 33.29999923706055}}}
}
```

8.3.2. Set screen brightness

REST

Request:

```
PATCH /api/v1/public

{"dev": {"display": {"brightness": 50}}}
```

Response:

```
{"dev": {"display": {"brightness": 50}}}
```

JSON-RPC

Request:

```
{
    "jsonrpc": "2.0",
    "id": 101,
    "method": "set",
    "params": {"dev": {"display": {"brightness": 50}}}
}
```

Response:

```
{
    "jsonrpc": "2.0",
    "id": 101,
    "result":{"dev": {"display": {"brightness": 50}}}
}
```

8.4. Example - Control screen gamma

8.4.1. Get screen gamma

REST

Request:

```
GET /api/v1/public?dev.display.gamma
```

Response:

```
{"dev": {"display": {"gamma": 2.4}}}
```

JSON-RPC

It is not possible to get an individual or specific property with JSON-RPC. You should get the current state using the 'state' method and then listen for updates every time the value changes.

Request:

```
{
    "jsonrpc": "2.0",
    "id": 103,
    "method": "state"
}
```

Response:

```
{
    "jsonrpc": "2.0",
    "id": 103,
    "result": {"dev": {"display": {"brightness": 28.200000762939453, "gamma": 1.5}}, ... }}
}
```

Notification (async):

```
{
    "jsonrpc": "2.0",
    "method": "update",
    "params": {"dev": {"display": {"gamma": 2}}}
}
```

8.4.2. Set screen gamma

REST

Request:

```
PATCH /api/v1/public

{"dev": {"display": {"gamma": 2.1}}}
```

Response:

```
{"dev": {"display": {"gamma": 2.1}}}
```

JSON-RPC

Request:

```
{
    "jsonrpc": "2.0",
    "id": 104,
    "method": "set",
    "params": {"dev": {"display": {"gamma": 2.1}}}
}
```

Response:

```
{
    "jsonrpc": "2.0",
    "id": 104,
    "result": {"dev": {"display": {"gamma": 2.0999999046325684}}}
}
```

8.5. Example - Control test patterns

8.5.1. Enable color bars test pattern

REST

Request:

```
PATCH /api/v1/public

{
    "dev": {
        "ingest": {
            "testPattern": {
                "enabled": true,
                "type": "colorBars"
            }
        }
    }
}
```

Response:

```
{
    "dev": {
        "ingest": {
            "testPattern": {
                "enabled": true,
                "type": "colorBars"
            }
        }
    }
}
```

8.5.2. Disable test pattern

REST

Request:

```
PATCH /api/v1/public

{
    "dev": {
        "ingest": {
            "testPattern": {
                "enabled": false
            }
        }
    }
}
```

Response:

```
{
    "dev": {
        "ingest": {
            "testPattern": {
                "enabled": false
            }
        }
    }
}
```

8.6. Example - Redundancy flip to main/backup

8.6.1. Flip to main

REST

Request:

```
PATCH /api/v1/public

{
    "dev": {
        "display": {
            "redundancy": {
                "state": "main"
            }
        }
    }
}
```

Response:

```
{
    "dev": {
        "display": {
            "redundancy": {
                "state": "active"
            }
        }
    }
}
```

8.6.2. Flip to backup

REST

Request:

```
PATCH /api/v1/public

{
    "dev": {
        "display": {
            "redundancy": {
                "state": "backup"
            }
        }
    }
}
```

Response:

```
{
    "dev": {
        "display": {
            "redundancy": {
                "state": "active"
            }
        }
    }
}
```