ABS MegaCam Series Network Cameras

– API, HTTP Interface Document

DOCUMENT HISTORY

Version	Date	Supported Firmware	Release Notes
1.00	2007-May-14	V3.0.0.0	Initial version
1.01	2007-Sep-27		Check un-implement function
1.02	2007-Oct-01		Update ptzconfig.cgi
1.03	2007-Oct-11	V3.0.2.0	Implement I/O - input.cgi and output.cgi
1.04	2007-Dec-03	V3.0.2.1571	Implement videostatus.cgi
1.05	2007-Dec-18	V3.0.2.1631	Implement receive.cgi
			Implement imagesize.cgi
1.06	2007-Dec-28	V3.0.2.1691 and later version	Revise Document

1 OVERVIEW

This document specifies the external HTTP-based application programming interface of the IP cameras and video servers with firmware version 3.00 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 notations

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.
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

URL syntax is written with the word "Syntax:" 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. 10.10.2.139.

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
```

URL syntax examples are written with "Example:" followed by a short 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:" 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. For the POST method, the parameters must be included in the body of the HTTP request. The CGIs are organized in function related directories. The file extension of the CGI is required.

Syntax:

```
http://<servername>/<subdir>[/<subdir>...]/<cgi>.<ext>

[?<parameter>=<value>[&<parameter>=<value>...]]
```

Example: List the Network parameters.

```
http://<servername>/param.cgi?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 Naming conventions and URL syntax

4.1.1 Obsolete CGI parameters

All CGI parameters and values in this specification could be used in all products with V3.00 firmware.

4.2 Server responses

4.2.1 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.

5.1 General

The requests specified in the General section are supported by all video products with firmware version 3.00 and above.

5.1.1 Add, update, remove 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>/param.cgi?<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>/param.cgi?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/param.cgi?action=list&group=Network
```

Example: List the names of all Event parameters.

```
http://myserver/param.cgi?action=list&group=Event.*.Name
```

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

Network.IPAddress=10.13.12.36\n

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>/param.cgi?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/param.cgi?action=update&Image.I0.Resolution=320
x240
```

Example: Set the maximum number of viewers to 5.

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

5.1.1.4 Add parameters

Note: Only applicable for dynamic parameter groups such as the event parameters.

Syntax:

```
http://<servername>/param.cgi?action=add[&<parameter>=<value>..
.]
```

with the following parameters and values

<parameter>=<value>	Values	Description
template=<string>	<template>	Use the specified <template> when creating the new group. The template is a file, which describes all parameters for this group. Depending on the product, different templates are available, please check the Release notes of the firmware version. See examples below.
group=<string>	Event, PTZ.PresetPos, GuardTour, GuardTour.G#.Tour	Specifies the parent group. The parent group defines where in the parameter structure the new group will be created. For example, if adding an event (template=event) and specify group=Event the new group will be available as Event.E<number>.

		Where <number> is the unique number for the group (see return values below). The character before <number> is generated from the last section of the group name. E.g. Event will generate the character E and Event.Notification will generate the character N.
<string>=<string>	<group.name>=<value>	Set a parameter in the newly created group. As the group number is not known before the group is created, the id-number is simply left out, see the examples below. The new group number is created dynamically and can be any number. This is why all parameters are specified to set without any group number. The base path to the parameter is specified as <group>.<uppercase first letter of group>.<parameter name>.
force	yes	The force parameter can be used to exceed limits set for adding dynamic parameter groups. These products can for example be configured for up to 10 event types. The force parameter can be used to exceed this maximum of events.

Example: Create a new event under the group Event and set the name to MyEvent and the Enabled parameter to yes.

```
http://myserver/param.cgi?action=add&group=Event

&template=event&Event.E.Name=MyEvent&Event.E.Enabled=yes
```

Example: A listing of the new group will output the following.

```
Event.E0.Name=MyEvent

Event.E0.Type=T

Event.E0.Enabled=yes

Event.E0.Active=no

Event.E0.Priority=1
Event.E0.Image=1

Event.E0.HWInputs=xxxx

Event.E0.SWInput=

Event.E0.Weekdays=1111111

Event.E0.Starttime=00:00

Event.E0.Duration=0

Event.E0.ImageURLSettingsEnabled=no

Event.E0.ImageURLSettings=

Event.E0.IncludePreTrigger=no

Event.E0.PreTriggerSize=0

Event.E0.PreTriggerInterval=1000

Event.E0.PreTriggerIntervalUnit=s

Event.E0.PreTriggerUnit=s

Event.E0.IncludePostTrigger=no

Event.E0.PostTriggerSize=0

Event.E0.PostTriggerInterval=1000

Event.E0.PostTriggerIntervalUnit=s

Event.E0.PostTriggerUnit=s

Event.E0.IncludeBestEffort=no

Event.E0.BestEffortInterval=1000

Event.E0.BestEffortDuration=0

Event.E0.BestEffortIntervalUnit=s

Event.E0.BestEffortDurationUnit=s
Event.E0.FileName=image.jpg

Event.E0.Suffix=%y-%m-%d_%H-%M-%S-%f

Event.E0.MaxSequenceNumber=-100
```

Note that in this example the id is E0. This can be any number, depending on if other events were added before. Parameters that are not specified in the request will have their default values, as specified in the configuration file.

5.1.1.5 Remove parameters

Note: Only applicable for dynamic parameter groups such as the event parameters.

Syntax:

```
http://<servername>/param.cgi?action=remove[&<parameter>=<value
>...]
```

with the following parameters and values

<parameter>=<value>	Values	Description
group=<string>[,<string>...]	<group>[,<group>]	Deletes the specified <group>

Example: Delete event group E2 and E4.

```
http://myserver/param.cgi?action=remove&group=Event.E2,Event.E4
```

5.1.1.6 Add/Remove server responses

These actions produce one of the following server responses:

Return: A successful add.

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

Content-Type: text/plain\n

\n

<entry> OK\r\n
```

Return: A successful remove.

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

Content-Type: text/plain\n

\n

OK\r\n
```

Return: Add new group failed. The group was not created, due to errors in the parameters to add command.

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

Content-Type: text/plain\n

\n

<additional error information>

# Request failed: <error message>\n
```

Return: Add new group failed. The group was created, but the specified parameters could not be set.

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

Content-Type: text/plain\n

\n

<additional error information>

# Error: <error message>\r\n
```

Return: Remove failed.

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

Content-Type: text/plain\n

\n
```

```
<additional error information>

# Request failed: <error message>\r\n
```

Example: Add new event entry and set the specified name.

```
http://myserver/param.cgi?action=add&group=Event&template=event

&Event.E.Name=MyEvent
```

Response:

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

Content-Type: text/plain\n

\n

E7 OK\r\n
```

Example: Delete an event entry.

```
http://myserver/param.cgi?action=remove&group=Event.E7
```

Response:

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

Content-Type: text/plain\n

\n

OK\r\n
```

5.1.2 Add, modify and delete users

Add a new user with password and group membership, modify the information and remove a user. Note: This request requires root access (root authorization).

Method: GET/POST

Syntax:

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

with the following parameters and values

<parameter>=<value>	Values	Description
action=<string>	add, update, remove or get	add = create a new user account. update = change account information of specified parameters if the account exists. remove = remove an existing account if it exists. get = get a list of the users which belong to each group defined.
user=<string>	<string>	The user account name, a non-existing user name. Valid characters are a thru z, A thru Z and 0 thru 9.
pwd=<string>	<string>	The unencrypted password of the account. Valid characters are a thru z, A thru Z and 0 thru 9.
grp=<string>	<string>	An existing primary group name of the account.
sgrp=<string>:[<string>...]	<string>[,<string>...]	Colon separated existing secondary group names of the account.
comment=<string>	<string>	The comment field of the account.

Example: Create a new administrator account.

```
http://myserver/pwdgrp.cgi?action=add&user=joe&pwd=foo&grp=axus
er

&sgrp=axadmin:axoper:axview&comment=Joe
```

Example: Change the password of an existing account.

```
http://myserver/pwdgrp.cgi?action=update&user=joe&pwd=bar
```

Example: Remove an account.

```
http://myserver/pwdgrp.cgi?action=remove&user=joe
```

Example: List groups and users.

```
http://myserver/pwdgrp.cgi?action=get
```

5.1.3 Factory default

Reload factory default. All parameters except Network.BootProto, Network.IPAddress, Network.SubnetMask, Network.Broadcast and Network.DefaultRouter are set to their factory default values.

Note: This requires administrator access (administrator authorization).

Method: GET

Syntax:

```
http://<servername>/factorydefault.cgi
```

5.1.4 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>/hardfactorydefault.cgi
```

5.1.5 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>/backup.cgi
```

Return:

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

Content-Type: application/x-tar\r\n

Content-Disposition: attachment; filename=backup_<MAC
address>.tar\r\n

\r\n

<file content of backup_<MAC address>.tar>
```

5.1.6 Restore

Upload a unit specific backup previously created by the backup.cgi.

Note: This requires administrator access (administrator authorization).

Method: POST

Syntax:

```
http://<servername>/restore.cgi
```

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 /restore.cgi? HTTP/1.0\r\n

Content-Type: multipart/form-data; boundary=AaBo3x\r\n

Content-Length: <content length>\r\n

\r\n
```

```
--AaBo3x\r\n
Content-Disposition: form-data; name="backup.tar";
filename="backup_<MAC address>.tar"\r\n
Content-Type: application/x-tar\r\n
\r\n
<file content of backup_<MAC address>.tar>
\r\n
--AaBo3x--\r\n
```

5.1.7 Firmware upgrade

Upgrade the firmware version.

Note: This requires administrator access (administrator authorization).

Method: POST

Syntax:

```
http://<servername>/firmwareupgrade.cgi[?<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 /firmwareupgrade.cgi?type=normal HTTP/1.0\r\n

Content-Type: multipart/form-data; boundary=AsCg5y\r\n

Content-Length: <content length>\r\n

\r\n

--AsCg5y\r\n

Content-Disposition: form-data; name="firmware.bin"; 
filename="firmware.bin"\r\n

Content-Type: application/octet-stream\r\n

\r\n

<firmware file content>

\r\n

--AsCg5y--\r\n
```

5.1.8 Restart server

Restart server.

Note: This requires administrator access (administrator authorization).

Method: GET

Syntax:

```
http://<servername>/restart.cgi
```

5.1.10 System logs

Get system log information.

Note: This requires administrator access (administrator authorization).

Note: The response is product/release-dependent.

Method: GET

Syntax:

```
http://<servername>/systemlog.cgi
```

Return:

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

Content-Type: text/plain\r\n

\r\n

<system log information>
```

5.1.11 System date and time

Get or set the system date and time.

Method: GET/POST

Syntax:

```
http://<servername>/date.cgi?<parameter>=<value>
```

with the following parameter and values

<parameter>=<value>	Values	Description
action=<string>	get or set	Specifies what to do. get = get the current date and time. set = set the current date and/or time.

5.1.11.1 Get system date and time

Syntax:

```
http://<servername>/date.cgi?action=get
```

Return:

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

Content-Type: text/plain\r\n

\r\n

<month> <day>, <year> <hour>:<minute>:<second>\r\n

Example:

HTTP/1.0 200 OK\r\n

Content-Type: text/plain\r\n

\r\n

Apr 03, 2003 15:16:04\r\n
```

5.1.11.2 Set system date and time

Syntax:

```
http://<servername>/date.cgi?action=set[&<parameter>=<value>...
]
```

with the following parameters and values

<parameter>=<value>	Values	Description
year=<int>	1970 - 2031	Current year.
month=<int>	1 - 12	Current month.
day=<int>	1 - 31	Current day.
hour=<int>	0 - 23	Current hour.
minute=<int>	0 - 59	Current minute.
second=<int>	0 - 59	Current second.
timezone=<string>	GMT	Specifies the time zone that the new date and/or time is given in. The camera translates the time into local time using whichever time zone has been

		specified through the web configuration. If omitted the new date and/or time is assumed to be in local time. Note: Requires that daylight saving time (DST) is turned off, and that the time mode of the camera is not to synchronize with an NTP server or with the computer time. Currently only GMT is considered valid input. The rest of the time zones are subject to future expansion.

The set action produces one of the following server responses:

Return: A successful set.

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

Content-Type: text/plain\r\n

\r\n

OK\r\n
```

Return: A failed set. Settings or syntax are probably incorrect.

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

Content-Type: text/plain\r\n

\r\n

Request failed: <error message>\r\n
```

Example: Set the date.

```
http://myserver/date.cgi?action=set&year=2005&month=4&day=3
```

Response:

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

```
Content-Type: text/plain\r\n

\r\n

OK\r\n
```

5.2 Image and Video

5.2.1 Image size

Get the actual image size of default image settings, or with given parameters.

Method: GET/POST

Syntax:

```
http://<servername>/imagesize.cgi?<parameter>=<value>[&<paramet
er>=<value>...]
```

with the following parameters and values

<parameter>=<value>	Values	Description
camera=<string>	1, 2, 3, 4 or quad	Returns image size of the camera. Note: This parameter is required.

Example: Request image size of default settings from camera 1.

```
http://myserver/imagesize.cgi?camera=1
```

Example: Returned data after a successful request.

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

Content-Type: text/plain\r\n

\r\n

image width = 176\n

image height = 144\n
```

Example: Request image size with supplied parameters for camera 1.

```
http://myserver/imagesize.cgi?camera=1&resolution=QCIF&compress
ion=60
```

Example: Returned data after a successful request.

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

Content-Type: text/plain\r\n

\r\n

image width = 160\n

image height = 120\n
```

5.2.2 Video status

Check the status of one or more video sources.

Method: GET

Syntax:

```
http://<servername>/videostatus.cgi?<parameter>=<value>
```

with the following parameter and values

<parameter>=<value>	Values	Description
status=<int>[[,<int>],...]	1 ... 41	Returns the status of one or more inputs.

Example: Request video status from input 1, 2, 3 and 4.

```
http://myserver/videostatus.cgi?status=1,2,3,4
```

Example: Returned data after a successful request.

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

Content-Type: text/plain\r\n
```

```
\r\n

Video 1 = video

Video 2 = no video

Video 3 = no video

Video 4 = video
```

5.2.4 JPEG/MJPG

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

5.2.4.1 JPEG image request

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

Method: GET

Syntax:

```
http://<servername>/jpg[/<camera>]/image.jpg
```

with the following parameter

Parameter	Values	Description
<camera>	1, 2, 3, 4 or quad Select	input source. Applies only to servers with more than one input source. Default: default camera.

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

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

Example: Request JPEG image from camera number 2 with default resolution and compression.

```
http://myserver/jpg/2/image.jpg
```

Example: Request JPEG image containing four video sources.

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

5.2.4.2 JPEG image (snapshot) CGI request

Request a JPEG image (snapshot) with specified properties.

Method: GET

Syntax:

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

with the following parameters and values

<parameter>=<value>	Values	Description
camera=<string>	1, 2, 3, 4 or quad	Applies only to video servers with more than one video input. Selects the source camera.

Example: Request a JPEG image from camera 1

```
http://myserver/image.cgi?camera=1
```

5.2.4.3 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
```

5.2.4.5 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>/mjpg[/<camera>]/video.mjpg
```

with the following parameter

Parameter	Values	Description
<camera>	1, 2, 3, 4 or quad	Select input source. Applies only to servers with more than one input source. Default: default camera.

Example: Request JPEG image stream from the 2nd camera with default resolution and compression.

```
http://myserver/mjpg/2/video.mjpg
```

Example: Request quad composed JPEG image stream with default resolution and compression.

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

5.2.4.6 MJPG video CGI request

Request a Multipart-JPEG image stream (video) with specified properties.

Method: GET

Syntax:

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

[?<parameter>=<value>[&<parameter>=<value>...]]
```

with the following parameters and values

<parameter>=<value>	Values	Description
camera=<string>	1, 2, 3, 4 or quad	Applies only to video servers with more than one video input. Selects the source camera.

Example: Request a Multipart JPEG image stream from camera 1

```
http://myserver/mjpg/video.cgi?camera=1
```

5.2.4.7 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 Multipart 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

 .

 .

 .
```

5.3 PTZ

The requests specified in the PTZ section are supported by those video products that have support for Pan/Tilt/Zoom devices. Note! Installing a PTZ driver is done in three main steps: 1. Upload the driver 1. Associate the driver with a serial port 1. Connect cameras to the serial port

PTZ driver update can accomplish step 1 and 2 in one request. PTZ administration can accomplish step 2 if the driver is already present in the product. Open serial port accomplishes step 3.

5.3.3 PTZ control

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>/com/ptz.cgi?<parameter>=<value>[&<parameter
>=<value>...]
```

with the following parameters and values

<parameter>=<value>	Values	Description
camera=<int>	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> extrem3=<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. Digital: Used to send the coordinates for the point in the image where the user clicked. This information is then

		used by the server to center the clicked point.
areazoom=<int>,<int>,<int>	<x>,<y>,<z>	Absolute: Centers on positions x,y (like the center command) and zooms by a factor of z/100. If z is more than 100 the image is zoomed in (for example; z=300 zooms in to 1/3 of the current field of view). If z is less than 100 the image is zoomed out (for example; z=50 zooms out to twice the current field of view). Relative: n/a Note 1: In some camera models, the precision of areazoom can be strongly improved by calibrating the lens offset parameters.

Note 2: The HTTP API for area zoom is currently only supported by PTZ and Dome cameras.

imagewidth=<int>	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, ...	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 irection. Relative: Moves the device approx. 50-90 degrees2 in the specified direction. Digital: Moves the image 25% of the image field width 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 Digital: Pans the device relative to the (0,0) position.
tilt	-180.0 - 180.0	Absolute: Tilts the device relative to the (0,0) position. Relative: n/a

		Digital: Tilts the device relative to the (0,0) position.
zoom=<int>	1 - 9999	Absolute: Zooms the device n steps. Relative: n/a Digital: Zooms the device n steps.
focus=<int>	1 - 9999	Absolute: Move Focus n steps. Relative: n/a Digital: n/a
iris=<int>	1 - 9999	Absolute: Move iris n steps. Relative: n/a Digital: 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. Digital: Pans the device 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. Digital: Tilts the device 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. Digital: Zooms the device 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. Digital: n/a
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. Digital: n/a
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. Digital: n/a
continuousirismove=<int>	-100 - 100	Continuous iris motion. Positive values mean iris open and negative values mean iris close. "0" means stop. Digital: n/a
auxiliary=<string>	<function name>	Activates/deactivates auxiliary functions of the device where <function name> is the name of the device-specific function. Digital: n/a
gotoserverpresetname=<string>	<preset name>	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. Digital: n/a
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. Digital: n/a (always "absolute").
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. Digital: n/a
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. Digital: n/a
speed=<int>	1 - 100	Sets the head speed of the device that is connected to the specified camera. Digital: n/a
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.

Example: Request information about which PTZ commands are available for camera 3.

```
http://myserver/com/ptz.cgi?info=1&camera=3
```

5.3.4 PTZ configuration

Configure PTZ preset positions. On Screen Display (OSD) control.

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

Method: GET/POST

Syntax:

```
http://<servername>/com/ptzconfig.cgi?

<parameter>=<value>[&<parameter>=<value>...]
```

with the following parameters and values

<parameter>=<value>	Values	Description
camera=<int>	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 name>	Associates the current position to <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>	Removes the specified preset position associated with <preset name>.
removeserverpresetno=<int>	1, ...	Removes the specified preset position.

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 add, remove, 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 Add a Motion Detection window

When adding a Motion Detection window, the template file motion is used. The group number should be excluded when adding a new Motion Detection window with specified values since the group number will be defined when the new dynamic group is created.

Example: Add a new Motion Detection window with default values.

```
http://myserver/param.cgi?action=add&group=Motion&template=moti
on
```

Example: Add a new Motion Detection window with specified values.

```
http://myserver/param.cgi?action=add&group=Motion&template=moti
on
```

```
&Motion.M.Name=Entrance&Motion.M.Top=500&Motion.M.Bottom=7000&M
otion.M.Left=5000

&Motion.M.Right=8500
```

Return:

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

Content-Type: text/plain\r\n

\r\n

M<group number> OK\r\n
```

5.4.2 Remove a Motion Detection window

Example: Remove Motion Detection window defined within Motion.M3 and Motion.M5.

```
http://myserver/param.cgi?action=remove&group=Motion.M3,group=M
otion.M5
```

5.4.3 Update the Motion Detection parameters

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

```
http://myserver/param.cgi?action=update&Motion.M1.Top=1500

&Motion.M1.Bottom=8000
```

5.4.4 List the Motion detection parameters

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

```
http://myserver/param.cgi?action=list&group=Motion.M1,group=Mot
ion.M2
```

Example: List all Motion Detection windows.

```
http://myserver/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

Input

Method: GET

Syntax:

```
http://<servername>/io/input.cgi?<parameter>=<value>[&<paramete
r>=<value>...]
```

with the following parameters and values

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

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>:<char>\r\n

and <n> is the I/O port number and <char> is / or H when the port
is active and \ or L when the port is inactive.

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/io/input.cgi?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

Output

Method: GET

Syntax:

```
http://<servername>/io/output.cgi?<parameter>=<value>[&<paramet
er>=<value>...]
```

with the following parameters and values

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

Example: Set output 1 active.

```
http://myserver/io/output.cgi?action=1:/
```

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

```
http://myserver/io/output.cgi?action=1:/300\500/300\
```

Example: Wait 1 second before setting output 1 active.

```
http://myserver/io/output.cgi?action=1:1000/
```

5.7 IP filter

The requests specified in the IP filter section are supported by products that support IP address filtering.

5.7.1 IP filter administration

Allow or deny the listed IP addresses to access the device.

Method: GET

Syntax:

```
http://<servername>/ipfilter.cgi?<parameter>=<value>[&<paramete
r>=<value>... ]
```

with the following parameters and values

<parameter>=<value>	Values	Description

action=<string>	add, remove, removeall, update or list	Specifies the action to take. add = Add new IP address (or addresses). remove = remove an entry in the IP address list. removeall = Remove all IP addresses. The IP address filtering function will automatically be disabled. update = Update settings for the IP address filtering function. list = List the settings for the IP address filtering function. ipaddress=<string>[%20<string>...] <IP addresses> The addresses allowed passing through the filter. A space separated list of IP addresses and network addresses in the CIDR notation (IP address/netmask bits). Note: If accessing the device via a proxy server, the proxy server's IP address must be added to the list of allowed addresses.
enable=<string>	yes,no	Enable/disable the IP address filtering function.
policy=<string>	allow,deny	Allow or deny access for the addresses in the list. If omitting this parameter the policy will be allow by default. Note: The policy used will apply for all the addresses in the list!
verify=<string>	yes,no	Verify that you are able to access the device with the new settings. A failed verification will make the settings to remain as before. verify is automatically set to yes when using requests that risk to make the device inaccessible, i.e. setting enable to yes, or adding, updating and removing IP addresses when the filter function is enabled. In the same way verify is by default set to no when using requests that do not risk to make the device inaccessible, i.e. listing filter settings or adding, updating and removing IP addresses when the filter function is disabled. This behaviour can be changed by setting this parameter in the request.

		yes = Use verification. no = Do not use verification. Note: setting verify to no, may cause the device to be inaccessible.

Example: Add a list of IP addresses and enable the IP address filtering function. Verification that the device is still accessible will automatically be done.

```
http://myserver/ipfilter.cgi?action=add

&ipaddress=10.13.10.12%2010.13.17.0/24&enable=yes
```

Example: List settings for the IP address filtering function.

```
http://myserver/ipfilter.cgi?action=list
```

Example: Remove an entry in the list of addresses. Verification will automatically be done if the IP filter function is enabled.

```
http://myserver/ipfilter.cgi?action=remove&ipaddress=10.13.10.1
2
```

Example: Add 10.13.10.12 to the list of addresses which will be allowed access to the device.

```
http://myserver/ipfilter.cgi?action=add&ipaddress=10.13.10.12

&policy=allow&enable=yes
```

Example: Add 10.13.10.12 to the list of addresses which will be denied access to the device.

```
http://myserver/ipfilter.cgi?action=add&ipaddress=10.13.10.12

&policy=deny&enable=yes
```

Example: Remove all IP addresses and automatically disable the IP address filtering function

```
http://myserver/ipfilter.cgi?action=removeall
```

5.7.2 Server responses

Return: A successful add, remove, removeall or update.

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

Content-Type: text/plain\r\n

\r\n

OK\r\n
```

Return: A list.

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

Content-Type: text/plain\r\n

\r\n

Accept addresses: <IP addresses>\r\n

Enabled: <yes/no>\r\n
```

Return: Verification failed. The settings did not take place.

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

Content-Type: text/plain\r\n

\r\n

Verification failed: IP address "<IP address>" is not an accepted
address\r\n
```

Example: List settings for the IP address filtering function, set verify to yes to check that my computers IP address is accepted.

```
http://myserver/ipfilter.cgi?action=list&verify=yes
```

Response: Only the IP address 10.13.10.12 is an accepted address. The computer 10.13.17.245 will not be able to access the device if enabling the IP address filtering function.

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

Content-Type: text/plain\r\n

\r\n

Accept addresses: 10.13.10.12\r\n

Enabled: no\r\n

Verification failed: IP address "10.13.17.245" is not an accepted
address
```

5.8 Audio

The requests specified in the Audio section are supported by products that have audio capability.

5.8.1 Audio MIME types

Supported MIME types for audio

audio/basic	which is G.711 ??law 64kbit/s
audio/32KADPCM	which is G.726 32kbit/s
audio/G723	which is G.726 24kbit/s

5.8.2 Audio data request

Request an audio stream.

Method: GET

Syntax:

```
http://<servername>/audio/receive.cgi[?<parameter>=<value>]
```

with the following parameters and values

<parameter>=<value>	Values	Description
httptype=<string>	singlepart,multipart	Choose streaming method. Default value is defined by the parameter Audiod.HttpMessageType

Example: Request a singlepart audio stream

```
http://myserver/audio/receive.cgi?httptype=singlepart
```

5.8.3 Singlepart audio data response

When an audio stream is requested/transmitted, the server returns/receives a continuous flow of audio packets. The content type is only set at the beginning of the connection. When the connection is up and running the audio packets will come right after another without any extra information between the packets. The message body contains a block of binary data. Each block of coded audio data is 240 byte.

Return:

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

Content-Type: <audio MIME>\r\n

\r\n

<Audio data>
```

Example: Singlepart Audio data encoded with G.711 ??law.

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

Content-Type: audio/basic\r\n

\r\n

<Audio data>

<Audio data>

<Audio data>

.
```

```
.

.
```

5.8.4 Multipart audio data response

When an audio stream is requested/transmitted, the server returns/receives a continuous flow of audio packets. The content type is "mutipart/x-mixed-replace" and each audio packet ends with a boundary string. The message body contains a block of binary data. Each block of coded audio data is 240 byte.

Return:

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

Content-Type: multipart/x-mixed-replace;
boundary=--<boundary>\r\n

\r\n

--<boundary>\r\n

<audio>

where the proposed <boundary> is:

myboundary

and the <audio> field is

Content-Type: <audio MIME>\r\n

\r\n

<Audio data>\r\n

--<boundary>\r\n

<audio>
```

Example: Multipart Audio data encoded with G.726 32kbit/s (G.721).

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

```
Content-Type: multipart/x-mixed-replace;boundary=myboundary\r\n

\r\n

--myboundary\r\n

Content-Type: audio/32KADPCM\r\n

\r\n

<Audio data>\r\n

--myboundary\r\n

Content-Type: audio/32KADPCM\r\n

\r\n

<Audio data>\r\n

--myboundary\r\n

Content-Type: audio/32KADPCM\r\n

<Audio data>\r\n

--myboundary\r\n

Content-Type: audio/32KADPCM\r\n

\r\n

<Audio data>\r\n

--myboundary\r\n

.

.

.
```

5.8.5 Audio data transmit

Transmit a Singlepart/Multipart Audio data stream. Each block of coded audio data is 240 byte.

Method: POST

Syntax:

```
http://<servername>/audio/transmit.cgi

There are no valid parameters and values.

Example 1: Singlepart audio data transmit with G.711 ??law
(authorization omitted)

POST /audio/transmit.cgi HTTP/1.0\r\n

Content-Type: audio/basic\r\n

Content-Length: 9999999\r\n

Connection: Keep-Alive\r\n

Cache-Control: no-cache\r\n

\r\n

<Audio data>

<Audio data>

<Audio data>
```

Example 2: Multipart audio data transmit with G.711 ??law (authorization omitted)

```
POST /audio/transmit.cgi HTTP/1.0\r\n

Content-Type: multipart/x-mixed-replace;
boundary=--myboundary\r\n

Content-Length: 9999999\r\n

Connection: Keep-Alive\r\n
```

```
Cache-Control: no-cache\r\n

\r\n

--myboundary\r\n

Content-Type: audio/basic\r\n

\r\n

<Audio data>\r\n

--myboundary\r\n

Content-Type: audio/basic\r\n

\r\n

<Audio data>

--myboundary\r\n

Content-Type: audio/basic\r\n

\r\n

<Audio data>

--myboundary\r\n

Content-Type: audio/basic\r\n

\r\n
```