# MIDRA™ 4K

# REST API Programmer's Guide

**For version v1.00.03**


ANALOG WAY®

*Pioneer in Analog, Leader in Digital*

# Table of contents

# 1. Presentation

## 1.1. Description

The REST API for Midra™ 4K is a simple way for you to automate your interaction with the Midra™ 4K presentation systems. The REST API for Midra™ 4K is RESTful and HTTP-based. Basically, this means that the communication is made through normal HTTP requests.

## 1.2. Server address

The base server address is: http://<ipadress>/api/tpp/v1 where <ipadress> is the IP address of the Midra™ 4K presentation system.

## 1.3. HTTP Requests

HTTP requests can be made with tons of tools, each modern programming language has its own HTTP functions and libraries. The REST API for Midra™ 4K will handle each request in a meaningful manner, depending on the action required.

| Method | Usage |
|---|---|
| GET | For simple retrieval of information about your screens, multiviewers, etc.., you should use the GET method. API will respond you with a JSON object. Using the returned information, you can eventually form additional requests. All the GET requests are made read-only, which means making a GET requests cannot change the state of any information stored on the Midra™ 4K presentation system. |
| POST | When you want to change an object property or trigger an action on the Midra™ 4K presentation system, you should choose POST method. The POST request includes all of the attributes necessary to change the desired object property or trigger the action. |

## 1.4. HTTP Statuses

When you make a request to the API, you will get a response including the data you want with standard HTTP statuses, including error codes.

In case of an unusual event, such as trying to recall a preset memory index that does not exist on the Midra™ 4K presentation system, the status code will have an error code. Besides that, the body of the request will contain additional information about the event to provide you the most conventional way to fix the flow. To make it clear, status codes are usually in between 2XX-4XX range.

| Code | Message | Description |
|---|---|---|
| 200 | OK | Successful operation (with content) |
| 204 | No Content | Successful operation (no content) |
| 400 | Bad Request | Syntax invalid |
| 404 | Not Found | The specified resource could not be found |
| 405 | Method Not Allowed | The specified request method is not allowed |
| 409 | Internal Server Error | Error reported by the server |

## 1.5. HTTP Responses

For each successful and unsuccessful request, a JSON-formatted response body will be sent back. If you make a request for a single object, say, for a screen, the resource root will be a single object containing the data you requested. If you request a collection, say, a group of screens, response body will contain a collection.

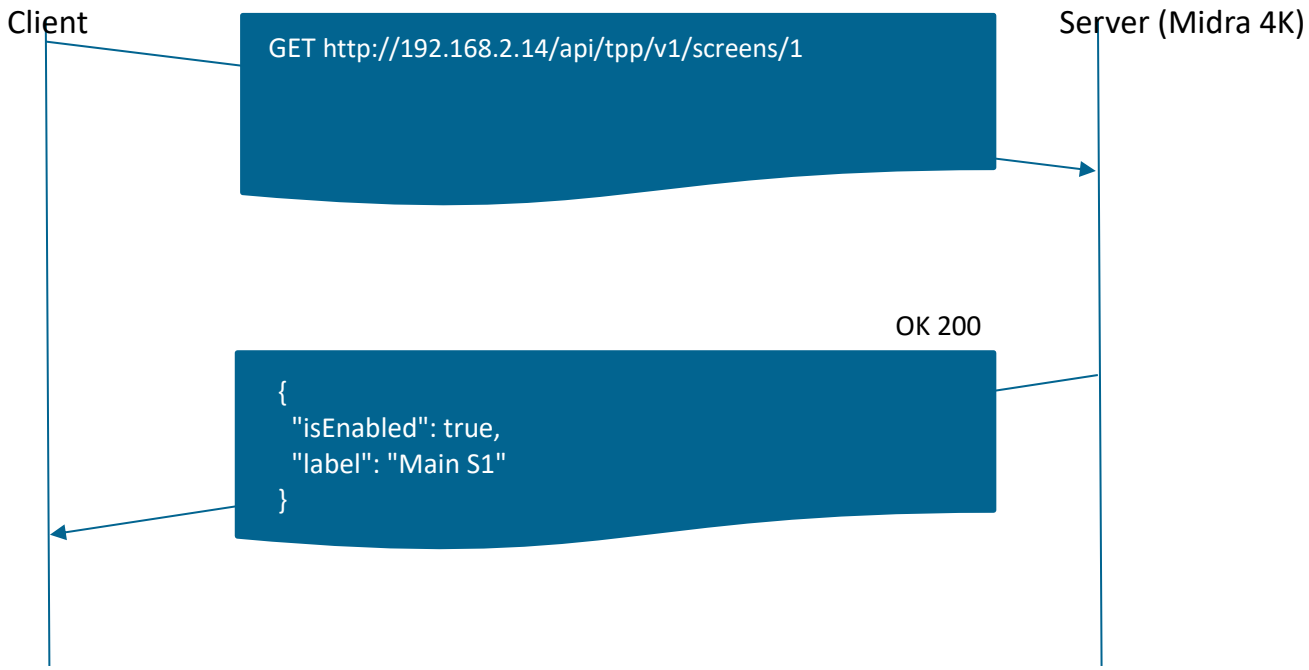
## 1.6. HTTP Parameters

Most of HTTP GET requests need query string parameters.

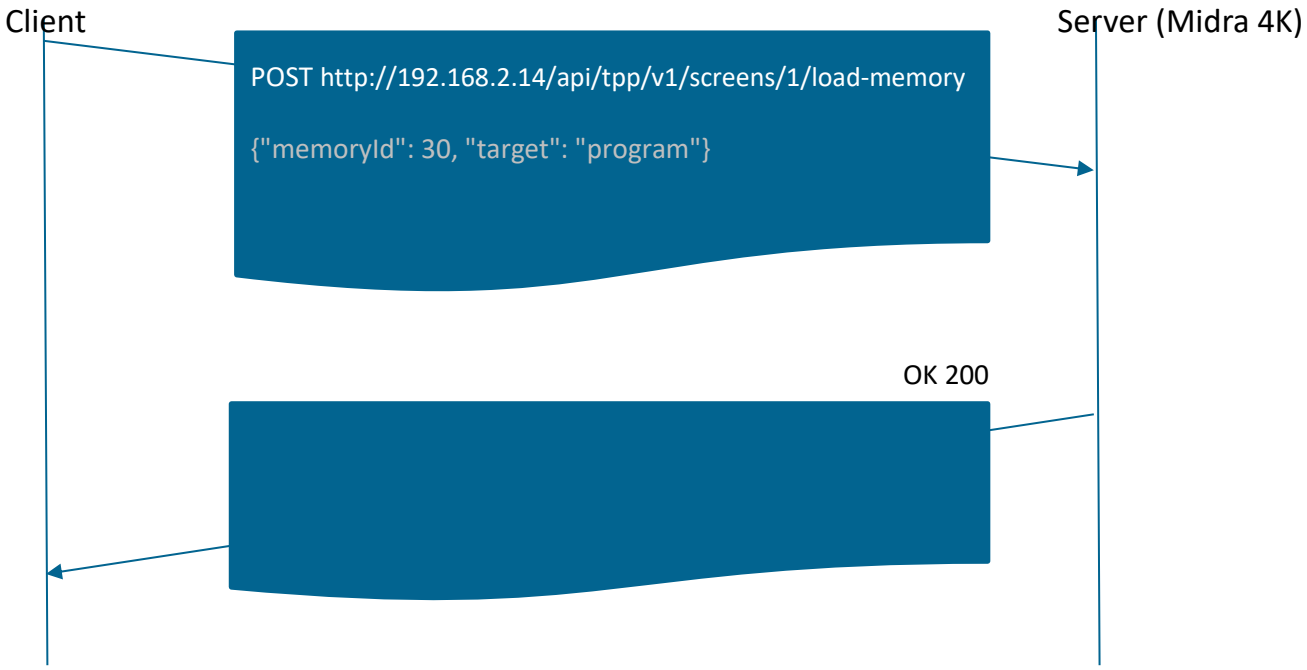
Most of HTTP POST requests need POST body parameters.

For all API requests, we provide examples calls with .NET C# command and raw TCP. With a single copy and paste, you can always try making a request and see the results.

## 1.7. GET request diagram



## 1.8. POST request diagram



# 2. System commands

## 2.1. Reading system information

**GET** /api/tpp/v1/system

**Response**

produces: application/json

| Name | Type | Description |
|---|---|---|
| type | string | the type of Midra 4K device: ‘QVU 4K’, ‘QMX 4K’, ‘PLS 4K’ or ‘EKS 4K’ |
| label | string | the device label |
| version | json | a JSON object containing the current firmware version (see below) |

**Field ‘version’**

produces: application/json

| Name | Type | Description |
|---|---|---|
| major | integer | firmware major version number |
| minor | integer | firmware minor version number |
| patch | integer | firmware patch version number |
| beta | boolean | true is the firmware version is a beta version, false if not |

**Response example**

```
{
  "type": "PLS 4K",
  "label": "Meeting Room",
  "version": {
    "major": 1,
    "minor": 0,
    "patch": 23,
    "beta": false
  }
}
```

## Example: Read system information

### Raw TCP socket (connected on port 80 of 192.168.2.140)

```
GET /api/tpp/v1/system HTTP/1.1<CR><LF><CR><LF>
```

<CR> replace with Carriage Return ASCII code: 13 (0x0D) <LF> replace with Line Feed ASCII code: 10 (0x0A)

### C#

```
var httpWebRequest = (HttpWebRequest)WebRequest.Create("http://192.168.2.140/api/tpp/v1/system");

var httpResponse = (HttpWebResponse) httpWebRequest.GetResponse();
using (var streamReader = new StreamReader(httpResponse.GetResponseStream()))
{
   var responseText = streamReader.ReadToEnd();
}
```

## 2.2. Rebooting the system

**POST** /api/tpp/v1/system/reboot

**Example: Reboot the system**

**Raw TCP socket (connected on port 80 of 192.168.2.140)**

```
POST /api/tpp/v1/system/reboot HTTP/1.1<CR><LF><CR><LF>
```

<CR> replace with Carriage Return ASCII code: 13 (0x0D) <LF> replace with Line Feed ASCII code: 10 (0x0A)

**C#**

```
var httpWebRequest = (HttpWebRequest)WebRequest.Create("http://192.168.2.140/api/tpp/v1/system/reboot");
httpWebRequest.ContentType = "application/json";
httpWebRequest.Method = "POST";

using (var streamWriter = new StreamWriter(httpWebRequest.GetRequestStream()))
{
   streamWriter.Write(“”);
   streamWriter.Flush();
}
```

## 2.3. Shutting down the system

**POST** /api/tpp/v1/system/shutdown

### Body

consumes: application/json

| Name | Type | Optional | Description |
|---|---|---|---|
| standby | boolean | No | true to activate the standby mode |

**Example: Shutdown the system**

### Raw TCP socket (connected on port 80 of 192.168.2.140)

```
POST /api/tpp/v1/system/shutdown HTTP/1.1<CR><LF>Content-Type:
application/json<CR><LF>Content-Length: 17<CR><LF><CR><LF>{"standby": true}
```

<CR> replace with Carriage Return ASCII code: 13 (0x0D) <LF> replace with Line Feed ASCII code: 10 (0x0A)

Important: the *Content-Length* header field value must contain a decimal number representing the number of bytes found in the payload of the message {"standby": true}

### C#

```
var httpWebRequest = (HttpWebRequest)WebRequest.Create("http://192.168.2.140/api/tpp/v1/system/shutdown");
httpWebRequest.ContentType = "application/json";
httpWebRequest.Method = "POST";

using (var streamWriter = new StreamWriter(httpWebRequest.GetRequestStream()))
{
   string json = new JavaScriptSerializer().Serialize(new { standby = "true" });
   streamWriter.Write(json);
}
```

## 2.4. Waking up the system

| POST | /api/tpp/v1/system/wakeup |
|---|---|

**Example: Wake up the system**

**Raw TCP socket (connected on port 80 of 192.168.2.140)**

```
POST /api/tpp/v1/system/wakeup HTTP/1.1<CR><LF><CR><LF>
```

<CR> replace with Carriage Return ASCII code: 13 (0x0D) <LF> replace with Line Feed ASCII code: 10 (0x0A)

**C#**

```
var httpWebRequest = (HttpWebRequest)WebRequest.Create("http://192.168.2.140/api/tpp/v1/system/wakeup");
httpWebRequest.ContentType = "application/json";
httpWebRequest.Method = "POST";

using (var streamWriter = new StreamWriter(httpWebRequest.GetRequestStream()))
{
   streamWriter.Write(“”);
   streamWriter.Flush();
}
```

# 3. Screen commands

## 3.1. Reading screen information

**GET** /api/tpp/v1/screens/{screenId}

### Request

| Name | Type | Description |
|---|---|---|
| screenId | integer | the screen number (from 1 to 2) |

### Response

produces: application/json

| Name | Type | Description |
|---|---|---|
| isEnabled | boolean | true if the screen is enabled, false if not |
| label | string | the screen label |
| layerMode | string | the layer mode (“mixing layers” or “split layers”) |

### Response example

```
{
  "isEnabled": true,
  "label": "Center"
  "layerMode": "mixing"
}
```

**Example: Read screen 2 information**

### Raw TCP socket (connected on port 80 of 192.168.2.140)

```
GET /api/tpp/v1/screens/2 HTTP/1.1<CR><LF><CR><LF>
```

<CR> replace with Carriage Return ASCII code: 13 (0x0D) <LF> replace with Line Feed ASCII code: 10 (0x0A)

### C#

```
var httpWebRequest = (HttpWebRequest)WebRequest.Create("http://192.168.2.140/api/tpp/v1/screens/2");

var httpResponse = (HttpWebResponse) httpWebRequest.GetResponse();
using (var streamReader = new StreamReader(httpResponse.GetResponseStream()))
{
   var responseText = streamReader.ReadToEnd();
}
```

## 3.2. Recalling a preset from memory to a single screen

**POST** /api/tpp/v1/screens/{screenId}/load-memory

### Request

| Name | Type | Description |
|---|---|---|
| screenId | integer | the screen number (from 1 to 2) |

### Body

consumes: application/json

| Name | Type | Optional | Description |
|---|---|---|---|
| memoryId | integer | No | the memory index (from 1 to 200) |
| target | string | Yes | the destination (“program” or “preview”). Default is “preview” |

**Example: Recall preset 30 to screen 2 (Preview)**

### Raw TCP socket (connected on port 80 of 192.168.2.140)

```
POST /api/tpp/v1/screens/2/load-memory HTTP/1.1<CR><LF>Content-Type: application/json<CR><LF>Content-Length: 37<CR><LF><CR><LF>{"memoryId": 30, "target": "preview"}
```

<CR> replace with Carriage Return ASCII code: 13 (0x0D) <LF> replace with Line Feed ASCII code: 10 (0x0A)

Important: the *Content-Length* header field value must contain a decimal number representing the number of bytes found in the payload of the message {"memoryId": 30, "target": "preview"}

### C#

```
var httpWebRequest = (HttpWebRequest)WebRequest.Create("http://192.168.2.140/api/tpp/v1/screens/2/load-memory");
httpWebRequest.ContentType = "application/json";
httpWebRequest.Method = "POST";

using (var streamWriter = new StreamWriter(httpWebRequest.GetRequestStream()))
{
   string json = new JavaScriptSerializer().Serialize(new { memoryId = 30, target = "preview" });
   streamWriter.Write(json);
}
```

## 3.4. Recalling a master preset from memory

**POST** /api/tpp/v1/load-master-memory

### Body

consumes: application/json

| Name | Type | Optional | Description |
|---|---|---|---|
| memoryId | integer | No | the memory index (from 1 to 100) |
| target | string | Yes | the destination (“program” or “preview”). Default is “preview” |

**Example: Recall master preset 10 to Preview**

### Raw TCP socket (connected on port 80 of 192.168.2.140)

```
POST /api/tpp/v1/load-master-memory HTTP/1.1<CR><LF>Content-Type:
application/json<CR><LF>Content-Length: 37<CR><LF><CR><LF>{"memoryId": 10, "target":
"preview"}
```

<CR> replace with Carriage Return ASCII code: 13 (0x0D) <LF> replace with Line Feed ASCII code: 10 (0x0A)

Important: the *Content-Length* header field value must contain a decimal number representing the number of bytes found in the payload of the message {"memoryId": 10, "target": "preview"}

### C#

```
var httpWebRequest = (HttpWebRequest)WebRequest.Create("http://192.168.2.140/api/tpp/v1/load-master-
memory");
httpWebRequest.ContentType = "application/json";
httpWebRequest.Method = "POST";

using (var streamWriter = new StreamWriter(httpWebRequest.GetRequestStream()))
{
   string json = new JavaScriptSerializer().Serialize(new { memoryId = 10,  target = "preview" });
   streamWriter.Write(json);
}
```

## 3.5. Reading a live layer status

**GET** /api/tpp/v1/screens/{screenId}/live-layers/{layerId}/presets/{target}

### Request

| Name | Type | Description |
|---|---|---|
| screenId | integer | the screen number (from 1 to 2) |
| layerId | integer | the layer number (from 1 to 4) |
| target | string | the destination (“program” or “preview”) |

### Response

produces: application/json

| Name | Type | Description |
|---|---|---|
| status | string | the layer status: "off", "open", "close", "cross", "flying", "flying depth" |
| sourceType | string | the type of source: "none", "color" or "input" |
| sourceId | integer | the source number |

### Response example

```
{
  "status": "open",
  "sourceType": "input",
  "sourceId": 8
}
```

**Example: Read layer 4 current status on screen 1 preview**

### Raw TCP socket (connected on port 80 of 192.168.2.140)

```
GET /api/tpp/v1/screens/1/live-layers/4/presets/preview HTTP/1.1<CR><LF><CR><LF>
```

<CR> replace with Carriage Return ASCII code: 13 (0x0D) <LF> replace with Line Feed ASCII code: 10 (0x0A)

### C#

```
var httpWebRequest = (HttpWebRequest)WebRequest.Create("http://192.168.2.140/api/tpp/v1/screens/1/live-layers/4/presets/preview");
var httpResponse = (HttpWebResponse) httpWebRequest.GetResponse();
using (var streamReader = new StreamReader(httpResponse.GetResponseStream()))
{
   var responseText = streamReader.ReadToEnd();
}
```

## 3.6. Setting a live layer source

**POST** /api/tpp/v1/screens/{screenId}/live-layers/{layerId}/presets/{target}/source

### Request

| Name | Type | Description |
|---|---|---|
| screenId | integer | the screen number (from 1 to 2) |
| layerId | Integer | the layer number (from 1 to 4) |
| target | string | the destination (“program” or “preview”) |

### Body

consumes: application/json

| Name | Type | Optional | Description |
|---|---|---|---|
| sourceType | string | No | the type of source: "none", "color", "input" |
| sourceId | integer | No | the source number |

**Example: Set screen 1 layer 3 source to live input 3 (Preview)**

### Raw TCP socket (connected on port 80 of 192.168.2.140)

```
POST /api/tpp/v1/screens/1/live-layers/3/presets/preview/source HTTP/1.1<CR><LF>Content-Type: application/json<CR><LF>Content-Length: 38<CR><LF><CR><LF>{"sourceType": "input", "sourceId": 3}
```

<CR> replace with Carriage Return ASCII code: 13 (0x0D) <LF> replace with Line Feed ASCII code: 10 (0x0A)

Important: the *Content-Length* header field value must contain a decimal number representing the number of bytes found in the payload of the message {"sourceType": "input", "sourceId": 3}

### C#

```
var httpWebRequest = (HttpWebRequest)WebRequest.Create("http://192.168.2.140/api/tpp/v1/screens/1/live-layers/3/presets/preview/source");
httpWebRequest.ContentType = "application/json";
httpWebRequest.Method = "POST";

using (var streamWriter = new StreamWriter(httpWebRequest.GetRequestStream()))
{
   string json = new JavaScriptSerializer().Serialize(new { sourceType = "input", sourceId = 3 });
   streamWriter.Write(json);
}
```

## 3.7. Reading background layer status

**GET** /api/tpp/v1/screens/{screenId}/background-layer/presets/{target}

### Request

| Name | Type | Description |
|---|---|---|
| screenId | integer | the screen number (from 1 to 2) |
| target | string | the destination (“program” or “preview”) |

### Response

produces: application/json

| Name | Type | Description |
|---|---|---|
| status | string | The layer status: "off", "open", "close" |
| sourceType | string | the type of source: "background-set" or "none" |
| sourceId | integer | the background set number (from 1 to 8) |

### Response example

```
{
  "status": "open",
  "sourceType": "background-set",
  "sourceId": 2
}
```

**Example: Read background layer status on screen 1 preview**

### Raw TCP socket (connected on port 80 of 192.168.2.140)

```
GET /api/tpp/v1/screens/1/background-layer/presets/preview HTTP/1.1<CR><LF><CR><LF>
```

<CR> replace with Carriage Return ASCII code: 13 (0x0D) <LF> replace with Line Feed ASCII code: 10 (0x0A)

### C#

```
var httpWebRequest =
(HttpWebRequest)WebRequest.Create("http://192.168.2.140/api/tpp/v1/screens/1/background-
layer/presets/preview");

var httpResponse = (HttpWebResponse) httpWebRequest.GetResponse();
using (var streamReader = new StreamReader(httpResponse.GetResponseStream()))
{
   var responseText = streamReader.ReadToEnd();
}
```

## 3.8. Setting a background layer source

**POST** /api/tpp/v1/screens/{screenId}/background-layer/presets/{target}/source

### Request

| Name | Type | Description |
|---|---|---|
| screenId | integer | the screen number (from 1 to 2) |
| target | string | the destination (“program” or “preview”) |

### Body

consumes: application/json

| Name | Type | Optional | Description |
|---|---|---|---|
| sourceType | string | No | the type of source: "background-set" or "none" |
| sourceId | integer | No | the background source number (from 1 to 8) |

**Example: Set screen 2 background to background set 3 (Preview)**

### Raw TCP socket (connected on port 80 of 192.168.2.140)

```
POST /api/tpp/v1/screens/2/background-layer/presets/preview/source HTTP/1.1<CR><LF>Content-Type: application/json<CR><LF>Content-Length: 47<CR><LF><CR><LF>{"sourceType": "background-set", "sourceId": 3}
```

<CR> replace with Carriage Return ASCII code: 13 (0x0D) <LF> replace with Line Feed ASCII code: 10 (0x0A)

Important: the *Content-Length* header field value must contain a decimal number representing the number of bytes found in the payload of the message {"sourceType": "background-set", "sourceId": 3}

### C#

```
var httpWebRequest =
(HttpWebRequest)WebRequest.Create("http://192.168.2.140/api/tpp/v1/screens/2/background-layer/presets/preview/source");
httpWebRequest.ContentType = "application/json";
httpWebRequest.Method = "POST";

using (var streamWriter = new StreamWriter(httpWebRequest.GetRequestStream()))
{
   string json = new JavaScriptSerializer().Serialize(new { sourceType = "background-set", sourceId = 3 });
   streamWriter.Write(json);
}
```

## 3.9. Reading foreground layer status

**GET** /api/tpp/v1/screens/{screenId}/foreground-layer/presets/{target}

### Request

| Name | Type | Description |
|---|---|---|
| screenId | integer | the screen number (from 1 to 2) |
| target | string | the destination (“program” or “preview”) |

### Response

produces: application/json

| Name | Type | Description |
|---|---|---|
| status | string | The layer status: "off", "open", "close" |
| sourceType | string | the type of source: "foreground-image" or "none" |
| sourceId | integer | the foreground preset number (from 1 to 4) |

### Response example

```
{
  "status": "open",
  "sourceType": "foreground-image",
  "sourceId": 2
}
```

**Example: Read foreground layer status on screen 1 preview**

### Raw TCP socket (connected on port 80 of 192.168.2.140)

```
GET /api/tpp/v1/screens/1/foreground-layer/presets/preview HTTP/1.1<CR><LF><CR><LF>
```

<CR> replace with Carriage Return ASCII code: 13 (0x0D) <LF> replace with Line Feed ASCII code: 10 (0x0A)

### C#

```
var httpWebRequest =
(HttpWebRequest)WebRequest.Create("http://192.168.2.140/api/tpp/v1/screens/1/foreground-
layer/presets/preview");

var httpResponse = (HttpWebResponse) httpWebRequest.GetResponse();
using (var streamReader = new StreamReader(httpResponse.GetResponseStream()))
{
   var responseText = streamReader.ReadToEnd();
}
```

## 3.10. Setting a foreground layer source

**POST** /api/tpp/v1/screens/{screenId}/foreground-layer/presets/{target}/source

**Request**

| Name | Type | Description |
|---|---|---|
| screenId | integer | the screen number (from 1 to 2) |
| target | string | the destination (“program” or “preview”) |

**Body**

consumes: application/json

| Name | Type | Optional | Description |
|---|---|---|---|
| sourceType | string | No | the type of source: "foreground-image" or "none" |
| sourceId | integer | No | the foreground source number (from 1 to 4) |

**Example: Set screen 2 foreground to foreground preset 3 (Preview)**

**Raw TCP socket (connected on port 80 of 192.168.2.140)**

```
POST /api/tpp/v1/screens/2/foreground-layer/presets/preview/source HTTP/1.1<CR><LF>Content-Type: application/json<CR><LF>Content-Length: 49<CR><LF><CR><LF>{"sourceType": "foreground-image", "sourceId": 3}
```

<CR> replace with Carriage Return ASCII code: 13 (0x0D) <LF> replace with Line Feed ASCII code: 10 (0x0A)

Important: the *Content-Length* header field value must contain a decimal number representing the number of bytes found in the payload of the message {"sourceType": "foreground-image", "sourceId": 3}

**C#**

```
var httpWebRequest =
(HttpWebRequest)WebRequest.Create("http://192.168.2.140/api/tpp/v1/screens/2/foreground-layer/presets/preview/source");
httpWebRequest.ContentType = "application/json";
httpWebRequest.Method = "POST";

using (var streamWriter = new StreamWriter(httpWebRequest.GetRequestStream()))
{
   string json = new JavaScriptSerializer().Serialize(new { sourceType = "foreground-image", sourceId = 3 });
   streamWriter.Write(json);
}
```

## 3.11. Single TAKE: Transitioning the Preview content to the Program (single screen)

**POST** /api/tpp/v1/screens/{screenId}/take

**Request**

| Name | Type | Description |
|---|---|---|
| screenId | integer | the screen number (from 1 to 2) |

**Example: TAKE screen 2**

**Raw TCP socket (connected on port 80 of 192.168.2.140)**

```
POST /api/tpp/v1/screens/2/take HTTP/1.1<CR><LF><CR><LF>
```

<CR> replace with Carriage Return ASCII code: 13 (0x0D) <LF> replace with Line Feed ASCII code: 10 (0x0A)

**C#**

```
var httpWebRequest = (HttpWebRequest)WebRequest.Create("http://192.168.2.140/api/tpp/v1/screens/2/take");
httpWebRequest.ContentType = "application/json";
httpWebRequest.Method = "POST";

using (var streamWriter = new StreamWriter(httpWebRequest.GetRequestStream()))
{
   streamWriter.Write(“”);
}
```

## 3.12. Global TAKE: Transitioning the Preview content to the Program (multiple screens)

**POST** /api/tpp/v1/take

**Body**

consumes: application/json

| Name | Type | Optional | Description |
|---|---|---|---|
| screenIds | list | No | list of screen indexes that will be transitioned |
| auxiliaryScreenIds | list | No | list of auxiliary screen indexes that will be transitioned |

**Example: Take screen 1 and auxiliary screen 1**

**Raw TCP socket (connected on port 80 of 192.168.2.140)**

```
POST /api/tpp/v1/take HTTP/1.1<CR><LF>Content-Type: application/json<CR><LF>Content-Length: 46<CR><LF><CR><LF>{"screenIds": [1], "auxiliaryScreenIds ": [1]}
```

<CR> replace with Carriage Return ASCII code: 13 (0x0D) <LF> replace with Line Feed ASCII code: 10 (0x0A)

Important: the *Content-Length* header field value must contain a decimal number representing the number of bytes found in the payload of the message {"screenIds": [1], "auxiliaryScreenIds ": [1]}

**C#**

```
var httpWebRequest = (HttpWebRequest)WebRequest.Create("http://192.168.2.140/api/tpp/v1/take");
httpWebRequest.ContentType = "application/json";
httpWebRequest.Method = "POST";

using (var streamWriter = new StreamWriter(httpWebRequest.GetRequestStream()))
{
   var screenIdList = new List<int>() { 1 };
   var auxIdList = new List< int>() { 1 };
   string json = new JavaScriptSerializer().Serialize(new { screenIds = screenIdList, auxiliaryScreenIds = auxIdList });
   streamWriter.Write(json);
}
```

# 4. Auxiliary screen commands

## 4.1. Reading auxiliary screen information

**GET** /api/tpp/v1/auxiliary-screens/{auxId}

### Request

| Name | Type | Description |
|---|---|---|
| auxId | integer | the auxiliary screen number (always 1, depending of the current mode) |

### Response

produces: application/json

| Name | Type | Description |
|---|---|---|
| IsEnabled | boolean | true is the auxiliary screen is enabled, false if not |
| label | string | the auxiliary screen label |

### Response example

```
{
  "isEnabled": true,
  "label": "DSM"
}
```

**Example: Read auxiliary screen 1 information**

### Raw TCP socket (connected on port 80 of 192.168.2.140)

```
GET /api/tpp/v1/auxiliary-screen/1 HTTP/1.1<CR><LF><CR><LF>
```

<CR> replace with Carriage Return ASCII code: 13 (0x0D) <LF> replace with Line Feed ASCII code: 10 (0x0A)

### C#

```
var httpWebRequest = (HttpWebRequest)WebRequest.Create("http://192.168.2.140/api/tpp/v1/auxiliary-screen/1");

var httpResponse = (HttpWebResponse) httpWebRequest.GetResponse();
using (var streamReader = new StreamReader(httpResponse.GetResponseStream()))
{
   var responseText = streamReader.ReadToEnd();
}
```

## 4.2. Recalling a preset from memory to a single auxiliary screen

**POST** /api/tpp/v1/auxiliary-screens/{auxId}/load-memory

**Request**

| Name | Type | Description |
|---|---|---|
| auxId | integer | the auxiliary screen number (always 1, depending of the current mode) |

**Body**

consumes: application/json

| Name | Type | Optional | Description |
|---|---|---|---|
| memoryId | integer | No | the memory index |
| target | string | Yes | the destination (“program” or “preview”). Default is “preview” |

**Example: Recall preset 5 to auxiliary screen 1 (Preview)**

**Raw TCP socket (connected on port 80 of 192.168.2.140)**

```
POST /api/tpp/v1/auxiliary-screens/1/load-memory HTTP/1.1<CR><LF>Content-Type:
application/json<CR><LF>Content-Length: 15<CR><LF><CR><LF>{"memoryId": 5}
```

<CR> replace with Carriage Return ASCII code: 13 (0x0D) <LF> replace with Line Feed ASCII code: 10 (0x0A)

Important: the *Content-Length* header field value must contain a decimal number representing the number of bytes found in the payload of the message {"memoryId": 5}

**C#**

```
var httpWebRequest = (HttpWebRequest)WebRequest.Create("http://192.168.2.140/api/tpp/v1/auxiliary-
screens/1/load-memory");
httpWebRequest.ContentType = "application/json";
httpWebRequest.Method = "POST";

using (var streamWriter = new StreamWriter(httpWebRequest.GetRequestStream()))
{
   string json = new JavaScriptSerializer().Serialize(new { memoryId = 5 });
   streamWriter.Write(json);
}
```

## 4.3. Reading the source of an auxiliary screen

**GET** /api/tpp/v1/auxiliary-screens/{auxId}/background-layer/presets/{target}/source

### Request

| Name | Type | Description |
|---|---|---|
| auxId | integer | the auxiliary screen number (always 1, depending of the current mode) |
| target | string | the destination (“program” or “preview”) |

### Response

produces: application/json

| Name | Type | Description |
|---|---|---|
| status | string | the layer status: "off", "open", "close" |
| sourceType | string | the type of source: "none", "input" or "screen" |
| sourceId | integer | the source number |

### Response example

```
{
  "status": "open",
  "sourceType": "input",
  "sourceId": 5
}
```

**Example: Read auxiliary screen 1 source (Preview)**

### Raw TCP socket (connected on port 80 of 192.168.2.140)

```
GET api/tpp/v1/auxiliary-screens/1/presets/preview/source HTTP/1.1<CR><LF><CR><LF>
```

<CR> replace with Carriage Return ASCII code: 13 (0x0D) <LF> replace with Line Feed ASCII code: 10 (0x0A)

### C#

```
var httpWebRequest = (HttpWebRequest)WebRequest.Create("http://192.168.2.140/api/tpp/v1/auxiliary-screens/1/presets/preview/source ");

var httpResponse = (HttpWebResponse) httpWebRequest.GetResponse();
using (var streamReader = new StreamReader(httpResponse.GetResponseStream()))
{
   var responseText = streamReader.ReadToEnd();
}
```

## 4.4. Setting the source of an auxiliary screen

**POST** /api/tpp/v1/auxiliary-screens/{auxId}/background-layer/presets/{target}/source

### Request

| Name | Type | Description |
|---|---|---|
| auxId | integer | the auxiliary screen number (always 1, depending of the current mode) |
| target | string | the destination (“program” or “preview”) |

### Body

consumes: application/json

| Name | Type | Optional | Description |
|---|---|---|---|
| sourceType | string | No | the type of source: "none", "input" or "screen" |
| sourceId | integer | No | the source number (from 1 to x) |

**Example: Set the source of auxiliary screen 1 to input 5 (Preview)**

### Raw TCP socket (connected on port 80 of 192.168.2.140)

```
POST /api/tpp/v1/auxiliary-screens/1/presets/preview/source HTTP/1.1<CR><LF>Content-Type: application/json<CR><LF>Content-Length: 38<CR><LF><CR><LF>{"sourceType": "input", "sourceId": 5}
```

<CR> replace with Carriage Return ASCII code: 13 (0x0D) <LF> replace with Line Feed ASCII code: 10 (0x0A)

Important: the *Content-Length* header field value must contain a decimal number representing the number of bytes found in the payload of the message {"sourceType": "input", "sourceId": 5}

### C#

```
var httpWebRequest = (HttpWebRequest)WebRequest.Create("http://192.168.2.140/api/tpp/v1/auxiliary-screens/1/presets/preview/source");
httpWebRequest.ContentType = "application/json";
httpWebRequest.Method = "POST";

using (var streamWriter = new StreamWriter(httpWebRequest.GetRequestStream()))
{
   string json = new JavaScriptSerializer().Serialize(new { sourceType = "inputs", sourceId = 5 });
   streamWriter.Write(json);
}
```

## 4.5. TAKE: Transitioning the Preview content to the Program (single auxiliary screen)

**POST** /api/tpp/v1/auxiliary-screens/{auxId}/take

**Request**

| Name | Type | Description |
|---|---|---|
| auxId | integer | the auxiliary screen number (from 1 to 1) |

**Example: Take auxiliary screen 1**

**Raw TCP socket (connected on port 80 of 192.168.2.140)**

```
POST /api/tpp/v1/auxiliary-screens/1/take HTTP/1.1<CR><LF><CR><LF>
```

<CR> replace with Carriage Return ASCII code: 13 (0x0D) <LF> replace with Line Feed ASCII code: 10 (0x0A)

**C#**

```
var httpWebRequest = (HttpWebRequest)WebRequest.Create("http://192.168.2.140/api/tpp/v1/auxiliary-screens/1/take");
httpWebRequest.ContentType = "application/json";
httpWebRequest.Method = "POST";

using (var streamWriter = new StreamWriter(httpWebRequest.GetRequestStream()))
{
   streamWriter.Write(“”);
}
```

# 5. Multiviewer commands

## 5.1. Reading multiviewer output information

**GET** /api/tpp/v1/multiviewer/

**Response**

produces: application/json

| Name | Type | Description |
|---|---|---|
| isEnabled | boolean | true is the multiviewer output is enabled, false if not |
| label | string | the multiviewer output label |

**Response example**

```
{
  "isEnabled": true,
  "label": "MVW1"
}
```

**Example: Read multiviewer output 1 information**

**Raw TCP socket (connected on port 80 of 192.168.2.140)**

```
GET /api/tpp/v1/multiviewer/ HTTP/1.1<CR><LF><CR><LF>
```

<CR> replace with Carriage Return ASCII code: 13 (0x0D) <LF> replace with Line Feed ASCII code: 10 (0x0A)

**C#**

```
var httpWebRequest = (HttpWebRequest)WebRequest.Create("http://192.168.2.140/api/tpp/v1/multiviewer/");

var httpResponse = (HttpWebResponse) httpWebRequest.GetResponse();
using (var streamReader = new StreamReader(httpResponse.GetResponseStream()))
{
   var responseText = streamReader.ReadToEnd();
}
```

## 5.2. Recalling a preset from memory to a multiviewer output

**POST** /api/tpp/v1/multiviewer/load-memory

**Body**

consumes: application/json

| Name | Type | Optional | Description |
|---|---|---|---|
| memoryId | integer | No | the memory index (from 1 to 20) |

**Example: Recall preset 20 to multiviewer output**

**Raw TCP socket (connected on port 80 of 192.168.2.140)**

```
POST /api/tpp/v1/multiviewer/load-memory HTTP/1.1<CR><LF>Content-Type:
application/json<CR><LF>Content-Length: 16<CR><LF><CR><LF>{"memoryId": 20}
```

<CR> replace with Carriage Return ASCII code: 13 (0x0D) <LF> replace with Line Feed ASCII code: 10 (0x0A)

Important: the *Content-Length* header field value must contain a decimal number representing the number of bytes found in the payload of the message {"memoryId": 20}

**C#**

```
var httpWebRequest = (HttpWebRequest)WebRequest.Create("http://192.168.2.140/api/tpp/v1/multiviewer/load-memory");
httpWebRequest.ContentType = "application/json";
httpWebRequest.Method = "POST";

using (var streamWriter = new StreamWriter(httpWebRequest.GetRequestStream()))
{
   string json = new JavaScriptSerializer().Serialize(new { memoryId = 20 });
   streamWriter.Write(json);
}
```

## 5.3. Reading the source of a multiviewer output widget

**GET** /api/tpp/v1/multiviewer/widgets/{widgetId}/source

### Request

| Name | Type | Description |
|---|---|---|
| widgetId | integer | the widget number (from 1 to 16) |

### Response

produces: application/json

| Name | Type | Description |
|---|---|---|
| sourceType | string | the type of source: "none", "input", "screen-program", "screen-preview" or "timer" |
| sourceId | integer | the source number |

### Response example

```
{
  "sourceType": "input",
  "sourceId": 2
}
```

**Example: Read the source of the widget 10 on multiviewer output**

### Raw TCP socket (connected on port 80 of 192.168.2.140)

```
GET /api/tpp/v1/multiviewer/widgets/10/source HTTP/1.1<CR><LF><CR><LF>
```

<CR> replace with Carriage Return ASCII code: 13 (0x0D) <LF> replace with Line Feed ASCII code: 10 (0x0A)

### C#

```
var httpWebRequest =
(HttpWebRequest)WebRequest.Create("http://192.168.2.140/api/tpp/v1/multiviewer/widgets/10/source");

var httpResponse = (HttpWebResponse) httpWebRequest.GetResponse();
using (var streamReader = new StreamReader(httpResponse.GetResponseStream()))
{
   var responseText = streamReader.ReadToEnd();
}
```

## 5.4. Reading the status of a multiviewer output widget

**GET** /api/tpp/v1/multiviewer/widgets/{widgetId}

### Request

| Name | Type | Description |
|---|---|---|
| widgetId | integer | the widget number (from 1 to 16) |

### Response

produces: application/json

| Name | Type | Description |
|---|---|---|
| isEnabled | boolean | true if the widget is enabled, false if not |

### Response example

```
{
  "isEnabled": true
}
```

**Example: Read the status of the widget 10 on multiviewer output**

### Raw TCP socket (connected on port 80 of 192.168.2.140)

```
GET /api/tpp/v1/multiviewer/widgets/10 HTTP/1.1<CR><LF><CR><LF>
```

<CR> replace with Carriage Return ASCII code: 13 (0x0D) <LF> replace with Line Feed ASCII code: 10 (0x0A)

### C#

```
var httpWebRequest =
(HttpWebRequest)WebRequest.Create("http://192.168.2.140/api/tpp/v1/multiviewer/widgets/10");

var httpResponse = (HttpWebResponse) httpWebRequest.GetResponse();
using (var streamReader = new StreamReader(httpResponse.GetResponseStream()))
{
   var responseText = streamReader.ReadToEnd();
}
```

## 5.5. Setting the source of a multiviewer output widget

**POST** /api/tpp/v1/multiviewer/widgets/{widgetId}/source

### Request

| Name | Type | Description |
|---|---|---|
| widgetId | integer | the widget number (from 1 to 16) |

### Body

consumes: application/json

| Name | Type | Optional | Description |
|---|---|---|---|
| sourceType | string | No | the type of source: "none", "input", "screen-program", "screen-preview" or "timer" |
| sourceId | integer | No | the source number |

**Example: Set the source of the widget 10 to input 7 on multiviewer output**

### Raw TCP socket (connected on port 80 of 192.168.2.140)

```
POST /api/tpp/v1/multiviewer/widgets/10/source HTTP/1.1<CR><LF>Content-Type: application/json<CR><LF>Content-Length: 38<CR><LF><CR><LF>{"sourceType": "input", "sourceId": 7}
```

<CR> replace with Carriage Return ASCII code: 13 (0x0D) <LF> replace with Line Feed ASCII code: 10 (0x0A)

Important: the *Content-Length* header field value must contain a decimal number representing the number of bytes found in the payload of the message {"sourceType": "input", "sourceId": 7}

### C#

```
var httpWebRequest =
(HttpWebRequest)WebRequest.Create("http://192.168.2.140/api/tpp/v1/multiviewer/widgets/10/source");
httpWebRequest.ContentType = "application/json";
httpWebRequest.Method = "POST";

using (var streamWriter = new StreamWriter(httpWebRequest.GetRequestStream()))
{
   string json = new JavaScriptSerializer().Serialize(new { sourceType = "input", sourceId = 7 });
   streamWriter.Write(json);
}
```

# 6. Source commands

## 6.1. Reading input information

**GET** /api/tpp/v1/inputs/{inputId}

**Request**

| Name | Type | Description |
|---|---|---|
| inputId | integer | the input number (from 1 to 10) |

**Response**

produces: application/json

| Name | Type | Description |
|---|---|---|
| isEnabled | boolean | true if the input is enabled, false if not |
| label | string | the input label |
| plugType | string | The type of active plug: "HDMI", "DP", "SDI" |
| isValid | boolean | true if the input signal is valid, false if not |

**Response example**

```
{
  "isEnabled": true,
  "label":  "Cam PTZ",
  "plugType": "HDMI”,
  "isValid": true,
}
```

**Example: Read input 2 information**

**Raw TCP socket (connected on port 80 of 192.168.2.140)**

```
GET /api/tpp/v1/inputs/2 HTTP/1.1<CR><LF><CR><LF>
```

<CR> replace with Carriage Return ASCII code: 13 (0x0D)  <LF> replace with Line Feed ASCII code: 10 (0x0A)

**C#**

```
var httpWebRequest = (HttpWebRequest)WebRequest.Create("http://192.168.2.140/api/tpp/v1/inputs/2");

var httpResponse = (HttpWebResponse) httpWebRequest.GetResponse();
using (var streamReader = new StreamReader(httpResponse.GetResponseStream()))
{
   var responseText = streamReader.ReadToEnd();
}
```

## 6.2. Reading foreground image information

**GET** /api/tpp/v1/screens/{screenId}/foreground-images/{foregroundImageId}

### Request

| Name | Type | Description |
|---|---|---|
| screenId | integer | the screen number (from 1 to 2) |
| foregroundImageId | integer | the image number (from 1 to 4) |

### Response

produces: application/json

| Name | Type | Description |
|---|---|---|
| isEmpty | boolean | true if the foreground image is empty, false if not |
| isValid | boolean | true if the foreground image is valid, false if not |
| label | string | the foreground image label |
| isEnabled | boolean | Tue if the foreground image is enabled, false if not |

### Response example

```
{
  "isEmpty": false,
  "isValid": true,
  "label": “FRG4”,
  "isEnabled": true
}
```

**Example: Read foreground image 2 information for screen 1**

### Raw TCP socket (connected on port 80 of 192.168.2.140)

```
GET /api/tpp/v1/screens/1/foreground-images/2 HTTP/1.1<CR><LF><CR><LF>
```

<CR> replace with Carriage Return ASCII code: 13 (0x0D) <LF> replace with Line Feed ASCII code: 10 (0x0A)

### C#

```
var httpWebRequest = (HttpWebRequest)WebRequest.Create("http://192.168.2.140/api/tpp/v1/screens/1/foreground-images/2");
var httpResponse = (HttpWebResponse) httpWebRequest.GetResponse();
using (var streamReader = new StreamReader(httpResponse.GetResponseStream()))
{
   var responseText = streamReader.ReadToEnd();
}
```

## 6.3. Reading background image information

**GET** /api/tpp/v1/screens/{screenId}/background-images/{backgroundImageId}

### Request

| Name | Type | Description |
|---|---|---|
| screenId | integer | the screen number (from 1 to 2) |
| backgroundImageId | integer | the image number (from 1 to 4) |

### Response

produces: application/json

| Name | Type | Description |
|---|---|---|
| IsEmpty | boolean | true if the background image is empty, false if not |
| isValid | boolean | true if the background image is valid, false if not |
| label | string | the background image label |
| isEnabled | boolean | true if the background image is enabled, false if not |

### Response example

```
{
  "isEmpty": false,
  "isValid": true,
  "label": “FRG4”,
  "isEnabled": true
}
```

**Example: Read background image 2 information for screen 1**

### Raw TCP socket (connected on port 80 of 192.168.2.140)

```
GET /api/tpp/v1/screens/1/background-images/2 HTTP/1.1<CR><LF><CR><LF>
```

<CR> replace with Carriage Return ASCII code: 13 (0x0D) <LF> replace with Line Feed ASCII code: 10 (0x0A)

### C#

```
var httpWebRequest = (HttpWebRequest)WebRequest.Create("http://192.168.2.140/api/tpp/v1/screens/1/background-images/2");
var httpResponse = (HttpWebResponse) httpWebRequest.GetResponse();
using (var streamReader = new StreamReader(httpResponse.GetResponseStream()))
{
   var responseText = streamReader.ReadToEnd();
}
```

## 6.4. Reading background set information

**GET** /api/tpp/v1/screens/{screenId}/background-sets/{backgroundSetId}

### Request

| Name | Type | Description |
|---|---|---|
| screenId | integer | the screen number (from 1 to 2) |
| backgroundSetId | integer | the background set number (from 1 to 8) |

### Response

produces: application/json

| Name | Type | Description |
|---|---|---|
| IsEmpty | boolean | true is the background set empty, false if not |

### Response example

```
{
  "isEmpty": false
}
```

**Example: Read background set 5 information for screen 2**

### Raw TCP socket (connected on port 80 of 192.168.2.140)

```
GET /api/tpp/v1/screens/2/background-sets/5 HTTP/1.1<CR><LF><CR><LF>
```

<CR> replace with Carriage Return ASCII code: 13 (0x0D) <LF> replace with Line Feed ASCII code: 10 (0x0A)

### C#

```
var httpWebRequest =
(HttpWebRequest)WebRequest.Create("http://192.168.2.140/api/tpp/v1/screens/2/background-sets/5");

var httpResponse = (HttpWebResponse) httpWebRequest.GetResponse();
using (var streamReader = new StreamReader(httpResponse.GetResponseStream()))
{
   var responseText = streamReader.ReadToEnd();
}
```

# 7. Using thumbnails

## 7.1. Introduction

Thumbnails of live inputs, still images, outputs and multiviewer outputs are available. These thumbnails are regularly refreshed (except still images thumbnails which are refreshed only on change).
Snapshot request rate must not be more than 1 per second.
Picture size is 256 pixels (width) by up to 256 pixels (height). Black borders are automatically added, depending on aspect ratio. Picture type is PNG.

## 7.2. Live inputs thumbnails URL

http://<ipadress>/api/device/snapshots/inputs/1
up to
http://<ipadress>/api/device/snapshots/inputs/10

## 7.3. Outputs thumbnails URL

http://< ipadress>/api/device/snapshots/outputs/1
up to
http://<ipadress>/api/device/snapshots/outputs/2

## 7.4. Foreground Images thumbnails URL (per Screen)

http://< ipadress>/api/device/snapshots/screens/{screenId}/top/1
up to
http://<ipadress>/api/device/snapshots/screens/{screenId}/top/4

## 7.5. Background Images thumbnails URL (per Screen)

http://< ipadress>/api/device/snapshots/screens/{screenId}/back/1
up to
http://<ipadress>/api/device/snapshots/screens/{screenId}/back/4

## 7.6. Multiviewer thumbnails URL

http://< ipadress>/api/device/snapshots/multiviewer