API, HTTP - Interface Specification

Revision: 1.1
Date: 2007-September-14

TABLE OF CONTENTS

- DOCUMENT HISTORY
- 1 OVERVIEW
  - 1.1 Product and firmware versions
- 2 REFERENCES
- 3 DEFINITIONS
  - 3.1 General notations
    - 3.1.1 General abbreviations
    - 3.1.2 Style convention
    - 3.1.3 General CGI URL syntax and parameters
    - 3.1.4 Parameter value convention
- 4 INTERFACE SPECIFICATION
  - 4.1 Server responses
    - 4.1.1 HTTP status codes
- 5 API GROUPS
  - 5.1 General
    - 5.1.1 Update and list parameters and their values
      - 5.1.1.1 List parameters
      - 5.1.1.2 List output format
      - 5.1.1.3 Update parameters
    - 5.1.2 Hard factory default
    - 5.1.3 Backup
    - 5.1.4 Restore
    - 5.1.5 Firmware upgrade
    - 5.1.6 Reboot server
  - 5.2 JPEG/MJPEG/MPEG-4
    - 5.2.1 JPEG image request
    - 5.2.2 JPEG image (snapshot) CGI request
    - 5.2.3 JPEG image response
    - 5.2.4 MJPG video request
    - 5.2.5 MJPG video response
    - 5.2.6 MPEG-4 video request
    - 5.2.7 MPEG-4 video response
  - 5.3 PTZ
    - 5.3.1 PTZ set
    - 5.3.2 PTZ configuration
  - 5.4 Motion Detection
    - 5.4.1 Update the Motion Detection parameters
    - 5.4.2 List the Motion Detection parameters

- 5.5 I/O
  - 5.5.1 I/O control
    - 5.5.1.1 Input
    - 5.5.1.2 Output
- 5.6 Audio data transmit

DOCUMENT HISTORY

Version	Date	Comment
1.1	2007-Sep-14	Initial version

1 OVERVIEW

This document specifies the external HTTP-based application programming interface of the camera and video servers with firmware version 0.3.14 and above.

The HTTP-based video interface provides the functionality for requesting single and multi-part images and for getting and setting internal parameter values. The image and CGI-requests are handled by the built-in Web server in the camera and video servers.

1.1 Product and firmware versions

The support for the HTTP API is product and firmware dependent. Please refer to the Release Notes for the actual product for compliance information.

2 REFERENCES

HTTP protocol

- Hypertext Transfer Protocol -- HTTP/1.0

3 DEFINITIONS

This section contains information on general usage of this document.

3.1 General notation

3.1.1 General abbreviations

The following abbreviations are used throughout this document

CGI *Common Gateway Interface* - a standardized method of communication between a client (e.g. a web browser) and a server (e.g. a web server).

TBD *To be done/designed* - signifies that the referenced section/subsection/entity is intended to be specified, but has not reached a level of maturity to be public at this time.

N/A *Not applicable* - a feature/parameter/value is of no use in a specific task

URL RFC 1738 describes the syntax and semantics for a compact string representation for a resource available via the Internet. These strings are called "Uniform Resource Locators" (URLs).

URI A Uniform Resource Identifier (URI) is a compact string of characters for identifying an abstract or physical resource. RFC 2396 describes the generic syntax of URI.

3.1.2 Style convention

In URL syntax and in descriptions of CGI parameters, text in italics within angle brackets denotes content that should be replaced with either a value or a string. When replacing the text string, the angle brackets must also be replaced. An example of this is the description of the name for the server, denoted with `<servername>` in the URL syntax description below, which is replaced with the string `myserver` in the URL syntax example, also shown below.

URL syntax is written with the word "Syntax:" shown in bold face, followed by a box with the referred syntax, as shown below. The name of the server is written as `<servername>`. This is intended to be replaced with the name of the actual server. This can either be a name, e.g. "thecam" or "thecam.adomain.net" or the associated IP number for the server, e.g. `192.168.1.100`.

Syntax:

```
http://<servername>/jpg/image.jpg
```

A description of returned data is written with "Return:" in bold face, followed by the returned data in a box. All data returned as HTTP-formatted, i.e. starting with the string `HTTP`, is line-separated with a Carriage Return and Line Feed (CRLF) printed as `\r\n`.

Return:

```
HTTP/1.0 <HTTP code> <HTTP text>\r\n
```

description and a light grey box with the example.

Example: Request default image.

```
http://myserver/jpg/image.jpg
```

Examples of what can be returned by the server from a request are written with "Example:" in bold face, followed by a short description and a light grey box with an example of the returned data.

Example: Returned data after a successful request.

```
HTTP/1.0 200 Ok\r\n
```

3.1.3 General CGI URL syntax and parameters

CGI URLs are written in lower-case. CGI parameters are written in lower-case and as one word. When the CGI request includes internal camera parameters, the internal parameters must be written exactly as named in the camera or video server. The CGIs are organized in function related directories under the *cgi-bin* directory. The file extension of the CGI is required.

Syntax:

```
http://<servername>/cgi-bin/<subdir>[/<subdir>...]/<cgi>
[?<parameter>=<value>[&<parameter>=<value>...]]
```

Example: List the Network parameters.

```
http://<servername>/cgi-bin/operator/param?action=list&group=Network
```

3.1.4 Parameter value convention

In tables defining CGI parameters and supported parameter values, the default value for optional parameters is system configured.

4 INTERFACE SPECIFICATION

4.1 Server responses

4.1.1 HTTP status codes

The built-in Web server uses the standard HTTP status codes.

Return:

```
HTTP/1.0 <HTTP code> <HTTP text>\r\n
```

with the following HTTP code and meanings

HTTP code	HTTP text	Description
200	OK	The request has succeeded, but an application error can still occur, which will be returned as an application error code.
204	No Content	The server has fulfilled the request, but there is no new information to send back.
302	Moved Temporarily	The server redirects the request to the URI given in the Location header.
400	Bad Request	The request had bad syntax or was impossible to fulfill.
401	Unauthorized	The request requires user authentication or the authorization has been refused.
404	Not Found	The server has not found anything matching the request.
409	Conflict	The request could not be completed due to a conflict with the current state of the resource.
500	Internal Error	The server encountered an unexpected condition that prevented it from fulfilling the request.
503	Service Unavailable	The server is unable to handle the request due to temporary overload.

Example: Request includes invalid file names.

```
HTTP/1.0 404 Not Found\r\n
```

5 API GROUPS

To make it easier for developers to get an idea of which API requests are supported for different products, the requests have been grouped together. Information about which groups are supported can be found in the product-specific release notes document, available for download from the web site.

5.1 General

The requests specified in the General section are supported by all video products with firmware version 4.00 and above.

5.1.1 Update and list parameters and their values

Note:

- These requests have different security levels. The security level for each parameter is specified in the parameter document.
- The URL must follow the standard way of writing a URL, (RFC 2396: Uniform Resource Identifiers (URI) Generic Syntax); that is, spaces and other reserved characters (";", "/", "?", ":", "@", "&", "=", "+", "," and "$") within a <parameter> or a <value> must be replaced with %<ASCII hex>. For example, in the string My camera, the space will have to be replaced with %20, My%20camera.

Method: GET/POST

Syntax:

```
http://<servername>/cgi-bin/view/param?
<parameter>=<value>[&<parameter>=<value>...]
http://<servername>/cgi-bin/operator/param?
<parameter>=<value>[&<parameter>=<value>...]
http://<servername>/cgi-bin/admin/param?
<parameter>=<value>[&<parameter>=<value>...]
```

with the following parameter and values

<parameter>=<value>	**Values**	**Description**
action=<*string*>	add, remove, update or list	Specifies the action to take. Depending on this parameter, various parameters may be set, as described in the following sections.

5.1.1.1 List parameters

Syntax:

```
http://<servername>/cgi-bin/view/param?action=list
[&<parameter>=<value>...]
http://<servername>/cgi-bin/operator/param?action=list
[&<parameter>=<value>...]
http://<servername>/cgi-bin/admin/param?action=list
[&<parameter>=<value>...]
```

with the following parameter and values

<parameter>=<value>	**Values**	**Description**
group=<*string*>[,<*string*>...]	<*group*[.*name*]>[,<*group*[.*name*]>...]	Returns the value of the camera parameter named <*group*>.<*name*>. If <*name*> is omitted, all the parameters of the <*group*> are returned. The camera parameters must be entered exactly as they are named in the camera or video server. Wildcard (*) can be used when listing parameters. See example below. If this parameter is omitted, all parameters in the device are returned.
responseformat	rfc	Get the HTTP response format according to standard. Response format: HTTP/1.0 200 OK\r\n Content-Type: text/plain\r\n \r\n

		<parameter pair>

Example: List the Network parameters.

```
http://myserver/cgi-bin/admin/param?action=list&group=Network
```

5.1.1.2 List output format

```
HTTP/1.0 200 OK\r\n
Content-Type: text/plain\n
\n
<parameter pair>
```

where <*parameter pair*> is

```
<parameter>=<value>\n
[ <parameter pair> ]
```

Example: Network query response.

```
HTTP/1.0 200 OK\r\n
Content-Type: text/plain\n
\n
root.Network.IPAddress=191.168.1.100\n
root.Network.SubnetMask=255.255.255.0\n
```

If the CGI request includes an invalid parameter value, the server returns an error message.
Return:

```
HTTP/1.0 200 OK\r\n
Content-Type: text/plain\n
\n
# Error: <description>\n
```

5.1.1.3 Update parameters

Syntax:

```
http://<servername>/cgi-bin/operator/param?action=update
```

```
[&<parameter>=<value>...]
http://<servername>/cgi-bin/admin/param?action=update
[&<parameter>=<value>...]
```

with the following parameters and values

<parameter>=<value>	**Values**	**Description**
<string>=<string>	*<group.name>=<value>*	Assigns *<value>* to the parameter *<group.name>*. The *<value>* must be URL-encoded when it contains non-alphanumeric characters. The camera parameters must be entered exactly as named in the camera or the video server.

Example: Set the default image resolution to 320x240 pixels.

```
http://myserver/cgi-bin/operator/param?
action=update&Image.I0.Appearance.Resolution=320x240
```

Example: Set the maximum number of viewers to 5.

```
http://myserver/cgi-bin/operator/param?
action=update&Image.MaxViewers=5
```

5.1.2 Hard factory default

Reload factory default. All parameters are set to their factory default value.

Note: This request requires administrator access (administrator authorization).

Method: GET

Syntax:

```
http://<servername>/cgi-bin/admin/hardfactorydefault
```

5.1.3 Backup

Download a unit specific backup of all files in the folder /etc in tar format.

Note: This requires administrator access (administrator authorization).

Method: GET

Syntax:

```
http://<servername>/cgi-bin/admin/backup
```

Return:

```
HTTP/1.0 200 OK\r\n
Content-Type: application/x-tar\r\n
Content-Disposition: attachment; filename=backup\r\n
\r\n
<file content of backup>
```

5.1.4 Restore

Upload a unit specific backup previously created by the backup.

Note: This requires administrator access (administrator authorization).

Method: POST

Syntax:

```
http://<servername>/cgi-bin/admin/restore
```

The file content is provided in the HTTP body according to the format given in RFC 1867. The body is created automatically by the browser if using HTML form with input type "file".

Example: Upload of backup, where "\r\n" has been omitted in the HTTP body.

```
POST /cgi-bin/admin/restore? HTTP/1.0\r\n
Content-Type: multipart/form-data; boundary=123456789\r\n
Content-Length: <content length>\r\n
\r\n
--123456789\r\n
<file content of backup>
\r\n
--123456789--\r\n
```

5.1.5 Firmware upgrade

Upgrade the firmware version.

Note: This requires administrator access (administrator authorization).

Method: POST

Syntax:

```
http://<servername>/cgi-bin/admin/firmwareupgrade[?<parameter>=<value>]
```

with the following parameters and values

<*parameter*>=<*value*>	Values	Description
type=<*string*>	normal, factorydefault	Specifies the type of firmware upgrade. normal = Upgrade and restore old settings. factorydefault = Upgrade and discard all settings. type is by default set to normal.

The file content is provided in the HTTP body according to the format given in RFC 1867. The body is created automatically by the browser if using HTML form with input type "file".

Example:

```
POST /cgi-bin/admin/firmwareupgrade?type=normal HTTP/1.0\r\n
Content-Type: multipart/form-data; boundary=oivazoivaz\r\n
Content-Length: <content length>\r\n
\r\n
--oivazoivaz\r\n
<firmware file content>
\r\n
--oivazoivaz--\r\n
```

5.1.6 Reboot server

Reboot server.

Note: This requires administrator access (administrator authorization).

Method: GET

Syntax:

```
http://<servername>/cgi-bin/admin/reboot
```

5.2 JPEG

The requests specified in the JPEG/MJPG section are supported by those video products that use JPEG and MJPG encoding.

5.2.1 JPEG image request

Returns an image with the default resolution and compression as defined in the system configuration.

Method: GET

Syntax:

```
http://<servername>/jpg/image.jpg
```

Example: Request JPEG image from default camera with default resolution and compression.

```
http://myserver/jpg/image.jpg
```

5.2.2 JPEG image (snapshot) CGI request

Request a JPEG image (snapshot) with specified properties.

Method: GET

Syntax:

```
http://<servername>/cgi-bin/jpg/image
```

product/release-dependent.

Example: Request a JPEG image .

```
http://myserver/cgi-bin/jpg/image.cgi
```

5.2.3 JPEG image response

When a JPEG image is requested, the server returns either the specified JPEG image file or an error.

Return:

```
HTTP/1.0 200 OK\r\n
Content-Type: image/jpeg\r\n
Content-Length: <image size>\r\n
\r\n
<JPEG image data>\r\n
```

Example: Requested JPEG image.

```
HTTP/1.0 200 OK\r\n
Content-Type: image/jpeg\r\n
Content-Length: 15656\r\n
\r\n
<JPEG image data>\r\n
```

5.2.4 MJPG video request

Returns a multipart image stream with the default resolution and compression as defined in the system configuration.

Method: GET

Syntax: Request Multipart JPEG image.

```
http://<servername>/video.mjpg
```

Example: Request JPEG image stream from camera .

```
http://myserver/video.mjpg
```

5.2.5 MJPG video response

When MJPG video is requested, the server returns a continuous flow of JPEG files. The content type is "multipart/x-mixed-replace" and each image ends with a boundary string <*boundary*>. The returned image and HTTP data is equal to the request for a single JPEG image.

Return:

```
HTTP/1.0 200 OK\r\n
Content-Type: multipart/x-mixed-replace;boundary=<boundary>\r\n
\r\n
--<boundary>\r\n
<image>
```

where the proposed <*boundary*> is

```
myboundary
```

and the returned <*image*> field is

```
Content-Type: image/jpeg\r\n
Content-Length: <image size>\r\n
\r\n
<JPEG image data>\r\n
--<boundary>\r\n
<image>
```

Example: Requested JPEG image.

```
HTTP/1.0 200 OK\r\n
Content-Type: multipart/x-mixed-replace;boundary=myboundary\r\n
\r\n
--myboundary\r\n
Content-Type: image/jpeg\r\n
Content-Length: 15656\r\n
\r\n
<JPEG image data>\r\n
--myboundary\r\n
Content-Type: image/jpeg\r\n
Content-Length: 14978\r\n
\r\n
<JPEG image data>\r\n
--myboundary\r\n
Content-Type: image/jpeg\r\n
Content-Length: 15136\r\n
\r\n
<JPEG image data>\r\n
--myboundary\r\n
 .
 .
 .
```

5.2.6 MPEG-4 video request

Returns a multipart image stream with the default resolution and compression as defined in the system configuration.

Method: GET

Syntax: Request Multipart JPEG image.

```
http://<servername>/video.mp4
```

Example: Request JPEG image stream from camera .

```
http://myserver/video.mp4
```

5.2.7 MPEG-4 video response

When MPEG-4 video is requested, the server returns a continuous flow of MPEG-4 video streams. The content type is "multipart/x-mixed-replace" and each image ends with a boundary string <*boundary*>. The returned image and HTTP data is equal to the request for a single JPEG image.

Return:

```
HTTP/1.0 200 OK\r\n
Content-Type: multipart/x-mixed-replace;boundary=<boundary>\r\n
\r\n
--<boundary>\r\n
<image>
```

where the proposed <*boundary*> is

```
myboundary
```

and the returned <*image*> field is

```
Content-Type: image/jpeg\r\n
Content-Length: <image size>\r\n
\r\n
<JPEG image data>\r\n
--<boundary>\r\n
<image>
```

Example: Requested MPEG-4 image.

```
HTTP/1.0 200 OK\r\n
Content-Type: multipart/x-mixed-replace;boundary=myboundary\r\n
\r\n
--myboundary\r\n
Content-Type: image/jpeg\r\n
Content-Length: 15656\r\n
\r\n
<JPEG image data>\r\n
--myboundary\r\n
Content-Type: image/jpeg\r\n
```

```
Content-Length: 14978\r\n
\r\n
<JPEG image data>\r\n
--myboundary\r\n
Content-Type: image/jpeg\r\n
Content-Length: 15136\r\n
\r\n
<JPEG image data>\r\n
--myboundary\r\n
 .
 .
 .
```

5.3 PTZ

The requests specified in the PTZ section are supported by those video products that have support for Pan/Tilt/Zoom devices.

5.3.1 PTZ set

To control the Pan, Tilt and Zoom behavior of a PTZ unit, the following PTZ control URL is used. This URL has view access rights.

Important:
Some PTZ units automatically reduce pan and tilt movements as the zoom factor increases. Therefore, the actual movement may be less than what is requested of these units.
The PTZ control is device-dependent. For information about supported parameters and actual parameter values, please check the specification of the PTZ driver you intend to use. The following table is only an overview.

Note:
The URL must follow the standard way of writing a URL, (RFC 2396: Uniform Resource Identifiers (URI) Generic Syntax); that is, spaces and other reserved characters (";", "/", "?", ":", "@", "&", "=", "+", "," and "$") within a <parameter> or a <value> must be replaced with %<ASCII hex>. For example, in the string My camera, the space will have to be replaced with %20, My%20camera.

Method: GET/POST

Syntax:

```
http://<servername>/cgi-
bin/operator/ptzset?<parameter>=<value>[&<parameter>=<value>...]
```

with the following parameters and values

<parameter>=<value>	**Values**	**Description**

camera=<*int*>	1, ... [1]	Applies only to video servers with more than one video input. Selects the source camera. If omitted, the default camera is used.
whoami=<*string*>	<*any value*>	Returns the name of the system-configured device driver.
center=<*int*>,<*int*> extrem[3]=<*int*>,<*int*>	<x>,<y>	**Absolute:** Used to send the coordinates for the point in the image where the user clicked. This information is then used by the server to calculate the pan/tilt move required to (approximately) center the clicked point. **Relative:** Used to send the coordinates for the point in the image where the user clicked. This information is then used by the server to calculate the direction and number of degrees to move. The number of degrees increases with the distance from the center of the image to the point clicked.
imagewidth=<*int*>	1, ... [1]	Required in conjunction with *center* if the image width displayed is different from the default size of the image, which is product-specific.
imageheight=<*int*>	1, ... [1]	Needed in conjunction with *center* if the image height is different from the default size, which is product-specific.
move=<*string*>	home, up, down, left, right, upleft, upright, downleft, downright	**Absolute:** Moves the device 5 degrees in the specified direction. **Relative:** Moves the device approx. 50-90 degrees[2] in the specified direction. **Note:** home is only valid if

		any home position has been previously set with "home=yes".
pan=<*float*>	-180.0 - 180.0	**Absolute:** Pans the device relative to the (0,0) position. --- **Relative:** n/a
tilt	-180.0 - 180.0	**Absolute:** Tilts the device relative to the (0,0) position. --- **Relative:** n/a
zoom=<*int*>	1 - 9999	**Absolute:** Zooms the device *n* steps. --- **Relative:** n/a
focus=<*int*>	1 - 9999	**Absolute:** Move Focus *n* steps. --- **Relative:** n/a
iris=<*int*>	1 - 9999	**Absolute:** Move iris *n* steps. --- **Relative:** n/a
rpan=<*float*>	-360.0 - 360.0	**Absolute:** Pans the device *n* degrees relative to the current position. --- **Relative:** Pans the device approx. *n* degrees relative to the current position
rtilt=<*float*>	-360.0 - 360.0	**Absolute:** Tilts the device *n* degrees relative to the current position. --- **Relative:** Tilts the device approx. *n* degrees relative to the current position.
rzoom=<*int*>	-9999 - 9999	**Absolute:** Zooms the device *n* steps relative to the current position. Positive values mean zoom in, negative values mean zoom out. ---

		Relative: Zooms the device approx. *n* steps relative to the current position. Positive values mean zoom in, negative values mean zoom out.
rfocus=<*int*>	-9999 - 9999	**Absolute:** Move Focus *n* steps relative to the current position. Positive values mean focus far, negative values mean focus near. **Relative:** Move Focus approx. *n* steps relative to the current position. Positive values mean focus far, negative values mean focus near.
riris=<*int*>	-9999 - 9999	**Absolute:** Move iris *n* steps relative to the current position. Positive values mean open iris, negative values mean close iris. **Relative:** Move iris approx. *n* steps relative to the current position. Positive values mean open iris, negative values mean close iris.
autofocus=<*string*>	on, off	Autofocus On/Off.
autoiris=<*string*>	on, off	Autoiris On/Off.
continuouspantiltmove=<*int*>,<*int*>	-100 - 100,-100 - 100	Continuous pan/tilt motion. Positive values mean right (pan) and up (tilt), negative values mean left (pan) and down (tilt). "0,0" means stop. Values as <*pan speed*>,<*tilt speed*>
continuouszoommove=<*int*>	-100 - 100	Continuous zoom motion. Positive values mean zoom in and negative values mean zoom out. "0" means stop.
continuousfocusmove=<*int*>	-100 - 100	Continuous focus motion.

		Positive values mean focus near and negative values mean focus far. "0" means stop.
continuousirismove=<*int*>	-100 - 100	Continuous iris motion. Positive values mean iris open and negative values mean iris close. "0" means stop.
auxiliary=<*string*>	<*function name*>	Activates/deactivates auxiliary functions of the device where <*function name*> is the name of the device-specific function.
gotoserverpresetname=<*string*>	<*preset name*>[4]	Move to the position associated with the <*preset name*>.
gotoserverpresetno=<*int*>	1, ...	Move to the position associated with the specified preset position number.
gotodevicepreset=<*int*>	<*preset pos*>	Bypasses the presetpos interface and tells the device to go directly to the preset position number <*preset pos*> stored in the device, where the <*preset pos*> is a device-specific preset position number.
bartype=<*string*>	absolute, relative	Used together with barcoord and determines how the bar shall be interpreted. If "absolute", the endpoints of the bar correspond to the current limits. If "relative", the behavior is device-dependent. The default interpretation is "*absolute*" for panbar, tiltbar and zoombar and "*relative*" for focusbar and irisbar.
barcoord=<*int*>,<*int*>	<*x*>,<*y*>	Used in conjunction with panbar, tiltbar, zoombar, focusbar or irisbar, to send coordinates to the server.
panbar=<*int*>,<*string*>	<*length*>,<*alignment*>	<*length*> is the length of the

		bar in pixels, which is needed in order to calculate the center of the bar. <*alignment*> is one of the strings "*horisontal*" or "*vertical*". The alignment string determines if the x (horisontal) or the y (vertical) coordinate from *barcoord* is used, i.e. if the bar is horisontal; use "*horisontal*" and if the bar is vertical; use *"vertical*" as alignment.
tiltbar=<*int*>,<*string*>	<*length*>,<*alignment*>	<*length*> is the length of the bar in pixels, which is needed in order to calculate the center of the bar. <*alignment*> is one of the strings "*horisontal*" or "*vertical*". The alignment string determines if the x (horisontal) or the y (vertical) coordinate from *barcoord* is used, i.e. if the bar is horisontal; use "*horisontal*" and if the bar is vertical; use *"vertical*" as alignment.
zoombar=<*int*>,<*string*>	<*length*>,<*alignment*>	<*length*> is the length of the bar in pixels, which is needed in order to calculate the center of the bar. <*alignment*> is one of the strings "*horisontal*" or "*vertical*". The alignment string determines if the x (horisontal) or the y (vertical) coordinate from *barcoord* is used, i.e. if the bar is

		horisontal; use "*horisontal*" and if the bar is vertical; use *"vertical*" as alignment.
focusbar=<*int*>,<*string*>	<*length*>,<*alignment*>	<*length*> is the length of the bar in pixels, which is needed in order to calculate the center of the bar. <*alignment*> is one of the strings "*horisontal*" or "*vertical*". The alignment string determines if the x (horisontal) or the y (vertical) coordinate from *barcoord* is used, i.e. if the bar is horisontal; use "*horisontal*" and if the bar is vertical; use *"vertical*" as alignment.
irisbar=<*int*>,<*string*>	<*length*>,<*alignment*>	<*length*> is the length of the bar in pixels, which is needed in order to calculate the center of the bar. <*alignment*> is one of the strings "*horisontal*" or "*vertical*". The alignment string determines if the x (horisontal) or the y (vertical) coordinate from *barcoord* is used, i.e. if the bar is horisontal; use "*horisontal*" and if the bar is vertical; use *"vertical*" as alignment.
speed=<*int*>	1 - 100	Sets the head speed of the device that is connected to the specified camera.
imagerotation=<*int*>	0, 90, 180, 270	If PTZ command refers to an image stream that is rotated differently than the current image setup, then the image stream rotation must be added to each command with this

		parameter to allow the server to compensate.
query=<*string*>	speed, position, presetposcam, presetposall	Returns the current parameter values.
info=<*int*>	1	Returns a description of available PTZ commands. No PTZ control is performed.

[1] Product-dependent. Check the product's specification.
[2] Actual values are device driver-specific.
[3] Obsolete.
[4] <*preset name*> is a string with a maximum of 31 characters, ~ is not allowed.

Example: Request information about which PTZ commands are available for camera 1.

```
http://myserver/cgi-bin/operator/ptzset?info=1&camera=1
```

5.3.2 PTZ configuration

Configure PTZ preset positions..

Note: This request requires operator access (operator authorization).

Method: GET/POST

Syntax:

```
http://<servername>/cgi-bin/operator/ptzconfig?
<parameter>=<value>[&<parameter>=<value>...]
```

with the following parameters and values

<***parameter***>=<***value***>	**Values**	**Description**
camera=<*int*>	1, ... [1]	Applies only to video servers with more than one video input. Selects the source camera. If omitted, the default camera is used.
setserverpresetname=<*string*>	<*preset*	Associates the current position to

	name>[1]	<*preset name*> as a preset position in the server.
setserverpresetno=<*int*>	1, ...	Saves the current position as a preset position number in the server.
home=<*string*>	yes	Makes the current position the home position for the camera. Used with *setserverpresetname* or *setserverpresetno.*
removeserverpresetname=<*string*>	<*preset name*>[1]	Removes the specified preset position associated with <*preset name*>.
removeserverpresetno=<*int*>	1, ...	Removes the specified preset position.
setdevicepreset=<*int*>	<*preset pos*>	Bypasses the presetpos interface and tells the device to save its current position as preset position <*preset pos*> directly in the device, where <*preset pos*> is a device-specific preset position number.

[1] <*preset name*> is a string with a maximum of 31 characters, ~ is not allowed.

5.4 Motion Detection

To be able to define Motion Detection parameters, the video product must have built-in Motion Detection.
A motion detection window is defined by several parameters. The motion detection parameters reside within a dynamic parameter group. Accordingly it is possible to list and update the motion detection parameters with param.cgi, The dynamic motion detection parameter groups are divided into sub groups of the main motion parameter group, i.e. Motion.M<group number>.<parameter name>. group number is a unique number which is stated when a new dynamic parameter group is created, i.e. Motion.M3.

5.4.1 Update the Motion Detection parameters

Example: Update the parameters for an existing Motion Detection window.

```
http://myserver/cgi-bin/operator/param?action=update&Motion.M1.Top=1500
&Motion.M1.Bottom=8000
```

5.4.2 List the Motion Detection parameters

Example: List the Motion.M1 and Motion.M2 parameters.

```
http://myserver/cgi-bin/operator/param?
action=list&group=Motion.M1,group=Motion.M2
```

Example: List all Motion Detection windows.

```
http://myserver/cgi-bin/operator/param.cgi?action=list&group=Motion
```

5.5 I/O

The requests specified in the I/O section are supported by those products that have Input/Output connectors.

5.5.1 I/O control

5.5.1.1 Input

Digital Input

Method: GET

Note: This request requires administrator access (administrator authorization).

Syntax:

```
http://<servername>/cgi-bin/admin/input?
<parameter>=<value>[&<parameter>=<value>...]
```

with the following parameters and values

<***parameter***>=<***value***>	**Values**	**Description**
check=<*int*>[,<*int*>, ...]	<*id1*>[,<*id2*>, ...] [1]	Returns the status (1 or 0) of one or more inputs numbered *id1 ,id2, ...*
checkactive=<*int*>[,<*int*>, ...]	<*id1*>[,<*id2*>, ...] [1]	Returns the status (active or inactive) of one or more inputs numbered *id1,id2, ...*
monitor=<*int*>[,<*int*>, ...][2]	<*id1*>[,<*id2*>, ...] [1]	Returns a multipart stream of

		"check" inputs (see return description below).

[1] Number of inputs may differ for different cameras and video servers. See the product's specification.
[2] Support for this parameter is product/release-dependent.

Return: "*monitor*", i.e., multipart "*check*" parameter

```
HTTP/1.0 200 OK\r\n
Content-Type: multipart/x-mixed-replace;boundary=<boundary>\r\n
\r\n
--<boundary>\r\n
<monitor data>
```

where the proposed boundary <*boundary*> is

```
ioboundary
```

and the <*monitor data*> part is

```
Content-Type: text/plain\r\n
\r\n
<check data>
--<boundary>\r\n
```

and <*check data*> is

```
IO<n>:<action char>\r\n
```

and <*n*> is the I/O port number and <*action char*> is the action character described in the table above.
Note: The output can contain extra blank lines, i.e., extra `\r\n` within the sections.

Example: Monitor data on input ports 1, 2, 3, and 4.

```
http://myserver/cgi-bin/input?monitor=1,2,3,4
```

Example: Monitor data on input port 1.

```
HTTP/1.0 200 OK\r\n
Content-Type: multipart/x-mixed-replace; boundary=ioboundary\r\n
\r\n
\r\n
\r\n
\r\n
--ioboundary\r\n
Content-Type: text/plain\r\n
\r\n
IO0:/\n
\r\n
\r\n
--ioboundary\r\n
```

```
Content-Type: text/plain\r\n
\r\n
IO0:H\n
\r\n
--ioboundary\r\n
Content-Type: text/plain\r\n
\r\n
\r\n
IO0:\\n
\r\n
\r\n
--ioboundary\r\n
Content-Type: text/plain\r\n
\r\n
\r\n
\r\n
\r\n
--ioboundary\r\n
Content-Type: text/plain\r\n
\r\n
\r\n
 .
 .
 .
```

5.5.1.2 Output

Digital Output

Method: GET

Note: This request requires administrator access (administrator authorization).

Syntax:

```
http://<servername>/cgi-bin/admin/output?
<parameter>=<value>[&<parameter>=<value>...]
```

with the following parameters and values

<***parameter***>=<***value***>	**Values**	**Description**
check=<*int*>[,<*int*>, ...]	<*id1*>[,<*id2*>, ...] [1]	Returns the status (1 or 0) of one or more outputs numbered *id1 ,id2, ...*
checkactive=<*int*>[,<*int*>, ...]	<*id1*>[,<*id2*>, ...] [1]	Returns the status (active or inactive) of one or more outputs numbered *id1 ,id2, ...*

monitor=<*int*>[,<*int*>, ...] [2]	<*id1*>[,<*id2*>, ...] [1]	Returns a multipart stream of "check" outputs (see return description below).
action=<*string*>	[<*id*>[1]]:<*a*>[<*wait*> <*a*> ...]	Sets the output relay <*id*> active or inactive and waits <*wait*> milliseconds. Note that only one output relay can be activated/deactivated per request. <*id*> = Output number. If omitted, output *1* is selected. <*a*> = Action character: / or \ / = active, \ = inactive. <*wait*> = Delay in milliseconds.

[1] Number of outputs may differ for different cameras and video servers. See the product's specification.
[2] Support for this parameter is product/release-dependent.

Example: Set output 1 active.

```
http://myserver/cgi-bin/admin/output?action=1:/
```

Example: Set two 300 ms pulses with 500 ms delay between the pulses on output 1.

```
http://myserver/cgi-bin/admin/output?action=1:/300\500/300\
```

Example: Wait 1 second before setting output 1 active.

```
http://myserver/cgi-bin/admin/output?action=1:1000/
```

5.6 Audio data transmit

Transmit a Singlepart/Multipart Audio data stream.

Method: POST

Syntax:

```
http://<servername>/cgi-bin/view/transmit
```

There are no valid parameters and values.

Example: Singlepart/Multipart Audio data

```
http://myserver/cgi-bin/view/transmit
```