

# utask, task.list & user.task.list

## TABLE OF CONTENTS

- **1. OVERVIEW**
- **2. UTASK**
- **3. TASK.LIST & USER.TASK.LIST**
  - **3.1 Syntax**
  - **3.2 Event trigger condition**
    - **3.2.1 once**
    - **3.2.2 immune**
    - **3.2.3 Date & time**
    - **3.2.4 Start-up**
    - **3.2.5 Patterns**
    - **3.2.6 Input connector**
    - **3.2.7 Video loss**
    - **3.2.8 Motion detection**
    - **3.2.9 Combine trigger conditions**
    - **3.2.10 not**
  - **3.3 Program**
  - **3.4 Built-in action**
  - **3.5 Arguments**
  - **3.6 Running a PHP script**

### 1. OVERVIEW

The Task Scheduler, utask, is a process that is capable of scheduling tasks based on either time of day or external events, e.g. when a digital input goes high, or a combination of these. A task that can be scheduled by utask is an executable file in the local file system, e.g. a binary or a script. This can for example be used to start the buffering of images or the uploading of files via FTP or SMTP.

Note! Shell script and PHP support is dependent on the product and firmware version. Please check what your Axis device supports, by visiting:

[http://www.axis.com/techsup/cam\\_servers/dev/php.htm](http://www.axis.com/techsup/cam_servers/dev/php.htm)

#### 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. UTASK

During startup, utask reads the configuration files `/etc/task.list` and `/etc/user.task.list` and parses them for event entries. If the files do not exist when utask is started, utask goes into standby mode. When `/etc/task.list` or `/etc/user.task.list` has been created or modified, a SIGUSR1 needs to be sent to utask in order for it to re-read the configuration files. This can be done with the kill command or by restarting the Axis device. Finding out the process ID of the utask application and using the kill command is handled a bit different in different products depending on which shell is used.

**Example 1:** In a telnet session, using the sash shell, to a product with firmware version 2.xx.

List the processes.

```
ps
```

Output example:

PID	TTY	STAT	TIME	COMMAND
1		S	--:--	init
2		S	--:--	(kflushd)
3		S	--:--	(kswapd)
4		S	--:--	(nfsiod)
5		S	--:--	(nfsiod)
6		S	--:--	(nfsiod)
7		S	--:--	(nfsiod)
8		D	--:--	(swapper)
20		S	--:--	/bin/syslogd
21		S	--:--	/bin/parhand
22		S	--:--	/bin/camd
23		S	--:--	/bin/dnrd
24		S	--:--	/bin/sftpd
26		S	--:--	/bin/pppwrapper
27		S	--:--	/bin/focus
28		R	--:--	/bin/motion
29		S	--:--	/bin/dstd
30		S	--:--	/bin/ssid
31		S	--:--	/bin/bufferd
32		S	--:--	/bin/boa
33		S	--:--	/bin/bootpc
34		S	--:--	/bin/iod
35		R	--:--	/bin/telnetd
36		S	--:--	/bin/utask
38		S	--:--	/bin/audioid
40		R	--:--	sash

We can see that the utask application has the process id 36.

Restart process 36.

```
kill -10 36
```

**Example 2:** In a telnet session, using the shell mish, to a product with firmware version 2.xx.

List the processes (the mish shell does not support the ps command).

```
sh -c 'ps'
```

Output will be something like in example 1, see above.

Restart process 36.

```
kill -USR1 36
```

**Example 3:** In a telnet session, using the shell ash or dash, to a product with firmware version 3.xx or 4.xx.

Restart the utask process.

```
kill -USR1 `pidof utask`
```

### 3. TASK.LIST & USER.TASK.LIST

The utask application is using two configuration files, both resident in the /etc directory. The task.list file is created, deleted or edited by the system when configuring events or an application from the products web site. Thus you should use the user.task.list file, which is provided for user-defined tasks, to prevent your own entries from being overwritten. The configuration files may not be present at first, and then it/they have to be created.

**Note:** The support for user.task.list was added in firmware 2.34 and 3.10. Products with firmware 4.xx that have support for embedded scripting have always had support for the user.task.list file.

#### 3.1 Syntax

An entry in both task.list and user.task.list has the following syntax.

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

Where

- {ID} is an identification string for the entry (optional). This can be used in combination with the built-in \*sig\* action (see below).
- <event> is the event trigger condition.
- <program> is the program to start.
- <arguments> are the arguments to be passed to the <program> (optional).

The entries in the configuration files are ordered by ascending priority, i.e. later entries precede earlier entries.

#### 3.2 Event trigger condition

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

Event	Description
[hh:mm-hh:mm]	Time period, start-stop.
[hh:mm+hh:mm]	Time period, start+duration. Max duration is 168:00 (7 days). <b>Note:</b> This event is only applicable to products with firmware version 4.xx.
time(h(..)m(..)s(..))	Time (hours, minutes, seconds).
[dd/mm-dd/mm]	Date period.
date(w(..)m(..)d(..))	Date (day of week, month, date).
pattern(..)(..)(..)	External events (e.g. alarms, motion detection).
once	Start the program only once even if the trigger remains.
immune	Do never interrupt the program.
not	Invert the trigger condition.

### IMPORTANT!

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

#### 3.2.1 once

Specifying once tells utask to run the program only once as long as the trigger condition is true. Utask checks for events ten times a second if not in standby mode. Thus if an event is specified to run at a certain time, e.g. `time(h(12))`, then in theory, this would run the program up to ten times per second for an hour, since the event is active for a whole hour. The boot or start-up event is an always-true event and once will prevent the script from being restarted for eternity.

#### 3.2.2 immune

Default behaviour is that the program is run for as long as the trigger condition is true. Specifying immune will prevent the scripts from being prematurely killed if the trigger set has a very short validity. Utask also starts a limited number of child processes. When the number of child processes has been exceeded, it has to decide how to proceed. Utask usually kill the process with minor priority (if the pending task has a higher priority than this process). Specifying immune protect the child processes from being prematurely killed.

#### 3.2.3 Date & time

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

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

**Example:** Run `/etc/myscript` twice every hour between 8 am to 4 pm.

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

**Example:** Run `/etc/myscript` at noon each day.

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

**Example:** Run `/etc/myscript` every Sunday in November and December.

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

### 3.2.4 Start-up

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

**Example:** Run `/etc/myscript` once at start-up.

```
once % /etc/myscript;
```

### 3.2.5 Patterns

Utask also schedules tasks based on messages that it receives on the utasksocket. The destination of this utasksocket may differ from product to product, it can be located either in the `/tmp` directory or in the `/var/run/utask` directory.

Patterns defined in the product can be the following.

Patterns	Description
IO<n>:/	<p>The meaning of this trigger is product dependent.</p> <ul style="list-style-type: none"><li>In products using firmware version 2.xx and 3.xx: Rising edge on the input connector. This condition will very rapidly change to IO&lt;n&gt;:H.</li><li>In products using firmware version 4.xx: Activation of the input connector. This condition will remain until IO&lt;n&gt;:\ is received.</li></ul> <p>In both cases &lt;n&gt; is the number of the input, starting from 0.</p>
IO<n>:H	<p>Input is high on the input connector, where &lt;n&gt; is the number of the input, starting from 0.</p>
IO<n>:\	<p>The meaning of this trigger is product dependent.</p> <ul style="list-style-type: none"><li>In products using firmware version 2.xx and 3.xx: Falling edge on the input connector. Where &lt;n&gt; is the number of the input, starting from 0.</li><li>In products using firmware 4.xx: Inactivation of the input connector. This condition will remain until IO&lt;n&gt;:/ is received.</li></ul> <p>In both cases &lt;n&gt; is the number of the input, starting from 0.</p>
IO<n>:L	<p>Input is low on the input connector, where &lt;n&gt; is the number of the input, starting from 0.</p>
V<n>:\	<p>Video is lost. Where &lt;n&gt; is the number of the video source, starting from 0. Only applicable to video servers.</p>
V<n>:/	<p>Video is back. Where &lt;n&gt; is the number of the video source, starting from 0. Only applicable to video servers.</p>
M<n>:/	<p>Motion detection starts. Where &lt;n&gt; is the window identifier, starting from 0.</p> <p>The behaviour of this trigger is product dependent.</p> <ul style="list-style-type: none"><li>In products using firmware version 2.xx and 3.xx: The condition will very rapidly change from M&lt;n&gt;:/ to M&lt;n&gt;:H.</li><li>In products using firmware 4.xx: This condition will remain as well as the M&lt;n&gt;:H condition until M&lt;n&gt;:\ is received.</li></ul>

M<n>:H	Motion is detected. Where <n> is the window identifier, starting from 0.
M<n>:\	Motion detection stops. Where <n> is the window identifier, starting from 0.  The behaviour of this trigger is product dependent. <ul style="list-style-type: none"> <li>• In products using firmware version 2.xx and 3.xx: The condition will very rapidly change from M&lt;n&gt;:\ to M&lt;n&gt;:L.</li> <li>• In products using firmware 4.xx: This condition will remain as well as the M&lt;n&gt;:L condition until M&lt;n&gt;:/ is received.</li> </ul>
M<n>:L	No motion detection. Where <n> is the window identifier, starting from 0.

### 3.2.6 Input connector

The behaviour of the input connector trigger is product dependent.

- In products using firmware version 2.xx and 3.xx the condition of the IO connector is IO<n>:/ (rising edge) when the input connector is set to high and then the condition very rapidly changes to IO<n>:H (high) and in the same way when the input connector is set to low, the triggering event is IO<n>:\ (falling edge) and then the condition very rapidly changes to IO<n>:L (low).
- In products using firmware version 4.xx it is possible to configure the input connector to be active on open circuit or on grounded circuit. The condition IO<n>:/ means that the input connector is being **activated**, depending on configuration this could be on open circuit or on grounded circuit. When the connector is activated the condition IO<n>:/ remains as well as the IO<n>:H or IO<n>:L depending on if the signal is high or low. These conditions remain until the input connector is inactivated. When the input connector is inactivated, the condition IO<n>:\ remains as well as the IO<n>:H or IO<n>:L condition until the input connector is activated again.

**Example:** Run /etc/myscript when digital input number 2 (number 1 since they are numbered from 0) is activated (high in products using firmware version 2.xx and 3.xx).

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

**Tip:** You can test that this is working by activating the virtual input via HTTP. Type this request in the address field of a browser (change myserver to the IP address or name of your Axis video product).

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

**Example:** Run /etc/myscript when digital input number 1 and 2 are activated at the same time in products using firmware version 4.xx.

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

**Example:** Run /etc/myscript when digital input number 1 and 2 are high at the same.

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

### 3.2.7 Video loss

**Example:** Run `/etc/myscript` when video is lost on camera 4 (number 3 since they are numbered from 0).

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

### 3.2.8 Motion detection

Motion detection is handled a bit different in different products. In products using firmware version 4.xx the motion detection daemon is not started by default, it is only active if there is an application using it. Thus it is not possible to trigger on motion detection without starting it first. Sending a message using the sockclient does starting the motion daemon, see example below. The message contains the START command and also a string used to identify the daemon. The identifier can later on be used to stop the daemon in combination with the STOP command, see example on this below, under "3.2.3 not".

Also the triggering conditions for motion detection vary between products.

- In products using firmware version 2.xx and 3.xx the condition of the motion detection trigger is `M<n>:/` when motion starts and then the condition very rapidly changes to `M<n>:H` (motion detected) and in the same way when motion stops, the trigger condition is `M<n>:\`, then the condition very rapidly changes to `M<n>:L` (no motion detected).
- In products using firmware version 4.xx the condition is `M<n>:/` when motion starts and this condition remains as well as the `M<n>:H` condition (motion detected) and then these conditions remain until motion stops. When motion stops, the condition is `M<n>:\` and this condition remains as well as the `M<n>:L` (no motion detected) until motion is detected again.

**Example:** Start the motion detection daemon at start-up. Note that this is only applicable to products using firmware version 4.xx.

```
once immune % /bin/sockclient : -message "START mysript"  
/var/run/motion/requestsocket;
```

Now it is possible to trigger on motion detection in the same way in all products.

**Example:** Run `/etc/myscript` when Motion is detected in window number 3 (number 2 since they are numbered from 0).

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

### 3.2.9 Combine events

An event can be combined with other events.

**Example:** Run `/etc/myscript` when input connector 1 is activated, from 5 pm to 8 am on Monday to Friday and on all hours on Saturday and Sunday.

```
date(w(1-5)) [17:00-08:00] pattern((IO0:/)) once immune % /etc/myscript;  
date(w(0,6)) pattern((IO0:/)) once immune % /etc/myscript;
```

### 3.2.10 not

Specifying not will invert the event that is set to trigger the program. If more than one event is set to trigger the event, for example, at May 1 **and** at 8 a clock **and** when input connector 1 is activated, adding not to this will execute the event if it is not May 1 **or** it is not 8 a clock **or** input connector 1 is inactivated. Specifying not, will not invert once and immune.

**Example:** Set output 1 high if motion is detected between 08:00 and 19:00 otherwise set output 1 low. **Note:** This example is only applicable to products using firmware version 4.xx.

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

In products using firmware version 2.xx and 3.xx the execution of iod must be placed in a separate script. This utask version cannot handle ":" and "\" in the arguments.

**Example:** Set output 1 high if motion is detected between 08:00 and 19:00 otherwise set output 1 low. **Note:** This example is only applicable to products using firmware version 2.xx (to use it in a product with firmware version 3.xx, change the line "#!/bin/mish" to "#!/bin/ash" in the script myscript below).

```
[08:00-19:00] pattern((M0:H)) once immune % /etc/myscript : 1;
[08:00-19:00] pattern((M0:H)) once immune not % /etc/myscript : 0;
```

/etc/myscript:

```
#!/bin/mish
if [ x$1 = x1 ]; then
    /bin/iod -script 1:/
fi
if [ x$1 = x0 ]; then
    /bin/iod -script 1:\
fi
```

not can also be used to stop the motion detection daemon in products using firmware version 4.xx if motion detection should not be performed 24 hours a day.

**Example:** Run the motion detection daemon at 8 am to 5 pm and trigger on motion detection in window 1 during the same time. Note that this is only applicable to products using firmware version 4.xx.

```
[08:00-17:00] once immune % /bin/sockclient : -message "START myscript"
/var/run/motion/requestsocket;
[08:00-17:00] once immune not % /bin/sockclient : -message "STOP myscript"
/var/run/motion/requestsocket;
[08:00-17:00] pattern((M0:/)) once immune % /etc/myscript;
```

### 3.3 Program

When specifying a program to run, the whole path to it has to be specified.

**Example:** Use the logger application, resident in the /usr/bin directory, to write a message to the log file when video is lost on camera 1. (The logger application is useful to check that the triggers set up are working as expected).

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

The program to start must be an executable file (not applicable to PHP script, see below). The chmod command can be used to set executable permissions to a file.

**Example:** In a telnet session, make the file /etc/myscript executable.

```
chmod 755 /etc/myscript
```

Check that the file has correct permissions.

```
ls -l /etc/myscript
```

This should result in an output like this.

```
-rwxr-xr-x 1 root    root    4 May 04 09:23 myscript
```

### 3.4 Built-in action

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

**Example:** Send the signal SIGUSR1 (10) to {myscript} every 5 minutes.

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

### 3.5 Arguments

Arguments can be passed to the program after a colon.

**Example:** Run /etc/myscript when video is lost on camera 1. Give the arguments "Video lost" and 1 to the script (the arguments can be used as \$1 and \$2 in the script. \$1 will hold the value "Video lost" and \$2 will hold the value 1).

```
pattern((V0:\)) once immune % /etc/myscript : "Video lost" 1;
```

**Example:** Run the logger application at start-up and give the argument "Hello world" to it.

```
once % /usr/bin/logger : "Hello world";
```

**Example:** Start an image buffer at start-up (Note: buffed is not supported by all products). Run /etc/myscript when the input connector 1 goes high and pass the name of the image buffer as an argument to the script.

```
once % /bin/bufferd : -start -buffername ftpbuffer -uri ftp://jpg/1/fullsize.jpg;  
pattern((IO0:/)) once immune % /etc/myscript : ftpbuffer;
```

### 3.6 Running a PHP script

Some products have support for PHP3 scripting. When starting a PHP3 script from the user.task.list, the PHP application must be started first, and then the PHP3 script is passed as an argument to the PHP application. Thus the PHP3 file does not need to be executable.

**Example:** Run /etc/myscript.php3 at start-up

```
once immune % /bin/php : /etc/myscript.php3;
```

Arguments to a PHP script are passed as an additional argument to the PHP application.

**Example:** Pass the arguments event="Video lost" and camera=1 to the PHP script.

```
pattern((V0:\)) once immune % /bin/php : /etc/myscript.php3 &event="Video  
lost"&camera=1;
```