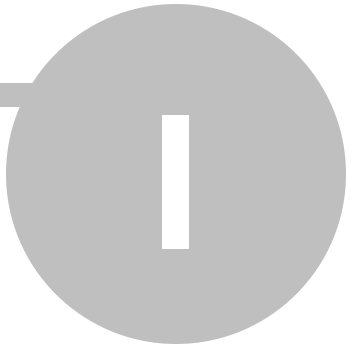
# Digifort HTTP API Interface
# API 1.6.0

# For Digifort 7.1.0.0

# Index

# Part



# 1 Overview

## 1.1 Description

This document specifies the HTTP base external programming interface of Digifort system.

The interface based on HTTP provides functionalities for accessing data of all the objects of the server, inserting data and alarms and request live and recorded images.

## 1.2 History

| Version | Date | Revision | Comments |
|---|---|---|---|
| 1.0.0 | 2009-Dec-1 | Éric Fleming Bonilha | First public version |
| 1.1.0 | 2010-Feb-23 | Éric Fleming Bonilha | Added Auxiliary PTZ command support |
| 1.1.0 | 2010-May-17 | Éric Fleming Bonilha | Added `SpotNumber` parameter to send an object to the virtual matrix |
| 1.1.0 | 2010-Jun-16 | Éric Fleming Bonilha | Added server event monitoring |
| 1.1.0 | 2011-Feb-02 | Éric Fleming Bonilha | - Updated the compatibility of the commands in order to reflect the addition of some features in Explorer and Standard versions<br>- Added a new variable type: APITimestamp<br>- Added new PTZ command: Area Zoom<br>- Added command to request a live MJPEG video stream<br>- Added session with LPR commands |
| 1.2.0 | 2011-Feb-28 | Éric Fleming Bonilha | - The command to create a safe authentication session no longer needs the mandatory `AuthSession=0` parameter<br>- The `Speed` parameter from area zoom command is no longer mandatory<br>- Added a new camera motion detection section |
| 1.2.0 | 2011-May-30 | Éric Fleming Bonilha | - The command to show an object on a virtual matrix monitor will now accept the "Analytics Configuration" and "LPR Configuration" object types<br>- Added 2 new object types in the command to retrieve the list of active monitors of virtual matrix: Analytics Configuration and LPR Configuration |
| 1.2.0 | 2011-Jun-21 | Éric Fleming Bonilha | - The command to monitor the server events will now accept the parameter `KeepAliveInterval` to |

| | | | |
|---|---|---|---|
| | | | control the sending of keep-alive events<br>- The command to monitor the LPR events will now accept the parameter `KeepAliveInterval` to control the sending of keep-alive events |
| 1.2.0 | 2011-Jun-28 | Éric Fleming Bonilha | Added new command to request a live media stream without transcoding |
| 1.2.1 | 2011-Sep-28 | Éric Fleming Bonilha | Added new analytics events to the command to monitor server events |
| 1.3.0 | 2011-Nov-09 | Éric Fleming Bonilha | - Added a new result code to the API commands. The return code 10 will now the used when the user does not have enough rights to access a given command<br>- The command to return a list of user screen views was changed from `GetScreenViews` to `GetUserScreenViews`<br>- Added a new command to return the list of public screen views<br>- Added a new command to display an user screen view in the virtual matrix<br>- Added a new command to display a public screen view in the virtual matrix<br>- The virtual matrix commands will not require the authentication with Admin user anymore, instead, the commands will now only require the authentication of an user with virtual matrix operating rights<br>- The command to request the list of global events will not require the authentication with Admin user anymore, instead, the command will now only require the user authentication.<br>- The command to trigger a global event will not require the authentication with Admin user anymore, instead, the command will now only require the authentication of an user with rights to trigger global events and with rights to access the global event to be triggered.<br>- The commands to request the list of maps and request the list of maps of an user were merged, now when requesting the list of maps the command will return only the list of maps that the user has access to and will not require the authentication with Admin user anymore. To keep compatibility with old API versions, the command `GetMapsViewRight` will still be able to be invoked, but it will be internally redirected to `GetMaps` instead.<br>- The commands to request the list of cameras and request the list of cameras of an user were merged, now when requesting the list of cameras the command will return only the list of cameras that the user has rights for live view or playback and will not require the authentication with Admin user anymore. To keep compatibility with old API versions, the command `GetCamerasViewRight` will still be able to be invoked, but it will be internally redirected to `GetCameras` instead. |

| | | | |
|---|---|---|---|
| 1.3.0 | 2012-Jan-04 | Éric Fleming Bonilha | - Added a new return code to cameras. The code 10001 can be returned when calling the routines GetSnapshot, GetMediaStream and GetJPEGStream if the camera is under privacy mode. |
| 1.3.0 | 2012-Jan-09 | Éric Fleming Bonilha | - Added a new command to request the data of the last LPR record<br>- Added a new command to request the image of the last LPR record |
| 1.3.0 | 2012-Mar-13 | Éric Fleming Bonilha | - The analytics events ANALYTICS_FOREIGN_OBJECT and ANALYTICS_MISSING_OBJECT were changed to ANALYTICS_ABANDONED_OBJECT and ANALYTICS_REMOVED_OBJECT in monitoring server events |
| 1.4.0 | 2012-Jun-13 | Éric Fleming Bonilha | - Added a new command to trigger the license plate recognition by using an external sensor<br>- Added new return code for LPR |
| 1.4.0 | 2012-Dec-05 | Éric Fleming Bonilha | - Added the new parameter OverrideShowCameras to the command to trigger a global event |
| 1.4.0 | 2012-Dec-12 | Éric Fleming Bonilha | - Added a new command to request the status of the cameras<br>- Added a new command to request the status of the I/O devices |
| 1.4.0 | 2013-Mar-22 | Éric Fleming Bonilha | - Added new parameter containing the media profiles of the cameras, on the command to request the list of cameras<br>- Added a new section with commands related to RTSP server<br>- Added a new parameter containing the name of LPR configuration on LPR records |
| 1.5.0 | 2013-Jul-08 | Éric Fleming Bonilha | - Added a new parameter to the command to trigger global events |
| 1.5.0 | 2013-Nov-06 | Éric Fleming Bonilha | - The analytics events ANALYTICS_SMOKE and ANALYTICS_FIRE were added to monitoring server events |
| 1.5.0 | 2013-Nov-06 | Éric Fleming Bonilha | - The software edition is now being informed in the command to request server information |
| 1.5.0 | 2013-Nov-11 | Éric Fleming Bonilha | - Added commands to control Manual Events from cameras<br>- The object type LPR_LIST was changed to LPR_EVENT in monitoring server events<br>- The event type LPR_PLATE_IN_LIST was changed to LPR in monitoring server events |
| 1.5.0 | 2013-Nov-12 | Éric Fleming Bonilha | - The command to request a live image will not return the result "Data not ready yet" anymore, it will return the image directly when it is ready |
| 1.5.0 | 2013-Dec-16 | Éric Fleming Bonilha | - Added a new description field to the current object of virtual matrix monitors |
| 1.5.0 | 2014-Mar-07 | Éric Fleming Bonilha | - The types of variables APITime and APITimestamp now includes milliseconds<br>- The following commands were affected by changes to variables APITime and APITimestamp:<br>Server/GetInfo |

| | | | |
|---|---|---|---|
| | | | Cameras/Playback/GetSnapshot<br>LPR/GetRecordData<br>LPR/GetLastRecordData<br>LPR/MonitorEvents<br>Events/Monitor<br>- Added new types of variables<br>- Added support to filters mask that can be used to filter results of some API commands<br>- The following commands now supports filters mask:<br>Cameras/IO/GetOutputActions<br>Cameras/ManualEvents/GetManualEvents<br>Cameras/GetCameras<br>Cameras/GetStatus<br>AlarmDevices/IO/GetOutputActions<br>AlarmDevices/GetAlarmDevices<br>AlarmDevices/GetStatus<br>Users/GetUsers<br>ScreenViews/GetUserScreenViews<br>ScreenViews/GetPublicScreenViews<br>Maps/GetMaps<br>GlobalEvents/GetGlobalEvents<br>LPR/GetLPRConfigurations<br>VirtualMatrix/GetActiveMonitors<br>VirtualMatrix/GetScreenMonitors<br>- Added commands to control camera bookmarks<br>- Added new command to start media playback in a monitor of virtual matrix<br>- New types of events for audio detection (AUDIO_LEVEL_LOW, AUDIO_LEVEL_HIGH) added to server events monitoring |
| 1.5.0 | 2014-Dec-01 | Éric Fleming Bonilha | - Added new commands for media playback<br>- New variable type Double<br>- New information of recording hours and estimative of recording hours on camera status<br>- New information with connection address and port for the list of cameras<br>- New events of FAILOVER and FAILBACK in server events monitoring<br>- Added values of reliability and hit to the commands to request date of an LPR record and data of the last LPR record<br>- Added command to search for LPR records<br>- Added command to search for Analytics records |
| 1.6.0 | 2015-Aug-04 | Éric Fleming Bonilha | - The type ALARM_DEVICE was changed to IO_DEVICE in server events monitoring<br>- Changed all commands of alarm devices to I/O devices |
| 1.6.0 | 2015-Sep-27 | Éric Fleming Bonilha | - Added Latitude and Longitude values to cameras in the list of cameras command<br>- New event of COMMUNICATION_RESTORED in server events monitoring<br>- New commands to control camera Privacy Mode<br>- New commands to control camera PTZ Patrol |

| | | | |
|---|---|---|---|
| | | | - New command to search for events |

| API Version | System Version |
|---|---|
| 1.0.0 | 6.3.0.0 |
| 1.1.0 | 6.4.0.0 |
| 1.2.0 | 6.5.0.0 - 6.5.0.1 |
| 1.3.0 | 6.6.0.0 |
| 1.4.0 | 6.7.0.0 |
| 1.5.0 | 7.0.0.0 |
| 1.6.0 | 7.1.0.0 |

# Part

# 2 Definitions

## 2.1 General notations

### 2.1.1 General abbreviations

The following abbreviations are used in this document

| Abbreviation | Description |
|---|---|
| N/A | Not applicable - The feature/parameter/value is not used in the specified task |
| URL | Uniform Resources Location (URL) is a compact string that represents a feature available in Internet. The RFC 1738 describes the syntax and semantics for a URL |
| URI | Uniform Resource Identifier (URI) is a compact string of characters for identifying an abstract or physical feature. The RFC 3986 describes the generic syntax of a URI |

### 2.1.2 Style conventions and general syntax of the URL

In URL syntax and in the descriptions of the arguments, text in italics with greater than and less than signs denote a content that must be substituted by a value or a string. When substituting the text, the greater than and less than signs must also be substituted. For example, the name of a camera is denoted by *<cameraname>* in description of URL syntax. In the example of URL syntax, *<cameraname>* is substituted by the string mycamera.

The URL syntax is written with the word "**Syntax:**" in bold face, accompanied by a box with the syntax of the command.

**Syntax:**

```
http://<server_address>/Interface/<object>[/<subobject>...]/<command>
[?<argument=value>[&<argument=value>...]][&<general_argument>...]
```

**Example 1:** Requests the version of the API with response in XML

```
http://192.168.0.1:8601/Interface/GetAPIVersion?ResponseFormat=XML
```

# Part

# 3 Interface

## 3.1 General arguments

All of the commands of the programming interface will accept some general arguments, which are:

**Arguments of basic authentication by parameters:**

| Argument | Valid values | Description |
|---|---|---|
| AuthUser | String | User for authentication |
| AuthPass | String | User password for authentication |

**Arguments of safe authentication:**

| Argument | Valid values | Description |
|---|---|---|
| AuthSession | Integer | ID of the authentication session returned by the command to create the authentication session |
| AuthData | String | Calculated authentication data based on the parameters returned by the command to create the authentication session |

**Arguments for the format of the response:**

| Argument | Valid values | Description |
|---|---|---|
| ResponseFormat | Text<br>XML | Format of the parameters of the response<br><br>Value: Text — Description: Response with parameters in text<br>Value: XML — Description: Response with parameters in XML |

## 3.2 Types of variables

This programming interface implements some formats of proper variables. The following are the variables and proper formats:

| Variable | Type of data | Description |
|---|---|---|
| APIDate | Date values | This format of variable is applied where data values are passed directly in the URL.<br><br>The format of date must be YYYY.MM.DD<br><br>Ex: 2009, November, 15<br>2009.11.15 |
| APITime | Time values | This format of variable is applied where time values are passed directly in the URL.<br><br>The format of time must be HH.MM.SS<br><br>Ex: 15:30:45<br>15.30.45 |
| APITimestamp | Time stamp values | This variable format is applied to return date and time values.<br><br>The format of this variable is YYYY-MM-DD HH:MM:SS |

| | | |
|---|---|---|
| | | Ex: 2009, November, 15, 15:30:45<br>2009-11-15 15:30:45 |
| APIMask | Text mask | Learn more in Filter Masks |
| APIMasks | Text masks | Set of masks to filter results from API.<br><br>Each mask must be delimited by comma character.<br><br>Ex: ABC*,123*,??6G[1-0] |
| APIDouble | Double Value | The variable APIDouble is used to represent fractional numbers and its format must follow the US standard format, ie, the decimal separator character is the period (".") and the thousands separator character is the comma (",").<br><br>The thousadans separator character (",") is not required.<br><br>Ex: 1,223,302.45, 1223302.45 |

## 3.3 Filter masks

Some API commands will allow the filtering of results using a filter mask.

The filter mask will compare the results with the specified mask, keeping only the valid results. The mask consists of literal characters, sets and wildcards.

Each literal character must match a single character in the string. The comparison to literal characters is case-insensitive.

Each set begins with an opening bracket ([) and ends with a closing bracket (]). Between the brackets are the elements of the set. Each element is a literal character or a range. Ranges are specified by an initial value, a dash (-), and a final value. Do not use spaces or commas to separate the elements of the set. A set must match a single character in the string. The character matches the set if it is the same as one of the literal characters in the set, or if it is in one of the ranges in the set. A character is in a range if it matches the initial value, the final value, or falls between the two values. All comparisons are case-insensitive. If the first character after the opening bracket of a set is an exclamation point (!), then the set matches any character that is not in the set.

Wildcards are asterisks (*) or question marks (?). An asterisk matches any number of characters. A question mark matches a single arbitrary character.

**Examples:**

```
Mask: ABC*
Result: Filter records starting with ABC.
Examples: ABCD, ABC123, ABCXYZ
```

```
Mask: ABC*123
Result: Filter records starting with ABC and ending with 123
Examples: ABCD123, ABC123, ABCXYZ123
```

```
Mask: ABC?123
Result: Filter records starting with ABC, containing any single
character and ending with 123
Examples: ABCD123, ABCX123, ABCY123
```

```
Mask: ABC??23
Result: Filter records starting with ABC, containing any two characters
and ending with 23
Examples: ABCD123, ABCXR23, ABCY923
```

```
Mask: ABC[XYZ]123
Result: Filter records starting with ABC, containing a single character
in the set of (X, Y or Z) and ending with 123
Examples: ABCX123, ABCY123, ABCZ123
```

```
Mask: ABC[!XYZ]123
Result: Filter records starting with ABC, containing a single character
not matching the set of (X, Y or Z) and ending with 123
Examples: ABCD123, ABCE123, ABCF123
```

```
Mask: ABC[D-G1-3]
Result: Filter records starting with ABC and containing a single character
in the set of (D to G) or (1 to 3)
Examples: ABCD, ABC3, ABCF
```

```
Mask: ABC[D-G1-3]??[!ABC1-3]XYZ*
Result: Filter records starting with ABC, containing a single character in
the set of (D to G) or (1 to 3), containing any two characters, containing
the literals XYZ and ending with any chain of characters
Examples: ABCD12UXYZ, ABC2Y1UXYZ12345
```

## 3.4 Authentication

To access the programming interface commands user and password authentication is required.

This programming interface supports 3 different types of authentication:

**Types of authentition:**

| Value | Description |
|---|---|
| Basic HTTP | Basic HTTP authentication |
| Basic with parameters | Basic authentication by parameters in the URL |
| Safe | Safe authentication by parameters in the URL |

### 3.4.1 Basic HTTP authentication

The basic HTTP authentication method can be used for authentication in the interface.

This is an unsafe authentication method, which most libraries of HTTP communication must support. This is a method that sends the user and password in Base 64 Code in the headers of the HTTP message.

**Warning:** We do not recommend the use of this type of authentication due to the high risk of exposure of the access credentials. Instead of using this type of authentication, give preference to the use of Safe Autentication. While safe authentication is the best and safest of the 3 supported authentication methods, is is also the most difficult to use, as it involves the use of hashing with MD5, which may not be supported by all of the libraries.

To learn more about the basic HTTP authentication, refer to the RFC 2617.

### 3.4.2 Basic authentication with parameters

Basic authentication with parameters is an unsafe form of authentication that sends the access credentials (User and password) by way of parameters of the URL using pure text.

While this type of authentication is the easiest to use, it is also the most unsafe, as the access credentials are sent without cryptography or any coding.

**Warning:** We do not recommend the use of this type of authentication due to the high risk of exposure of the access credentials. Instead of using this type of authentication, give preference to the use of Safe Authentication. While being the best and safest of the 3 supported methods, safe authentication is also the most difficult to use, as it involves the use of hashing with MD5, which may not be supported by all of the libraries. When the use of safe authentication is not possible, give preference to Basic HTTP Authentication instead of Basic Autentication with parameters

To use this type of authentication, add the `AuthUser` and `AuthPass` parameters in the URL of the command.

**Parameters of authentication:**

| Parameter | Type | Description |
|---|---|---|
| `AuthUser` | String | User for authentication |
| `AuthPass` | String | Password for authentication |

**Example 1:** Requests the version of the API with authentication with the user admin, without password

```
http://192.168.0.1:8601/Interface/GetAPIVersion?AuthUser=admin
```

**Example 2:** Requests the version of the API with auhtentication with the user admin, and the password pass

```
http://192.168.0.1:8601/Interface/GetAPIVersion?AuthUser=admin&
AuthPass=pass
```

### 3.4.3 Safe authentication

Safe authentication with parameters is the recommended method for authentication in the programming interface. This is the safest of the 3 available methods, however, it is also the most difficult to implementar, as it uses Hashing MD5 techniques, which may not be available in all of the libraries.

The implementation of this method was conceived using as its base some concepts of HTTP Digest authentication (RFC 2617), though with simpler methods.

Due to the nature of HTTP protocol HTTP not maintaining a TCP connection for all requests, to authenticate using safe authentication, we must first create an authentication session.

After creating the session, it's necessary to calculate the autentication data.

After calculating the authentication data, simply add the `AuthSession` and `AuthData parameters in the` URL of the command to be executed.

**Parameters of authentication:**

| Parameter | Type | Description |
|---|---|---|
| `AuthSession` | String | ID of the authentication session returned by the command for to create the authentication session |
| `AuthData` | String | Authentication data calculated based on the parameters returnad by the command to create the authentication session |

**Example 1:** Requests the version of the API with safe authentication

```
http://192.168.0.1:8601/Interface/GetAPIVersion?AuthSession=1&
AuthData=AF63604073043A3C47FB5A506D8A8EFD
```

With the safe authentication session created, you must now keep it open.

### 3.4.3.1 Creating an authentication session

Create an authentication session.

**Compatibility:** All editions

**Methodo:** HTTP GET

**Syntax:**

```
http://<server_address>/Interface/CreateAuthSession[?<general_argument>
[&<general_argument>...]]
```

**Response:**
An authentication session will be created, and the ID and the random value NOnce will be generated.

**HTTP Return:** 200 OK

**Parameters of return:**

| Parameter | Type | Description |
|---|---|---|
| `ID` | Integer | ID of the authentication session |
| `NONCE` | String | Random value for hashing the access credentials |

**Response in text:**
The parameters of response in text will obey the following syntax:

```
<field>=<value>
```

| Parameter | Description |
|---|---|
| *field* | Name of the field |
| *value* | Value of the field |

**Example of return in text:**

```
RESPONSE_CODE=0
RESPONSE_MESSAGE=OK
ID=2
NONCE=D11193880891BEB0931A52E3F3CA322F
```

**Response in XML:**
The parameters if responst in XML will obey the following syntax:

```
<Session>
  <ID>ID</ID>
  <NOnce>NONCE</NOnce>
</Session>
```

**Example of return in XML:**

```
<?xml version="1.0" encoding="UTF-8" ?>
<Response>
  <Code>0</Code>
  <Message>OK</Message>
  <Data>
    <Session>
      <ID>3</ID>
      <NOnce>76569D03D9D479201EDB1FAB36DBEDEA</NOnce>
    </Session>
  </Data>
</Response>
```

### 3.4.3.2 Calculating the authentication data

To calculate the authentication data, we use the following formula:

```
AuthData = MD5Hash(NOnce + ":" + UpperCase(Username) + ":" + MD5Hash(Password));
```

Where `MD5Hash` is the call for a function which accepts the string as input parameter and returns a Hash MD5 with values in hexadecimal.

Where `UpperCase` is the call for a function which accepts a string as input parameter and returns this string with the characters in upper case.

Where `Username` is the user of authentication.

Where `Password` is the password of the user of authentication.

Therefore, `AuthData` is the result of the Hash MD5 of the concatenation of the value NOnce sent by the server, with the colon character ":", with te user of authentication in upper case characters, with the colon character ":", with the Hash MD5 of the password of authentication.

**Example of calculation of the authentication for the user "admin" with the password "pass":**

**NOnce value received:**

```
NONCE=68F1EE37050F456851DC90D62791839E
```

**Calculation of AuthData:**

```
AuthData = MD5Hash(68F1EE37050F456851DC90D62791839E + ":" + "ADMIN" +
  ":" + MD5Hash(pass));
```

**Result of AuthData:**

```
AuthData=AF63604073043A3C47FB5A506D8A8EFD
```

#### 3.4.3.3 Maintaining an authentication session

Due to the nature of HTTP protocol not maintaining TCP connections (consequently making it difficult to maintain authentication sessions), after creating a safe authentication session, you must keep it open during the entire period of use of the API.

By default, an authentication session will expire in 60 seconds, if there is no activity in the API using its authentication ID. If its authentication session expires, it will be necessary to create a new session.

To keep an authentication session open, simply maintain activity in the API by call of some command, thus updating the hour of the last call of the API and maintaining the session for 60 more seconds. However, a special command was created for maintaining an authentication session, and the use of this command is recommended in case the activity in the interface is not constant.

**Compatibility:** All editions

**Method:** HTTP GET

**Syntax:**

```
http://<server_address>/Interface/UpdateAuthSession[?<argument=value>
[&<argument=value>...][&<general_argument>...]]
```

**Arguments:**

| Argument | Valid values | Description |
|---|---|---|
| AuthSession=<*Integer*> | Integer >= 1 | ID of the authentication session to be kept open |
| AuthData=<*String*> | String | Data of the authentication of the session |

**Response:**
Default response of the API.

**HTTP Return:** 200 OK

**Parameters of return:** Default return of the API

## 3.5 Responses

The responses to the commands of API can be formatted in two types: Text and XML.

### 3.5.1 Response in text

The responses in text format will be in the format of a list of Parameters and Values:

```
<parameter_1>=<value>
<parameter_2>=<value>
..
<parameter_n>=<value>
```

**Example:**

```
RESPONSE_CODE=0
RESPONSE_MESSAGE=OK
COUNT=10
```

#### 3.5.1.1 List of parameters in text

When the response of a command returns a list of parameters (Ex: List of cameras, list of users, etc...), these parameters will obey the following syntax:

```
<object>_<num>_<field>=<value>
```

**Where:**

| Value | Description |
|---|---|
| *Object* | Name of the type of object |
| *Num* | Number of the record |
| *Field* | Name of the field |
| *Value* | Value of the field |

**Example 1:** List of cameras

```
CAMERA_1_NAME=Entrance
CAMERA_1_DESCRIPTION=Front Camera
CAMERA_1_MODEL=Generic
CAMERA_1_DEVICETYPE=2
CAMERA_2_NAME=Backdoor
CAMERA_2_DESCRIPTION=Door Camera
CAMERA_2_MODEL=Generic
CAMERA_2_DEVICETYPE=2
```

**Example 2:** List of users

```
USER_1_NAME=admin
USER_1_DESCRIPTION=System Administration Account
USER_2_NAME=Charlie
USER 2 DESCRIPTION=Charlie Brown
```

### 3.5.2 Response in XML

The responses in XML will be inserted into the tag root `<Response></Response>`.

The default return parameters will be inside the tag root, whereby the parameters of response of the called command will be returned inside the tag `<Data></Data>`.

**Example:**

```
<Response>
  <Code>0</Code>
  <Message>OK</Message>
  <Data>
    <ApiVersion>
      <Name>Digifort HTTP API Server</Name>
      <Version>1.1.0</Version>
      <Major>1</Major>
      <Minor>1</Minor>
      <BugFix>0</BugFix>
    </ApiVersion>
  </Data>
</Response>
```

#### 3.5.2.1 List of parameters in XML

When the response of a command returns a list of parameters (Ex: List of cameras, list of users, etc...), these parameters will obey the following syntax:

```
<Objects>
  <Count>COUNT</Count>
  <Object>
    <Field>FIELD_VALUE</Field>
  </Object>
  ..
  <Object>
    <Field>FIELD_VALUE</Field>
  </Object>
</Objects>
```

**Where:**

| Value | Description |
|---|---|
| Object | Name of the type of object |
| Count | Total number of registers |
| Field | Name of the field |
| FIELD_VALUE | Value of the field |

**Example 1:** List of cameras

```
<?xml version="1.0" encoding="UTF-8" ?>
<Response>
  <Code>0</Code>
  <Message>OK</Message>
  <Data>
    <Cameras>
      <Count>2</Count>
      <Camera>
        <Name>Camera1</Name>
        <Description>My Camera</Description>
        <Model>Generic</Model>
        <DeviceType>1</DeviceType>
      </Camera>
      <Camera>
        <Name>Camera2</Name>
        <Description>My Video Server</Description>
        <Model>Generic</Model>
        <DeviceType>2</DeviceType>
      </Camera>
    </Cameras>
  </Data>
</Response>
```

**Example 2:** List of users

```
<?xml version="1.0" encoding="UTF-8" ?>
<Response>
  <Code>0</Code>
  <Message>OK</Message>
  <Data>
```

```
  <Users>
    <Count>2</Count>
    <User>
      <Name>admin</Name>
      <Description>System administration account</Description>
    </User>
    <User>
      <Name>Guest</Name>
      <Description>Guest user</Description>
    </User>
  </Users>
 </Data>
</Response>
```

### 3.5.3 Default parameters of return

All commands of the API will return a default response and the response data of the called command.

**Default parameters of response:**

| Parameter | Type | Description |
|---|---|---|
| CODE | Integer | Code of the response |
| MESSAGE | String | Message of the response |

**List of parameters of response in text:**
The parameters of response in text will obey the following syntax:

```
RESPONSE_<field>=<value>
```

| Parameter | Description |
|---|---|
| *field* | Name of the field |
| *value* | Value of the field |

**Example of return in text:**

```
RESPONSE_CODE=0
RESPONSE_MESSAGE=OK
```

**List of parameters of response in XML:**
The parameters of response in XML will obey the following syntax:

```
<Response>
 <Code>CODE</Code>
 <Message>MESSAGE</Message>
</Response>
```

**Example of return in XML:**

```
<?xml version="1.0" encoding="UTF-8" ?>
<Response>
 <Code>0</Code>
 <Message>OK</Message>
</Response>
```

### 3.5.4 Return codes

The tables below summarize all of the error return codes and messages of API HTTP

**General errors (Applicable to any command):**

| Value | Message |
|---|---|
| 0 | OK |
| 1 | Missing parameter |
| 2 | RESERVED (NOT CURRENTLY IN USE) |
| 3 | Object not found |
| 4 | Object deactivated |
| 5 | You don´t have enough rights to access the desired object |
| 6 | Data not ready yet |
| 7 | Invalid parameter value |
| 8 | To access this command you need to use the Admin user account |
| 9 | Authentication required |
| 10 | You don´t have enough rights to use the specified command |

**Safe Authentication Errors (Applicable only to commands for safe authentication):**

| Value | Message |
|---|---|
| 100 | Invalid or expired authentication session |
| 101 | Authentication error |

**Camera Errors (Applicable only to camera commands:**

| Value | Message |
|---|---|
| 10000 | Camera out of order |
| 10001 | Camera is under privacy mode |
| 10100 | You don´t have enough rights to control PTZ cameras |
| 10101 | You don´t have enough rights to control the specified camera |
| 10102 | The camera is locked by another user |
| 10103 | The PTZ controls for the specified camera are disabled |
| 10104 | Invalid PTZ operation |
| 10105 | Command not supported by the PTZ driver |
| 10200 | Image not found for the specified date and time |
| 10201 | Decoding error |
| 10202 | Database error |

**I/O Errors (Applicable only to I/O commands):**

| Value | Message |
|---|---|
| 20000 | Alarm output action not found |

**Virtual Matrix Errors (Applicable only to virtual matrix commands):**

| Value | Message |
|---|---|
| 30000 | Monitor not found |

**LPR Errors (Applicable only to LPR commands):**

| Value | Message |
|---|---|
| 40000 | Record not found |
| 40100 | LPR Configuration out of order |

# Part IV

# 4 API Groups

## 4.1 Version

Commands for API versioning

### 4.1.1 Requesting the version of API

Requests the version of API HTTP

**Compatibility:** All editions

**Security level:** Requires authentication of the user

**Method:** HTTP GET

**Syntax:**

```
http://<server_address>/Interface/GetAPIVersion[?<general_argument>
[&<general_argument>...]]
```

**Example 1:** Requests the version of API with response in XML

```
http://192.168.0.1:8601/Interface/GetAPIVersion?ResponseFormat=XML
```

**Example 2:** Requests the version of API with response in Text and authentication of the user admin

```
http://192.168.0.1:8601/Interface/GetAPIVersion?ResponseFormat=Text&
AuthUser=admin
```

**Response:**
A list of parameter-value pairs is returned

**HTTP Return:** 200 OK

**Parameters of return:**

| Parameter | Type | Description |
|---|---|---|
| NAME | String | Na of the API HTTP server |
| VERSION | String | Version of API HTTP (MAJOR.MINOR.BUGFIX) |
| MAJOR | Integer | Main Version of API |
| MINOR | Integer | Features Version of API |
| BUGFIX | Integer | Corrections Version of API |

**Example of return in text:**

```
RESPONSE_CODE=0
RESPONSE_MESSAGE=OK
NAME=Digifort HTTP API Server
VERSION=1.0.0
MAJOR=1
MINOR=0
BUGFIX=0
```

**Example of return in XML:**

```
<?xml version="1.0" encoding="UTF-8" ?>
<Response>
  <Code>0</Code>
  <Message>OK</Message>
  <Data>
    <ApiVersion>
      <Name>Digifort HTTP API Server</Name>
      <Version>1.0.0</Version>
      <Major>1</Major>
      <Minor>0</Minor>
      <BugFix>0</BugFix>
    </ApiVersion>
  </Data>
</Response>
```

## 4.2 Server

Commands to query server info

### 4.2.1 Requesting the machine code of the server

Requests the machine code of the server. The machine code of the server is the code that identifies the licenses of each computer, you can use it to identify the servers.

**Compatibility:** All editions

**Security level:** Requires authentication of the user

**Method:** HTTP GET

**Syntax:**

```
http://<server_address>/Interface/Server/GetMachineCode
[?<general_argument>[&<general_argument>...]]
```

**Example 1:** Requests the machine code of the server with response in XML

```
http://192.168.0.1:8601/Interface/Server/GetMachineCode?ResponseFormat=XML
```

**Example 2:** Requests the machine code of the server with response in Text and authentication of the user admin

```
http://192.168.0.1:8601/Interface/Server/GetMachineCode?
ResponseFormat=Text&AuthUser=admin
```

**Response:**
A list of parameter-value pairs is returned

**HTTP Return HTTP:** 200 OK

**Parameters of return:**

| Parameter | Type | Description |
|---|---|---|
| MACHINECODE | String | Machine code of the server |

**Example of return in text:**

```
RESPONSE_CODE=0
RESPONSE_MESSAGE=OK
MACHINECODE=AF20-DGF-9016277-C3681*1CFDC9/5AD0-MKEY-D921A7
```

**Example of return in XML:**

```
<?xml version="1.0" encoding="UTF-8" ?>
<Response>
  <Code>0</Code>
  <Message>OK</Message>
  <Data>
    <MachineCode>
      <MachineCode>AF20-DGF-9016277-C3681*1CFDC9/5AD0-MKEY-D921A7</MachineCode>
    </MachineCode>
  </Data>
</Response>
```

### 4.2.2 Requesting data about the server

Requests data of the server.

**Compatibility:** All editions

**Security level:** Requires authentication of the user

**Method:** HTTP GET

**Syntax:**

```
http://<server_address>/Interface/Server/GetInfo
[?<general_argument>[&<general_argument>...]]
```

**Example 1:** Requests the data of the server with response in XML

```
http://192.168.0.1:8601/Interface/Server/GetInfo?ResponseFormat=XML
```

**Example 2:** Requests the data of the server with response in text and authentication of the user admin

```
http://192.168.0.1:8601/Interface/Server/GetInfo?
ResponseFormat=Text&AuthUser=admin
```

**Response:**
A list of parameter-value pairs is returned

**HTTP Return HTTP:** 200 OK

**Parameters of return:**

| Parameter | Type | Description |
|---|---|---|
| EDITION | String | Edition of the server |
| VERSION | String | Version of the server |
| RELEASEDATE | APIDate | Release date of the version |
| RELEASETYPE | String | Type of release of the version |
| PLATFORM | String | Platform of the server |
| UPTIME | Integer | The number of seconds that the server is active |
| DATE | APIDate | Date of the server |
| TIME | APITime | Time of the server |

**Example of return in text:**

```
RESPONSE_CODE=0
RESPONSE_MESSAGE=OK
EDITION=Enterprise
VERSION=6.3.0.0
RELEASEDATE=2009.11.18
RELEASETYPE=Alpha 3
PLATFORM=Windows XP/2003/Vista/2008/7
UPTIME=51
DATE=2009.11.23
TIME=15.26.12
```

**Example of return in XML:**

```
<?xml version="1.0" encoding="UTF-8" ?>
<Response>
 <Code>0</Code>
 <Message>OK</Message>
 <Data>
  <Info>
   <Edition>ENTERPRISE</Edition>
   <Version>6.3.0.0</Version>
   <ReleaseDate>2009.11.18</ReleaseDate>
   <ReleaseType>Alpha 3</ReleaseType>
   <Platform>Windows XP/2003/Vista/2008/7</Platform>
   <UpTime>222</UpTime>
   <Date>2009.11.23</Date>
   <Time>15.29.03</Time>
  </Info>
 </Data>
</Response>
```

### 4.2.3 Requesting data of usage of the server

Requests data about the usage of the server.

**Compatibility:** All editions

**Security level:** Requires authentication of the user

**Method:** HTTP GET

**Syntax:**

```
http://<server_address>/Interface/Server/GetUsage
[?<general_argument>[&<general_argument>...]]
```

**Example 1:** Requests data about the usage of the server with response in XML

```
http://192.168.0.1:8601/Interface/Server/GetUsage?ResponseFormat=XML
```

**Example 2:** Requests data about the usage of the server with response in text and authentication of the user admin

```
http://192.168.0.1:8601/Interface/Server/GetUsage?
ResponseFormat=Text&AuthUser=admin
```

**Response:**
A list of parameter-value pairs is returned

**HTTP Return HTTP:** 200 OK

**Parameters of return:**

| Parameter | Type | Description |
|---|---|---|
| PROCESSOR | Integer | Processor utilization in percentage (0 to 100) |
| GLOBALMEMORY | Integer64 | General memory usage in bytes |
| SERVERMEMORY | Integer64 | Server memory usage in bytes |
| CONNECTIONS | Integer | Total number of TCP connections of clients connected to the server |
| CLIENTS | Integer | Total number of distinct clients connected to the server |
| INPUTTRAFFIC | Integer64 | Input traffic in Kbps (Kbits per second) |
| OUTPUTTRAFFIC | Integer64 | Output traffic in Kbps (Kbits per second) |

**Example of return in text:**

```
RESPONSE_CODE=0
RESPONSE_MESSAGE=OK
PROCESSOR=5
GLOBALMEMORY=2206560256
SERVERMEMORY=44388
CONNECTIONS=56
CLIENTS=4
INPUTTRAFFIC=5847
OUTPUTTRAFFIC=20384
```

**Example of return in XML:**

```
<?xml version="1.0" encoding="UTF-8" ?>
<Response>
  <Code>0</Code>
  <Message>OK</Message>
  <Data>
    <Stats>
      <Processor>2</Processor>
      <GlobalMemory>2211676160</GlobalMemory>
      <ServerMemory>44388</ServerMemory>
      <Connections>56</Connections>
      <Clients>4</Clients>
      <InputTraffic>5847</InputTraffic>
      <OutputTraffic>20384</OutputTraffic>
    </Stats>
  </Data>
</Response>
```

## 4.3 Cameras

Commands to control cameras

### 4.3.1 Requesting the list of cameras

Requests the list of cameras that the user has live view or playback rights

**Compatibility:** All editions

**Security level:** Requires authentication of the user

**Method:** HTTP GET

**Syntax:**

```
http://<server_address>/Interface/Cameras/GetCameras[?<argument=value>
[&<argument=value>...][&<general_argument>...]]
```

**Arguments:**

| Argument | Valid values | Description |
|---|---|---|
| Cameras=<*APIMasks*> | String | Mask to filter the results. Specify which cameras must be returned based on the provided masks. |
| Fields=<*String*> | Name<br>Description<br>Model<br>DeviceType<br>ConnectionAddress<br>ConnectionPort<br>Latitude<br>Longitude<br>MediaProfiles | Specifies the list of desired fields. In case this parameter is omitted, all of the fields will be sent.<br><br>The fields must be separated by commas |

| Name | Description |
|---|---|
| Name | Name of the camera |
| Description | Description of the camera |
| Model | Model of the camera |
| DeviceType | Type of devices |
| ConnectionAddress | Camera network address |
| ConnectionPort | Camera network port |
| Latitude | Camera latitude |
| Longitude | Camera longitude |
| MediaProfiles | List of camera media profiles |

**Example 1:** Requests the list of cameras with all of the fields and response in XML

```
http://192.168.0.1:8601/Interface/Cameras/GetCameras?ResponseFormat=XML
```

**Example 2:** Requests the list of cameras with all of the fields and response in text

```
http://192.168.0.1:8601/Interface/Cameras/GetCameras?ResponseFormat=Text
```

**Example 3:** Requests the list of cameras with only name and description, response in XML and authentication of the user Admin

```
http://192.168.0.1:8601/Interface/Cameras/GetCameras?Fields=Name,
Description&ResponseFormat=XML&AuthUser=admin
```

**Example 4:** Request the list of cameras starting with A, with just name and description, response in XML and authentication with Admin user

```
http://192.168.0.1:8601/Interface/Cameras/GetCameras?Cameras=A*&
Fields=Name,Description&ResponseFormat=XML&AuthUser=admin
```

**Response:**
A list of all of the cameras that the user has live view or playback rights is returned. The fields returned in the will depend on the values informed in the argument `Fields`

**HTTP Return:** 200 OK

**Parameters of return:**

**Fixed Parameter:**

| Parameter | Type | Description |
|---|---|---|
| COUNT | Integer | Total number of cameras |

**Parameters in the list of cameras:**

| Parameter | Type | Description |
|---|---|---|
| NAME | String | Name of the camera |
| DESCRIPTION | String | Description of the camera |
| MODEL | String | Model of the camera |
| DEVICETYPE | Integer | Type of device<br>Type: 1 = IP Camera; 2 = Video Server; 3 = Network Video Recorder (NVR); 4 = Digital Video Recorder (DVR) |
| CONNECTIONADDRESS | String | Camera network connection address. This information will only be available if the user has rights to camera register. |
| CONNECTIONPORT | String | Camera network connection port. This information will only be available if the user has rights to camera register. |
| LATITUDE | APIDouble | Camera latitude with 6 digit precision |
| LONGITUDE | APIDouble | Camera longitude with 6 digit precision |
| MEDIAPROFILES | String | Names of the camera media profiles, comma separated |

**List of cameras:**
The parameters of the list of cameras will depend on the type of response (Text or XML).

**List of cameras with response in text:**
The parameters of response in text will obey the following syntax:

```
CAMERA_<num>_<field>=<value>
```

| Parameter | Description |
|---|---|
| *num* | Number of the record |
| *field* | Name of the field |
| *value* | Value of the field |

**Example of return in text:**

```
RESPONSE_CODE=0
RESPONSE_MESSAGE=OK
COUNT=2
CAMERA_1_NAME=Camera1
CAMERA_1_DESCRIPTION=My Camera
CAMERA_1_MODEL=Generic
CAMERA_1_DEVICETYPE=1
CAMERA_1_CONNECTIONADDRESS=192.168.1.1
CAMERA_1_CONNECTIONPORT=80
CAMERA_1_LATITUDE=0.000000
CAMERA_1_LONGITUDE=0.000000
```

```
CAMERA_1_MEDIAPROFILES=Recording,Visualization
CAMERA_2_NAME=Camera2
CAMERA_2_DESCRIPTION=My Video Server
CAMERA_2_MODEL=Generic
CAMERA_2_DEVICETYPE=2
CAMERA_2_CONNECTIONADDRESS=192.168.1.2
CAMERA_2_CONNECTIONPORT=80
CAMERA_2_LATITUDE=-23.630363
CAMERA_2_LONGITUDE=-46.554916
CAMERA_2_MEDIAPROFILES=Recording,Visualization
```

**List of cameras with response in XML:**
The parameters of response in XML will obey the following syntax:

```
<Cameras>
 <Count>COUNT</Count>
 <Camera>
  <Name>NAME</Name>
  <Description>DESCRIPTION</Description>
  <Model>MODEL</Model>
  <DeviceType>DEVICETYPE</DeviceType>
  <ConnectionAddress>CONNECTION_ADDRESS</ConnectionAddress>
  <ConnectionPort>CONNECTION_PORT</ConnectionPort>
  <Latitude>LATITUDE</Latitude>
  <Longitude>LONGITUDE</Longitude>
  <MediaProfiles>MEDIAPROFILES</MediaProfiles>
 </Camera>
</Cameras>
```

**Example of return in XML:**

```
<?xml version="1.0" encoding="UTF-8" ?>
<Response>
 <Code>0</Code>
 <Message>OK</Message>
 <Data>
  <Cameras>
   <Count>2</Count>
   <Camera>
    <Name>Camera1</Name>
    <Description>My Camera</Description>
    <Model>Generic</Model>
    <DeviceType>1</DeviceType>
    <ConnectionAddress>192.168.1.1</ConnectionAddress>
    <ConnectionPort>80</ConnectionPort>
    <Latitude>0.000000</Latitude>
    <Longitude>0.000000</Longitude>
    <MediaProfiles>Recording,Visualization</MediaProfiles>
   </Camera>
   <Camera>
    <Name>Camera2</Name>
    <Description>My Video Server</Description>
    <Model>Generic</Model>
    <DeviceType>2</DeviceType>
    <ConnectionAddress>192.168.1.2</ConnectionAddress>
```

```
      <ConnectionPort>80</ConnectionPort>
      <Latitude>-23.630363</Latitude>
      <Longitude>-46.554916</Longitude>
      <MediaProfiles>Recording,Visualization</MediaProfiles>
    </Camera>
  </Cameras>
 </Data>
</Response>
```

### 4.3.2 Requesting the status of the cameras

Request the status of the cameras of which the user has live view or playback rights.

**Compatibility:** All editions

**Security level:** Requires authentication of the user

**Method:** HTTP GET

**Syntax:**

```
http://<server_address>/Interface/Cameras/GetStatus[?<argument=value>
[&<argument=value>...][&<general_argument>...]]
```

**Arguments:**

<table>
<tr><th>Argument</th><th>Valid values</th><th>Description</th></tr>
<tr><td>Cameras=<APIMasks></td><td>String</td><td>Mask to filter the results. Specify which cameras must be returned based on the provided masks.</td></tr>
<tr><td>Fields=<String></td><td>Active<br>Working<br>RecordingHours<br>RecordingHoursEstimative</td><td>Specifies the list of desired fields. In case this parameter is omitted, all of the fields will be sent.<br><br>The fields must be separated by commas<br><br><table><tr><th>Name</th><th>Description</th></tr><tr><td>Active</td><td>Identify if the camera is active</td></tr><tr><td>Working</td><td>Identify if the camera is working</td></tr><tr><td>RecordingHours</td><td>Amount of recording hours</td></tr><tr><td>RecordingHoursEstimative</td><td>Estimative of recording hours</td></tr></table></td></tr>
<tr><td>Active=<Boolean></td><td>TRUE<br>FALSE</td><td>In case this parameter is specified, a filter will be applied and only the cameras that matches this filter will be returned, thus, in case the value Active=TRUE is specified, the result will only include the active cameras, while if the value Active=FALSE is specified, the result will only include the deactivated cameras.</td></tr>
<tr><td>Working=<Boolean></td><td>TRUE<br>FALSE</td><td>In case this parameter is specified, a filter will be applied and only the cameras that matches this filter will be returned, thus, in case the value Working=TRUE is specified, the result will only</td></tr>
</table>

| | | |
|---|---|---|
| | | include the working cameras, while if the value `Working=FALSE` is specified, the result will only include the cameras that are out of order. |

**Example 1:** Request the status of all cameras with all fields and response in XML

```
http://192.168.0.1:8601/Interface/Cameras/GetStatus?ResponseFormat=XML
```

**Example 2:** Request the status of all active cameras with response in text

```
http://192.168.0.1:8601/Interface/Cameras/GetStatus?Active=TRUE&
ResponseFormat=Text
```

**Example 3:** Request the status of all active cameras starting with A, with response in text

```
http://192.168.0.1:8601/Interface/Cameras/GetStatus?Cameras=A*&
Active=TRUE&ResponseFormat=Text
```

**Example 4:** Request the status of all active cameras that are not working, with response in XML and authentication with admin user (No password)

```
http://192.168.0.1:8601/Interface/Cameras/GetStatus?Active=TRUE&
Working=FALSE&ResponseFormat=XML&AuthUser=admin
```

**Response:**
A list with the status of all of the cameras that the user has live view or playback rights is returned. The fields returned in the will depend on the values informed in the argument `Fields`

**HTTP Return:** 200 OK

**Parameters of return:**

**Fixed Parameter:**

| Parameter | Type | Description |
|---|---|---|
| COUNT | Integer | Total number of cameras |

**Parameters in the list of status of cameras:**

| Parameter | Type | Description |
|---|---|---|
| NAME | String | Name of the camera |
| ACTIVE | Boolean | Identify if the camera is actve |
| WORKING | Boolean | Identify if the camera is working |
| RECORDINGHOURS | APIDouble | Number of recorded hours. This parameter will only be available if the user has rights to view the status of cameras |
| RECORDINGHOURSESTIMATIVE | APIDouble | Estimative of recording hours. This parameter will only be available if the user has rights to view the status of cameras |

**List of status of cameras:**
The parameters of the list of status of cameras will depend on the type of response (Text or XML).

**List of status of cameras with response in text:**
The parameters of response in text will obey the following syntax:

```
CAMERA_<num>_<field>=<value>
```

| Parameter | Description |
|---|---|
| *num* | Number of the record |
| *field* | Name of the field |
| *value* | Value of the field |

**Example of return in text:**

```
RESPONSE_CODE=0
RESPONSE_MESSAGE=OK
COUNT=2
CAMERA_1_NAME=Camera1
CAMERA_1_ACTIVE=TRUE
CAMERA_1_WORKING=TRUE
CAMERA_1_RECORDINGHOURS=1389.2
CAMERA_1_RECORDINGHOURSESTIMATIVE=1389.4
CAMERA_2_NAME=Camera2
CAMERA_2_ACTIVE=TRUE
CAMERA_2_WORKING=FALSE
CAMERA_2_RECORDINGHOURS=1388.2
CAMERA_2_RECORDINGHOURSESTIMATIVE=1389.4
```

**List of status of cameras with response in XML:**

The parameters of response in XML will obey the following syntax:

```
<Cameras>
 <Count>COUNT</Count>
 <Camera>
  <Name>NAME</Name>
  <Active>ACTIVE</Active>
  <Working>WORKING</Working>
  <RecordingHours>RECORDING_HOURS</RecordingHours>
  <RecordingHoursEstimative>RECORDING_HOURS_ESTIMATIVE</RecordingHoursEstimative>
 </Camera>
</Cameras>
```

**Example of return in XML:**

```
<?xml version="1.0" encoding="UTF-8" ?>
<Response>
 <Code>0</Code>
 <Message>OK</Message>
 <Data>
  <Cameras>
   <Count>2</Count>
   <Camera>
    <Name>Camera1</Name>
    <Active>TRUE</Active>
    <Working>TRUE</Working>
    <RecordingHours>1389.2</RecordingHours>
    <RecordingHoursEstimative>1389.3</RecordingHoursEstimative>
   </Camera>
   <Camera>
    <Name>Camera2</Name>
    <Active>TRUE</Active>
    <Working>FALSE</Working>
    <RecordingHours>1388.2</RecordingHours>
    <RecordingHoursEstimative>1389.4</RecordingHoursEstimative>
   </Camera>
  </Cameras>
 </Data>
</Response>
```

### 4.3.3 Requesting a live image (Snapshot)

Requests a live image (JPEG Snapshot) of the specified camera.

**Compatibility:** All editions

**Security level:** Requires authentication of the user

**Method:** HTTP GET

**Syntax:**

```
http://<server_address>/Interface/Cameras/GetSnapshot?<argument=value>
[&<argument=value>...][&<general_argument>...]
```

**Arguments:**

| Argument | Valid values | Description |
|---|---|---|
| Camera=<String>* | String | Name of the camera |
| Width=<Integer> | Integer. X >= 1 | Width of the snapshot image<br><br>If this parameter is omitted, the default value of 352 will be used |
| Height=<Integer> | Integer. X >= 1 | Height of the snapshot image<br><br>If this parameter is omitted, the default value of 240 will be used |
| Quality=<Integer> | Integer. 0 >= X <= 100 | JPEG Compression quality of the snapshot image<br><br>If this parameter is omitted, the default value of 50 will be |

| | | |
|---|---|---|
| | | used |

* Mandatory parameters

**Example 1:** Requests the image of a camera with the default values and error response in XML

```
http://192.168.0.1:8601/Interface/Cameras/GetSnapshot?Camera=Camera1&
ResponseFormat=XML
```

**Example 2:** Requests the image of a camera with size of 640x480, compression 70 and error response in text

```
http://192.168.0.1:8601/Interface/Cameras/GetSnapshot?Camera=Camera1&
Width=640&Height=480&Quality=70&ResponseFormat=Text
```

**Response:**
In case of success, the JPEG image will be returned. In the case of error, a code and an error message will be returned. A error can be returned if the user does not have live viewing rights of the camera, if the camera is deactivated, if the camera was not found, if the camera is out of order or if the camera is under privacy mode.

**HTTP Return:** 200 OK

**Parameters of return:** Default return of API

### 4.3.4 Requesting a live media stream

Request a live media stream of the specified camera. The media stream will be sent using the coding from the chosen media profile.

**Compatibility:** All editions

**Security level:** Requires authentication of the user

**Method:** HTTP GET

**Syntax:**

```
http://<server_address>/Interface/Cameras/GetMediaStream?<argument=value>
[&<argument=value>...][&<general_argument>...]
```

**Arguments:**

| Argument | Valid values | Description |
|---|---|---|
| `Camera=<String>*` | String | Camera name |
| `Profile=<String>` | Recording<br>Visualization<br>Custom | Type of media profile:<br><br>Recording - Request the media stream using the default recording profile<br>Visualization - Request the media stream using the default visualization profile<br>Custom - Request the media stream using the profile specified on `CustomProfile` argument<br><br>If this parameter is omitted, the default value Recording will be used |
| `CustomProfile=<String>` | String | Media profile name in case the value of argument `Profile` is "Custom" |

* Mandatory parameters

**Example 1:** Request a media stream from a camera using the default values and error response in XML

```
http://192.168.0.1:8601/Interface/Cameras/GetMediaStream?Camera=Camera1&
ResponseFormat=XML
```

**Example 2:** Request a media stream from a camera using the default visualization profile and error response in text

```
http://192.168.0.1:8601/Interface/Cameras/GetMediaStream?Camera=Camera1&
Profile=Visualization&ResponseFormat=Text
```

**Example 3:** Request a media stream from a camera using the custom profile "HighResolution" and error response in text

```
http://192.168.0.1:8601/Interface/Cameras/GetMediaStream?Camera=Camera1&
Profile=Custom&CustomProfile=HighResolution&ResponseFormat=Text
```

**Response:**
In case of success, a media stream will be sent by using HTTP Multipart `x-mixed-replace` transmission. Each media frame is separated by the multipart boundary `--DigifortBoundary`.

After the multipart boundary, a little HTTP header will be sent containing the media frame type and the frame size

| Fields | Valid values | Description |
|---|---|---|
| Content-Type | image/jpeg | JPEG image |
| | image/wavelet | WAVELET image |
| | video/mpeg | Raw MPEG-4 video |
| | video/h263 | Raw H.263 video |
| | video/h264 | Raw H.264 video |
| | application/octet-stream | Unknown format |
| Content-Length | Integer | The size (In bytes) of the media frame |

In the case of error, a code and an error message will be returned. A error can be returned if the user does not have live viewing rights of the camera, if the camera is deactivated, if the camera was not found, if the camera is out of order or if the camera is under privacy mode.

**Example of JPEG media stream:**

```
--DigifortBoundary
Content-Type: image/jpeg
Content-Length: 35463

JPEG_DATA
JPEG_DATA
..
..
..
JPEG_DATA
--DigifortBoundary
Content-Type: image/jpeg
Content-Length: 34236

JPEG_DATA
JPEG_DATA
..
..
..
JPEG_DATA
```

**Example of H.264 media stream:**

```
--DigifortBoundary
Content-Type: video/h264
Content-Length: 10436

H264_DATA
H264_DATA
..
..
..
H264_DATA
--DigifortBoundary
Content-Type: video/h264
Content-Length: 2548

H264_DATA
H264_DATA
..
..
..
H264_DATA
```

**HTTP Return:** 200 OK

**Parameters of return:** Default return of API

### 4.3.5 Requesting a live video stream (MJPEG)

Requests a live video stream (MJPEG) of the specified camera.

**Warning:** This command can overload the server processing. This will occur because in order to deliver

the live MJPEG video stream the server will transcode the video that it is receiving from the camera.

**Compatibility:** All editions

**Security level:** Requires authentication of the user

**Method:** HTTP GET

**Syntax:**

```
http://<server_address>/Interface/Cameras/GetJPEGStream?<argument=value>
[&<argument=value>...][&<general_argument>...]
```

**Arguments:**

| Argument | Valid values | Description |
|---|---|---|
| `Camera=<String>*` | String | Camera name |
| `Width=<Integer>` | Integer. X >= 1 | Image width<br><br>If this parameter is omitted, the default value of 352 will be used |
| `Height=<Integer>` | Integer. X >= 1 | Image height<br><br>If this parameter is omitted, the default value of 240 will be used |
| `Quality=<Integer>` | Integer. 0 >= X <= 100 | JPEG image compression quality value<br><br>If this parameter is omitted, the default value of 50 will be used |
| `FPS=<Integer>` | Integer. 1 >= X <= 30 | Amount of frames per second<br><br>If this parameter is omitted, the default value of 30 will be used |

* Mandatory parameters

**Note:** The server will not be able to deliver more frames than it is receiving from the camera by using the recording profile

**Example 1:** Request an MJPEG video stream from a camera with default values and error response in XML

```
http://192.168.0.1:8601/Interface/Cameras/GetJPEGStream?Camera=Camera1&
ResponseFormat=XML
```

**Example 2:** Request an MJPEG video stream from a camera with 640x480 resolution, compression 70, 10 frames per second and error response in text

```
http://192.168.0.1:8601/Interface/Cameras/GetJPEGStream?Camera=Camera1&
Width=640&Height=480&Quality=70&FPS=10&ResponseFormat=Text
```

**Response:**

In case of success, a stream of JPEG images will be sent by using HTTP Multipart `x-mixed-replace` transmission. Each image is separated by the multipart boundary `--DigifortBoundary`.
In the case of error, a code and an error message will be returned. A error can be returned if the user does not have live viewing rights of the camera, if the camera is deactivated, if the camera was not found, if the camera is out of order or if the camera is under privacy mode.

**Example of JPEG image stream:**

```
--DigifortBoundary
Content-Type: image/jpeg
Content-Length: 35463

JPEG_DATA
JPEG_DATA
..
..
..
JPEG_DATA
--DigifortBoundary
Content-Type: image/jpeg
Content-Length: 34236

JPEG_DATA
JPEG_DATA
..
..
..
JPEG_DATA
```

**HTTP Return:** 200 OK

**Parameters of return:** Default return of API

## 4.3.6 Playback

Commands for camera media playback

### 4.3.6.1 Requesting a recorded image (Snapshot)

Requests a recorded image (JPEG Snapshot) of a specified camera.

**Compatibility:** All editions

**Security level:** Requires authentication of the user

**Method:** HTTP GET

**Syntax:**

```
http://<server_address>/Interface/Cameras/Playback/GetSnapshot?
<argument=value>[&<argument=value>...][&<general_argument>...]
```

**Arguments:**

| Argument | Valid values | Description |
|---|---|---|
| Camera=<*String*>* | String | Name of the camera |
| Date=<*APIDate*>* | Date | Date of the image in APIDate format |
| Time=<*APITime*>* | Time | Time of the image in APITime format |
| Width=<*Integer*> | Integer. X >= -1 && X != 0 | Width of the snapshot image<br><br>Use the value -1 for the original width of the recorded |

| | | |
|---|---|---|
| | | image<br><br>If this parameter is omitted, the default value of 352 will be used |
| Height=<*Integer*> | Integer. X >= -1 && X != 0 | Height of the snapshot image<br><br>Use the value -1 for the original height of the recorded image<br><br>If this parameter is omitted, the default value of 240 will be used |
| Quality=<*Integer*> | Integer. 0 >= X <= 100 | Quality of the JPEG compression of the snapshot image<br><br>If this parameter os omitted, the default value of 50 will be used |

* Mandatory parameters

**Example 1:** Requests the image of a camera of the 15th of November, 2009 at 10:00:00 with error response in XML

```
http://192.168.0.1:8601/Interface/Cameras/Playback/GetSnapshot?
Camera=Camera1&Date=2009.11.15&Time=10.00.00&ResponseFormat=XML
```

**Example 2:** Requests the image of a camera of the 20th of November, 2009 at 15:55:20 with size of 640x480, compression 70 and error response in text

```
http://192.168.0.1:8601/Interface/Cameras/Playback/GetSnapshot?
Camera=Camera1&Date=2009.11.20&Time=15.55.20&Width=640&Height=480&
Quality=70&ResponseFormat=Text
```

**Response:**
In the case of success, a JPEG image will be returned. In the case of error, a code and as error message will be returned. An error can be returned if the user does not have viewing rights of the recording of the camera, if the camera is not found or if the image was not found.

**HTTP Return:** 200 OK

**Parameters of return:** Default return of API

#### 4.3.6.2 Requesting a recorded media stream

Requests a recorded media stream from the specified camera. The media stream will be sent on its original format (No media transcoding will be performed).

**Compatibility:** All editions

**Security level:** Requires authentication of the user

**Method:** HTTP GET

**Syntax:**

```
http://<server_address>/Interface/Cameras/Playback/GetMediaStream?
<argument=value>[&<argument=value>...][&<general_argument>...]
```

**Arguments:**

| Argument | Valid values | Description |
|---|---|---|
| Camera=<*String*>* | String | Camera name |
| StartDate=<*APIDate*>* | Date | Start date of recordings on APIDate format |
| StartTime=<*APITime*>* | Time | Start time of recordings on APITime format |
| EndDate=<*APIDate*> | Date | End date of recordings on APIDate format<br><br>If this parameter is omitted, the current date will be used |
| EndTime=<*APITime*> | Time | End time of recordings on APITime format<br><br>If this parameter is omitted, the current time will be used |
| Push=<*Boolean*> | TRUE<br>FALSE | TRUE - Send recorded footage as fast as possible, with no playback speed control (Use when downloading recordings).<br><br>FALSE - Send recorded footage with playback speed control of 1x<br><br>If this parameter is omitted, the default value FALSE will be used |
| Audio=<*Boolean*> | TRUE<br>FALSE | TRUE - Send audio multiplexed with video (If audio is recorded)<br><br>FALSE - Send video only<br><br>If this parameter is omitted, the default value FALSE will be used |

* Mandatory parameters

**Example 1:** Request recordings of camera "Camera1" from January 01, 2014 10:00:00AM to January 02, 2014 09:00:00AM with error response in XML

```
http://192.168.0.1:8601/Interface/Cameras/Playback/GetMediaStream?
Camera=Camera1&StartDate=2014.01.01&StartTime=10.00.00.000&
EndDate=2014.01.02&EndTime=09.00.00.000&ResponseFormat=XML
```

**Example 2:** Request recordings of camera "Camera1" from January 01, 2014 to date, with audio, without playback speed control and error response in text

```
http://192.168.0.1:8601/Interface/Cameras/Playback/GetMediaStream?
Camera=Camera1&StartDate=2014.01.01&StartTime=00.00.00.000&Audio=TRUE&
Push=TRUE&ResponseFormat=Text
```

**Example 3:** Request recordings of camera "Camera1" from January 25, 2014 to date, with audio and playback speed control

```
http://192.168.0.1:8601/Interface/Cameras/Playback/GetMediaStream?
Camera=Camera1&StartDate=2014.01.20&StartTime=00.00.00.000&Audio=TRUE
```

**Response:**
In case of success, a media stream will be sent by using HTTP Multipart `x-mixed-replace` transmission. Each media frame is separated by the multipart boundary `--DigifortBoundary`.

The main HTTP response header will contain the following custom headers:

| Header | Valid values | Description |
|---|---|---|
| DGF-FrameCount | Integer | Specifies the amount of recorded frames that will be sent within the transmission |

After the multipart boundary, a little HTTP header will be sent containing information about the frame:

| Header | Valid values | Description |
|---|---|---|
| Content-Type | image/jpeg | JPEG image |
| | image/wavelet | WAVELET image |
| | video/mpeg | MPEG-4 video |
| | video/h263 | H.263 video |
| | video/h264 | H.264 video |
| | audio/L24 | L-PCM |
| | audio/basic | G.711 audio |
| | audio/G726-16 | G.726 audio in 16kbps |
| | audio/G726-24 | G.726 audio in 24kbps |
| | audio/G726-32 | G.726 audio in 32kbps |
| | audio/G726-40 | G.726 audio in 40kbps |
| | audio/AAC | AAC audio |
| | application/octet-stream | Unrecognized format |
| Content-Length | Integer | Specify the size (In bytes) of the media frame |
| DGF-FrameNumber | Integer | Frame number |
| DGF-FrameDate | APIDate | Frame date in APIDate format |
| DGF-FrameTime | APITime | Frame time in APITime format |
| DGF-FrameType | jpeg | JPEG image |
| | jpeg-2000 | JPEG-2000 image |
| | h.263/I-Frame | H.263 frame of type I |
| | h.263/P-Frame | H.263 frame of type P |
| | h.263/PB-Frame | H.263 frame of type PB |
| | h.263/B-Frame | H.263 frame of type B |
| | h.263/EI-Frame | H.263 frame of type EI |
| | h.263/EP-Frame | H.263 frame of type EP |
| | mpeg-4/I-Frame | MPEG-4 frame of type I |
| | mpeg-4/P-Frame | MPEG-4 frame of type P |
| | mpeg-4/B-Frame | MPEG-4 frame of type B |
| | h.264/I-Frame | H.264 frame of type I |
| | h.264/P-Frame | H.264 frame of type P |
| | h.264/B-Frame | H.264 frame of type B |
| | h.264/I-Field | H.264 field of type I |
| | h.264/P-Field | H.264 field of type P |
| | h.264/B-Field | H.264 field of type B |
| | audio/L24 | L-PCM audio chunk |
| | audio/basic | G.711 audio chunk |
| | audio/G726-16 | G.726 audio chunk in 16kbps |
| | audio/G726-24 | G.726 audio chunk in 24kbps |
| | audio/G726-32 | G.726 audio chunk in 32kbps |
| | audio/G726-40 | G.726 audio chunk in 40kbps |
| | audio/AAC | AAC audio chunk |
| | application/octet-stream | Unrecognized format |

In the case of error, a code and an error message will be returned. A error can be returned if the user does not have live viewing rights of the camera, if the camera is deactivated or if the camera was not found.

In case no recordings are found the connection will be gracefully closed and the HTTP custom header DGF-FrameCount will contain value 0

**Example of JPEG media stream:**

```
HTTP/1.1 200 OK
Connection: close
Content-Type: multipart/x-mixed-replace; boundary=--DigifortBoundary
DGF-FrameCount: 103531

--DigifortBoundary
Content-Type: image/jpeg
Content-Length: 35463
DGF-FrameNumber: 1
DGF-FrameDate: 2014.12.01
DGF-FrameTime: 10.00.00.755
DGF-FrameType: jpeg

JPEG_DATA
JPEG_DATA
..
..
..
JPEG_DATA
--DigifortBoundary
Content-Type: image/jpeg
Content-Length: 34236
DGF-FrameNumber: 2
DGF-FrameDate: 2014.12.01
DGF-FrameTime: 10.00.01.126
DGF-FrameType: jpeg

JPEG_DATA
JPEG_DATA
..
..
..
JPEG_DATA
```

**Example of H.264 media stream**

```
HTTP/1.1 200 OK
Connection: close
Content-Type: multipart/x-mixed-replace; boundary=--DigifortBoundary
DGF-FrameCount: 47263

--DigifortBoundary
Content-Type: video/h264
Content-Length: 10436
DGF-FrameNumber: 1
DGF-FrameDate: 2014.12.01
DGF-FrameTime: 10.00.00.231
DGF-FrameType: h.264/I-Frame

H264_DATA
H264_DATA
..
..
..
H264_DATA
--DigifortBoundary
Content-Type: video/h264
Content-Length: 2548
DGF-FrameNumber: 2
DGF-FrameDate: 2014.12.01
DGF-FrameTime: 10.00.00.438
DGF-FrameType: h.264/P-Frame

H264_DATA
H264_DATA
..
..
..
H264_DATA
```

**HTTP Return:** 200 OK

**Parameters of return:** Default return of API

#### 4.3.6.3 Requesting a recorded media stream (MJPEG)

Requests a recorded media stream from the specified camera. The media stream will be sent in MJPEG format (It will be transcoded from its original format, leading to a higher use of server CPU). This command does not transmit audio recordings, only transcoded video in MJPEG format.

**Compatibility:** All editions

**Security level:** Requires authentication of the user

**Method:** HTTP GET

**Syntax**

```
http://<server_address>/Interface/Cameras/Playback/GetJPEGStream?
<argument=value>[&<argument=value>...][&<general argument>...]
```

**Arguments:**

| Argument | Valid values | Description |
|---|---|---|
| Camera=<*String*>* | String | Camera name |
| StartDate=<*APIDate*>* | Date | Start date of recordings on APIDate format |
| StartTime=<*APITime*>* | Time | Start time of recordings on APITime format |
| EndDate=<*APIDate*> | Date | End date of recordings on APIDate format<br><br>If this parameter is omitted, the current date will be used |
| EndTime=<*APITime*> | Time | End time of recordings on APITime format<br><br>If this parameter is omitted, the current time will be used |
| Width=<*Integer*> | Integer. X >= -1 && X != 0 | Custom image width<br><br>Specify value -1 to use the original recorded image width<br><br>If this parameter is omitted, the default value -1 will be used |
| Height=<*Integer*> | Integer. X >= -1 && X != 0 | Custom image height<br><br>Specify value -1 to use the original recorded image height<br><br>If this parameter is omitted, the default value -1 will be used |
| Quality=<*Integer*> | Integer. 0 >= X <= 100 | JPEG compression quality<br><br>If this parameter is omitted, the default value 50 will be used |
| Push=<*Boolean*> | TRUE<br>FALSE | TRUE - Send recorded footage as fast as possible, with no playback speed control (Use when downloading recordings).<br><br>FALSE - Send recorded footage with playback speed control of 1x<br><br>If this parameter is omitted, the default value FALSE will be used |

* Mandatory parameters

**Example 1:** Request recordings of camera "Camera1" from January 01, 2014 10:00:00AM to January 02, 2014 09:00:00AM with error response in XML

```
http://192.168.0.1:8601/Interface/Cameras/Playback/GetJPEGStream?
Camera=Camera1&StartDate=2014.01.01&StartTime=10.00.00.000&
EndDate=2014.01.02&EndTime=09.00.00.000&ResponseFormat=XML
```

**Example 2:** Request recordings of camera "Camera1" from January 01, 2014 to date, without playback speed control and error response in text

```
http://192.168.0.1:8601/Interface/Cameras/Playback/GetJPEGStream?
```

```
Camera=Camera1&StartDate=2014.01.01&StartTime=00.00.00.000&
Push=TRUE&ResponseFormat=Text
```

**Example 3:** Request recordings of camera "Camera1" from January 25, 2014 to date, with playback speed control, custom size of 352x240 and compression quality of 80

```
http://192.168.0.1:8601/Interface/Cameras/Playback/GetJPEGStream?
Camera=Camera1&StartDate=2014.01.20&StartTime=00.00.00.000&
Width=352&Height=240&Quality=80
```

**Response:**
In case of success, a JPEG stream will be sent by using HTTP Multipart `x-mixed-replace` transmission. Each JPEG frame is separated by the multipart boundary `--DigifortBoundary`.

The main HTTP response header will contain the following custom headers:

| Header | Valid values | Description |
|---|---|---|
| DGF-FrameCount | Integer | Specifies the amount of recorded frames that will be sent within the transmission |

After the multipart boundary, a little HTTP header will be sent containing information about the frame:

| Header | Valid values | Description |
|---|---|---|
| Content-Type | image/jpeg | JPEG image |
| Content-Length | Integer | Specify the size (In bytes) of the media frame |
| DGF-FrameNumber | Integer | Frame number |
| DGF-FrameDate | APIDate | Frame date in APIDate format |
| DGF-FrameTime | APITime | Frame time in APITime format |

In the case of error, a code and an error message will be returned. A error can be returned if the user does not have live viewing rights of the camera, if the camera is deactivated or if the camera was not found.

In case no recordings are found the connection will be gracefully closed and the HTTP custom header DGF-FrameCount will contain value 0

**Example of JPEG media stream:**

```
HTTP/1.1 200 OK
Connection: close
Content-Type: multipart/x-mixed-replace; boundary=--DigifortBoundary
DGF-FrameCount: 103531

--DigifortBoundary
Content-Type: image/jpeg
Content-Length: 35463
DGF-FrameNumber: 1
DGF-FrameDate: 2014.12.01
DGF-FrameTime: 10.00.00.755

JPEG_DATA
JPEG_DATA
..
..
..
JPEG_DATA
--DigifortBoundary
Content-Type: image/jpeg
Content-Length: 34236
DGF-FrameNumber: 2
DGF-FrameDate: 2014.12.01
DGF-FrameTime: 10.00.01.126

JPEG_DATA
JPEG_DATA
..
..
..
JPEG_DATA
```

**HTTP Return:** 200 OK

**Parameters of return:** Default return of API

## 4.3.7 PTZ

Commands for PTZ control and auxiliary commands

Not all cameras will support all of the commands supported by the system. To know which groups of commands a given camera supports, use the **Supported Commands** command

### 4.3.7.1 Simple

Simple PTZ allows the control of movable cameras in a simple way, with only a command call.

**Compatibility:** All editions

**Security level:** Requires authentication of the user

**Method:** HTTP GET

**Syntax:**

```
http://<server_address>/Interface/Cameras/PTZ/Simple?<argument=value>
[&<argument=value>...][&<general_argument>...]
```

**Arguments:**

| Argument | Valid values | Description |
|---|---|---|
| Camera=<String>* | String | Name of the camera |
| Operation=<String>* | MoveLeft<br>MoveRight<br>MoveUp<br>MoveDown<br>MoveUpLeft<br>MoveUpRight<br>MoveDownLeft<br>MoveDownRight<br>Home<br>ZoomTele<br>ZoomWide<br>FocusNear<br>FocusFar<br>IrisOpen<br>IrisClose | Control operation to be executed<br><br>Operation: Description<br>MoveLeft: Moves the camera to the left<br>MoveRight: Moves the camera to the right<br>MoveUp: Moves the camera upwards<br>MoveDown: Moves the camera downwards<br>MoveUpLeft: Moves the camera upwards and to the left simultaneously<br>MoveUpRight: Moves the camera upwards and to the right simultaneously<br>MoveDownLeft: Moves the camera downwards and to the left simultaneously<br>MoveDownRight: Moves the camera downwards and to the right simultaneously<br>Home: Calls the Home position<br>ZoomTele: Zoom in<br>ZoomWide: Zoom out<br>FocusNear: Focuses on objects near the camera<br>FocusFar: Focuses on objects far from the camera<br>IrisOpen: Opens Iris<br>IrisClose: Closes Iris |
| Speed=<Integer> | Integer. 0 >= X <= 100 | Speed of operation. If this parameter is omitted, the default value of 100 will be used |

* Mandatory parameters

**Note:** Some commands/parameters may not work with some cameras due to limitations of the communication interface itself developed by the camera's manufacturer. The basic parameters of Up, Down, Left, Right, Zoom In and Zoom Out should work for all cameras, whereas the parameters of Simultaneous Movement, Home Position, Focus and Iris may vary from one manufacturer to another.

**Example 1:** Moves a camera to the left at a speed of 50

```
http://192.168.0.1:8601/Interface/Cameras/PTZ/Simple?Camera=Camera1&
Operation=MoveRight&Speed=50
```

**Example 2:** Moves a camera downwards at a speed of 100 and response in text

```
http://192.168.0.1:8601/Interface/Cameras/PTZ/Simple?Camera=Camera1&
Operation=MoveDown&Speed=100&ResponseFormat=Text
```

**Examplo 3:** Execute zoom in a camera with response in XML and authentication of the user User1

```
http://192.168.0.1:8601/Interface/Cameras/PTZ/Simple?Camera=Camera1&
Operation=ZoomTele&ResponseFormat=XML&AuthUser=User1&AuthPass=User1Pass
```

**Response:**
Default response of API. In case the "Simple" command group is not supported by the camera's driver (due to restrictions in the communication interface developed by the camera's manufacturer), the error code 10105 (Command not supported by the PTZ driver) will be returned.

**HTTP Return HTTP:** 200 OK

**Parameters of return:** Default return of API

#### 4.3.7.2 Relative

By way of relative movement control, you will be able to control the movement of the camera in degrees starting from the present position.

**Compatibility:** All editions

**Security level:** Requires auhtentication of the user

**Method:** HTTP GET

**Syntax:**

```
http://<server_address>/Interface/Cameras/PTZ/Relative?<argument=value>
[&<argument=value>...][&<general_argument>...]
```

**Arguments:**

<table>
<tr><th>Argument</th><th>Valid values</th><th>Description</th></tr>
<tr><td>Camera=<String>*</td><td>String</td><td>Name of the camera</td></tr>
<tr><td>Operation=<String>*</td><td>Pan<br>Tilt<br>Zoom<br>Focus<br>Iris</td><td>Control operation to be executed
<table>
<tr><th>Operation</th><th>Description</th></tr>
<tr><td>Pan</td><td>Relative operation of Pan (leftwards and rightwards movement)</td></tr>
<tr><td>Tilt</td><td>Relative operation of Tilt (Upwards and Downwards movement)</td></tr>
<tr><td>Zoom</td><td>Relative operation of Zoom</td></tr>
<tr><td>Focus</td><td>Relative operation of Focus</td></tr>
<tr><td>Iris</td><td>Relative operation of Iris</td></tr>
</table></td></tr>
<tr><td>Value=<Integer>*</td><td>Integer. Consult the table of valid values</td><td>Value of the operation in degrees for Pan/Tilt and steps for Zoom, Focus and Iris
<table>
<tr><th>Operation</th><th>Values of the operation</th></tr>
<tr><td>Pan</td><td>-360 to 360<br><br>Negative values = Left<br>Positive values = Right</td></tr>
</table></td></tr>
</table>

| | | |
|---|---|---|
| | | Tilt: -360 to 360<br><br>Negative value = Up<br>Positive values = Down<br>Zoom: -100 to 100<br><br>Negative values = Zoom Out<br>Positive values = Zoom In<br>Focus: -100 to 100<br><br>Negative values = Focus Near<br>Positive values = Focus Far<br>Iris: -100 a 100<br><br>Negative values = Close Iris<br>Positive values = Open Iris |
| `Speed=<Integer>` | Integer. 0 >= X <= 100 | Speed of operation. If this parameter is omitted, the default value of 100 will be used |

* Mandatory parameters

**Note:** Some commands/parameters may not work with some cameras due to limitations of the communication interface itself developed by the camera's manufacturer.

**Example 1:** Moves a camera 40 degrees to the right at a speed of 50

```
http://192.168.0.1:8601/Interface/Cameras/PTZ/Relative?Camera=Camera1&
Operation=Pan&Value=40&Speed=50
```

**Example 2:** Moves a camera downwards 10 degrees at a speed of 100 and response in text

```
http://192.168.0.1:8601/Interface/Cameras/PTZ/Relative?Camera=Camera1&
Operation=Tilt&Value=10&Speed=100&ResponseFormat=Text
```

**Example 3:** Executes 20 steps of zoom in a camera with response in XML and authentication of the user User1

```
http://192.168.0.1:8601/Interface/Cameras/PTZ/Relative?Camera=Camera1&
Operation=Zoom&Value=20&ResponseFormat=XML&AuthUser=User1&
AuthPass=User1Pass
```

**Responste:**
Default response of API. In case the "Relative" command group is not supported by the camera's driver (due to restrictions in the communication interface developed by the camera's manufacturer), the error code 10105 (Command not supported by the PTZ driver) will be returned.

**HTTP Return HTTP:** 200 OK

**Parameters of return:** Default return of API

#### 4.3.7.3 Absolute

By way of absolute movement control, you will be able to position the camera in given coordenates in degrees.

**Compatibility:** All editions

**Security level:** Requires authentication of the user

**Method:** HTTP GET

**Syntax:**

```
http://<server_address>/Interface/Cameras/PTZ/Absolute?<argument=value>
[&<argument=value>...][&<general_argument>...]
```

**Arguments:**

<table>
<tr><th>Argument</th><th>Valid values</th><th>Description</th></tr>
<tr><td>Camera=<String>*</td><td>String</td><td>Name of the camera</td></tr>
<tr><td>Operation=<String>*</td><td>Pan<br>Tilt<br>Zoom<br>Focus<br>Iris</td><td>Control operation to be executed
<table>
<tr><th>Operation</th><th>Description</th></tr>
<tr><td>Pan</td><td>Absolute operation of Pan (Leftwards and rightwards movement)</td></tr>
<tr><td>Tilt</td><td>Absolute operation of Tilt (Upwards and downwards movement)</td></tr>
<tr><td>Zoom</td><td>Absolute operation of Zoom</td></tr>
<tr><td>Focus</td><td>Absolute operation of Focus</td></tr>
<tr><td>Iris</td><td>Absolute operation of Iris</td></tr>
</table></td></tr>
<tr><td>Value=<Integer>*</td><td>Integer. Consult the table of valid values</td><td>Value of the operation in degrees for Pan/Tilt and steps for Zoom, Focus and Iris
<table>
<tr><th>Operation</th><th>Values of the operation</th></tr>
<tr><td>Pan</td><td>-180 to 180<br><br>Negative values = Left<br>Positive values = Right</td></tr>
<tr><td>Tilt</td><td>-180 to 180<br><br>Negative values = Up<br>Positive values = Down</td></tr>
<tr><td>Zoom</td><td>1 to 100</td></tr>
<tr><td>Focus</td><td>1 to 100</td></tr>
<tr><td>Iris</td><td>1 to 100</td></tr>
</table></td></tr>
<tr><td>Speed=<Integer></td><td>Integer. 0 >= X <= 100</td><td>Speed of operation. If this parameter is omitted, the default value of 100 will be used</td></tr>
</table>

* Mandatory parameters

**Note:** Some commands/parameters may not work with some cameras due to limitations of the communication interface itself developed by the camera's manufacturer.

**Example 1:** Moves a camera to the Pan position of 40 degrees at a speed of 50

```
http://192.168.0.1:8601/Interface/Cameras/PTZ/Absolute?Camera=Camera1&
Operation=Pan&Value=40&Speed=50
```

**Example 2:** Moves a camera to the Tilt position of 10 degrees at a speed of 100 and response in text

```
http://192.168.0.1:8601/Interface/Cameras/PTZ/Absolute?Camera=Camera1&
Operation=Tilt&Value=10&Speed=100&ResponseFormat=Text
```

**Example 3:** Moves the lens of the camera to a position of 20 absolute steps of zoom with response in XML and authentication of the user User1

```
http://192.168.0.1:8601/Interface/Cameras/PTZ/Absolute?Camera=Camera1&
Operation=Zoom&Value=20&ResponseFormat=XML&AuthUser=User1&
AuthPass=User1Pass
```

**Response:**
Default response of API. In case the "Absolute" command group is not supported by the camera's driver (due to restrictions in the communication interface itself developed by the camera's manufacturer), the error code 10105 (Command not supported by the PTZ driver) will be returned.

**HTTP Return HTTP:** 200 OK

**Parameters of return:** Default return of API

#### 4.3.7.4 Area Zoom

By way of area zoom coommand, you will be able to position the camera on a given image rectangle.

**Compatibility:** All editions

**Security level:** Requires authentication of the user

**Method:** HTTP GET

**Syntax:**

```
http://<server_address>/Interface/Cameras/PTZ/AreaZoom?<argument=value>
[&<argument=value>...][&<general_argument>...]
```

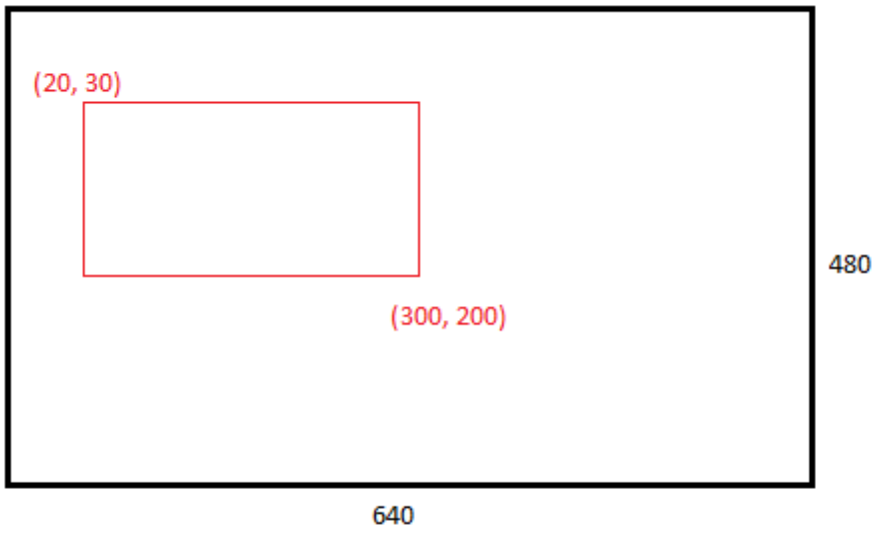
**Arguments:**

| Argument | Valid values | Description |
|---|---|---|
| Camera=<*String*>* | String | Camera name |
| Width=<*Integer*>* | Integer. X >= 1 | Image width |
| Height=<*Integer*>* | Integer. X >= 1 | Image height |
| Left=<*Integer*>* | Integer. X >= 0 <= Width | Left position of rectangle |
| Right=<*Integer*>* | Integer. X >= Left <= Width | Right position of rectangle |
| Top=<*Integer*>* | Integer. X >= 0 <= Height | Top position of rectangle |
| Bottom=<*Integer*>* | Integer. X >= Top <= Height | Bottom position of rectangle |
| Speed=<*Integer*> | Integer. 0 >= X <= 100 | Speed of operation |

* Mandatory parameters

**Note:** Some commands/parameters may not work with some cameras due to limitations of the communication interface itself developed by the camera's manufacturer.

**Example:** Perform zoom on an area with speed of 50 using the following image as reference:




```
http://192.168.0.1:8601/Interface/Cameras/PTZ/AreaZoom?Camera=Camera1&
Width=640&Height=480&Left=20&Top=30&Right=300&Bottom=200&Speed=50
```

**Response:**
Default response of API. In case the "Area Zoom" command group is not supported by the camera's driver (due to restrictions in the communication interface itself developed by the camera's manufacturer), the error code 10105 (Command not supported by the PTZ driver) will be returned.

**HTTP Return HTTP:** 200 OK

**Parameters of return:** Default return of API

#### 4.3.7.5 Simultaneous

By way of simultaneous PT, also known as "Click and Center", you will supply X and Y coordenates of the camera, moving the camera to the desired location.

**Compatibility:** All editions

**Security level:** Requires authentication of the user

**Method:** HTTP GET

**Syntax:**

```
http://<server_address>/Interface/Cameras/PTZ/Simultaneous?<argument=value>
[&<argument=value>...][&<general_argument>...]
```

**Arguments:**

| Argument | Valid values | Description |
|---|---|---|
| Camera=<*String*>* | String | Name of the camera |
| X=<*Integer*>* | Integer. 0 >= X <= 640 | Position X over the proportion of an image with width of 640 pixels |
| Y=<*Integer*>* | Integer. 0 >= Y <= 480 | Position Y over the proportion of an image with height of 480 pixels |
| Speed=<*Integer*> | Integer. 0 >= X <= 100 | Speed of operation. If this parameter is omitted, the default value of 100 will be used |

* Mandatory parameters

**Note:** The values of X and Y are calculated based on an image of 640x480. To calculate X and Y using different proportions, simply apply the Rule of 3 on the image.
Ex:
Image of 352x240
Desired position: X = 50, Y = 20
Calculation of X:
(50 * 640) / 352 = 90
Calculation of Y:
(20 * 480) / 240 = 40
X = 90, Y = 40

**Note:** Some commands/parameters may not work with some cameras due to limitations of the communication interface itself developed by the camera's manufacturer.

**Example 1:** Center a camera on the position X = 20, Y = 30 at a speed of 50

```
http://192.168.0.1:8601/Interface/Cameras/PTZ/Simultaneous?Camera=Camera1&X=20&Y=30&Speed=50
```

**Example 2:** Moves a camera to the position X = 500, Y = 440 at a speed of 100 and response in text

```
http://192.168.0.1:8601/Interface/Cameras/PTZ/Simultaneous?Camera=Camera1&X=500&Y=440&Speed=100&ResponseFormat=Text
```

**Example 3:** Moves a camera to the position X = 10, Y = 350 with response in XML and authentication of the user User1

```
http://192.168.0.1:8601/Interface/Cameras/PTZ/Simultaneous?Camera=Camera1&X=10&Y=350&ResponseFormat=XML&AuthUser=User1&AuthPass=User1Pass
```

**Response:**
Default response of API. In case the "Absolute" command group is not supported by the camera's driver (due to restrictions in the communication interface itself developed by the camera's manufacturer), the error code 10105 (Command not supported by the PTZ driver) will be returned.

**HTTP Return:** 200 OK

**Parameters of return:** Default return of API

#### 4.3.7.6 Continuous

By way of the continuous movement command, you will be able to control the cameras emulating a joystick

**Compatibility:** All editions

**Security level:** Requires authentication of the user

**Method:** HTTP GET

**Syntax:**

```
http://<server_address>/Interface/Cameras/PTZ/Continuous?<argument=value>
[&<argument=value>...][&<general_argument>...]
```

**Arguments:**

<table>
<tr><th>Argument</th><th>Valid values</th><th>Description</th></tr>
<tr><td>Camera=<String>*</td><td>String</td><td>Name of the camera</td></tr>
<tr><td>Pan=<Integer></td><td>Integer. -100 >= X <= 100</td><td>Movement speed for Pan<br><table><tr><th>Type</th><th>Description</th></tr><tr><td>0</td><td>Stop</td></tr><tr><td>Negative values</td><td>Left</td></tr><tr><td>Positive values</td><td>Right</td></tr></table>If this parameter is omitted, the default value of 0 (Stop) will be used</td></tr>
<tr><td>Tilt=<Integer></td><td>Integer. -100 >= X <= 100</td><td>Movement speed for Tilt<br><table><tr><th>Type</th><th>Description</th></tr><tr><td>0</td><td>Stop</td></tr><tr><td>Negative values</td><td>Up</td></tr><tr><td>Positive values</td><td>Down</td></tr></table>If this parameter is omitted, the default value of 0 (Stop) will be used</td></tr>
<tr><td>Zoom=<Integer></td><td>Integer. -100 >= X <= 100</td><td>Movement speed for Zoom<br><table><tr><th>Type</th><th>Description</th></tr><tr><td>0</td><td>Stop</td></tr><tr><td>Negative values</td><td>Zoom Out</td></tr><tr><td>Positive values</td><td>Zoom In</td></tr></table>If this parameter is omitted, the default value of 0 (Stop) will be used</td></tr>
<tr><td>Focus=<Integer></td><td>Integer. -100 >= X <= 100</td><td>Movement speed for Focus<br><table><tr><th>Type</th><th>Description</th></tr><tr><td>0</td><td>Stop</td></tr><tr><td>Negative values</td><td>Focus Near</td></tr><tr><td>Positive values</td><td>Focus Far</td></tr></table>If this parameter is omitted, the default value of 0 (Stop) will be used</td></tr>
<tr><td>Iris=<Integer></td><td>Integer. -100 >= X <= 100</td><td>Movement Speed for Iris<br><table><tr><th>Type</th><th>Description</th></tr><tr><td>0</td><td>Stop</td></tr></table></td></tr>
</table>

| | | |
|---|---|---|
| | | Negative values: Close Iris<br>Positive values: Open Iris<br><br>If this parameter is omitted, the default value of 0 (Stop) will be used |

* Mandatory parameters

**Note:** When a movement command is sent, the camera will apply the movement until the command with the parameters for Stop is sent.

**Note:** Some commands/parameters may not work with some cameras due to limitations of the communication interface itself developed by the camera's manufacturer. The basic parameters of Up, Down, Left, Right, Zoom In and Zoom Out should work for all cameras, whereas the parameters of Simultaneous Movement, Home Position, Focus and Iris may vary from one manufacturer to another.

**Example 1:** Moves a camera to the right and downwards at Pan speed 50 and Tilt speed 60

```
http://192.168.0.1:8601/Interface/Cameras/PTZ/Continuous?Camera=Camera1&
Pan=50&Tilt=60
```

**Example 2:** Stops the Pan and Tilt movement of a camera with response in text

```
http://192.168.0.1:8601/Interface/Cameras/PTZ/Continuous?Camera=Camera1&
Pan=0&Tilt=0&ResponseFormat=Text
```

**Example 3:** Executes Zoom In in a camera at a speed of 100, response in XML and authentication of the user User1

```
http://192.168.0.1:8601/Interface/Cameras/PTZ/Continuous?Camera=Camera1&
Zoom=100&ResponseFormat=XML&AuthUser=User1&AuthPass=User1Pass
```

**Response:**
Default response of API. In case the "Continuous" command group is not supported by the camera's driver (due to restrictions in the communication interface itself developed by the camera's manufacturer), the error code 10105 (Command not supported by the PTZ driver) will be returned.

**HTTP Return:** 200 OK

**Parameters of return:** Default return of API

#### 4.3.7.7 Auto Focus

By way of this command, you will be able to activate or deactivate the Auto Focus.

**Compatibility:** All editions

**Security level:** Requires authentication of the user

**Method:** HTTP GET

**Syntax:**

```
http://<server_address>/Interface/Cameras/PTZ/AutoFocus?<argument=value>
[&<argument=value>...][&<general_argument>...]
```

**Arguments:**

| Argument | Valid values | Description |
|---|---|---|

| | | |
|---|---|---|
| Camera=<*String*>* | String | Name of the camera |
| Operation=<*String*>* | ON<br>OFF | Turn on or off the Auto Focus |

* Mandatory parameters

**Note:** Some commands/parameters may not work with some cameras due to limitations of the communication interface itself developed by the camera's manufacturer.

**Example 1:** Activates the auto focus of a camera with response in text

```
http://192.168.0.1:8601/Interface/Cameras/PTZ/AutoFocus?Camera=Camera1&
Operation=ON&ResponseFormat=Text
```

**Example 2:** Deactivates the auto focus of a camera with response in XML

```
http://192.168.0.1:8601/Interface/Cameras/PTZ/AutoFocus?Camera=Camera1&
Operation=OFF&ResponseFormat=XML
```

**Response:**
Default response of API. In case the "Absolute" command group is not supported by the camera's driver (due to restrictions in the communication interface itself developed by the camera's manufacturer), the error code 10105 (Command not supported by the PTZ driver) will be returned.

**HTTP Return:** 200 OK

**Parameters of return:** Default return of API

#### 4.3.7.8 Auto Iris

By way of this command, you will be able to activate or deactivate the Auto Iris.

**Compatibility:** All editions

**Security level:** Requires authentication of the user

**Method:** HTTP GET

**Syntax:**

```
http://<server_address>/Interface/Cameras/PTZ/AutoIris?<argument=value>
[&<argument=value>...][&<general_argument>...]
```

**Arguments:**

| Argument | Valid values | Description |
|---|---|---|
| Camera=<*String*>* | String | Name of the camera |
| Operation=<*String*>* | ON<br>OFF | Turn on or off the Auto Iris |

* Mandatory parameters

**Note:** Some commands/parameters may not work with some cameras due to limitations of the communication interface itself developed by the camera's manufacturer.

**Example 1:** Activates the auto iris of a camera with response in text

```
http://192.168.0.1:8601/Interface/Cameras/PTZ/AutoIris?Camera=Camera1&
Operation=ON&ResponseFormat=Text
```

**Example 2:** Deactivates the auto iris of a camera with response in XML

```
http://192.168.0.1:8601/Interface/Cameras/PTZ/AutoIris?Camera=Camera1&
Operation=OFF&ResponseFormat=XML
```

**Response:**
Default response of API. In case the "Absolute" command group is not supported by the camera's driver (due to restrictions in the communication interface itself developed by the camera's manufacturer), the error code 10105 (Command not supported by the PTZ driver) will be returned.

**HTTP Return:** 200 OK

**Parameters of return:** Default return of API

#### 4.3.7.9 Menu control

Some cameras (especially analog ones) allow the remote control of the menu. By way of this command, you will have total control over the menu of cameras.

**Compatibility:** All editions

**Security level:** Requires authentication of the user

**Method:** HTTP GET

**Syntax:**

```
http://<server_address>/Interface/Cameras/PTZ/MenuControl?<argument=value>
[&<argument=value>...][&<general_argument>...]
```

**Arguments:**

| Argument | Valid values | Description |
|---|---|---|
| Camera=<String>* | String | Name of the camera |
| Operation=<String>* | Open<br>Close<br>Left<br>Right<br>Up<br>Down<br>Enter<br>Cancel | Control operation to be executed |

| Operation | Description |
|---|---|
| Open | Open the menu |
| Close | Close the menu |
| Left | Move the cursor to the left |
| Right | Move the cursor to the right |
| Up | Move the cursor upwards |
| Down | Move the cursor downwards |
| Enter | "Enter" key |
| Cancel | Cancel the operation |

* Mandatory parameters

**Note:** Some commands/parameters may not work with some cameras due to limitations of the communication interface itself developed by the camera's manufacturer.

**Exemple 1:** Open the menu of a camera

```
http://192.168.0.1:8601/Interface/Cameras/PTZ/MenuControl?Camera=Camera1&
Operation=Open
```

**Example 2:** Close the menu of a camera with response in text

```
http://192.168.0.1:8601/Interface/Cameras/PTZ/MenuControl?Camera=Camera1&
Operation=Close&ResponseFormat=Text
```

**Example 3:** Move the menu's cursor downwards with response in XML and authentication of the user User1

```
http://192.168.0.1:8601/Interface/Cameras/PTZ/MenuControl?Camera=Camera1&
Operation=Down&ResponseFormat=XML&AuthUser=User1&AuthPass=User1Pass
```

**Response:**
Default response of API. In case the "Absolute" command group is not supported by the camera's driver (due to restrictions in the communication interface itself developed by the camera's manufacturer), the error code 10105 (Command not supported by the PTZ driver) will be returned.

**HTTP Return:** 200 OK

**Parameters of return:** Default return of API

#### 4.3.7.10 Presets

Commands for control of presets

##### 4.3.7.10.1 List of presets

Requests the list of presets of a camera.

**Compatibility:** All editions

**Security level:** Requires authentication of the user

**Method:** HTTP GET

**Syntax:**

```
http://<server_address>/Interface/Cameras/PTZ/GetPresets?<argument=value>
[&<argument=value>...][&<general_argument>...]
```

**Arguments:**

| Argument | Valid values | Description |
|---|---|---|
| Camera=<String>* | String | Name of the camera |

* Mandatory parameters

**Example 1:** Requests the list of presets of a camera with response in text

```
http://192.168.0.1:8601/Interface/Cameras/PTZ/GetPresets?Camera=Camera1&
ResponseFormat=Text
```

**Example 2:** Requests the list of presets of a camera with response in XML

```
http://192.168.0.1:8601/Interface/Cameras/PTZ/GetPresets?Camera=Camera1&
ResponseFormat=XML
```

**Response:**
A list with the presets of a specified camera is returned.

**HTTP Return:** 200 OK

**Parameters of return:**

**Fixed Parameters:**

| Parameter | Type | Description |
|---|---|---|
| COUNT | Integer | Total number of presets |

**Parameters of the list of presets:**

| Parameter | Type | Description |
|---|---|---|
| ID | Integer | Number of the preset |
| DESCRIPTION | String | Description of the preset |

**List of presets:**
The parameters of the list of presets will depend on the type of response (Text or XML).

**List of presets with response in text:**
The parameters of response in text will obey the following syntax:

```
PRESET_<num>_<field>=<value>
```

| Parameter | Description |
|---|---|
| *num* | Number of the record |
| *field* | Name of the field |
| *value* | Value of the field |

**Example of return in text:**

```
RESPONSE_CODE=0
RESPONSE_MESSAGE=OK
COUNT=3
PRESET_1_ID=0
PRESET_1_DESCRIPTION=Position 1
PRESET_2_ID=1
PRESET_2_DESCRIPTION=Position 2
PRESET_3_ID=2
PRESET_3_DESCRIPTION=Position 3
```

**List of presets with response in XML:**
The parameters of response in XML will obey the following syntax:

```
<Presets>
  <Count>COUNT</Count>
  <Preset>
    <ID>ID</ID>
    <Description>DESCRIPTION</Description>
  </Preset>
</Presets>
```

**Example of return in XML:**

```
<?xml version="1.0" encoding="UTF-8" ?>
<Response>
 <Code>0</Code>
 <Message>OK</Message>
 <Data>
  <Presets>
   <Count>3</Count>
   <Preset>
    <ID>0</ID>
    <Description>Position 1</Description>
   </Preset>
   <Preset>
    <ID>1</ID>
    <Description>Position 2</Description>
   </Preset>
   <Preset>
    <ID>2</ID>
    <Description>Position 3</Description>
   </Preset>
  </Presets>
 </Data>
</Response>
```

#### 4.3.7.10.2 Call a preset

Command for calling a preset of a camera.

**Compatibility:** All editions

**Security level:** Requires authentication of the user

**Method:** HTTP GET

**Syntax:**

```
http://<server_address>/Interface/Cameras/PTZ/CallPreset?<argument=value>
[&<argument=value>...][&<general_argument>...]
```

**Arguments:**

| Argument | Valid values | Description |
|---|---|---|
| Camera=<String>* | String | Name of the camera |
| Value=<Integer>* | Integer. X >= 0 | Number of the preset |
| Speed=<Integer> | Integer. 0 >= X <= 100 | Speed of operation. If this parameter is omitted, the default value of 100 will be used |

* Mandatory parameters

**Note:** Some commands/parameters may not work with some cameras due to limitations of the communication interface itself developed by the camera's manufacturer.

**Example 1:** Call the preset 10 with speed of 90

```
http://192.168.0.1:8601/Interface/Cameras/PTZ/CallPreset?Camera=Camera1&
Value=10&Speed=90
```

**Example 2:** Call the preset 15 with response in text

```
http://192.168.0.1:8601/Interface/Cameras/PTZ/CallPreset?Camera=Camera1&
Value=15&ResponseFormat=Text
```

**Example 3:** Call the preset 20 with response in XML and authentication of the user User1

```
http://192.168.0.1:8601/Interface/Cameras/PTZ/CallPreset?Camera=Camera1&
Value=20&ResponseFormat=XML&AuthUser=User1&AuthPass=User1Pass
```

**Response:**
Default response of API. In case the "Absolute" command group is not supported by the camera's driver (due to restrictions in the communication interface itself developed by the camera's manufacturer), the error code 10105 (Command not supported by the PTZ driver) will be returned.

**HTTP Return:** 200 OK

**Parameters of return:** Default return of API

#### 4.3.7.11 Call a pattern

Command for calling the pattern of a camera.

**Compatibility:** All editions

**Security level:** Requires authentication of the user

**Method:** HTTP GET

**Syntax:**

```
http://<server_address>/Interface/Cameras/PTZ/CallPattern?<argument=value>
[&<argument=value>...][&<general_argument>...]
```

**Arguments:**

| Argument | Valid values | Description |
|---|---|---|
| Camera=<String>* | String | Name of the camera |
| Value=<Integer>* | Integer. X >= 0 | Number of the pattern |
| Speed=<Integer> | Integer. 0 >= X <= 100 | Speed of operation. If this parameter is omitted, the default value of 100 will be used |

* Mandatory parameters

**Note:** Some commands/parameters may not work with some cameras due to limitations of the communication interface itself developed by the camera's manufacturer.

**Example 1:** Call the pattern 10 with speed of 90

```
http://192.168.0.1:8601/Interface/Cameras/PTZ/CallPattern?Camera=Camera1&
Value=10&Speed=90
```

**Example 2:** Call the pattern 15 with response in text

```
http://192.168.0.1:8601/Interface/Cameras/PTZ/CallPattern?Camera=Camera1&
Value=15&ResponseFormat=Text
```

**Example 3:** Call the pattern 20 with response in XML and authentication of the user User1

```
http://192.168.0.1:8601/Interface/Cameras/PTZ/CallPattern?Camera=Camera1&
Value=20&ResponseFormat=XML&AuthUser=User1&AuthPass=User1Pass
```

**Response:**
Default response of API. In case the "Absolute" command group is not supported by the camera's driver (due to restrictions in the communication interface itself developed by the camera's manufacturer), the error code 10105 (Command not supported by the PTZ driver) will be returned.

**HTTP Return:** 200 OK

**Parameters of return:** Default return of API

#### 4.3.7.12 Windshield wiper

By way of this command, you will be able to activate or deactivate the windshield wiper (if the camera has one).

**Compatibility:** All editions

**Security level:** Requires authentication of the user

**Method:** HTTP GET

**Syntax:**
```
http://<server_address>/Interface/Cameras/PTZ/Wiper?<argument=value>
[&<argument=value>...][&<general_argument>...]
```

**Arguments:**

| Argument | Valid values | Description |
|---|---|---|
| Camera=<String>* | String | Name of the camera |
| Operation=<String>* | ON<br>OFF | Turns the windshield wiper on or off |

* Mandatory parameters

**Note:** Some commands/parameters may not work with some cameras due to limitations of the communication interface itself developed by the camera's manufacturer.

**Example 1:** Activates the windshield wiper of a camera with response in text
```
http://192.168.0.1:8601/Interface/Cameras/PTZ/Wiper?Camera=Camera1&
Operation=ON&ResponseFormat=Text
```

**Example 2:** Deactivates the windshield wiper of a camera with response in XML
```
http://192.168.0.1:8601/Interface/Cameras/PTZ/Wiper?Camera=Camera1&
Operation=OFF&ResponseFormat=XML
```

**Response:**
Default response of API. In case the "Absolute" command group is not supported by the camera's driver (due to restrictions in the communication interface itself developed by the camera's manufacturer), the error code 10105 (Command not supported by the PTZ driver) will be returned.

**HTTP Return:** 200 OK

**Parameters of return:** Default return of API

#### 4.3.7.13 Auxiliary

With this command you can activate or deactivate the auxiliary controls the camera (If it has).

**Compatibility:** All editions

**Security level:** Requires authentication of the user

**Method:** HTTP GET

**Syntax:**

```
http://<server_address>/Interface/Cameras/PTZ/Auxiliary?<argument=value>
[&<argument=value>...][&<general_argument>...]
```

**Arguments:**

| Arguments | Valid values | Description |
|---|---|---|
| Camera=<String>* | String | Camera name |
| Operation=<String>* | ON<br>OFF | Turn on or off the auxiliary control |
| Value=<Integer>* | Integer | Auxiliary control number |

* Mandatory parameters

**Note:** Some commands/parameters may not work with some cameras due to limitations of the communication interface itself developed by the camera's manufacturer.

**Example 1:** Activates the auxiliary command 1 of a camera with response in text

```
http://192.168.0.1:8601/Interface/Cameras/PTZ/Auxiliary?Camera=Camera1&
Operation=ON&Value=1&ResponseFormat=Text
```

**Example 2:** Deactivates the auxiliary command 1 of a camera with response in XML

```
http://192.168.0.1:8601/Interface/Cameras/PTZ/Auxiliary?Camera=Camera1&
Operation=OFF&Value=1&ResponseFormat=XML
```

**Example 3:** Activates the auxiliary command 3 of a camera with response in XML

```
http://192.168.0.1:8601/Interface/Cameras/PTZ/Auxiliary?Camera=Camera1&
Operation=ON&Value=3&ResponseFormat=XML
```

**Response:**
Default response of API. In case the "Auxiliary" command group is not supported by the camera's driver (due to restrictions in the communication interface itself developed by the camera's manufacturer), the error code 10105 (Command not supported by the PTZ driver) will be returned.

**HTTP Return:** 200 OK

**Parameters of return:** Default return of API

#### 4.3.7.14 Patrol

Commands to control and query status of PTZ Patrol of a given camera

##### 4.3.7.14.1 Requesting the list of patrols

Request the list of PTZ Patrols of a given camera

**Compatibility:** Standard, Professional, Enterprise

**Security level:** Requires authentication of the user

**Method:** HTTP GET

**Syntax:**

```
http://<server_address>/Interface/Cameras/PTZ/GetPatrols?<argument=value>
[&<argument=value>...][&<general_argument>...]
```

**Arguments:**

| Argument | Valid values | Description |
|---|---|---|
| Camera=<String>* | String | Name of the camera |

* Mandatory parameters

**Example 1:** Request the list of PTZ Patrols of a camera with response in text

```
http://192.168.0.1:8601/Interface/Cameras/PTZ/GetPatrols?Camera=Camera1&
ResponseFormat=Text
```

**Example 2:** Request the list of PTZ Patrols of a camera with response in XML

```
http://192.168.0.1:8601/Interface/Cameras/PTZ/GetPatrols?Camera=Camera1&
ResponseFormat=XML
```

**Response:**
A list with the PTZ Patrols of a specified camera is returned.

**HTTP Return:** 200 OK

**Parameters of return:**

**Fixed Parameters:**

| Parameter | Type | Description |
|---|---|---|
| COUNT | Integer | Total number of patrols |

**Parameters of the list of patrols:**

| Parameter | Type | Description |
|---|---|---|
| ID | Integer | PTZ Patrol ID |
| NAME | String | PTZ Patrol Name |
| DESCRIPTION | String | PTZ Patrol Description |

**List of patrols:**
The parameters of the list of patrols will depend on the type of response (Text or XML).

**List of patrols with response in text:**
The parameters of response in text will obey the following syntax:

```
PATROL_<num>_<field>=<value>
```

| Parameter | Description |
|---|---|
| *num* | Number of the record |
| *field* | Name of the field |
| *value* | Value of the field |

**Example of return in text:**

```
RESPONSE_CODE=0
RESPONSE_MESSAGE=OK
COUNT=2
PATROL_1_ID=0
PATROL_1_NAME=Day
PATROL_1_DESCRIPTION=Daytime operation
PATROL_2_ID=1
PATROL_2_NAME=Night
PATROL_2_DESCRIPTION=Nighttime operation
```

**List of patrols with response in XML:**

The parameters of response in XML will obey the following syntax:

```
<Patrols>
 <Count>PATROL_COUNT</Count>
 <Patrol>
  <ID>PATROL_ID</ID>
  <Name>PATROL_NAME</Name>
  <Description>PATROL_DESCRIPTION</Description>
 </Patrol>
</Patrols>
```

**Example of return in XML:**

```
<?xml version="1.0" encoding="UTF-8" ?>
<Response>
 <Code>0</Code>
 <Message>OK</Message>
 <Data>
  <Patrols>
   <Count>2</Count>
   <Patrol>
    <ID>0</ID>
    <Name>Day</Name>
    <Description>Daytime operation</Description>
   </Patrol>
   <Patrol>
    <ID>1</ID>
    <Name>Night</Name>
    <Description>Nighttime operation</Description>
   </Patrol>
  </Patrols>
 </Data>
</Response>
```

4.3.7.14.2 Requesting the status of PTZ patrol

Command to query the current status of PTZ Patrol of a given camera

**Compatibility:** Standard, Professional, Enterprise

**Security level:** Requires authentication of the user

**Method:** HTTP GET

**Syntax:**

```
http://<server_address>/Interface/Cameras/PTZ/GetPatrolStatus?
<argument=value>[&<argument=value>...][&<general_argument>...]
```

**Arguments:**

| Argument | Valid values | Description |
|---|---|---|
| Camera=<String>* | String | Name of the camera |

* Mandatory parameters

**Example 1:** Query PTZ Patrol status of "Camera1"

```
http://192.168.0.1:8601/Interface/Cameras/PTZ/GetPatrolStatus?
Camera=Camera1
```

**Example 2:** Query PTZ Patrol status of "Camera1" with response in text

```
http://192.168.0.1:8601/Interface/Cameras/PTZ/GetPatrolStatus?
Camera=Camera1&ResponseFormat=Text
```

**Response:**
A list of parameter-value pairs is returned

**HTTP Return HTTP:** 200 OK

**Parameters of return:**

| Parameter | Type | Description |
|---|---|---|
| PATROL_TYPE | AUTO<br>MANUAL | Type of PTZ patrol |
| CURRENTPATROLID | Integer | ID of current PTZ patrol |
| CURRENTPATROLNAME | String | Name of current PTZ patrol |
| PAUSED | Boolean | Status of PTZ patrol<br><br>TRUE - PTZ patrol is paused<br>FALSE - PTZ patrol is running |

**Example of return in text:**

```
RESPONSE_CODE=0
RESPONSE_MESSAGE=OK
PATROLTYPE=AUTO
CURRENTPATROLID=0
CURRENTPATROLNAME=Day
PAUSED=FALSE
```

**Example of return in XML:**

```
<?xml version="1.0" encoding="UTF-8" ?>
<Response>
  <Code>0</Code>
  <Message>OK</Message>
  <Data>
    <Status>
      <PatrolType>AUTO</PatrolType>
      <CurrentPatrolID>0</CurrentPatrolID>
      <CurrentPatrolName>Day</CurrentPatrolName>
      <Paused>TRUE</Paused>
    </Status>
  </Data>
</Response>
```

#### 4.3.7.14.3 Controlling PTZ Patrol

Command to control the PTZ Patrol of a given camera

**Compatibility:** Standard, Professional, Enterprise

**Security level:** Requires authentication of the user

**Method:** HTTP GET

**Syntax:**

```
http://<server_address>/Interface/Cameras/PTZ/Patrol?
<argument=value>[&<argument=value>...][&<general_argument>...]
```

**Arguments:**

| Argument | Valid values | Description |
|---|---|---|
| `Camera=<String>*` | String | Name of the camera |
| `Action=<Boolean>*` | START<br>PAUSE<br>RESUME | START - Start a given patrol<br>PAUSE - Pause current patrol operation<br>RESUME - Resume current patrol operation |
| `ID=<Integer>**` | Integer | ID of the patrol<br><br>This parameter is mandatory when using `Action=START` |

* Mandatory parameters
** Conditionally mandatory parameters

**Example 1:** Start patrol 1 of "Camera1"

```
http://192.168.0.1:8601/Interface/Cameras/PTZ/Patrol?Camera=1&
Action=START&ID=1
```

**Example 2:** Pause current patrol of "Camera1"

```
http://192.168.0.1:8601/Interface/Cameras/PTZ/Patrol?Camera=1&
Action=PAUSE
```

**Example 3:** Resume current patrol of "Camera1"

```
http://192.168.0.1:8601/Interface/Cameras/PTZ/Patrol?Camera=1&
Action=RESUME
```

**Response:**

Default response of API.

**HTTP Return:** 200 OK

**Parameters of return:** Default return of API

#### 4.3.7.15 Supported commands

Returns a list of the PTZ commands that are supported by a given camera.

**Compatibility:** All editions

**Security level:** Requires authentication of the user

**Method:** HTTP GET

**Syntax:**

```
http://<server_address>/Interface/Cameras/PTZ/GetSupportedCommands?
<argument=value>[&<argument=value>...][&<general_argument>...]
```

**Arguments:**

| Argument | Valid values | Description |
|---|---|---|
| Camera=<String>* | String | Name of the camera |

* Mandatory parameters

**Example 1:** Requests the supported PTZ commands of a camera with response in text

```
http://192.168.0.1:8601/Interface/Cameras/PTZ/GetSupportedCommands?
Camera=Camera1&ResponseFormat=Text
```

**Example 2:** Requests the supported PTZ commands PTZ of a camera with response in XML and authentication of the user User1

```
http://192.168.0.1:8601/Interface/Cameras/PTZ/GetSupportedCommands?
Camera=Camera1&ResponseFormat=XML&AuthUser=User1&AuthPass=User1Pass
```

**Response:**
A list of parameter-value pairs is returned

**HTTP Return:** 200 OK

**Parameters of return:**

| Parameter | Type | Description |
|---|---|---|
| SIMPLE | Boolean | Support for the commands of the type "Simple" |
| RELATIVE | Boolean | Support for the commands of the type "Relative" |
| ABSOLUTE | Boolean | Support for the commands of the type "Absolute" |
| AREAZOOM | Boolean | Support for the commands of the type "AreaZoom" |
| SIMULTANEOUS | Boolean | Support for the commands of the type "Simultaneous" |
| CONTINUOUS | Boolean | Support for the commands of the type "Continuous" |
| AUTOFOCUS | Boolean | Support for the commands of the type "Auto Focus" |
| AUTOIRIS | Boolean | Support for the commands of the type "Auto Iris" |
| MENUCONTROL | Boolean | Support for the commands of the type "Control of Menu" |
| CALLPATTERN | Boolean | Support for the commands of the type "Call a Pattern" |
| CALLPRESET | Boolean | Support for the commands of the type "Call a Preset" |
| WIPER | Boolean | Support for the commands of the type "Windshield Wiper" |

| Parameter | Type | Description |
|---|---|---|
| AUXILIARY | Boolean | Support for the commands of the type "Auxiliary" |

**Example of return in text:**

```
RESPONSE_CODE=0
RESPONSE_MESSAGE=OK
SIMPLE=TRUE
RELATIVE=FALSE
ABSOLUTE=TRUE
AREAZOOM=FALSE
SIMULTANEOUS=FALSE
CONTINUOUS=TRUE
AUTOFOCUS=TRUE
AUTOIRIS=TRUE
MENUCONTROL=TRUE
CALLPATTERN=TRUE
CALLPRESET=TRUE
WIPER=TRUE
AUXILIARY=TRUE
```

**Example of return in XML:**

```
<?xml version="1.0" encoding="UTF-8" ?>
<Response>
  <Code>0</Code>
  <Message>OK</Message>
  <Data>
    <Commands>
      <Simple>TRUE</Simple>
      <Relative>FALSE</Relative>
      <Absolute>TRUE</Absolute>
      <AreaZoom>FALSE</AreaZoom>
      <Simultaneous>FALSE</Simultaneous>
      <Continuous>TRUE</Continuous>
      <AutoFocus>TRUE</AutoFocus>
      <AutoIris>TRUE</AutoIris>
      <MenuControl>TRUE</MenuControl>
      <CallPattern>TRUE</CallPattern>
      <CallPreset>TRUE</CallPreset>
      <Wiper>TRUE</Wiper>
      <Auxiliary>TRUE</Auxiliary>
    </Commands>
  </Data>
</Response>
```

## 4.3.8 I/O

Commands to control camera I/Os

### 4.3.8.1 Requesting the status of the input ports

Requests the status of the alarm input ports of a camera.

**Compatibility:** Standard, Professional, Enterprise

**Security level:** Admin

**Method:** HTTP GET

**Syntax:**

```
http://<server_address>/Interface/Cameras/IO/GetInputPortStatus?
<argument=value>[&<argument=value>...][&<general_argument>...]
```

**Arguments:**

| Argument | Valid values | Description |
|---|---|---|
| Camera=<*String*>* | String | Nome da câmera |

* Mandatory parameters

**Example 1:** Request the status of the alarm input ports of a camera with response in text

```
http://192.168.0.1:8601/Interface/Cameras/IO/GetInputPortStatus?
Camera=Camera1&ResponseFormat=Text
```

**Example 2:** Request the status of the alarm input ports of a camera with response in XML

```
http://192.168.0.1:8601/Interface/Cameras/IO/GetInputPortStatus?
Camera=Camera1&ResponseFormat=XML
```

**Response:**
A list with the status of all of the alarm input ports of the specified camera is returned.

**HTTP Return:** 200 OK

**Parameters of return:**

**Fixed Parameters:**

| Parameter | Type | Description |
|---|---|---|
| COUNT | Integer | Total number of alarm input ports |

**Parameters of the list of ports:**

| Parameter | Type | Description |
|---|---|---|
| STATE | String | State of the input port<br><br>Type / Description:<br>SHORT — Port closed<br>OPEN — Port open<br>UNKNOWN — State unknown |

**List of the alarm input ports:**
The parameters of the list of alarm input ports will depend on the type of response (Text or XML).

**List of alarm input ports with response in text:**
The parameters of response in text will obey the following syntax:

```
PORT_<num>_<field>=<value>
```

| Parameter | Description |
|---|---|
| *num* | Number of the record |
| *field* | Name of the field |

| Parameter | Description |
|---|---|
| *value* | Value of the field |

**Example of return in text:**

```
RESPONSE_CODE=0
RESPONSE_MESSAGE=OK
COUNT=4
PORT_1_STATE=OPEN
PORT_2_STATE=OPEN
PORT_3_STATE=SHORT
PORT_4_STATE=OPEN
```

**List of alarm input ports with response in XML:**

The parameters of response in XML will obey the following syntax:

```
<Ports>
 <Count>COUNT</Count>
 <Port>
  <State>STATE</State>
 </Port>
</Ports>
```

**Example of return in XML:**

```
<?xml version="1.0" encoding="UTF-8" ?>
<Response>
 <Code>0</Code>
 <Message>OK</Message>
 <Data>
  <Ports>
   <Count>4</Count>
   <Port>
    <State>OPEN</State>
   </Port>
   <Port>
    <State>OPEN</State>
   </Port>
   <Port>
    <State>SHORT</State>
   </Port>
   <Port>
    <State>OPEN</State>
   </Port>
  </Ports>
 </Data>
</Response>
```

#### 4.3.8.2 Requesting the status of input events

Requests the status of alarm input events of a camera.

**Compatibility:** Standard, Professional, Enterprise

**Security level:** Admin

**Method:** HTTP GET

**Syntax:**

```
http://<server_address>/Interface/Cameras/IO/GetInputEventStatus?
<argument=value>[&<argument=value>...][&<general_argument>...]
```

**Arguments:**

| Argument | Valid values | Description |
|---|---|---|
| Camera=<*String*>* | String | Name of the camera |

* Mandatory parameters

**Example 1:** Request the status of alarm input events of camera with response in text

```
http://192.168.0.1:8601/Interface/Cameras/IO/GetInputEventStatus?
Camera=Camera1&ResponseFormat=Text
```

**Example 2:** Request the status of the alarm input events of camera with response in XML

```
http://192.168.0.1:8601/Interface/Cameras/IO/GetInputEventStatus?
Camera=Camera1&ResponseFormat=XML
```

**Response:**
A list with the status of all of the alarm input events of the specified camera is returned.

**HTTP Return:** 200 OK

**Parameters of return:**

**Fixed parameters:**

| Parameter | Type | Description |
|---|---|---|
| COUNT | Integer | Total number of alarm input events |

**Parameters of the list of events:**

| Parameter | Type | Description |
|---|---|---|
| NAME | String | Name of the event |
| STATE | String | State of the input event<br><br>Type: ACTIVE — Event active (Occurring at the moment)<br>Type: INACTIVE — Event inactive (Not occurring) |

**List of the alarm input events:**
The parameters of the list of alarm input events will depend on the type of response (Text or XML).

**List of alarm input events with response in text:**
The parameters of response in text will obey the following syntax:

```
EVENT_<num>_<field>=<value>
```

| Parameter | Description |
|---|---|
| *num* | Number of the record |
| *field* | Name of the field |
| *value* | Value of the field |

**Example of return in text:**

```
RESPONSE_CODE=0
RESPONSE_MESSAGE=OK
COUNT=2
EVENT_1_NAME=Event 1
EVENT_1_STATE=ACTIVE
EVENT_2_NAME=Event 2
EVENT_2_STATE=INACTIVE
```

**List of alarm input events with response in XML:**

Os parâmetros de resposta em XML irão seguir a seguinte sintaxe:

```
<Events>
 <Count>COUNT</Count>
 <Event>
  <Name>NAME</Name>
  <State>STATE</State>
 </Event>
</Events>
```

**Example of return in XML:**

```
<?xml version="1.0" encoding="UTF-8" ?>
<Response>
 <Code>0</Code>
 <Message>OK</Message>
 <Data>
  <Events>
   <Count>2</Count>
   <Event>
    <Name>Event 1</Name>
    <State>ACTIVE</State>
   </Event>
   <Event>
    <Name>Event 2</Name>
    <State>INACTIVE</State>
   </Event>
  </Events>
 </Data>
</Response>
```

#### 4.3.8.3 Requesting the status of output ports

Requests the status of the alarm output ports of a camera.

**Compatibility:** Standard, Professional, Enterprise

**Security level:** Admin

**Method:** HTTP GET

**Syntax:**

```
http://<server_address>/Interface/Cameras/IO/GetOutputPortStatus?
<argument=value>[&<argument=value>...][&<general_argument>...]
```

**Arguments:**

| Argument | Valid values | Description |
|---|---|---|
| `Camera=<String>*` | String | Name of the camera |

* Mandatory parameters

**Example 1:** Request the status of the alarm output ports of a camera with response in text

```
http://192.168.0.1:8601/Interface/Cameras/IO/GetOutputPortStatus?
Camera=Camera1&ResponseFormat=Text
```

**Example 2:** Request the status of alarm output ports of a camera with response in XML

```
http://192.168.0.1:8601/Interface/Cameras/IO/GetOutputPortStatus?
Camera=Camera1&ResponseFormat=XML
```

**Response:**
A list with the status of all of the alarm output ports of the specified camera is returned.

**HTTP Return:** 200 OK

**Parameters of return:**

**Fixed parameters:**

| Parameter | Type | Description |
|---|---|---|
| COUNT | Integer | Total number of alarm output ports |

**Parameters of the list of ports:**

| Parameter | Type | Description |
|---|---|---|
| STATE | String | State of the output port<br><br>Type: SHORT — Port closed<br>Type: OPEN — Port open<br>Type: UNKNOWN — State unknown |

**List of alarm output ports:**
The parameters of the list of alarm output ports will depend on the type of response (Text or XML).

**List of alarm output ports with response in text:**
The parameters of response in text will obey the following syntax:

```
PORT_<num>_<field>=<value>
```

| Parameter | Description |
|---|---|
| *num* | Number of the record |
| *field* | Name of the field |
| *value* | Value of the field |

**Example of return in text:**

```
RESPONSE_CODE=0
RESPONSE_MESSAGE=OK
COUNT=4
PORT_1_STATE=OPEN
PORT_2_STATE=OPEN
PORT_3_STATE=SHORT
PORT_4_STATE=OPEN
```

**List of alarm output ports with response in XML:**
The parameters of response in XML will obey the following syntax:

```
<Ports>
 <Count>COUNT</Count>
 <Port>
  <State>STATE</State>
 </Port>
</Ports>
```

**Example of return in XML:**

```
<?xml version="1.0" encoding="UTF-8" ?>
<Response>
 <Code>0</Code>
 <Message>OK</Message>
 <Data>
  <Ports>
   <Count>4</Count>
   <Port>
    <State>OPEN</State>
   </Port>
   <Port>
    <State>OPEN</State>
   </Port>
   <Port>
    <State>SHORT</State>
   </Port>
   <Port>
    <State>OPEN</State>
   </Port>
  </Ports>
 </Data>
</Response>
```

#### 4.3.8.4 Requesting the list of output actions

Requests the list of output actions of a camera.

**Compatibility:** Standard, Professional, Enterprise

**Security level:** Admin

**Method:** HTTP GET

**Syntax:**

```
http://<server_address>/Interface/Cameras/IO/GetOutputActions?
<argument=value>[&<argument=value>...][&<general_argument>...]
```

**Arguments:**

| Argument | Valid values | Description |
|---|---|---|
| Camera=<*String*>* | String | Name of the camera |
| Actions=<*APIMasks*> | String | Mask to filter the results. Specify which actions must be returned based on the provided masks. |

* Mandatory parameter

**Example 1:** Request the list of output actions of a camera with response in text

```
http://192.168.0.1:8601/Interface/Cameras/IO/GetOutputActions?
Camera=Camera1&ResponseFormat=Text
```

**Example 2:** Request the list of output actions of a camera with response in XML

```
http://192.168.0.1:8601/Interface/Cameras/IO/GetOutputActions?
Camera=Camera1&ResponseFormat=XML
```

**Example 3:** Request the list of output actions that starts with letter "A" of a camera with response in XML

```
http://192.168.0.1:8601/Interface/Cameras/IO/GetOutputActions?
Camera=Camera1&Actions=A*&ResponseFormat=XML
```

**Response:**
A list with all of the output actions of the specified camera is returned.

**HTTP Return:** 200 OK

**Parameters of return:**

**Fixed parameters:**

| Parameter | Type | Description |
|---|---|---|
| COUNT | Integer | Total number of output actions |

**Parameters of the list of actions:**

| Parameter | Type | Description |
|---|---|---|
| NAME | String | Name of the action |

**List of output actions:**
The parameters of the list of output actions will depend on the type of response (Text or XML).

**List of output actions with response in text:**
The parameters of response in text will obey the following syntax:

```
ACTION_<num>_<field>=<value>
```

| Parameter | Description |
|---|---|
| *num* | Number of the record |
| *field* | Name of the field |
| *value* | Value of the field |

**Example of return in text:**

```
RESPONSE_CODE=0
RESPONSE_MESSAGE=OK
COUNT=2
ACTION_1_NAME=Action 1
ACTION_2_NAME=Action 2
```

**List of output actions with response in XML:**
The parameters of response in XML will obey the following syntax:

```
<Actions>
 <Count>COUNT</Count>
 <Action>
  <Name>NAME</Name>
 </Action>
</Actions>
```

**Example of return in XML:**

```
<?xml version="1.0" encoding="UTF-8" ?>
<Response>
 <Code>0</Code>
 <Message>OK</Message>
 <Data>
  <Actions>
   <Count>2</Count>
   <Action>
    <Name>Action 1</Name>
   </Action>
   <Action>
    <Name>Action 2</Name>
   </Action>
  </Actions>
 </Data>
</Response>
```

#### 4.3.8.5 Triggering an output action

By way of this command you will be able to trigger a script of output actions.

**Compatibility:** Standard, Professional, Enterprise

**Security level:** Admin

**Method:** HTTP GET

**Syntax:**

```
http://<server_address>/Interface/Cameras/IO/TriggerOutputAction?
<argument=value>[&<argument=value>...][&<general_argument>...]
```

**Arguments:**

| Argument | Valid values | Description |
|---|---|---|
| Camera=<String>* | String | Name of the camera |
| Action=<String>* | String | Name of the action |

* Mandatory parameters

**Example 1:** Activate an output action of a camera with response in text

```
http://192.168.0.1:8601/Interface/Cameras/IO/TriggerOutputAction?
Camera=Camera1&Action=Action1&ResponseFormat=Text
```

**Example 2:** Activate an output action of a camera with response in XML

```
http://192.168.0.1:8601/Interface/Cameras/IO/TriggerOutputAction?
Camera=Camera1&Action=Action1&ResponseFormat=XML
```

**Response:**
Default response of API.

**HTTP Return:** 200 OK

**Parameters of return:** Default return of API

## 4.3.9 Motion detection

Commands for camera motion detection

### 4.3.9.1 Notifying motion detection

Notify movement from a camera.

**Note:** This command must be used directly by cameras which support motion detection notification by HTTP

**Tip:** To better understand the mechanism of camera motion detection notification, consult the document "Using Hardware Motion Detection"

**Compatibility:** All editions

**Security level:** Use the camera username and password configured in its register on the system

**Method:** HTTP GET

**Syntax:**

```
http://<server_address>/Interface/Cameras/MotionDetection/Notify?
<argument=value>[&<argument=value>...][&<general_argument>...]
```

**Arguments:**

| Arguments | Valid values | Description |
|---|---|---|
| Camera=<String>* | String | Camera name |
| Motion=<String> | Start<br>End<br>Instant | Type of motion detection notification<br><br>Name / Description<br>Start — Motion started<br>End — Motion ended<br>Instant — Motion (Must be sent constantly)<br><br>If this parameter is omitted, the default value INSTANT will be used |

* Mandatory fields

**Usage:**
There are two ways of using this command: By using `Motion=Start` and `Motion=End` parameters and by using `Motion=Instant` parameter.

`Motion=Start` and `Motion=End` parameters must be used by cameras that notifies the start and end of motion. When the system receives `Motion=Start` parameter, it will start the camera recording and will keep recording it until `Motion=End` parameter is received, therefore if the system doesn´t receive `Motion=End` parameter, the recording will not stop.

`Motion=Instant`, on the other hand, will start the camera recording and will stop it as soon as the post alarm buffer (configured on system) is full, therefore the cameras that doesn´t notify the start and end of motion, must notify motion by using this parameter on a frequency not inferior than the post alarm buffer size.

**Example 1:** Notify motion detection on camera Camera1, using its access credentials ROOT and PASS, with response in XML

```
http://192.168.0.1:8601/Interface/Cameras/MotionDetection/Notify?
Camera=Camera1&AuthUser=root&AuthPass=pass&ResponseFormat=XML
```

**Example 2:** Notify the start of motion on camera Camera1, using its access credentials ROOT and PASS, with response in text

```
http://192.168.0.1:8601/Interface/Cameras/MotionDetection/Notify?
Camera=Camera1&Motion=Start&AuthUser=root&AuthPass=pass&ResponseFormat=Text
```

**Example 3:** Notify the end of motion on camera Camera1, using its access credentials ROOT and PASS, with response in XML

```
http://192.168.0.1:8601/Interface/Cameras/MotionDetection/Notify?
Camera=Camera1&Motion=End&AuthUser=root&AuthPass=pass&ResponseFormat=XML
```

**Response:**
Default response of API.

**HTTP Return:** 200 OK

**Parameters of return:** Default return of API

## 4.3.10 Manual events

Commands to control camera manual events

### 4.3.10.1 Requesting the list of manual events

Requests the list of global events that the user has rights to access.

**Compatibility:** Professional, Enterprise

**Security level:** Requires authentication of the user with rights to trigger manual events

**Method:** HTTP GET

**Syntax:**

```
http://<server_address>/Interface/Cameras/ManualEvents/GetManualEvents
[?<argument=value>[&<argument=value>...]][&<general_argument>...]
```

**Arguments:**

| Argument | Valid values | Description |
|---|---|---|
| `Camera=<String>*` | String | Camera name |
| `ManualEvents=<APIMasks>` | String | Mask to filter the results. Specify which manual events must be returned based on the provided masks. |
| `Fields=<String>` | Name<br>Description | Specifies the list of desired fields. If this parameter is omitted, all of the fields will be sent.<br><br>The fields must be separated by commas<br><br>Name: Name of the manual event<br>Description: Description of the manual event |

**Example 1:** Request the list of manual events from camera 1 with all of the fields and response in XML

```
http://192.168.0.1:8601/Interface/Cameras/ManualEvents/GetManualEvents?
Camera=01&ResponseFormat=XML
```

**Example 2:** Request the list of manual events from camera 1 with all of the fields and response in text

```
http://192.168.0.1:8601/Interface/Cameras/ManualEvents/GetManualEvents?
Camera=01&ResponseFormat=Text
```

**Example 3:** Request the list of manual events from camera 2 with only name, response in XML and authentication of the user Admin

```
http://192.168.0.1:8601/Interface/Cameras/ManualEvents/GetManualEvents?
Camera=02&Fields=Name&ResponseFormat=XML&AuthUser=admin&AuthPass=pass
```

**Example 4:** Request the list of manual events starting with letter "A" from camera 2, with only name, response in XML and authentication with Admin user

```
http://192.168.0.1:8601/Interface/Cameras/ManualEvents/GetManualEvents?
Camera=02&ManualEvents=A*&Fields=Name&ResponseFormat=XML&AuthUser=admin&
AuthPass=pass
```

**Response:**
A list with all of the manual events from the specified camera is returned. The fields returned in the list will depend on the values informed in the argument `Fields`

**HTTP Return:** 200 OK

**Parameters of return:**

**Fixed parameters:**

| Parameter | Type | Description |
|---|---|---|
| `COUNT` | Integer | Total number of manual events |

**Parameters of the list of manual events:**

| Parameter | Type | Description |
|---|---|---|
| `NAME` | String | Name of the manual event |
| `DESCRIPTION` | String | Description of the manual event |

**List of manual events:**

The parameters of the list of manual events will depend on the type of response (Text or XML).

**List of manual events with response in text:**
The parameters of response in text will obey the following syntax:

```
MANUALEVENT_<num>_<field>=<value>
```

| Parameter | Description |
|---|---|
| *num* | Number of the record |
| *field* | Name of the field |
| *value* | Value of the field |

**Example of return in text:**

```
RESPONSE_CODE=0
RESPONSE_MESSAGE=OK
COUNT=2
MANUALEVENT_1_NAME=Event1
MANUALEVENT_1_DESCRIPTION=Event 1
MANUALEVENT_2_NAME=Event2
MANUALEVENT_2_DESCRIPTION=Event 2
```

**List of manual events with response in XML:**
The parameters of response in XML will obey the following syntax:

```
<ManualEvents>
  <Count>COUNT</Count>
  <ManualEvent>
    <Name>NAME</Name>
    <Description>DESCRIPTION</Description>
  </ManualEvent>
</ManualEvents>
```

**Example of return in XML:**

```
<?xml version="1.0" encoding="UTF-8" ?>
<Response>
  <Code>0</Code>
  <Message>OK</Message>
  <Data>
    <ManualEvents>
      <Count>2</Count>
      <ManualEvent>
        <Name>Event1</Name>
        <Description>Event 1</Description>
      </ManualEvent>
      <ManualEvent>
        <Name>Event2</Name>
        <Description>Event 2</Description>
      </ManualEvent>
    </ManualEvents>
  </Data>
</Response>
```

#### 4.3.10.2 Triggering a manual event

By way of this command, you will be able to trigger a manual event from a camera.

**Compatibility:** Standard, Professional, Enterprise

**Security level:** Requires authentication of the user with rights to trigger manual events

**Method:** HTTP GET

**Syntax:**

```
http://<server_address>/Interface/Cameras/ManualEvents/TriggerManualEvent?
<argument=value>[&<argument=value>...][&<general_argument>...]
```

**Arguments:**

| Argument | Valid values | Description |
|---|---|---|
| Camera=<*String*>* | String | Name of the camera |
| Event=<*String*>* | String | Name of the global event |
| OverrideEmailMessage=<*String*> | String | Overrides an e-mail message, in case the action of e-mail transmission for the event is selected |
| OverrideOperatorMessage=<*String*> | String | Overrides an instant message sent to the operators, in case the action of message transmission to the operators is selected |
| OverrideShowObjects=<*String*> | String | Overrides the list of objects to be sent to the operator inside an alarm popup. See below how the string must be generated. |

* Mandatory parameters

**OverrideShowObjects**
The list of objects to must be formated in the following way:

```
OBJECT_ID_1;OBJECT_NAME_1,OBJECT_ID_2;OBJECT_NAME_2
```

Being, OBJECT_ID the ID of the object and OBJECT_NAME the name of the objects. The below table displays the possible values for OBJECT_ID:

| OBJECT_ID | Description |
|---|---|
| 1 | Camera |
| 9 | Map |
| 12 | Analytics configuration |
| 13 | LPR configuration |

For example, to show cameras Camera1, Camera2 and the maps Map1 e Map2, the list must be defined as:

```
1;Camera1,1;Camera2,9;Map1,9;Map2
```

**Example 1:** Trigger the manual event "Event1" of camera 1 with response in text

```
http://192.168.0.1:8601/Interface/Cameras/ManualEvents/TriggerManualEvent?
Camera=01&Event=Event1&ResponseFormat=Text
```

**Example 2:** Trigger the manual event "Event2" of camera 1 with response in XML

```
http://192.168.0.1:8601/Interface/Cameras/ManualEvents/TriggerManualEvent?
Camera=01&Event=Event2&ResponseFormat=XML
```

**Example 3:** Trigger the manual event "Event1" of camera 2 with response in XML and authentication with user Admin

```
http://192.168.0.1:8601/Interface/Cameras/ManualEvents/TriggerManualEvent?
Camera=02&Event=Event1&ResponseFormat=XML&AuthUser=admin&AuthPass=pass
```

**Example 4:** Trigger the manual event "Event1" of camera 2 overriding the message to be sent to the operators with the message "External alarm" and overriding the objects to the displayed by Camera1, Camera2, Map1, Map2, with response in XML and authentication with user Admin

```
http://192.168.0.1:8601/Interface/Cameras/ManualEvents/TriggerManualEvent?
Camera=02&Event=Event1&OverrideOperatorMessage=External alarm&
OverrideShowObjects=1;Camera1,1;Camera2,9;Map1,9;Map2&
ResponseFormat=XML&AuthUser=admin&AuthPass=pass
```

**Response:**
Default response of API.

**HTTP Return:** 200 OK

**Parameters of return:** Default return of API

## 4.3.11 Bookmarks

Commands to control camera bookmarks

### 4.3.11.1 Adding a new bookmark

Create a new bookmark in camera recordings

**Compatibility:** Professional, Enterprise

**Security level:** Requires user authentication with rights to create bookmarks

**Method:** HTTP GET

**Syntax:**

```
http://<server_address>/Interface/Cameras/Bookmarks/Add?
<argument=value>[&<argument=value>...][&<general_argument>...]
```

**Arguments:**

| Argument | Valid values | Description |
|---|---|---|
| Title=<*String*>* | String | Bookmark title |
| Color=<*String*>* | Red<br>Yellow<br>Blue<br>Navy<br>Aqua<br>Green<br>Lime<br>Fuchsia<br>Purple<br>Maroon | Bookmark color |

| | | |
|---|---|---|
| | Olive<br>Teal | |
| StartDate=<*APIDate*>* | Data | Bookmark start date |
| StartTime=<*APITime*>* | Hora | Bookmark start time |
| Cameras=<*String*>* | String | List of cameras that will be part of the new bookmark. Camera names must be comma-separated. |
| EndDate=<*APIDate*> | Data | Bookmark end date. If omitted, the start date will be used |
| EndTime=<*APITime*> | Hora | Bookmark end time. If omitted, the start time will be used |
| Remarks=<*String*> | String | Bookmark remarks |

* Mandatory parameters

**Example 1:** Add a punctual bookmark (Containing only start date and time) "Alarm" of red color, with start date and time of March 07, 2014 15:48:34.450 for cameras "Camera1" and "Camera2"

```
http://192.168.0.1:8601/Interface/Cameras/Bookmarks/Add?Title=Alarm&
Color=Red&StartDate=2014.03.07&StartTime=15.58.34.450&
Cameras=Camera1,Camera2
```

**Example 2:** Add a ranged bookmark with title "Door open" of blue color with start date and time of Mark 07, 2014 16:00:04.510 and final date and time of March 07, 2014 16:10:23.100 for cameras "Camera1" and "Camera2" and remark "Front door is open"

```
http://192.168.0.1:8601/Interface/Cameras/Bookmarks/Add?Title=Door%20open&
Color=Blue&StartDate=2014.03.07&StartTime=16.00.04.510&
EndDate=2014.03.07&EndTime=16.10.23.100&Cameras=Camera1,Camera2&
Remarks=Front%20door%20is%20open
```

**Response:**
Default response of API.

**HTTP Return:** 200 OK

**Parameters of return:** Default return of API

### 4.3.11.2 Searching bookmarks

Perform a search for bookmarks in the database

**Compatibility:** Professional, Enterprise

**Security level:** Requires user authentication with rights to search for bookmarks

**Method:** HTTP GET

**Syntax:**

```
http://<server_address>/Interface/Cameras/Bookmarks/Search
[?<argument=value>[&<argument=value>...][&<general_argument>...]]
```

**Arguments:**

| Argument | Valid values | Description |
|---|---|---|
| Keyword=<*String*> | String | Keyword to search on bookmarks |
| KeywordExact=<*Boolean*> | TRUE | TRUE - Search for exact keyword only |

| | | |
|---|---|---|
| | FALSE | FALSE - Search for keyword contained inside bookmark<br><br>If omitted, the default value FALSE will be used |
| SearchRemarks=<*Boolean*> | TRUE<br>FALSE | Search in remarks as well (Instead of only searching by title)<br><br>If omitted, the default value FALSE will be used<br><br>Note: Search might be slower when activating this parameter |
| Colors=<*String*> | Red<br>Yellow<br>Blue<br>Navy<br>Aqua<br>Green<br>Lime<br>Fuchsia<br>Purple<br>Maroon<br>Olive<br>Teal | Search for bookmarks of the specified colors only.<br><br>You can specify multiple colors. In this case, the color names must be comma-separated. |
| StartDate=<*APIDate*> | Date | Start date for date / time range |
| StartTime=<*APITime*> | Time | Start time for date / time range |
| EndDate=<*APIDate*> | Date | End date for date / time range. If this parameter is omitted, the value specified in StartDate parameter will be used |
| EndTime=<*APITime*> | Time | End time for date / time range. If this parameter is omitted, the value specified in StartTime parameter will be used |
| Cameras=<*String*> | String | List of cameras from bookmark. The list with camera names must be comma-separated. |

**Example 1:** Search for all bookmarks that contains the word "Alarm"

```
http://192.168.0.1:8601/Interface/Cameras/Bookmarks/Search?Keyword=Alarm
```

**Example 2:** Search for all red and blue bookmarks

```
http://192.168.0.1:8601/Interface/Cameras/Bookmarks/Search?Color=Red,Blue
```

**Example 3:** Search for all red bookmarks of camera "Camera1" that has the keyword "Alarm", with response in text

```
http://192.168.0.1:8601/Interface/Cameras/Bookmarks/Search?Keyword=Alarm&
Cameras=Camera1&Color=Red&ResponseFormat=Text
```

**Response:**
A list with all found bookmarks will be returned.

**HTTP Return:** 200 OK

**Return parameters:**

**Fixed parameters:**

| Parameter | Type | Description |
|---|---|---|
| COUNT | Integer | Bookmark count |

**Bookmark list parameters:**

| Parameter | Type | Description |
|---|---|---|
| TITLE | String | Bookmark title |
| COLOR | Red<br>Yellow<br>Blue<br>Navy<br>Aqua<br>Green<br>Lime<br>Fuchsia<br>Purple<br>Maroon<br>Olive<br>Teal | Bookmark color |
| STARTDATE | APITimestamp | Bookmark start date and time |
| ENDDATE | APITimestamp | Bookmark end date and time |
| REMARKS | String | Bookmark remarks |
| USER | String | User that created the bookmark |
| CAMERAS | String | List of cameras of which the bookmark belongs to (Comma-separated) |

**List of bookmarks:**
The parameters of the list of bookmarks will depend on the response format (Text or XML).

**List of bookmarks with response in text:**
The parameters of response in text will obey the following syntax:

```
BOOKMARK_<num>_<field>=<value>
```

| Parameter | Description |
|---|---|
| *num* | Number of the record |
| *field* | Name of the field |
| *value* | Value of the field |

**Example of result in text format:**

```
RESPONSE_CODE=0
RESPONSE_MESSAGE=OK
COUNT=2
BOOKMARK_1_TITLE=Test
BOOKMARK_1_COLOR=Red
BOOKMARK_1_STARTDATE=2014-06-03 10:10:01.302
BOOKMARK_1_ENDDATE=2014-06-03 10:10:01.302
BOOKMARK_1_REMARKS=
BOOKMARK_1_USER=admin
BOOKMARK_1_CAMERAS=Camera1
BOOKMARK_2_TITLE=Alarm
BOOKMARK_2_COLOR=Red
BOOKMARK_2_STARTDATE=2014-03-07 15:58:34.450
BOOKMARK_2_ENDDATE=2014-03-07 15:58:34.450
BOOKMARK_2_REMARKS=
BOOKMARK_2_USER=admin
BOOKMARK_2_CAMERAS=Camera1,Camera2
```

**List of bookmarks with response in XML:**
The parameters of response in XML will obey the following syntax:

```
<Bookmarks>
 <Count>COUNT</Count>
 <Bookmark>
  <Title>TITLE</Title>
  <Color>COLOR</Color>
  <StartDate>STARTDATE</StartDate>
  <EndDate>ENDDATE</EndDate>
  <Remarks>REMARKS</Remarks>
  <User>USER</User>
  <Cameras>CAMERAS</Cameras>
 </Bookmark>
</Bookmarks>
```

**Example of return in XML:**

```
<?xml version="1.0" encoding="UTF-8" ?>
<Response>
 <Code>0</Code>
 <Message>OK</Message>
 <Data>
  <Bookmarks>
   <Count>2</Count>
   <Bookmark>
    <Title>Test</Title>
    <Color>Red</Color>
    <StartDate>2014-06-03 10:10:01.302</StartDate>
    <EndDate>2014-06-03 10:10:01.302</EndDate>
    <Remarks/>
    <User>admin</User>
    <Cameras>Camera1</Cameras>
   </Bookmark>
   <Bookmark>
    <Title>Alarm</Title>
    <Color>Red</Color>
    <StartDate>2014-03-07 15:58:34.450</StartDate>
    <EndDate>2014-03-07 15:58:34.450</EndDate>
    <Remarks/>
    <User>admin</User>
    <Cameras>Camera1,Camera2</Cameras>
   </Bookmark>
  </Bookmarks>
 </Data>
</Response>
```

## 4.3.12 Privacy Mode

Commands to control camera privacy mode

### 4.3.12.1 Controlling privacy mode

Command to activate or deactivate privacy mode of a given camera

**Compatibility:** Professional, Enterprise

**Security level:** Requires authentication of user with rights to control privacy mode

**Method:** HTTP GET

**Syntax:**

```
http://<server_address>/Interface/Cameras/PrivacyMode/SetPrivacyMode?
<argument=value>[&<argument=value>...][&<general_argument>...]
```

**Arguments:**

| Argument | Valid values | Description |
|---|---|---|
| Camera=<String>* | String | Name of the camera |
| Active=<Boolean>* | TRUE<br>FALSE | TRUE - Activate privacy mode<br><br>FALSE - Deactivate privacy mode |

* Mandatory parameters

**Example 1:** Activate privacy mode of "Camera1"

```
http://192.168.0.1:8601/Interface/Cameras/PrivacyMode/SetPrivacyMode?
Camera=Camera1&Active=TRUE
```

**Example 2:** Deactivate privacy mode of "Camera1" with response in text

```
http://192.168.0.1:8601/Interface/Cameras/PrivacyMode/SetPrivacyMode?
Camera=Camera1&Active=FALSE&ResponseFormat=Text
```

**Response:**
Default response of API.

**HTTP Return:** 200 OK

**Parameters of return:** Default return of API

#### 4.3.12.2 Requesting the status of privacy mode

Command to query the current status of privacy mode of a given camera

**Compatibility:** Professional, Enterprise

**Security level:** Requires authentication of the user

**Method:** HTTP GET

**Syntax:**

```
http://<server_address>/Interface/Cameras/PrivacyMode/GetStatus?
<argument=value>[&<argument=value>...][&<general_argument>...]
```

**Arguments:**

| Argument | Valid values | Description |
|---|---|---|
| Camera=<String>* | String | Name of the camera |

* Mandatory parameters

**Example 1:** Query privacy mode status of "Camera1"

```
http://192.168.0.1:8601/Interface/Cameras/PrivacyMode/GetStatus?
Camera=Camera1
```

**Example 2:** Query privacy mode status of "Camera1" with response in text

```
http://192.168.0.1:8601/Interface/Cameras/PrivacyMode/GetStatus?
Camera=Camera1&ResponseFormat=Text
```

**Response:**
A list of parameter-value pairs is returned

**HTTP Return HTTP:** 200 OK

**Parameters of return:**

| Parameter | Type | Description |
|---|---|---|
| ACTIVE | Boolean | Privacy mode activation status |

**Example of return in text:**

```
RESPONSE_CODE=0
RESPONSE_MESSAGE=OK
ACTIVE=FALSE
```

**Example of return in XML:**

```
<?xml version="1.0" encoding="UTF-8" ?>
<Response>
  <Code>0</Code>
  <Message>OK</Message>
  <Data>
    <Status>
      <Active>FALSE</Active>
    </Status>
  </Data>
</Response>
```

## 4.4 I/O Devices

Commands to control I/O devices

### 4.4.1 Requesting the list of I/O devices

Requests the list of I/O devices registered in the server.

**Compatibility:** Standard, Professional, Enterprise

**Security level:** Admin

**Method:** HTTP GET

**Syntax:**

```
http://<server_address>/Interface/IODevices/GetIODevices
[?<argument=value>[&<argument=value>...]][&<general_argument>...]
```

**Arguments:**

| Argument | Valid values | Description |
|---|---|---|
| IODevices=<*APIMasks*> | String | Mask to filter the results. Specify which I/O devices must be returned based on the provided masks. |
| Fields=<*String*> | Name<br>Description<br>Model | Specifies the list of desired fields. In case this parameter is omitted, all of the fields will be sent.<br><br>The fields must be separated by commas<br><br>**Name** / **Description**<br>Name / Name of the I/O device<br>Description / Description of the I/O device<br>Model / Model of the I/O device |

**Example 1:** Requests the list of I/O devices with all of the fields and response in XML

```
http://192.168.0.1:8601/Interface/IODevices/GetIODevices?
ResponseFormat=XML
```

**Example 2:** Requests the list of I/O devices with all of the fields and response in text

```
http://192.168.0.1:8601/Interface/IODevices/GetIODevices?
ResponseFormat=Text
```

**Example 3:** Requests the list of I/O devices with only name and description, response in XML and authentication of the user Admin

```
http://192.168.0.1:8601/Interface/IODevices/GetIODevices?Fields=Name,
Description&ResponseFormat=XML&AuthUser=admin
```

**Example 4:** Request the list of I/O devices starting with "A", with only name and description, response in XML format and authentication with Admin user

```
http://192.168.0.1:8601/Interface/IODevices/GetIODevices?
IODevices=A*&Fields=Name,Description&ResponseFormat=XML&AuthUser=admin
```

**Response:**
A list with all of the I/O devices registered in the system is returned. The fields returned in the list will depend on the values informed in the argument `Fields`

**HTTP Return:** 200 OK

**Parameters of return:**

**Fixed parameters:**

| Parameter | Type | Description |
|---|---|---|
| COUNT | Integer | Total number of I/O devices |

**Parameters of the list of I/O devices:**

| Parameter | Type | Description |
|---|---|---|
| NAME | String | Name of the I/O device |
| DESCRIPTION | String | Description of the I/O device |
| MODEL | String | Model of the de I/O device |

**List of I/O devices:**
The parameters of the list of I/O devices will depend on the type of response (Text or XML).

**List of I/O devices with response in text:**
The parameters of response in text will obey the following syntax:

```
IODEVICE_<num>_<field>=<value>
```

| Parameter | Description |
|---|---|
| *num* | Number of the record |
| *field* | Name of the field |
| *value* | Value of the field |

**Example of return in text:**

```
RESPONSE_CODE=0
RESPONSE_MESSAGE=OK
COUNT=2
IODEVICE_1_NAME=Device 1
IODEVICE_1_DESCRIPTION=I/O Device 1
IODEVICE_1_MODEL=Generic
IODEVICE_2_NAME=Device 2
IODEVICE_2_DESCRIPTION=I/O Device 2
IODEVICE_2_MODEL=Generic
```

**List of I/O devices with response in XML:**
The parameters of response in XML will obey the following syntax:

```
<IODevices>
 <Count>COUNT</Count>
 <IODevice>
  <Name>NAME</Name>
  <Description>DESCRIPTION</Description>
  <Model>MODEL</Model>
 </IODevice>
</IODevices>
```

**Example of return in XML:**

```
<?xml version="1.0" encoding="UTF-8" ?>
<Response>
 <Code>0</Code>
 <Message>OK</Message>
 <Data>
  <IODEvices>
   <Count>2</Count>
   <IODevice>
    <Name>Device 1</Name>
    <Description>I/O device 1</Description>
    <Model>Generic</Model>
   </IODevice>
   <IODevice>
    <Name>Device 2</Name>
    <Description>I/O device 2</Description>
    <Model>Generic</Model>
   </IODevice>
  </IODevices>
 </Data>
</Response>
```

### 4.4.2 Requesting the status of I/O devices

Requests the status of the I/O devices from the system.

**Compatibility:** Standard, Professional, Enterprise

**Security level:** Admin

**Method:** HTTP GET

**Syntax:**

```
http://<server_address>/Interface/IODevices/GetStatus[?<argument=value>
[&<argument=value>...][&<general_argument>...]]
```

**Arguments:**

| Argument | Valid values | Description |
|---|---|---|
| IODevices=<*APIMasks*> | String | Mask to filter the results. Specify which I/O devices must be returned based on the provided masks. |
| Fields=<*String*> | Active<br>Working | Specifies the list of desired fields. In case this parameter is omitted, all of the fields will be sent.<br><br>The fields must be separated by commas<br><br>**Nome** / **Descrição**<br>Active – Identify if the I/O device is active<br>Working – Identify if the I/O device is working |
| Active=<*Boolean*> | TRUE<br>FALSE | In case this parameter is specified, a filter will be applied and only the I/O devices that matches this filter will be returned, thus, in case the value `Active=TRUE` is specified, the result will only include the active I/O devices, while if the value `Active=FALSE` is specified, the result will only include the deactivated I/O devices. |
| Working=<*Boolean*> | TRUE<br>FALSE | In case this parameter is specified, a filter will be applied and only the I/O devices that matches this filter will be returned, thus, in case the value `Working=TRUE` is specified, the result will only include the working I/O devices, while if the value `Working=FALSE` is specified, the result will only include the I/O devices that are out of order. |

**Example 1:** Request the status of all I/O devices with all fields and response in XML

```
http://192.168.0.1:8601/Interface/IODevices/GetStatus?ResponseFormat=XML
```

**Example 2:** Request the status of all active I/O devices with response in text

```
http://192.168.0.1:8601/Interface/IODevices/GetStatus?Active=TRUE&
ResponseFormat=Text
```

**Example 3:** Request the status of all active devices starting with "A", with response in text

```
http://192.168.0.1:8601/Interface/IODevices/GetStatus?IODevices=A*&
Active=TRUE&ResponseFormat=Text
```

**Example 4:** Request the status of all active I/O devices that are not working, with response in XML and authentication with admin user (No password)

```
http://192.168.0.1:8601/Interface/IODevices/GetStatus?Active=TRUE&
Working=FALSE&ResponseFormat=XML&AuthUser=admin
```

**Response:**
A list with the status of all of I/O devices is returned. The fields returned in the will depend on the values informed in the argument `Fields`

**HTTP Return:** 200 OK

**Parameters of return:**

**Fixed Parameter:**

| Parameter | Type | Description |
|---|---|---|
| COUNT | Integer | Total number of I/O devices |

**Parameters in the list of status of I/O devices:**

| Parameter | Type | Description |
|---|---|---|
| NAME | String | Name of the I/O device |
| ACTIVE | Boolean | Identify if the I/O device is active |
| WORKING | Boolean | Identify if the I/O device is working |

**List of status of I/O devices:**
The parameters of the list of status of I/O devices will depend on the type of response (Text or XML).

**List of status of I/O devices with response in text:**
The parameters of response in text will obey the following syntax:

```
IODEVICE_<num>_<field>=<value>
```

| Parameter | Description |
|---|---|
| *num* | Number of the record |
| *field* | Name of the field |
| *value* | Value of the field |

**Example of return in text:**

```
RESPONSE_CODE=0
RESPONSE_MESSAGE=OK
COUNT=2
IODEVICE_1_NAME=Device 1
IODEVICE_1_ACTIVE=TRUE
IODEVICE_1_WORKING=TRUE
IODEVICE_2_NAME=Device 2
IODEVICE_2_ACTIVE=TRUE
IODEVICE_2_WORKING=FALSE
```

**List of status of I/O devices with response in XML:**
The parameters of response in XML will obey the following syntax:

```
<IODevices>
  <Count>COUNT</Count>
  <IODevice>
    <Name>NAME</Name>
    <Active>ACTIVE</Active>
    <Working>WORKING</Working>
  </IODevice>
</IODevices>
```

**Example of return in XML:**

```
<?xml version="1.0" encoding="UTF-8" ?>
<Response>
  <Code>0</Code>
  <Message>OK</Message>
  <Data>
```

```
  <IODevices>
    <Count>2</Count>
    <IODevice>
      <Name>Device 1</Name>
      <Active>TRUE</Active>
      <Working>TRUE</Working>
    </IODevice>
    <IODevice>
      <Name>Device 2</Name>
      <Active>TRUE</Active>
      <Working>FALSE</Working>
    </IODevice>
  </IODevices>
 </Data>
</Response>
```

## 4.4.3 I/O

Commands to control the I/Os of I/O devices

### 4.4.3.1 Requesting the status of input ports

Requests the status of the alarm input ports of an I/O device.

**Compatibility:** Standard, Professional, Enterprise

**Security level:** Admin

**Method:** HTTP GET

**Syntax:**

```
http://<server_address>/Interface/IODevices/IO/GetInputPortStatus?
<argument=value>[&<argument=value>...][&<general_argument>...]
```

**Arguments:**

| Argument | Valid values | Description |
|---|---|---|
| Device=<String>* | String | Name of the I/O device |

* Mandatory parameters

**Example 1:** Requests the status of the alarm input ports of an I/O device with response in text

```
http://192.168.0.1:8601/Interface/IODevices/IO/GetInputPortStatus?
Device=Device1&ResponseFormat=Text
```

**Example 2:** Requests the status of the alarm input ports of an I/O device with response in XML

```
http://192.168.0.1:8601/Interface/IODevices/IO/GetInputPortStatus?
Device=Device1&ResponseFormat=XML
```

**Response:**
A list with the status of all of the alarm input ports of the specified I/O device is returned.

**HTTP Return:** 200 OK

**Parameters of return:**

**Fixed parameters:**

| Parameter | Type | Description |
|---|---|---|
| COUNT | Integer | Total number of alarm input ports |

**Parameters of the list of ports:**

| Parameter | Type | Description |
|---|---|---|
| STATE | String | State of the input port<br><br>Type: SHORT — Port closed<br>Type: OPEN — Port open<br>Type: UNKNOWN — State unknown |

**List of alarm input ports:**
The parameters of the list of alarm input ports will depend on the type of response (Text or XML).

**List of alarm input ports with response in text:**
The parameters of response in text will obey the following syntax:

```
PORT_<num>_<field>=<value>
```

| Parameter | Description |
|---|---|
| *num* | Number of the record |
| *field* | Name of the field |
| *value* | Value of the field |

**Example of return in text:**

```
RESPONSE_CODE=0
RESPONSE_MESSAGE=OK
COUNT=4
PORT_1_STATE=OPEN
PORT_2_STATE=OPEN
PORT_3_STATE=SHORT
PORT_4_STATE=OPEN
```

**List of alarm input ports with response in XML:**
The parameters of response in XML will obey the following syntax:

```
<Ports>
  <Count>COUNT</Count>
  <Port>
    <State>STATE</State>
  </Port>
</Ports>
```

**Example of return in XML:**

```
<?xml version="1.0" encoding="UTF-8" ?>
<Response>
 <Code>0</Code>
 <Message>OK</Message>
 <Data>
  <Ports>
   <Count>4</Count>
   <Port>
    <State>OPEN</State>
   </Port>
   <Port>
    <State>OPEN</State>
   </Port>
   <Port>
    <State>SHORT</State>
   </Port>
   <Port>
    <State>OPEN</State>
   </Port>
  </Ports>
 </Data>
</Response>
```

#### 4.4.3.2 Requesting the status of input events

Requests the status of the events of alarm input of an I/O device.

**Compatibility:** Standard, Professional, Enterprise

**Security level:** Admin

**Method:** HTTP GET

**Syntax:**

```
http://<server_address>/Interface/IODevices/IO/GetInputEventStatus?
<argument=value>[&<argument=value>...][&<general_argument>...]
```

**Arguments:**

| Argument | Valid values | Description |
|---|---|---|
| Device=<String>* | String | Name of the I/O device |

* Mandatory parameters

**Example 1:** Requests the status of the alarm input events of an I/O device with response in text

```
http://192.168.0.1:8601/Interface/IODevices/IO/GetInputEventStatus?
Device=Device1&ResponseFormat=Text
```

**Example 2:** Requests the status of the alarm input events of an I/O device with response in XML

```
http://192.168.0.1:8601/Interface/IODevices/IO/GetInputEventStatus?
Device=Device1&ResponseFormat=XML
```

**Response:**
A list of the status of all of the alarm input events of the specified I/O device is returned.

**HTTP Return:** 200 OK

**Parameters of return:**

**Fixed parameters:**

| Parameter | Type | Description |
|---|---|---|
| COUNT | Integer | Total number of alarm input events |

**Parameters of the list of events:**

| Parameter | Type | Description |
|---|---|---|
| NAME | String | Name of the event |
| STATE | String | State of the input event |

| Type | Description |
|---|---|
| ACTIVE | Event active (Occurring at this moment) |
| INACTIVE | Event inactive (Not occurring) |

**List of alarm input events:**
The parameters of the list of alarm input events will depend on the type of response (Text or XML).

**List of the alarm input events with response in text:**
The parameters of response in text will obey the following syntax:

```
EVENT_<num>_<field>=<value>
```

| Parameter | Description |
|---|---|
| *num* | Number of the record |
| *field* | Name of the field |
| *value* | Value of the field |

**Example of return in text:**

```
RESPONSE_CODE=0
RESPONSE_MESSAGE=OK
COUNT=2
EVENT_1_NAME=Event 1
EVENT_1_STATE=ACTIVE
EVENT_2_NAME=Event 2
EVENT_2_STATE=INACTIVE
```

**List of alarm input events with response in XML:**
The parameters of response in XML will obey the the following syntax:

```
<Events>
  <Count>COUNT</Count>
  <Event>
    <Name>NAME</Name>
    <State>STATE</State>
  </Event>
</Events>
```

**Example of return in XML:**

```
<?xml version="1.0" encoding="UTF-8" ?>
<Response>
 <Code>0</Code>
 <Message>OK</Message>
 <Data>
  <Events>
   <Count>2</Count>
   <Event>
    <Name>Event 1</Name>
    <State>ACTIVE</State>
   </Event>
   <Event>
    <Name>Event 2</Name>
    <State>INACTIVE</State>
   </Event>
  </Events>
 </Data>
</Response>
```

#### 4.4.3.3 Requesting the status of the output ports

Requests the status of the alarm output ports of an I/O device.

**Compatibility:** Standard, Professional, Enterprise

**Security level:** Admin

**Method:** HTTP GET

**Syntax:**

```
http://<server_address>/Interface/IODevices/IO/GetOutputPortStatus?
<argument=value>[&<argument=value>...][&<general_argument>...]
```

**Arguments:**

| Argument | Valid values | Description |
|---|---|---|
| Device=<String>* | String | Name of the I/O device |

* Mandatory parameters

**Example 1:** Requests the status of the alarm output ports of an I/O device with response in text

```
http://192.168.0.1:8601/Interface/IODevices/IO/GetOutputPortStatus?
Device=Device1&ResponseFormat=Text
```

**Example 2:** Requests the status of the alarm output ports of an I/O device with response in XML

```
http://192.168.0.1:8601/Interface/IODevices/IO/GetOutputPortStatus?
Device=Device1&ResponseFormat=XML
```

**Response:**
A list with the status of all of the alarm output ports of a specified I/O device is returned.

**HTTP Return:** 200 OK

**Parameters of return:**

**Fixed parameters:**

| Parameter | Type | Description |
|---|---|---|
| COUNT | Integer | Total number of alarm output ports |

**Parameters of the list of ports:**

| Parameter | Type | Description |
|---|---|---|
| STATE | String | State of the output port<br><br>Type: SHORT — Port closed<br>Type: OPEN — Port open<br>Type: UNKNOWN — State unknown |

**List of alarm output ports:**
The parameters of the alarm output reports will depend on the type of response (Text or XML).

**List of alarm output ports with response in text:**
The parameters of response in text will obey the following syntax:

```
PORT_<num>_<field>=<value>
```

| Parameter | Description |
|---|---|
| *num* | Number of the record |
| *field* | Name of the field |
| *value* | Value of the field |

**Example of return in text:**

```
RESPONSE_CODE=0
RESPONSE_MESSAGE=OK
COUNT=4
PORT_1_STATE=OPEN
PORT_2_STATE=OPEN
PORT_3_STATE=SHORT
PORT_4_STATE=OPEN
```

**List of alarm output ports with response in XML:**
The parameters of response in XML will obey the following syntax:

```
<Ports>
  <Count>COUNT</Count>
  <Port>
    <State>STATE</State>
  </Port>
</Ports>
```

**Example of return in XML:**

```
<?xml version="1.0" encoding="UTF-8" ?>
<Response>
 <Code>0</Code>
 <Message>OK</Message>
 <Data>
  <Ports>
   <Count>4</Count>
   <Port>
    <State>OPEN</State>
   </Port>
   <Port>
    <State>OPEN</State>
   </Port>
   <Port>
    <State>SHORT</State>
   </Port>
   <Port>
    <State>OPEN</State>
   </Port>
  </Ports>
 </Data>
</Response>
```

#### 4.4.3.4 Requesting the list of output actions

Requests the list of output actions of an I/O device.

**Compatibility:** Standard, Professional, Enterprise

**Security level:** Admin

**Method:** HTTP GET

**Syntax:**

```
http://<server_address>/Interface/IODevices/IO/GetOutputActions?
<argument=value>[&<argument=value>...][&<general_argument>...]
```

**Arguments:**

| Argument | Valid values | Description |
|---|---|---|
| Device=<String>* | String | Name of the I/O device |
| Actions=<APIMasks> | String | Mask to filter the results. Specify which actions must be returned based on the provided masks. |

* Mandatory parameters

**Example 1:** Request the list of output actions of an I/O device with response in text

```
http://192.168.0.1:8601/Interface/IODevices/IO/GetOutputActions?
Device=Device1&ResponseFormat=Text
```

**Example 2:** Request the list of output actions of an I/O device with response in XML

```
http://192.168.0.1:8601/Interface/IODevices/IO/GetOutputActions?
Device=Device1&ResponseFormat=XML
```

**Example 3:** Request the list of output actions starting with "A" of an I/O device, with response in XML

```
http://192.168.0.1:8601/Interface/IODevices/IO/GetOutputActions?
Device=Device1&Actions=A*&ResponseFormat=XML
```

**Response:**
A list with all of the output actions of the specified I/O device is returned.

**HTTP Return:** 200 OK

**Parameters of return:**

**Fixed parameters:**

| Parameter | Type | Description |
|---|---|---|
| COUNT | Integer | Total number of output actions |

**Parameters of the list of actions:**

| Parameter | Type | Description |
|---|---|---|
| NAME | String | Name of the action |

**List of the output actions:**
The parameters of the list of output actions will depend on the type of response (Text or XML).

**List of output actions with response in text:**
The parameters of response in text will obey the following syntax:

```
ACTION_<num>_<field>=<value>
```

| Parameter | Description |
|---|---|
| *num* | Number of the record |
| *field* | Name of the field |
| *value* | Value of the field |

**Example of returno in text:**

```
RESPONSE_CODE=0
RESPONSE_MESSAGE=OK
COUNT=2
ACTION_1_NAME=Action 1
ACTION_2_NAME=Action 2
```

**List of output actions with response in XML:**
Os parâmetros de resposta em XML irão seguir a seguinte sintaxe:

```
<Actions>
  <Count>COUNT</Count>
  <Action>
    <Name>NAME</Name>
  </Action>
</Actions>
```

**Example of return in XML:**

```
<?xml version="1.0" encoding="UTF-8" ?>
<Response>
 <Code>0</Code>
 <Message>OK</Message>
 <Data>
  <Actions>
   <Count>2</Count>
   <Action>
    <Name>Action 1</Name>
   </Action>
   <Action>
    <Name>Action 2</Name>
   </Action>
  </Actions>
 </Data>
</Response>
```

#### 4.4.3.5 Triggering an output action

By way of this command you will be able to trigger a script output actions.

**Compatibility:** Standard, Professional, Enterprise

**Security level:** Admin

**Method:** HTTP GET

**Syntax:**

```
http://<server_address>/Interface/IODevices/IO/TriggerOutputAction?
<argument=value>[&<argument=value>...][&<general_argument>...]
```

**Arguments:**

| Argument | Valid values | Description |
|---|---|---|
| Device=<String>* | String | Name of the I/O device |
| Action=<String>* | String | Name of the action |

* Mandatory parameters

**Example 1:** Activate an output action of an I/O device with response in text

```
http://192.168.0.1:8601/Interface/IODevices/IO/TriggerOutputAction?
Device=Device1&Action=Action1&ResponseFormat=Text
```

**Example 2:** Activate an output action of an I/O device with response in XML

```
http://192.168.0.1:8601/Interface/IODevices/IO/TriggerOutputAction?
Device=Device1&Action=Action1&ResponseFormat=XML
```

**Response:**
Default response of API.

**HTTP Return:** 200 OK

**Parameters of return:** Default return of API

# 4.5 Users

Commands to control server users

## 4.5.1 Requesting the list of users

Requests the list of user registered in the server.

**Compatibility:** All editions

**Security level:** Admin

**Method:** HTTP GET

**Syntax:**

```
http://<server_address>/Interface/Users/GetUsers[?<argument=value>
[&<argument=value>...]][&<general_argument>...]
```

**Arguments:**

| Argument | Valid velues | Description |
|---|---|---|
| `Users=<APIMasks>` | String | Mask to filter the results. Specify which users must be returned based on the provided masks. |
| `Fields=<String>` | Name<br>Description | Specifies the list if desired fields. In case this parameter is omitted, all of the fields will be sent.<br><br>The fields must be separated by commas<br><br>Name: Name of the user<br>Description: Description of the user |

**Example 1:** Request the list of users with all of the fields and response in XML

```
http://192.168.0.1:8601/Interface/Users/GetUsers?ResponseFormat=XML
```

**Example 2:** Request the list of users with all of the fields and response in text

```
http://192.168.0.1:8601/Interface/Users/GetUsers?ResponseFormat=Text
```

**Example 3:** Request the list of users with only name, response in XML and authentication of the user Admin

```
http://192.168.0.1:8601/Interface/Users/GetUsers?Fields=Name&
ResponseFormat=XML&AuthUser=admin&AuthPass=pass
```

**Example 4:** Request the list of users starting with "A", with only name, response in XML and authentication with Admin user

```
http://192.168.0.1:8601/Interface/Users/GetUsers?Users=A*&Fields=Name&
ResponseFormat=XML&AuthUser=admin&AuthPass=pass
```

**Response:**
A list with all of the users registered in the system is returned. The fields returned in the will depend on the values informed in the argument `Fields`

**HTTP Return:** 200 OK

**Parameters of return:**

**Fixed parameters:**

| Parameter | Type | Description |
|---|---|---|
| COUNT | Integer | Total number of users |

**Parameters in the list of users:**

| Parameter | Type | Description |
|---|---|---|
| NAME | String | Name of the user |
| DESCRIPTION | String | Description of the user |

**List of users:**
The parameters of the list of user will depend on the type of response (Text or XML).

**List of users with response in text:**
The parameters of the response in text will obey the following syntax:

```
USER_<num>_<field>=<value>
```

| Parameter | Description |
|---|---|
| *num* | Number of the record |
| *field* | Name of the field |
| *value* | Value of the field |

**Example of return in text:**

```
RESPONSE_CODE=0
RESPONSE_MESSAGE=OK
COUNT=2
USER_1_NAME=admin
USER_1_DESCRIPTION=System administration account
USER_2_NAME=Guest
USER_2_DESCRIPTION=Guest user
```

**List of the users with response in XML:**
The parameters of the response in XML will obey the following syntax:

```
<Users>
  <Count>COUNT</Count>
  <User>
    <Name>NAME</Name>
    <Description>DESCRIPTION</Description>
  </User>
</Users>
```

**Example of return in XML:**

```
<?xml version="1.0" encoding="UTF-8" ?>
<Response>
 <Code>0</Code>
 <Message>OK</Message>
 <Data>
  <Users>
   <Count>2</Count>
   <User>
    <Name>admin</Name>
    <Description>System administration account</Description>
   </User>
   <User>
    <Name>Guest</Name>
    <Description>Guest user</Description>
   </User>
  </Users>
 </Data>
</Response>
```

## 4.6 Screenstyles

Commands to control screenstyles

### 4.6.1 Requesting the list of screenstyles

Requests the list of screenstyles registered in the server.

**Compatibility:** Standard, Professional, Enterprise

**Security level:** Requires authentication of the user

**Method:** HTTP GET

**Syntax:**

```
http://<server_address>/Interface/ScreenStyles/GetScreenStyles
[?<general_argument>[&<general_argument>...]]
```

**Example 1:** Requests the list of screenstyles with response in XML

```
http://192.168.0.1:8601/Interface/ScreenStyles/GetScreenStyles?
ResponseFormat=XML
```

**Example 2:** Requests the list of screenstyles with response in text

```
http://192.168.0.1:8601/Interface/ScreenStyles/GetScreenStyles?
ResponseFormat=Text
```

**Response:**
A list with all of the screenstyles registered in the system is returned.

**HTTP Return:** 200 OK

**Parameters of return:**

**Fixed parameters:**

| Parameter | Type | Description |
|---|---|---|
| COUNT | Integer | Total number of screenstyles |

**Parameters of the list of screenstyles:**

| Parameter | Type | Description |
|---|---|---|
| ID | Integer | Identification of the screenstyle |

**List of screenstyles:**
The parameters of the list of screenstyles will depend on the type of response (Text or XML).

**List of screenstyles with response in text:**
The parameters of response in text will obey the following syntax:

```
SCREENSTYLE_<num>_<field>=<value>
```

| Parameter | Description |
|---|---|
| *num* | Number of the record |
| *field* | Name of the field |
| *value* | Value of the field |

**Example of return in text:**

```
RESPONSE_CODE=0
RESPONSE_MESSAGE=OK
COUNT=8
SCREENSTYLE_1_ID=1399
SCREENSTYLE_2_ID=6278
SCREENSTYLE_3_ID=9983
SCREENSTYLE_4_ID=13538
SCREENSTYLE_5_ID=16712
SCREENSTYLE_6_ID=18393
SCREENSTYLE_7_ID=25660
SCREENSTYLE_8_ID=31698
```

**List of screenstyles with response in XML:**
The parameters of response in XML will obey the following syntax:

```
<ScreenStyles>
  <Count>COUNT</Count>
  <ScreenStyle>
    <ID>ID</ID>
  </ScreenStyle>
</ScreenStyles>
```

**Example of return in XML:**

```
<?xml version="1.0" encoding="UTF-8" ?>
<Response>
 <Code>0</Code>
 <Message>OK</Message>
 <Data>
  <ScreenStyles>
   <Count>8</Count>
   <ScreenStyle>
    <ID>1399</ID>
   </ScreenStyle>
   <ScreenStyle>
    <ID>6278</ID>
   </ScreenStyle>
   <ScreenStyle>
    <ID>9983</ID>
   </ScreenStyle>
   <ScreenStyle>
    <ID>13538</ID>
   </ScreenStyle>
   <ScreenStyle>
    <ID>16712</ID>
   </ScreenStyle>
   <ScreenStyle>
    <ID>18393</ID>
   </ScreenStyle>
   <ScreenStyle>
    <ID>25660</ID>
   </ScreenStyle>
   <ScreenStyle>
    <ID>31698</ID>
   </ScreenStyle>
  </ScreenStyles>
 </Data>
</Response>
```

### 4.6.2 Requesting the image of a screenstyle

Requesting the illustrative image of a screenstyle in JPEG.

**Compatibility:** Standard, Professional, Enterprise

**Security level:** Requires authentication of the user

**Method:** HTTP GET

**Syntax:**

```
http://<server_address>/Interface/ScreenStyles/GetScreenStyleImage?
<argument=value>[&<argument=value>...][&<general_argument>...]
```

**Arguments:**

| Argument | Valid values | Description |
|---|---|---|
| ID=<*Integer*>* | Integer | ID of the screenstyle |
| Selected=<*Integer*> | 0, 1 | Indicates whether the image will be displayed with a |

| | | |
|---|---|---|
| | | normal border (in black) or a selected border (in red).<br><br>if this parameter is omitted, the default value 0 will be used.<br><br>**Value** / **Description**<br>0 – Normal border (black)<br>1 – Selected border (red) |

* Mandatory parameters

**Example 1:** Requests the image of screenstyle 1399 with error response in XML

```
http://192.168.0.1:8601/Interface/ScreenStyles/GetScreenStyleImage?
ID=1399&ResponseFormat=XML
```

**Example 2:** Requests the image of screenstyle 6278 with error response in text

```
http://192.168.0.1:8601/Interface/ScreenStyles/GetScreenStyleImage?
ID=6278&ResponseFormat=Text
```

**Example 3:** Requests the image of screenstyle 6278 with a selected border and error response in text

```
http://192.168.0.1:8601/Interface/ScreenStyles/GetScreenStyleImage?
ID=6278&Selected=1&ResponseFormat=Text
```

**Response:**
In case of success, a JPEG image will be returned. In case of error, a code and an error message will be returned. An error can be returned if the screenstyle is not found.

**HTTP Return:** 200 OK

**Parameters of return:** Default return of API

## 4.7 Screen views

Commands to control screen views

### 4.7.1 Requesting the list of screen views of the user

Request the list of screen views of the user used to request the command

**Compatibility:** All editions

**Security level:** Requires authentication of the user

**Method:** HTTP GET

**Syntax:**

```
http://<server_address>/Interface/ScreenViews/GetUserScreenViews
[?<argument=value>[&<argument=value>...]][&<general_argument>...]
```

**Arguments:**

| Argument | Valid values | Description |
|---|---|---|
| ScreenStyleID=<Integer> | Integer | Filters the results to display only the screen views of the specified screenstyle. |

| | | |
|---|---|---|
| | | If this parameter is omitted, all of the screen views of the user will be listed. |
| ScreenViews=<*APIMasks*> | String | Mask to filter the results. Specify which screen views must be returned based on the provided masks. |

**Example 1:** Request the list of screen views of the user with response in XML

```
http://192.168.0.1:8601/Interface/ScreenViews/GetUserScreenViews?
ResponseFormat=XML
```

**Example 2:** Request the list of screen view of the user guest1 for the screenstyle 9983 and response in text

```
http://192.168.0.1:8601/Interface/ScreenViews/GetUserScreenViews?
ScreenStyleID=9983&AuthUser=guest1&AuthPass=guestpass&ResponseFormat=Text
```

**Example 3:** Request the list of screen views staring with "A", from user "guest1" for screen style 9983 with response in text

```
http://192.168.0.1:8601/Interface/ScreenViews/GetUserScreenViews?
ScreenStyleID=9983&ScreenViews=A*&AuthUser=guest1&AuthPass=guestpass&
ResponseFormat=Text
```

**Response:**
A list with all of the screen views of the user is returned

**HTTP Return:** 200 OK

**Parameters of return:**

**Fixed parameters:**

| Paremeter | Type | Description |
|---|---|---|
| COUNT | Integer | Total number of screen views |

**Parameters of the list of screen views:**

| Parameter | Type | Description |
|---|---|---|
| NAME | String | Name of the screen view |
| SCREENSTYLEID | Integer | ID of the screenstyle of the screen view |

**List of screen views:**
The parameters of the list of screen views will depend on the type of response (Text or XML)

**List of screen views with response in text:**
The parameters of response in text will obey the following syntax:

```
SCREENVIEW_<num>_<field>=<value>
```

| Parameter | Description |
|---|---|
| *num* | Number of the record |
| *field* | Name of the field |
| *value* | Value of the field |

**Example of return in text:**

```
RESPONSE_CODE=0
RESPONSE_MESSAGE=OK
COUNT=3
SCREENVIEW_1_NAME=View 1
SCREENVIEW_1_SCREENSTYLEID=6278
SCREENVIEW_2_NAME=View 2
SCREENVIEW_2_SCREENSTYLEID=9983
SCREENVIEW_3_NAME=View 3
SCREENVIEW_3_SCREENSTYLEID=9983
```

**List of screen views with response in XML:**
The parameters of response in XML will obey the following syntax

```
<ScreenViews>
 <Count>COUNT</Count>
 <ScreenView>
  <Name>NAME</Name>
  <ScreenStyleID>SCREENSTYLEID</ScreenStyleID>
 </ScreenView>
</ScreenViews>
```

**Example of return in XML:**

```
<?xml version="1.0" encoding="UTF-8" ?>
<Response>
 <Code>0</Code>
 <Message>OK</Message>
 <Data>
  <ScreenViews>
   <Count>3</Count>
   <ScreenView>
    <Name>View 1</Name>
    <ScreenStyleID>6278</ScreenStyleID>
   </ScreenView>
   <ScreenView>
    <Name>View 2</Name>
    <ScreenStyleID>9983</ScreenStyleID>
   </ScreenView>
   <ScreenView>
    <Name>View 3</Name>
    <ScreenStyleID>9983</ScreenStyleID>
   </ScreenView>
  </ScreenViews>
 </Data>
</Response>
```

### 4.7.2 Requesting the list of public screen views

Requests the list of public screen views of the system.

**Compatibility:** All editions

**Security level:** Requires authentication of the user

**Method:** HTTP GET

**Syntax:**

```
http://<server_address>/Interface/ScreenViews/GetPublicScreenViews
[?<argument=value>[&<argument=value>...]][&<general_argument>...]
```

**Arguments:**

| Argument | Valid values | Description |
|---|---|---|
| ScreenStyleID=<*Integer*> | Integer | Filters the results to display only the screen views of the specified screenstyle.<br><br>If this parameter is omitted, all of the public screen views will be listed. |
| ScreenViews=<*APIMasks*> | String | Mask to filter the results. Specify which screen views must be returned based on the provided masks. |

**Example 1:** Request the list of public screen views with response in XML

```
http://192.168.0.1:8601/Interface/ScreenViews/GetPublicScreenViews?
ResponseFormat=XML
```

**Example 2:** Request the list of public screen view for the screenstyle 9983 and response in text

```
http://192.168.0.1:8601/Interface/ScreenViews/GetPublicScreenViews?
ScreenStyleID=9983&ResponseFormat=Text
```

**Example 3:** Reqiest the list of public screen views for screenstyle 9983 that starts with letter "A", with response in text

```
http://192.168.0.1:8601/Interface/ScreenViews/GetPublicScreenViews?
ScreenStyleID=9983&ScreenViews=A*&ResponseFormat=Text
```

**Response:**
A list with all of the public screen views is returned

**HTTP Return:** 200 OK

**Parameters of return:**

**Fixed parameters:**

| Paremeter | Type | Description |
|---|---|---|
| COUNT | Integer | Total number of screen views |

**Parameters of the list of screen views:**

| Parameter | Type | Description |
|---|---|---|
| NAME | String | Name of the screen view |
| SCREENSTYLEID | Integer | ID of the screenstyle of the screen view |

**List of screen views:**
The parameters of the list of screen views will depend on the type of response (Text or XML)

**List of screen views with response in text:**
The parameters of response in text will obey the following syntax:

```
SCREENVIEW_<num>_<field>=<value>
```

| Parameter | Description |
|---|---|
| *num* | Number of the record |
| *field* | Name of the field |
| *value* | Value of the field |

**Example of return in text:**

```
RESPONSE_CODE=0
RESPONSE_MESSAGE=OK
COUNT=3
SCREENVIEW_1_NAME=View 1
SCREENVIEW_1_SCREENSTYLEID=6278
SCREENVIEW_2_NAME=View 2
SCREENVIEW_2_SCREENSTYLEID=9983
SCREENVIEW_3_NAME=View 3
SCREENVIEW_3_SCREENSTYLEID=9983
```

**List of screen views with response in XML:**

The parameters of response in XML will obey the following syntax

```
<ScreenViews>
 <Count>COUNT</Count>
 <ScreenView>
  <Name>NAME</Name>
  <ScreenStyleID>SCREENSTYLEID</ScreenStyleID>
 </ScreenView>
</ScreenViews>
```

**Example of return in XML:**

```
<?xml version="1.0" encoding="UTF-8" ?>
<Response>
 <Code>0</Code>
 <Message>OK</Message>
 <Data>
  <ScreenViews>
   <Count>3</Count>
   <ScreenView>
    <Name>View 1</Name>
    <ScreenStyleID>6278</ScreenStyleID>
   </ScreenView>
   <ScreenView>
    <Name>View 2</Name>
    <ScreenStyleID>9983</ScreenStyleID>
   </ScreenView>
   <ScreenView>
    <Name>View 3</Name>
    <ScreenStyleID>9983</ScreenStyleID>
   </ScreenView>
  </ScreenViews>
 </Data>
</Response>
```

# 4.8 Maps

Commands to control maps

## 4.8.1 Requesting the list of maps

Requests the list of maps that the user has rights to access.

**Compatibility:** Standard, Professional, Enterprise

**Security level:** Requires authentication of the user

**Method:** HTTP GET

**Syntax:**

```
http://<server_address>/Interface/Maps/GetMaps[?<argument=value>
[&<argument=value>...]][&<general_argument>...]
```

**Arguments:**

| Argument | Valid values | Description |
|---|---|---|
| `Maps=<APIMasks>` | String | Mask to filter the results. Specify which maps must be returned based on the provided masks. |
| `Fields=<String>` | Name<br>Description | Specifies the list of desired fields. If this parameter is omitted, all of the fields will be sent.<br><br>The fields must be separated by commas |

| Name | Description |
|---|---|
| Name | Name of the map |
| Description | Description of the map |

**Example 1:** Request the list of maps with all of the fields and response in XML

```
http://192.168.0.1:8601/Interface/Maps/GetMaps?ResponseFormat=XML
```

**Example 2:** Request the list of maps with all of the fields and response in text

```
http://192.168.0.1:8601/Interface/Maps/GetMaps?ResponseFormat=Text
```

**Example 3:** Request the list of maps with only name, response in XML and authentication of the user Admin

```
http://192.168.0.1:8601/Interface/Maps/GetMaps?Fields=Name&
ResponseFormat=XML&AuthUser=admin&AuthPass=pass
```

**Example 4:** Request the list of maps starting with "A", with only name, response in XML and authentication with Admin user

```
http://192.168.0.1:8601/Interface/Maps/GetMaps?Maps=A*&Fields=Name&
ResponseFormat=XML&AuthUser=admin&AuthPass=pass
```

**Response:**
A list with all of the maps that the user has rights to access is returned. The fields returned in the list will depend on the values informed in the argument `Fields`

**HTTP Return:** 200 OK

**Parameters of return:**

**Fixed parameters:**

| Parameter | Type | Description |
|---|---|---|
| COUNT | Integer | Total number of maps |

**Parameters of the list of maps:**

| Parameter | Type | Description |
|---|---|---|
| NAME | String | Name of the map |
| DESCRIPTION | String | Description of the map |

**List of maps:**
The parameters of the list of maps will depend on the type of response (Text or XML).

**List of maps with response in text:**
The parameters of response in text will obey the following syntax:

```
MAP_<num>_<field>=<value>
```

| Parameter | Description |
|---|---|
| *num* | Number of the record |
| *field* | Name of the field |
| *value* | Value of the field |

**Example of return in text:**

```
RESPONSE_CODE=0
RESPONSE_MESSAGE=OK
COUNT=2
MAP_1_NAME=Map1
MAP_1_DESCRIPTION=Map 1
MAP_2_NAME=Map2
MAP_2_DESCRIPTION=Map 2
```

**List of maps with response in XML:**
The parameters of response in XML will obey the following syntax:

```
<Maps>
  <Count>COUNT</Count>
  <Map>
    <Name>NAME</Name>
    <Description>DESCRIPTION</Description>
  </Map>
</Maps>
```

**Example of return in XML:**

```
<?xml version="1.0" encoding="UTF-8" ?>
<Response>
 <Code>0</Code>
 <Message>OK</Message>
 <Data>
  <Maps>
   <Count>2</Count>
   <Map>
    <Name>Map1</Name>
    <Description>Map 1</Description>
   </Map>
   <Map>
    <Name>Map2</Name>
    <Description>Map 2</Description>
   </Map>
  </Maps>
 </Data>
</Response>
```

# 4.9 Global events

Commands to control global events

## 4.9.1 Requesting the list of global events

Requests the list of global events that the user has rights to access.

**Compatibility:** Standard, Professional, Enterprise

**Security level:** Requires authentication of the user

**Method:** HTTP GET

**Syntax:**

```
http://<server_address>/Interface/GlobalEvents/GetGlobalEvents
[?<argument=value>[&<argument=value>...]][&<general_argument>...]
```

**Arguments:**

| Argument | Valid values | Description |
|---|---|---|
| GlobalEvents=<*APIMasks*> | String | Mask to filter the results. Specify which global events must be returned based on the provided masks. |
| Fields=<*String*> | Name<br>Description | Specifies the list of desired fields. If this parameter is omitted, all of the fields will be sent.<br><br>The fields must be separated by commas<br><br>**Name** — Name of the global event<br>**Description** — Description of the global event |

**Example 1:** Requests the list of global events with all of the fields and response in XML

```
http://192.168.0.1:8601/Interface/GlobalEvents/GetGlobalEvents?
ResponseFormat=XML
```

**Example 2:** Requests the list of global events with all of the fields and response in text

```
http://192.168.0.1:8601/Interface/GlobalEvents/GetGlobalEvents?
ResponseFormat=Text
```

**Example 3:** Requests the list of global events with only name, response in XML and authentication of the user Admin

```
http://192.168.0.1:8601/Interface/GlobalEvents/GetGlobalEvents?Fields=Name&
ResponseFormat=XML&AuthUser=admin&AuthPass=pass
```

**Example 4:** Request the list of global events starting with "A", with only name, response in XML and authentication with Admin user

```
http://192.168.0.1:8601/Interface/GlobalEvents/GetGlobalEvents?
GlobalEvents=A*&Fields=Name&ResponseFormat=XML&AuthUser=admin&
AuthPass=pass
```

**Response:**
A list with all of the global events that the user has rights of accessing is returned. The fields returned in the list will depend on the values informed in the argument `Fields`

**HTTP Return:** 200 OK

**Parameters of return:**

**Fixed parameters:**

| Parameter | Type | Description |
|---|---|---|
| COUNT | Integer | Total number of global events |

**Parameters of the list of global events:**

| Parameter | Type | Description |
|---|---|---|
| NAME | String | Name of the global event |
| DESCRIPTION | String | Description of the global event |

**List of global events:**
The parameters of the list of global events will depend on the type of response (Text or XML).

**List of global events with response in text:**
The parameters of response in text will obey the following syntax:

```
GLOBALEVENT_<num>_<field>=<value>
```

| Parameter | Description |
|---|---|
| *num* | Number of the record |
| *field* | Name of the field |
| *value* | Value of the field |

**Example of return in text:**

```
RESPONSE_CODE=0
RESPONSE_MESSAGE=OK
COUNT=2
GLOBALEVENT_1_NAME=Event1
GLOBALEVENT_1_DESCRIPTION=Event 1
GLOBALEVENT_2_NAME=Event2
GLOBALEVENT_2_DESCRIPTION=Event 2
```

**List of global events with response in XML:**
The parameters of response in XML will obey the following syntax:

```
<GlobalEvents>
 <Count>COUNT</Count>
 <GlobalEvent>
  <Name>NAME</Name>
  <Description>DESCRIPTION</Description>
 </GlobalEvent>
</GlobalEvents>
```

**Example of return in XML:**

```
<?xml version="1.0" encoding="UTF-8" ?>
<Response>
 <Code>0</Code>
 <Message>OK</Message>
 <Data>
  <GlobalEvents>
   <Count>2</Count>
   <GlobalEvent>
    <Name>Event1</Name>
    <Description>Event 1</Description>
   </GlobalEvent>
   <GlobalEvent>
    <Name>Event2</Name>
    <Description>Event 2</Description>
   </GlobalEvent>
  </GlobalEvents>
 </Data>
</Response>
```

### 4.9.2 Triggering a global event

By way of this command, you will be able to trigger a global event.

**Compatibility:** Standard, Professional, Enterprise

**Security level:** Requires authentication of the user with rights to trigger global events

**Method:** HTTP GET

**Syntax:**

```
http://<server_address>/Interface/GlobalEvents/TriggerGlobalEvent?
<argument=value>[&<argument=value>...][&<general_argument>...]
```

**Arguments:**

| Argument | Valid values | Description |
|---|---|---|
| `Event=<String>*` | String | Name of the global event |
| `Message=<String>` | String | Message of the global event (to be registered in the server's log) |
| `OverrideEmailMessage=<String>` | String | Overrides an e-mail message, in case the action of e-mail transmission for the event is selected |
| `OverrideOperatorMessage=<String>` | String | Overrides an instant message sent to the operators, in case the action of message transmission to the operators is selected |
| `OverrideShowObjects=<String>` | String | Overrides the list of objects to be sent to the operator inside an alarm popup. See below how the string must be generated. |
| `OverrideShowCameras=<String>` | String | Overrides the list of cameras to be sent to the operator inside an alarm popup. This parameter should contain the name of the cameras on a comma separated way.<br><br>This parameter will only be effective if the parameter `OverrideShowObjects` is not specified.<br><br>OBS: This parameter is only being mantained for compatibility. Use the new parameter `OverrideShowObjects` for new developments |

* Mandatory parameters

**OverrideShowObjects**

This parameter was introduced as a replacement to the old `OverrideShowCameras` parameter, being now more comprehensive, providing the possibility to choose the objects to be sent in the popup including maps, cameras, analytics configurations and LPR configurations.
The list of objects to must be formated in the following way:

```
OBJECT_ID_1;OBJECT_NAME_1,OBJECT_ID_2;OBJECT_NAME_2
```

Being, OBJECT_ID the ID of the object and OBJECT_NAME the name of the objects. The below table displays the possible values for OBJECT_ID:

| OBJECT_ID | Description |
|---|---|
| 1 | Camera |
| 9 | Map |
| 12 | Analytics configuration |
| 13 | LPR configuration |

For example, to show cameras Camera1, Camera2 and the maps Map1 e Map2, the list must be defined as:

```
1;Camera1,1;Camera2,9;Map1,9;Map2
```

**Example 1:** Trigger a global event with the message "Test message" and response in text
```
http://192.168.0.1:8601/Interface/GlobalEvents/TriggerGlobalEvent?
Event=Event1&Message=Test message&ResponseFormat=Text
```

**Example 2:** Trigger a global event, overriding the message to be sent to the operators with the message "External alarm" and response in XML
```
http://192.168.0.1:8601/Interface/GlobalEvents/TriggerGlobalEvent?
Event=Event1&OverrideOperatorMessage=External alarm&ResponseFormat=XML
```

**Example 3:** Trigger a global event, with the message "Test message", overriding the message to be sent to the operators with the message "External alarm" and response in XML
```
http://192.168.0.1:8601/Interface/GlobalEvents/TriggerGlobalEvent?
Event=Event1&Message=Test message&OverrideOperatorMessage=External alarm&
ResponseFormat=XML
```

**Example 4:** Trigger a global event, with the message "Test message", overriding the message to be sent to the operators with the message "External alarm" and overriding the objects Camera1, Camera2, Map1, Map2 and response in XML
```
http://192.168.0.1:8601/Interface/GlobalEvents/TriggerGlobalEvent?
Event=Event1&Message=Test message&OverrideOperatorMessage=External alarm&
OverrideShowObjects=1;Camera1,1;Camera2,9;Map1,9;Map2&ResponseFormat=XML
```

**Response:**
Default response of API.

**HTTP Return:** 200 OK

**Parameters of return:** Default return of API

## 4.10 Virtual matrix

Commands to control virtual matrix

### 4.10.1 Requesting the list of active monitors

Requests the list of active monitors in the virtual matrix of the server.

**Compatibility:** Standard, Professional, Enterprise

**Security level:** Requires authentication of the user with rights to operate the virtual matrix

**Method:** HTTP GET

**Syntax:**
```
http://<server_address>/Interface/VirtualMatrix/GetActiveMonitors
[?<argument=value>[&<argument=value>...]][&<general_argument>...]
```

**Arguments:**

| Argument | Valid values | Description |
|---|---|---|
| Monitors=<*APIMasks*> | String | Mask to filter the results. Specify which monitors must be returned based on the provided masks. |

<table>
<tr>
<td>Fields=<String></td>
<td>MonitorID<br>ClientID<br>ClientAddress<br>ClientUser<br>CurrentObjectType<br>CurrentObjectName<br>CurrentObjectDescription</td>
<td>Specifies the list of desired fields. If this parameter is omitted, all of the fields will be sent.<br><br>The fields must be separated by commas
<table>
<tr><th>Name</th><th>Description</th></tr>
<tr><td>MonitorID</td><td>Identification of the monitor</td></tr>
<tr><td>ClientID</td><td>Identification of the client that owns the monitor</td></tr>
<tr><td>ClientAddress</td><td>IP Address of the client that owns the monitor</td></tr>
<tr><td>ClientUser</td><td>User of the client that owns the monitor</td></tr>
<tr><td>CurrentObjectType</td><td>Type of the object currently being displayed in the monitor</td></tr>
<tr><td>CurrentObjectName</td><td>Name of the object currently being displayed in the monitor</td></tr>
<tr><td>CurrentObjectDescription</td><td>Description of the object currently being displayed in the monitor</td></tr>
</table>
</td>
</tr>
</table>

**Example 1:** Request the list of active monitors with all of the fields and response in XML

```
http://192.168.0.1:8601/Interface/VirtualMatrix/GetActiveMonitors?
ResponseFormat=XML
```

**Example 2:** Request the list of active monitors with all of the fields and response in text

```
http://192.168.0.1:8601/Interface/VirtualMatrix/GetActiveMonitors?
ResponseFormat=Text
```

**Example 3:** Request the list of active monitors with only ID of the monitor, response in XML and authentication of the user Admin

```
http://192.168.0.1:8601/Interface/VirtualMatrix/GetActiveMonitors?
Fields=MonitorID&ResponseFormat=XML&AuthUser=admin&AuthPass=pass
```

**Example 4:** Request the list of active monitors starting with "A", with only monitor ID, response in XML and authentication with Admin user

```
http://192.168.0.1:8601/Interface/VirtualMatrix/GetActiveMonitors?
Monitors=A*&Fields=MonitorID&ResponseFormat=XML&AuthUser=admin&
AuthPass=pass
```

**Response:**
A list with all of the active monitors of the virtual matrix of the server is returned. The fields returned in the list will depend on the values informed in the argument `Fields`

**HTTP Return:** 200 OK

**Parameters of return:**

**Fixed parameters:**

| Parameter | Type | Description |
|---|---|---|
| COUNT | Integer | Total number of active monitors |

**Parameters of the list of active monitors:**

| Parameter | Type | Description |
|---|---|---|
| MONITORID | String | Identification of the monitor (Configured in the Surveillance Client) |
| CLIENTID | String | Identification of the client that owns the monitor |
| CLIENTADDRESS | String | IP Address of the client that owns the monitor |
| CLIENTUSER | String | User of the client that owns the monitor |
| CURRENTOBJECTTYPE | Integer | Type of object currently being displayed in the monitor<br>Value: 0 - No object; 1 - Live camera; 2 - Map; 3 - Screen view; 4 - Analytics configuration; 5 - LPR configuration |
| CURRENTOBJECTNAME | String | Name of the object currently being displayed in the monitor |
| CURRENTOBJECTDESCRIPTION | String | Description of the object currently being displayed in the monitor |

**List of active monitors:**
The parameters of the list of active monitors will depend on the type of response (Text or XML).

**List of active monitors with response in text:**
The parameters of response in text will obey the following syntax:

```
MONITOR_<num>_<field>=<value>
```

| Parameter | Description |
|---|---|
| *num* | Number of the record |
| *field* | Name of the field |
| *value* | Value of the field |

**Example of return in text:**

```
RESPONSE_CODE=0
RESPONSE_MESSAGE=OK
COUNT=2
MONITOR_1_MONITORID=MONITOR 1
MONITOR_1_CLIENTID=AF209016277C36811CFDC95AD0D921A7
MONITOR_1_CLIENTADDRESS=192.168.0.2
MONITOR_1_CLIENTUSER=admin
MONITOR_1_CURRENTOBJECTTYPE=1
MONITOR_1_CURRENTOBJECTNAME=Camera 1
MONITOR_1_CURRENTOBJECTDESCRIPTION=Entrance camera
MONITOR_2_MONITORID=MONITOR 2
MONITOR_2_CLIENTID=ABF3928474CFE7E7FCABC89DBCA98378
MONITOR_2_CLIENTADDRESS=192.168.0.3
MONITOR_2_CLIENTUSER=admin
MONITOR_2_CURRENTOBJECTTYPE=2
MONITOR_2_CURRENTOBJECTNAME=Map 1
MONITOR_2_CURRENTOBJECTDESCRIPTION=First floor
```

**List of active monitors with response in XML:**
Os parâmetros de resposta em XML irão seguir a seguinte sintaxe:

```
<Monitors>
 <Count>COUNT</Count>
 <Monitor>
  <MonitorID>MONITORID</MonitorID>
  <ClientID>CLIENTID</ClientID>
  <ClientAddress>CLIENTADDRESS</ClientAddress>
  <ClientUser>CLIENTUSER</ClientUser>
  <CurrentObjectType>CURRENTOBJECTTYPE</CurrentObjectType>
  <CurrentObjectName>CURRENTOBJECTNAME</CurrentObjectName>
  <CurrentObjectDescription>CURRENTOBJECTDESCRIPTION</CurrentObjectDescription>
 </Monitor>
</Monitors>
```

**Example of return in XML:**

```
<?xml version="1.0" encoding="UTF-8" ?>
<Response>
 <Code>0</Code>
 <Message>OK</Message>
 <Data>
  <Monitors>
   <Count>2</Count>
   <Monitor>
    <MonitorID>Monitor 1</MonitorID>
    <ClientID>AF209016277C36811CFDC95AD0D921A7</ClientID>
    <ClientAddress>192.168.0.2</ClientAddress>
    <ClientUser>admin</ClientUser>
    <CurrentObjectType>1</CurrentObjectType>
    <CurrentObjectName>Camera 1</CurrentObjectName>
    <CurrentObjectDescription>Entrance camera</CurrentObjectDescription>
   </Monitor>
   <Monitor>
    <MonitorID>Monitor 2</MonitorID>
    <ClientID>ABF3928474CFE7E7FCABC89DBCA98378</ClientID>
    <ClientAddress>192.168.0.3</ClientAddress>
    <ClientUser>admin</ClientUser>
    <CurrentObjectType>2</CurrentObjectType>
    <CurrentObjectName>Map 1</CurrentObjectName>
    <CurrentObjectDescription>First floor</CurrentObjectDescription>
   </Monitor>
  </Monitors>
 </Data>
</Response>
```

### 4.10.2 Requesting the list of viewed monitors

Requests the list of viewed monitors in the virtual matrix of the server. This list delivers the registers of all of the monitors of the virtual matrix of the clients that connected to the server in the last 24 hours. This list does not include the registers of currently active monitors.

**Compatibility:** Standard, Professional, Enterprise

**Security level:** Requires authentication of the user with rights to operate the virtual matrix

**Method:** HTTP GET

**Syntax:**

```
http://<server_address>/Interface/VirtualMatrix/GetSeenMonitors
[?<argument=value>[&<argument=value>...]][&<general_argument>...]
```

**Arguments:**

| Argument | Valid values | Description |
|---|---|---|
| Monitors=<*APIMasks*> | String | Mask to filter the results. Specify which monitors must be returned based on the provided masks. |
| Fields=<*String*> | MonitorID<br>ClientID | Specifies the list of desired fields. If this parameter is omitted, all of the fields will be sent. |

| | | |
|---|---|---|
| | ClientAddress<br>ClientUser | The fields must be separated by commas<br><br>**Name** / **Description**<br>MonitorID — Identification of the monitor<br>ClientID — Identification of the client that owns the monitor<br>ClientAddress — IP Address of the client that owns the monitor<br>ClientUser — User of the client that owns the monitor |

**Example 1:** Request the list of seen monitors with all of the fields and response in XML

```
http://192.168.0.1:8601/Interface/VirtualMatrix/GetSeenMonitors?
ResponseFormat=XML
```

**Example 2:** Request the list of seen monitors with all of the fields and response in text

```
http://192.168.0.1:8601/Interface/VirtualMatrix/GetSeenMonitors?
ResponseFormat=Text
```

**Example 3:** Request the list of seen monitors with only ID of the monitor, response in XML and authentication of the user Admin

```
http://192.168.0.1:8601/Interface/VirtualMatrix/GetSeenMonitors?
Fields=MonitorID&ResponseFormat=XML&AuthUser=admin&AuthPass=pass
```

**Example 4:** Request the list of seen monitors starting with "A", with only monitor ID, response in XML and authentication with Admin user

```
http://192.168.0.1:8601/Interface/VirtualMatrix/GetSeenMonitors?
Monitors=A*&Fields=MonitorID&ResponseFormat=XML&AuthUser=admin&
AuthPass=pass
```

**Response:**
A list with all of the viewed monitors in the virtual matrix of the server is returned. The fields returned in the list will depend on the values informed in the argument `Fields`

**HTTP Return:** 200 OK

**Parameters of return:**

**Fixed parameters:**

| Parameter | Type | Description |
|---|---|---|
| COUNT | Integer | Total number of viewed monitors |

**Parameters of the list of viewed monitors:**

| Parameter | Type | Description |
|---|---|---|
| MONITORID | String | Identification of the monitor (Configured in the Surveillance Client) |
| CLIENTID | String | Identification of the client that owns the monitor |
| CLIENTADDRESS | String | IP Address of the client that owns the monitor |
| CLIENTUSER | String | User of the client that owns the monitor |

**List of viewed monitors:**

The parameters of the list of viewed monitors will depend on the type of response (Text or XML).

**List of viewed monitors with response in text:**
The parameters of response in text will obey the following syntax:

```
MONITOR_<num>_<field>=<value>
```

| Parameter | Description |
|---|---|
| *num* | Number of the record |
| *field* | Name of the field |
| *value* | Value of the field |

**Example of return in text:**

```
RESPONSE_CODE=0
RESPONSE_MESSAGE=OK
COUNT=2
MONITOR_1_MONITORID=MONITOR 1
MONITOR_1_CLIENTID=AF209016277C36811CFDC95AD0D921A7
MONITOR_1_CLIENTADDRESS=192.168.0.2
MONITOR_1_CLIENTUSER=admin
MONITOR_2_MONITORID=MONITOR 2
MONITOR_2_CLIENTID=ABF3928474CFE7E7FCABC89DBCA98378
MONITOR_2_CLIENTADDRESS=192.168.0.3
MONITOR_2_CLIENTUSER=admin
```

**List of viewed monitor with response in XML:**
The parameters of response in XML will obey the following syntax:

```
<Monitors>
  <Count>COUNT</Count>
  <Monitor>
    <MonitorID>MONITORID</MonitorID>
    <ClientID>CLIENTID</ClientID>
    <ClientAddress>CLIENTADDRESS</ClientAddress>
    <ClientUser>CLIENTUSER</ClientUser>
  </Monitor>
</Monitors>
```

**Example of return in XML:**

```
<?xml version="1.0" encoding="UTF-8" ?>
<Response>
 <Code>0</Code>
 <Message>OK</Message>
 <Data>
  <Monitors>
   <Count>2</Count>
   <Monitor>
    <MonitorID>Monitor 1</MonitorID>
    <ClientID>AF209016277C36811CFDC95AD0D921A7</ClientID>
    <ClientAddress>192.168.0.2</ClientAddress>
    <ClientUser>admin</ClientUser>
   </Monitor>
   <Monitor>
    <MonitorID>Monitor 2</MonitorID>
    <ClientID>ABF3928474CFE7E7FCABC89DBCA98378</ClientID>
    <ClientAddress>192.168.0.3</ClientAddress>
    <ClientUser>admin</ClientUser>
   </Monitor>
  </Monitors>
 </Data>
</Response>
```

### 4.10.3 Displaying an object in a monitor

By way of this command, you will be able to display an object in any monitor of the virtual matrix.

**Tip:** By way of subsequent calls of this command, you will be able to display the same object in several monitors of the virtual matrix simultaneously.

**Compatibility:** Standard, Professional, Enterprise

**Security level:** Requires authentication of the user with rights to operate the virtual matrix

**Method:** HTTP GET

**Syntax:**

```
http://<server_address>/Interface/VirtualMatrix/ShowObject?
<argument=value>[&<argument=value>...][&<general_argument>...]
```

**Arguments:**

<table>
<tr><th>Argument</th><th>Valid values</th><th>Description</th></tr>
<tr><td>MonitorID=<String>*</td><td>String</td><td>Identification of the monitor</td></tr>
<tr><td>SpotNumber=<Integer>*</td><td>Integer. X =- 1<br>Integer. X >= 1</td><td>Spot number
<table>
<tr><th>Value</th><th>Description</th></tr>
<tr><td>-1</td><td>Object in fullscreen</td></tr>
<tr><td>>= 1</td><td>Spot number</td></tr>
</table>
</td></tr>
<tr><td>ObjectType=<Integer>*</td><td>0, 2</td><td>Type of object
<table>
<tr><th>Type</th><th>Description</th></tr>
</table>
</td></tr>
</table>

| | | |
|---|---|---|
| | | 0 Live camera<br>2 Map<br>3 Analytics Configuration<br>4 LPR Configuration |
| ObjectName=<*String*>* | String | Name of the object to be displayed |

* Mandatory parameters

**Example 1:** Displays a camera in monitor 1 of the virtual matrix with response in text

```
http://192.168.0.1:8601/Interface/VirtualMatrix/ShowObject?
MonitorID=Monitor 1&ObjectType=0&ObjectName=Camera1&ResponseFormat=Text
```

**Example 2:** Displays a map in monitor 25 of the virtual matrix with response in XML

```
http://192.168.0.1:8601/Interface/VirtualMatrix/ShowObject?
MonitorID=Monitor 25&ObjectType=2&ObjectName=Map1&ResponseFormat=XML
```

**Example 3:** Displays a map in spot 2 of monitor 25 of the virtual matrix with response in XML

```
http://192.168.0.1:8601/Interface/VirtualMatrix/ShowObject?
MonitorID=Monitor 25&SpotNumber=2&ObjectType=2&ObjectName=Map1&
ResponseFormat=XML
```

**Response:**
Default response of API.

**HTTP Return:** 200 OK

**Parameter of return:** Default return of API

### 4.10.4 Displaying an user screen view in a monitor

By way of this command, you will be able to display a user screen view in any monitor of the virtual matrix.

**Tip:** By way of subsequent calls of this command, you will be able to display the same screen view in several monitors of the virtual matrix simultaneously.

**Compatibility:** Standard, Professional, Enterprise

**Security level:** Requires authentication of the user with rights to operate the virtual matrix

**Method:** HTTP GET

**Syntax**

```
http://<server_address>/Interface/VirtualMatrix/ShowUserScreenView?
<argument=value>[&<argument=value>...][&<general_argument>...]
```

**Arguments:**

| Argument | Valid values | Description |
|---|---|---|
| MonitorID=<*String*>* | String | Identification of the monitor |
| ScreenStyleID=<*Integer*>* | Integer | ID of the screenstyle |

| | | |
|---|---|---|
| ScreenViewName=<*String*>* | String | Name of the user view |

* Mandatory parameters

**Example 1:** Display the view "View 1" of screenstyle 6278 in monitor 1 of the virtual matrix with response in text

```
http://192.168.0.1:8601/Interface/VirtualMatrix/ShowUserScreenView?
MonitorID=Monitor 1&ScreenStyleID=6278&ScreenViewName=View 1&
ResponseFormat=Text
```

**Example 2:** Display the view "View 2" of user guest1 of screenstyle 1 in monitor 1 of the virtual matrix with response in XML

```
http://192.168.0.1:8601/Interface/VirtualMatrix/ShowUserScreenView?
MonitorID=Monitor 1&ScreenStyleID=1&ScreenViewName=View 2&AuthUser=guest1&
AuthPass=guestpass&ResponseFormat=XML
```

**Response:**
Default response of API.

**HTTP Return:** 200 OK

**Parameter of return:** Default return of API

### 4.10.5 Displaying a public screen view in a monitor

By way of this command, you will be able to display a public screen view in any monitor of the virtual matrix.

**Tip:** By way of subsequent calls of this command, you will be able to display the same screen view in several monitors of the virtual matrix simultaneously.

**Compatibility:** Standard, Professional, Enterprise

**Security level:** Requires authentication of the user with rights to operate the virtual matrix

**Method:** HTTP GET

**Syntax**

```
http://<server_address>/Interface/VirtualMatrix/ShowPublicScreenView?
<argument=value>[&<argument=value>...][&<general_argument>...]
```

**Arguments:**

| Argument | Valid values | Description |
|---|---|---|
| MonitorID=<*String*>* | String | Identification of the monitor |
| ScreenStyleID=<*Integer*>* | Integer | ID of the screenstyle |
| ScreenViewName=<*String*>* | String | Name of the public screen view |

* Mandatory parameters

**Example 1:** Display the view "View 1" of screenstyle 6278 in monitor 1 of the virtual matrix with response in text

```
http://192.168.0.1:8601/Interface/VirtualMatrix/ShowPublicScreenView?
MonitorID=Monitor 1&ScreenStyleID=6278&ScreenViewName=View 1&
ResponseFormat=Text
```

**Example 2:** Display the view "View 2" of screenstyle 1 in monitor 2 of the virtual matrix with response in XML

```
http://192.168.0.1:8601/Interface/VirtualMatrix/ShowPublicScreenView?
MonitorID=Monitor 2&ScreenStyleID=1&ScreenViewName=View 2&
ResponseFormat=XML
```

**Response:**
Default response of API.

**HTTP Return:** 200 OK

**Parameter of return:** Default return of API

### 4.10.6 Starting media playback in a monitor

By using this command you can start a playback session of any camera in any monitor of virtual matrix.

**Tip:** By way of subsequent calls of this command, you will be able to display the same object in several monitors of the virtual matrix simultaneously.

**Compatibility:** Standard, Professional, Enterprise

**Security level:** Requires authentication of the user with rights to operate the virtual matrix

**Method:** HTTP GET

**Syntax:**

```
http://<server_address>/Interface/VirtualMatrix/ShowObject?
<argument=value>[&<argument=value>...][&<general_argument>...]
```

**Arguments:**

| Argument | Valid values | Description |
|---|---|---|
| MonitorID=<*String*>* | String | Monitor identification |
| Cameras=<*String*>* | String | Name of cameras for media playback (List must be comma-separated) |
| SecondsAgo=<*Integer*>* | Integer | Open media playback of X seconds ago<br><br>Specify -1 or omit this value for using StartDate, StartTime, EndDate and EndTime parameters instead. |
| StartDate=<*APIDate*> | Data | Start date of media session |
| StartTime=<*APITime*> | Hora | Start time of media session |
| EndDate=<*APIDate*> | Data | End date of media session. If this parameter is omitted, the current date will be used |
| EndTime=<*APITime*> | Hora | End time of media session. If this parameter is omitted, the current time will be used |

* Mandatory parameters

**Example 1:** Start media playback of cameras "Camera1" and "Camera2" in monitor "Monitor1" to view recordings of last 5 minutes

```
http://192.168.0.1:8601/Interface/VirtualMatrix/StartMediaPlayback?
MonitorID=Monitor1&Cameras=Camera1,Camera2&SecondsAgo=300
```

**Example 2:** Start media playback of camera "Camera1" in monitor "Monitor1" to view recordings from March 07, 2014 17:15:00.000 to March 08, 2014 17:00:00.000

```
http://192.168.0.1:8601/Interface/VirtualMatrix/StartMediaPlayback?
MonitorID=Monitor1&Cameras=Camera1&SecondsAgo=-1&StartDate=2014.03.07&
StartTime=17.15.00.000&EndDate=2014.03.08&EndTime=17.00.00.000
```

**Example 3:** Start media playback of camera "Camera1" in monitor "Monitor1" to view recordings from March 07, 2014 17:15:00.000 to date.

```
http://192.168.0.1:8601/Interface/VirtualMatrix/StartMediaPlayback?
MonitorID=Monitor1&Cameras=Camera1&SecondsAgo=-1&StartDate=2014.03.07&
StartTime=17.15.00.000
```

**Response:**
Default response of API.

**HTTP Return:** 200 OK

**Parameter of return:** Default return of API

## 4.11 Events

Commands to monitor and search for events

### 4.11.1 Searching for events records

Perform a search for event records in the database

**Compatibility:** All editions

**Security level:** Requires user authentication with rights to search for event records

**Method:** HTTP GET

**Syntax:**

```
http://<server_address>/Interface/Events/Search
[?<argument=value>[&<argument=value>...][&<general_argument>...]]
```

**Arguments:**

| Argument | Valid values | Description |
|---|---|---|
| `StartDate=<APIDate>` | Date | Search for records starting with the specified date |
| `StartTime=<APITime>` | Hora | Search for records starting with the specified time |
| `EndDate=<APIDate>` | Data | End date of search. Search for all records inside the range of start and end.<br><br>If this parameter is omitted, the value specified in `StartDate` parameter will be used |

| | | |
|---|---|---|
| EndTime=<*APITime*> | Hora | End time of search. Search for all records inside the range of start and end.<br><br>If this parameter is omitted, the value specified in StartTime parameter will be used |
| EventTypes=<*String*> | String | List of types of events to search. The list must be comma-separated.<br><br>**Event Types**<br>AlarmInput<br>DeviceCommunication<br>RecordingError<br>MotionDetection<br>AudioLevelDetection<br>ManualEvent<br>ScheduledEvent<br>GlobalEvent<br>Analytics<br>LPR<br>ServerFailover |

**Example 1:** Search for all records between January 01, 2014 and February 01, 2014

```
http://192.168.0.1:8601/Interface/Events/Search?StartDate=2014.01.01&
StartTime=00.00.00.000&EndDate=2014.02.01&EndTime=23.59.59.999
```

**Example 2:** Search for all records of AlarmInput and LPR

```
http://192.168.0.1:8601/Interface/Events/Search?Events=AlarmInput,LPR
```

**Response:**
A list with all found event records is returned.

**HTTP Return:** 200 OK

**Parameters of return:**

**Fixed Parameter:**

| Parameter | Type | Description |
|---|---|---|
| COUNT | Integer | Total of records from all event types |

**Parameters of records list:**

| Parameter | Type | Description |
|---|---|---|
| ALARMINPUTCOUNT | Integer | Number of records of alarm input events |
| DEVICECOMMUNICATIONCOUNT | Integer | Number of records of device communication events |
| RECORDINGERRORCOUNT | Integer | Number of records of recording error events |
| MOTIONDETECTIONCOUNT | Integer | Number of records of motion detection events |
| AUDIOLEVELDETECTIONCOUNT | Integer | Number of records of audio level detection events |
| MANUALEVENTCOUNT | Integer | Number of records of manual events |

| Parameter | Type | Description |
|---|---|---|
| SCHEDULEDEVENTCOUNT | Integer | Number of records of scheduled events |
| GLOBALEVENTCOUNT | Integer | Number of records of global events |
| ANALYTICSCOUNT | Integer | Number of records of analytics events |
| LPRCOUNT | Integer | Number of records of LPR events |
| SERVERFAILOVERCOUNT | Integer | Number of records of server failover events |
| RECORDNUMBER | Integer | Number of records on database |
| DATETIME | APITimestamp | Date and time of the event |
| EVENTNAME* | String | Name of the event |
| DEVICENAME* | String | Name of the device |
| ANALYTICSTYPE* | String | Type of analytics |
| ZONE* | String | Analytics zone name |
| LICENSEPLATE* | String | License plate string |
| USERNAME* | String | Name of user |
| IP* | String | Network address |
| AUDIOLEVEL* | String | Type of audio level |
| SERVERFAILOVER* | String | Type of failover event |
| DEVICECOMMUNICATIONEVENT* | String | Type of device communication event |
| DEVICECOMMUNICATIONFAILURETIME* | String | Total failure time of device |

* These parameters are event-dependent and may only be available in certain types of events

**List of records:**
The parameters of the list of event records will depend on the type of response (Text or XML).

**List of event records with response in text:**
The parameters of response in text will obey the following syntax:

```
<eventtype>RECORD_<num>_<field>=<value>
```

| Parameter | Description |
|---|---|
| *eventtype* | Type of event<br><br>**Event Types**<br>ALARMINPUT<br>DEVICECOMMUNICATION<br>RECORDINGERROR<br>MOTIONDETECTION<br>AUDIOLEVELDETECTION<br>MANUALEVENT<br>SCHEDULEDEVENT<br>GLOBALEVENT<br>ANALYTICS<br>LPR<br>SERVERFAILOVER |
| *num* | Number of the record |
| *field* | Name of the field |
| *value* | Value of the field |

**Example of return in text:**

```
RESPONSE_CODE=0
RESPONSE_MESSAGE=OK
COUNT=2
DEVICECOMMUNICATIONCOUNT=1
ANALYTICSCOUNT=1
DEVICECOMMUNICATIONRECORD_1_RECORDNUMBER=7324163
DEVICECOMMUNICATIONRECORD_1_DATETIME=2015-09-27 17:00:17.308
DEVICECOMMUNICATIONRECORD_1_DEVICENAME=Internal I/O Device
DEVICECOMMUNICATIONRECORD_1_DEVICECOMMUNICATIONEVENT=Communication failure
ANALYTICSRECORD_1_RECORDNUMBER=7324209
ANALYTICSRECORD_1_DATETIME=2015-09-27 17:09:11.295
ANALYTICSRECORD_1_DEVICENAME=04
ANALYTICSRECORD_1_ANALYTICSTYPE=Face detection
```

**List of events records with response in XML:**
The parameters of response in XML will obey the following syntax:

```
<EventLogRecords>
 <Count>COUNT</Count>
 <ServerFailoverCount>FAILOVER_EVENT_COUNT</ServerFailoverCount>
 <LPRCount>LPR_EVENT_COUNT</LPRCount>
 <AnalyticsCount>ANALYTICS_EVENT_COUNT</AnalyticsCount>
 <GlobalEventCount>GLOBAL_EVENT_COUNT</GlobalEventCount>
 <ScheduledEventCount>SCHEDULED_EVENT_COUNT</ScheduledEventCount>
 <ManualEventCount>MANUAL_EVENT_COUNT</ManualEventCount>
 <AudioLevelDetectionCount>AUDIO_LEVEL_EVENT_COUNT</AudioLevelDetectionCount>
 <MotionDetectionCount>MOTION_DETECTION_EVENT_COUNT</MotionDetectionCount>
 <RecordingErrorCount>RECORDING_ERROR_EVENT_COUNT</RecordingErrorCount>
 <DeviceCommunicationCount>DEVICE_COMM_EVENT_COUNT</DeviceCommunicationCount>
 <AlarmInputCount>ALARM_INPUT_EVENT_COUNT</AlarmInputCount>

 <!--
   Each event type will have a subsection containing its events list and properties, below is an example
   of LPR events
 -->
 <LPRRecords>
   <LPRRecord>
     <RecordNumber>RECORD_NUMBER</RecordNumber>
     <DateTime>DATE_TIME</DateTime>
     <DeviceName>DEVICE_NAME</DeviceName>
     <LicensePlate>LICENSE_PLATE</LicensePlate>
   </LPRRecord>
 </LPRRecords>

</EventLogRecords>
```

**Example of return in XML:**

```
<?xml version="1.0" encoding="UTF-8" ?>
<Response>
 <Code>0</Code>
 <Message>OK</Message>
 <Data>
  <EventLogRecords>
   <Count>2</Count>
   <DeviceCommunicationCount>1</DeviceCommunicationCount>
   <AnalyticsCount>1</AnalyticsCount>
   <DeviceCommunicationRecords>
    <DeviceCommunicationRecord>
     <RecordNumber>7324163</RecordNumber>
     <DateTime>2015-09-27 17:00:17.308</DateTime>
     <DeviceName>Internal I/O Device</DeviceName>
     <DeviceCommunicationEvent>Communication failure</DeviceCommunicationEvent>
    </DeviceCommunicationRecord>
   </DeviceCommunicationRecords>
   <AnalyticsRecords>
    <AnalyticsRecord>
     <RecordNumber>7324209</RecordNumber>
     <DateTime>2015-09-27 17:09:11.295</DateTime>
     <DeviceName>04</DeviceName>
     <AnalyticsType>Face detection</AnalyticsType>
    </AnalyticsRecord>
   </AnalyticsRecords>
  </EventLogRecords>
 </Data>
</Response>
```

### 4.11.2 Monitoring server events

Start the server events monitoring.

**Compatibility:** All editions

**Security level:** Admin

**Method:** HTTP GET

**Syntax:**

```
http://<server_address>/Interface/Events/Monitor
[?<argument=value>[&<argument=value>...]][&<general_argument>...]
```

**Arguments:**

| Argument | Valid values | Description |
|---|---|---|
| KeepAliveInterval=<*Integer*> | Integer. 1 >= X <= 120 | Specify the time (in seconds) of Keep-Alive packet sending. The Keep-Alive packet is used to control the connection. If this parameter is omitted, the default value of 5 will be used |

**Example 1:** Start the server events monitoring with response in XML

```
http://192.168.0.1:8601/Interface/Events/Monitor?ResponseFormat=XML
```

**Example 2:** Start the server events monitoring with response in text and keep-alive interval of 60 seconds

```
http://192.168.0.1:8601/Interface/Events/Monitor?ResponseFormat=Text&
KeepAliveInterval=60
```

**Example 3:** Start the server events monitoring with response in XML and authentication of the user Admin

```
http://192.168.0.1:8601/Interface/Events/Monitor?
ResponseFormat=XML&AuthUser=admin&AuthPass=pass
```

**Response:**
A stream of events data will be sent using HTTP Multipart `x-mixed-replace` transmission.
The response data stream is continuous, so an event of Keep-Alive will be sent every X seconds (Configurable by `KeepAliveInterval` parameter), with this event you can verify if the TCP connection is still open.
Each event is separated by the multipart boundary `--DigifortBoundary`.

**HTTP Return:** 200 OK

**Parameter of return:**
The return parameters are divided into two subcategories: `ObjectData` and `EventData`.

**ObjectData parameters:**

| Parameter | Type | Description |
|---|---|---|
| NAME | String | Name of the object that triggered the event |
| TYPE | String | Type of object that triggered the event<br><br>Value: CAMERA — Description: Camera object<br>Value: IO_DEVICE — Description: I/O device object<br>Value: SERVER — Description: Digifort server<br>Value: SCHEDULED_EVENT — Description: Scheduled event object<br>Value: GLOBAL_EVENT — Description: Global event object<br>Value: ANALYTICS_CONFIGURATION — Description: Analytics object<br>Value: LPR_EVENT — Description: LPR event object |

**EventData parameters:**

| Parameter | Type | Description |
|---|---|---|
| NAME | String | Event name. Only applicable to events: ALARM_INPUT, MANUAL, TIMER. |
| TYPE | String | Event type<br><br>Value: ALARM_INPUT — Description: Alarm input event. Triggered by: CAMERA, ALARM_DEVICE<br>Value: MANUAL — Description: Manual event. Triggered by: CAMERA<br>Value: TIMER — Description: Timer event. Triggered by any object<br>Value: COMMUNICATION_FAILURE — Description: Device communication failure event. Triggered by: CAMERA, ALARM_DEVICE |

| Parameter | Type | Description | |
|---|---|---|---|
| | | COMMUNICATION_RESTORED | Device communication restored event. Triggered by: CAMERA, ALARM_DEVICE |
| | | RECORDING_FAILURE | Camera recording failure event. Triggered by: CAMERA |
| | | MOTION | Camera motion detection event. Triggered by: CAMERA |
| | | SCHEDULED | Scheduled event. Triggered by: SCHEDULED_EVENT |
| | | GLOBAL | Global event. Triggered by: GLOBAL_EVENT |
| | | AUDIO_LEVEL_LOW | Audio level is low. Triggered by CAMERA |
| | | AUDIO_LEVEL_HIGH | Audio level is high. Triggered by CAMERA |
| | | FAILOVER | Failover event indicating that a server being monitored is not working. Triggered by FAILOVER_SERVER_MONITOR |
| | | FAILBACK | Failover event indicating that a server being monitored is working again. Triggered by FAILOVER_SERVER_MONITOR |
| | | KEEP_ALIVE | HTTP Connection Keep-Alive message. Sent every 5 seconds to keep the TCP / HTTP connection |
| | | ANALYTICS_PRESENCE | Analytics presence rule event. Triggered by: ANALYTICS_CONFIGURATION |
| | | ANALYTICS_ENTER | Analytics enter rule event. Triggered by: ANALYTICS_CONFIGURATION |
| | | ANALYTICS_EXIT | Analytics exit rule event. Triggered by: ANALYTICS_CONFIGURATION |
| | | ANALYTICS_APPEAR | Analytics appear rule event. Triggered by: ANALYTICS_CONFIGURATION |
| | | ANALYTICS_DISAPPEAR | Analytics disappear rule event. Triggered by: ANALYTICS_CONFIGURATION |
| | | ANALYTICS_STOP | Analytics stop rule event. Triggered by: ANALYTICS_CONFIGURATION |
| | | ANALYTICS_LOITER | Analytics loitering rule event. Triggered by: ANALYTICS_CONFIGURATION |
| | | ANALYTICS_DIRECTION | Analytics direction filter rule event. Triggered by: ANALYTICS_CONFIGURATION |
| | | ANALYTICS_SPEED | Analytics speed filter rule event. Triggered by: |

| Parameter | Type | Description |
|---|---|---|
| | | ANALYTICS_CONFIGURATION |
| | | ANALYTICS_TAILGATING: Analytics tailgating filter rule event. Triggered by: ANALYTICS_CONFIGURATION |
| | | ANALYTICS_COUNTING_LINE_A: Analytics counting line rule event for direction A: Triggered by: ANALYTICS_CONFIGURATION |
| | | ANALYTICS_COUNTING_LINE_B: Analytics counting line rule event for direction B: Triggered by: ANALYTICS_CONFIGURATION |
| | | ANALYTICS_TAMPER: Analytics camera tampering rule event. Triggered by: ANALYTICS_CONFIGURATION |
| | | ANALYTICS_ABANDONED_OBJECT: Analytics abandoned object rule event. Triggered by: ANALYTICS_CONFIGURATION |
| | | ANALYTICS_REMOVED_OBJECT: Analytics removed object rule event. Triggered by: ANALYTICS_CONFIGURATION |
| | | ANALYTICS_SMOKE: Analytics smoke detection rule event. Triggered by: ANALYTICS_CONFIGURATION |
| | | ANALYTICS_FIRE: Analytics fire detection rule event. Triggered by: ANALYTICS_CONFIGURATION |
| | | ANALYTICS_FACE_DETECTION: Analytics face detection rule event. Triggered by: ANALYTICS_CONFIGURATION |
| | | LPR: LPR event. Triggered by: LPR_EVENT |
| TIMESTAMP | APITimestamp | Date and time of the event |
| UTCTIMESTAMP | APITimestamp | UTC date and time of the event |

**Example of return in text:**

```
--DigifortBoundary
Content-Type: text/plain; charset=UTF-8
Content-Length: 217

RESPONSE_CODE=0
RESPONSE_MESSAGE=OK
OBJECTDATA_NAME=Event1
OBJECTDATA_TYPE=GLOBAL_EVENT
EVENTDATA_NAME=
EVENTDATA_TYPE=GLOBAL
EVENTDATA_TIMESTAMP=2009-11-15 10:55:30.347
EVENTDATA_UTCTIMESTAMP=2009-11-15 13:55:30.347
--DigifortBoundary
Content-Type: text/plain; charset=UTF-8
Content-Length: 203

RESPONSE_CODE=0
RESPONSE_MESSAGE=OK
OBJECTDATA_NAME=
OBJECTDATA_TYPE=
EVENTDATA_NAME=
EVENTDATA_TYPE=KEEP_ALIVE
EVENTDATA_TIMESTAMP=2009-11-15 10:55:32.550
EVENTDATA_UTCTIMESTAMP=2009-11-15 13:55:32.550
--DigifortBoundary
..
..
```

**Example of return in XML:**

```
--DigifortBoundary
Content-Type: text/xml; charset=UTF-8
Content-Length: 323

<?xml version="1.0" encoding="UTF-8" ?>
<Response>
  <Code>0</Code>
  <Message>OK</Message>
  <Data>
    <Event>
      <ObjectData>
        <Name>Event1</Name>
        <Type>GLOBAL_EVENT</Type>
      </ObjectData>
      <EventData>
        <Name></Name>
        <Type>GLOBAL</Type>
        <Timestamp>2009-11-15 10:55:30.347</Timestamp>
        <UTCTimestamp>2009-11-15 13:55:30.347</UTCTimestamp>
      </EventData>
    </Event>
  </Data>
</Response>
--DigifortBoundary
Content-Type: text/xml; charset=UTF-8
Content-Length: 337

<?xml version="1.0" encoding="UTF-8" ?>
<Response>
  <Code>0</Code>
  <Message>OK</Message>
  <Data>
    <Event>
      <ObjectData>
        <Name></Name>
        <Type></Type>
      </ObjectData>
      <EventData>
        <Name></Name>
        <Type>KEEP_ALIVE</Type>
        <Timestamp>2009-11-15 10:55:32.550</Timestamp>
        <UTCTimestamp>2009-11-15 13:55:32.550</UTCTimestamp>
      </EventData>
    </Event>
  </Data>
</Response>
--DigifortBoundary
..
..
```

# 4.12 LPR

Commands to control LPR (License Plate Recognition)

## 4.12.1 Requesting the list of LPR Configurations

Request the list of LPR Configurations registered in the server

**Compatibility:** Standard, Professional, Enterprise

**Security level:** Admin

**Method:** HTTP GET

**Syntax:**

```
http://<server_address>/Interface/LPR/GetLPRConfigurations[?<argument=value>
[&<argument=value>...][&<general_argument>...]]
```

**Arguments:**

| Argument | Valid values | Description |
|---|---|---|
| LPRConfigurations=<*APIMasks*> | String | Mask to filter the results. Specify which LPR Configurations must be returned based on the provided masks. |
| Fields=<*String*> | Name<br>Description<br>Camera | Specifies the list of desired fields. In case this parameter is omitted, all of the fields will be sent.<br><br>The fields must be separated by commas<br><br>**Name** / **Description**<br>Name / Name of the LPR Configuration<br>Description / Description of the LPR Configuration<br>Camera / Associated camera with the LPR Configuration |

**Example 1:** Request the list of LPR Configurations with all fields and response in XML

```
http://192.168.0.1:8601/Interface/LPR/GetLPRConfigurations?
ResponseFormat=XML
```

**Example 2:** Request the list of LPR Configurations with all fields and response in text

```
http://192.168.0.1:8601/Interface/LPR/GetLPRConfigurations?
ResponseFormat=Text
```

**Example 3:** Request the list of LPR Configurations with only name and description, response in XML and authentication of the user Admin

```
http://192.168.0.1:8601/Interface/LPR/GetLPRConfigurations?Fields=Name,
Description&ResponseFormat=XML&AuthUser=admin
```

**Example 4:** Request the list of LPR Configurations starting with "A", with only name and description, response in XML and authentication with Admin user

```
http://192.168.0.1:8601/Interface/LPR/GetLPRConfigurations?
LPRConfigurations=A*&Fields=Name,Description&ResponseFormat=XML&
AuthUser=admin
```

**Response:**
A list of all of the LPR Configurations registered in the system is returned. The fields returned in the will depend on the values informed in the argument `Fields`

**HTTP Return:** 200 OK

**Parameters of return:**

**Fixed Parameter:**

| Parameter | Type | Description |
|---|---|---|
| COUNT | Integer | Total number of LPR Configurations |

**Parameters in the list of LPR Configurations:**

| Parameter | Type | Description |
|---|---|---|
| NAME | String | Name of the LPR Configuration |
| DESCRIPTION | String | Description of the LPR Configuration |
| CAMERA | String | Name of the camera associated to the LPR Configuration |

**List of LPR Configurations:**
The parameters of the list of LPR Configurations will depend on the type of response (Text or XML).

**List of LPR Configurations with response in text:**
The parameters of response in text will obey the following syntax:

```
LPRCONFIGURATION_<num>_<field>=<value>
```

| Parameter | Description |
|---|---|
| *num* | Number of the record |
| *field* | Name of the field |
| *value* | Value of the field |

**Example of return in text:**

```
RESPONSE_CODE=0
RESPONSE_MESSAGE=OK
COUNT=2
LPRCONFIGURATION_1_NAME=Configuration1
LPRCONFIGURATION_1_DESCRIPTION=My Configuration
LPRCONFIGURATION_1_CAMERA=Camera1
LPRCONFIGURATION_2_NAME=Configuration2
LPRCONFIGURATION_2_DESCRIPTION=My Configuration
LPRCONFIGURATION_2_CAMERA=Camera2
```

**List of LPR Configurations with response in XML:**
The parameters of response in XML will obey the following syntax:

```
<LPRConfigurations>
  <Count>COUNT</Count>
  <LPRConfiguration>
    <Name>NAME</Name>
    <Description>DESCRIPTION</Description>
```

```
    <Camera>CAMERA</Camera>
  </LPRConfiguration>
</LPRConfigurations>
```

**Example of return in XML:**

```
<?xml version="1.0" encoding="UTF-8" ?>
<Response>
  <Code>0</Code>
  <Message>OK</Message>
  <Data>
    <LPRConfigurations>
      <Count>2</Count>
      <LPRConfiguration>
        <Name>Configuration1</Name>
        <Description>My Configuration</Description>
        <Camera>Camera1</Camera>
      </LPRConfiguration>
      <LPRConfiguration>
        <Name>Configuration2</Name>
        <Description>My Configuration</Description>
        <Camera>Camera2</Camera>
      </LPRConfiguration>
    </LPRConfigurations>
  </Data>
</Response>
```

### 4.12.2 Requesting the list of surrounding cameras of an LPR Configuration

Request the list of surrounding cameras of an LPR Configuration.

**Compatibility:** Standard, Professional, Enterprise

**Security level:** Admin

**Method:** HTTP GET

**Syntax:**

```
http://<server_address>/Interface/LPR/GetLPRConfigurations[?<argument=value>
[&<argument=value>...][&<general_argument>...]]
```

**Arguments:**

| Argument | Valid values | Description |
|---|---|---|
| LPRConfiguration=<*String*>* | String | LPR Configuration name |
| SurroundingCameras=<*APIMasks*> | String | Mask to filter the results. Specify which surrounding cameras must be returned based on the provided masks. |

* Mandatory parameters

**Example 1:** Request the list of surrounding cameras of LPR Configuration "LPR1", with response in XML

```
http://192.168.0.1:8601/Interface/LPR/GetSurroundingCameras?
LPRConfiguration=LPR1&ResponseFormat=XML
```

**Example 2:** Request the list of surrounding cameras starting with "A" of LPR Configuration "LPR1", with

response in text

```
http://192.168.0.1:8601/Interface/LPR/GetSurroundingCameras?
LPRConfiguration=LPR1&SurroundingCameras=A*&ResponseFormat=Text
```

**Response:**
A list with all surrounding cameras of the specified LPR Configuration is returned

**HTTP Return:** 200 OK

**Parameters of return:**

**Fixed Parameter:**

| Parameter | Type | Description |
|---|---|---|
| COUNT | Integer | Total of surrounding cameras |

**Parameters of surrounding cameras list:**

| Parameter | Type | Description |
|---|---|---|
| NAME | String | Name of surrounding camera |

**List of surrounding cameras:**
The parameters of the list of surrounding cameras will depend on the type of response (Text or XML).

**List of surrounding cameras with response in text:**
The parameters of response in text will obey the following syntax:

```
SURROUNDINGCAMERA_<num>_<field>=<value>
```

| Parameter | Description |
|---|---|
| *num* | Number of the record |
| *field* | Name of the field |
| *value* | Value of the field |

**Example of return in text:**

```
RESPONSE_CODE=0
RESPONSE_MESSAGE=OK
COUNT=4
SURROUNDINGCAMERA_1_NAME=Camera1
SURROUNDINGCAMERA_2_NAME=Camera2
SURROUNDINGCAMERA_3_NAME=Camera3
SURROUNDINGCAMERA_4_NAME=Camera4
```

**List of surrounding cameras with response in XML:**
The parameters of response in XML will obey the following syntax:

```
<SurroundingCameras>
  <Count>COUNT</Count>
  <SurroundingCamera>
    <Name>NAME</Name>
  </SurroundingCamera>
</SurroundingCameras>
```

**Example of return in XML:**

```
<?xml version="1.0" encoding="UTF-8" ?>
<Response>
 <Code>0</Code>
 <Message>OK</Message>
 <Data>
  <SurroundingCameras>
   <Count>4</Count>
   <SurroundingCamera>
    <Name>Camera1</Name>
   </SurroundingCamera>
   <SurroundingCamera>
    <Name>Camera2</Name>
   </SurroundingCamera>
   <SurroundingCamera>
    <Name>Camera3</Name>
   </SurroundingCamera>
   <SurroundingCamera>
    <Name>Camera4</Name>
   </SurroundingCamera>
  </SurroundingCameras>
 </Data>
</Response>
```

### 4.12.3 Searching for LPR records

Perform a search for LPR records in the database

**Compatibility:** Standard, Professional, Enterprise

**Security level:** Requires user authentication with rights to search for LPR records

**Method:** HTTP GET

**Syntax:**

```
http://<server_address>/Interface/LPR/Search
[?<argument=value>[&<argument=value>...][&<general_argument>...]]
```

**Arguments:**

| Argument | Valid values | Description |
|---|---|---|
| `StartDate=<APIDate>` | Date | Search for records starting with the specified date |
| `StartTime=<APITime>` | Hora | Search for records starting with the specified time |
| `EndDate=<APIDate>` | Data | End date of search. Search for all records inside the range of start and end.<br><br>If this parameter is omitted, the value specified in `StartDate` parameter will be used |
| `EndTime=<APITime>` | Hora | End time of search. Search for all records inside the range of start and end.<br><br>If this parameter is omitted, the value specified in `StartTime` parameter will be used |

| | | |
|---|---|---|
| Cameras=<*String*> | String | List of cameras to search. The list with camera names must be comma-separated |
| LicensePlates=<*String*> | String | List of license plates to search. The list with license plates must be comma-separated.<br><br>The plates could include the special character asterisk ("*") that can be used as mask for search |
| LPRConfigurations=<*String*> | String | List of LPR Configurations to search. The list with LPR Configuration names must be comma-separated |
| OrderBy=<*String*> | String | Sort the records by the specified column.<br><br>**Columns**<br>RecordNumber<br>DateTime<br>LicensePlate<br>Camera<br>LPRConfiguration<br><br>If this parameter is omitted, the records will be displayed in retrieval order (Faster method with less server memory usage) |

**Example 1:** Search for all records between January 01, 2014 and February 01, 2014

```
http://192.168.0.1:8601/Interface/LPR/Search?StartDate=2014.01.01&
StartTime=00.00.00.000&EndDate=2014.02.01&EndTime=23.59.59.999
```

**Example 2:** Search for all records of plates ABC0001 and DEF1234

```
http://192.168.0.1:8601/Interface/LPR/Search?LicensePlates=ABC0001,DEF1234
```

**Example 3:** Search for all plates starting with "A", sorting the result by license plate

```
http://192.168.0.1:8601/Interface/LPR/Search?LicensePlates=A*&
OrderBy=LicensePlate
```

**Response:**
A list with all found LPR records is returned.

**HTTP Return:** 200 OK

**Parameters of return:**

**Fixed Parameter:**

| Parameter | Type | Description |
|---|---|---|
| COUNT | Integer | Total of records |

**Parameters of records list:**

| Parameter | Type | Description |
|---|---|---|
| RECORDNUMBER | Integer | Database record number |
| DATETIME | APITimestamp | Record date and time |
| LICENSEPLATE | String | Recognized license plate string |

<table>
<tr><th>Parameter</th><th>Type</th><th>Description</th></tr>
<tr><td>CAMERA</td><td>String</td><td>Name of camera used for recognition</td></tr>
<tr><td>LPRCONFIGURATION</td><td>String</td><td>Name of configuration used for recognition</td></tr>
<tr><td>RELIABILITY</td><td>String</td><td>Reliability of recogition<br><table><tr><th>Value</th><th>Description</th></tr><tr><td>No Reliability</td><td>No reliability value. The used LPR engine does not provide reliability values</td></tr><tr><td>Low</td><td>Low reliability</td></tr><tr><td>Medium</td><td>Medium reliability</td></tr><tr><td>High</td><td>High reliability</td></tr></table></td></tr>
<tr><td>HIT</td><td>String</td><td>This value identifies if the engine believes having<br><br>This value identifies whether the engine believed to have hit or wrong the recognized value<br><table><tr><th>Value</th><th>Description</th></tr><tr><td>Error</td><td>Error on plate recognition</td></tr><tr><td>Hit</td><td>The engine believes to have hit the value</td></tr><tr><td>Uncertainty</td><td>The engine is not sure to have hit the value</td></tr></table></td></tr>
</table>

**List of records:**
The parameters of the list of LPR records will depend on the type of response (Text or XML).

**List of LPR records with response in text:**
The parameters of response in text will obey the following syntax:

```
LPRRECORD_<num>_<field>=<value>
```

| Parameter | Description |
|---|---|
| *num* | Number of the record |
| *field* | Name of the field |
| *value* | Value of the field |

**Example of return in text:**

```
RESPONSE_CODE=0
RESPONSE_MESSAGE=OK
COUNT=2
LPRRECORD_1_RECORDNUMBER=392927
LPRRECORD_1_DATETIME=2014-11-13 14:19:35.912
LPRRECORD_1_LICENSEPLATE=AEJ0400
LPRRECORD_1_CAMERA=40
LPRRECORD_1_LPRCONFIGURATION=LPR 3
LPRRECORD_1_RELIABILITY=Low
LPRRECORD_1_HIT=Uncertainty
LPRRECORD_2_RECORDNUMBER=410268
LPRRECORD_2_DATETIME=2014-11-28 17:44:48.424
LPRRECORD_2_LICENSEPLATE=ABC0460
LPRRECORD_2_CAMERA=40
LPRRECORD_2_LPRCONFIGURATION=LPR 3
LPRRECORD_2_RELIABILITY=Low
LPRRECORD_2_HIT=Uncertainty
```

**List of LPR records with response in XML:**
The parameters of response in XML will obey the following syntax:

```
<LPRRecords>
 <Count>COUNT</Count>
 <LPRRecord>
  <RecordNumber>RECORD_NUMBER</RecordNumber>
  <DateTime>DATE_TIME</DateTime>
  <LicensePlate>LICENSE_PLATE</LicensePlate>
  <Camera>CAMERA_NAME</Camera>
  <LPRConfiguration>LPR_CONFIGURATION</LPRConfiguration>
  <Reliability>RELIABILITY_VALUE</Reliability>
  <Hit>HIT_VALUE</Hit>
 </LPRRecord>
</LPRRecords>
```

**Example of return in XML:**

```
<?xml version="1.0" encoding="UTF-8" ?>
<Response>
 <Code>0</Code>
 <Message>OK</Message>
 <Data>
  <LPRRecords>
   <Count>2</Count>
   <LPRRecord>
    <RecordNumber>392927</RecordNumber>
    <DateTime>2014-11-13 14:19:35.912</DateTime>
    <LicensePlate>AEJ0400</LicensePlate>
    <Camera>40</Camera>
    <LPRConfiguration>LPR 3</LPRConfiguration>
    <Reliability>Low</Reliability>
    <Hit>Uncertainty</Hit>
   </LPRRecord>
   <LPRRecord>
    <RecordNumber>410268</RecordNumber>
    <DateTime>2014-11-28 17:44:48.424</DateTime>
    <LicensePlate>ABC0460</LicensePlate>
    <Camera>40</Camera>
    <LPRConfiguration>LPR 3</LPRConfiguration>
    <Reliability>Low</Reliability>
    <Hit>Uncertainty</Hit>
   </LPRRecord>
  </LPRRecords>
 </Data>
</Response>
```

### 4.12.4 Requesting the data of an LPR record

Request the data of an LPR record.

**Compatibility:** Standard, Professional, Enterprise

**Security level:** Admin

**Method:** HTTP GET

**Syntax:**

```
http://<server_address>/Interface/LPR/GetRecordData[?<argument=value>
[&<argument=value>...][&<general_argument>...]]
```

**Arguments:**

| Argument | Valid values | Description |
|---|---|---|
| RecordNumber=<*Integer*>* | Integer. X >= 0 | LPR record number |

* Mandatory parameters

**Example 1:** Request the record 40 with response in XML

```
http://192.168.0.1:8601/Interface/LPR/GetRecordData?RecordNumber=40&
ResponseFormat=XML
```

**Example 2:** Request the record 237 with response in text

```
http://192.168.0.1:8601/Interface/LPR/GetRecordData?RecordNumber=237&
ResponseFormat=Text
```

**Example 3:** Request the record 500 with response in XML and authentication with Admin user

```
http://192.168.0.1:8601/Interface/LPR/GetRecordData?RecordNumber=500&
ResponseFormat=XML&AuthUser=admin
```

**Response:**
If the record is found, a list of parameter-value pairs is returned, otherwise, the error `4000 Record not found` is returned

**HTTP Return:** 200 OK

**Parameters of return:**

<table>
<tr><th>Parameter</th><th>Type</th><th>Description</th></tr>
<tr><td>RECORDNUMBER</td><td>Integer</td><td>Record number</td></tr>
<tr><td>DATETIME</td><td>APITimestamp</td><td>Date and time of the record in timestamp format</td></tr>
<tr><td>LPRCONFIGURATION</td><td>String</td><td>Name of the LPR Configuration</td></tr>
<tr><td>CAMERA</td><td>String</td><td>Camera name</td></tr>
<tr><td>LICENSEPLATE</td><td>String</td><td>Recognized plate characters</td></tr>
<tr><td>RELIABILITY</td><td>String</td><td>Reliability of reading
<table>
<tr><th>Value</th><th>Description</th></tr>
<tr><td>No Reliability</td><td>No reliability value. The used LPR engine does not provide reliability values</td></tr>
<tr><td>Low</td><td>Low reliability</td></tr>
<tr><td>Medium</td><td>Medium reliability</td></tr>
<tr><td>High</td><td>High reliability</td></tr>
</table>
</td></tr>
<tr><td>HIT</td><td>String</td><td>This value identifies if the engine believes having<br><br>This value identifies whether the engine believed to have hit or wrong the recognized value
<table>
<tr><th>Value</th><th>Description</th></tr>
<tr><td>Error</td><td>Error on plate recognition</td></tr>
<tr><td>Hit</td><td>The engine believes to have hit the value</td></tr>
<tr><td>Uncertainty</td><td>The engine is not sure to have hit the value</td></tr>
</table>
</td></tr>
</table>

**Example of return in text:**

```
RESPONSE_CODE=0
RESPONSE_MESSAGE=OK
RECORDNUMBER=40
DATETIME=2009-11-15 15:40:23.453
LPRCONFIGURATION=LPR1
CAMERA=Camera1
LICENSEPLATE=ABC1234
RELIABILITY=High
HIT=Hit
```

**Example of return in XML:**

```
<?xml version="1.0" encoding="UTF-8" ?>
<Response>
 <Code>0</Code>
 <Message>OK</Message>
 <Data>
  <RecordData>
   <RecordNumber>40</RecordNumber>
   <DateTime>2009-11-15 15:40:23.453</DateTime>
   <LPRConfiguration>LPR1</LPRConfiguration>
   <Camera>Camera1</Camera>
   <LicensePlate>ABC1234</LicensePlate>
   <Reliability>High</Reliability>
   <Hit>Hit</Hit>
  </RecordData>
 </Data>
</Response>
```

### 4.12.5 Requesting the image of an LPR record

Request the image of an LPR record.

**Compatibility:** Standard, Professional, Enterprise

**Security level:** Admin

**Method:** HTTP GET

**Syntax:**

```
http://<server_address>/Interface/LPR/GetRecordImage[?<argument=value>
[&<argument=value>...][&<general_argument>...]]
```

**Arguments:**

| Argument | Valid values | Description |
|---|---|---|
| RecordNumber=<*Integer*>* | Integer. X >= 0 | LPR record number |

* Mandatory parameters

**Example 1:** Request the image of record 40 with error response in XML

```
http://192.168.0.1:8601/Interface/LPR/GetRecordImage?RecordNumber=40&
ResponseFormat=XML
```

**Example 2:** Request the image of record 237 with error response in text

```
http://192.168.0.1:8601/Interface/LPR/GetRecordImage?RecordNumber=237&
ResponseFormat=Text
```

**Example 3:** Request the image of record 500 with error response in XML and authentication with Admin user

```
http://192.168.0.1:8601/Interface/LPR/GetRecordImage?RecordNumber=500&
ResponseFormat=XML&AuthUser=admin
```

**Response:**
If the record is found, its JPEG image is returned, otherwise, the error `4000 Record not found` is returned.

**HTTP Return:** 200 OK

**Parameters of return:** Default return of API

### 4.12.6 Requesting the data of the last LPR record

Request the data of the last LPR record of a camera or database.

**Compatibility:** Standard, Professional, Enterprise

**Security level:** Admin

**Method:** HTTP GET

**Syntax:**

```
http://<server_address>/Interface/LPR/GetLastRecordData[?<argument=value>
[&<argument=value>...][&<general_argument>...]]
```

**Arguments:**

| Argument | Valid values | Description |
|---|---|---|
| `Camera=<String>` | String | Camera name. If this parameter is omitted, the command will return the last database record, regardless of camera. |

**Example 1:** Request the last record of a camera with response in XML

```
http://192.168.0.1:8601/Interface/LPR/GetLastRecordData?Camera=Camera1&
ResponseFormat=XML
```

**Example 2:** Request the last record of a camera with response in text

```
http://192.168.0.1:8601/Interface/LPR/GetLastRecordData?Camera=Camera1&
ResponseFormat=Text
```

**Example 3:** Request the last record of the database with response in XML

```
http://192.168.0.1:8601/Interface/LPR/GetLastRecordData?ResponseFormat=XML
```

**Response:**
If the record is found, a list of parameter-value pairs is returned, otherwise, the error `4000 Record not found` is returned

**HTTP Return:** 200 OK

**Parameters of return:**

| Parameter | Type | Description |
|---|---|---|
| RECORDNUMBER | Integer | Record number |
| DATETIME | APITimestamp | Date and time of the record in timestamp format |
| LPRCONFIGURATION | String | Name of the LPR Configuration |
| CAMERA | String | Camera name |
| LICENSEPLATE | String | Recognized plate characters |
| RELIABILITY | String | Reliability of reading<br><br>Value: No Reliability — No reliability value. The used LPR engine does not provide reliability values<br>Value: Low — Low reliability<br>Value: Medium — Medium reliability<br>Value: High — High reliability |
| HIT | String | This value identifies if the engine believes having<br><br>This value identifies whether the engine believed to have hit or wrong the recognized value<br><br>Value: Error — Error on plate recognition<br>Value: Hit — The engine believes to have hit the value<br>Value: Uncertainty — The engine is not sure to have hit the value |

**Example of return in text:**

```
RESPONSE_CODE=0
RESPONSE_MESSAGE=OK
RECORDNUMBER=40
DATETIME=2009-11-15 15:40:23.540
LPRCONFIGURATION=LPR1
CAMERA=Camera1
LICENSEPLATE=ABC1234
RELIABILITY=Low
HIT=Uncertainty
```

**Example of return in XML:**

```
<?xml version="1.0" encoding="UTF-8" ?>
<Response>
 <Code>0</Code>
 <Message>OK</Message>
 <Data>
  <RecordData>
   <RecordNumber>40</RecordNumber>
   <DateTime>2009-11-15 15:40:23.540</DateTime>
   <LPRConfiguration>LPR1</LPRConfiguration>
   <Camera>Camera1</Camera>
   <LicensePlate>ABC1234</LicensePlate>
   <Reliability>Low</Reliability>
   <Hit>Uncertainty</Hit>
  </RecordData>
 </Data>
</Response>
```

### 4.12.7 Requesting the image of the last LPR record

Request the image of the last LPR record of a camera or database.

**Compatibility:** Standard, Professional, Enterprise

**Security level:** Admin

**Method:** HTTP GET

**Syntax:**

`http://<server_address>/Interface/LPR/GetLastRecordImage[?<argument=value>[&<argument=value>...][&<general_argument>...]]`

**Arguments:**

| Argument | Valid values | Description |
|---|---|---|
| `Camera=<String>` | String | Camera name. If this parameter is omitted, the command will return the last database record, regardless of camera. |

**Example 1:** Request the image of the last record of a camera with error response in XML

```
http://192.168.0.1:8601/Interface/LPR/GetLastRecordImage?Camera=Camera1&
ResponseFormat=XML
```

**Example 2:** Request the image of the last record of a camera with error response in text

```
http://192.168.0.1:8601/Interface/LPR/GetLastRecordImage?Camera=Camera1&
ResponseFormat=Text
```

**Example 3:** Request the image of the last record of database with error response in text

```
http://192.168.0.1:8601/Interface/LPR/GetLastRecordImage?ResponseFormat=XML
```

**Response:**
If the record is found, its JPEG image is returned, otherwise, the error `4000 Record not found` is returned.

**HTTP Return:** 200 OK

**Parameters of return:** Default return of API

### 4.12.8 Triggering the recognition

This command should be used as a kind of external sensor to start or stop the license plate recognition from a LPR Configuration.

**Note:** This command will only work in LPR Configurations set to use the external sensor

**Compatibility:** Standard, Professional, Enterprise

**Security level:** Admin

**Method:** HTTP GET

**Syntax:**

```
http://<server_address>/Interface/LPR/ExternalSensorControl?
<argument=value>[&<argument=value>...][&<general_argument>...]
```

**Arguments:**

| Argument | Valid values | Description |
|---|---|---|
| LPRConfiguration=*<String>** | String | Name of the LPR Configuration |
| Action=*<String>* | Start<br>Stop<br>Instant | Action to be executed<br><br>Operation: Start — Description: Start the recognition<br>Operation: Stop — Description: Stop the recognition<br>Operation: Instant — Description: Instant recognition<br><br>If this parameter is omitted, the default value of "Instant" will be used |

* Mandatory parameters

**Usage:**
When the LPR Configuration is set to use an external sensor, the system will not start the license plate recognition until this API command is called, so this command should be used to start or stop the license plate recognition according to the integration requirements of an external system.

There are two ways to work with an external sensor: Start/Stop and Instant.

Start/Stop: In this type of operation, this trigger command should be called a time to start the recognition and again to stop the recognition. Once started, the system will begin to analyse all the received images in search for license plates and will stop only when this trigger command is called with the parameter `Action=Stop`.

Instant: In this type of operation, after calling this trigger command with parameter `Action=Instant`, the system will analyse the next three received images in search of license plates and will automatically stop, waiting for the next call for analysis.

**Example 1:** Start the license plate recognition from a configuration with response in text

```
http://192.168.0.1:8601/Interface/LPR/ExternalSensorControl?
LPRConfiguration=LPR1&Action=Start&ResponseFormat=Text
```

**Example 2:** Stop the license plate recognition from a configuration with response in XML
```
http://192.168.0.1:8601/Interface/LPR/ExternalSensorControl?
LPRConfiguration=LPR1&Action=Stop&ResponseFormat=XML
```

**Example 3:** Start the instant plate recognition from a configuration with response in XML
```
http://192.168.0.1:8601/Interface/LPR/ExternalSensorControl?
LPRConfiguration=LPR1&Action=Instant&ResponseFormat=XML
```

**Response:**
Default response of API.

**HTTP Return:** 200 OK

**Parameters of return:** Default return of API

### 4.12.9 Monitoring the LPR events

Starts the monitoring of the LPR events from the server. By using this command, you will receive in real time, the characters of recognized license plates from all LPR Configurations.

**Compatibility:** Standard, Professional, Enterprise

**Security level:** Admin

**Method:** HTTP GET

**Syntax:**
```
http://<server_address>/Interface/LPR/MonitorEvents
[?<argument=value>[&<argument=value>...]][&<general_argument>...]
```

**Arguments:**

| Argument | Valid values | Description |
|---|---|---|
| `KeepAliveInterval=<Integer>` | Integer. 1 >= X <= 120 | Specify the time (in seconds) of Keep-Alive packet sending. The Keep-Alive packet is used to control the connection. If this parameter is omitted, the default value of 5 will be used |

**Example 1:** Start the LPR event monitoring with response in XML
```
http://192.168.0.1:8601/Interface/LPR/MonitorEvents?ResponseFormat=XML
```

**Example 2:** Start the LPR event monitoring with response in text and keep-alive interval of 60 seconds
```
http://192.168.0.1:8601/Interface/LPR/MonitorEvents?ResponseFormat=Text&
KeepAliveInterval=60
```

**Example 3:** Start the LPR event monitoring with response in XML and authentication with Admin user
```
http://192.168.0.1:8601/Interface/LPR/MonitorEvents?
ResponseFormat=XML&AuthUser=admin&AuthPass=pass
```

**Response:**
A stream of events data will be sent using HTTP Multipart `x-mixed-replace` transmission.
The response data stream is continuous, so an event of Keep-Alive will be sent every X seconds (Configurable by `KeepAliveInterval` parameter), with this event you can verify if the TCP connection

is still open.
Each event is separated by the multipart boundary `--DigifortBoundary`.

**HTTP Return:** 200 OK

**Parameter of return:**
The return parameters are divided into two subcategories: `EventData` e `ObjectData`.

**EventData parameters:**

| Parameter | Type | Description |
|---|---|---|
| TYPE | String | Event type<br><br>Value: PLATE_RECOGNIZED — Description: Recognized plate event<br>Value: KEEP_ALIVE — Description: HTTP Connection Keep-Alive message. Sent every 5 seconds to keep the TCP / HTTP connection |
| RECORDNUMBER | Integer | LPR record number |
| TIMESTAMP | APITimestamp | Date and time of the event |
| UTCTIMESTAMP | APITimestamp | UTC date and time of the event |
| PLATE | String | Characters of the recognized license plate |

**ObjectData parameters:**

| Parameter | Type | Description |
|---|---|---|
| NAME | String | Name of the LPR Configuration that triggered the event |
| CAMERA | String | Name of the camera associated with the LPR Configuration that triggered the event |

**Example of return in text:**

```
--DigifortBoundary
Content-Type: text/plain; charset=UTF-8
Content-Length: 217

RESPONSE_CODE=0
RESPONSE_MESSAGE=OK
EVENTDATA_TYPE=PLATE_RECOGNIZED
EVENTDATA_TIMESTAMP=2009-11-15 10:55:30
EVENTDATA_UTCTIMESTAMP=2009-11-15 13:55:30
EVENTDATA_PLATE=ABC1234
OBJECTDATA_NAME=Configuration1
OBJECTDATA_CAMERA=Camera1
--DigifortBoundary
Content-Type: text/plain; charset=UTF-8
Content-Length: 45

RESPONSE_CODE=0
RESPONSE_MESSAGE=OK
EVENTDATA_TYPE=KEEP_ALIVE
--DigifortBoundary
..
..
```

**Example of return in XML:**

```
--DigifortBoundary
Content-Type: text/xml; charset=UTF-8
Content-Length: 323

<?xml version="1.0" encoding="UTF-8" ?>
<Response>
  <Code>0</Code>
  <Message>OK</Message>
  <Data>
    <Event>
      <EventData>
        <Type>PLATE_RECOGNIZED</Type>
        <Timestamp>2009-11-15 10:55:30</Timestamp>
        <UTCTimestamp>2009-11-15 13:55:30</UTCTimestamp>
        <Plate>ABC1234</Plate>
      </EventData>
      <ObjectData>
        <Name>Configuration1</Name>
        <Camera>Camera1</Camera>
      </ObjectData>
    </Event>
  </Data>
</Response>
--DigifortBoundary
Content-Type: text/xml; charset=UTF-8
Content-Length: 213

<?xml version="1.0" encoding="UTF-8" ?>
<Response>
  <Code>0</Code>
  <Message>OK</Message>
  <Data>
    <Event>
      <EventData>
        <Type>KEEP_ALIVE</Type>
      </EventData>
    </Event>
  </Data>
</Response>
--DigifortBoundary
..
..
```

## 4.13 Analytics

Commands to control Analytics

### 4.13.1 Searching for Analytics records

Perform a search for Analytics records in the database

**Compatibility:** Standard, Professional, Enterprise

**Security level:** Requires user authentication with rights to search for Analytics records

**Method:** HTTP GET

**Syntax:**

```
http://<server_address>/Interface/Analytics/Search
[?<argument=value>[&<argument=value>...][&<general_argument>...]]
```

**Arguments:**

| Argument | Valid values | Description |
|---|---|---|
| StartDate=<*APIDate*> | Data | Search for records starting with the specified date |
| StartTime=<*APITime*> | Hora | Search for records starting with the specified time |
| EndDate=<*APIDate*> | Data | End date of search. Search for all records inside the range of start and end.<br><br>If this parameter is omitted, the value specified in StartDate parameter will be used |
| EndTime=<*APITime*> | Hora | End time of search. Search for all records inside the range of start and end.<br><br>If this parameter is omitted, the value specified in StartTime parameter will be used |
| Cameras=<*String*> | String | List of cameras to search. The list with camera names must be comma-separated |
| ObjectClasses=<*String*> | String | List of object classes to search. The list of values must be comma-separated |
| Zones=<*String*> | String | List of zones to search. The list of values must be comma-separated |
| EventTypes=<*String*> | String | List of event types to search. The list of values must be comma-separated<br><br>Valid event type values:<br>PRESENCE<br>ENTER<br>EXIT<br>APPEAR<br>DISAPPEAR<br>STOPPED<br>LOITERING<br>DIRECTION<br>SPEED<br>TAILGATING<br>COUNTING_LINE_A<br>COUNTING_LINE_B<br>TAMPERING<br>ABANDONED_OBJECT<br>REMOVED_OBJECT<br>FACE_DETECTION |
| OrderBy=<*String*> | String | Sort the records by the specified column. |

| | | |
|---|---|---|
| | | <table><tr><th>Columns</th></tr><tr><td>RecordCode</td></tr><tr><td>Camera</td></tr><tr><td>StartDate</td></tr><tr><td>EndDate</td></tr><tr><td>Zone</td></tr><tr><td>EventType</td></tr><tr><td>ObjectClass</td></tr></table><br>If this parameter is omitted, the records will be displayed in retrieval order (Faster method with less server memory usage) |

**Example 1:** Search for all records between January 01, 2014 and February 01, 2014

```
http://192.168.0.1:8601/Interface/Analytics/Search?StartDate=2014.01.01&
StartTime=00.00.00.000&EndDate=2014.02.01&EndTime=23.59.59.999
```

**Example 2:** Search for all records of Presence and Direction events, sorting by start date

```
http://192.168.0.1:8601/Interface/Analytics/Search?
EventTypes=PRESENCE,DIRECTION&OrderBy=StartDate
```

**Response:**
A list with all found Analytics records is returned.

**HTTP Return:** 200 OK

**Parameters of return:**

**Fixed Parameter:**

| Parameter | Type | Description |
|---|---|---|
| COUNT | Integer | Total of records |

**Parameters of records list:**

| Parameter | Type | Description |
|---|---|---|
| RECORDCODE | Integer | Database record number |
| CAMERA | String | Camera name |
| STARTDATE | APITimestamp | Record start date and time |
| ENDDATE | APITimestamp | Record end date and time |
| ZONE | String | Name of the zone |
| EVENTTYPE | PRESENCE<br>ENTER<br>EXIT<br>APPEAR<br>DISAPPEAR<br>STOPPED<br>LOITERING<br>DIRECTION<br>SPEED<br>TAILGATING<br>COUNTING_LINE_A | Type of analytics event |

| Parameter | Type | Description |
|---|---|---|
| | COUNTING_LINE_B<br>TAMPERING<br>ABANDONED_OBJECT<br>REMOVED_OBJECT<br>FACE_DETECTION | |
| OBJECTCLASS | String | Class of object |

**List of records:**
The parameters of the list of Analytics records will depend on the type of response (Text or XML).

**List of Analytics records with response in text:**
The parameters of response in text will obey the following syntax:

```
ANALYTICSRECORD_<num>_<field>=<value>
```

| Parameter | Description |
|---|---|
| *num* | Number of the record |
| *field* | Name of the field |
| *value* | Value of the field |

**Example of return in text:**

```
RESPONSE_CODE=0
RESPONSE_MESSAGE=OK
COUNT=2
ANALYTICSRECORD_1_RECORDCODE=98370
ANALYTICSRECORD_1_CAMERA=04
ANALYTICSRECORD_1_STARTDATE=2015-09-27 00:12:21.351
ANALYTICSRECORD_1_ENDDATE=2015-09-27 00:12:21.351
ANALYTICSRECORD_1_ZONE=Entrance
ANALYTICSRECORD_1_EVENTTYPE=PRESENCE
ANALYTICSRECORD_1_OBJECTCLASS=Person
ANALYTICSRECORD_2_RECORDCODE=98371
ANALYTICSRECORD_2_CAMERA=02
ANALYTICSRECORD_2_STARTDATE=2015-09-27 00:25:40.000
ANALYTICSRECORD_2_ENDDATE=2015-09-27 00:25:40.000
ANALYTICSRECORD_2_ZONE=Exit
ANALYTICSRECORD_2_EVENTTYPE=COUNTING_LINE_A
ANALYTICSRECORD_2_OBJECTCLASS=Unclassified
```

**List of Analytics records with response in XML:**
The parameters of response in XML will obey the following syntax:

```
<AnalyticsRecords>
 <Count>COUNT</Count>
 <AnalyticsRecord>
  <RecordCode>RECORD_CODE</RecordCode>
  <Camera>CAMERA_NAME</Camera>
  <StartDate>START_DATE</StartDate>
  <EndDate>END_DATE</EndDate>
  <Zone>ZONE_NAME</Zone>
  <EventType>EVENT_TYPE</EventType>
  <ObjectClass>OBJECT_CLASS</ObjectClass>
 </AnalyticsRecord>
</AnalyticsRecords>
```

**Example of return in XML:**

```
<?xml version="1.0" encoding="UTF-8" ?>
<Response>
 <Code>0</Code>
 <Message>OK</Message>
 <Data>
  <AnalyticsRecords>
   <Count>2</Count>
   <AnalyticsRecord>
    <RecordCode>98370</RecordCode>
    <Camera>04</Camera>
    <StartDate>2015-09-27 00:12:21.351</StartDate>
    <EndDate>2015-09-27 00:12:21.351</EndDate>
    <Zone>Entrance</Zone>
    <EventType>PRESENCE</EventType>
    <ObjectClass>Person</ObjectClass>
   </AnalyticsRecord>
   <AnalyticsRecord>
    <RecordCode>98371</RecordCode>
    <Camera>02</Camera>
    <StartDate>2015-09-27 00:25:40.000</StartDate>
    <EndDate>2015-09-27 00:25:40.000</EndDate>
    <Zone>Exit</Zone>
    <EventType>COUNTING_LINE_A</EventType>
    <ObjectClass>Unclassified</ObjectClass>
   </AnalyticsRecord>
  </AnalyticsRecords>
 </Data>
</Response>
```

## 4.14 RTSP

Commands to control RTSP

### 4.14.1 Requesting the RTSP server settings

Request the RTSP server settings

**Compatibility:** All editions

**Security level:** Admin

**Method:** HTTP GET

**Syntax:**

```
http://<server_address>/Interface/RTSP/GetConfig
[?<general_argument>[&<general_argument>...]]
```

**Example 1:** Request the RTSP server settings with response in XML format

```
http://192.168.0.1:8601/Interface/RTSP/GetConfig?ResponseFormat=XML
```

**Example 2:** Request the RTSP server settings with response in text format and authentication with admin user

```
http://192.168.0.1:8601/Interface/RTSP/GetConfig?
ResponseFormat=Text&AuthUser=admin
```

**Response:**
A list of parameter-value pairs is returned

**HTTP Return:** 200 OK

**Parameters of return:**

| Parameter | Type | Description |
|---|---|---|
| ACTIVATE | Boolean | Server activated or deactivated |
| PORT | Integer | RTSP server port |

**Example of return in text:**

```
RESPONSE_CODE=0
RESPONSE_MESSAGE=OK
ACTIVATE=TRUE
PORT=554
```

**Example of return in XML:**

```
<?xml version="1.0" encoding="UTF-8" ?>
<Response>
  <Code>0</Code>
  <Message>OK</Message>
  <Data>
    <Config>
      <Activate>TRUE</Activate>
      <Port>554</Port>
    </Config>
  </Data>
</Response>
```

## 4.14.2 Requesting the status of RTSP server

Request the status of the RTSP server

**Compatibility:** All editions

**Security level:** Admin

**Method:** HTTP GET

**Syntax:**

```
http://<server_address>/Interface/RTSP/GetStatus
[?<general_argument>[&<general_argument>...]]
```

**Example 1:** Request the status of RTSP server with response in XML format

```
http://192.168.0.1:8601/Interface/RTSP/GetStatus?ResponseFormat=XML
```

**Example 2:** Request the status of RTSP server with response in text format and authentication with admin user

```
http://192.168.0.1:8601/Interface/RTSP/GetStatus?
ResponseFormat=Text&AuthUser=admin
```

**Response:**
A list of parameter-value pairs is returned

**HTTP Return:** 200 OK

**Parameters of return:**

| Parameters | Type | Description |
|---|---|---|
| ACTIVE | Boolean | RTSP server is active or not |
| PORT | Integer | RTSP server port |
| CONNECTIONS | Integer | Total connections with the server |
| USERS | Integer | Total authenticated connections with the server |
| OUTPUTTRAFFIC | Integer | RTSP server output traffic measured in bytes per second |

**Example of return in text:**

```
RESPONSE_CODE=0
RESPONSE_MESSAGE=OK
ACTIVE=TRUE
PORT=554
CONNECTIONS=10
USERS=10
OUTPUTTRAFFIC=1729405
```

**Example of return in XML:**

```
<?xml version="1.0" encoding="UTF-8" ?>
<Response>
  <Code>0</Code>
  <Message>OK</Message>
  <Data>
    <Status>
      <Active>TRUE</Active>
      <Port>554</Port>
      <Connections>10</Connections>
      <Users>10</Users>
      <OutputTraffic>1729405</OutputTraffic>
    </Status>
  </Data>
</Response>
```