Scripting in Axis Network Cameras and Video Servers

Table of Contents

1 Introduction

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

The document provides numerous examples of ready-made scripts that were written for the most common applications. These scripts can be used for your own purposes, after making the changes required by your particular application.

IMPORTANT NOTICES!

Axis Communications AB provides **no** guarantee that any of the examples shown in this document will work for any particular application.

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

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

Please bear in mind that the flash chip manufacturer estimates the number of writes to the flash chips to about 100,000. Writing a lot of temporary files to the flash memory should thus be avoided. Use the ram disk mounted on `/tmp` instead.

2 Embedded Scripts

The embedded scripts that can be used in Axis camera and video server products can be of 2 different types: **shell scripts** or **PHP3 scripts**. These scripts can be used for many different purposes and can, for example, start the buffering of images or the uploading of files via FTP or SMTP.

Scripts can be run in several different ways. One example is when the built-in Task Scheduler in the Axis products is used to start programs or scripts when certain events occur. Another example is when a script is run via a web page, for e.g. controlling a pan-tilt unit.

The Axis products that currently support the use of these scripts are as follows:

Product	Firmware Version	Shell Support	PHP3 Support
AXIS 2100	2.30 or higher	Yes	No
AXIS 2120	2.30 or higher	Yes	No
AXIS 2420	2.12	Yes (from 2.20)	Yes
AXIS 2400	2.20	Yes	Yes
AXIS 2401	2.20	Yes	Yes

2.1 PHP

PHP (Hypertext Preprocessor) was chosen as one of the scripting languages to use in Axis camera/video products for the following reasons:

- PHP is a well known and widely used scripting language. In February 2001, an estimated 23% of all Internet web servers were using PHP. (The closest "competing" script language was Perl, which was used on about 7% of the servers).
- The language syntax closely resembles Java, Perl and C, making it easy to learn for anyone with basic programming skills.
- Version 3, PHP3, can easily be scaled down to the smaller footprint required for embedded devices.
- PHP allows the rapid creation of dynamic web pages; the PHP code is simply embedded into the HTML code. At the most basic level, PHP can do anything that any other scripting language or CGI program can do, such as collect HTML form data and generate dynamic page content.
- PHP is a complete scripting language, with functionality such as file operations, network sockets, an e-mail client (SMTP) and an FTP client, all of which are very useful in network camera/video server applications.

The combination of integrated event handling in Axis Video products and a powerful scripting language such as PHP, provides skilled developers with endless opportunities to tailor the functionality of Axis camera/video products to fit their specific needs.

Axis has removed some of the functions from the standard PHP3, and also created some additional functions to suit our products. The Axis modified version of PHP3 is called **PHP3-Lite**. Among the functions removed are those that handle:

- Database functions

- PDF functions
- Math functions

Axis recommends checking that PHP3-Lite supports the functions required before creating custom scripts. Consult the PHP3-Lite manual on our Web site at: www.axis.com

2.2 Shell

A shell is a *command processor* or *interpreter* that reads and executes the commands entered by the user. A shell is also a programming language in which programs can be written for the shell to interpret. These programs are therefore 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. This default shell is 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 Axis camera/video products.

In the latest release, Axis products use a modified Linux operating system called *ELinux*, which opens up more possibilities for the server. This is why the shell *mish* (the minix shell, with most of the features of the Bourne shell) was chosen as the script interpreter for Axis products.

3 Using scripts in Axis Camera/Video products

This document contain examples of complete scripts that can be used as they are, or after being modified to suit your own purposes. The scripts have been written for the most common applications. Some of these scripts require a `task.list` file, and these are also provided.

Scripts can be activated in several different ways:

- by the Task Scheduler, which handles input from alarms and motion detection events
- via Telnet
- via the web server

These different methods are all described in the sections that follow.

3.1 Uploading scripts to the Camera/Video Server:

A new script or a script created by the wizard and subsequently modified must first be uploaded to the camera/video server before it can be used. To do this, follow the instructions below:

1. Open a DOS session and open an FTP session from the path where the script file resides, e.g.: `c:\axis\scripts`.
2. Open an FTP session to the camera/video server: E.g.: `ftp <Server ip-address>`.
3. Then type the user name: e.g. `root`.
4. Then type the password: e.g. `pass`.
5. Change to the directory where you want to store the script: e.g. `cd etc/scripts`.
6. Set the transfer mode to text: type `ASCII`.
7. Then upload the file to the server with the command: put `<filename>`**.**
8. To make a shell script file executable: type `chmod 755 <filename>`. If that doesn't work, type instead `site chmod 755`.

Tip: A very useful program for fetching or uploading scripts to the product is *Ultra Edit*. This can be downloaded from http://www.ultraedit.com/

3.2 Running Scripts with the Task Scheduler

The Task Scheduler (i.e. `utask`) is used to start programs or scripts when events occur. The scripts can be shell-scripts or PHP3-scripts, and can start the buffering of images or the uploading of files via FTP or SMTP.

The Task Scheduler handles event detection based on date and time, alarm inputs, motion detection, can start any task and also handles process management.

At start-up, `utask` (the Task Scheduler) reads the configuration file `/etc/task.list` and parses it for event entries. If the file does not exist when `utask` is started, then `utask` goes into standby mode. When `/etc/task.list` has been created, or modified, a `SIGUSR1` needs to be sent to `utask` in order for it to re-read `/etc/task.list`. This can be done by issuing the command `kill -10 <utask-pid>`. (The same effect is also achieved by simply restarting the Axis Server).

3.2.1 Syntax for /etc/task.list

An entry in `/etc/task.list` has the following syntax:

```
<event>%<program>:<args>;
```

<event> = is the event description

<program> = is the program to start

<args> = are the arguments to be passed to <program>.

`<event>`

The maximum number of events is 64. An event description should contain one or more of the following definitions.

Definition	Description
`[hh:mm-hh:mm]`	Time period.
`time( h(...) m(...) s(...) )`	Time(hours, minutes, seconds).
`[dd/mm-dd/mm]`	Date period.
`date( w(...) m(...) d(...) )`	Date(day, month, date).
`pattern( (...) (...) ... )`	External events (e.g. alarms, motion detection).
`Once`	Start only once.
`immune`	Never interrupted.

Time and Date values are given as a list of values (separated by commas) and/or intervals (separated by minus signs):

Hour = 0-23
Minute = 0-59
Seconds = 0-59
Day of the week = 0-6 (0 = Sunday)
Month = 1-12
Date = 1-31.

Patterns are defined as strings and follow the specific format `<Client>:<Data>`;

`<Client>`
If the client application sends `<Data>` to `utask`, the Data must be atomic, i.e. if the client wants to notify two events at the same time, it must write:
`<Client>:<Data1>;<Client>:<Data2>;`.

Patterns are read by `utask` from `/tmp/utasksocket`. `Utask` can start a maximum of five (5) child processes. When the maximum number of child processes has been exceeded, `utask` must decide how to proceed. The process with the lowest priority (if the pending task has a higher priority than this process) will then be killed. The entries in the configuration file (`task.list`) are ordered by ascending priority, i.e. later entries precede earlier entries.

IMPORTANT!
The default behavior when an event is triggered is that the program runs, exits and starts again, for as long as the triggering event is valid.

`once`
Specifying `once` tells `utask` to skip starting the program as long as the event has not been invalidated. `Utask` checks for events ten times a second (if not in standby mode). Thus, if an event is specified to run at a certain point in time, e.g. `time(h(0) m(0) s(0))`, then, in theory, this would run the program up to ten times, since the event is active for a whole second.

`immune`
Child processes can be protected against premature killing by using the `immune` tag in the script entry. An immune script will always run until it terminates.

3.2.1.1 Trigger Patterns for /etc/task.list

Trigger Patterns for alarm input connectors:

`IO<n>:/`	(Rising edge)
`IO<n>:H`	(Input is high)
`IO<n>:\`	(Falling edge)
`IO<n>:L`	(Input is low)

Where n is the number of the alarm input used, depending on the product, `<n>={0,1,...,3}`

Trigger Patterns for boot

boot is defined by an "always true" event that is started once:

```
once % program : params ;
```

This will start the program only at startup or if `utask` is terminated and restarted.

Trigger Patterns for Motion Detection (2120/2420)

`MO<n>:/`	(Motion starts)
`MO<n>:H`	(Motion is detected)
`MO<n>:\`	(Motion stops)
`MO<n>:L`	(No motion)

with `<n>={0,1,2,3,4,5}` for the window identifier.

Trigger Patterns for Video Loss (2400/2401)

`V<n>:\`	(Video lost)
`V<n>:/`	(Video back)

with, depending on product, `<n>={0,1,...,3}`

3.2.1.2 Example of a task.list

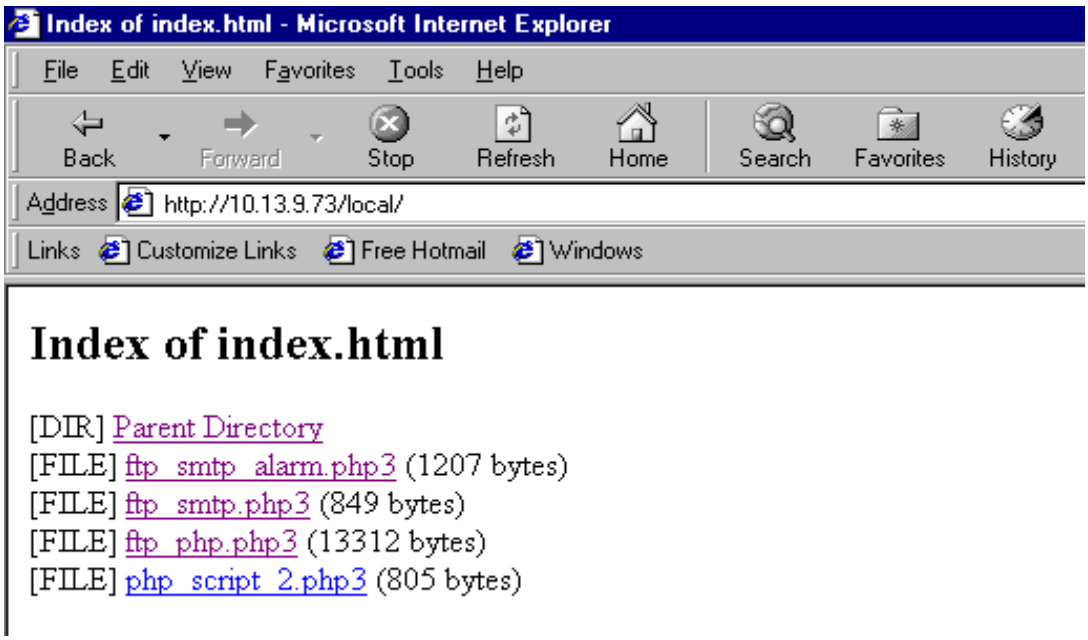
```
# Execute the alarm_ftp_net script if a positive transition is
detected on the Input
# Output 0 every day of the week
date(w(0,1,2,3,4,5,6)) pattern ((IO0:/)) immune once %
/etc/alarm_ftp_net : CAM1 IO0;

# FTP transfer once every 15 minutes, uninterrupted using
alarm_ftp_net script.
time(m(0,15,30,45)) immune once % /etc/alarm_ftp_net : CAM1 IO0;

# Sends an alert message every time I1 and I2 are high at the same
time...
pattern((IO1:H)(IO2:H)) once % alert : "Input 1 and 2 detected";
```

3.3 Running scripts via a web server

Running a script via a web server means that web pages containing dynamic content can be created. This is also a very useful way of debugging scripts that will later be triggered from the Task Scheduler.

Upload your PHP3 script to the server via ftp to the `/etc/httpd/html/` directory. This directory is reached via `/local` from the web server.
To run a script via the web server, follow the instructions below:

1. Open your browser.
2. Type the IP-address for the camera/video server in the URL: e.g.:
   `http://10.13.9.75/local`
3. If the file doesn't appear, click the browser's **Refresh** button. Click on the script to run it.



Running Scripts via Telnet

For development purposes, it may be convenient to connect to the camera/video Server via **Telnet**. Depending on which product is used, this is either enabled by default and uses authentication, or it can be enabled by editing the `/etc/inittab`. If the product requires editing of the `/etc/inittab` there will be no authentication for the Telnet connection and no password will be required for access.

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.

3.3.1 To enable Telnet support

Open an FTP session to the camera/video server and type the commands shown in bold below:

```
C:\Axis >ftp <ip address of video Server>
Connected to <ip address of video server>
220 Axis 2400 Video Server 2.20 Jul 27 2001 ready.
User (<ip address of Video Server>:(none)): root
331 User name okay, need password.
Password: pass (if not changed from default)
230 User logged in, proceed.
ftp> ascii
ftp> cd etc
ftp> get inittab
200 Command okay.
150 Opening data connection.
226 Transfer complete.
ftp: 1380 bytes received in 0,01Seconds 138,00Kbytes/sec.
```

Now open the downloaded `inittab` file in an editor and find the following line:

```
#telnetd:3:respawn:/bin/telnetd
```

Remove the preceding #:

```
telnetd:3:respawn:/bin/telnetd
```

Save the file.

Go back to the ftp session and continue:

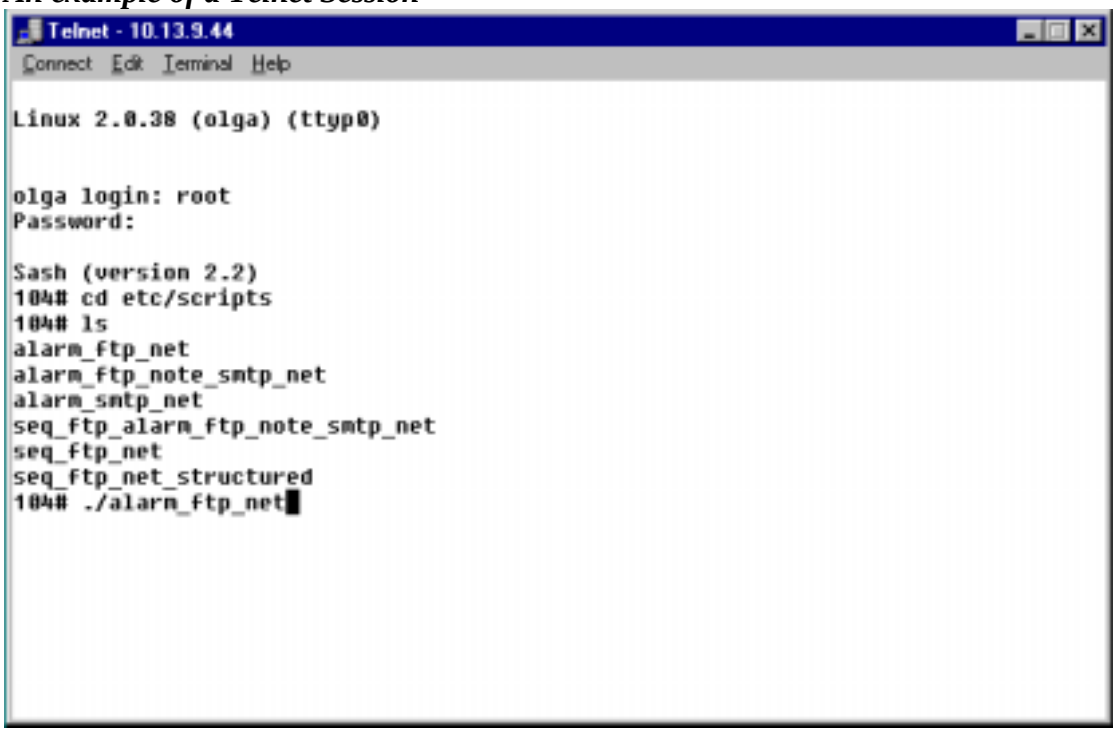
```
ftp> put inittab
200 Command okay.
150 Opening data connection.
226 Transfer complete.
ftp: 1414 bytes sent in 0,00Seconds 1414000,00Kbytes/sec.
250 Command successful.
ftp> bye
221 Goodbye.
```

Restart the video server and it will now be accessible via Telnet. To activate the script via Telnet, follow the instructions below.

For a shell script: activate the script by typing e.g. `./<filename>`

For a PHP3 script: type e.g. `php <filename>`

An example of a Telnet Session



3.4 Included Helper Applications

This section explains some of the applications shipped with the Axis camera/video servers.

Note: **A good general rule is NOT to use a function if it is not fully understood!**

3.4.1 The image buffer - `bufferd`

The application `bufferd` captures images and stores them on the ram-disk in a FIFO-order, i.e. the latest image overwrites the oldest.
By default, `initd` starts a `bufferd` as a daemon. This daemon is listening for messages on the socket `/tmp/bufferdsocket`. Consequently, starting several daemons is not recommended, as this would result in one daemon overwriting what the others write.

The preferred way of using `bufferd` is depicted in the example below. When a call to `bufferd` is made, the started process writes messages to the socket `/tmp/bufferdsocket` and then dies.

Depending on which options are used when starting a buffer, `bufferd` starts one or two processes for handling image capturing. These processes capture a number of images according to the argument given.

3.4.1.1 Options

The valid parameters for `bufferd` are:

Option	Description
`-start`	Start a buffer.
`-reset`	Remove a buffer.
`-stop`	Stop a buffer.
`-d`	Start as a daemon. This option should only be used when starting up from `initd`, or when the first instance is to be started.
`-buffername <buffername>`	The name of the buffer on which the options `-start`, `-stop` and `-reset` should act. `bufferd` will create a directory in `/tmp` named `<buffername>`, in which the images will be stored.
`-uri <request-string>`	The request-uri. Specification of the image-format as described in the **HTTP-API specification** (pdf-document). The protocol should be FTP instead of HTTP. This document can be found on www.axis.com, on the Camera & Video developer pages.
`-postdelay <delay>`	Delay between images in milliseconds.
`-predelay <delay>`	Same as `–postdelay`, but sets an additional variable.
`-pre <number>`	Number of pre-alarm images, i.e. number of images to save before stopping the buffer.
`-post <number>`	Number of post-alarm images, i.e. the number of images to save after stopping the buffer.
`-format <format-string>`	Naming convention for the files.
`-snapshot`	Take one picture.
`-immediate`	Stops the buffer without waiting for it to complete. Only used in conjunction with `–stop`.

3.4.1.2 Default Values & Settings

If the option `-buffername` is omitted, the buffer used defaults to `buffer (/tmp/buffer)`.

The protocol specification in the request-uri should be `ftp`, and not `http`, as this does not generate http-specific headers.

The default delays (`postdelay` and `predelay`) are one(1) second (1000 milliseconds).

The default number of images before and after an alarm (pre and post) is 10.

The format-option specifies how the files are named.

Character expansion is supported with:

- `%y` (year)
- `%m` (month)
- `%d` (date)
- `%H` (hour)

- %M (minute)
- %S (second).

These correspond to the date and time the file was created.

Also supported is %i, which specifies the file number, and which also can be followed by a number specifying how many digits are allowed.
The default format is snapshot_%y%m%d_%H%M%S_%i4.

3.4.1.3 Examples

The first example shows how to take one (1) image with camera one (1). The resulting image will be of the size 352x288 (the resolution and other attributes depend on the product). The image will be stored in /tmp/SNAP/ and named snapshot.

```
bufferd -start -buffername SNAP -snapshot -pre 1 -format
snapshot -uri ftp://axis-cgi/jpg/1/352x288.jpg
```

The next example can be used in a PHP3-script, and is equivalent to the example above.

```
function shoot() {
$command = "bufferd -start -buffername SNAP";
$command .= " -snapshot -pre 1";
$command .= " -format snapshot";
$command .= " -uri ftp://axis-cgi/jpg/1/352x288.jpg ";
system($command);
}
```

The following example creates a buffer that saves its images to the directory /tmp/ALARM/, where the files are named according to the creation time. The captured images in this case will be 176x144 and are captured every two (2) seconds by the first camera. Five (5) images before and two (2) images after the buffer is stopped are saved.

```
bufferd -start -buffername ALARM -uri ftp://axis-
cgi/jpg/1/176x144.jpg -pre 5 -post 2 -postdelay 2000
```

This example shows how to stop the buffer started above.

```
bufferd -stop ALARM
```

This example resets (i.e. stops buffering and clears the directory) the buffer specified in the last example.

```
bufferd -reset ALARM
```

Let's say a method of buffering images is required, and when an arbitrary event occurs, an alarm is read. The images taken will be handled and the buffering should then continue. The following example illustrates the steps to take (using the examples above):

```
bufferd -start -buffername ALARM -uri ftp://axis-
cgi/jpg/1/176x144.jpg -pre 5 -post 2 -postdelay 2000
```

(This starts the buffer and can be specified in a start-up script or as an event or `utask`) When an event occurs, as specified for `utask` (*see below*) the following command sequence is used to handle the buffer:

```
bufferd -stop ALARM
{ handle images}
bufferd -reset ALARM
bufferd -start -buffername ALARM -uri ftp://axis-
cgi/jpg/1/176x144.jpg -pre 5 -post 2 -postdelay 2000
```

The handling of the images can, for example, be done with the PHP3-function `ftp()`, shown below (*see also* `/usr/php/template_upload_cont.php3` *on the device*).

3.4.2 sftpclient

This client is used for FTP connections. If you have an FTP server, you can use this client to send/receive files to/from a specified server. A passive mode is also available for the client. A typical example is to upload snapshots to a specified FTP server when an event is detected.

Usage: `sftpclient [options]`

Option	Description
`-p <host>`	Put local file to host remote dir [remote file].
`-i <host>`	Interactive to/from host [remote dir].
`-m <host>`	Put all files in local dir to host remote dir.
`-n <port nbr>`	Specify remote port number, default is 21.
`-t <file>`	Use temporary file name during upload.
`-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.
`-u username`	The user to login as.
`-w password`	The password to use for the user.
`-L`	Log errors to syslog facility instead of stderr.
`-s`	Use passive mode ftp.

Example

Upload all the files in `/tmp` to the directory `/upload` on the FTP Server 10.13.9.40 using the port 21, with the username `user` and the password `pass`. Errors will be logged to `syslog`.

```
sftpclient -L -m 10.13.9.40 -n 21 -c /upload -k /tmp -u user -w pass
```

3.4.3 smtpclient

This client is used for SMTP connections. You can send e-mail with attached files and specify a range of common parameters, i.e.; subject, address of the sender for the replies, address to send copy to , etc. A typical example is to send an e-mail with snapshots when an alert is detected.

Usage: `smtpclient [options] recipients...`

Option	Description
`-s <string>` *	Subject line of message.
`-f <addr>` *	Address of the sender.
`-r <addr>`	Address of the sender for replies.
`-e <addr>`	Address to send delivery errors to.
`-c <addr>`	Address to send copy of message to.
`-a <file>`	The file to attach. Binary MIME attachment, if no mime encode is set, MIME-style is set to `1 = application/octet.`
`-d <directory>`	Binary MIME directory attachment, if no mime encode is set, MIME-style is set to `1 = application/octet.` The entire contents of the directory are uploaded.
`-b <file>` *	Read mail body from file. (`CRLF` must be used as the line feed).
`-S <host>`	Host where MTA can be contacted via SMTP.
`-P <num>`	Port where MTA can be contacted via SMTP.
`-M`	Use MIME-style translation to quoted-printable `1=application/octet, 2=image/jpeg.`
`-L`	Log errors to syslog facility instead of `stderr`.

* = Required

Example

Send an e-mail entitled `Alert` from `someone@somewhere.com` via the SMTP server `mail.somewhere.com` to the recipient `somebody1@somewhere.com.` Also send a copy to `somebody2@somewhere.com.` The body will be read from the file `/tmp/body` and the files included in the directory `/tmp/SNAP1` will be attached to the mail.

```
smtpclient -s Alert -S $mail.somewhere.com -f someone@somewhere.com -c
somebody2@somewhere.com -b /tmp/body -M 2 -d /tmp/SNAP1
somebody1@somewhere.com
```

3.4.4 shttpclient

This client is used for HTTP connections. Its main uses are;

- for sending alarm notifications to a remote web server by simply accessing a URL with a CGI-script
- for uploading images/files via HTTP
- for sending dynamic information such as when a IP-number is received via DHCP

The `shttpclient` supports basic authentication for web servers and proxy servers.

Usage: `shttpclient [options] url`

Option	Description
`-i`	Input_file
`-o`	Output_file.
`-u`	The user
`-w`	The user's password.
`-x`	The address of the proxy server
`-n`	The port to use on the proxy server.
`-a`	The user name to use for the proxy server.
`-b`	The password the user must supply to use the proxy server.

Examples

Fetch a single live image from the camera webserver and store it on the same camera as `/tmp/snap.jpg`

```
shttpclient -o /tmp/snap.jpg http://127.0.0.1/axis-cgi/jpg/image.cgi
```

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

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

Send an alarm with CGI information (alarm=dooralarm) in the URL.

```
shttpclient -u username -w mypass
'http://www.somewhere.com/cgi-bin/trigger.cgi?alarm=dooralarm'
```

`shttpclient` Tips:

- Use the local host **http://127.0.0.1** 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 the HTTP-API for Axis cameras/video servers.

3.4.5 statusled

Usage: `statusled <color>`

Set the status-led color, where `<color>` is one of the following:

- `'off'`
- `'green'`
- `'yellow'`
- `'red'`

The indicator flashes briefly and briefly displays orange during the 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.

4 An Introduction to PHP3

PHP3 is a server-side HTML-embedded scripting language. A simple answer, but what does this mean?

An example:

```
1
2 <html>
3     <head>
4         <title>Example</title>
5     </head>
6     <body>
7         <?PHP echo "Hi, I'm a PHP script!"; ?>
8     </body>
9 </html>
10
```

Notice how this is different from a CGI script written in other languages such as Perl or C. Instead of writing a program with lots of commands to output HTML, you write an HTML script with some embedded code to do something (in this case, output some text). The PHP code is enclosed in special *start* and *end tags* that allow you to jump into and out of PHP mode.

What distinguishes PHP from e.g. client-side JavaScript is that the code is executed on the server. If you were to have a script similar to the one above on your server, the client would receive the results of running that script, with no way of determining what the underlying code looks like.

A lot of information about PHP is available on the Internet, for example, at **http://www.php.net**.

4.1 The PHP-libs

Included in the distributed firmware are a few PHP3-scripts that define functions intended to help developers make the most of the camera/server's functionality. These scripts are located in the directory `/usr/php`. The files currently included are:

```
/usr/php/alert.lib
/usr/php/ftp.lib
/usr/php/log.lib
/usr/php/mail.lib
/usr/php/ppp.lib
```

In order to use these functions the files need to be included in the script using them. The arguments used in the function calls also need to be defined/declared in the calling script.

4.1.1 alert.lib

This script implements a function that connects to a host and leaves a message. This function requires an application that can handle the alert to be running on the contacted host.

```
function alert($host,$protocol,$port,$message)
{
...
}
```

Parameter	Description
`$host`	(not null) The host-name or address to send the alert to.
`$protocol`	Specify the protocol (0 = TCP, 1 = UDP). Default is 0.
`$port`	The port to connect to. Default is 15.
`$message`	the message to be sent (ASCII).

The timeout for a connection-attempt is 60 seconds. On error, the function returns 1 (connection failed), 2 (a parameter was missing) or `$SOCK_CONNECT_FAILED`. If the connection fails, this is logged in the `syslog`.

4.1.2 ftp.lib

This function has encapsulated the built-in PHP3-functions for FTP.

```
function
ftp($host,$user,$pass,$time,$delay,$source,$destination,$suffix,$count
ermax,$startcount,$port,$passive_mode)
{
...
}
```

Parameter	Description
`$host`	(not null) FTP Host name.
`$user`	(not null) User name.
`$password`	(not null) Password.
`$time`	Transfer period in seconds (time < 0 is infinite (i.e. never leave this function), time >= 0 is once).
`$delay`	Delay between transfers in milliseconds.
`$source`	(not null) Source specification. If the source file is a directory, the command will try to transfer each file in this directory. When transferring a whole directory, time and delay are ignored.
`$destination`	(not null) Destination specification.
`$suffix`	Defines the type of stamp to be added to the filename. Valid options are: `default`, `date`, `sequence` and `sequence_max`.
`$countermax`	Defines the suffix maximum index. Only applies when `$suffix = "sequence"`.
`$startcount`	Defines start value for the suffix index. Only applies when `$suffix = ("sequence"|"sequence_max")`.
`$port`	Port on the FTP-server to connect to.

Parameter	Description
`$passive_mode`	Choose passive mode. `on` or `off`. In passive mode, data connections are initiated by the client, rather than by the server.

On errors, the function returns one of the following:

- 1 (connection failed)
- 2 (login failed)
- 3 (upload failed)
- 4 (parameter error)
- 5 (could not set passive mode)

Parameter errors occurs when any of `$host, $user, $pass, $source` or `$destination` is unspecified, or when the parameter `$suffix` has an invalid value.

4.1.3 log.lib

This function allows a PHP3-script to append an entry to the system log.

```
function log($message)
{
...
}
```

Parameter	Description
`$message`	Message to be appended to the system log.

The function makes a call to the PHP3-function `error_log($message,0)`.

4.1.4 mail.lib

This function issues a system call to `smtpclient`.

```
function mail($subject,$from,$reply,$to,$copy,$file,$attach)
{
...
}
```

Parameter	Description
`$subject`	Message subject (default is `no subject`).
`$from`	(not null) Message author.
`$reply`	Reply should be sent to this user (default is `$from`).
`$to`	(not null) Recipients.
`$copy`	Copy the message to other recipients.
`$file`	Message read from file.
`$attach`	File to attach.

The script requires a valid mail-server to be entered in the camera/video server settings.

Although the argument `$file` can be left unspecified, this is NOT recommended, as this sets `smtpclient` to interactive mode and thus locks the execution of PHP.

On error, the function returns 1 (no sender specified) or 2 (no recipient specified) Otherwise the function returns 0.

4.1.5 ppp.lib

These functions allow a PHP3-script to control the use of ppp connections through the ppp-wrapper.

```
function ppp_getpid()
{
...
}
```

Called from `ppp_start()` and `ppp_stop()`. Returns current `pppwrapper.pid` as reported by the file `/var/log/pppwrapper.pid`.

```
function ppp_getstat( $num_of_checks )
{
...
}
```

Called from `ppp_online()`. Returns current ppp status by reading the `pppstat.log`.
Returns 1 if it reads `ON`, otherwise returns 0.
`$num_of_checks` is the number of attempts to read the file `/tmp/pppstat.log`.

```
function ppp_start()
{
...
}
```

Called from `ppp_online()`.
Sends `SIGUSR1 (10)` to the currently running ppp-wrapper.
If the wrapper is not running, the function returns -1 and an error is logged in the `syslog`.

```
function ppp_stop()
{
...
}
```

Called from `ppp_offline()`. Sends `SIGUSR2 (12)` to the currently running ppp-wrapper.
If the wrapper isn't running, the function returns –1 and an error is logged in the `syslog`.

```
function ppp_online($behavior, $max_attempt, $ppp_interval)
{
...
}
```

Initializes the ppp connection.
Returns 1 if the connection is established, -1 for errors and logs the error in the `syslog`.
`$behavior` describes the ppp connection's behavior when closing the connection, and can be either `CloseAfter` or `Optimized`.
`$max_attempt` is the number of attempts to connect before failing.
`$ppp_interval` is the delay in seconds between connection attempts.

```
function ppp_offline($behavior, $sec)
{
...
}
```

Closes the ppp connection
`$behavior` describes the ppp connection's behavior and can be either `CloseAfter` or `Optimized`.
`$sec` is the time in seconds to wait before calling `ppp_stop()` if `$behaviour="CloseAfter".` If `$behaviour="Optimized"` there will be no wait, `ppp_stop` is called immediately.

4.1.6 Examples

The first example shows how to use the FTP-function together with the `bufferd` example:

```
<?php

include "/usr/php/ftp.lib";
include "/usr/php/log.lib";
$host="host.domain";
$user="user";
$pass="password";
$time="0";
$delay="0";
$source="/tmp/SNAP/snapshot.jpg";
$destination="snapshot.jpg";
$suffix="date";
$countermax="";
$startcount="";
$port="21";
```

```
$passive_mode="on";
shoot();

$res = ftp($host, $user, $pass, $time, $delay, $source, $destination,
$suffix, $countermax, $startcount, $port, $passive_mode);

if($res == 1)
{
  log("upload script - Could not connect to host");
}
else if($res == 2)
{
  log("upload script - Could not log on  to host");
}
else if($res == 3)
{
  log("upload script - Could not put the file");
}
else if($res == 4)
{
  log("upload script - something wrong with parameters");
}
else if($res == 5)
{
  log("upload script - could not turn passive mode on/off");
}
?>
```

The second example depicts how to use the mail function, and is the script mentioned as `mail_syslog.php` in the `utask` example:

```
<?php

include "/usr/php/mail.lib";
include "/usr/php/log.lib";

$subject="Todays messages";
$from="root@camera.domain";
$reply="";
$to="cam-admin@host.domain";
$copy="";
$body="/var/log/messages";
$attach="";
$res = mail($subject, $from, $reply, $to, $copy, $body, $attach);
if($res == 1)
{
  log("mail script - Could not send mail:no sender\n");
}
else if($res == 2)
{
  log("mail script - Could not send mail:no recipient\n");
}
?>
```

4.2 PHP3 script examples

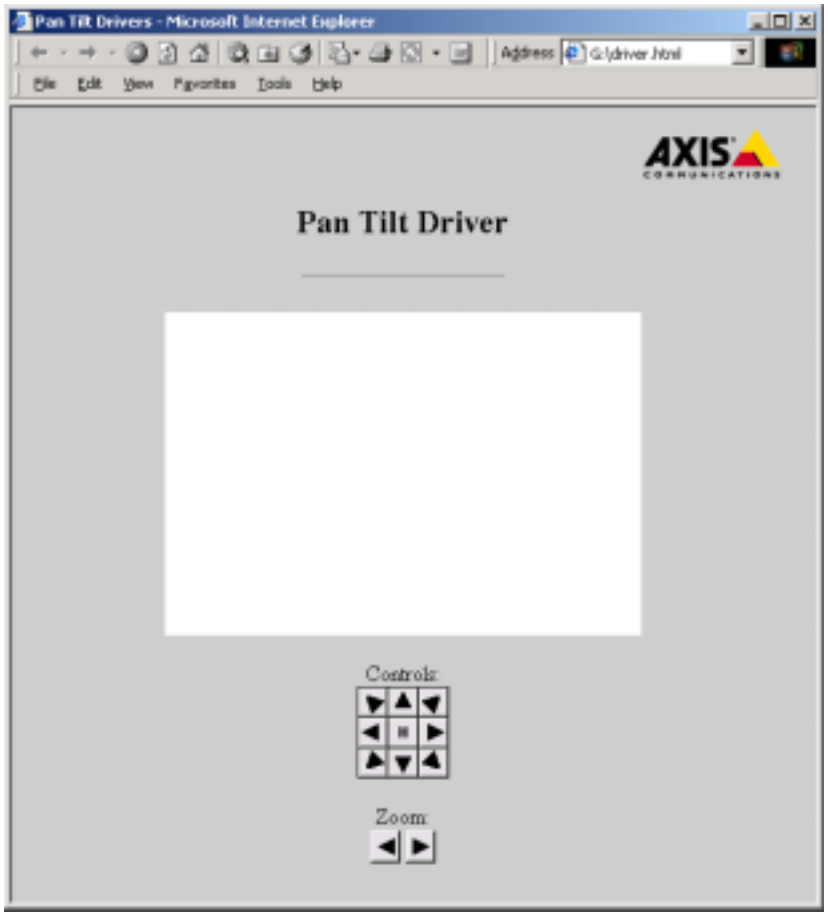
The complete PHP3 scripts shown here can be used for the most common applications. The scripts can also be downloaded from the Axis web site at **www.axis.com**.

PHP Scripting Tips

- To use a browser for debugging your scripts, write the log messages as html.
- To log a message to syslog (enabled by default), include in the script:
  `log_error(mymessage,0);`
- To disable logging, include the row: `error_reporting (E,0);`
- To enable it again, include: `error_reporting (E,ALL);`

4.2.1 Example 1 – PTZ Control

The purpose of this example is to make a PHP3 driver for a pan-tilt unit. An html interface is used to control the camera. The example is written for the Axis 2400/2401Video Server and a Pelco pan-tilt unit, and will, with minor changes, work on most pan-tilt units.



All the files in this example are available on the Axis website. We will only show the script parts that are important to understanding the example.

4.2.1.1 Driver.php3

The script moves the camera according to the button pressed by the user. The HTML interface sends a single parameter to the script: the number of the button pressed. The appropriate data is sent to the serial port, according to the number/button pressed.

The script uses the HTTP-API (available on **www.axis.com**) to write data to the serial port.

```
<?
switch($param) // Move the camera according to the button pressed
{
      case 1: // Up Left
            fopen("http://127.0.0.1/axis-
            cgi/com/serial.cgi?port=1\&write=FF020008003F49","w");
            fopen("http://127.0.0.1/axis-
            cgi/com/serial.cgi?port=1\&write=FF0200043F0045","w");
            usleep(100000); // Wait 0.1 seconds
            fopen("http://127.0.0.1/axis-
            cgi/com/serial.cgi?port=1\&write=FF020000000002","w");
            break;

      case 2: // Up
            fopen("http://127.0.0.1/axis-
            cgi/com/serial.cgi?port=1\&write=FF020008003F49","w");
            usleep(100000); // Wait 0.1 seconds
            fopen("http://127.0.0.1/axis-
            cgi/com/serial.cgi?port=1\&write=FF020000000002","w");
            break;

      case 3: // Up Right
            fopen("http://127.0.0.1/axis-
            cgi/com/serial.cgi?port=1\&write=FF020008003F49","w");
            fopen("http://127.0.0.1/axis-
            cgi/com/serial.cgi?port=1\&write=FF0200023F0043","w");
            usleep(100000); // Wait 0.1 seconds
            fopen("http://127.0.0.1/axis-
            cgi/com/serial.cgi?port=1\&write=FF020000000002","w");
            break;

      case 4: // Right
            fopen("http://127.0.0.1/axis-
            cgi/com/serial.cgi?port=1\&write=FF0200023F0043","w");
            usleep(100000); // Wait 0.1 seconds
            fopen("http://127.0.0.1/axis-
            cgi/com/serial.cgi?port=1\&write=FF020000000002","w");
            break;

      case 5: // Down Right
            fopen("http://127.0.0.1/axis-
            cgi/com/serial.cgi?port=1\&write=FF020010003F51","w");
            fopen("http://127.0.0.1/axis-
            cgi/com/serial.cgi?port=1\&write=FF0200023F0043","w");
            usleep(100000); // Wait 0.1 seconds
            fopen("http://127.0.0.1/axis-
            cgi/com/serial.cgi?port=1\&write=FF020000000002","w");
```

```
            break;

      case 6: // Down
            fopen("http://127.0.0.1/axis-
            cgi/com/serial.cgi?port=1\&write=FF020010003F51","w");
            usleep(100000); // Wait 0.1 seconds
            fopen("http://127.0.0.1/axis-
            cgi/com/serial.cgi?port=1\&write=FF020000000002","w");
            break;

      case 7: // Down Left
            fopen("http://127.0.0.1/axis-
            cgi/com/serial.cgi?port=1\&write=FF020010003F51","w");
            fopen("http://127.0.0.1/axis-
            cgi/com/serial.cgi?port=1\&write=FF0200043F0045","w");
            usleep(100000); // Wait 0.1 seconds
            fopen("http://127.0.0.1/axis-
            cgi/com/serial.cgi?port=1\&write=FF020000000002","w");
            break;

      case 8: // Left
            fopen("http://127.0.0.1/axis-
            cgi/com/serial.cgi?port=1\&write=FF0200043F0045","w");
            usleep(100000); // Wait 0.1 seconds
            fopen("http://127.0.0.1/axis-
            cgi/com/serial.cgi?port=1\&write=FF020000000002","w");
            break;

      case 9: // Zoom Out
            fopen("http://127.0.0.1/axis-
            cgi/com/serial.cgi?port=1\&write=FF020040000042","w");
            usleep(1000000); // Wait 0.1 seconds
            fopen("http://127.0.0.1/axis-
            cgi/com/serial.cgi?port=1\&write=FF020000000002","w");
            break;

      case 10: // Zoom In
            fopen("http://127.0.0.1/axis-
            cgi/com/serial.cgi?port=1\&write=FF020020000022","w");
            usleep(1000000); // Wait 0.1 seconds
            fopen("http://127.0.0.1/axis-
            cgi/com/serial.cgi?port=1\&write=FF020000000002","w");
            break;
}
printf(" "); //Return data to avoid the popup windows saying
"Document contains no data"
?>
```

4.2.2 Example 2 – FTP Upload of Images

This script will upload 2 pre-alarm and 2 post-alarm images via FTP. It is fairly general and may be used in many different configurations.

4.2.2.1 The task.list

```
once immune % /bin/bufferd : -start -buffername CAM1 -pre 2 -post 2 -predelay
1000 -postdelay 1000 -uri ftp://jpg/1/352x288.jpg;

date(w(0,1,2,3,4,5,6)) pattern((IO0:/)) immune once % /bin/sh : -c php
/etc/httpd/html/alarm_ftp_net.php3;
```

The first entry in the `task.list` will start a buffer (`CAM1`) that continuously fetches 2 pre-alarm images from camera 1 and is prepared to fetch 2 post-alarm images. The images are fetched at rate of 1 frame per second (1000ms).

The second entry specifies that when input one goes high (rising flank), then the script **`alarm_ftp_net.php3`** will be executed.

4.2.2.2 The Script

The first section of the script contains the available parameters, which may be changed according to requirements.

```
< ?

$buffer_prefix = "CAM";                 // The prefix of the name
                                        //of the buffer(s) started
$sources = "1";                         // The index of the name(s)
                                        //of the buffer(s) started
$image_format = "fullsize";             // The format of the images
                                        //specified to be taken
                                        //according to the HTTP-API
$pre = 2;                               // Number of pre alarm
                                        //images to be taken
$post = 2;                              // Number of post alarm
                                        //images to be taken
$predelay = 1000;                       // Delay between pre images
                                        //in milliseconds
$postdelay = 1000;                      // Delay between post
                                        //images in milliseconds
$ftp_server = "10.13.9.130";            // The server to upload to
$port = "21";                           // The port to connect to
$user = "user";                         // The user to login as
$pass = "pass";                         // The pass to use for the
                                        //user
$passive_mode = "no";                   // Choose passive mode on
                                        //("yes") or off ("no").
                                        //(See documentation on FTP
                                        //protocol)
$destination = "upload/2400test";       // The path to append to
                                        //all uploads. This path
                                        //must exist on the
                                        //server prior to upload.
```

The second part of the script is where it all happens:

1. All pre/post alarm buffers previously started by utask are stopped.
2. Waits until all post-alarm images are stored.
3. FTP transfer of all images.
4. Restart of pre/post alarm buffers.
5. Exit.

```
for($c=0;$c<(strlen($sources));$c++)
{                                       // Stop the, by utask,
                                        //started buffers
      $command="bufferd -stop -buffername
      ".$buffer_prefix.substr($sources,$c,1);
      system($command);
}

error_reporting(0);

for($c=0;$c<(strlen($sources));$c++)
{                                       // For each buffer specified
      $status_file =
      "/tmp/".$buffer_prefix.substr($sources,$c,1)."/status";

      while(!is_file($status_file))
      {                                 // Wait until bufferd is
                                        //ready with the images, i.e.
                                        //the status file is present
            sleep(($predelay+$postdelay)/1000);
      }
      unlink($status_file);
}
error_reporting(E_ALL);
$session = ftp_connect($ftp_server, $port);

if($session)
{                                       // Connection successfully
                                        //established

      if(ftp_login($session, $user, $pass))
      {                                 // Successful login attempt

            if (!ftp_pasv($session, $passive_mode == "yes"))
            {
                  error_log("Could not set passive mode",0);
            }
            else
{                                       // Passive mode successfully
                                        //set

                  for($i=0;$i < (strlen($sources));$i++)
                  {                     // For each buffer specified


                     $directory="/tmp/".$buffer_prefix.substr($sources
                     ,$i,1);
                     $buffer_handle = opendir($directory);
While($file_name = readdir($buffer_handle))
                     {                  // And for each file in the
                                        //corresponding buffer
```

```
                                        //directory

                        if(($file_name != ".") &&($file_name != ".."))
                        {               // That is a regular file
                              $file =
                              "/tmp/".$buffer_prefix.substr($sources,$i
                              ,1)."/".$file_name;
                              $destination_name = $destination .
                              $buffer_prefix.substr($sources,$i,1);
                              $destination_name .=
                              strchr($file_name,"_");
                                        //extract timestamp from
                                        //src, append to dst
                              if(!ftp_put($session, $destination_name,
                              $file, FTP_BINARY))
                              {         // Upload that file to the
                                        //FTP server
                                    error_log("Could not upload file
                                    ".$file." as
                                    ".$destination_name."\n",0);
                              }
                              unlink($file);
                        }
                  }
                  closedir($buffer_handle);
            }
      }
}
else
{
      error_log("Could not login as ".$user." on ".$ftp_server,0);
}
      ftp_quit($session);
}
else
{
      error_log("Could not connect to ".$ftp_server,0);
}

error_log("Restarting buffer(s)",0);

for($c=0;$c<(strlen($sources));$c++)
{                                       // Reset and restart buffers
      $command="bufferd -reset -buffername
      ".$buffer_prefix.substr($sources,$c,1);
      system($command);
      $command="bufferd -start -buffername
      ".$buffer_prefix.substr($sources,$c,1)." -pre ".strval($pre)." -
      post ".strval($post)." -predelay ".strval($predelay)." -postdelay
      ".strval($postdelay)." -uri
      ftp://jpg/".substr($sources,$c,1)."/".$image_format.".jpg";
      system($command);
}
?>
```

4.2.3 Example 3 – FTP and E-mail on Event

This script will upload 2 pre-alarm and 2 post-alarm images via FTP and also send an e-mail as notification of the event. In addition to the parameters in example 1, the following parameters are added:

```
<?
$smtp_server = "mail.somewhere.com";// The server to use as mail server
$subject = "'Alarm trigged'";       // The subject to use in the mail
$from = "someone@somewhere.com";    // The specified sender
$reply = " someone1@somewhere.com ";// The specified recipient
$cc = " someone2@somewhere.com ";   // The specified recipient of a
                                    //copy of this mail
$body = "/tmp/var/log/messages";    // The body to insert into the
                                    // mail. Note that this must be
                                    //specified and point to a valid
                                    //file
$to = " someone@somewhere.com ";    // The specified recipient
```

The script itself is identical to example 1, except for the extra lines shown below.

```
for($c=0;$c<(strlen($sources));$c++)
{
   $command="bufferd -stop -buffername
   ".$buffer_prefix.substr($sources,$c,1);
   system($command);
}
//This is where the code specific to Example 2 starts.

$command = "smtpclient";
$command .=" -L -S ".$smtp_server;
$command .=" -s ".$subject;
$command .=" -f ".$from;
$command .=" -r ".$reply;
$command .=" -c ".$cc;
$command .=" -b ".$body;
$command .=" ".$to;
exec($command);

//This is where the code specific to Example 2 ends.

error_reporting(0);

……the rest of the script as for example 1.
```

4.2.4 Example 4 – Sequential FTP Upload

When input 0 goes high, this script will, for 10 seconds, upload images via FTP, at 2 second intervals. The uploaded images can be named by specifying a suffix. These can be date, incremental sequence number or limited sequence (overwrites files when maximum is reached).

The parameters come first:

```
<?

$buffer_prefix = "SNAP";                    // The prefix of the name
                                            //of the buffer(s) started
$sources = "1";                             // The index of the name(s) of the
                                            // buffer(s) started
$image_format = "fullsize";                 // The format of the images
                                            //specified to be taken
                                            //according to the HTTP-API
$delay = 2000;                              // Delay between images //taken
$file_format = "snapshot";                  // The name to be given to
                                            //the local file, excluding
                                            //the extension
$ftp_server = "10.13.9.130";                // The server to upload to
$port = "21";                               // The port to connect to
$user = "user";                             // The user to login as
$pass = "pass";                             // The pass to use for the
                                            //user
$passive_mode = "no";                       // Choose passive mode on
                                            //("yes") or off ("no").
                                            //(See documentation on FTP
                                            //protocol)
$destination = "upload/2400";               // The path to append to all
                                            //uploads. This path must exist on
                                            // the server prior to upload.
$time = 10;                                 // The time, in seconds, to stay
                                            // in this script. A value of -1
                                            // means indefinitely.
$suffix= "date";                            // The type of suffix to use on
                                            //the uploaded files. Use "date"
                                            //for the date,
                                            //$suffix = "sequence"; or
                                            //"sequence" for an index
                                            //$counter_max = 10; Limited by an
                                            //integer. $suffix =
                                            //"sequence_max";  Or
                                            //"sequence_max" for an index up
                                            //to the internal maximum integer.
```

The script itself follows:

```
error_reporting(E_ALL);
function conv($value)
{                                          // A function for converting
                                           //single digit integers into
                                           //two digit integers
      if($value < 10) {
            $value = "0$value";
      }
      return strval($value);
}

for($c=0;$c<(strlen($sources));$c++)
{                                          // Start the buffers as
                                           //specified by the parameters
                                           //above
      //$command="bufferd -reset -buffername
      //".$buffer_prefix.substr($sources,$c,1);
      //system($command);
      $command="bufferd -start -buffername ";
      $command.=$buffer_prefix.substr($sources,$c,1);
      $command.=" -snapshot -pre 1 -predelay ";
      $command.=strval($delay);
      $command.=" -uri ftp://jpg/";
      $command.=substr($sources,$c,1);
      $command.="/";
      $command.=$image_format;
      $command.=".jpg -format ";
      $command.=$file_format;
      system($command);
}

error_reporting(0);
$current_counter = 1;
if($fd = fopen("/tmp/counter","r"))
{                                          // If the file looked for
                                           //exists, retrieve the index to
                                           //start at
      while($buf = fread($fd,8))
      {
            $current_counter .= $buf;
      }
      $current_counter = intval($current_counter);
      fclose($fd);
}

if(($suffix=="sequence") && $current_counter > $counter_max)
{                                          // If the loaded index exceeds
                                           //the specified maximum, reset
                                           //index
      $current_counter = 1;
      if($fd = fopen("/tmp/counter","w"))
      {
            $buf = fwrite($fd,strval($current_counter));
            fclose($fd);
      }
}
error_reporting(E_ALL);
```

```
$session = ftp_connect($ftp_server, $port);
if($session)
{                                       // Connection successfully
                                        //established
      if(ftp_login($session, $user, $pass))
      {                                 // Successful login attempt
            if (!ftp_pasv($session, $passive_mode == "yes"))
            {
ftp_quit($session);
                  error_log("Could not set passive mode",0);
            }
            else
            {                           // Passive mode successfully
                                        //set
                  $start_time = gettimeofday();
                  $current_time = $start_time;
                  $session_time = 0;
                  $inc = 0;
                  $active_buffer = -1;
                  $failures=0;
                  while( ($failures<(strlen($sources)*2)) && (
                  ($session_time<$time)||($time==-1) ))
                  {                     //Upload images as long as
                                        //not too many errors have
                                        //occurred and session time
                                        //has not been exceeded
                        $loop_start = gettimeofday();
                        $active_buffer++;
                        if($active_buffer==strlen($sources))
                        {               // Loop through the indexes
                                        //specified as sources
                              $active_buffer=0;
                        }
                        $source_file="/tmp/".$buffer_prefix.substr
                        ($sources,$active_buffer,1)."/".$file_format.".jpg";

                                        //Build the destination name
                                        //according to the suffix
                                        //specified

                        $dest=$destination.$buffer_prefix.substr
                        ($sources,$active_buffer,1);
if($suffix == "date")
                        {
                              $tinfo = getdate(time());
                              $dest .= "_" . conv($tinfo["year"]);
                              $dest .= "-" . conv($tinfo["mon"]);
                              $dest .= "-" . conv($tinfo["mday"]);
                              $dest .= "_";
                              $dest .= conv($tinfo["hours"]);
                              $dest .= conv($tinfo["minutes"]);
                              $dest .= conv($tinfo["seconds"]);
                        }
                        else if($suffix == "sequence_max")
                        {
                              $dest .= "_" . strval($current_counter);
                              $inc++;
                              if($inc >= strlen($sources))
                              {
                                    $current_counter++;
```

```
                                    error_reporting(0);
                                    if($fd = fopen("/tmp/counter","w"))
                                    {
                                          $buf =
                                          fwrite($fd,strval($current_counter
                                          ));
                                          fclose($fd);
                                    }
                                    error_reporting(E_ALL);
                                    $inc = 0;
                              }
                        }
                        else if($suffix == "sequence")
                        {
                              if($current_counter > $counter_max)
                              {
                                    $counter = 1;
                              }
                              $dest .= "_" . strval($current_counter);
                              $inc++;
                              if($inc >= $buffer_count)
                              {
                                    $current_counter++;
                                    error_reporting(0);
                                    if($fd = fopen("/tmp/counter","w"))
                                    {
                                          $buf =
                                          fwrite($fd,strval($current_counter
                                          ));
                                          fclose($fd);
                                    }
                                    error_reporting(E_ALL);
                                    $inc = 0;
                              }
                        }

                        $dest .= ".jpg";
                        error_reporting(0);
                        if(is_file($source_file))
                        {                   //If the source file is present
                              if(!ftp_put($session, $dest, $source_file,
                              FTP_BINARY))
                              {             //Upload the source file
                                    $failures++;
                                    error_log("Could not upload file
                                    ".$source_file." as ".$dest." on
                                    ".$ftp_server,0);
                              }
                              else
                              {             //If successful upload, remove the
                                            //uploaded file indicating capture
                                            //of a new image
                                    $failures=0;
                                    unlink($source_file);
                              }
                        }
                        else
                        {
                              error_reporting(E_ALL);
                              $failures++;
```

```
                              error_log("No such file: ".$source_file,0);
                        }
                        $current_time = gettimeofday();
                        $ellapsed = ($current_time["sec"] -
                        $loop_start["sec"]) * 1000000;

                        if($current_time["usec"] > $loop_start["usec"])
                        $ellapsed += $current_time["usec"] -
                        $loop_start["usec"];

                        else
                              $ellapsed -= $current_time["usec"] -
                        $loop_start["usec"];
                        $wait = ($delay * 1000) / strlen($sources);

                        if($ellapsed < $wait)
                        {                  // Wait if needed (in order to
                                           //follow the delay specified
                                           //and spread the traffic)
                              usleep($wait - $ellapsed);
                        }

                                           //Calculate session time
                        $session_time = ($current_time["sec"] -
                        $start_time["sec"]);

                        if($current_time["usec"] > $start_time["usec"])
                              $session_time += ($current_time["usec"] -
                        $start_time["usec"])/1000000;


else
                              $session_time -= ($current_time["usec"] -
                        $start_time["usec"])/1000000;

                  }
                  if($session_time>=$time) error_log("Timed upload
                  complete",0);
                  if($failures>=(strlen($sources)*2)) error_log("Too many
                  consecutive failures",0);
            }
      }
      else
      {
            ftp_quit($session);
            error_log("Could not log in as ".$user." on ".$ftp_server,0);
      }
      ftp_quit($session);
}
else
{
      error_log("Could not connect to ".$ftp_server.":".$port,0);
}

for($c=0;$c<(strlen($sources));$c++)
{                                  //Reset buffers
      //$command="bufferd -stop -buffername
      //".$buffer_prefix.substr($sources,$c,1);
      //system($command);
      $command="bufferd -reset -buffername
      ".$buffer_prefix.substr($sources,$c,1);
```

```
      system($command);
}

?>
```

4.2.5 Example 5 – Send Images via E-mail

This script will send 4 (2 pre-alarm and 2 post-alarm) images as attachments in an e-mail.

The parameters available are:

```
<?

$buffer_prefix = "CAM";                  // The prefix of the name of the
                                         //buffer(s) started
$sources = "1";                          // The index of the name(s) of the
                                         //buffer(s) started
$image_format = "fullsize";              // The format of the images
                                         //specified to be taken
                                         //according to the HTTP-API
$pre = 2;                                // Number of pre alarm images
                                         //to be taken
$post = 2;                               // Number of post alarm
                                         //images to be taken
$predelay = 1000;                        // Delay between pre images
                                         //in milliseconds
$postdelay = 1000;                       // Delay between post
                                         //images in milliseconds
$smtp_server = "mail.somewhere.com";     // The server to use as
                                         //mail server
$subject = "test";                       // The subject to use in
                                         //the mail
$from = " someone@somewhere.com";        // The specified sender
$reply = " someone@somewhere.com ";      // The specified recipient
                                         //for replies
$cc = " someone@somewhere.com ";         // The specified recipient for a
                                         //copy of this mail
$body = "/tmp/var/log/messages";         // The body to insert into the
                                         //mail. Note that this must be
                                         //specified and point to a valid
                                         //file
$to = " someone@somewhere.com ";         // The specified recipient
```

The script itself:

```
error_reporting(E_ALL);
error_log("Stopping buffer(s)",0);

for($c=0;$c<(strlen($sources));$c++)
      {                                    // Stop the, by utask,
                                           //started buffers
      $command="bufferd -stop -buffername
      ".$buffer_prefix.substr($sources,$c,1);
      system($command);
}

for($c=0;$c<(strlen($sources));$c++)
{                                          // For each buffer specified
      $status_file = "/tmp/".$buffer_prefix.substr($sources,$c,1)."/status";
      error_reporting(0);

      while(!is_file($status_file))
      {                                    // Wait until bufferd is ready
                                           //with the images, i.e. the status
                                           //file is present
            sleep(($predelay+$postdelay)/1000);
      }
      unlink($status_file);
}
error_reporting(E_ALL);

for($c=0;$c<(strlen($sources));$c++)
{                                          // For each buffer, mail the
                                           //directory containing the images
                                           //taken
      $command = "smtpclient";
      $command .=" -S ".$smtp_server;
      $command .=" -b ".$body;
      $command .=" -M 2 -d /tmp/".$buffer_prefix.substr($sources,$c,1);
      $command .=" -s ".$subject;
      $command .=" -f ".$from;
      $command .=" -r ".$reply;
      $command .=" -c ".$cc;
      $command .=" ".$to;
      system($command);
}

error_log("Restarting buffer(s)",0);

for($c=0;$c<(strlen($sources));$c++)
{                                          //Reset and restart buffers
      $command="bufferd -reset -buffername
      ".$buffer_prefix.substr($sources,$c,1);
      system($command);
      $command="bufferd -start -buffername
      ".$buffer_prefix.substr($sources,$c,1)." -pre ".strval($pre)." -post
      ".strval($post)." -predelay ".strval($predelay)." -postdelay
      ".strval($postdelay)." -uri
      ftp://jpg/".substr($sources,$c,1)."/".$image_format.".jpg";
      system($command);
}

?>
```

5 An Introduction to Shells in General

A *shell* is a programming language that is fully equipped with:

- variables
- conditional and iterative constructs

5.1 The `mish` shell

The commands available in `mish` are essentially the same as the ones commonly used in other shells on any Linux/Unix system, so the manual pages can often be useful. The difference is that the commands in `mish` have been simplified, i.e. options have been removed. Besides these modifications, some extra commands have been added. These are not documented as a manual page.

The shell includes a help command, which lists the functions available in the shell.

When programming shell scripts, you must begin with the sequence `#!/bin/mish`, before starting on your code. The statement after `#!` states the name of the program used to interpret the code in your script.

Another good practice when programming is to add comments to your code. All comments are preceded with the hash (#) sign.

When a shell script is created, the file will not be executable. By default, the file will be a read/write file, and the user will not be able to run or execute it. To make a file executable, the file permission must be changed. Use the command `chmod` to make a shell file executable.

5.2 Shell commands

Name

sh, ., break, case, cd, continue, eval, exec, exit, export, for,if, read, readonly,set, shift, trap, umask, wait, while – shell

Synopsis

`sh [-eiknqstvx`u**`][-c` *`str`*`] [`*`file`*`]`**

Options

- **-c** Execute the commands in *`str`*
- **-e** Quit on error
- **-i** Interactive mode; ignore `QUIT,TERMINATE, INTERRUPT`
- **-k** Look for `name=value` everywhere on command line
- **-n** Do not execute commands
- **-q** Change `qflag` from `sig_ign` to `sig_del`
- **-s** Read commands from standard input
- **-t** Exit after reading and executing one command

- **-v** Echo input lines as they are read
- **-x** Trace
- **-u** Unset variables

Example

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

Description

Sh is the shell, which forms the user's main interface with the system. On startup, the shell reads `/etc/pro`-file `and $HOME/.profile`, if they exist, and executes any commands they contain. The Minix shell has most of the features of the V7 (Bourne) shell, including redirection of input and output, pipes, magic characters, background processes, and shell scripts. A brief summary follows, but whole books have been written on shell programming alone.

Some of the more common notations are:

```
date                    # Regular command
sort <file              # Redirect stdin (standard input)
sort <file1 >file2      # Redirect stdin and stdout
cc file.c 2>error       # Redirect stderr
a.out >f 2>&1           # Combine standard output and standard error
sort <file1 >>file2     # Append output to file2
sort <file1 >file2 &    # Background job
(ls -l; a.out) &        # Run two background commands sequentially
sort <file | wc         # Two-process pipeline
sort <f | uniq | wc     # Three-process pipeline
ls -l *.c               # List all files ending in .c
ls -l [a-c]*            # List all files beginning with a, b,or c
ls -l ?                 # List all one-character file names
ls \?                   # List the file whose name is question mark
ls '???'                # List the file whose name is three question marks
v=/usr/ast              # Set shell variable v
ls -l $v                # Use shell variable v
PS1='Hi! '              # Change the primary prompt to Hi!
PS2='More: '            # Change the secondary prompt to More:
ls -l $HOME             # List the home directory
echo $PATH              # Echo the search path
echo $?                 # Echo exit status of previous command in decimal
echo $$                 # Echo shell's pid in decimal
echo $!                 # Echo PID of last background process
echo $#                 # Echo number of parameters (shell script)
echo $2                 # Echo second parameter (shell script)
echo "$2"               # Echo second parameter without expanding spaces
echo $*                 # Echo all parameters (shell script)
echo $@                 # Echo all parameters (shell script)
echo "$@"               # Echo all parameters without expanding spaces
```

The shell uses the following variables for specific purposes:

SHELL	the path of the current shell
HOME	the default value for the cd(1) command
PATH	the directories to be searched to find commands
IFS	the internal field separators for command strings
PS1	the primary shell prompt
PS2	the secondary shell prompt

There are various forms of substitution on the shell command line:

‘...‘	Command string between back-quotes is replaced by its output
"..."	Permits variable substitution between quotes
’...’	Inhibits variable substitution between quotes
$VAR	Replaced by contents of variable VAR
${VAR}	Delimits variable VAR from any following string

The expressions below depend on whether or not VAR has ever been set. If VAR has been set, they give:

${VAR-str}	Replace expression by VAR, else by str
${VAR=str}	Replace expression by VAR, else by str and set VAR to str
${VAR?str}	Replace expression by VAR, else print str and exit shell
${VAR+str}	Replace expression by str, else by null string

If a colon is placed after VAR, the expressions depend on whether or not VAR is currently set and non-null.

The shell has a number of built-in commands:

:	return true status
.fn	execute shell script fn on current path
break [n]	break from a for,until or while loop; exit n levels
continue [n]	continue a for,until or while loop; resume n:th loop
cd [dir]	change current working directory; move to $HOME
eval cmd	rescan cmd, performing substitutions
eval	rescan the current command line
exec cmd	execute cmd without creating a new process
exec <|>	with no command name, modify shell I/O
exit [n]	exit a shell program, with exit value n
export [var]	export var to shell’s children; list exported variables
pwd	print the name of the current working directory
read var	read a line from stdin and assign to var
readonly [var]	make var read-only; list read-only variables
set -f	set shell flag (+f unsets flag)
set str	set positional parameter to str
set	show the current shell variables
shift	reassign positional parameters (except ${0}) one left
times	print accumulated user and system times for processes

`trap arg sigs`	trap signals `sigs` and run `arg` on receipt
`trap`	list trapped signals
`umask [n]`	set the user file creation mask; show the current `umask`
wait [n]	wait for process pid `n`; wait for all processes

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.
`(...)`	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

See also:

echo(1), **expr**(1), **pwd**(1), **true**(1).

5.3 Additional Commands Available with Busybox

BusyBox combines tiny versions of many common UNIX 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 any small or embedded system. 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.

5.3.1 basename

Syntax: `basename FILE [SUFFIX]`

Strips directory path and suffixes from `FILE`. If specified, also removes any trailing `SUFFIX`.

Example:

```
$ basename /usr/local/bin/foo
foo
$ basename /usr/local/bin/
bin
$ basename /foo/bar.txt .txt
bar
```

5.3.2 cat

Syntax: `cat [FILE]...`

Concatenates FILE(s) and prints them to stdout.

Example:

```
$ cat /proc/uptime
110716.72 17.67
```

5.3.3 chroot

Syntax: `chroot NEWROOT [COMMAND...]`

Run COMMAND with root directory set to NEWROOT.

Example:

```
$ ls -l /bin/ls
lrwxrwxrwx    1 root     root          12 Apr 13 00:46
/bin/ls -> /BusyBox
$ mount /dev/hdc1 /mnt -t minix
$ chroot /mnt
$ ls -l /bin/ls
-rwxr-xr-x    1 root     root        40816 Feb  5 07:45
/bin/ls*
```

5.3.4 cp

Syntax: `cp [OPTION]... SOURCE DEST`

Copies `SOURCE` to `DEST`, or multiple `SOURCE(s)` to `DIRECTORY`.

```
-a      Same as -dpR
-d      Preserves links
-p      Preserves file attributes if possible
-f      force (implied; ignored) - always set
-R      Copies directories recursively
```

5.3.5 cut

Syntax: `cut [OPTION]... [FILE]...`

Prints selected fields from each input FILE to standard output.

```
Options:
      -b LIST     Output only bytes from LIST
      -c LIST     Output only characters from LIST
      -d CHAR     Use CHAR instead of tab as the field delimiter
      -s          Output only the lines containing delimiter
      -f N        Print only these fields
      -n          Ignored

Example:
            $ echo "Hello world" | cut -f 1 -d ' '
            Hello
            $ echo "Hello world" | cut -f 2 -d ' '
            world
```

5.3.6 date

Syntax: `date [OPTION]... [+FORMAT]`

Displays the current time in the given FORMAT, or sets the system date.

```
Options:
      -R              Outputs RFC-822 compliant date string
      -d STRING       display time described by STRING, not `now'
      -s              Sets time described by STRING
      -u              Prints or sets Coordinated Universal Time

Example:
            $ date
            Wed Apr 12 18:52:41 MDT 2000
```

5.3.7 dd

Syntax: `dd [if=FILE] [of=FILE] [bs=N] [count=N] [skip=N][seek=N][conv=notrunc|sync]`

Copy a file, converting and formatting according to options

```
            if=FILE         read from FILE instead of stdin
            of=FILE         write to FILE instead of stdout
            bs=N            read and write N bytes at a time
            count=N         copy only N input blocks
            skip=N          skip N input blocks
            seek=N          skip N output blocks
            conv=notrunc    don't truncate output file
            conv=sync       pad blocks with zeros
```

Numbers may be suffixed by c (x1), w (x2), b (x512), kD (x1000), k (x1024), MD (x1000000), M (x1048576), GD (x1000000000) or G (x1073741824).

```
Example:
            $ dd if=/dev/zero of=/dev/ram1 bs=1M count=4
            4+0 records in
```

```
4+0 records out
```

5.3.8 df

Syntax: `df [-hmk] [filesystem ...]`

Print the filesystem space used and space available.

```
Options:

        -h   print sizes in human readable format (e.g., 1K 243M 2G )
        -m   print sizes in megabytes
        -k   print sizes in kilobytes(default)
```

```
Example:
$ df
Filesystem        1k-blocks   Used        Available   Use%  Mounted on
/dev/sda3         8690864     8553540     137324      98%   /
/dev/sda1         64216       36364       27852       57%   /boot
$ df /dev/sda3
Filesystem        1k-blocks   Used        Available   Use%  Mounted on
/dev/sda3         8690864     8553540     137324      98%   /
```

5.3.9 dirname

Syntax: `dirname [FILENAME ...]`

Strips non-directory suffix from `FILENAME`

```
Example:
          $ dirname /tmp/foo
          /tmp
          $ dirname /tmp/foo/
          /tmp
```

5.3.10 du

Syntax: `du [-lshmk] [FILE]...`

Summarizes disk space used for each `FILE` and/or directory. Disk space is printed in units of 1024 bytes.

```
Options:
        -l   count sizes many times if hard linked
        -s   display only a total for each argument
        -h   print sizes in human readable format (e.g., 1K 243M 2G )
        -m   print sizes in megabytes
        -k   print sizes in kilobytes(default)

Example:
        $ du
        16      ./CVS
        12      ./kernel-patches/CVS
        80      ./kernel-patches
        12      ./tests/CVS
        36      ./tests
        12      ./scripts/CVS
```

```
16      ./scripts
12      ./docs/CVS
104     ./docs
2417    .
```

5.3.11 echo

Syntax: `echo [-neE] [ARG ...]`

Prints the specified `ARGs` to `stdout`

```
Options:
      -n    suppress trailing newline
      -e    interpret backslash-escaped characters (i.e. \t=tab etc)
      -E    disable interpretation of backslash-escaped characters

      Example:
                $ echo "Erik is cool"
                Erik is cool
                $  echo -e "Erik
                is
                cool"
                Erik
                is
                cool
                $ echo "Erik
                is
                cool"
                Erik
                is
                cool
```

5.3.12 env

Syntax: `env [-] [-iu] [name=value ...] [command]`

Prints the current environment or runs a program after setting up the specified environment.

```
        Options:
                -, -i   start with an empty environment
                -u      remove variable from the environment
```

5.3.13 expr

Syntax: `expr EXPRESSION`

Prints the value of `EXPRESSION` to standard output.

```
EXPRESSION may be:

ARG1 |  ARG2    ARG1 if it is neither null nor 0, otherwise ARG2
ARG1 &  ARG2    ARG1 if neither argument is null or 0, otherwise 0
ARG1 <  ARG2    ARG1 is less than ARG2
ARG1 <= ARG2    ARG1 is less than or equal to ARG2
ARG1 =  ARG2    ARG1 is equal to ARG2
ARG1 != ARG2    ARG1 is unequal to ARG2
ARG1 >= ARG2    ARG1 is greater than or equal to ARG2
ARG1 >  ARG2    ARG1 is greater than ARG2
```

```
ARG1 +  ARG2    arithmetic sum of ARG1 and ARG2
ARG1 -  ARG2    arithmetic difference of ARG1 and ARG2
ARG1 *  ARG2    arithmetic product of ARG1 and ARG2
ARG1 /  ARG2    arithmetic quotient of ARG1 divided by ARG2
ARG1 %  ARG2    arithmetic remainder of ARG1 divided by ARG2
STRING : REGEXP               anchored pattern match of REGEXP in STRING
match STRING REGEXP           same as STRING : REGEXP
substr STRING POS LENGTH      substring of STRING, POS counted from 1
index STRING CHARS            index in STRING where any CHARS is found,
                              or 0
length STRING                 length of STRING
quote TOKEN                   interpret TOKEN as a string, even if
                              it is a keyword like `match' or an
                              operator like `/'
( EXPRESSION )                value of EXPRESSION
```

Beware that many operators need to be escaped or quoted for shells. Comparisons are arithmetic if both ARGs are numbers, else lexicographical. Pattern matches return the string matched between \(and \) or null; if \(and \) are not used, they return the number of characters matched or 0.

5.3.14 false

Syntax: `false`

Return an exit code of `FALSE (1)`.

Example:

```
$ false
$ echo $?
1
```

5.3.15 fbset

Syntax: `fbset [options] [mode]`

Show and modify frame buffer settings.

Example:

```
$ fbset
mode "1024x768-76"
# D: 78.653 MHz, H: 59.949 kHz, V: 75.694 Hz
geometry 1024 768 1024 768 16
timings 12714 128 32 16 4 128 4
accel false
rgba 5/11,6/5,5/0,0/0
endmode
```

5.3.16 find

Syntax: `find [PATH...] [EXPRESSION]`

Search for files in a directory hierarchy. The default `PATH` is the current directory; default `EXPRESSION` is `'-print'`

`EXPRESSION may consist of:`

```
-follow          Dereference symbolic links.
-name PATTERN    File name (leading directories removed) matches PATTERN.
-print           Print (default and assumed).
-type X          Filetype matches X (where X is one of: f,d,l,b,c,...)
-perm PERMS      Permissions match any of (+NNN); all of (-NNN);
                 or exactly (NNN)
-mtime TIME      Modified time is greater than (+N); less than (-N);
                 or exactly (N) days
```

Example:

```
$ find / -name /etc/passwd
/etc/passwd
```

5.3.17 grep

Syntax: `grep [-ihHnqvs] pattern [files...]`

Search for `PATTERN` in each `FILE` or standard input.

`Options:`

```
-H      prefix output lines with filename where match was found
-h      suppress the prefixing filename on output
-i      ignore case distinctions
-l      list names of files that match
-n      print line number with output lines
-q      be quiet. Returns 0 if result was found, 1 otherwise
-v      select non-matching lines
-s      suppress file open/read error messages
```

Example:

```
$ grep root /etc/passwd
root:x:0:0:root:/root:/bin/bash
$ grep ^[rR]oo. /etc/passwd
root:x:0:0:root:/root:/bin/bash
```

5.3.18 halt

Syntax: `halt`

Halt the system.

5.3.19 head

Syntax: `head [OPTION] [FILE]...`

Print first 10 lines of each `FILE` to standard output. With more than one `FILE`, precede each with a header giving the file name. With no `FILE`, or when `FILE` is -, read standard input.

`Options:`

```
-n NUM    Print first NUM lines instead of first 10
```

Example:

```
$ head -n 2 /etc/passwd
root:x:0:0:root:/root:/bin/bash
```

```
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
```

5.3.20 hostname

Syntax: `hostname [OPTION] {hostname | -F file}`

Get or set the hostname or DNS domain name. If a hostname is given (or a file with the -F parameter), the host name will be set.

```
Options:
      -s              Short
      -i              Addresses for the hostname
      -d              DNS domain name
      -F, --file FILE Use the contents of FILE to specify the hostname

Example:
            $ hostname
            slag
```

5.3.21 id

Syntax: `id [OPTIONS]... [USERNAME]`

Print information for USERNAME or the current user

```
Options:
      -g      prints only the group ID
      -u      prints only the user ID
      -n      print a name instead of a number (with for -ug)
      -r      prints the real user ID instead of the effective ID (with -ug)

      Example:
            $ id
            uid=1000(andersen) gid=1000(andersen)
```

5.3.22 init

Syntax: `init`

`init` is the parent of all processes. This version of `init` is designed to be run only by the kernel.

BusyBox `init` doesn't support multiple runlevels. The runlevels field of the `/etc/inittab` file is completely ignored by BusyBox init. If you want runlevels, use `sysvinit`.

BusyBox init works just fine without an inittab. If no inittab is found, it has the following default behavior:

```
::sysinit:/etc/init.d/rcS
::askfirst:/bin/sh
::ctrlaltdel:/sbin/reboot
::shutdown:/sbin/swapoff -a
::shutdown:/bin/umount -a -r
```

If it detects that `/dev/console` is _not_ a serial console, it will also run:

```
tty2::askfirst:/bin/sh
tty3::askfirst:/bin/sh
tty4::askfirst:/bin/sh
```

If you choose to use an `/etc/inittab` file, the inittab entry format is as follows:

```
<id>:<runlevels>:<action>:<process>
<id>:
```

WARNING!
This field has a non-traditional meaning for BusyBox init! The `id` field is used by BusyBox init to specify the controlling tty for the specified process to run on. The contents of this field are appended to `"/dev/"` and used as-is. There is no need for this field to be unique, although if it isn't you may have strange results. If this field is left blank, the controlling `tty` is set to the console. Also note that if BusyBox detects that a serial console is in use, then only entries whose controlling `tty` is either the serial console or `/dev/null` will be run. BusyBox init does nothing with `utmp`. We don't need a `utmp`.

`<runlevels>:`

The `runlevels` field is completely ignored.

`<action>:`

Valid actions include: `sysinit, respawn, askfirst, wait, once, ctrlaltdel, and shutdown`.

The available actions can be classified into two groups: actions that are run only once, and actions that are re-run when the specified process exits.

`Run only-once actions:`

`sysinit` is the first item run on boot. `init` waits until all `sysinit` actions are completed before continuing. Following the completion of all `sysinit` actions, all `wait` actions are run.

'`wait`' actions, such as `sysinit` actions, cause `init` to wait until the specified task completes. 'once' actions are asyncronous, therefore, `init` does not wait for them to complete. `'ctrlaltdel'` actions are run when the system detects that someone on the system console has pressed the CTRL-ALT-DEL key combination. Typically one wants to run `'reboot'` at this point to cause the system to reboot. Finally the `'shutdown'` action specifies the actions to be taken when `init` is told to reboot. Unmounting filesystems and disabling `swap` is a very good here.

`Run repeatedly actions:`

`'respawn'` actions are run after the `'once'` actions. When a process started with a `'respawn'` action exits, `init` automatically restarts it. Unlike `sysvinit`, BusyBox `init` does not stop processes from respawning out of control. The `'askfirst'` actions act just like `respawn`, except that before running the specified process it displays the line `"Please press Enter to activate this console"` and then waits for the user to press enter before starting the specified process.

Unrecognized actions (like `initdefault`) will cause `init` to emit an error message, and then go along with its business. All actions are run in the reverse order from how they appear in `/etc/inittab`.

`<process>:`

Specifies the process to be executed and it's command line.

`Example /etc/inittab file:`

```
# This is run first except when booting in single-user mode.
#
::sysinit:/etc/init.d/rcS

# /bin/sh invocations on selected ttys
#
# Start an "askfirst" shell on the console (whatever that may be)
::askfirst:-/bin/sh
# Start an "askfirst" shell on /dev/tty2-4
tty2::askfirst:-/bin/sh
tty3::askfirst:-/bin/sh
tty4::askfirst:-/bin/sh

# /sbin/getty invocations for selected ttys
#
tty4::respawn:/sbin/getty 38400 tty5
tty5::respawn:/sbin/getty 38400 tty6

# Example of how to put a getty on a serial line (for a terminal)
#
#::respawn:/sbin/getty -L ttyS0 9600 vt100
#::respawn:/sbin/getty -L ttyS1 9600 vt100
#
# Example how to put a getty on a modem line.
#::respawn:/sbin/getty 57600 ttyS2

# Stuff to do before rebooting
::ctrlaltdel:/sbin/reboot
::shutdown:/bin/umount -a -r
::shutdown:/sbin/swapoff -a
```

5.3.23 ln

Syntax: `ln [OPTION] TARGET... LINK_NAME|DIRECTORY`

Create a link named `LINK_NAME` or `DIRECTORY` to the specified `TARGET`

You may use `'--'` to indicate that all following arguments are non-options.

```
Options:
      -s      make symbolic links instead of hard links
      -f      remove existing destination files
      -n      no dereference symlinks - treat like normal file

Example:
      $ ln -s BusyBox /tmp/ls
      $ ls -l /tmp/ls
      lrwxrwxrwx    1 root     root  7 Apr 12 18:39 ls -> BusyBox*
```

5.3.24 logger

Syntax: `logger [OPTION]... [MESSAGE]`

Write `MESSAGE` to the system log. If `MESSAGE` is omitted, log `stdin`.

```
Options:
      -s    Log to stderr as well as the system log.
      -t    Log using the specified tag (defaults to user name).
      -p    Enter the message with the specified priority.
            This may be numerical or a ``facility.level'' pair.

Example:
            $ logger "hello"
```

5.3.25 logname

Syntax: `logname`

Print the name of the current user.

```
Example:
            $ logname
            root
```

5.3.26 logread

Syntax: `logread`

Shows the messages from `syslogd` (using circular buffer).

5.3.27 ls

Syntax: `ls [-1AacCdeFilnpLRrSsTtuvwxXhk] [filenames...]`

List directory contents.

```
Options:
            -1      list files in a single column
            -A      do not list implied . and ..
            -a      do not hide entries starting with .
            -C      list entries by columns
            -c      with -l: show ctime
            -d      list directory entries instead of contents
            -e      list both full date and full time
            -F      append indicator (one of */=@|) to entries
            -i      list the i-node for each file
            -l      use a long listing format
            -n      list numeric UIDs and GIDs instead of names
            -p      append indicator (one of /=@|) to entries
            -L      list entries pointed to by symbolic links
            -R      list subdirectories recursively
            -r      sort the listing in reverse order
            -S      sort the listing by file size
            -s      list the size of each file, in blocks
            -T NUM  assume Tabstop every NUM columns
            -t      with -l: show modification time
```

```
        -u      with -l: show access time
        -v      sort the listing by version
        -w NUM  assume the terminal is NUM columns wide
        -x      list entries by lines instead of by columns
        -X      sort the listing by extension
        -h      print sizes in human readable format (e.g., 1K 243M 2G )
        -k      print sizes in kilobytes(default)
```

5.3.28 mkdir

Syntax: `mkdir [OPTION] DIRECTORY...`

Create the `DIRECTORY(ies)`, if they do not already exist.

```
Options:
      -m      set permission mode (as in chmod), not rwxrwxrwx - umask
      -p      no error if existing, make parent directories as needed

Example:
            $ mkdir /tmp/foo
            $ mkdir /tmp/foo
            /tmp/foo: File exists
            $ mkdir /tmp/foo/bar/baz
            /tmp/foo/bar/baz: No such file or directory
            $ mkdir -p /tmp/foo/bar/baz
```

5.3.29 mkfifo

Syntax: `mkfifo [OPTIONS] name`

Creates a named pipe (identical to `'mknod name p'`)

```
Options:
      -m      create the pipe using the specified mode (default a=rw)
```

5.3.30 mknod

Syntax: `mknod [OPTIONS] NAME TYPE MAJOR MINOR`

Create a special file (block, character, or pipe).

```
Options:

      -m      create the special file using the specified mode (default a=rw)

TYPEs include:

      b:      Make a block (buffered) device.
      c or u: Make a character (un-buffered) device.
      p:      Make a named pipe. MAJOR and MINOR are ignored for named pipes.

      Example:
            $ mknod /dev/fd0 b 2 0
            $ mknod -m 644 /tmp/pipe p
```

5.3.31 mount

Syntax: `mount [flags] device directory [-o options,more-options]`

Mount a filesystem.

```
Flags:
      -a:             Mount all filesystems in fstab.
      -f:             "Fake" Add entry to mount table but don't mount it.
      -n:             Don't write a mount table entry.
      -o option:      One of many filesystem options, listed below.
      -r:             Mount the filesystem read-only.
      -t fs-type:     Specify the filesystem type.
      -w:             Mount for reading and writing (default).

Options for use with the "-o" flag:
      async/sync:     Writes are asynchronous / synchronous.
      atime/noatime:  Enable / disable updates to inode access times.
      dev/nodev:      Allow use of special device files / disallow them.
      exec/noexec:    Allow use of executable files / disallow them.
      loop:           Mounts a file via loop device.
      suid/nosuid:    Allow set-user-id-root programs / disallow them.
      remount:        Re-mount a mounted filesystem, changing its flags.
      ro/rw:          Mount for read-only / read-write.

There are EVEN MORE flags that are specific to each filesystem. see the
written documentation for those.

      Example:
                  $ mount
                  /dev/hda3 on / type minix (rw)
                  proc on /proc type proc (rw)
                  devpts on /dev/pts type devpts (rw)
                  $ mount /dev/fd0 /mnt -t msdos -o ro
                  $ mount /tmp/diskimage /opt -t ext2 -o loop
```

5.3.32 mv

Syntax: `mv SOURCE DEST or: mv SOURCE... DIRECTORY`

Rename `SOURCE` to `DEST`, or move `SOURCE(s)` to `DIRECTORY`.

```
      Example:
                  $ mv /tmp/foo /bin/bar
```

5.3.33 poweroff

Syntax: `poweroff`

Halt the system and request that the kernel shut off the power.

5.3.34 printf

Syntax: `printf FORMAT [ARGUMENT...]`

Formats and prints `ARGUMENT(s)` according to `FORMAT`, where `FORMAT` controls the output exactly as in `C printf`.

```
Example:
            $ printf "Val=%d
            " 5
            Val=5
```

5.3.35 pwd

Syntax: `pwd`

Print the full filename of the current working directory.

```
Example:
            $ pwd
            /root
```

5.3.36 rdate

Syntax: `rdate [OPTION] HOST`

Get and possibly set the system date and time from a remote `HOST`.

```
Options:
      -s      Set the system date and time (default).
      -p      Print the date and time.
```

5.3.37 reboot

Syntax: `reboot`

Reboot the system.

5.3.38 rm

Syntax: `rm [OPTION]... FILE...`

Remove (unlink) the FILE(s). You may use `'--'` to indicate that all following arguments are non-options.

```
Options:
      -I          always prompt before removing each destinations
      -f          remove existing destinations, never prompt
      -r or –R    remove the contents of directories recursively

      Example:
                  $ rm -rf /tmp/foo
```

5.3.39 rmdir

Syntax: `rmdir [OPTION]... DIRECTORY...`

Remove the `DIRECTORY(ies)`, if they are empty.

Example:

```
# rmdir /tmp/foo
```

5.3.40 sed

Syntax: `sed [-Vhnef] pattern [files...]`

```
Options:
-n                suppress automatic printing of pattern space
-e script         add the script to the commands to be executed
-f scriptfile     add the contents of script-file to the commands to be
                  executed
-h                display this help message
```

If no -e or -f is given, the first non-option argument is taken as the sed script to interpret. All remaining arguments are names of input files; if no input files are specified, then the standard input is read.

Example:

```
$ echo "foo" | sed -e 's/f[a-zA-Z]o/bar/g'
bar
```

5.3.41 sleep

Syntax: `sleep N`

Pause for `N` seconds.

Example:

```
$ sleep 2
[2 second delay results]
```

5.3.42 sort

Syntax: `sort [-n] [-r] [FILE]...`

Sorts lines of text in the specified files.

Example:

```
$ echo -e "e
f
b
d
c
a" | sort
a
b
c
d
e
f
```

5.3.43 stty

Syntax: `stty [-a|g] [-F device] [SETTING]...`

Without arguments, prints baud rate, line discipline, and deviations from `stty sane`.

```
Options:
      -F device       open device instead of stdin
      -a              print all current settings in human-readable form
      -g              print in stty-readable form
      [SETTING]       see documentation
```

5.3.44 sync

Syntax: `sync`

Write all buffered filesystem blocks to disk.

5.3.45 tail

Syntax: `Tail [OPTION]... [FILE]...`

Print last 10 lines of each `FILE` to standard output. With more than one `FILE`, precede each with a header giving the file name. With no `FILE`, or when `FILE` is -, read standard input.

```
Options:
      -c N[kbm]       output the last N bytes
      -n N[kbm]       print last N lines instead of last 10
      -f              output data as the file grows
      -q              never output headers giving file names
      -s SEC          wait SEC seconds between reads with -f
      -v              always output headers giving file names
```

If the first character of N (bytes or lines) is a '+', output begins with the Nth item from the start of each file, otherwise, print the last N items in the file. N bytes may be suffixed by k (x1024), b (x512), or m (1024^2).

```
Example:
            $ tail -n 1 /etc/resolv.conf
            nameserver 10.0.0.1
```

5.3.46 tee

Syntax: `tee [OPTION]... [FILE]...`

Copy standard input to each `FILE`, and also to standard output.

```
Options:
      -a      append to the given FILEs, do not overwrite

Example:
            $ echo "Hello" | tee /tmp/foo
            $ cat /tmp/foo
            Hello
```

5.3.47 test

Syntax: `test EXPRESSION or [ EXPRESSION ]`

Checks file types and compares values returning an exit code determined by the value of `EXPRESSION`.

```
Example:
            $ test 1 -eq 2
            $ echo $?
            1
            $ test 1 -eq 1
            $ echo $?
            0
            $ [ -d /etc ]
            $ echo $?
            0
            $ [ -d /junk ]
            $ echo $?
            1
```

5.3.48 touch

Syntax: `touch [-c] file [file ...]`

Update the last-modified date on the given file[s].

```
Options:
      -c      Do not create any files

Example:
            $ ls -l /tmp/foo
            /bin/ls: /tmp/foo: No such file or directory
            $ touch /tmp/foo
            $ ls -l /tmp/foo
            -rw-rw-r--    1 andersen andersen  0 Apr 15 01:11 /tmp/foo
```

5.3.49 tr

Syntax: `tr [-cds] STRING1 [STRING2]`

Translate, squeeze, and/or delete characters from standard input, writing to standard output.

```
Options:
      -c    take complement of STRING1
      -d    delete input characters coded STRING1
      -s    squeeze multiple output characters of STRING2 into one character

      Example:
            $ echo "gdkkn vnqkc" | tr [a-y] [b-z]
            hello world
```

5.3.50 true

Syntax: `true`

Return an exit code of `TRUE (0)`.

```
Example:
            $ true
            $ echo $?
            0
```

5.3.51 tty

Syntax: `tty`

Print the file name of the terminal connected to standard input.

```
Options:
      -s      print nothing, only return an exit status

Example:
            $ tty
            /dev/tty2
```

5.3.52 umount

Syntax: `umount [flags] filesystem|directory`

Unmount file systems.

```
Flags:
      -a:     Unmount all file systems in /etc/mtab
      -n:     Don't erase /etc/mtab entries
      -r:     Try to remount devices as read-only if mount is busy
      -f:     Force filesystem umount (i.e. unreachable NFS server)
      -l:     Do not free loop device (if a loop device has been used)

      Example:
            $ umount /dev/hdc1
```

5.3.53 uname

Syntax: `uname [OPTION]...`

Print certain system information. With no `OPTION`, same as -s.

```
Options:
      -a      print all information
      -m      the machine (hardware) type
      -n      print the machine's network node hostname
      -r      print the operating system release
      -s      print the operating system name
      -p      print the host processor type
      -v      print the operating system version

      Example:
            $ uname -a
```

```
Linux debian 2.2.15pre13 #5 Tue Mar 14 16:03:50 MST 2000
i686 unknown
```

5.3.54 uniq

Syntax: `uniq [OPTION]... [INPUT [OUTPUT]]`

Discard all but one of successive identical lines from `INPUT` (or standard input), writing to `OUTPUT` (or standard output).

```
Options:
      -c      prefix lines by the number of occurrences
      -d      only print duplicate lines
      -u      only print unique lines

Example:
            $ echo -e "a
            a
            b
            c
            c
            a" | sort | uniq
            a
            b
            c
```

5.3.55 usleep

Syntax: `usleep N`

Pause for `N` microseconds.

```
Example:
            $ usleep 1000000
            [pauses for 1 second]
```

5.3.56 wc

Syntax: `wc [OPTION]... [FILE]...`

Print line, word, and byte counts for each `FILE`, and a total line if more than one `FILE` is specified. With no `FILE`, read standard input.

```
Options:
      -c      print the byte counts
      -l      print the newline counts
      -L      print the length of the longest line
      -w      print the word counts

Example:
            $ wc /etc/passwd
            31      46    1365 /etc/passwd
```

5.3.57 whoami

Syntax: `whoami`

Prints the user name associated with the current effective user id.

5.3.58 xargs

Syntax: `xargs [COMMAND] [ARGS...]`

Executes `COMMAND` on every item given by standard input.

```
Example:
            $ ls | xargs gzip
            $ find . -name '*.c' -print | xargs rm
```

5.3.59 yes

Syntax: `yes [OPTION]... [STRING]...`

Repeatedly outputs a line with all specified `STRING(s)`, or `'y'`.

5.4 LIBC NSS

GNU Libc uses the Name Service Switch (NSS) to configure the behavior of the C library for the local environment, and to configure how it reads system data, such as passwords and group information. BusyBox has made it Policy that it will never use NSS, and will never use and libc calls that make use of NSS. This allows you to run an embedded system without the need for installing an /etc/nsswitch.conf file and without and /lib/libnss_* libraries installed.

If you are using a system that is using a remote LDAP server for authentication via GNU libc NSS, and you want to use BusyBox, then you will need to adjust the BusyBox source. Chances are though, that if you have enough space to install of that stuff on your system, then you probably want the full GNU utilities.

See also: textutils(1), shellutils(1), etc...

MAINTAINER
Erik Andersen <andersee@debian.org> <andersen@lineo.com>

5.4.1 Authors

The following people have contributed code to BusyBox, whether they know it or not.

Erik Andersen <andersen@lineo.com>, <andersee@debian.org>
: Tons of new stuff, major rewrite of most of the core apps, tons of new apps as noted in header files.

Edward Betts <edward@debian.org>
: expr, hostid, logname, tty, wc, whoami, yes

John Beppu <beppu@lineo.com>
: du, head, nslookup, sort, tee, uniq

Brian Candler <B.Candler@pobox.com>
tiny-ls(ls)

Randolph Chung <tausq@debian.org>
fbset, ping, hostname, and mkfifo

Dave Cinege <dcinege@psychosis.com>
more(v2), makedevs, dutmp, modularization, auto links file, various fixes, Linux Router Project maintenance

Karl M. Hegbloom <karlheg@debian.org>
cp_mv.c, the test suite, various fixes to utility.c, &c.

Daniel Jacobowitz <dan@debian.org>
mktemp.c

Matt Kraai <kraai@alumni.carnegiemellon.edu>
documentation, bugfixes

John Lombardo <john@deltanet.com>
dirname, tr

Glenn McGrath <bug1@netconnect.com.au>
ar.c

Vladimir Oleynik <dzo@simtreas.ru>
cmdedit, stty-port, locale, various fixes and irreconcilable critic of everything not perfect.

Bruce Perens <bruce@pixar.com>
Original author of BusyBox. His code is still in many apps.

Chip Rosenthal <chip@unicom.com>, <crosenth@covad.com>
wget - Contributed by permission of Covad Communications

Pavel Roskin <proski@gnu.org>
Lots of bugs fixes and patches.
Gyepi Sam <gyepi@praxis-sw.com>
Remote logging feature for syslogd

Linus Torvalds <torvalds@transmeta.com>
mkswap, fsck.minix, mkfs.minix

Mark Whitley <markw@lineo.com>
sed remix, bug fixes, style-guide, etc.

Charles P. Wright <cpwright@villagenet.com>
gzip, mini-netcat(nc)

Enrique Zanardi <ezanardi@ull.es>
tarcat (since removed), loadkmap, various fixes, Debian maintenance.

5.5 Using Variables

As is the case with almost any language, the use of variables is very important in shell programs. 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 wanted to assign a value of 5 to the variable count, you would enter:

```
count=5
```

Note that 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 wanted to print the value stored in the count variable to the screen, you would enter the following command:

```
echo $count
```

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

5.6 Built-in Shell Variables

The shell is aware of special kinds of variable called **positional parameters**. Positional parameters are used to refer to the parameters 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 parameter is stored in a variable named 1; the second parameter is stored in a variable named 2, and so on. The shell reserves these variable names so that you can't use them as variables you define. To access the values stored in these variables, you must precede the variable name with a dollar sign ($), just as you do with variables you define.

Variable Symbol	Description
`$?`	Contains the exit value returned by the last executed command
`$$`	Contains the process ID number of the shell
`$!`	The process number of the most recent asynchronously executed command.
`$-`	Contains the flags that were passed to the shell when it was invoked or flags that were set using the set command.
`$#`	Contains the number of arguments to the shell.
`$*`	Contains the 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 ...."`

Variable Symbol	Description
$@	Contains the 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, $@ divides arguments containing spaces into separate arguments.

5.7 The importance of Quotation Marks

The use of the different types of quotation marks is very important in shell programming. To perform different functions, the shell uses both kinds of quotation marks and the backslash character. The double quotation marks (""), the single quotation marks ('), and the backslash (\) are all used to hide special characters from the shell. Each of these methods hides varying degrees of special characters from the shell.

Double quotation marks are the least powerful of the three methods. When you surround characters with double quotes, all the white space characters are hidden from the shell, but all other special characters are still 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 wanted to assign the string hello world to the variable hello, you would type the following command:

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

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

Single quotes are the most powerful form of quoting. They hide all special characters from the shell. This is useful if the command you enter is intended for a program other than the shell.

```
greeting="hello there $LOGNAME"
```

This would store the value `hello there root` into the variable `greeting` if you were logged in as `root`. If you tried to write this command using single quotes it wouldn't work, because the single quotes would hide the dollar sign from the shell and the shell wouldn't know that it was supposed to perform a variable substitution. The variable `greeting` would be assigned the value `hello there $LOGNAME` if you wrote the command using single quotes.

Backslash quoting is used most often when you want to hide only a single character from the shell. This is usually done when you want to include a special character in a string. 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
```

The backslash in this example would hide the dollar sign from the shell. If the backslash were not there, the shell would try to find a variable named 5 and perform a variable substitution on that variable. Assuming that no variable named 5 were defined, the shell would assign a value of .00 to the `disk_price` variable. This is because the shell would substitute a value of `null` for the $5 variable.

The back quote marks (') perform a different function. They are used when you want to use the results of a command in another command. For example, if you wanted to set the value of the variable contents equal to the list of files in the current directory, you would type the following command:

```
contents='ls'
´
```

5.8 The test Command

A command called `test` is used to 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. The `test` command has the following syntax:

```
test expression
```

or

```
[ expression ]
```

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.

5.8.1 Integer operators

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

5.8.2 String operators

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

5.8.3 File operators

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

5.8.4 Logical operators

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

5.8.5 Conditional statements

The `if` statement

The syntax of the `if` statement is:

```
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 is an abbreviation of `else if`. This statement is executed only if none of the expressions associated with the `if` statement or any `elif` statements before it were 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 were true.

The `case` statement

The case statement enables you to compare a pattern with several other patterns and execute a block of code if a match is found. The syntax of the case statement is:

```
case string in
str1)
commands;;
str2)
commands;;
*)
commands;;
esac
```

`string` is compared to `str1` and `str2`. If one of these strings matches `string1`, 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 because the asterisk matches all strings.

The `for` statement

The `for` statement executes the commands that are contained within it a specified number of times. The syntax of the for statement is:

```
for var1 in list
do
commands
done
```

In this form, the `for` statement executes once for each item in the 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.

The second form of `for` statement has the following syntax:

```
for var1
do
statements
done
```

In this form, the `for` statement executes once for each item in the variable `var1`. When this syntax of the `for` statement is used, the shell program assumes that the `var1` variable contains all the positional parameters that were passed in to the shell program on the command line.

The `while` statement

Another iteration statement offered by the shell programming language is the `while` statement. This statement causes a block of code to be executed while a provided conditional expression is true. The syntax for the `while` statement is the following:

```
while [ expression ];
do
statements
done
```

The `until` statement

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. The syntax for the `until` statement is:

```
until [ expression ]
do
commands
done
```

In practice the `until` statement is not very useful, because any `until` statement you write can also be written as a `while` statement.

The `shift` command

The `shift` command moves the current values stored in the positional parameters to the left one position. For example, if the values of the current positional parameters are

```
$1 = -r $2 = file1 $3 = file2
```

and you executed the `shift` command

```
shift
```

the resulting positional parameters would be:

```
$1 = file1 $2 = file2
```

You can also move the positional parameters over more than 1 place, by specifying a number with the `shift` command. The following would shift the positional parameters two places:

```
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 identifiedwhich option should be coming next.

6 Shell Script Examples

The following pages contain complete shell scripts, which are customized for the most common applications.

Shell Scripting Tips

- Include `#!/bin/mish -x` to debug your scripts. Run via Telnet.
- A very useful program for fetching or uploading scripts to the product, *Ultra Edit*, can be downloaded from http://www.ultraedit.com/

6.1 The configuration file

The configuration file is located in the `/etc/applications` directory. At the beginning of each script, a configuration file is sourced, to identify the variables inside.

This file allows you to use the same scripts, but with different parameters for each of them. A typical example is to upload files to a particular ftp-server for each camera. You would then create different config files and call the script with the correct parameters.

```
# Read the config file
. /etc/applications/config_$1

# Read the optional config file
if [ -e /etc/applications/pre_config_$1 ]; then
. /etc/applications/pre_config_$1
fi
```

Edit your own parameters below:

```
# --- General camera parameters ---
# The index of the name(s) of the buffer(s) started
sources="1"
# The format of the images specified to be taken according to HTTP-API
image_format="fullsize"
# The name to be given to the local file, excluding the extension
file_format="snapshot"
# Number of pre alarm images to be taken
pre=2
# Number of post alarm images to be taken
post=2
# Delay between pre-images in milliseconds
predelay=1000
# Delay between post-images in milliseconds
postdelay=1000
# Delay between image taken
delay=2000
# The time in seconds to stay in the script. A value of -1 means
# indefinitely
time=10
# The index of the name(s) of the buffer(s) started for sequential
# images
```

```
seq_sources="1"
# The format of the images specified to be taken according to the HTTP-
# API for the sequential images
seq_image_format="fullsize"

# The type of the suffix to use on the uploaded files. Either "date"
# for the date, or "sequence" for an index limited by $countermax or
# "sequenced_max" for an index up to the internal maximum integer
suffix="sequenced_max"
# Define the maximum value of the counter
counter_max=200

# --- FTP parameters ---
# The server to upload to
ftp_server="10.13.9.210"
# The port to connect to
port=21
# The user to login as
user=user
# The pass to use for the user
pass=pass
# Choose passive mode on ("yes") or off ("no") (See documentation on
# FTP protocol)
passive_mode="no"
# The path to append to all uploads. This path must exist on the
# server prior to upload
destination="upload/2400test"
# The path to append to sequential uploads. This path must exist on
# the server prior to upload
seq_destination="upload/sequential"

# --- SMTP parameters ---
# The server to use as mail server
smtp_server="mail.somewhere.com"
# The subject to use in the mail
subject="'Alarm'"
# The specified sender
from="someone@somewhere.com"
# The specified receiver of reply
reply="someone@somewhere.com"
# The specified receiver of a copy of this mail
cc="someone@somewhere.com "
# The body to insert into the mail. Note that this must be specified
# and point to a valid file
body="/tmp/var/log/messages"
# The specified recipient
to="someone@somewhere.com"
```

6.2 Script examples

The script examples that follow are located in the ROM file system of the Axis camera/video server. The scripts are, in their default implementations, "configured" by the Application Wizard, which is available from the product's configuration pages. The scripts are shown here as they actually are, and the configuration options described won't actually change the scripts themselves.

It is, of course, also possible to write your own custom scripts from scratch, or to modify the existing ones as required. The new or modified script must then be placed in the read/write area of the flash memory: e.g. `/etc/script.`

6.2.1 Example 1 – Upload via FTP

This script will upload 2 pre-alarm and 2 post-alarm images via FTP. It is fairly general and may be used in many different configurations.

The following variables defined in the camera/video server's config file are used in this script:

```
# --- General camera parameters ---
# The index of the name(s) of the buffer(s) started
sources="1"
# The format of the images specified to be taken according to HTTP-API
image_format="fullsize"
# Number of pre alarm images to be taken
pre=2
# Number of post alarm images to be taken
post=2
# Delay between pre-images in milliseconds
predelay=1000
# Delay between post-images in milliseconds
postdelay=1000

# --- FTP parameters ---
# The server to upload to
ftp_server="10.13.9.210"
# The port to connect to
port=21
# The user to login as
user=user
# The password to use for the user
pass=pass
# Choose passive mode on ("yes") or off ("no") (See documentation on
# FTP protocol)
passive_mode="no"
# The path to append to all uploads. This path must exist on the
# server prior to upload
destination="upload/2400test"
```

The script itself continues:

```
#!/bin/mish
PATH=/bin:/sbin:/usr/bin:/usr/sbin

# Read the config file
. /etc/applications/config_$1

# Read the optional config file
if [ -e /etc/applications/pre_config_$1 ]; then
. /etc/applications/pre_config_$1
fi

# Stop the, by utask, started buffers
bufferd -stop -buffername $2

# Path of the status file
status_file="/tmp/$2/status"

# Wait until bufferd is ready with the images, i.e. the status file is
# present and delete it after
expr $predelay + $postdelay > /tmp/A
tmp=`cat /tmp/A`; expr $tmp / 1000 > /tmp/A
tmp=`cat /tmp/A`; rm /tmp/A
while [ ! -f $status_file ]; do
sleep $tmp
done
rm $status_file

# FTP connection
if [ x$passive_mode = xyes ]; then
sftpclient -L -s -m $ftp_server –n $port -c $destination -k /tmp/$2 -u $user
-w $pass
else
sftpclient -L -m $ftp_server –n $port -c $destination -k /tmp/$2 -u $user -w
$pass
fi

# Reset and restart buffers
bufferd -reset -buffername $2
bufferd -start -buffername $2 -pre "$pre" -post "$post" -predelay
"$predelay" -postdelay "$postdelay" -uri
ftp://jpg/$sources/$image_format".jpg"
```

6.2.1.1 Task.list

The first entry in the `task.list` will start a buffer (`CAM1`) that continuously fetches 2 pre-alarm images from camera 1 and is prepared to fetch 2 post-alarm images. The images are fetched at a rate of 1 frame per second (1000ms).

```
# When the server is running, start capturing images in a buffer named
# IO0 with 2 pre and post-alarm pictures, 1000 ms of delay between pre
# and post images with the specified URI.
once immune % /bin/bufferd : -start -buffername IO0 -pre 2 -post 2 -predelay
1000 -postdelay 1000 -uri ftp://jpg/1/352x288.jpg;

# If a positive transition is detected on the IO port 0, execute the
# script for the first camera and the buffer named IO0 (the same as the
```

```
# previous started one)
date(w(01,2,3,4,5,6)) pattern ((IO0:/)) immune once %
/etc/scripts/alarm_ftp_net : CAM1 IO0;
```

6.2.2 Example 2 – Upload via FTP and E-mail

This script will upload 2 pre-alarm and 2 post-alarm images via FTP and also send an e-mail as notification of the event.

The following variables defined in the config file of the camera are used in this script:

```
# --- General camera parameters ---
# The index of the name(s) of the buffer(s) started
sources="1"
# The format of the images specified to be taken according to HTTP-API
image_format="fullsize"
# Number of pre alarm images to be taken
pre=2
# Number of post alarm images to be taken
post=2
# Delay between pre-images in milliseconds
predelay=1000
# Delay between post-images in milliseconds
postdelay=1000

# --- FTP parameters ---
# The server to upload to
ftp_server="10.13.9.210"
# The port to connect to
port=21
# The user to login as
user=user
# The pass to use for the user
pass=pass
# Choose passive mode on ("yes") or off ("no") (See documentation on
# FTP protocol)
passive_mode="no"
# The path to append to all uploads. This path must exist on the
# server prior to upload
destination="upload/2400test"

# --- SMTP parameters ---
# The server to use as mail server
smtp_server="mail.somewhere.com"
# The subject to use in the mail
subject="'Alarm'"
# The specified sender
from="someone@somewhere.com"
# The specified receiver of reply
reply="someone@somewhere.com"

# The specified recipient of a copy of this mail
cc="someone@somewhere.com "
# The body to insert into the mail. Note that this must be specified and
# point to a valid file
body="/tmp/var/log/messages"
# The specified recipient
to="someone@somewhere.com"
```

```
#!/bin/mish
PATH=/bin:/sbin:/usr/bin:/usr/sbin

# Read the config file
. /etc/applications/config_$1

# Read the optional config file
if [ -e /etc/applications/pre_config_$1 ]; then
      . /etc/applications/pre_config_$1
fi

# Stop the, by utask, started buffers
bufferd -stop -buffername $2

# Test if cc is present, if yes add -c $cc to the command, otherwise
# leave blank
if [ x$cc != "x" ]; then
      copy="-c $cc"
fi
# Send the mail
smtpclient -s $subject -S $smtp_server -f $from -r $reply $copy -b $body $to
if [ $? -eq 1 ]; then
      logger -t $0[$$] SMTP failed!
fi

# Path of the status file
status_file="/tmp/$2/status"

# Wait until bufferd is ready with the images, i.e. the status file is
# present and delete it after
expr $predelay + $postdelay > /tmp/B
tmp=`cat /tmp/B`; expr $tmp / 1000 > /tmp/B
tmp=`cat /tmp/B`; rm /tmp/B
while [ ! -f $status_file ]; do
      sleep $tmp
done
rm $status_file

# FTP connection
if [ x$passive_mode = xyes ]; then
      sftpclient -L -s -m $ftp_server –n $port -c $destination -k
      /tmp/$2 –u $user -w $pass
else
      sftpclient -L -m $ftp_server –n $port -c $destination -k /tmp/$2 -u
      $user -w $pass
fi

# Reset and restart buffers
bufferd -reset -buffername $2
bufferd -start -buffername $2 -pre "$pre" -post "$post" -predelay
"$predelay" -postdelay "$postdelay" -uri
ftp://jpg/$sources/$image_format".jpg"
```

6.2.3 Example 3 – Sequential Upload via FTP

When input 0 goes high, this script will, for 10 seconds, upload images via FTP, at 2 second intervals. The uploaded images can be named by specifying a suffix. These can be date, incremental sequence number or limited sequence (overwrites files when maximum is reached).

The following variables defined in the config file of the camera are used in this script:

```
# --- General camera parameters ---
# The index of the name(s) of the buffer(s) started
sources="1"
# The format of the images specified to be taken according to HTTP-API
image_format="fullsize"
# The name to be given to the local file, excluding the extension
file_format="snapshot"
# Delay between image taken
delay=2000
# The time in seconds to stay in the script. A value of -1 means
# indefinitely
time=10
# The type of the suffix to use on the uploaded files. Either "date" for
# datum or "sequence" for an index limited by $countermax or
# sequenced_max" for an index up to the internal maximum integer
suffix="sequenced_max"
# Define the maximum value of the counter
counter_max=200

# --- FTP parameters ---
# The server to upload to
ftp_server="10.13.9.210"
# The port to connect to
port=21
# The user to login as
user=user
# The pass to use for the user
pass=pass
# Choose passive mode on ("yes") or off ("no") (See documentation on
# FTP protocol)
passive_mode="no"
# The path to append to all uploads. This path must exist on the
# server prior to upload
destination="upload/2400test"
```

The script itself:

```
#!/bin/mish

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

# Read the config file
. /etc/applications/config_$1

# Read the optional config file
if [ -e /etc/applications/pre_config_$1 ]; then
. /etc/applications/pre_config_$1
fi
```

```
# Start the buffers as specified by the parameters
bufferd -start -buffername $2 -snapshot -pre 1 -predelay "$delay" -uri
ftp://jpg/"$sources"/"$image_format".jpg -format snapshot_%y%m%d_%H%M%S.jpg

# Set the counter variable according to the file /tmp/counter
if [ -e /tmp/counter ]; then
      current_counter=`cat /tmp/counter`
else
      current_counter=001
fi

session_time=0 # Time during this session
expr $time \* 1000 > /tmp/C
time=`cat /tmp/C`; rm /tmp/C # Convert the time in milliseconds
source_file="/tmp/$2" # Path of the directory of the image
while [ ! -d $source_file ]; do # Wait the creation of the directory
done
cd $source_file # Change to the directory specified above
exit=0
while [ $exit = 0 ]; do # While no errors appeared
      # If the time to stay in the script has been reach
      if [ $session_time -ge $time ]; then
            if [ $time -ne -1000 ]; then # If not infinite
                  exit=1 # Exit the script
            fi
      # Calculate the session time
      else
            expr $session_time + $delay > /tmp/C
            session_time=`cat /tmp/C`; rm /tmp/C
      fi

      if [ $exit -ne 1 ]; then
            # Path of the directory of the image
            source_file="/tmp/$2"
            OK=0 # Image not found
            # While the image has not been found
            while [ $OK = 0 ]; do
                  tmp=`ls` # List the file on the directory
                  # If there is a file inside
                  if [ ! -z $tmp ]; then
                        # Do not take an incomplete image
                        if [ $tmp != tmpimage ]; then
OK=1 # The image has been found
                              # Path of the image
                              source_file=$source_file"/"$tmp
                        fi
                  fi
            done

            # If the uploaded file must be sequenced for an index
            if [ $suffix = sequenced_max ]; then
                  # Modify the path of the new file
                  source_file="/tmp/$2"/"snapshot_$current_counter".jpg
                  # Rename the file
                  cp $tmp $source_file; rm $tmp
                  # Increment the current counter
                  expr $current_counter + 1 > /tmp/counter
                  current_counter=`cat /tmp/counter`
                  # Change this current counter on 3 digit if necessary
                  if [ $current_counter -lt 10 ]; then
```

```
                        current_counter="00$current_counter"
                        echo $current_counter > /tmp/counter
                  else
                        if [ $current_counter -lt 100 ]; then
                              current_counter="0$current_counter"
                              echo $current_counter > /tmp/counter
                        fi
                  fi
            fi

            # If the uploaded file must be sequenced for an index
            # limited by the value of the variable $counter_max
            if [ $suffix = sequenced ]; then
                  # If the limit is reached restart current counter at 1
                  if [ $current_counter -gt $counter_max ]; then
                        current_counter=001
                  fi
                  # Modify the path of the new file
                  source_file="/tmp/$2"/"snapshot_$current_counter".jpg
                  # Rename the file
                  cp $tmp $source_file; rm $tmp
                  # Increment the current counter
                  expr $current_counter + 1 > /tmp/counter
                  current_counter=`cat /tmp/counter`
                  # Change this current counter on a 3 digit if
                  # necessary
                  if [ $current_counter -lt 10 ]; then
                        current_counter="00$current_counter"
                        echo $current_counter > /tmp/counter
                  else
                        if [ $current_counter -lt 100 ]; then
                              current_counter="0$current_counter"
                              echo $current_counter > /tmp/counter
                        fi
                  fi
            fi
# FTP connection
            if [ x$passive_mode = xyes ]; then # Passive mode
                  sftpclient -L -s -m $ftp_server -n $port -c
                  $destination -k /tmp/$2 -u $user -w $pass
            else # Non passive mode
                  sftpclient -L -m $ftp_server -n $port -c $destination
                  -k /tmp/$2 -u $user -w $pass
            fi
            if [ $? -eq 1 ]; then # If an error appeared
                  logger -t $0[$$] FTP failed! # Log this error
                  exit=1 # Exit the script
            else
            rm $source_file # Delete the files which has been uploaded
            fi
      fi
done

# Reset buffers
bufferd -reset -buffername $2
```

6.2.3.1 Task.list

This is the `task.list`. used to run the script above.

```
# If a positive transition is detected on the IO port 0, execute the
# script for the first camera and the buffer named SNAP1
date(w(0,1,2,3,4,5,6)) pattern ((IO0:/)) immune once %
/etc/scripts/seq_ftp_net : CAM1 SNAP1;
```

6.2.4 Example 4 – Upload Images via E-mail

This script will send 4 (2 pre-alarm and 2 post-alarm) images as attachments in an e-mail.

The following variables defined in the config file of the camera are used:

```
# --- General camera parameters ---
# The index of the name(s) of the buffer(s) started
sources="1"
# The format of the images specified to be taken according to HTTP-API
image_format="fullsize"
# Number of pre alarm images to be taken
pre=2
# Number of post alarm images to be taken
post=2
# Delay between pre images in milliseconds
predelay=1000
# Delay between post-images in milliseconds
postdelay=1000

# --- SMTP parameters ---
# The server to use as mail server
smtp_server="mail.somewhere.com"
# The subject to use in the mail
subject="'Alarm'"
# The specified sender
from="someone@somewhere.com"
# The specified receiver of reply
reply="someone@somewhere.com"
# The specified receiver of a copy of this mail
cc="someone@somewhere.com "
# The body to insert into the mail. Note that this must be specified and
# point to a valid file
body="/tmp/var/log/messages"
# The specified recipient
to="someone@somewhere.com"
```

The script itself:

```
#!/bin/mish

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

# Read the config file
. /etc/applications/config_$1

# Read the optional config file
if [ -e /etc/applications/pre_config_$1 ]; then
      . /etc/applications/pre_config_$1
fi

# Stop the, by utask, started buffers
bufferd -stop -buffername $2

# Path of the status file
status_file="/tmp/$2/status"

# Wait until bufferd is ready with the images, i.e. the status file is
# present and delete it after
expr $predelay + $postdelay > /tmp/E
tmp=`cat /tmp/E`; expr $tmp / 1000 > /tmp/E
tmp=`cat /tmp/E`; rm /tmp/E
while [ ! -f $status_file ]; do
      sleep $tmp
done
rm $status_file

# Test if cc is present, if yes add -c $cc to the command, otherwise
# leave blank
if [ x$cc != "x" ]; then
      copy="-c $cc"
fi
# Send the mail
smtpclient -s $subject -S $smtp_server -f $from -r $reply $copy -b $body -M
2 -d /tmp/$2 $to
if [ $? -eq 1 ]; then
      logger -t $0[$$] SMTP failed!
Fi

# Reset and restart buffers
bufferd -reset -buffername $2
bufferd -start -buffername $2 -pre "$pre" -post "$post" -predelay
"$predelay" -postdelay "$postdelay" -uri
ftp://jpg/$sources/$image_format".jpg"
```

6.2.4.1 Task.list

This `task.list` can be used to run the script above.

```
# When the server is running, start capturing images in a buffer named
# IO0 with 2 pre and post-alarm pictures, 1000 ms of delay between pre
# and post images with the specified URI.
once immune % /bin/bufferd : -start -buffername IO0 -pre 2 -post 2 -predelay
```

```
1000 -postdelay 1000 -uri ftp://jpg/1/352x288.jpg;

# If a positive transition is detected on the IO port 0, execute the
# script for the first camera and the buffer named IO0 (the same as the
# previous started one)
date(w(0,1,2,3,4,5,6)) pattern ((IO0:/)) immune once %
/etc/scripts/alarm_smtp_net : CAM1 IO0;
```

6.2.5 Example 5 – Sequential Upload with Notification via E-mail

This script will upload sequential images specified via FTP. The uploaded images will be ordered in a structure defined by their date. Images from the alarm buffers will be uploaded and an e-mail will be sent, as specified by the appropriate parameters.

The following variables defined in the config file of the camera are used in this script:

```
# --- General camera parameters ---
# The index of the name(s) of the buffer(s) started
sources="1"
# The format of the images specified to be taken according to HTTP-API
image_format="fullsize"
# The name to be given to the local file, excluding the extension
file_format="snapshot"
# Number of pre alarm images to be taken
pre=2
# Number of post alarm images to be taken
post=2
# Delay between pre-images in milliseconds
predelay=1000
# Delay between post-images in milliseconds
postdelay=1000
# Delay between image taken
delay=2000
# The time in seconds to stay in the script. A value of -1 means
# indefinitely
time=10
# The index of the name(s) of the buffer(s) started for sequential
# images
seq_sources="1"
# The format of the images specified to be taken according to the HTTP-
# API for the sequential images
seq_image_format="fullsize"

# --- FTP parameters ---
# The server to upload to
ftp_server="10.13.9.210"
# The port to connect to
port=21
# The user to login as
user=user
# The pass to use for the user
pass=pass
# Choose passive mode on ("yes") or off ("no") (See documentation on
# FTP protocol)
passive_mode="no"
# The path to append to all uploads. This path must exist on the
# server prior to upload
destination="upload/2400test"
```

```
# The path to append to sequential uploads. This path must exist on the
# server prior to upload
seq_destination="upload/sequential"

# --- SMTP parameters ---
# The server to use as mail server
smtp_server="mail.somewhere.com"
# The subject to use in the mail
subject="'Alarm'"
# The specified sender
from="someone@somewhere.com"
# The specified receiver of reply
reply="someone@somewhere.com"
# The specified receiver of a copy of this mail
cc="someone@somewhere.com "
# The body to insert into the mail. Note that this must be specified and
# point to a valid file
body="/tmp/var/log/messages"
# The specified recipient
to="someone@somewhere.com"
```

The script itself:

```
#!/bin/mish

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

# Read the config file
. /etc/applications/config_$1

# Read the optional config file
if [ -e /etc/applications/pre_config_$1 ]; then
      . /etc/applications/pre_config_$1
fi

# Stop the alarm buffers
bufferd -stop -buffername $2

# Test if cc is present, if yes add -c $cc to the command, otherwise
# leave blank
if [ x$cc != "x" ]; then
      copy="-c $cc"
fi
# Send the mail
smtpclient -s $subject -S $smtp_server -f $from -r $reply $copy -b $body $to
if [ $? -eq 1 ]; then
      logger -t $0[$$] SMTP failed!
fi

# Path of the status file
status_file="/tmp/$2/status"

# Wait until bufferd is ready with the images, i.e. the status file is
# present and delete it after
expr $predelay + $postdelay > /tmp/F
tmp=`cat /tmp/F`; expr $tmp / 1000 > /tmp/F
tmp=`cat /tmp/F`; rm /tmp/F
```

```
while [ ! -f $status_file ]; do
      sleep $tmp
done
rm $status_file

# FTP connection
if [ x$passive_mode = xyes ]; then
      sftpclient -L -s -m $ftp_server -n $port -c $destination -k
      /tmp/$2 -u $user -w $pass
else
      sftpclient -L -m $ftp_server -n $port -c $destination -k /tmp/$2 –
      u $user -w $pass
fi
if [ $? -eq 1 ]; then
      logger -t $0[$$] FTP failed!
fi

# Reset and restart buffer for alarm images
bufferd -reset -buffername $2

bufferd -start -buffername $2 -pre "$pre" -post "$post" -predelay
"$predelay" -postdelay "$postdelay" -uri
ftp://jpg/$sources/$image_format".jpg"

# Start buffer for sequential images
bufferd -start -buffername $3 -snapshot -pre 1 -predelay "$delay" -uri
ftp://jpg/"$seq_sources"/"$seq_image_format".jpg

session_time=0 # Time during this session
expr $time \* 1000 > /tmp/F
time=`cat /tmp/F`; rm /tmp/F # Convert the time in milliseconds
# Path of the directory of the sequential image
source_file="/tmp/$3"
while [ ! -d $source_file ]; do # Wait the creation of the directory
done
cd $source_file # Change to the directory specified above
exit=0
while [ $exit = 0 ]; do # While no errors appeared
      # If the time to stay in the script has been reach
      if [ $session_time -ge $time ]; then
            if [ $time -ne -1000 ]; then # If not infinite
                  exit=1 # Exit the script
            fi
      # Calculate the session time
      else
            expr $session_time + $delay > /tmp/F
            session_time=`cat /tmp/F`; rm /tmp/F
      fi

      if [ $exit -ne 1 ]; then
            # Path of the directory of the image
            source_file="/tmp/$3"
            OK=0 # Image not found
            # While the image has not been found
            while [ $OK = 0 ]; do
                  tmp=`ls` # List the file on the directory
                  # If there is a file inside
                  if [ ! -z $tmp ]; then
                        # Do not take an incomplete image
                        if [ $tmp != tmpimage ]; then
```

```
                              OK=1 # The image has been found
                              # Path of the image
                              source_file=$source_file"/"$tmp
                        fi
                  fi
            done

            # FTP connection
            if [ x$passive_mode = xyes ]; then # Passive mode
                  sftpclient -L -s -m $ftp_server -n $port -c
                  $seq_destination -k /tmp/$3 -u $user -w $pass
            else # Non passive mode
                  sftpclient -L -m $ftp_server -n $port -c
                  $seq_destination -k /tmp/$3 -u $user -w $pass
            fi
if [ $? -eq 1 ]; then # If an error appeared
                  logger -t $0[$$] FTP failed! # Log this error
                  exit=1 # Exit the script
            else
                  rm $source_file # Delete the files uploaded
            fi
      fi
done

# Reset buffers
bufferd -reset -buffername $3
```

6.2.5.1 TASK.LIST

This `task.list` can be used to run the script above.

```
# When the server is running, start capturing images in a buffer named
# IO0 with 2 pre and post-alarm pictures, 1000 ms of delay between pre
# and post images with the specified URI.
once immune % /bin/bufferd : -start -buffername IO0 -pre 2 -post 2 -predelay
1000 -postdelay 1000 -uri ftp://jpg/1/352x288.jpg;

# If a positive transition is detected on the IO port 0, execute the
# script for the first camera and the buffer named IO0 (the same as the
# previous started one) and take snapshot in a buffer named SNAP3
date(w(0,1,2,3,4,5,6)) pattern ((IO0:/)) immune once %
/etc/scripts/alarm_ftp_alarm_ftp_note_smtp_net : CAM1 IO0 SNAP3;
```

7 Troubleshooting

7.1 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.

7.2 Product related problems

If you run into problems, please try these steps first, before contacting your local supplier:

1. Consult the Troubleshooting section of the product's Users Guide.
2. Visit the Axis product Support Web, available at **www.axis.com** and verify that the product contains the latest available software version. Updated trouble shooting information and the latest software for the unit can also be found here.
3. Consult the FAQ and technical notes on the Axis product support page for additional help.
4. Contact the local supplier where the product was purchased, for assistance.

The Log file can prove a useful diagnostic tool when attempting to resolve any problems that might occur. From the **Support** menu, click **Log File** to display the latest records, events, and commands executed by the product since the last **Restart** of the system. Always attach the log file and parameter list when contacting Axis support.