Scripting Guide

Scripting in Axis Network Cameras and Video Servers

This document is intended as a guide for application developers and describes how to use scripting in Axis Network Cameras and Video Servers (in this guide called ‘product’). The reader is presumed to have prior knowledge of shell scripting, and also of Linux and Linux systems in general.

Here, you will find numerous examples of scripts for the most common tasks. These scripts can be used for your own purposes, after making the modifications required by your particular product and application.

This guide applies to products with firmware version 4.40 or higher.

Important!

Axis Communications AB provides no guarantee that any of the examples shown in this document will work for any particular application or that the descriptions will be valid for future firmware or product versions.

Axis Communications AB cannot and will not be held liable for any damage inflicted to any product as a result of the examples or instructions mentioned in this document.

Axis Communications AB reserves the right to make changes to this document and to product specifications without prior notice.

Please bear in mind that the flash chip has a maximum lifespan of about 100,000 writes. For this reason, writing temporary files to the flash memory should be avoided; instead, use the RAM disk mounted on `/tmp`.

AXIS COMMUNICATIONS

www.axis.com

Axis Communications AB provides NO support for application development of any kind. The information here is provided "as is", and there is no guarantee that any of the examples shown will work in your particular application.

Table of contents

Axis Communications AB provides NO support for application development of any kind. The information here is provided "as is", and there is no guarantee that any of the examples shown will work in your particular application.

Axis Communications AB provides NO support for application development of any kind. The information here is provided "as is", and there is no guarantee that any of the examples shown will work in your particular application.

Axis Communications AB provides NO support for application development of any kind. The information here is provided "as is", and there is no guarantee that any of the examples shown will work in your particular application.

Why use scripts?

Scripting is a quick and very useful method to increase the functionality of your product, functionality that is not already available through the Web interface or AXIS VAPIX.

Some examples:

- Let your PTZ camera patrol between preset positions and regularly upload images. This is quite a common task for Axis' PTZ cameras. However, some cameras do not have this functionality built-in since they are not built for continuous movements. But if you are aware of this fact - and do not move the camera too frequently - you can add a script to handle the task.
- Trigger the output connector if the network connection is lost.
- Create your own Web interface in the product.
- Combine different triggers for an event. For example, combine both motion detection and an active input connector.
- Upload images via FTP or e-mail.
- … and much more.

When scripting is not enough

For some tasks, however, scripting languages are too slow as well as inadequate. For example, do not use scripting if you want to:

- Follow a person with a PTZ camera.
- Use customized motion detection.
- Add a custom serial port protocol.

To manage these tasks, you have to write, compile and embed a C-application.

For more information on embedded applications, see the Axis Application Development Partner (ADP) Program at `http://www.axis.com`. The ADP Program assists partners to fully integrate Axis network cameras and video products in end-user solutions by providing technical information, development support and application components.

Before you begin...

Ensure that you have a sufficient amount of memory free in the product; every script you add will use up some memory. Also, do not forget that for every software upgrade, there will most likely be less memory free for custom scripts since we add new functionality for every new version.

Read the datasheet

Start by looking at the datasheet to see how much RAM and flash memory your product is equipped with. Note that there may be a difference between models in the same series. See Axis' Product Interface Guide at `http://www.axis.com`.

Backup your scripts

When you upgrade the firmware, your `user.task.list` will be saved and restored, but your custom scripts will be erased. For this reason, ensure that you have a backup of the scripts.

User name and password are not encrypted

User name and password are not encrypted, but are sent in clear text over the network when using FTP, `telnet,` and as arguments in certain applications.

Check how much memory is available

To check how much memory that is presently free:

1. Open a `telnet` session to your product (see the `telnet` section).
2. Run the `df` command (to see how much flash memory is available).
3. Then run the `free` command (to see how much RAM memory is available).

`df` displays the amount of disk space available on the filesystem. Disk space is shown in 1K blocks by default.

Example:

```
[root@axis-00408c7ccb28 /]1074# df

Filesystem   1k-blocks    Used Available Use% Mounted on

/dev/flash3      13336   13336         0 100% /

/dev/flash2       2048     400      1648  20% /mnt/flash

tmpfs            15520     248     15272   2% /var

tmpfs             9216       0      9216   0% /var/cache/recorder
```

`free` displays the total amount of free and used physical and swap memory in the system, as well as the buffers used by the kernel.

Axis Communications AB provides NO support for application development of any kind. The information here is provided "as is", and there is no guarantee that any of the examples shown will work in your particular application.

Example:

```
[root@axis-00408c7ccb28 /]1089# free
         total       used       free     shared    buffers
  Mem:   31040      15752      15288          0       1656
 Swap:       0          0          0
Total:   31040      15752      15288
```

Example settings

In this document, we use these example parameters:

```
IP address     192.168.0.90
Login          root
Password       password
```

Set up your product

In this guide, we use certain directories for custom scripting and custom Web pages. To be able to follow the examples, we recommend that you create the same directories in your product.

Using a `telnet` client, connect to the product:

```
[claesl@juniperus ~]$ telnet 192.168.0.90
Trying 192.168.0.90...
Connected to 192.168.0.90 (192.168.0.90).
Escape character is '^]'.
4.40
Linux 2.6.17 on a cris (11:49:56)
axis-00408c7ccb28 login: root
Password:
```

Note

The password will not be echoed to the screen.

Change to the `/usr/html/local` directory and create a directory called `scripts`

```
cd /usr/html/local
mkdir scripts
```

Close the connection:

```
exit
Connection closed by foreign host.
```

Axis Communications AB provides NO support for application development of any kind. The information here is provided "as is", and there is no guarantee that any of the examples shown will work in your particular application.

```
[claesl@juniperus ~]$
```

Note

If `telnet` has not been enabled in the product you will get the error message `Connection refused` (Linux) or `Connect failed` (Windows). See the `telnet` section for information on how to enable `telnet`.

Entering scripts

Most products have a built-in editor, `editcgi` that makes working with scripts much easier.

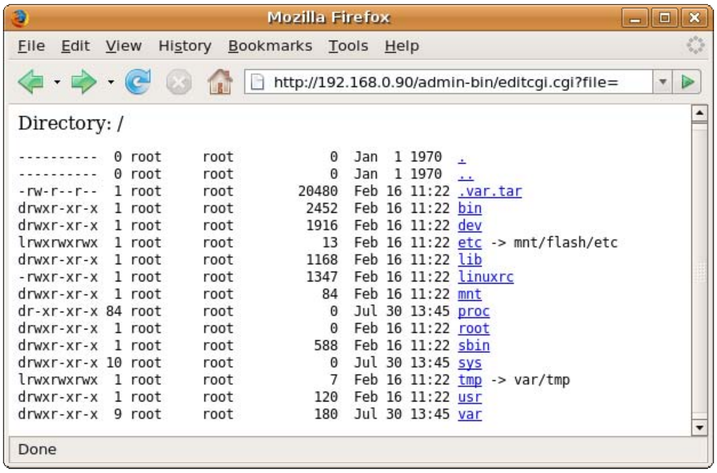
With this editor you can browse the filesystem, look at and modify scripts and HTML code. You can also change file permissions and create new files (an easy way to do this is to open an existing file and save it under a new name).

However, you cannot use the built-in editor to create new directories, i.e. the path to the file you want to create must exist. To create a new directory, use `FTP` or `telnet` instead.

Using editcgi - the built-in editor

To start the built-in editor, either open it through the Web interface of the product (`http://192.168.0.90 | Setup | System Options | Advanced | Scripting` and click `Open Script Editor`) or browse to

```
http://192.168.0.90/admin-bin/editcgi.cgi?file=
```


Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://192.168.0.90/admin-bin/editcgi.cgi?file=
Done

Directory: /

```
----------  0 root     root           0  Jan  1 1970  .
----------  0 root     root           0  Jan  1 1970  ..
-rw-r--r--  1 root     root       20480  Feb 16 11:22 .var.tar
drwxr-xr-x  1 root     root        2452  Feb 16 11:22 bin
drwxr-xr-x  1 root     root        1916  Feb 16 11:22 dev
lrwxrwxrwx  1 root     root          13  Feb 16 11:22 etc -> mnt/flash/etc
drwxr-xr-x  1 root     root        1168  Feb 16 11:22 lib
-rwxr-xr-x  1 root     root        1347  Feb 16 11:22 linuxrc
drwxr-xr-x  1 root     root          84  Feb 16 11:22 mnt
dr-xr-xr-x 84 root     root           0  Jul 30 13:45 proc
drwxr-xr-x  1 root     root           0  Feb 16 11:22 root
drwxr-xr-x  1 root     root         588  Feb 16 11:22 sbin
drwxr-xr-x 10 root     root           0  Jul 30 13:45 sys
lrwxrwxrwx  1 root     root           7  Feb 16 11:22 tmp -> var/tmp
drwxr-xr-x  1 root     root         120  Feb 16 11:22 usr
drwxr-xr-x  9 root     root         180  Jul 30 13:45 var
```

Open the file you wish to edit by clicking on it. Example:
Click on

```
etc
```

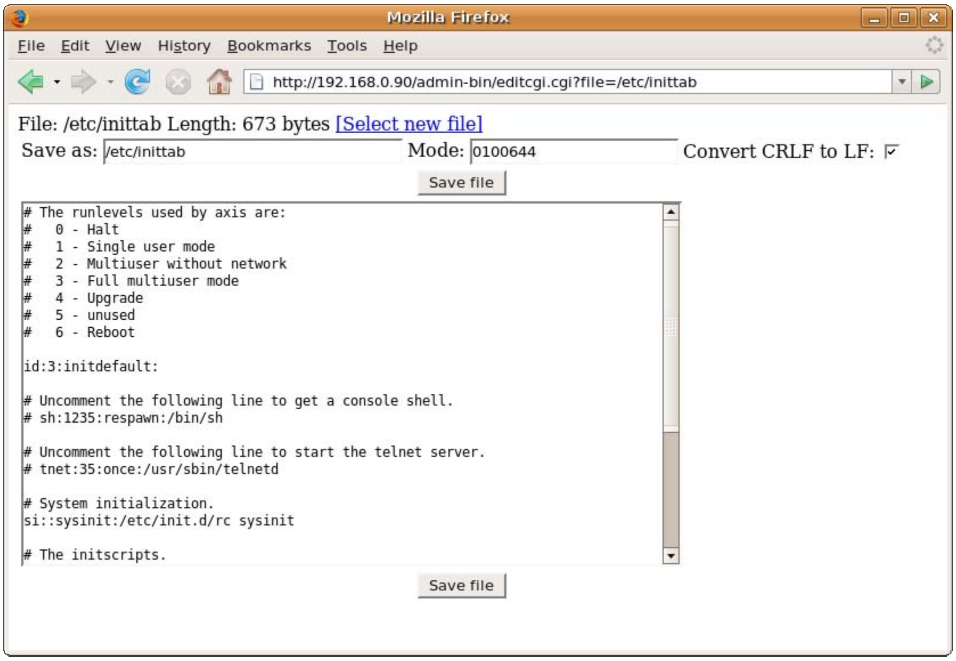
to change to the `etc` directory and then click on

```
inittab
```

to edit the file `/etc/inittab`.

You can also go straight to the file you want to open by entering its path/filename:

```
http://192.168.0.90/admin-bin/editcgi.cgi?file=/etc/inittab
```



http://192.168.0.90/admin-bin/editcgi.cgi?file=/etc/inittab

File: /etc/inittab Length: 673 bytes [Select new file]

Save as: /etc/inittab Mode: 0100644 Convert CRLF to LF: ☑

```
# The runlevels used by axis are:
#   0 - Halt
#   1 - Single user mode
#   2 - Multiuser without network
#   3 - Full multiuser mode
#   4 - Upgrade
#   5 - unused
#   6 - Reboot

id:3:initdefault:

# Uncomment the following line to get a console shell.
# sh:1235:respawn:/bin/sh

# Uncomment the following line to start the telnet server.
# tnet:35:once:/usr/sbin/telnetd

# System initialization.
si::sysinit:/etc/init.d/rc sysinit

# The initscripts.
```

How to convert line breaks

Tick the `Convert CRLF to LF` option and Windows' CRLF will automatically be converted into proper Linux line breaks, i.e. a sole LF.

Do not use a word processor to enter scripts

Word processors insert characters that can be misinterpreted in a script, for instance special quotes and hyphens.

How to set correct file mode/permissions

With `editcgi` you also set the file mode. A script should normally be executable, read and write enabled (file mode `0100755`, corresponding to `-rwxr-xr-x`), while a configuration file like `inittab` does not have to be executable, just read and write enabled (file mode `0100644`, corresponding to `-rw-r--r--`).

How to create new files

You can also use the editor to create new files.

Open any file which has the file mode you want, for example one which is executable, writable and readable. To minimize the risk of inadvertently destroying the original file, first change the file name (and the path, if needed) – then edit, and finally save the file.

Note

The flash memory has a limited lifespan, estimated by the manufacturer to a maximum of 100,000 writes. If possible, use the RAM memory mounted on `/tmp` instead.

telnet

`telnet` is a very powerful tool and is, therefore, not enabled by default. `telnet` is supported by most products and with `telnet` you can...

- Run scripts and applications in the product.
- Run shell commands.
- Create and delete directories and files.
- Set permissions on files.

How to enable telnet

To enable `telnet`...

1. Browse to
   `http://192.168.0.90/admin-bin/editcgi.cgi?file=/etc/inittab`
2. Locate the line
   `# tnet:35:once:/usr/sbin/telnetd`
3. Uncomment the line (i.e. delete the #).
4. Save the file.
5. Restart the product.

Now, `telnet` is enabled.

Important!

This option should only be enabled for experimental use. Never leave `telnet` access enabled when using the camera/video server on a public site.

Testing the telnet connection

1. Using `telnet`, connect to the product:

```
[claesl@juniperus ~]$ telnet 192.168.0.90

Trying 192.168.0.90...

Connected to 192.168.0.90 (192.168.0.90).

Escape character is '^]'.

4.40

Linux 2.6.17 on a cris (11:49:56)

axis-00408c7ccb28 login: root
```

```
Password:
```

Note

The password will not be echoed to the screen.

2. List the directory structure by using the `ls` command:

   ```
   [root@axis-00408c7ccb28 /]482# ls
   bin      etc      linuxrc  proc     sbin     tmp      var
   dev      lib      mnt      root     sys      usr
   ```

3. Close the connection:

   ```
   [root@axis-00408c7ccb28 /]482# exit
   Connection closed by foreign host.
   [claesl@juniperus ~]$
   ```

How to edit scripts using telnet

With a `telnet` session open to the product, you can use the built-in version of `vi` or `sed` to edit scripts.

How to disable telnet

To disable `telnet`, browse to

```
http://192.168.0.90/admin-bin/editcgi.cgi?file=/etc/inittab
```

locate the line

```
tnet:35:once:/usr/sbin/telnetd
```

insert a # in front of the line (i.e. the line will be commented out), save the file, and restart the product. `telnet` is now disabled.

FTP

All Axis video products have support for FTP.

Using FTP you can...

- Get files from the product.
- Upload files to the product.
- Set permissions, for instance to make a script executable.
- Create new directories.

Note

When you want to transfer files using FTP, ensure that you set the correct transfer mode:
`Binary` is used for images, video streams and other binary files.
`ASCII` is used for all text files.

How to invoke an FTP session

Using an FTP client, connect to the product.

```
[claesl@juniperus ~]$ ftp 192.168.0.90
Connected to 192.168.0.90.
220 AXIS 232D+ Network Dome Camera 4.40 (Feb 16 2007) ready.
Name (192.168.0.90:claesl): root
331 User name okay, need password.
Password:
230 User logged in, proceed.
Remote system type is UNIX.
Using binary mode to transfer files.
```

Note

The password will not be echoed to the screen.

How to close an FTP session

```
ftp> quit
221 Goodbye.
```

Using FTP to download a script from the product

```
[claesl@juniperus ~]$ ftp 192.168.0.90
Connected to 192.168.0.90.
220 AXIS 232D+ Network Dome Camera 4.40 (Feb 16 2007) ready.
Name (192.168.0.90:claesl): root
331 User name okay, need password.
Password:
230 User logged in, proceed.
Remote system type is UNIX.
Using binary mode to transfer files.
```

Set transfer mode to `ASCII`:

```
ftp> ascii
```

```
200 Command okay.
```

Change to the `/etc` directory:

```
ftp> cd /etc
250 Command successful.
```

And get the file (i.e. `inittab` in this case):

```
ftp> get inittab
local: inittab remote: inittab
227 Entering Passive Mode (192,168,0,90,4,3)
150 Opening data connection.
226 Transfer complete.
673 bytes received in 0.003 seconds (2.2e+02 Kbytes/s)
ftp> quit
221 Goodbye.
```

When you download a file it will be placed in your current directory.

Using FTP to upload a script to the product

```
[claesl@juniperus ~]$ ftp 192.168.0.90
Connected to 192.168.0.90.
220 AXIS 232D+ Network Dome Camera 4.40 (Feb 16 2007) ready.
Name (192.168.0.90:claesl): root
331 User name okay, need password.
Password:
230 User logged in, proceed.
Remote system type is UNIX.
Using binary mode to transfer files.
```

Set transfer mode to `ASCII`:

```
ftp> ascii
200 Command okay.
```

Change to the `/usr/html/local/scripts` directory:

```
ftp> cd /usr/html/local/scripts
250 Command successful.
```

And upload the file (i.e. `my_script` in this case):

```
ftp> put my_script
local: my_script remote: my_script
227 Entering Passive Mode (192,168,0,90,4,4)
```

```
150 Opening data connection.
226 Transfer complete.
673 bytes sent in 6e-05 seconds (1.1e+04 Kbytes/s)
ftp> chmod 755 my_script
200 Command okay.
ftp> quit
221 Goodbye.
```

When you upload a file, it will be fetched from your current directory.

Special considerations when using a Windows FTP client

If you are using a Windows FTP client, certain commands cannot be used as they normally would be in Linux. Two additional commands are needed:

1. `quote` – to send arguments, verbatim, to the remote FTP server.
2. `site` – to execute commands residing in the remote FTP server.

Commands that need quote

To see which commands need to be preceded by quote, invoke an FTP session and type `remotehelp`:

```
ftp> remotehelp
214-The following commands are implemented.
   USER    QUIT     PASS    SYST    HELP    PORT   PASV    LIST
   NLST    RETR     STOR    TYPE    MKD     RMD    DELE    PWD
   CWD     SITE     CDUP    RNFR    RNTO    NOOP   EPRT    EPSVr
214 End of list.
ftp>
```

Example – make directory from a Windows FTP client

Since the `mkd` (make directory) command is found in the `remotehelp` list, you have to precede it with the `quote` command:

```
ftp> cd /tmp
250 Command successful.
ftp> quote
Command line to send mkd my_directory
250 Command successful.
```

Commands that need both quote and site

Too see which commands need to be preceded by both `quote` and `site`, invoke an FTP session and type `remotehelp site`:

```
ftp> remotehelp site

214-The following SITE commands are implemented.

   CHMOD   REBOOT

214 End of list.

ftp>
```

Example – change file permissions from a Windows FTP client

Since the `chmod` (change file permission) command is found in the `remotehelp site` list, you have to precede it with both `quote` and `site`:

```
ftp> quote

Command line to send site chmod 755 my_script

200 Command okay.
```

Running scripts

Scripts can be invoked in several ways:

- Manually, using `telnet`.
- Automatically, for instance invoked by the task scheduler, which handles input from alarms, motion detection and other events.
- Manually, using a Web browser.

Running scripts using telnet

For development purposes, it is convenient to test scripts using `telnet`. Here is a short, experimental shell script, named `my_script` and located in `/usr/html/local/scripts`:

```
#!/bin/sh
```

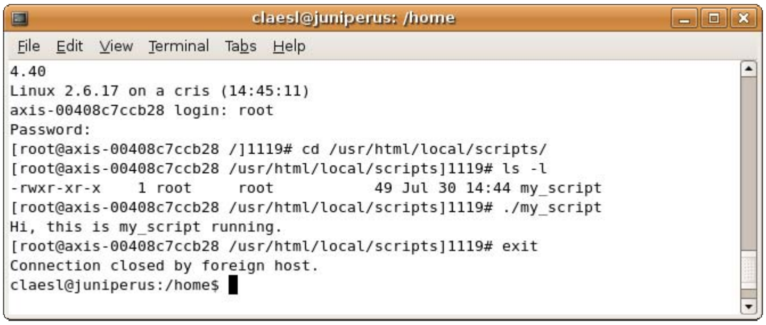
```
echo "Hi, this is my_script running."
```

To run the script, start `telnet` and log in to the product. Change directory to `/usr/html/local/scripts` and execute the script simply by typing the name of the script:

```
./my_script
```

As you can see, the script has first been made executable (`-rwxr-xr-x`) using the command

```
chmod 755 my_script
```



```
4.40
Linux 2.6.17 on a cris (14:45:11)
axis-00408c7ccb28 login: root
Password:
[root@axis-00408c7ccb28 /]1119# cd /usr/html/local/scripts/
[root@axis-00408c7ccb28 /usr/html/local/scripts]1119# ls -l
-rwxr-xr-x    1 root     root           49 Jul 30 14:44 my_script
[root@axis-00408c7ccb28 /usr/html/local/scripts]1119# ./my_script
Hi, this is my_script running.
[root@axis-00408c7ccb28 /usr/html/local/scripts]1119# exit
Connection closed by foreign host.
claesl@juniperus:/home$
```

Note

If `telnet` has not been enabled in the product you will get the error message `Connection refused` (Linux) or `Connect failed` (Windows).

Automate scripts with the task scheduler

The task scheduler (i.e. `utask`) can be used to invoke scripts when certain events occur. It can start any task and can handle process management as well.

The utask application has two configuration files: the `task.list` file and the `user.task.list` file.

Axis Communications AB provides NO support for application development of any kind. The information here is provided "as is", and there is no guarantee that any of the examples shown will work in your particular application.

- The `task.list` file is used by the event application of the product. When you set up an event in the product, the application sets up a schedule for when to trigger the event in this `task.list` file.
- The `user.task.list` is for custom tasks, where you specify your own triggers for an event or trigger own scripts, etc.

You can specify a large number of events to trigger your script – either a single event or a combination of events.

The task scheduler, `utask`, is a process that is capable of scheduling tasks based on either time of day or external events, e.g. when a digital input becomes active, or a combination of these. Any executable file in the local filesystem can be triggered by `utask`. This can, for example, be used to start buffering images or upload files using FTP or SMTP.

Built-in triggers

By editing `user.task.list` you can, for example, trigger a script:

- At a certain time, within a specified timeframe, every hour, every minute, etc.
- At a specific date, every Sunday, every other month, etc.
- On boot.
- When a digital signal becomes active or inactive.
- On audio detect.
- On motion detect when motion starts, when motion stops or when motion starts or stops.
- When video is lost on a certain channel or on any channel.
- When a PTZ camera reaches or leaves a preset position.
- When the IR cut filter is turned on or turned off.
- When temperature rises or falls outside the operating range of the product.
- On tampering.
- Manually (virtual input).
- And you can set up events of your own, as well as combine events.

When an event is triggered, the default behaviour is to execute the script, exit and then start again, for as long as the triggering condition is true. But using options, you can set an event to:

- Execute the script without being interrupted (even if the trigger condition changes).
- Run just once (even if the trigger condition remains).

Passing arguments to scripts

In `user.task.list` you can also pass arguments to the shell script or application. The script will pick them up in the order they are written in the `user.task.list`.

For specifications and examples, see the Event handling section.

Running scripts via a Web browser

You can create static as well as dynamic Web pages in your product. Testing via a Web browser is often a practical way to debug scripts that later will be automatically invoked.

Default product Web page

The default Live View & Configuration Web page of the product is password protected and can normally be reached by browsing to

```
http://192.168.0.90
```

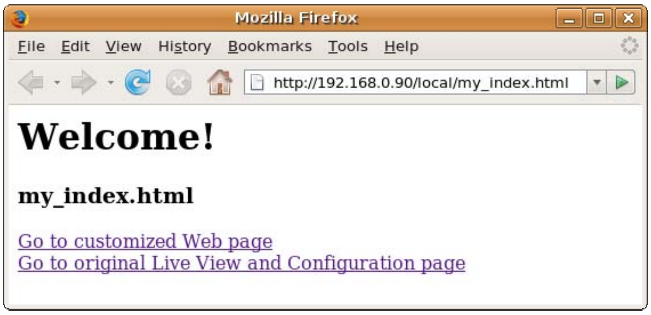
or

```
http://192.168.0.90/index.shtml
```

Custom Web pages

In the default configuration, custom Web pages are reached through the address `http://192.168.0.90/local/<filename>`. Example:

```
http://192.168.0.90/local/my_index.html
```


Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://192.168.0.90/local/my_index.html

Welcome!

my_index.html

Go to customized Web page
Go to original Live View and Configuration page

The internal Web server is configured in `/etc/httpd/conf/boa.conf`. For additional information and examples, see the section Creating custom Web pages and CGI scripts.

Specific Axis commands and helper applications

In this section you will find the following specific Axis commands and helper applications:

- buffer_handler
  Capture images and store them on the RAM disk.
- logger
  Write messages to log files. Very useful for debugging purposes.
- nc
  Send a message to a host on specified port.
- parhandclient
  List, update, add and remove product parameters.
- sftpclient
  Upload images from the product to a remote FTP server.
- shttpclient
  Open HTTP connections to the internal Web server or to a remote HTTP server.
- smtpclient
  Send e-mails with attachments.
- sockclient
  Send messages to internal product applications.
- statusled
  Set/reset the product LEDs.

buffer_handler

`buffer_handler` is used for storing images on the RAM disk.

You can specify how many images you want in the pre-alarm buffer and in the post-alarm buffer. When started, `buffer_handler` will store all images in FIFO order, meaning that when the specified number of images in the buffer is reached, the latest image will overwrite the eldest. In this way, you always have a specified number of images in the buffer.

When an alarm occurs, `buffer-handler` will shift mode: the pre-alarm images will be saved and `buffer_handler` begins to save the specified number of post-alarm images.

`buffer_handler` can also be set to best effort mode, which means that `buffer_handler` will capture an image; as soon as an uploading process has sent it and removed the old one, a new image will be captured and uploaded, and so on. This feature can be used for long and continuous uploads over a congested network.

Syntax

```
buffer_handler [options]
```

`buffer_handler` captures JPEG-images and stores them in the RAM-disk area under `/var/run/buffer_handler/buffers/<buffer name>`. The amount of memory reserved for the alarm buffers is fixed, which means that the number of images that can be stored is limited. The exact size of the RAM disk and the area reserved for the alarm buffers vary between products, please check the data sheet of your product.

If `buffer_handler` fails to write an image to the disk because it is full, it will retry until it finally succeeds (for instance when other images are removed from the disk).

Options

All options can be specified in two different ways, with one single character or with a longer, more descriptive name.

Note

A space separates the option and the value in the short form, while an equal sign separates them in the long form.

`-a number`
Number of images in the pre-alarm buffer.
Long form: `--pre_buf_size=number`

`-b number`
Number of images in the post-alarm buffer.
Long form: `--post_buf_size=number`

`-c milliseconds`
Minimum interval in milliseconds between pre-alarm images.
Long form: `--pre_interval=milliseconds`

`-d  milliseconds`
Minimum interval in milliseconds between post-alarm images.
Long form: `--post_interval=milliseconds`

Axis Communications AB provides NO support for application development of any kind. The information here is provided "as is", and there is no guarantee that any of the examples shown will work in your particular application.

`-e milliseconds` Minimum interval between best-effort images.
Long form: `--beffort_interval=milliseconds`

`-f seconds` Duration of best-effort buffering. `0` = forever.
Long form: `--beffort_duration=seconds`

`-g number` Buffer size when operating in best effort mode. Default is one image.
Long form: `--beffort_buf_size=number`

`-h` Help text.
Long form: `--help`

`-i name` An ID label for the `buffer_handler` process. This ID is used as base name for generated files. Each `buffer_handler` process must have a different label.
Long form: `--id=identification_name`

`-p priority` Execution priority for the process: `0` = low, `1` = normal, `2` = high.
Long form: `--priority={0,1,2}`

`-q query` Arguments to be added to `--api_req <request>` (or `-r <request>`) below, e.g. `-r "camera=1&resolution=4CIF"`. For available arguments, see AXIS VAPIX.
Long form: `--req_arg=query`

`-r request` The URI from where `buffer_handler` should read images. The format is described in detail in AXIS VAPIX. Default format of the URI is `/jpg/image.jpg`.
Long form: `--api_req=request`

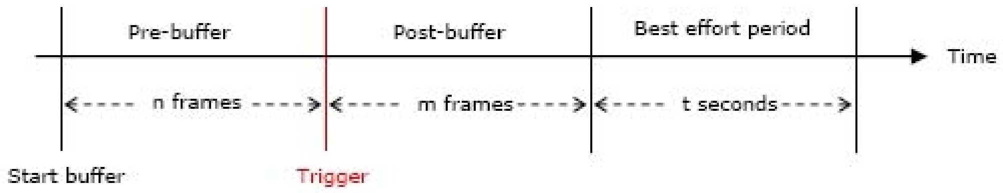
`-s` Other process will remove files.
Long form: `--ext_sync`

`-F` Use FIFO for messages to other process (see examples below).
Long form: `--fifo`

Buffer modes

`buffer_handler` can operate in three modes:

- Pre-alarm mode
- Post-alarm mode
- Best effort mode



Pre-alarm mode

In pre-alarm mode `buffer_handler` continuously stores images to the disk in FIFO-order. This means that when n images have been stored, the oldest image is overwritten by a new one, so that the pre-alarm buffer never will contain more than n images. How many images to use in the pre-buffer (n) can be specified using the `-a` (e.g. `-a 10`) or `--pre_buffer_size` (`--pre_buffer_size=10`) argument.

Example – 10 image buffer

Start `buffer_handler` with a pre-alarm buffer of `10` images. Capture 2 images per second:

```
buffer_handler -a 10 -c 500 -i testbuf &
```

The & sign is added to make the process run in the background. This is necessary when invoking the command from the command line in a `telnet` session or running it in a script.

At start-up, `buffer_handler` writes its process ID to a file called `/var/run/buffer_handler/<buffer_id>.pid`, where `<buffer_id>` is specified by the `-i` (or `--id`) argument (in this case `testbuf`).

Example – read buffer ID

Read the process ID of the testbuf:

```
cat /var/run/buffer_handler/testbuf.pid
```

When running this command at the shell prompt, it will output a number, for example `628`.

Example – switch mode

When `buffer_handler` receives the `USR1` signal it switches from pre-alarm mode to post-alarm mode. This can be triggered with the `kill` command.

Switch from pre-alarm mode to post-alarm mode:

```
kill -USR1 `cat /var/run/buffer_handler/testbuf.pid`
```

In this example we did not specify a post-alarm buffer, so this command simply stops `buffer_handler` from operating in pre-alarm mode. The `buffer_handler` process then suspends itself and waits for a `TERM` signal, which terminates the process.

Post-alarm mode

`buffer_handler` switches to post-alarm mode when it receives a `USR1` signal. In this mode, `buffer_handler` stores a specified number of images to the disk at a specified interval. Then, it switches to best effort mode or, if no best effort period has been specified, suspends itself, waiting for a `TERM` signal.

The size of the post-alarm buffer is specified with the `-b` (or `--post_buf_size`) option, and the interval is specified with the `-d` (or `--post_interval`) option.

Example – switch to post-alarm mode

Start `buffer_handler` with a pre-alarm buffer of `6` images and a post-alarm buffer of `10` images:

```
buffer_handler -a 6 -b 10 -c 500 -d 500 -i my_buf &
```

Wait 3 seconds, then send the `USR1` signal to the process:

```
kill -USR1 `cat /var/run/buffer_handler/my_buf.pid`
```

Wait another 5 seconds, now there should be 16 images stored in the `/var/run/buffer_handler/buffers/my_buf directory`.

Best effort mode

In best effort mode (or in un-buffered mode), `buffer_handler` only buffers one image at a time (however, this can be altered with the `-g` or the `--beffort_buffer_size` option). It captures one image, stores it under `/var/run/buffer_handler/buffers/<buffer_id>`, waits for an external program (i.e. an uploading process) to process the image and delete it afterwards. Then `buffer_handler` continues with the next image.

This mode is intended to be used for long, continuous uploads over networks that might be congested. To use best effort, a time period in seconds must be specified with the `-e` (or `--beffort_interval`) option and an interval in milliseconds (with the `-f` or the `--beffort_duration` option). `buffer_handler` will then try to capture images at the specified rate during the specified period. It depends on the uploading process if it actually captures any images at the specified rate successfully.

Example – best-effort mode

Start `buffer_handler` in best effort mode:

```
buffer_handler -e 1 -f 1000 -i one_shot &
```

The FIFO

When you use the `-F` (or `--fifo`) option, `buffer_handler` will also create a file called `<buffer_id>.fifo`, located in the `/var/run/buffer directory`. This FIFO is intended to be used in an uploading process, specifying which files that should be uploaded.

Example – create and read a FIFO

Create a FIFO:

```
buffer_handler -a 2 -b 2 -c 500 -d 500 -i my_fifo_test -F &
```

Wait a few seconds and switch to post-alarm mode:

Axis Communications AB provides NO support for application development of any kind. The information here is provided "as is", and there is no guarantee that any of the examples shown will work in your particular application.

```
kill -USR1 `cat /var/run/buffer_handler/my_fifo_test.pid`
```

List files in buffer:

```
ls /var/run/buffer_handler/buffers/my_fifo_test
```

Output example:

```
1164379561:773232.jpg

1164379562:273212.jpg

...

done
```

Captured files are named `<sec>:<usec>.jpg`, where `<sec>` is the number of seconds since 00:00:00 UTC, January 1, 1970, and `<usec>` is the fraction in microseconds.

Note

This naming syntax can sometimes cause problems as some FTP servers do not accept the colon character.

`buffer_handler` also creates a file called `done` when the last image has been captured. Through this file, the uploading process will know when everything has been captured.

The `buffer_handler` process removes the FIFO and all captured files when it terminates. To terminate the process an explicit `TERM` signal must be sent to it:

```
kill `cat /var/run/buffer_handler/my_fifo_test.pid`
```

logger

`logger` is a common Linux command which makes entries in the system log.

It is of great use when debugging scripts. However, when this debugging information is no longer needed, calls to `logger` should be commented out or removed to increase execution speed.

Syntax

```
logger [options] [message]
```

Write message to the system log (`/var/log/messages`). If message is omitted, log `stdin`. How messages are to be logged is set in `/etc/syslog.conf`.

Options

`-s`	Log to `stderr` as well as the system log.
`-t tag`	Log using the specified tag. Default is user name.
`-p priority`	Enter the message with specified priority, entered as a `facility.level` pair. See below for descriptions.

Facilities

`auth`	Security/authorization messages.
`cron`	Clock daemon.
`daemon`	Other system daemons.
`kern`	Kernel messages.
`lpr`	Print messages.
`news`	Usenet news messages.
`syslog`	Messages generated internally by `syslogd`.
`user`	User level messages.
`local<n>`	n = `0...7`. Local messages.

Axis Communications AB provides NO support for application development of any kind. The information here is provided "as is", and there is no guarantee that any of the examples shown will work in your particular application.

Note

In its default configuration (see `/etc/syslog.conf`), the product sees no difference between these facilities.

Levels

Depending on level, logged messages will be flagged as `INFO`, `WARNING` or `CRITICAL`.

INFO level:

`info`	Informational messages.
`notice`	Conditions that are not error conditions, but which may require special handling.

WARNING level:

`warning`	Warning messages.

CRITICAL level:

`err`	Error messages.
`crit`	Critical conditions, such as hardware errors.
`alert`	A condition that should be corrected immediately.
`emerg`	A panic condition.

Special level (not logged by default):

`debug`	For custom logging. See example below.

Example – simple logging

Log a message to `/var/log/messages`:

```
logger "my_script runs"
```

When running the command, a line like this will be added to `/var/log/messages`:

```
<INFO    > Nov 21 11:59:48 axis-00408c7ccb28 root: my_script runs
```

Example – add custom tag

Log a message to `/var/log/messages` with a custom tag:

```
logger -t my_script "my_script runs"
```

When running the command, a line like this will be added to the log file `/var/log/messages`:

```
<INFO    > Nov 21 13:00:41 axis-00408c7ccb28 my_script: my_script runs
```

Example – change message level

Log a message to `/var/log/messages` with message level warning:

```
logger -t level_test -p user.warning "A warning"
```

When running the command, a line like this will be added to the log file `/var/log/messages`:

```
<WARNING > Nov 21 13:00:42 axis-00408c7ccb28 level_test: A warning
```

Log a message to `/var/log/messages` with message level critical:

```
logger -t level_test -p user.crit "A critical message"
```

When running the command, a line like this will be added to the log file `/var/log/messages`:

```
<CRITICAL> Nov 21 13:00:43 axis-00408c7ccb28 level_test: A critical
message
```

Add a custom log

For debugging purposes it may be practical to keep your test messages out of the system log. To direct them to a custom log, just edit `/etc/syslog.conf`.

Add these lines to `/etc/syslog.conf` and restart the product:

```
# Log all level debug messages in one place.

local0.debug        /var/log/my_debug_log
```

When the product is restarted you will find a new, empty file in the `/var/log` directory:

```
-rw-r--r--    1 root     root            0 Nov 21 16:13 my_debug_log
```

Example – log a debug message

Run the command:

```
logger -t my_script -p local0.debug "my_script runs"
```

When running the command a line like this will be added to the custom log file `/var/log/my_debug_log`:

```
Nov 21 16:24:26 axis-00408c7ccb28 my_script: my_script runs
```

How to use logger in scripts

Log the value of the argument passed from `user.task.list` and log that the script is running, both messages to be sent to `/var/log/my_debug_log`. Also log that the script has started to `/var/log/messages`. The script is to be run once when the product is started.

Ensure that `my_debug_log` has been enabled in `/etc/syslog.conf` (see above).

Add these lines to `/etc/user.task.list`:

```
# start script at boot

once % /usr/html/local/scripts/my_script : "Started on product start";
```

The file `/usr/html/local/scripts/my_script` contains these lines:

```
#!/bin/sh

# Log the argument passed from user.task.list
logger -t my_script -p local0.debug $1

# Log that the script is running
logger -t my_script -p local0.debug "my_script runs"

# Also send a line to /var/log/messages
logger -t my_script  "$1"
```

When running the script two lines like these will be added to the log file `/var/log/my_debug_log`:

```
Nov 21 19:49:49 axis-00408c7ccb28 my_script: Started on product start
Nov 21 19:49:49 axis-00408c7ccb28 my_script: my_script runs
```

And a line like this will be added to `/var/log/messages`:

```
<INFO    > Nov 21 19:49:49 axis-00408c7ccb28 my_script: Started on
product start
```

nc

nc, netcat, is a common Linux command which is valuable for event handling. It reads and writes data across network connections, using the TCP or UDP protocol. The BusyBox nc included in the product is a limited version of GNU nc, i.e. not all GNU options are supported.

nc can be used to send a string from a product to a specific port on a remote host. The standard input is then sent to the host, and anything that comes back across the connection is sent to the standard output. This continues indefinitely, until the network side of the connection shuts down.

Note

This behaviour is different from most other applications, which shut down everything and exit after having received an `end-of-file` on the standard input.

Syntax

```
nc [options] [ip] [port]
```

Open a pipe to `ip:port`.

Options

`-4`	Use IPv4 (default).
`-6`	Use IPv6.
`-u`	UDP mode.
`-p port`	Local port number.
`-i secs`	Delay interval for lines sent.
`-w secs`	Timeout for connects and final net reads.
`-D dscp`	Set IP dscp field.
`-P prio`	Set VLAN user priority.

Example – send TCP notification to a remote host

Send the string `"Alarm on my network camera"` to the host `10.13.17.71` at port `4444`. Set the timeout to `10` seconds

```
echo "Alarm on my cam" | nc -w 10 10.13.17.71 4444
```

Axis Communications AB provides NO support for application development of any kind. The information here is provided "as is", and there is no guarantee that any of the examples shown will work in your particular application.

parhandclient

`parhandclient` is a built-in application that can be used to get, set, add and remove parameters (see description in AXIS VAPIX). The parameters available in a specific product can be viewed in the Parameter List or the Server Report in the Web interface. The parameter syntax is `[root].<group>.<name>`, e.g. `root.Network.IPAddress` (`root` is optional).

Syntax

```
parhandclient [--nolog] [--nocgi] [--nosync] [--sigonly]
[--casesensitive] <commands>
```

Set/get product parameters.

Commands

`get <name> [<file> [<type>]]`
Get a single parameter.

`name` Name of parameter to get.

`file` File to which the value will be written. Use a hyphen (-) as file to send the output to stdout.

`type` Described below.

`getgroup <group> [<file> [<type>]]`
Get a group of parameters.

`group` Name of group to get, e.g. `root.Network`.

`file` File to which the values will be written. Use a hyphen (-) as file to send the output to stdout.

`type` Described below.

`getgrouplist <group> [<file> [<type>]]`
List all subgroups within a group.

`group` Name of group to get, e.g. `root.Network`.

`file` File to which the values will be written. Use a hyphen (-) as file to send the output to stdout.

`type` Described below.

Axis Communications AB provides NO support for application development of any kind. The information here is provided "as is", and there is no guarantee that any of the examples shown will work in your particular application.

`getlist <infile> <outfile> <type>`

Get the value of the parameters listed in `<infile>` to `<outfile>`.

`infile` File where the parameter names are specified, one name per line.

`outfile` File to where the value of the parameters specified in `<infile>` will be written.

`type` Described below.

`readgroupfile <file>`

Read a definition file where groups and parameters are defined. Typically used when the group/parameters are to be used only when an application (e.g. a driver) is loaded. Saves memory since unused groups will not load `parhand`.

`file` The definiton file. For sample definition files, see `/usr/etc/param`.

`set <name> <value>`

Set a single parameter.

`name` Name of parameter to set.

`value` Parameter value.

`setparams <file>`

Set parameters from a file.

`file` File where the parameters are specified. This file has the same format as the output from the command
`parhandclient getgroup root <file> NAMEVALUE`
which makes `getgroup/setparams` a practical combination for backup/restore operations.

`sync`

Used when writing several parameters at the same time, previously set with the `--nosync` option.

`addgroup`

Create dynamic parameter group. See below.

`deletegroup`

Delete dynamic parameter group. See below.

`upgradedynparams <directory | file>`

Upgrade dynamic parameters (used only for system upgrades).

Type

`SCRIPT` Get parameter(s) in a format suitable for scripts, e.g. `<parameter_name>='<value>'`.

`VALUE` Get parameter value(s) in the format `"<value>"`.

`NAMEVALUE` Get parameter(s) in the format `<parameter name>="<value>"`.

`NAMEVALUESECTIONS` Get parameter(s) in the format `[group] <parameter name>="<value>"`.

`FORM` Get parameter(s) in a format used for HTML documents, i.e. `<input type='hidden' name='conf_<parameter group>_<parameter name>' value='<value>'>`.

`RAW` Get parameter(s) in raw format without any conversion.

Options

`--nolog` When used, any error during the execution of `parhandclient` will not be logged in `syslog`.

`--nocgi` When calling `parhandclient` or activating a script with a call to `parhandclient` from a Web page, a new line will be added to separate the HTTP header from the answer. This could cause the return value to be corrupt. Always use this option when calling `parhandclient` from a script to avoid corrupted return values from `parhandclient`.

`--nosync` Used when setting several parameters at the same time to avoid numerous writings to the flash memory. Use the `sync` command to write all changes at once to the flash memory. The applications subscribing to parameter changes will only be notified once, i.e. when all new values have been set (at the sync).

`--sigonly` Used when setting parameter(s) in RAM. It sends a signal to the application using the parameter(s) to re-read them. The parameter(s) will not be written to the flash memory, so when restarting the product the parameter(s) will lose the values set with this option.

`--casesensitive`	Normally, group and subgroup names are not case sensitive (i.e. parhandclient considers `root.snmp` and `root.SNMP` as identical). To force case sensitivity, use this option.

The dots following the output examples in the section below mean "and so on". All `get` and `getgroup` examples can be tested in a `telnet` session to the product. The `set` commands can also be tested, but be sure to set the parameters to valid values. Bear in mind that the flash chip will be written to every time a `set` command is used.

Note

If the receiving file does not exist, it will be created. If it exists, it will be overwritten.

Add dynamic parameter group

Syntax

```
parhandclient addgroup <group> [--number <number>] <template>
```

Create a dynamic parameter group under the parent `<group>`.

`group`	Parent parameter group under which the new parameter group will be created.
`number`	Number to suffix the newly created parameter group with. If omitted, the number will be chosen automatically.
`template`	Name of file template to use when creating parameter groups. For format of template files, see below.

If `--number <number>` is specified, the new parameter group will have a suffix number `<number>` if number is not already taken. Otherwise the number is chosen automatically, starting from zero (`0`).

The parameter group created will be named after the first letter in the parent parameter group and concatenated with the number mentioned earlier.

Note

Parent parameter group can NOT be root. It must be in the format *`root.<group>[.<subgroups>. ...]`*

The newly created parameter group is built from two template files (named `<template>_grp.conf` and `<template>_par.conf`). `parhandclient` will look for these files in `/usr/etc/sysconfig/template` and `/etc/sysconfig/template`.

- `/usr/etc/sysconfig/template` is a read-only directory and is therefore used only for built-in templates.
- `/etc/sysconfig/template` is used for your own templates as well as those created at run-time.

Axis Communications AB provides NO support for application development of any kind. The information here is provided "as is", and there is no guarantee that any of the examples shown will work in your particular application.

Template file layout

The template files should have `REPLACE_GROUP`, `REPLACE_FILE` and `REPLACE` as markers for the parameter handler application. These will be changed to appropriate values when the parameter group is created by the parameter handler application in the product.

The parameter group file `<template>_grp.conf`:

```
group REPLACE_GROUP {
   file = "REPLACE_FILE"
   parser = Standard2parser
   param <parameter 1> {
      mount = "Standard2{REPLACE}{<parameter 1>}"
   }
   param <parameter 2> {
      mount = "Standard2{REPLACE}{<parameter 2>}"
   }
   param <parameter 3> {
      mount = "Standard2{REPLACE}{<parameter 3>}"
   }
}
```

The parameter file `<template>_par.conf`:

```
[REPLACE]
<parameter 1> = <parameter 1 value>
<parameter 2> = <parameter 2 value>
<parameter 3> = <parameter 3 value>
```

Delete dynamic parameter group

Syntax

```
parhandclient deletegroup <group>
```

Delete the parameter group `<group>` and all parameters in it. All parameter groups below this parameter group are also deleted. The command can be used for all parameter groups except `root` (i.e. even those who have not been created with the `addgroup` command).

However, configuration files are not deleted for parameter groups that have not been created with `addgroup` since they can have arbitrary names and are often stored on a read-only partition. When the parameter handler is re-started, these parameter groups and parameters will show up in the parameter list again.

Example – get name and value

Get all parameters in the product to the standard output in the format `<parameter name>="<value>"`:

```
parhandclient getgroup root - NAMEVALUE
```

Output:

```
root.Bandwidth.Limit="0"

root.Brand.Brand="AXIS"

root.Brand.ProdFullName="AXIS 232D+ Network Dome Camera"

...
```

Example – get value

Get all network parameters to the standard output in the format `"<value>"`:

```
parhandclient getgroup root.Network - VALUE
```

Output:

```
""

"dhcp"

"192.168.0.90"

"255.255.255.0"

"192.168.0.255"

...
```

Example – get, form format

Get a single parameter to a file in the format `<input type='hidden' name='conf_<parameter group>_<parameter name>' value='<value>'>`:

```
parhandclient get root.Network.IPAddress /tmp/my_form.html FORM
```

Output (in file `/tmp/my_form.html`):

```
<input type='hidden' name='conf_Network_IPAddress'
value='192.168.0.90'>
```

Example – get, raw

Get a single parameter to the standard output in the format `<value>`:

```
parhandclient get root.Network.IPAddress - RAW
```

Output:

```
192.168.0.90
```

Example – get to file

Get all network parameters to a file in the format `[group] <parameter name> = "<value>"`:

```
parhandclient getgroup root.Network /tmp/network_params
NAMEVALUESECTIONS
```

Output (in file `/tmp/network_params`):

```
[Network]
DomainName                = ""
BootProto                 = "dhcp"
IPAddress                 = "192.168.0.90"
SubnetMask                = "255.255.255.0"
...
```

Example – get from infile

Create a file with the format `<parameter name>="value"` with the parameters specified in `/tmp/parameters`:

The file `/tmp/parameters` contains:

```
root.Network.IPAddress
root.SMTP.MailServer1
root.Time.TimeZone
```

Enter this command line:

```
parhandclient getlist /tmp/parameters /tmp/param_value NAMEVALUE
```

Output (in file `/tmp/param_value`):

```
root.Network.IPAddress="192.168.0.90"
root.SMTP.MailServer1="192.168.0.5"
root.Time.TimeZone="GMT"
```

Example – get, script format

Get a single parameter to a script file in the format `<parameter_name>='<value>'`:

```
parhandclient get root.Network.IPAddress /tmp/ipaddress SCRIPT
```

Output (in file `/tmp/ipaddress`):

```
root_Network_IPAddress='192.168.0.90'
```

Example – get group list

Get a list of all groups under the `root.Time` group:

```
parhandclient getgrouplist root.Time
```

Axis Communications AB provides NO support for application development of any kind. The information here is provided "as is", and there is no guarantee that any of the examples shown will work in your particular application.

Output:

```
"DST"
"NTP"
```

Example – set parameter

Set a single parameter:

```
parhandclient set root.Network.HostName my-axis-product
```

Example – set several parameters

Set several parameters. To save the flash memory from unnecessary writings, the parameters will not be written until the `sync` command is sent:

```
parhandclient --nosync set root.Network.IPAddress 192.168.0.90

parhandclient --nosync set root.Network.SubnetMask 255.255.255.0

parhandclient --nosync set root.Network.HostName my-axis-product

parhandclient sync
```

Example – set temporary parameter

Set a single parameter in the RAM memory. A signal will be sent to the application using the parameter to re-read the configuration. Since the parameter is not saved to the flash memory, the parameter will revert to its old value (i.e. the one it had before this command was issued) when the product is restarted:

```
parhandclient --sigonly set root.Network.HostName my-axis-product
```

Example – set parameters from infile

Set parameters from a file:

Example file (`/tmp/parameters`):

```
root.Network.IPAddress="192.168.0.90"

root.Network.SubnetMask="255.255.255.0"

root.Network.HostName="my-axis-product"
```

Command:

```
parhandclient setparams /tmp/parameters
```

Examples – add dynamic parameter groups using parhandclient

Create a dynamic parameter group named `Example` directly under `root`. The template files are `example_file_grp.conf` and `example_file_par.conf`.

Command:

```
parhandclient addgroup root.Example example_file
```

Output:

```
root.Example.E0
```

Create a dynamic parameter group named `AnotherExample` with number 3 under `root.Example.E0`. The template files are `somename_grp.conf` and `somename_par.conf`.

Command:

```
parhandclient addgroup root.Example.E0.AnotherExample --number 3
somename
```

Output:

```
root.Example.E0.AnotherExample.A3
```

Example – delete a dynamic parameter group using parhandclient

Delete parameter group `root.Example.E0` and any parameter groups the `E0` group has.

Command:

```
parhandclient deletegroup root.Example.E0
```

Example – using built-in templates

In this example you will create dynamic parameters, in this case an event, using built-in templates. The name of the parent group is `Event` and the template files to be used are `event_grp.conf` and `event_par.conf`.

The template `event_grp.conf` defines the parameters needed as well as their types. Here is an excerpt of the template, describing the first ten parameters:

event_grp.conf (excerpt)

```
group REPLACE_GROUP {


      file = "REPLACE_FILE"
      parser = Standard2parser
      setter = "/bin/parhand2utask"
      secLevel = "4444"
      param Name {
            mount = "Standard2{REPLACE}{Name}"
            type = "string"
      }
      param Type {
            mount = "Standard2{REPLACE}{Type}"
            type = "enum:S,T"
      }
      param Enabled {
            mount = "Standard2{REPLACE}{Enabled}"
            type = "yes_no"
      }
      param Active {
            mount = "Standard2{REPLACE}{Active}"
            setter = "disabled"
            type = "nosync"
      }
      param Priority {
            mount = "Standard2{REPLACE}{Priority}"
            type = "enum:0,1,2"
      }
      param Image {
            mount = "Standard2{REPLACE}{Image}"
            type = "int"
      }
      param HWInputs {
            mount = "Standard2{REPLACE}{HWInputs}"
            type = "string"
      }
      param SWInput {
            mount = "Standard2{REPLACE}{SWInput}"
            type = "string"
      }
      param Weekdays {
            mount = "Standard2{REPLACE}{Weekdays}"
            type = "string"
      }
      param Starttime {
            mount = "Standard2{REPLACE}{Starttime}"
            type = "string"
      }
      ...
```

The template `event_par.conf` sets the default values of the parameters described above. Here is an excerpt of the template, showing the default values for the first ten parameters:

event_par.conf (excerpt)

```
[REPLACE]
Name = "New Event"
Type = T
Enabled = yes
Active = no
Priority = 1
Image = 0
HWInputs = xxxx
SWInput =
Weekdays = 1111111
Starttime = 00:00
...
```

To create a new event group, issue the command (using `telnet` or by executing a script)

```
parhandclient addgroup Event event
```

The command will return the name of the new group, for example

```
Event.E3
```

To list the parameters you just created, issue the command

```
parhandclient getgroup Event.E3
```

and this list will be produced (excerpt):

```
root.Event.E3.Name="New Event"
root.Event.E3.Type="T"
root.Event.E3.Enabled="yes"
root.Event.E3.Active="no"
root.Event.E3.Priority="1"
root.Event.E3.Image="0"
root.Event.E3.HWInputs="xxxx"
root.Event.E3.SWInput=""
root.Event.E3.Weekdays="1111111"
root.Event.E3.Starttime="00:00"
...
```

Using parhandclient in scripts

Example: Get the network settings of the product using `parhandclient` and send them in an e-mail to an administrator when the product is started. `parhandclient` will also get the IP address which is then used as a part of the sender address. For more information, see the `smtpclient` command.

The script file (`/usr/html/local/scripts/notify`):

```
#!/bin/sh

# this script is set to run once at boot
# Use parhandclient to create a file containing the Network parameters
parhandclient getgroup root.Network /tmp/nwparams NAMEVALUE

# Configure SMTP parameters
# Get the IP address by parhandclient and use it
# to create a from address
from="`parhandclient --nocgi get root.Network.IPAddress -
RAW`@somewhere.com"

# The server to use as mail server
smtp_server="smtpserver.somewhere.com"

# The subject line for the mail
subject="Network_parameters"

# The body to insert into the mail.
# Note that this must be specified and point at a valid file
body="/tmp/nwparams"

# Recipient
to="admin@somewhere.com"

# Send the mail
smtpclient -s $subject -S $smtp_server -f $from -b $body $to
```

Edit the `/etc/user.task.list` file to invoke the script at boot:

```
immune once % /usr/html/local/scripts/notify;
```

Restart the product to make it re-read the configuration file and an e-mail such as this will be sent at boot:

```
From: 192.168.0.90@somewhere.com
Subject: Network_parameters
Sender: root@192.168.0.90
```

```
Message-ID: <1164194683.880237.482-root@192.168.0.90>
To: admin@somewhere.com
Date: Wed, 22 Nov 2006 11:24:37 +0100 (CET)

root.Network.DomainName=""
root.Network.BootProto="dhcp"
root.Network.IPAddress="192.168.0.90"
root.Network.SubnetMask="255.255.255.0"
...
```

sftpclient

`sftpclient` is a built-in application that can be used to upload files to a remote FTP server from the product. This functionality is often used to send images to a specified FTP server when an event occurs.

The examples here can be tested from the shell prompt in a `telnet` session to the product. Before testing the examples, be sure to change the arguments to valid values and make sure that the files used are valid.

Syntax

```
sftpclient [options]
```

Options

`-p <server>`	Put a local file to a remote FTP server.
`-i <server>`	Open an interactive FTP session to this server.
`-m <server>`	Put all files in local directory to remote server directory.
`-M <server>`	Put all files received on stdin to remote server directory.
`-n <port>`	Port number used on remote server (default = `21`).
`-s`	Use passive mode FTP.
`-t <file>`	Use temporary file.
`-f <fifo>`	Read input from `fifo`.
`-c <remote dir>`	Remote directory to start in.
`-d <remote file>`	File to get/put.
`-k <local dir>`	Local directory to start in.
`-l <local file>`	File to get/put.
`-q`	QoS dsfield:user-priority for control flow.
`-Q`	QoS dsfield:user-priority for data flow.
`-u <username>`	The user name for the remote FTP server.
`-w <password>`	The password for the remote FTP server.
`-T <seconds>`	Seconds to timeout on response waits.

`-I <seconds>`	Connection timeout (log out if no uploads to server were made within the last `seconds`).
`-L`	Log errors to `syslog` instead of `stderr`.
`-D`	Remove local file when it has been uploaded.
`-F`	Create remote directories.

Backup server options

`-B <server>`	Backup remote FTP server to use if primary fails.
`-N <port>`	Remote port number on backup remote FTP server (default is `21`).
`-S`	Use passive mode ftp for backup remote FTP server.
`-C <remote dir>`	Remote directory to start in on backup remote FTP server.
`-U <username>`	User name on backup remote FTP server.
`-W <password>`	Password on backup remote FTP server.

Note

Support for the options above is highly product and firmware dependent. To see supported options, open a `telnet` session to the product and type `sftpclient`. (The `-g` (get) option is not implemented.)

Upload a single file

Upload the file `/var/log/messages` to the `root` directory on FTP server `10.13.9.40` with the user name `user` and password `password`.

```
sftpclient -p 10.13.9.40 -u user -w password -l /var/log/messages
```

Example – upload file and change its name

Upload the file `/var/log/messages` to the directory `/uploads` on FTP server `10.13.9.40` and name the file `log.txt`.

```
sftpclient -p 10.13.9.40 -u user -w password -l /var/log/messages -c
/uploads -d log.txt
```

Example – move file

Upload the file `/tmp/my_image.jpg` to the directory `/uploads` on FTP server `10.13.9.40` and then delete the file (i.e. move the file).

```
sftpclient -p 10.13.9.40 -u user -w password -l /tmp/my_image.jpg -c
/uploads -D
```

Axis Communications AB provides NO support for application development of any kind. The information here is provided "as is", and there is no guarantee that any of the examples shown will work in your particular application.

Upload several files

Put all the files in the directory `/tmp/my_images` to the directory `/images` on FTP server `10.13.9.40`.

```
sftpclient -m 10.13.9.40 -u user -w password -k /tmp/my_images -c
/images
```

Example – upload and rename several files

Put all the files in the directory `/tmp/my_images` to the directory `/images` on FTP server `10.13.9.40` and name them `image00001.jpg`, `image00002.jpg`, etc.

```
sftpclient -m 10.13.9.40 -u user -w password -k /tmp/my_images -c
/images -d image#s.jpg
```

Open an interactive FTP session

Open an interactive FTP session to FTP server `10.13.9.40` with user name `user` and password `password`. Put the file `/var/log/messages` in the remote directory `/uploads` and name the file `log.txt`. Responses from the FTP server are marked in bold text in the example.

```
sftpclient -i 10.13.9.40 -u user -w password

OK

cd /uploads

0

put /var/log/messages log.txt

0

bye

1
```

Create a directory on an FTP server

Open an interactive FTP session to FTP server `10.13.9.40`. Create a directory (named `test`). Responses from the FTP server are marked in bold text in the example.

```
sftpclient -i 10.13.9.40 -u user -w password

OK

mkdir test

0

bye

1
```

Example – create remote directories if needed

```
sftpclient -p 10.13.9.40 -u user -w password -l /var/log/messages -F
-d /my_home/level_1/level_2/level_3/uploaded_msgs
```

Axis Communications AB provides NO support for application development of any kind. The information here is provided "as is", and there is no guarantee that any of the examples shown will work in your particular application.

Using the `-F` option, `sftpclient` will create the directories needed (i.e. those specified after the `-d` option) if they do not exist. Without the `–F` option, the entire path must already exist for the command to succeed.

FTP backup server

Use an FTP backup server if the primary FTP server fails.

```
sftpclient -m 10.13.9.40 -u user -w password -k /tmp/my_images
-c /images -B 10.13.9.50 -U user -W password -C /images
```

Upload files received on stdin

`sftpclient` can also be run in `mput` mode (the `-M` option), which means that it reads the names of which files to upload from `stdin` and gradually uploads them to an FTP server. `sftpclient` continues until an empty file name is encountered.

Example – upload files using a FIFO

This shell script starts an image buffer (using the command `buffer_handler`) and uploads captured images to an FTP server. The buffer handler tells `sftpclient` which files to upload through a FIFO.

Note

Files in this cache cannot be deleted like normal files since each is tied to a counter which keeps track of how many clients presently use the file. When the counter reaches zero, the cache will delete the file. So instead of deleting the physical file, its counter should be decreased, which is done through `libcache_unlink.so`. It replaces the standard `unlink` function (through the shell variable `LD_PRELOAD`).

```
#!/bin/sh

# Start a buffer handler with a pre-alarm and
# a post-alarm buffer of 5 images each
buffer_handler -a 5 -b 5 -c 500 -d 500 -F -i my_buf &

# Sleep for 3 seconds to make sure the pre-images are taken
sleep 3

# Get the process id of buffer_handler
PID=`cat /var/run/buffer_handler/my_buf.pid`

# Stop the buffer to make it save the pre-images and
```

Axis Communications AB provides NO support for application development of any kind. The information here is provided "as is", and there is no guarantee that any of the examples shown will work in your particular application.

```
# start taking the post-images
kill -USR1 $PID

# FTP server info
SERVER="10.13.9.40"
USER="user"
PASS="password"
UPLOADPATH="/uploads"

# Uploaded files will be named using this format
FORMAT="image%y%m%d_%H%M%S%f.jpg"

# Files to be uploaded will be read from this FIFO
FIFO="/var/run/buffer_handler/my_buf.fifo"

# Go to working directory
cd /var/run/buffer_handler/buffers/my_buf

# Use the LD_PRELOAD macro to replace the unlink function
# in glibc with the one in libcache_unlink.so
export LD_PRELOAD=libcache_unlink.so:$LD_PRELOAD

# Launch sftpclient
# sftpclient continues uploading until an empty name
# is encountered
sftpclient -M $SERVER -u $USER -w $PASS -c $UPLOADPATH -d "$FORMAT" -D
-f $FIFO

# Terminate buffer_handler when done
kill $PID
```

shttpclient

`shttpclient` is a built-in application that can be used for HTTP connections both externally and internally in the product.

It can be used internally in the product to send requests to the local host, for example to capture a single image from the internal Web server using AXIS VAPIX and save it in the `/tmp` directory. It can also be used to open a connection to a remote Web server and for example upload images to an application on the HTTP server.

`shttpclient` supports basic authentication for Web servers and proxy servers. The local host (`http://127.0.0.1`) can be used as a simple method of fetching single live images from the camera for temporary storage on `/tmp` prior to SMTP or FTP transfer. `shttpclient` can trigger any of the functions in AXIS VAPIX for Axis' video products.

Note

`shttpclient` cannot be run from a CGI script since the internal Web server (boa) can only handle one request at a time.

Syntax

```
shttpclient [options] url
```

Options

`-i <input file>`	File to POST.
`-o <output file>`	File to GET.
`-f <newfile>`	Rename input file to `newfile`.
`-T <seconds>`	Response timeout.
`-F <fifo>`	Read files to POST from `fifo`.
`-u <username>`	User name.
`-w <password>`	User's password.
`-x <proxy>`	Address of proxy server.
`-n <port>`	Port to use on proxy server.
`-a <proxyuser>`	User name on proxy server.
`-b <proxypass>`	User's password on proxy server.
`-q <dsfield:user-priority>`	QoS dsfield:user-priority for control flow.

Axis Communications AB provides NO support for application development of any kind. The information here is provided "as is", and there is no guarantee that any of the examples shown will work in your particular application.

`-p`	Print messages to `stdout`.
`-I`	Read files to POST from `stdin`.
`-D`	Delete input files after POST.

Example – store single image in product

Use local host (`http://127.0.0.1`) to fetch a single live image from the product Web server and store it on the same product as `/tmp/snap.jpg`:

```
shttpclient -o /tmp/snap.jpg -u root -w password
http://127.0.0.1/axis-cgi/jpg/image.cgi
```

Example – store single image in remote server

Upload the image `/tmp/snap.jpg` to a remote Web server. (The script `upload.cgi` must be able to receive the incoming file.) Log on as user `username` with password `password`:

```
shttpclient -i /tmp/snap.jpg -u username -w password
http://www.somewhere.com/cgi-bin/upload.cgi
```

Using shttpclient in scripts

Example – upload to remote host

Fetch a single live image from the camera Web server and store it on the same camera (local host, i.e. `http://127.0.0.1`) as `/tmp/snap.jpg`. The image will then be uploaded to a remote Web server (the script `upload.cgi` must be able to receive the incoming file):

```
#!/bin/sh

# Fetch the image and save it as /tmp/snap.jpg

shttpclient -o /tmp/snap.jpg -u root -w password
http://127.0.0.1/axis-cgi/jpg/image.cgi

# Upload the file to a remote Web server (the script upload.cgi
# must be able to receive the incoming file).

shttpclient -i /tmp/snap.jpg -u username -w mypass
http://www.somewhere.com/cgi-bin/upload.cgi
```

Example – send image in e-mail

Fetch an image and store it as `/tmp/snap.jpg` and send it as an attachment in a mail to `someone@somewhere.com` (for more information, see the `smtpclient` command):

```
#!/bin/sh

# Fetch the image and save it as /tmp/snap.jpg

shttpclient -o /tmp/snap.jpg -u root -w password
http://127.0.0.1/axis-cgi/jpg/image.cgi
```

```
# Send the image to someone@somewhere.com

smtpclient -s "my_image" -S "smtpserver.somewhere.com" -f
"axis.product@somewhere.com" -b "/var/log/messages" -M 2 -a
"/tmp/snap.jpg" someone@somewhere.com
```

smtpclient

Using `smtpclient` you can send e-mails with attached files from a product. Before testing the examples here, be sure to change the arguments to valid values and make sure that the files are valid.

Syntax

```
smtpclient [options] recipients ...
```

Options

All options can be specified in two different ways, with one single character or with a longer, more descriptive name.

Note

A space separates the option and the value in the short form while an equal sign separates them in the long form.

Message options

`-s str *`
Subject line of message.
Long form: `--subject=str`

`-u file`
Read subject line from `file`.
Long form: `--subject-file=file`

`-f addr *`
Address of the sender.
Long form: `--from=addr`

`-r addr`
Address of the sender for replies.
Long form: `--reply-to=addr`

`-e addr`
Address to send delivery errors to.
Long form: `--errors-to=addr`

`-c addr`
Address to send copy of message to.
Long form: `--carbon-copy=addr`

`-a file`
Binary MIME attachment. If no mime encode is set, MIME-style is set to `1` = application/octet.
Long form: `--attachment=file`

`-d dir`
Binary MIME directory attachment. If no mime encode is set, MIME-style is set to `1` = application/octet.
Long form: `--directory=dir`

`-i`
Interactive attachment mode, attachments are read from `stdin`.
Long form: `--interactive`

`-b file *` Read mail body from `file`.
Long form: `--body=file`

`-T secs` Seconds to timeout response wait.
Long form: `--timeout=secs`

`*` = Required.

Processing options

`-S host` Host where MTA can be contacted via SMTP.
Long form: `--smtp-host=host`

`-B host` Backup host to use if primary host is unavailable.
Long form: `--backup-host=host`

`-P port` Port where MTA can be contacted via SMTP.
Long form: `--smtp-port=port`

`-Q port` Port where MTA on backup host can be contacted via SMTP.
Long form: `--smtp-port2=port`

`-M n` Use MIME-style translation to quoted-printable or base 64:
`1` = application/octet, `2` = image/jpeg, `3` = plain/text.
Long form: `--mime-encode=n`

`-C str` Body in charset `str`.
Long form: `--charset=str`

`-L` Log errors to `syslog` facility instead of `stderr`.
Long form: `--use-syslog`

`-D` Delete body/subject/attachment files after they are processed.
Long form: `--delete-files`

`-A n` Maximum number of attachments per mail. Split up in several mails if there are more.
Long form: `--max-attach=n`

`-q n:n` Set QoS dsfield and user-priority.
Long form: `--qos-field=n:n`

`-F str` Name attachments according to the format string.
Long form: `--format=str`

`-g id` User id for authentication.
Long form: `--user=id`

`-G pwd` User password for authentication.
Long form: `--pass=pwd`

`-k id`
User id for authentication on backup server.
Long form: `--user2=id`

`-K pwd`
User password for authentication on backup server.
Long form: `--pass2=pwd`

`-w str`
Weakest allowed SMTP authentication mechanism, one of: `DIGEST-MD5 CRAM-MD5 LOGIN PLAIN`.
Long form: `--weakest=str`

`-x str`
Weakest allowed SMTP authentication mechanism, for login to backup server.
Long form: `--weakest2=str`

`-j host`
Host for POP server. If present POP-login will be used instead of SMTP-authentication.
Long form: `--pop-server=host`

`-l host`
Host for POP server. This one is for login to backup mail server.
Long form: `--pop-server2=host`

Feedback

`-v`
Enable verbose logging messages.
Long form: `--verbose`

`-V`
Display version string.
Long form: `--version`

`-h`
Display the help page.
Long form: `--help`

Example – e-mail body

Send an e-mail via the SMTP server (`-S`) `10.13.178.7` with the subject (`-s`) `test`, the log file as the body (`-b`), the sender (`-f`) `axis.product@somewhere.com` to the recipient `someone@somewhere.com`:

```
smtpclient -S 10.13.178.7 -s 'test' -b '/var/log/messages' -f
'axis.product@somewhere.com' someone@somewhere.com
```

Example – e-mail with image attachment

Use the same arguments as in the example above and add the file `/tmp/snapshot.jpg` as an attachment. Since the file is an image, the `-M` option must be set to 2 (i.e. `image/jpeg`):

```
smtpclient -S 10.13.178.7 -s 'test' -b '/var/log/messages' -f
'axis.product@somewhere.com' -M 2 -a '/tmp/snapshot.jpg'
someone@somewhere.com
```

Example – e-mail with several attachments

Use the same arguments as in the first example and add the directory `/tmp/SNAP` as an attachment. Since the files in the directory are images, the `-M` option must be set to `2` (i.e. `image/jpeg`):

```
smtpclient -S 10.13.178.7 -s 'test' -b '/var/log/messages' -f
'axis.product@somewhere.com' -M 2 -d '/tmp/SNAP' someone@somewhere.com
```

Using smtpclient in scripts

Example: Notify someone by e-mail when the product boots. Add the IP address in the senders address to show which product sent the message and add the log file as body of the e-mail. For more information also see the command `parhandclient` and the Event handling section.

The source of `/usr/html/local/scripts/notify`:

```
#!/bin/sh

PATH=/bin:/sbin:/usr/bin:/usr/sbin

# Use the parhandclient application to get the IP address of the

# product and create an e-mail address from it

from=`parhandclient get root.Network.IPAddress - RAW`@somewhere.com


# Note that the body in the mail must be a valid file.

smtpclient -s "Notification" -S "10.13.178.7" -f $from -b
"/var/log/messages" "admin@somewhere.com"
```

Add this line to the `/etc/user.task.list` file to make the script execute at boot:

```
immune once % /usr/html/local/scripts/notify;
```

Example – e-mail triggered by an alarm

Send an e-mail to someone when an alarm occurs. Create a file that will be used as the body and send the directory `/var/run/buffer_handler/buffers/my_buf` as an attachment. Use the argument passed by `user.task.list` in the body. Access the argument as `$1`.

Source of `/usr/html/local/scripts/alarm`:

```
#!/bin/sh

PATH=/bin:/sbin:/usr/bin:/usr/sbin

# Use the parhandclient application to get the IP address of the
product and create an e-mail address from it

from=`parhandclient get root.Network.IPAddress - RAW`@somewhere.com


# Create the body file. Use the argument passed by the user.task.list

echo "Alarm on $1" > /tmp/body
```

```
smtpclient -s "Alarm" -S "10.13.178.7" -f $from -b "/tmp/body" -M 2 -d
"/var/run/buffer_handler/buffers/my_buf" "someone@somewhere.com"
```

Add this line to the `/etc/user.task.list` file. Add the camera that triggered the alarm as an argument to the script:

```
pattern((IO0:/)) immune once % /usr/html/local/scripts/alarm : CAM0;
```

sockclient

Syntax

```
sockclient [-timeout <time in seconds>] [-out <file>] -message
<message> socket
```

Write a message to a socket and wait until a response is read or `timeout` seconds have elapsed. The response is sent to `stdout` (default) or to `-out`, which must be a valid file.

Options

```
-timeout <secs>
```

> 0	Wait timeout seconds for response to be read.
= 0	Do not wait for a response.
< 0	Blocks on read.

The default timeout value is `1` second. Fractions of a second are allowed as timeout values.

Example – using sockclient to trigger a task

Using `sockclient` you can trigger a task defined in `user.task.list`. Add this line to `/etc/user.task.list`:

```
pattern((my_event:/)) once immune %
/usr/html/local/scripts/my_triggered_event;
```

Restart the product to make it re-read the configuration file.

You can trigger the task using a script (`/usr/html/local/scripts/my_trigger`):

```
sockclient -message "my_event:/;" -timeout 0
/var/run/utask/utasksocket
```

When the script `/usr/html/local/scripts/my_trigger` is invoked, it will send the message `my_event:/` to `utask`, which will recognize the pattern and start the script `my_triggered_event`.

Axis Communications AB provides NO support for application development of any kind. The information here is provided "as is", and there is no guarantee that any of the examples shown will work in your particular application.

statusled

Syntax

```
statusled [off|green|red|yellow|flash [INTERVAL]|init [INTERVAL]|usage
[INTERVAL]]
```

Set the status-LED.

The indicator flashes and briefly displays orange during start-up and self-test routines. The indicator then displays green to indicate a healthy unit status. Red will be displayed only if a problem has occurred.

Event handling

utask, task.list and user.task.list

`utask` is a task scheduler, capable of starting tasks based on either time of day or external events (for instance when a digital input becomes active), or a combination of these. A task scheduled by `utask` has to be an executable script or binary in the local filesystem. This can for example be used to start buffering images, to upload files via FTP or SMTP and much more.

utask

During startup, `utask` reads its configuration files and parses them for event entries. If the configuration files do not exist when `utask` starts, `utask` goes into standby mode.

Whenever the configuration files have been created or modified, you have to restart `utask` to make it re-read these files. This can be done with the `killall –USR1` command or by restarting the product.

Example – restart utask

In a `telnet` session, you restart the `utask` process and force it to re-read its configuration files with this command:

```
killall –USR1 utask
```

Configuration files

The `utask` application uses two configuration files, both in the `/etc` directory

- `task.list` - The `task.list` file is created, deleted or edited by the system. Do not use it for your own purposes, because it will be overwritten when you add/change/delete events via the product's Web interface.
  The action scripts for these events are stored in `/etc/event`. Since this directory is in the writable area you can modify these scripts if needed.
- `user.task.list` – This is the `utask` configuration file provided for user-defined tasks, to prevent your own entries from being overwritten.

Syntax

An entry in `user.task.list` has the following syntax.

```
{ID} <event> % <program> : <arguments>;
```

Where

`{ID}`	Identification string for the entry (optional). Can be used in combination with the built-in `*sig*` action (see below).
`<event>`	Event trigger condition.
`<program>`	Program (script or binary) to start.

`<arguments>`	Arguments to be passed to `<program>` (optional).

Note

If there are any conflicting entries in the configuration file, later commands will override earlier commands.

Event trigger conditions

An event trigger should contain one or more of the following definitions:

Event	Description
`[hh:mm-hh:mm]`	Time period, start–stop.
`[hh:mm+hhh:mm]`	Time period, start+duration. Max duration is `168:00` (7 days).
`time(h(..)m(..)s(..))`	
`[dd/mm-dd/mm]`	Date period.
`date(w(..)m(..)d(..))`	Date (day of week, month, date).
`pattern((..)(..)...)`	External events (e.g. alarms, motion detection).
`once`	Start the task only once, even if the trigger condition remains true.
`immune`	Never interrupt the program.
`not`	Invert the trigger condition.

Important

The default behaviour when an event is triggered is to run the program, exit and then start again, for as long as the triggering condition is true.

once

Specifying `once` tells `utask` to run the program only once as long as the trigger condition is true. `utask` checks for events ten times a second if not in standby mode. Therefore, if an event is specified to run at a certain time, e.g. `time(h(12))`, then in theory, this would run the program up to ten times per second for an hour, since the event becomes active for the entire hour.

The boot or start-up event is an always-true event and the `once` directive will prevent the script from being restarted over and over again.

immune

Default behaviour is that the task is run for as long as the trigger condition is true. Specifying `immune` will prevent the task from being prematurely killed if the trigger set has a very short duration. `utask` also starts a limited number of child processes.

When the number of child processes has been exceeded, it has to decide on how to proceed. `utask` usually kills the process with less priority (if the pending task has a higher priority than this process). The `immune` directive protects the child processes from being prematurely killed.

Date and time

Date and time values are entered as a list of values (separated by commas) and/or intervals (separated by hyphens).

Time/date	Values
`Hour`	`0-23`
`Minute`	`0-59`
`Seconds`	`0-59`
`Day of the week`	`0-6` (where `0` = Sunday).
`Month`	`1-12`
`Date`	`1-31`

Examples – date and time

Run `/etc/my_script` twice per hour between 8 am and 4 pm.

```
time(h(8-16)m(0,30)) once % /etc/my_script;
```

Run `/etc/my_script` at noon each day.

```
[12:00-12:01] once % /etc/my_script;
```

Run `/etc/my_script` every Sunday in November and December.

```
date(w(0)d(1-31)m(11,12)) once % /etc/my_script;
```

Start-up

The boot or start-up event will run the script when the product is started or when `utask` is restarted. Note that this is an always-true event.

Example – run at start-up

Run `/etc/my_script` once at start-up.

```
once % /etc/my_script;
```

Trigger patterns

utask is also able to schedule tasks based on messages that it receives on the `utask` socket. The location of this `utask` socket may differ from product to product, it can be located either in the `/tmp` directory or in the `/var/run/utask` directory.

Input

`IO<n>:<trigger>` where <n> is an integer, starting with `0`.

Triggers:

`/`	Virtual input is active.
`\`	Virtual input is inactive.

Examples:

`IO0:/` = trigger when virtual input 1 becomes active.
`IO1:\` = trigger when virtual input 2 becomes inactive.

Run `/etc/my_script` when digital input number 2 (= number `1` since numbering is zero based) becomes active:

```
pattern((IO1:/)) once immune % /etc/my_script;
```

Tip: An easy way to test that this is working is by activating the virtual input via HTTP. Enter this request in the address field of a browser:

```
http://192.168.0.90/axis-cgi/io/virtualinput.cgi?action=2:/
```

Run `/etc/my_script` when digital inputs number 1 and 2 become active at the same time:

```
pattern((IO0:/)(IO1:/)) once immune % /etc/my_script;
```

Boot

`BOOT:/`	Trigger on boot.

Video

`V<channel no>:<trigger>` where `<channel no>` can be `0 ... n`, denoting a single video channel (counted from `0`), or `x`, indicating any video channel.

Triggers:

`\`	Video lost.
`/`	Video is back.

Examples:

`V0:\` = trigger when video is lost on video channel 1.
`Vx:\` = trigger when video is lost on any video channel.

Run /etc/my_script when video is lost on camera 4 (= number 3 since numbering is zero based):

```
pattern((V3:\)) once immune % /etc/my_script;
```

Motion detection

```
M<motion window no>:<trigger>
```

Triggers:

/	Motion starts.
\	Motion stops.
x	Motion starts or stops.

Examples:

M1:/ = trigger when motion is detected in motion window 2.
M1:x = trigger when motion starts or stops in motion window 2.

Start the motion detection daemon at start-up:

```
once immune % /bin/sockclient : -message “START my_script”
/var/run/motion/requestsocket;
```

Run /etc/my_script when motion is detected in window number 3 (= number 2 since numbering is zero based):

```
pattern((M2:/)) once immune % /etc/my_script;
```

The motion detection daemon is not started by default, it is only active when there is an application using it. Therefore it is not possible to trigger on motion detection without first starting the daemon. To start the motion daemon, see example. The message contains the START command and also a string used to identify the daemon. The identifier can later be used to stop the daemon in combination with the STOP command, see example below, as well as under the not command.

PTZ preset position

P<PTZ preset param>:<state> where <PTZ preset param> is the last part of the name of the preset position parameter (i.e. PTZ.PresetPos.**P#**).

/	PTZ is at the preset position.
\	PTZ is leaving the preset position.

Examples:

PP1:/ = trigger when at the PTZ position which is stored in the parameter PTZ.PresetPos.P1.

Axis Communications AB provides NO support for application development of any kind. The information here is provided "as is", and there is no guarantee that any of the examples shown will work in your particular application.

`PP2:\` = trigger when leaving the PTZ position which is stored in the parameter `PTZ.PresetPos.P2`.

IR cut filter

`IR0:<state>`

`\`	IR cut filter is turned on
`/`	IR cut filter is turned off

Examples:

`IR0:\` = trigger when the IR cut filter is turned on.
`IR0:/` = trigger when the IR cut filter is turned off.

Tampering

`T<no>:<state>` where `<no>` is the camera number, starting with `0`.

Example:

`T0:/` = trigger when tampering is detected for camera 1.

Audio

`A0:<state>`

`/`	Audio rises above level specified in `Audio.A0.AlarmLevel`.
`\`	Audio falls below level specified in `Audio.A0.AlarmLevel`.
`x`	Audio passes level specified in `Audio.A0.AlarmLevel`.

Example:

`A0:/` = trigger when audio rises above level specified in `Audio.A0.AlarmLevel`.

Temperature

`TEMP0:<state>`

Example:

`TEMP0:/` = trigger when temperature is outside (i.e. above or below) the temperature range of the product.

Combined events

An event can be combined with other events.

Example:

Run `/etc/my_script` when input connector 1 becomes active between 5 pm and 8 am on Monday to Friday and on all hours on Saturday and Sunday.

```
date(w(1-5)) [17:00-08:00] pattern((IO0:/)) once immune %
/etc/my_script;
```

```
date(w(0,6)) pattern((IO0:/)) once immune % /etc/my_script;
```

not

The `not` directive will invert the trigger condition. If more than one event is set to trigger the event (for example: at May 1 and at 8 o'clock and when input connector 1 becomes active), adding `not` will trigger the event if it is not May 1 or it is not 8 o'clock or input connector 1 becomes inactive.

However, specifying `not` will not invert the directions `once` and `immune`.

Example:

Set output `1` high if motion is detected between `08:00` and `19:00`, otherwise set output 1 low.

```
[08:00-19:00] pattern((M0:/)) once immune % /bin/iod : -script 1:/;
```

```
[08:00-19:00] pattern((M0:/)) once immune not % /bin/iod : -script
1:\;
```

Example – stopping the motion detection daemon

`not` can also be used to stop the motion detection daemon if motion detection should not be performed 24 hours a day.

Run the motion detection daemon between 8 am and 5 pm and trigger on motion detection in window 1 during the same time.

```
[08:00-17:00] once immune % /bin/sockclient : -message “START
my_script” /var/run/motion/requestsocket;
```

```
[08:00-17:00] once immune not % /bin/sockclient : -message “STOP
my_script” /var/run/motion/requestsocket;
```

```
[08:00-17:00] pattern((M0:/)) once immune % /etc/my_script;
```

Specifying task to run

When specifying a task to run, its full path has to be specified.

Example – task to run

Use the logger application (located in `/usr/bin`), to write a message to the log file when video is lost on camera 1.

```
pattern((V0:\)) once immune % /usr/bin/logger : “Video lost on camera
1”
```

Note

The logger application is useful to check that the triggers you have set up work as expected.

Example – make a task executable

The file to execute (a script or a binary) must be executable. In a `telnet` session, the `chmod` command is used to set executable permissions to a file (`/etc/my_script` in this example):

```
chmod 755 /etc/my_script
```

Check that the file has correct permissions:

```
ls -l /etc/my_script
```

This should result in an output like this:

```
-rwxr-xr-x 1 root     root     4 May 07 09:23 my_script
```

Built-in actions

It is possible to send a signal (as an integer) to any process started by `utask` with the identification `{ID}`. The default signal value is 15 (SIGTERM).

Example – send signal to a process

Send the signal SIGUSR1 (`10`) to `{my_script}` every 5 minutes.

```
{my_script} once % /etc/my_script;
time(m(0,5,10,15,20,25,30,35,40,45,50,55)) % *sig* {my_script} : 10;
```

Passing arguments

Arguments can be passed to the program after a colon.

Sample user.task.list

```
# /etc/user.task.list - User defined tasks for utask.

# Example 1: Write a message to syslog at noon each day.
#[12:00-12:01] once % /usr/bin/logger : "It's noon";

# Example 2: Write a message to syslog when the message hello:world is
# received on the utask socket.
#pattern((hello:world)) once % /usr/bin/logger : "Hello world!";

# Example 3: Control output 1 with input 1.
#pattern((IO0:/)) once immune % /bin/iod : -script 1:/;
```

```
#pattern((IO0:\)) once immune % /bin/iod : -script 1:\;
```

Example – trigger a task in user.task.list

To activate the task in Example 2 in the sample `user.task.list`, first uncomment the line (by removing the #) to make it read

```
pattern((hello:world)) once % /usr/bin/logger : "Hello world!";
```

In a `telnet` session, force the `utask` process to re-read its configuration files (using the `killall –USR1` command) and then trigger the script (using `sockclient`):

```
killall –USR1 utask
```

```
sockclient -message "hello:world;" -timeout 0 /var/run/utask/utasksocket
```

This line will be added to `/var/log/messages`:

```
<INFO    > May 10 10:54:21 axis-00408c7ccb28 root: Hello world!
```

Event descriptions

Default texts for event descriptions and event values are defined in the file `/etc/event_desc.list`. The texts are defined per event.

Sample event_desc.list

```
#
# event_desc.list
#

[EVENT_DESCRIPTIONS]
videoloss = Video loss on video %d
boot = Restarting
manual = Manual trigger input port(s) %s went %s
motion = Motion '%s' on motion window '%s'
input = Input port(s) %s went %s
preset = Preset position '%s' on camera %d was reached

[EVENT_VALUES]
active = active
inactive = inactive
high = high
low = low
start = started
stop = stopped
```

Text variables

Text variables that can be used (together with plain text) for dynamic event notifications:

`%a`	Abbreviated weekday name.
`%A`	Full weekday name.
`%b`	Abbreviated month name.
`%B`	Full month name.
`%c`	Date and time representation.
`%C`	Century number (year/100) as a 2-digit integer.
`%d`	Day of month as a decimal number (range 01 to 31).
`%D`	Equivalent to `%m/%d/%y`.
`%e`	As `%d`, but leading zeros are replaced by a space.
`%f`	1/100 second fraction.
`%G`	ISO 8601 year with Century as a decimal number. The 4-digit year corresponding to the ISO week number (see `%V`). This has the same format and value as `%y`, except that if the ISO week number belongs to the previous or next year, then that year is used instead.
`%g`	As `%G`, but without Century, i.e. with a 2-digit year (00-99).
`%h`	Equivalent to `%b`.
`%H`	Hour as a decimal number, using the 24-hour clock (range 00 to 23).
`%I`	Hour as a decimal number, using the 12-hour clock (range 01 to 12).
`%j`	Day of the year as a decimal number (range 001 to 366).
`%k`	Hour (24-hour clock) as a decimal number (range 0 to 23); single digits are preceded by a space (also see `%H`) .
`%l`	Hour (12-hour clock) as a decimal number (range 1 to 12); single digits are preceded by a space (also see `%I`).
`%m`	Month as a decimal number (range 01 to 12).
`%M`	Minute as a decimal number (range 00 to 59).
`%n`	New line character.

%p	AM or PM, according to the given time value (or the corresponding strings for the current locale). Noon is treated as PM and midnight as AM.
%P	As %p, but in lowercase: am or pm (or a corresponding string for the current locale).
%r	Time in am or pm notation.
%R	Time in 24-hour notation (%H:%M). For a version including seconds, see %T.
%s	Number of seconds since 1970-01-01 00:00:00 UTC.
%S	Second as a decimal number (range 00 to 61).
%t	Tab character.
%T	Time in 24-hour notation (%H:%M:%S).
%u	Day of the week as a decimal, range 1 to 7, Monday = 1 (also see %w).
%U	Week number of the current year as a decimal number, range 00 to 53, starting with the first Sunday as the first day of week 01 (also see %V and %W).
%V	ISO 8601:1988 week number of the current year as a decimal number, in the range 01 to 53, where week 1 is the first week that has at least 4 days in the current year, and with Monday as the first day of the week (also see %U and %W).
%w	Day of the week as a decimal, in the range 0 to 6. Sunday = 0 (also see %u).
%W	Week number of the current year as a decimal number, in the range 00 to 53, starting with the first Monday as the first day of week 01.
%x	Date representation without time.
%X	Time representation without date.
%y	Year as a decimal number without the century (range 00 to 99).
%Y	Year as a decimal number, including the Century.
%z	Time zone as hour offset from GMT. Required to emit RFC822-conformant dates (using "%a, %d %b %Y %H:%M:%S %z").
%Z	Time zone or name or abbreviation.

Axis Communications AB provides NO support for application development of any kind. The information here is provided "as is", and there is no guarantee that any of the examples shown will work in your particular application.

`%%`	The `%` character.
`#s`	Sequence number.
`#I`	IP address of the product.
`#m`	Short MAC address (last 6 characters).
`#M`	Long MAC address (all characters).
`#n`	Host name.
`#v`	Video source number.
`#b`	Bit rate in kbps.
`#B`	Bit rate in Mbps.
`#r`	Frame rate (2 decimals).
`#R`	Frame rate (no decimals).
`##`	The # character.

Creating custom Web pages and CGI scripts

All HTML-files that you store in the `/usr/html/local` directory will be available through a directory, named `local`:

```
http://192.168.0.90/local/<filename>
```

Check memory available

Adding custom Web pages to your product will use some memory. To check that you have sufficient resources, open a `telnet` session to your product and run the `df` command to see how much flash memory is available, followed by the `free` command to see how much RAM memory is available.

Local Web directories

In `/usr/html/local` you will find three subdirectories, each with its access rights set for the user group in question:

- `administrator`
- `operator`
- `viewer`

User groups

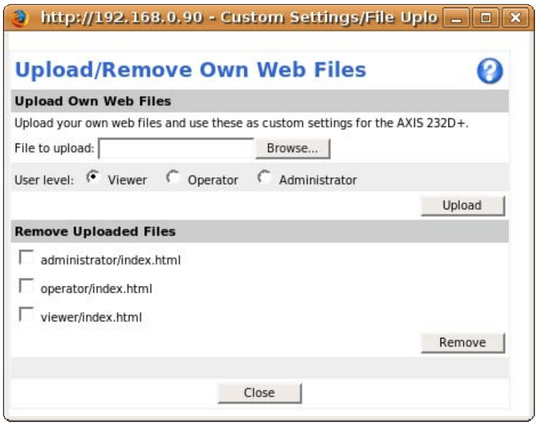
User groups are administered via the Live View & Configuration page of the product (`Setup | Basic Configuration | Users`). By selecting a certain group you define the access rights for a user; where the level of access can be described as follows:

- `administrator` - An administrator has unrestricted access to the whole product and its setup tools.
- `operator` - An operator can view the video stream, create and modify event types and modify all settings not included in the System options.
- `viewer` – A viewer has access to the video stream only.

Upload/remove your own Web files

A practical way to manage your own Web files is through the Live View & Configuration page of the product (`Setup | Live View Config | Layout | Use custom settings | Configure`, then click the `Upload/Remove`… button).

Axis Communications AB provides NO support for application development of any kind. The information here is provided "as is", and there is no guarantee that any of the examples shown will work in your particular application.



Upload/Remove Own Web Files

Upload Own Web Files

Upload your own web files and use these as custom settings for the AXIS 232D+.

File to upload:

User level: ● Viewer ○ Operator ○ Administrator

Remove Uploaded Files

□ administrator/index.html

□ operator/index.html

□ viewer/index.html

Here, we have uploaded three HTML-files, one in each directory.

Use own home page

When you have uploaded a Web file, you can make it the default home page using the Live View & Configuration page (`Setup | Live View Config | Layout | Use custom settings | Configure`). Select which page to use.

These pages are addressed like this:

`http://192.168.0.90/local/adminstrator/index.html`

`http://192.168.0.90/local/operator/index.html`

`http://192.168.0.90/local/viewer/index.html`

When you use a home page of your own, the product’s normal Web pages will be available through:

`http://192.168.0.90/operator/basic.shtml`

`http://192.168.0.90/view/index.shtml`

Configuring the Web server

An additional way to alter the default Web behaviour of the product is to change the configuration of the internal Web server, `/etc/httpd/conf/boa.conf`, preferably by using the built-in editor. The product has to be restarted for the changes to be applied.

Note

The built-in Web server, boa, is single threaded. For this reason, commands that are run by boa cannot be invoked by a CGI script. For example, `shttpclient` cannot be used within a script run by boa.

Change default start page

The Live View & Configuration Web pages of the product are located in `/usr/html`. By default, the password protected page `/usr/html/index.shtml` is invoked when someone is browsing to

Axis Communications AB provides NO support for application development of any kind. The information here is provided "as is", and there is no guarantee that any of the examples shown will work in your particular application.

```
http://192.168.0.90
```

To change the default Web page of the product to a custom Web page (for instance named `my_index.html`, located in `/usr/html/local`), add this line to the Alias section in `/etc/httpd/conf/boa.conf`:

```
Alias  /   /usr/html/local/my_index.html
```

Save `boa.conf` and re-start the product. From now on, `my_index.html` will be the default Web page of the product.

Several Alias directives may appear in the configuration file.

With the default Web page changed in this manner, you reach the Live View & Configuration Web page of the product by using its expanded address:

```
http://192.168.0.90/index.shtml
```

Referring to locations

When referring to files in the `/usr/html/local/` directory in HTML code, use the address

```
/local/<your_file>
```

When you want to refer to files resident in the `/usr/html` directory (which is the directory we use for the product's own Web pages), use the address

```
/<Axis_file>
```

Example – customized my_index.html

```
<html>

<head>

</head>

<body>

<h1>Welcome!</h1>

<h3>my_index.html</h3>

<a href="/local/my_index.html">Go to customized Web page</a></p>

<br/>

<a href="/index.shtml">Go to original Live view &amp; Configuration
page</a>

</body>

</html>
```

Examples – local references

In these examples, the Web page is located in `/usr/html/local/`.

Refer to an image resident in `/usr/html/local/pics/`:

```
<img src="/local/pics/my_picture.gif">
```

Axis Communications AB provides NO support for application development of any kind. The information here is provided "as is", and there is no guarantee that any of the examples shown will work in your particular application.

Refer to an image resident in `/usr/html/pics/`:

```
<img src="/pics/help.gif">
```

To refer to a file resident in a directory outside the `/` or `/local` paths, create a symbolic link to that directory in the `/usr/html/local` directory. Example: To refer to a file resident in the `/tmp` directory, start by creating a symbolic link to the `/tmp` directory (use `telnet` for this):

```
cd /usr/html/local
ln -s /tmp tmp
```

Now you can use this reference in your HTML code:

```
<img src="/local/tmp/snapshot.jpg">
```

Run scripts through an HTTP request

Shell scripts can also be invoked by an HTTP request.

Enable script requests in boa.conf

Create a directory for the scripts and set a ScriptAlias for it in `/etc/httpd/conf/boa.conf`.

Example: You want to store your scripts in the `/usr/html/local/scripts` directory. Open `boa.conf` and add this line to the ScriptAlias section:

```
ScriptAlias  /scripts/  /usr/html/local/scripts/
```

Make sure that the files in the scripts directory are executable by setting their file mode to `0100755` in the built-in editor, or by using FTP or `telnet` to perform a

```
chmod 755 my_memory_check
```

Several ScriptAlias directives may appear in the configuration file.

Save `boa.conf` and re-start the product.

Scripts that return data

If you want the script to produce any output, it has to send proper HTTP headers back to the caller. Here is an example script (it is located in `/usr/html/local/scripts/` and it is named `my_memory_check`):

```
#!/bin/sh

echo -e "Cache-Control: no-cache\r"
echo -e "Pragma: no-cache\r"
echo -e "Expires: Thu, 01 Dec 1994 16:00:00 GMT\r"
echo -e "Content-Type: text/html\r"
echo -e "\r"
echo
echo "<html><body><h1>Memory check</h1><pre>"
```

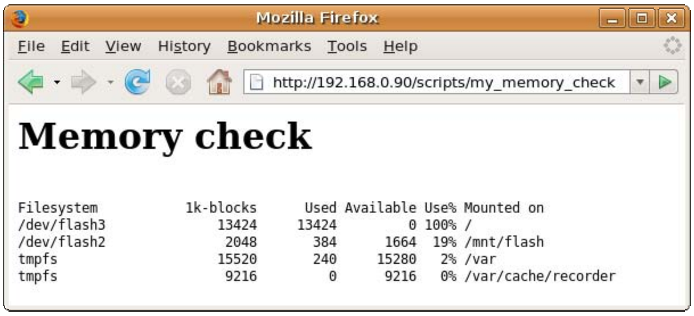
Axis Communications AB provides NO support for application development of any kind. The information here is provided "as is", and there is no guarantee that any of the examples shown will work in your particular application.

```
df
echo "</pre></body></html>"
```

To invoke the script, browse to

```
http://192.168.0.90/scripts/my_memory_check
```

Output:


Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://192.168.0.90/scripts/my_memory_check

Memory check

```
Filesystem           1k-blocks      Used Available Use% Mounted on
/dev/flash3              13424     13424         0 100% /
/dev/flash2               2048       384      1664  19% /mnt/flash
tmpfs                    15520       240     15280   2% /var
tmpfs                     9216         0      9216   0% /var/cache/recorder
```

Scripts that expect arguments

You can also pass arguments to your scripts. Let us assume you would like to pass two arguments to a script, process them, and send the result back as a Web page.

Since you want the script to produce an output, it has to send proper HTTP headers back to the caller (we assume it is located in `/usr/html/local/scripts/` and that the script is named `hello_product`):

```
#!/bin/sh


echo -e "Cache-Control: no-cache\r"
echo -e "Pragma: no-cache\r"
echo -e "Expires: Thu, 01 Dec 1994 16:00:00 GMT\r"
echo -e "Content-Type: text/html\r"
echo -e "\r"
echo
echo "<html><body><h1>$1 $2!</h1></body></html>"
```

You invoke the script and pass the arguments `Hello` and `world` to it with this request:

```
http://192.168.0.90/scripts/hello_product?Hello+world
```

The + is used to separate the arguments in the request. The arguments will be accessed as $1 and $2 in the script, in the same order as they were defined in the request. Output:

```
Hello world!
```

Using AXIS VAPIX API in your scripts

In your scripts you can also use the functionality in AXIS VAPIX, considered to be the global application programming interface (API) standard in the rapidly growing field of Network Video based on IP. For full information, see `http://www.axis.com`.

Example – restart button

Include on a Web page a button that restarts the product. According to the AXIS VAPIX specification, the syntax to restart the product programmatically is

```
http://192.168.0.90/axis-cgi/admin/restart.cgi
```

and the method to use for the Web form is

```
get
```

This information makes it possible for you to write an HTML page with a button to perform the same task:

```
<html>

<head>

<meta http-equiv="Expires" content="0">

<meta http-equiv="Pragma" content="no-cache">

<meta http-equiv="Cache-Control" content="no-cache">

</head>

<body>

<form name="formRestart" action="/axis-cgi/admin/restart.cgi"
method="get">

<input type="Submit" value="Restart">

</form>

</body>

</html>
```

Example – button to reload factory default settings

Include on a Web page a button that reloads the factory default settings (all parameters – except the IP address – will be set to their original factory settings). According to the AXIS VAPIX specification, the syntax to reload the factory default settings programmatically is

```
http://192.168.0.90/axis-cgi/admin/factorydefault.cgi
```

and the method to use for the Web form is

```
get
```

This information makes it possible for you to write an HTML page with a button to perform the same task:

```
<html>
```

```
<head>
<meta http-equiv="Expires" content="0">
<meta http-equiv="Pragma" content="no-cache">
<meta http-equiv="Cache-Control" content="no-cache">
</head>
<body>
<form name="formFactoryDefault" action="/axis-
cgi/admin/factorydefault.cgi" method="get">
<input type="Submit" value="Reload factory default settings">
</form>
</body>
</html>
```

Using Server Side Includes

You can use all Server Side Includes that are embedded in the product as well as add custom ones.

For example, by utilizing the server manager (`sm.srv`) you can access the parameters in your product and by including the server manager in your shtml file you can get and set parameters via custom Web pages.

Note that Web pages using SSI must have an `.shtml` extension. Example: `my_get_parameters.shtml`.

The parameter list

For parameters available in your product, see its parameter list.

The parameter format in the server manager is

`root_<parameter group>_<parameter name>`

It is important that you use this format when naming the objects on the Web page.

Note!

Always make sure that the parameter you are setting has a valid value.

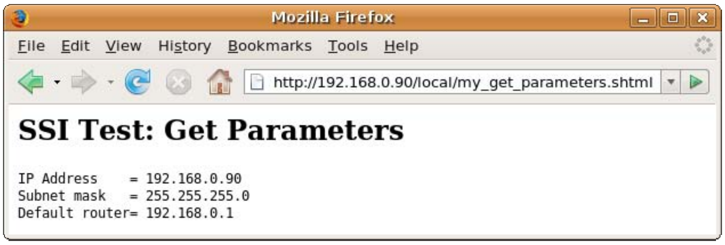
To view the parameter list, browse to `http://192.168.0.90`, select `Setup | System Options | Support | Logs & Reports | Information` and click on `Parameter List`.

Example – get parameters

Get some parameters in the `Network.eth0` and `Network.Routing` groups (IP address, subnet mask and default router) and show their values in a custom Web page, named `my_get_parameters.shtml`:

```
<html>

<head>

<meta http-equiv="Expires" content="0">

<meta http-equiv="Pragma" content="no-cache">

<meta http-equiv="Cache-Control" content="no-cache">

<!--#include virtual="/sm/sm.srv?action=get&group=Network.eth0.IPAddress,Network.eth0.SubnetMask,Network.Routing.DefaultRouter" -->

</head>

<body>

<h2>SSI Test</h2>

<pre>

IP Address     = <!--#echo var="root_Network_eth0_IPAddress" -->

Subnet mask    = <!--#echo var="root_Network_eth0_SubnetMask" -->

Default router= <!--#echo var="root_Network_Routing_DefaultRouter" -->

</pre>

</body>

</html>
```

Output:


Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://192.168.0.90/local/my_get_parameters.shtml

SSI Test: Get Parameters

```
IP Address     = 192.168.0.90
Subnet mask    = 255.255.255.0
Default router= 192.168.0.1
```

Example – set parameters using the server manager

Create a Web page with the ability to set some of the Network parameters:

```
<html>

<head>

<!--#include virtual="/sm/sm.srv?action=get&group=Network" -->

</head>
```

Axis Communications AB provides NO support for application development of any kind. The information here is provided "as is", and there is no guarantee that any of the examples shown will work in your particular application.

```
<body>
<form name="form" action="/sm/sm.srv " method="post">
IP Address: <input type="text" name="root_Network_IPAddress"
value="<!--#echo var="root_Network_IPAddress" -->"><br>
<input type="submit" value="Save">
<input type="hidden" name="action" value="modify">
</form>
</body>
</html>
```

Example – set parameters using embedded param.cgi

This example sets the `FriendlyName` parameter to `my_cam`:

```
http://192.168.0.90/axis-
cgi/admin/param.cgi?action=update&Network.UPnP.FriendlyName=my_cam
```

Example – redirection

The server manager can also direct the browser to another page:

```
<html>
<head>
<meta http-equiv="Expires" content="0">
<meta http-equiv="Pragma" content="no-cache">
<meta http-equiv="Cache-Control" content="no-cache">
</head>
<body>
<form name="form" action="/sm/sm.srv" method="post" id="Form1">
<input type="HIDDEN" name="return_page" value="/local/my_index.html"
id="Hidden1">
<input type="SUBMIT" value="Click to redirect" id="Submit1"
name="Submit1">
</form>
</body>
</html>
```

Sample scripts

Enable/disable scripts using a CGI script

If you have a script that is triggered by events you can use a CGI script to enable/disable the script with a simple HTTP request.

In this example we store the script permanently in non-volatile memory (i.e. in the `/usr/html/local/scripts` directory) to make it remain even after the product is restarted.

We will use these components:

1. A CGI script, named `script_enable_disable`, located in the `/usr/html/local/scripts` directory.
2. An alarm script, named `my_alarm_script`, located in the `/usr/html/local/scripts` directory.
3. Two new tasks in the `/etc/user.task.list`.
4. A status file `/tmp/enable_status` to signal present enable/disable status.

Note

Ensure that script requests have been enabled for the `/usr/html/local/scripts` directory. See the section Enable script requests in boa.conf.

Create the enable/disable CGI script

Using the built-in editor, create the enable/disable CGI script in the `/usr/html/local/scripts` directory. Name it `script_enable_disable` and make it executable.

```
#!/bin/sh

echo -e "Cache-Control: no-cache\r"
echo -e "Pragma: no-cache\r"
echo -e "Expires: Thu, 01 Dec 1994 16:00:00 GMT\r"
echo -e "Content-Type: text/html\r"
echo -e "\r"
echo

if [ $1 = enable ]; then
   echo "on" > /tmp/enable_status
elif [ $1 = disable ]; then
   echo "off" > /tmp/enable_status
```

Axis Communications AB provides NO support for application development of any kind. The information here is provided "as is", and there is no guarantee that any of the examples shown will work in your particular application.

```
fi


echo "<html><body><h1>The alarm script is now $1d</h1></body></html>"
```

The script will write the string on to the status file `/tmp/enable_status` when you call it with the argument `enable`. When called with the argument `disable`, the script will write the string `off` to the status file. This on/off flag is read by the alarm script (see below).

Create the alarm script

Create an alarm script to send a TCP alarm when triggered (but only if the script is enabled). Name it `my_alarm_script`, save it in the `/usr/html/local/scripts` directory and make it executable.

```
#! /bin/sh


enabled=`cat /tmp/enable_status`
# If the script is enabled send a TCP alarm
if [ $enabled = on ]; then
   echo "$1 triggered" | nc 10.93.127.77 4444
fi
```

Set the tasks

Using the built-in editor, open `/etc/user.task.list` and add two new tasks (one to enable the script at startup, one to invoke the alarm script when triggered):

```
# enable the alarm script at startup
once immune % /usr/html/local/scripts/script_enable_disable : enable;
# run the alarm script when the first input connector is triggered
pattern((IO0:/)) once immune % /usr/html/local/scripts/my_alarm_script
: "IO 1";
```

Remember to restart `utask` to make it read the new settings. This can be done with the

```
killall –USR1 utask
```

command or by restarting the product.

Enable the script with an HTTP request

To enable `my_alarm_script`, send this HTTP request:

```
http://192.168.0.90/scripts/script_enable_disable?enable
```

Disable the script with an HTTP request

To disable `my_alarm_script`, send this HTTP request:

Axis Communications AB provides NO support for application development of any kind. The information here is provided "as is", and there is no guarantee that any of the examples shown will work in your particular application.

```
http://192.168.0.90/scripts/script_enable_disable?disable
```

Axis Communications AB provides NO support for application development of any kind. The information here is provided "as is", and there is no guarantee that any of the examples shown will work in your particular application.

Tips & tricks

General information

- The more video streams you open, the less RAM memory will be available.
- Scripts consume much resources from the product and from the video streams.
- Study the scripts in the product; much of the code can be re-used for other tasks. See, for example, how these scripts are built up:

  ```
  /usr/html/axis-cgi/lib/functions.sh
  /usr/html/axis-cgi/admin/systemlog.cgi
  /usr/html/axis-cgi/admin/lib/systemlog.sh
  /usr/html/axis-cgi/admin/uptime.cgi
  /usr/html/axis-cgi/admin/lib/uptime.sh
  ```

Visit the Axis developer pages, available at `http://www.axis.com`, for FAQs, wiki, cookbook and scripting assistance.

How to trigger an alarm (e.g. to start an event)

The input connector can be triggered by virtual input according to AXIS VAPIX:

```
http://192.168.0.90/axis-cgi/io/virtualinput.cgi?action=1:/
```

Also see the `sockclient` command for methods triggering an event using `telnet` or a script.

Debugging methods

Debug information

When debugging a script, there are some flags built into the shell which can be of great assistance.

- x – the echo flag
- v – the verbose flag
- e – the exit flag

These flags can be set (with a `-`, e.g. `–x`) and unset (with a `+`, e.g. `+x`) and they can be combined (e.g. `–xv`).

Example script:

```
#!/bin/sh

echo "my_script is running"
```

When this script is executed without any flags

```
sh my_script
```

it will report

```
my_script is running
```

x – the echo flag

When the echo flag is set, every line of the script will be echoed to `stdout` before the line is executed (but after the line has been evaluated).

Invoke the script with the echo flag set:

```
sh –x my_script
```

and it will report

```
+ echo my_script is running

my_script is running
```

v – the verbose flag

With the verbose flag set, every line of the script will be echoed to `stdout` before the line has been evaluated.

e – the exit flag

With the exit flag set, the script will immediately terminate when an error occurs.

It may often be useful to save the output to a file, to be able to debug it later.

```
#!/bin/sh -e

exec 2>/tmp/${0##*/}.debug

set –x

...
```

This header will ensure that all executed lines will be saved in a file in the `/tmp` directory. The name of the file will be the same as the name of the script with `.debug` as suffix.

The set command

To trace long scripts, the `set` command is useful, since it permits you to turn on/off flags inside a script. Example script:

```
#!/bin/sh

set +x

echo "line 1 unset at start"

echo "line 2 unset at start"

set -x

echo "line 3 set"

echo "line 4 set"

set +x

echo "line 5 unset again"
```

Axis Communications AB provides NO support for application development of any kind. The information here is provided "as is", and there is no guarantee that any of the examples shown will work in your particular application.

```
echo "line 6 unset again"
```

When executed, the script will produce this output:

```
line 1 unset at start

line 2 unset at start

+ echo line 3 set

line 3 set

+ echo line 4 set

line 4 set

+ set +x

line 5 unset again

line 6 unset again
```

Log debugging information

With the `logger` command you can make entries in the system log. For more information and examples, see the `logger` section.

Axis Communications AB provides NO support for application development of any kind. The information here is provided "as is", and there is no guarantee that any of the examples shown will work in your particular application.

Troubleshooting

Script related problems

Axis Communications AB does not provide support for application development of any kind. The information here is provided "as is", and there is no guarantee that any of the examples shown will work in your particular application.

Axis' Application Development Partner (ADP) Program

To receive additional support, consider joining the Axis' Application Development Partner (ADP) Program. The ADP Program assists partners to fully integrate Axis network cameras and video products in end-user solutions by providing technical information, development support and application components.

For information, see `http://www.axis.com`.

Product related problems

If you run into problems, please try these steps:

1. Consult the troubleshooting guide: browse to the product, go to `Setup | System Options | Support | Support Overview` and click `Guide`.
2. Visit the Axis product support Web, available at `http://www.axis.com`, and verify that the product contains the latest available software version. Updated troubleshooting information and the latest software for the product can also be found there.
3. Consult the FAQ and technical notes on the Axis product support page for additional help.
4. The Server Report can prove a useful diagnostic tool when attempting to resolve problems. Browse to the product, go to `Setup | System Options | Support | Support Overview` and click `Server Report`. Always attach the server report when contacting Axis Support.
5. Contact your local supplier, i.e. where the product was purchased, for assistance.

Reference

Links to general Linux information

For general Linux information, see The Linux Documentation Project at `http://www.tldp.org`. There you will find a wide range of Howtos, Guides, FAQs and man pages.

Filesystem overview

This section describes the file and directory structure in a typical Axis Camera and Video Server. It is not a complete description of the directory structure, but rather a quick guide to what-is-where.

For a full description of the Linux Filesystem Hierarchy Standard, see `http://www.pathname.com/fhs/`.

The root directory

The following directories belong in the root directory.

`/bin`	Essential command binaries.
`/dev`	Device files.
`/etc`	Host-specific system configuration. Linked to `/mnt/flash/etc`.
`/lib`	Shared libraries and kernel modules.
`/mnt`	Mount point for temporarily mounted filesystems and devices (network, JFFS).
`/proc`	Kernel and process information virtual filesystem.
`/root`	Home directory for the superuser.
`/sbin`	System binaries.
`/sys`	Kernel and process information virtual filesystem.
`/tmp`	Temporary files. Linked to `/var/tmp`.
`/usr`	Shareable, read-only data, e.g. installation/administration Web pages, system scripts (initialization, flashing, etc) and parameter handler configuration.
`/var`	Variable data files (spool directories and files, administrative and logging data, transient and temporary files).

Axis Communications AB provides NO support for application development of any kind. The information here is provided "as is", and there is no guarantee that any of the examples shown will work in your particular application.

Directories you would normally use

`/etc/conf.d` Contains configuration scripts.

`/etc/event` Contains event scripts.
Default texts for event descriptions are defined in the file `/etc/event_desc.list`. See the Event handling section.

`/etc/httpd/conf` Contains Web server configuration files.
`/etc/httpd/conf/boa.conf` is the configuration file for the Web server. For additional information and examples, see the section Creating custom Web pages and CGI scripts.
`/etc/httpd/conf/mime.types` defines which file types that will be recognized.

`/etc/sysconfig` Contains parameter configuration files.

`/mnt/flash` Mount point for the flash memory.
If you need to store own binaries (non-volatile) in the product, make a `bin` directory for them here. Example:
`mkdir /mnt/flash/bin`

`/sbin` Contains system binaries, e.g. files that are needed for flashing the product, such as `rc.factorydefault`, `rc.fsupgrade` and `rc.translateparams`.

`/tmp` Contains temporary data (RAM disk, volatile), like images captured by the camera.

`/usr` Contains shareable, read-only data, such as the installation/administration Web pages, system scripts (initialization, flashing, etc) and the configuration files for the parameter handler.

`/usr/etc/sysconfig/template`
Contains templates for dynamic product parameters.

`/usr/html` Contains the product's internal Web pages.

`/usr/html/local` Directory for custom Web pages. See the section Creating custom Web pages and CGI scripts.

The shell

The shell is the command processor or interpreter that reads and executes commands in the product. A shell is also a programming language in which scripts can be written for the shell to interpret. Hence they are known as shell scripts. Shell scripts allow users to customize their environments by adding their own commands. When a user logs in, a default shell will be called up (i.e. the login shell).

Given the combination of flexibility and relatively small memory requirements, shell scripts are very well suited to performing less complex tasks in embedded devices such as the product.

The shell is a POSIX compliant shell that aims at being as small as possible. For most tasks, it is significantly faster than, for instance, Bash.

The commands available are essentially the same as the ones commonly used in other shells on any Linux system, so the normal manual pages can often be quite useful. The difference is that the commands here have been simplified, i.e. some options have been removed. However, some additional commands have also been added to the version you will find in your product.

When writing shell scripts, begin the first line of the script with the sequence

```
#!/bin/sh
```

The statement after `#!` (i.e. shebang or hashbang) specifies the correct interpreter to use for the code in your script.

When a shell script is created, the script will not automatically be executable. To make a script executable, the file permission must be changed, by using the command chmod.

For more information on shell handling, see
`http://www.linuxcommand.org/man_pages/sh1.html`

For information about Bash, see Bash tutorial at `http://www.freeos.com/guides/lsst/` and the advanced Bash scripting guide at `http://www.tldp.org/LDP/abs/html/`.

Built-in shell commands

```
sh, :, ./, break, case, cd, continue, eval, exec, exit, export, for,
if, read, readonly, set, shift, test, times, trap, umask, wait, while
```

`sh` is a command language interpreter that executes commands read from the standard input or from a file.

Syntax

```
sh [-vx] [file]
```

Options

`-v`	Echo input lines as they are read.
`-x`	Trace.

Example

```
sh script       #Run a shell script
```

On startup, the shell reads `/etc/profile` and `$HOME/.profile`, if they exist, and executes any commands they contain.

Common shell notations

Some common shell notations are (excerpt):

`date`	Run a regular command (`date`, in this case).
`date > file`	Redirect `stdout` (standard output) to `file`.
`date >> file`	Append output to `file`.
`sort < file`	Redirect `stdin` (standard input), i.e. read the command input from `file`.
`sort < file | wc`	Two-process pipeline.
`ls -l *.c`	List all files ending in `.c`.
`v=/usr/ast`	Set shell variable `v`.
`ls -l $v`	Use shell variable `v`.
`echo $PATH`	Echo the search path.
`echo $#`	Echo number of arguments (shell script).
`echo $2`	Echo second argument (shell script).

Shell built-in commands

The shell has a number of built-in commands (excerpt):

`cd dir`	Change current working directory.
`pwd`	Print the name of the current working directory.
`shift`	Shift positional parameters to the left.

Shell flow control

The shell also contains a programming language, which has the following operators and flow control statements:

`#`	Comment. The rest of the line is ignored.
`=`	Assignment. Set a shell variable.
`&&`	Logical AND. Execute second command only if first succeeds.
`||`	Logical OR. Execute second command only if first fails.
`test`	Evaluate conditional expressions.
`(...)`	Group. Execute enclosed commands before continuing.
`for`	For loop (`for... in... do... done`).

`case`	Case statement (`(case... )... ;;... esac`).
`esac`	Case statement end.
`while`	While loop (`while... do... done`).
`do`	Do/For/While loop start (`do... until...`).
`done`	For/While loop end.
`if`	Conditional statement (`if... else... elif... fi`).
`in`	For loop selection.
`then`	Conditional statement start.
`else`	Conditional statement alternative.
`elif`	Conditional statement end.
`until`	Do loop end.
`fi`	Conditional statement end.

Using variables

As is the case with almost any language, the use of variables is very important in shell scripts. You can assign a value to a variable simply by typing the variable name followed by the equal sign and the value you want to assign to the variable. For example, if you want to assign a value of 5 to the variable `count`, enter

```
count=5
```

You do not have to declare the variable as you would if you were programming in C or Pascal. This is because the shell language is a non-typed interpretive language. This means that you can use the same variable to store character strings that you use to store integers.

Once you have stored a value in a variable, how do you get the value back out? You do this in the shell by preceding the variable name with a dollar sign ($). If you want to print the value stored in the `count` variable to the screen, enter the following command:

```
echo $count
```

If you omit the `$` from the preceding command, the `echo` command would simply display the word `count`.

Built-in shell variables

The shell is aware of a special kind of variable called positional parameters. Positional parameters are used to refer to the arguments that were passed to the shell program on the command line or a shell function by the shell script that invoked the function.

When you run a shell program that requires or supports a number of command line options, each of these options is stored in a positional parameter. The first argument is stored in a variable named 1; the second argument is stored in a variable named 2, and so on. The shell reserves these variable names so that you cannot use them as variables defined by you. To access the values stored in these variables, precede the variable name with a dollar sign ($), just as you do with variables you define.

Symbols

`$?` Exit value from last executed command.

`$$` Process id number of the shell.

`$!` Process number of the most recent asynchronously executed command.

`$-` Flags that were passed to the shell when it was invoked or flags that were set using the set command.

`$#` Number of arguments to the shell.

`$*` Current argument list. By itself `$*` is equivalent to `$1, $2` and so on, up to the number of arguments. The construct `"$*"` is equivalent to `"$1, $2 ..."`.

`$@` Argument list. By itself, `$@` is equivalent to `$1, $2` and so on up to the number of arguments. The construct `"$@"` is equivalent to `"$1", "$2"...`, which preserves the argument list. Without quotes, `$@` splits arguments containing spaces into separate arguments.

The importance of quotation marks

The use of the different types of quotation marks is very important in shell programming. The double quotation mark (`"double"`), the single quotation mark (`'single'`), and the backslash (`\`) are all used to hide special characters from the shell, while the back quote (`` `back` ``) is used when you want to use the output from a command. See below.

Double quotes

Double quotation marks (`" "`) are the weakest quotes. When you surround characters with double quotes, all white space characters are hidden from the shell, but all other special characters are interpreted. This type of quoting is most useful when you are assigning strings that contain more than one word to a variable. For example, if you want to assign the string `"hello world"` to the variable `hello`, you would use the following command:

```
hello="hello world"
```

This command would store the string `"hello world"` into the variable `hello` as one variable.

Single quotes

Single quotes (`' '`) are the most powerful form of quotes. They protect all special characters from the shell, which is useful if the command you enter is intended for a program instead of the shell (i.e. the program is to interpret the characters, not the shell).

Using single quotes:

```
greeting='hello there $LOGNAME'

echo $greeting
```

would produce:

```
hello there $LOGNAME
```

Using double quotes:

```
greeting="hello there $LOGNAME"

echo $greeting
```

would produce:

```
hello there root
```

Backslash quotes

Backslash quoting is used when you want to protect a single character from the shell. For example, if you wanted to store the price of a box of computer disks into a variable named `disk_price`, you would use the following command:

```
disk_price=\$5.00
```

Here, the backslash will protect the dollar sign from being converted by the shell. If the backslash was not there, the shell would try to find a variable named 5 and perform a variable substitution on that variable. If no variable named 5 was defined, the shell would assign a value of `00` to the `disk_price` variable, since the shell would substitute a value of `null` for the non-existent $5 variable.

Back quotes

Back quotes (`` ` ` ``) perform a different function. They are used when you want to use the result of a command as input to another command. For example, if you want to set the value of the variable `contents` equal to the list of files in the current directory, use this command:

```
contents=`ls`
```

The test command

Syntax

```
test expression
```

or

```
test [ expression ]
```

Axis Communications AB provides NO support for application development of any kind. The information here is provided "as is", and there is no guarantee that any of the examples shown will work in your particular application.

Evaluate conditional expressions. You would typically use the `test` command to evaluate a condition that is used in a conditional statement, or to evaluate the entry or exit criteria for an iteration statement.

Several built-in operators can be used with the `test` command. These operators can be classified in four groups: integer operators, string operators, file operators, and logical operators.

Integer operators

`int1 -eq int2`	True if `int1` is equal to `int2`.
`int1 -ge int2`	True if `int1` is greater than or equal to `int2`.
`int1 -gt int2`	True if `int1` is greater than `int2`.
`int1 -le int2`	True if `int1` is less than or equal to `int2`.
`int1 -lt int2`	True if `int1` is less than `int2`.

String operators

`str1 = str2`	True if `str1` is identical to `str2`.
`str1 != str2`	True if `str1` is not identical to `str2`.
`str`	True if `str` is not null.
`-n str`	True if the length of `str` is greater than zero.
`-z str`	True if the length of `str` is equal to zero.

File operators

`-d filename`	True if `filename` is a directory.
`-f filename`	True if `filename` is an ordinary file.
`-r filename`	True if the process can read `filename`.
`-s filename`	True if `filename` has a non-zero length.
`-z filename`	True if the process can write `filename`.
`-x filename`	True if `filename` is executable.
`-e filename`	True if `filename` exists.

Logical operators

`! expr`	True if `expr` is not true.
`expr1 -a expr2`	True if `expr1` and `expr2` are true.
`expr1 && expr2`	True if `expr1` and `expr2` are true.
`expr1 -o expr2`	True if expr1 or `expr2` is true.
`expr1 || expr2`	True if `expr1` or `expr2` is true.

Conditional statements

if

Syntax

```
if [ expression ];
then
   commands
elif [ expression2 ];
then
   commands
else
   commands
fi
```

The `elif` and `else` clauses are both optional parts of the `if` statement. The `elif` statement (which is an abbreviation of else if) is executed only if none of the expressions associated with the `if` statement or any `elif` statements before it are true.

The commands associated with the `else` statement are executed only if none of the expressions associated with the `if` statement or any of the `elif` statements are true.

Example

This script expects one argument given on the command line:

```
#!/bin/sh
if [ $# -ne 1 ]; then
   echo "Please supply one argument."
else
   echo "Thank you!"
fi
```

Axis Communications AB provides NO support for application development of any kind. The information here is provided "as is", and there is no guarantee that any of the examples shown will work in your particular application.

case

Syntax

```
case string in
   str1)
   commands;;
   str2)
   commands;;
   *)
   commands;;
esac
```

The `case` statement enables you to compare a pattern with several other patterns and execute a block of code if a match is found.

`string` is compared to `str1` and `str2`. If either `str1` or `str2` matches `string`, the commands up until the double semicolon (`;;`) are executed. If neither `str1` nor `str2` matches `string`, the commands associated with the asterisk (`*`) are executed. This is the default case condition since `*` matches all strings.

Example

This script looks at argument `1`:

```
#!/bin/sh
case $1 in
   "a") echo "You supplied argument a";;
   "b") echo "You entered argument b";;
   *) echo "Argument unknown";;
esac
```

for

Syntax

```
for var1 in list
do
   commands
done
```

or

```
for var1
do
```

```
   statements
done
```

The `for` statement executes the commands that are contained within it a specified number of times.

In the first form, the `for` statement executes once for each item in `list`. This list can be a variable that contains several words separated by spaces, or it can be a list of values that is typed directly into the statement. Each time through the loop, the variable `var1` is assigned the current item in the list, until the last one is reached.

Example, first form

The script

```
#!/bin/sh
for i in 1 2 3
do
   echo $i
done
```

when invoked as

```
sh script
```

would produce

```
1
2
3
```

Example, second form

The script

```
#!/bin/sh
for i
do
   echo $i
done
```

would expect the list to come from the command line:

```
sh script 1 2 3
```

Output:

```
1
2
3
```

Axis Communications AB provides NO support for application development of any kind. The information here is provided "as is", and there is no guarantee that any of the examples shown will work in your particular application.

while

Syntax

```
while [ expression ];
do
   statements
done
```

Another iteration statement offered by the shell programming language is the `while` statement. This statement causes a block of code to be executed while the expression is true.

Example

```
#!/bin/sh
while test $1
do
   echo "Argument = $1"
   shift
done
```

The script prints all arguments supplied on the command line.

until

Syntax

```
until [ expression ]
do
   commands
done
```

The `until` statement is very similar in syntax and function to the `while` statement. The only real difference between the two is that the `until` statement executes its code block while its conditional expression is false, and the `while` statement executes its code block while its conditional expression is true.

In practice the `until` statement is not very useful, since any `until` statement you write can also be written as a `while` statement.

Example

```
#!/bin/sh
until test -z $1
do
   echo "Argument = $1"
```

Axis Communications AB provides NO support for application development of any kind. The information here is provided "as is", and there is no guarantee that any of the examples shown will work in your particular application.

```
   shift
done
```

The script prints all arguments supplied on the command line.

shift

Syntax

```
shift [n]
```

Shift positional parameters to the left. For example, if the values of the current positional parameters are

```
$1=-r $2=file1 $3=file2
```

and you execute the `shift` command

```
shift
```

the resulting positional parameters will be:

```
$1=file1 $2=file2
```

You can also move the positional parameters more than one step by specifying number of steps with the `shift` command. The following would shift the positional parameters two steps:

```
shift 2
```

This is a very useful command when you have a shell program that needs to parse command line options. This is true because options are typically preceded by a hyphen and a letter that indicates what the option is to be used for. Because options are usually processed in a loop of some kind, you often want to skip to the next positional parameter once you have identified which option should be coming next.

Example

```
#!/bin/sh
my_counter=1
while test $1
do
   echo "Argument $my_counter = $1"
   shift
   my_counter=`expr $my_counter + 1`
done
```

The script prints all arguments supplied on the command line.

Embedded commands

The product comes with many commands that you can use when writing scripts of your own.

To save valuable space in the product, most of the general commands are available through BusyBox. BusyBox combines tiny versions of many common Linux utilities into a single, small executable. It provides minimalist replacements for most of the utilities you usually find in fileutils, shellutils, findutils, textutils, grep, etc. BusyBox provides a fairly complete POSIX environment for the product.

The utilities in BusyBox generally have fewer options than their full-featured GNU cousins; however, the options that are included provide the expected functionality and behave very much like their GNU counterparts.

BusyBox commands

The command `busybox` will show which BusyBox commands that are embedded in your product.

Note

The commands available are firmware and product dependent and Axis cannot guarantee that these commands will be available in future firmware or products.

Listed below in alphabetical order are Firmware 4.40 BusyBox commands. For a full description, syntax, options and examples, see `http://www.busybox.net`.

`basename`	Strip directory path and suffixes from file. If specified, also remove any trailing suffix.
`cal`	Display a calendar.
`cat`	Concatenate files and print to `stdout`.
`chgrp`	Change the group membership of each file to group.
`chmod`	Change file permission.
`chown`	Change owner and/or group of each file to owner and/or group.
`cmp`	Compare files.
`cp`	Copy source to dest, or multiple sources to directory.
`cut`	Print selected fields from each input file to standard output.
`date`	Display the current time in the given format, or set the system date.
`dc`	This is a tiny RPN calculator that understands the following operations: `+`, `add`, `-`, `sub`, `*`, `mul`, `/`, `div`, `%`, `mod`, `**`, `exp`, `and`, `or`, `not`, `eor`.
`dd`	Copy a file, convert and format it according to options.

`df`	Print the filesystem space used and space available.
`dirname`	Strip non-directory suffix from filename.
`dmesg`	Print or control the system's message buffer.
`du`	Summarize disk space used for each file and/or directory. Disk space is printed in units of 1024 bytes.
`echo`	Print the specified argument(s) to `stdout`.
`env`	Print the current environment or run a program after setting up the specified environment.
`expr`	Print the value of expression to standard output.
`false`	Return an exit code of false (`1`).
`find`	Search for files in a directory hierarchy.
`free`	Display the amount of free and used system memory.
`grep`	Search for pattern in each file or standard input.
`gunzip`	Uncompress file.
`gzip`	Compress file(s) with maximum compression. When file is `'-'` or unspecified, read standard input. Implies `-c`.
`head`	Print first 10 lines of each file to standard output.
`hostname`	Get or set the hostname or DNS domain name. If a hostname is given (or a file with the `-F` argument), the host name will be set.
`id`	Print information for user name or the current user.
`ipcalc`	Calculate IP network settings from an IP address.
`kill`	Send a signal (default is SIGTERM) to the specified process(es).
`killall`	Send a signal (default is SIGTERM) to the specified process(es).
`ln`	Create a link named link_name or directory to the specified target.
`logger`	Write message to the system log. If message is omitted, log `stdin`.
`login`	Begin a new session on the system.
`ls`	List directory contents.
`md5sum`	Print or check MD5 checksums.

`mkdir`	Create directory if it does not already exist.
`mkfifo`	Create a named pipe.
`mknod`	Create a special file (block, character, or pipe).
`mktemp`	Create a temporary file with its name based on a template.
`more`	View a file one screenful at a time.
`mount`	Mount a filesystem. Without arguments, mounted filesystems are shown.
`mv`	Rename source to dest, or move source(s) to directory.
`nc`	Open a pipe to ip:port.
`netstat`	Display Linux networking information.
`pidof`	List the PIDs of all processes with names that match the names on the command line.
`ping`	Send ICMP ECHO_REQUEST packets to network hosts.
`ping6`	Send ICMPv6 ECHO_REQUEST packets to network hosts.
`ps`	Report process status.
`readlink`	Display the value of a symbolic link.
`renice`	Change priority of running processes.
`rm`	Remove file.
`rmdir`	Remove directory if it is empty.
`sed`	Edit a stream, i.e. to perform basic text transformations on an input stream (a file or input from a pipeline).
`sleep`	Pause for n seconds.
`sort`	Sort lines of text in specified file(s).
`stty`	Show/change terminal settings.
`sync`	Write all buffered filesystem blocks to disk.
`tail`	Print last 10 lines of each file to standard output.
`tar`	Manipulate (create, extract or list) tar archive files.

`tee`	Copy standard input to each file, and also to standard output.
`test`	Check file types and compare values returning an exit code determined by the value of expression.
`top`	View processor activity in real time.
`touch`	Update the last-modified date on the given file(s).
`tr`	Translate, squeeze, and/or delete characters.
`true`	Return an exit code of true (`0`).
`umount`	Unmount filesystems.
`uname`	Print certain system information.
`uniq`	Discard all but one of successive identical lines from input (or standard input), writing to output (or standard output).
`uptime`	Display the time since the last boot.
`usleep`	Pause for n microseconds.
`vi`	Edit file.
`wc`	Print line, word, and byte counts for each file, and a total count if more than one file is specified.
`which`	Locate a command.
`who`	Print the current user name(s) and related information.
`xargs`	Execute command on every item given by standard input.
`zcat`	Uncompress to stdout.