

Program 3000 - Electronic Locking Modules (ELM)
Program *ELMwatcher*
Status 09/2010

Contents

1.	Overview	4
2.	Installation and start	5
2.1.	Installation from the EMKA-ELM CD	5
2.2.	Download and Installation from the Website	5
2.3.	Start ELMwatcher	5
3.	Sending and receiving ELM notification traps	6
4.	Using ELMwatcher	7
4.1.	Main window	7
4.1.1.	Trap list panel	7
4.1.1.1.	Displaying notification traps	8
4.1.1.2.	Deleting notifications	9
4.1.2.	Console panel	10
4.2.	Options	10
4.2.1.	Preferences	10
4.2.1.1.	Display	10
4.2.1.2.	Sound	12
4.2.1.3.	Log file	13
4.2.1.4.	Command	14
4.2.1.5.	E-mail	14
4.2.1.6.	TCL	16
4.2.1.7.	SNMP	19
4.2.2.	Language	19
4.2.3.	Licensing	20
5.	Appendix	22
5.1.	A short introduction to TCL	22
5.1.1.	Basic syntax	22
5.1.2.	Variables	22
5.1.3.	Command substitution	23
5.1.4.	Quotes and braces	23
5.1.5.	Control structures	24
5.1.6.	Where do commands come from?	24
5.1.7.	Other features	25
5.2.	TCL Interface of ELMwatcher	25
5.2.1.	TCL extensions provided by ELMwatcher	25
5.2.1.1.	ew::PlaySoundFile	26
5.2.1.2.	ew::Beep	27
5.2.1.3.	ew::CreateProcess	28
5.2.1.4.	ew::ConnectMailServer	29
5.2.1.5.	ew::DisconnectMailServer	30
5.2.1.6.	ew::SendMail	31
5.2.1.7.	ew::GetBrowser	33
5.2.1.8.	ew::SnmpGetRequest	34
5.2.1.9.	ew::SnmpGetResultCount	35
5.2.1.10.	ew::SnmpGetResult	35
5.2.1.11.	ew::SnmpGetErrorMsg	36
5.2.1.12.	ew::SnmpGetErrorStatus	36

5.2.1.13.	ew::SnmpGetErrorIndex	36
5.2.1.14.	ew::Library	37
5.2.1.15.	ew::Phase	37
5.2.1.16.	ew::Trap_AgentIPAddress	38
5.2.1.17.	ew::Trap_Community	38
5.2.1.18.	ew::Trap_Enterprise	38
5.2.1.19.	ew::Trap_EventDescription	39
5.2.1.20.	ew::Trap_ReceivedAt	39
5.2.1.21.	ew::Trap_Generic	39
5.2.1.22.	ew::Trap_Specific	40
5.2.1.23.	ew::Trap_GroupNr	40
5.2.1.24.	ew::Trap_Nr	40
5.2.1.25.	ew::Trap_TimeStamp	40
5.2.1.26.	ew::Trap_ELMAAlarmLine	41
5.2.1.27.	ew::Trap_ELMAAlarmSources	41
5.2.1.28.	ew::Trap_ELMAAlarmHandles	42
5.2.1.29.	ew::Trap_ELMAAlarmUpSensors	42
5.2.1.30.	ew::Trap_ELMAAlarmLoSensors	42
5.2.1.31.	ew::Trap_TCN	43
5.2.1.32.	ew::Trap_Description	43
5.2.2.	HTMLPageDemo - an example of a notification TCL script.....	43

1. Overview

ELMwatcher is a program for monitoring SNMP Trap notification messages generated by an **EMKA Elelectronic Locking Modules (ELM)** system. It lets you manage received notifications in various ways and includes mechanisms for informing users about the notified events.

ELM Notifications can be viewed in a list window, logged to files and forwarded by E-mail.

ELMwatcher displays details for each notification message and can summarize multiple repeated notifications. The application can also be configured in a way that the main window opens and brings to foreground when a notification is received. A sound file can be played on a received notification. The notifications can also be used to invoke other programs. Furthermore, user defined Tool Command Language (TCL) scripts can be executed by *ELMwatcher* on notification. The notification data is accessible within TCL scripts. *ELMwatcher's* build in TCL interpreter provides the possibility to specify individual reactions on each notification.

2. Installation and start

The installation of *ELMwatcher* requires Microsoft Windows® 2000, Microsoft Windows® XP or Microsoft Windows® 7 on the workstation. The network protocol TCP/IP is required.

2.1. Installation from the EMKA-ELM CD

- Put the EMKA-ELM-CD into the CD-ROM-drive. If the AutoRun feature for CDs is enabled, the ELM installation program ('Install.exe') starts automatically. If you see a dialog box, then the installation program has started. If the installation program does not appear, then AutoRun may be disabled. In this case start 'Windows Explorer', double-click the CD-ROM drive to open it and double-click the 'Install.exe' file to begin the installation.
- Click on the button **ELMwatcher installation**.
- Follow the installation instructions.

2.2. Download and Installation from the Website

- Download the *ELMwatcher* installation package from the EMKA electronic Website (<http://www.emka-electronic.com>). You will get a compressed file (ZIP format).
- Extract the contained files into a directory on your hard disk.
- Start the 'Setup.exe' file.
- Follow the installation instructions.

2.3. Start ELMwatcher

Start *ELMwatcher* by double-clicking the *ELMwatcher* icon that was installed together with the software. You can find the icon either on your desktop or in the folder where you have installed the program. Alternatively, you can also start the application by using the **Start** taskbar menu button and browsing for the EMKA *ELMwatcher* application in the **Programs** submenu.

Apart from the *ELMwatcher* application, the following programs and files are also accessible from the EMKA *ELMwatcher* submenu:

- This manual as Adobe® Portable Document Format (PDF) file.
- A Tool Command Language (TCL) shell program. You may use it to learn more about TCL.
- The TCL Reference Manual.

In chapter 4.2.1.7 and in the appendix, you will find more about what can be done with TCL.

3. Sending and receiving ELM notification traps

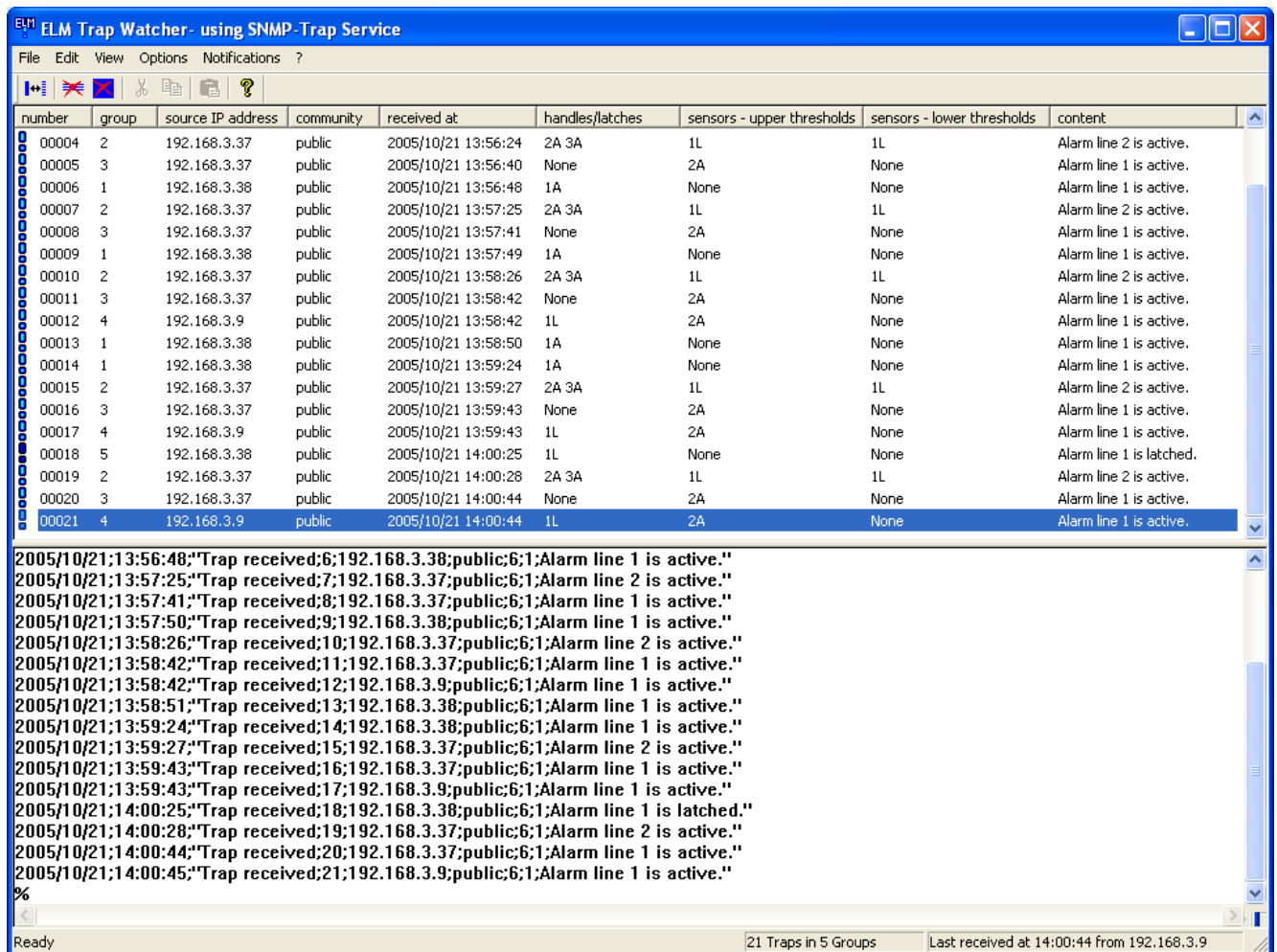
ELM notification traps are sent via network from the communication module 3000-U14 which is part of each ELM system. The ELM system must be configured to send traps on ELM alarm events. This concerns mainly the trap destination IP addresses. A trap destination is a workstation running a trap receiver program. Up to ten trap destinations can be set on an ELM system. Furthermore, the trap repetition rate can be set for each trap destination. The settings can be made for each ELM alarm line individually. The ELM SNMP Manager program *ELMcontrol* can be used to configure traps on ELM systems. See the *ELMcontrol* manual for the details.

ELMwatcher can receive traps by using the Microsoft Trap Service if this service is available (default on Windows NT/2000/XP). Alternatively, traps can be received by listening on the standard SNMP trap port (Windows 98/ME/NT/2000/XP, default on Windows 98/ME).

4. Using ELMwatcher

4.1. Main window

The main window of *ELMwatcher* consists of a Trap List window panel, displaying the list of all received traps and the console, showing event logs and allows for input of TCL commands which will be executed immediately. The title bar shows the used trap receiving mechanism. The main window also includes the menu bar; some menu commands can be quickly invoked using the toolbar buttons. The status bar on the bottom of the main window shows the state of *ELMwatcher* as well as a statistical overview of the so far received traps.



4.1.1. Trap list panel

The Trap list window panel in the upper area of the main window displays a list of all received SNMP Trap notifications if displaying is turned on. The list is updated every time a new notification is received.

On the **Display** tab of the **Preferences** window (choose **Options / Preferences** from the main menu strip) displaying the trap list can be turned on/off by marking/unmarking the checkbox **Enable list window output**.

4.1.1.1. Displaying notification traps

Two kinds of lists can be shown alternatively:

- Each received trap is shown in a separate line.
- Traps are grouped depend on there source IP address and there content (reason of trap). That means, each group contains traps repetitions. Each group is shown in a separate line.

You can switch between these two display modes by marking/unmarking the menu command **Show repeating notifications only once**. You will find this setting in the **View** menu from the main menu strip. You can also use the leftmost button of the tool bar to toggle between display modes. The display mode set via menu or tool bar will not be saved when closing *ELMwatcher*. On the **Display** tab of the **Preferences** window (**Options / Preferences** from the main menu) you will find a corresponding check box where you can set the display mode which shall be used on program start.

The trap list window panel displays the basic information about the received notification in the following columns:

Number

Shows an ongoing number of the received traps.

Group

Shows an ongoing number of the group to which the received trap belongs. All repetitions of a trap will be assigned to the same group.

Source IP address

The IP address of the ELM system which sent the trap.

Community

The SNMP community of the ELM system which sent the trap.

Received at

Date and Time at which the trap was received.

Handles/Latches

Shows a list of handles or latches which caused the alarm. Every handle or latch is indicated by its number followed by the letter 'A' if the handle/latch is in 'Active' alarm status; or followed by the letter 'L' if the handle/latch is in 'Latched' alarm status. An 'A' usually means that the handle or latch is still opened. The letter 'L' means that the handle or latch is already closed again.

Note: The option '**Find out and show alarm causes**' at the **Options / Preferences / Display** dialog must be turned on for showing alarm causes. *ELMwatcher* must have permission for read access by the ELM system (at least login level 1 is needed).

Sensors – upper thresholds

Shows a list of sensors which caused the alarm due to crossing the upper set point. Every sensor is indicated by its number followed by the letter 'A' if the sensor-threshold is in 'Active' alarm status; or followed by the letter 'L' if the sensor-threshold is in 'Latched' alarm status. An 'A' usually means that the upper threshold is still exceeded. The letter 'L' means that the sensor-value is already below the upper threshold again.

Note: The option '**Find out and show alarm causes**' at the **Options / Preferences / Display** dialog must be turned on for showing alarm causes. ELMwatcher must have permission for read access by the ELM system (at least login level 1 is needed).

Sensors – lower thresholds

Shows a list of sensors which caused the alarm due to crossing the lower set point. Every sensor is indicated by its number followed by the letter 'A' if the sensor-threshold is in 'Active' alarm status; or followed by the letter 'L' if the sensor-threshold is in 'Latched' alarm status. An 'A' usually means that the sensor-value is still below the lower threshold. The letter 'L' means that the sensor-value is already above the lower threshold again.

Note: The option '**Find out and show alarm causes**' at the **Options / Preferences / Display** dialog must be turned on for showing alarm causes. ELMwatcher must have permission for read access by the ELM system (at least login level 1 is needed).

Content

Informs about the reason of sending the trap (for example: "**Alarm line 1 is active.**").

If the trap list panel shows repeating notifications only once, the individual trap numbers are not shown. Instead, the following additional columns are shown:

First received at

Date and Time at which the first trap of the group was received.

Last received at

Date and Time at which the last trap of the group was received up till now.

Count

Shows the total number of traps of the group up till now.

You can sort the list panel on each of the columns by clicking the head of the column. A repeated click changes the sort order.

4.1.1.2. Deleting notifications

Dependend on the current display mode, you can delete one or several notifications or one or more groups of notifications from the list as follows.

Select the list entries which shall be deleted by clicking on the lines within the list. Hold CTRL key down while clicking to select several lines. To select several lines in sequence, select the first line, and then hold down Shift while clicking the last line.

From the main menu strip, choose **Notifications** and then **Delete selected notifications**. The toolbar contains a corresponding button.

You can delete the whole list by choosing **Delete all notifications** from the **Notifications** menu. The toolbar contains a corresponding button.

4.1.2. Console panel

In the Console panel window in the lower area of the main window certain events, warnings and errors are shown. These entries are logged to file(s) if so configured on the **Log file** tab of the **Preferences** window (**Options / Preferences** from the main menu). Every log entry consists of one line and comprises several paragraphs and starts always with date and time at which the event has occurred. The individual paragraphs are separated from each other by semicolon.

At the end of list a prompt is shown (%). You can enter TCL-commands to execute by *ELMwatcher's* TCL interpreter. This feature can be used, for example, when testing installed TCL scripts.

4.2. Options

4.2.1. Preferences

The Preferences window (choose **Options / Preferences** from the main menu strip) is used mainly for selecting the way of monitoring notifications. There are several kinds of output options which are grouped on different tabs. The following screen shots of the tabs show the default settings at the first program startup; only the list output and an example sound file is enabled. On subsequent startups, the settings made from the previous sessions are used.

4.2.1.1. Display

The **Display** tab of the **Preferences** window allows for settings of on-line monitoring by watching the trap list panel in the main window.



Available options on this tab are:

Enable list window output

Enables/disables output to the trap list window panel of the main window (see paragraph 4.1.1.1).

Show repeating notifications only once

Toggles display mode of the trap list window panel of the main window (see paragraph 4.1.1.1).

Delete oldest notifications if this total count is reached:

To avoid memory overflow if *ELMwatcher* is running permanently over a long period of time, the number of saved notifications in the trap list can be limited automatically. Once the given limit is reached, the oldest entry will be deleted upon each new incoming trap if this option is marked.

Delete notifications if they are older than:

If marked, all saved notifications which are older then the given periof of time will be deleted automatically.

Set ELMwatcher window to foreground

This option brings the main window to foreground if a notification is received. If *ELMwatcher* is minimized, the main window will be opened.

Select last received notification in list

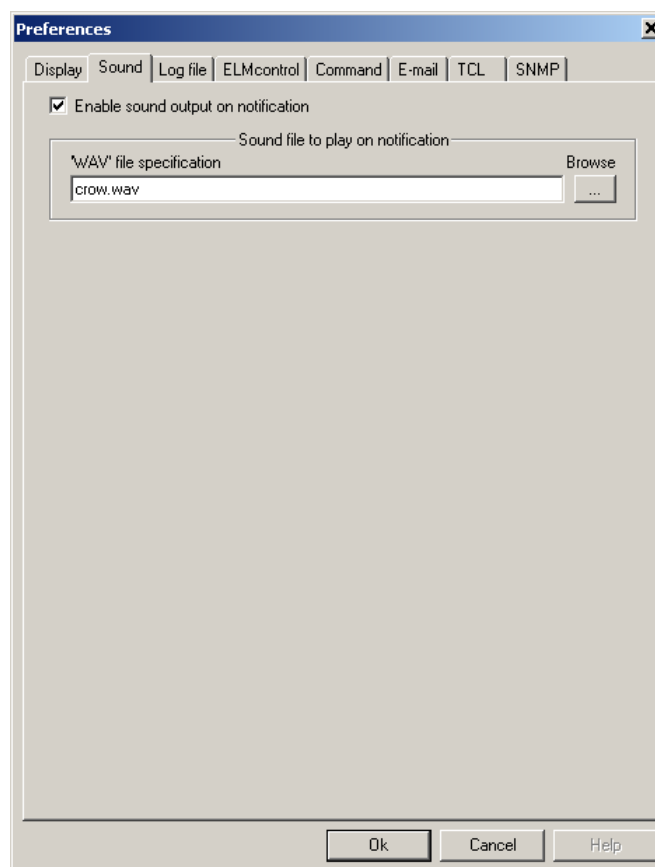
If marked and a new trap is received, the trap list window panel scrolls until the new entry appended is visible. The entry will be highlighted.

Find out and show alarm causes

If marked and a new trap is received, ELMwatcher requests the current alarm causes (Handles/Latches, Sensors) from the ELM system and shows them within the trap list window. ELMwatcher must have permission for read access by the ELM system (at least login level 1 is needed).

4.2.1.2. Sound

The **Sound** tab of the **Preferences** window allows playing of a sound file upon each incoming notification.



Available options on this tab are:

Enable sound output on notification

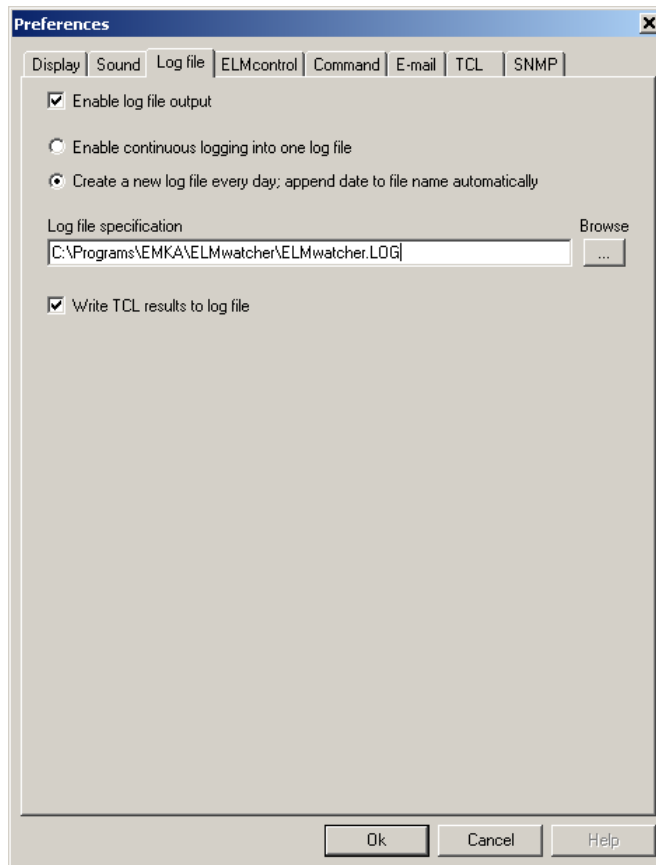
Enables/disables sound output if receiving a notification.

'WAV' file specification

If sound output is enabled, you can enter path and name of a sound file (WAV-format). The **Browse** button lets you navigate through your filesystem.

4.2.1.3. Log file

The **Log file** tab of the **Preferences** window controls event logging to file(s).



Available options on this tab are:

Enable log file output

Enables/disables event logging to file(s).

Enable continuous logging into one file / Create a new log file every day; append date to file name automatically

These radio buttons determine whether all log entries are saved into one file or whether daily a new file shall be started.

Log file specification

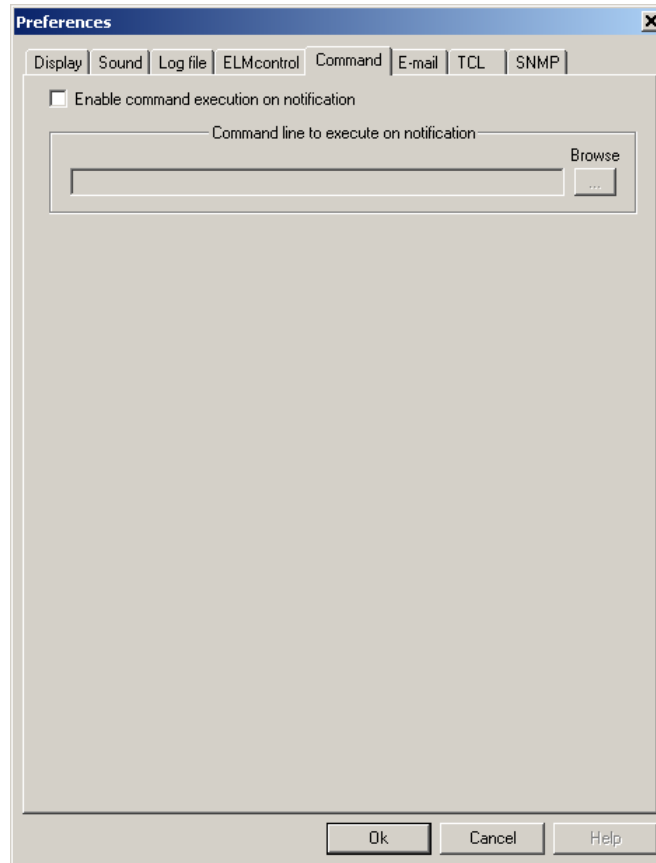
This input field allows for specification of path and file name of log file(s). If only one single file shall be written, this is the file name of this log file. Otherwise the given name is interpreted as a base name which is automatically completed with the current date to the actual used log file name. The **Browse** button lets you navigate through your filesystem.

Write TCL results to log file

If marked, output of the TCL interpreter is also saved to the log file.

4.2.1.4. Command

The Command tab allows for execution of a command line.



Available options on this tab are:

Enable command execution on notification

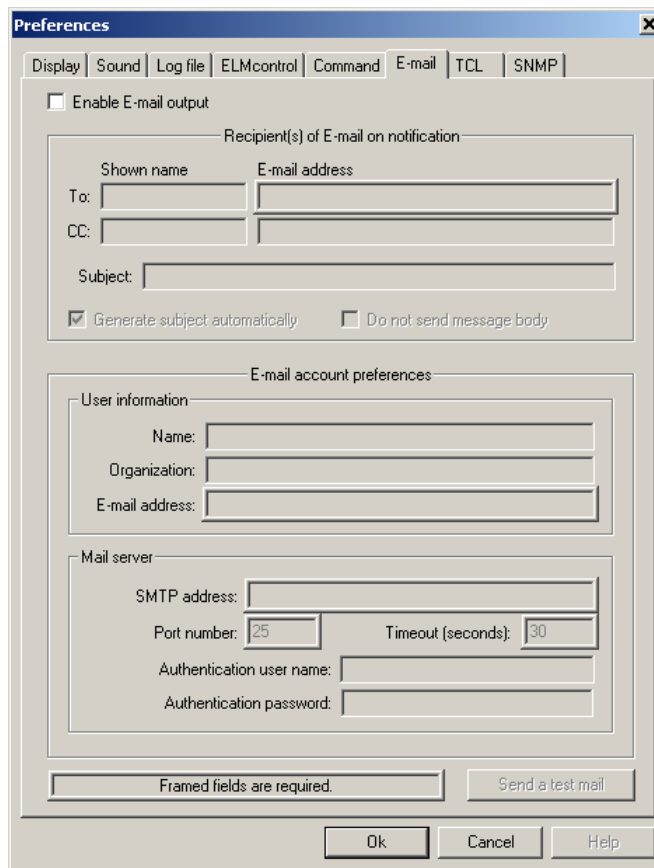
Enables/disables execution of a command line if a notification is received.

Command line to execute on notification

This input field contains the command line to execute on notification. This can be, for example, the path and name of a file of type .exe or .bat. Start parameters can be specified. Use the **Browse** button to navigate through your filesystem.

4.2.1.5. E-mail

The **E-Mail** tab contains settings for sending E-mail if a notification is received. All framed input fields are mandatory if E-mail output is enabled.



Available options on this tab are:

Enable E-mail output

Enables/disables sending of E-mail on notification.

The area **Recipient(s) of E-mail on notification** contains the settings which are relevant to the recipient(s) and the content of the E-mail.

The area **E-mail account preferences** includes the sender related settings. It contains the sub areas **User information** and **Mail server**.

A detailed explanation of the settings follows:

Shown name

Enter the name of the recipient that should be shown by the recipient's E-mail client program into the input field in the **To** row. The Shown name field is not mandatory but should be provided. You can enter the name of a carbon copy recipient in the **CC** row.

E-mail address

Enter the E-mail address of the recipient into the input field in the **To** row. You can add a carbon copy recipient by entering his E-mail address into the input field in the **CC** row.

Generate subject automatically

If marked, the subject will be created containing basic information about the received trap.



Subject

You can enter a subject text into this input field if the subject shall not be generated automatically.

Do not send message body

If marked, no message body will be sended with the E-mail. This may be useful if the E-mail is forwarded to a mobile phone. If not marked, the message body containing detailed trap information will be generated automatically.

Name, Organisation, E-mail address (User information)

Enter the name, organization name and E-mail address of the sender into the corresponding input fields. The E-mail address is mandatory.

SMTP address, Port number, Timeout (Mail server)

These informations are needed to connect the Mail Server. Enter the name or the IP address of the computer running the Mail Server into the **SMTP address** field.

The port number can be specified in the **Port number** field. Mail servers usually use port 25 which is the default setting.

If connecting the Mail Server, *ELMwatcher* waits for response until the **Timeout** value is exceeded. You can determine how long *ELMwatcher* shall wait. The default value is 30 seconds.

Authentication user name, Authentication password (Mail server)

If the Mail Server requires authentication, enter user name and password into these input fields. Otherwise leave them blank.

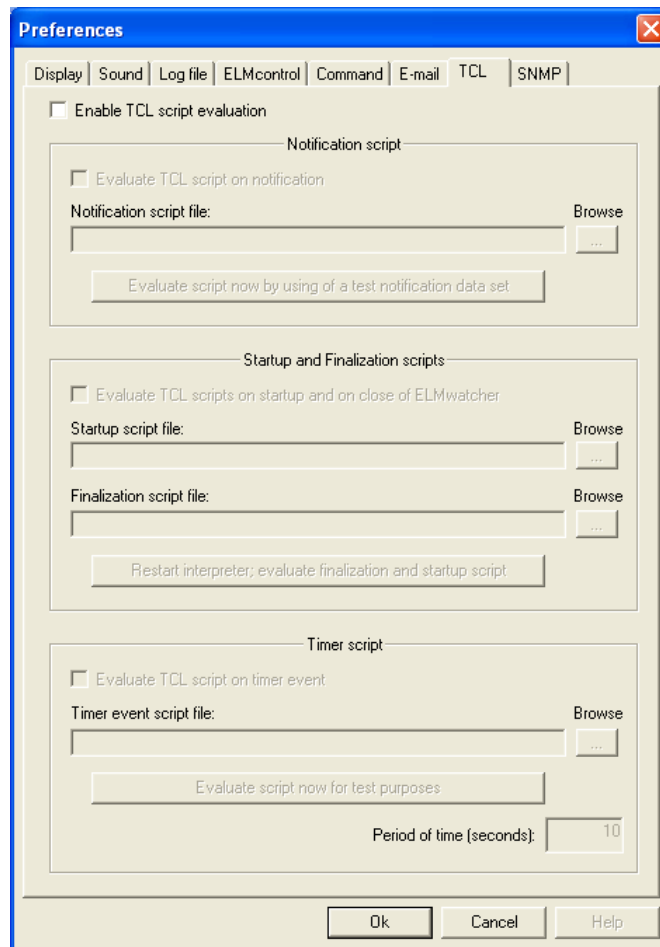
Send a test mail

You may test the settings by clicking the **Send a test mail** button. An E-mail will be sent using the settings of the E-mail tab. A test notification data set will be used for that. A message box informs you about whether sending the E-mail was succesful or not.

4.2.1.6. TCL

TCL (Tool Command Language) is easy to learn and you can create a useful program in minutes! *ELMwatcher's* build-in TCL interpreter provides the most powerful and flexible way of how to define actions on a trap notification. For example, you can define individual actions for each trap notification of each ELM system.

See the appendix of this document and the TCL/TK website <http://www.tcl.tk> for more information about TCL and a detailed description *ELMwatcher's* TCL interface.



Available options on this tab are:

Enable TCL script evaluation

Enables/disables evaluation of TCL scripts

Notification script

Evaluate TCL script on notification

Enables/disables evaluation of a TCL script in case of an incoming trap notification.

Notification script file:

This input field allows for specification of path and file name of the TCL script file that shall be evaluated in case of an incoming notification. Upon each incoming notification *ELMwatcher* updates the notification related TCL-variables (see appendix) and executes the specified script file. The **Browse** button lets you navigate through your filesystem.

Evaluate script now by using of a test notification data set

You may test the notification script file by clicking this button. A test notification data set will be used for setting up the notification related variables (see appendix).

Startup and Finalization scripts

Evaluate TCL scripts on startup and on close of ELMwatcher

Enables/disables evaluation of two special TCL script files which will be evaluated after startup and before closing *ELMwatcher*.

Startup script file:

This input field allows for specification of path and file name of the TCL script file that shall be evaluated after startup of *ELMwatcher*. Initializations which should be done only once can be placed into this script file. The **Browse** button lets you navigate through your filesystem.

Finalization script file:

This input field allows for specification of path and file name of the TCL script file that shall be evaluated before closing *ELMwatcher*. Cleanups which should be done only once can be placed into this file. The **Browse** button lets you navigate through your filesystem.

Restart interpreter; evaluate finalization and startup script

You may test the startup script file and the finalization script file by clicking this button. *ELMwatcher* evaluates the specified finalization script file and closes the TCL interpreter. After that *ELMwatcher* restarts the TCL interpreter and evaluates the entered startup script file.

Timer script

Evaluate TCL script on timer event

Enables/disables periodical evaluation of a TCL script.

Timer event script file:

This input field allows for specification of path and file name of the TCL script file that shall be evaluated periodically. The **Browse** button lets you navigate through your filesystem.

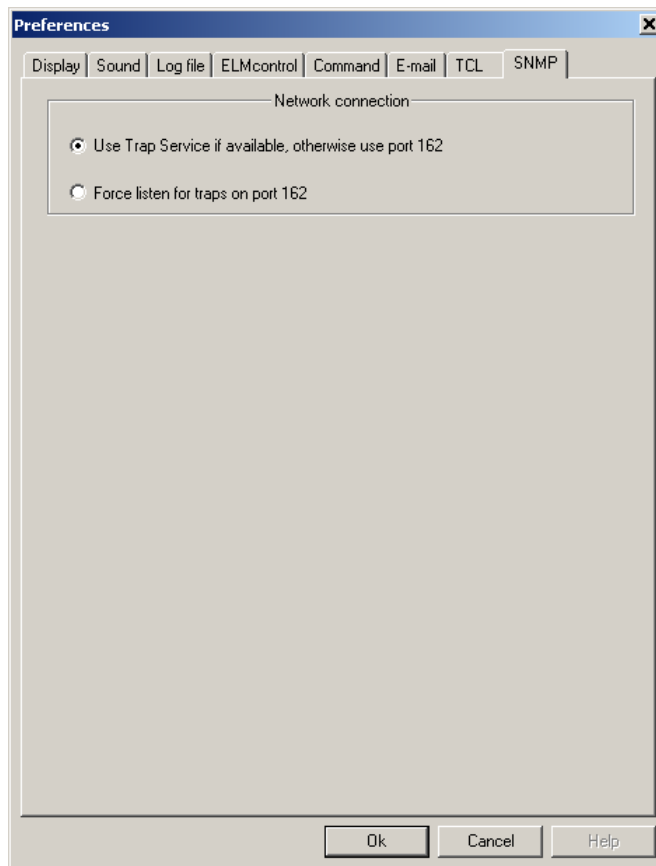
Evaluate script now for test purposes

You may test the specified timer script file by clicking this button.

Period of time (seconds)

Specification of the period of time between evaluations of the timer script file (interval in seconds). The default value is 10 seconds. The time needed for script evaluation should be shorter than the time entered in this input field.

4.2.1.7. SNMP



Available options on this tab are:

Use Trap Service if available, otherwise use port 162

This is the default setting. If the operating system of the computer supports the Microsoft SNMP TRAP Service (provided by MS-Windows NT/2000/XP systems), the TRAP Service will be used. If not (on Windows 98/ME systems), *ELMwatcher* will listen for SNMP notifications on the standard trap port 162 directly if possible.

Note, that using the TRAP Service has advantages if more than one trap receiver program shall be active on a single computer.

If the TRAP Service is not installed or deactivated on an operating system that supports it basically, an information message about installing the TRAP service is displayed. You may decide to follow the installation guidance, or select the option **Force listen for traps on port 162**.

Force listen for traps on port 162

If marked, *ELMwatcher* will always listen for SNMP notifications on the standard trap port 162 directly even if the operating system supports the Microsoft SNMP TRAP Service. You should set this option if you decide not to install or to deactivate the TRAP Service on an operating system that supports this service.

4.2.2. Language

The initial installation of *ELMwatcher* sets the workspace language to the language of the operating system. *ELMwatcher* currently supports English and German. If the installation program detects an

operating system of another language, English will be set as default. The preferred language can be selected in *ELMwatcher* as follows:

- Select the menu **Options** from the main menu strip.
- Select the menu entry **Language**.

The following window is displayed:



- Selected your preferred language and click on **Ok**.

The change of language will be effective when the program is next started.

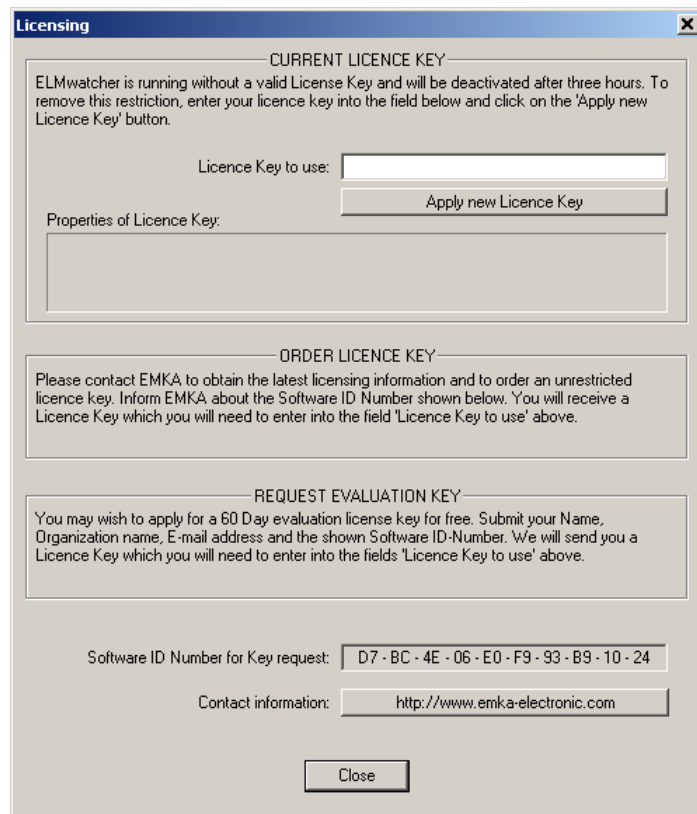
4.2.3. Licensing

ELMwatcher needs a **Licence Key** to run without restrictions. Without a License Key *ELMwatcher* can be started as often as wished, but every time it will be deactivated after three hours from start. However, there are no functional restrictions if running without a Licence Key. This allows for a full evaluation.

Please contact EMKA to obtain the latest licensing information and to order a Licence Key. You may also wish to apply for a 60 Day evaluation License Key for free. Proceed as follows:

- Select the menu **Options** from the main menu strip.
- Select the menu entry **Licensing**.

The following window is displayed:



Licensing

— CURRENT LICENCE KEY —

ELMwatcher is running without a valid License Key and will be deactivated after three hours. To remove this restriction, enter your licence key into the field below and click on the 'Apply new Licence Key' button.

Licence Key to use:

Apply new Licence Key

Properties of Licence Key:

— ORDER LICENCE KEY —

Please contact EMKA to obtain the latest licensing information and to order an unrestricted licence key. Inform EMKA about the Software ID Number shown below. You will receive a Licence Key which you will need to enter into the field 'Licence Key to use' above.

— REQUEST EVALUATION KEY —

You may wish to apply for a 60 Day evaluation license key for free. Submit your Name, Organization name, E-mail address and the shown Software ID-Number. We will send you a Licence Key which you will need to enter into the fields 'Licence Key to use' above.

Software ID Number for Key request:

Contact information:

Close

- Submit the code sequence shown within the field **Software ID Number for Key request** to EMKA.

You will obtain a similar code sequence from EMKA. This will be your Licence Key.

- Enter the obtained Licence Key into the field **Licence Key to use** and click the button **Apply new Licence Key**.

5. Appendix

The following introduction to TCL and many more information can be found at the website <http://www.tcl.tk/scripting>. The TCL Reference Manual and the TCL shell program **tclsh** are installed together with *ELMwatcher*. You may also use *ELMwatcher*'s console panel within the main window to try out TCL commands interactively. Note, that there are additional TCL functions specific to *ELMwatcher*. These functions (see chapter 5.2) are only available within *ELMwatcher*. The **tclsh** program provides only the basic TCL functionality.

5.1. A short introduction to TCL

Tcl is a very simple programming language. If you have programmed before, you can learn enough to write interesting Tcl programs within a few hours. This chapter provides a quick overview of the main features of Tcl. After reading this you'll probably be able to start writing simple Tcl scripts on your own; however, we recommend that you consult one of the many available Tcl books for more complete information.

5.1.1. Basic syntax

Tcl scripts are made up of *commands* separated by newlines or semicolons. Commands all have the same basic form illustrated by the following example:

```
expr 20 + 10
```

This command computes the sum of 20 and 10 and returns the result, 30. You can try out this example and all the others in this page by typing them to a Tcl application such as **tclsh**; after a command completes, **tclsh** prints its result.

Each Tcl command consists of one or more *words* separated by spaces. In this example there are four words: **expr**, 20, +, and 10. The first word is the name of a command and the other words are *arguments* to that command. All Tcl commands consist of words, but different commands treat their arguments differently. The **expr** command treats all of its arguments together as an arithmetic expression, computes the result of that expression, and returns the result as a string. In the **expr** command the division into words isn't significant: you could just as easily have invoked the same command as

```
expr 20+10
```

However, for most commands the word structure is important, with each word used for a distinct purpose.

All Tcl commands return results. If a command has no meaningful result then it returns an empty string as its result.

5.1.2. Variables

Tcl allows you to store values in variables and use the values later in commands. The **set** command is used to write and read variables. For example, the following command modifies the variable **x** to hold the value 32:

```
set x 32
```

The command returns the new value of the variable. You can read the value of a variable by invoking **set** with only a single argument:

```
set x
```

You don't need to declare variables in Tcl: a variable is created automatically the first time it is set. Tcl variables don't have types: any variable can hold any value.

To use the value of a variable in a command, use *variable substitution* as in the following example:

```
expr $x*3
```

When a \$ appears in a command, Tcl treats the letters and digits following it as a variable name, and substitutes the value of the variable in place of the name. In this example, the actual argument received by the expr command will be 32*3 (assuming that variable x was set as in the previous example). You can use variable substitution in any word of any command, or even multiple times within a word:

```
set cmd expr
set x 11
$cmd $x*$x
```

5.1.3. Command substitution

You can also use the result of one command in an argument to another command. This is called *command substitution*:

```
set a 44
set b [expr $a*4]
```

When a [appears in a command, Tcl treats everything between it and the matching] as a nested Tcl command. Tcl evaluates the nested command and substitutes its result into the enclosing command in place of the bracketed text. In the example above the second argument of the second set command will be 176.

5.1.4. Quotes and braces

Double-quotes allow you to specify words that contain spaces. For example, consider the following script:

```
set x 24
set y 18
set z "$x + $y is [expr $x + $y]"
```

After these three commands are evaluated variable z will have the value 24 + 18 is 42. Everything between the quotes is passed to the set command as a single word. Note that (a) command and variable substitutions are performed on the text between the quotes, and (b) the quotes themselves are not passed to the command. If the quotes were not present, the set command would have received 6 arguments, which would have caused an error.

Curly braces provide another way of grouping information into words. They are different from quotes in that no substitutions are performed on the text between the curly braces:

```
set z {$x + $y is [expr $x + $y]}
```

This command sets variable z to the value "\$x + \$y is [expr \$x + \$y]".

5.1.5. Control structures

Tcl provides a complete set of control structures including commands for conditional execution, looping, and procedures. Tcl control structures are just commands that take Tcl scripts as arguments. The example below creates a Tcl procedure called `power`, which raises a base to an integer power:

```
proc power {base p} {  
    set result 1  
    while {$p > 0} {  
        set result [expr $result * $base]  
        set p [expr $p - 1]  
    }  
    return $result  
}
```

This script consists of a single command, `proc`. The `proc` command takes three arguments: the name of a procedure, a list of argument names, and the body of the procedure, which is a Tcl script. Note that everything between the curly brace at the end of the first line and the curly brace on the last line is passed verbatim to `proc` as a single argument. The `proc` command creates a new Tcl command named `power` that takes two arguments. You can then invoke `power` with commands like the following:

```
power 2 6  
power 1.15 5
```

When `power` is invoked, the procedure body is evaluated. While the body is executing it can access its arguments as variables: `base` will hold the first argument and `p` will hold the second. The body of the `power` procedure contains three Tcl commands: `set`, `while`, and `return`. The `while` command does most of the work of the procedure. It takes two arguments, an expression (`$p > 0`) and a body, which is another Tcl script. The `while` command evaluates its expression argument using rules similar to those of the C programming language and if the result is true (nonzero) then it evaluates the body as a Tcl script. It repeats this process over and over until eventually the expression evaluates to false (zero). In this case the body of the `while` command multiplied the result value by `base` and then decrements `p`. When `p` reaches zero the result contains the desired power of `base`. The `return` command causes the procedure to exit with the value of variable `result` as the procedure's result.

5.1.6. Where do commands come from?

As you have seen, all of the interesting features in Tcl are represented by commands. Statements are commands, expressions are evaluated by executing commands, control structures are commands, and procedures are commands.

Tcl commands are created in three ways. One group of commands is provided by the Tcl interpreter itself. These commands are called *builtin commands*. They include all of the commands you have seen so far and many more (see below). The builtin commands are present in all Tcl applications. The second group of commands is created using the Tcl extension mechanism. *ELMwatcher* uses this mechanism to provide a set of extension commands which are specific to ELM trap handling. Other applications may implement other additional functionality to TCL. Thus the set of available Tcl commands varies from application to application. There are also numerous extension packages that can be incorporated into any Tcl application. One of the best known extensions is Tk, which provides powerful facilities for building graphical user interfaces. Other extensions provide object-oriented programming, database access, more graphical capabilities, and a variety of other features. One of

Tcl's greatest advantages for building integration applications is the ease with which it can be extended to incorporate new features or communicate with other resources.

The third group of commands consists of procedures created with the `proc` command, such as the `power` command created above.

5.1.7. Other features

Tcl contains many other commands besides the ones used in the preceding examples. Here is a sampler of some of the features provided by the builtin Tcl commands:

- More control structures, such as `if`, `for`, `foreach`, and `switch`.
- String manipulation, including a powerful regular expression matching facility. Arbitrary-length strings can be passed around and manipulated just as easily as numbers.
- I/O, including files on disk, network sockets, and devices such as serial ports. Tcl provides particularly simple facilities for socket communication over the Internet.
- File management: Tcl provides several commands for manipulating file names, reading and writing file attributes, copying files, deleting files, creating directories, and so on.
- Subprocess invocation: you can run other applications with the `exec` command and communicate with them while they run.
- Lists: Tcl makes it easy to create collections of values (lists) and manipulate them in a variety of ways.
- Arrays: you can create structured values consisting of name-value pairs with arbitrary string values for the names and values.
- Time and date manipulation.
- Events: Tcl allows scripts to wait for certain events to occur, such as an elapsed time or the availability of input data on a network socket.

5.2. TCL Interface of ELMwatcher

5.2.1. TCL extensions provided by ELMwatcher

Most of the output options which can be chosen at the **Preferences** window are also accessible from TCL scripts. *ELMwatcher* defines a TCL extension package named **ELMwatcher**. This package provides specific variables and functions within the namespace **ew**.

In the following the provided functions are explained in detail. Most parameters start with a hyphen (-) followed by the name of the parameter. This keyword is followed by the value of the parameter.

Parameters having a keyword can be notated in any order when passing them to a function.

Parameters to functions enclosed in question marks (?) are optional parameters. The others are mandatory.

5.2.1.1. ew::PlaySoundFile

Synopsis:

```
ew::PlaySoundFile -file file ?-sync sync?
```

Description:

The ew::PlaySoundFile function plays a sound specified by the given filename. See the **SoundDemo** script included within the *ELMwatcher* distribution. You will find it in the subdirectory “\EMKA\ELMwatcher\tcl\EWscripts\SoundDemo” from *ELMwatcher*’s installation directory.

Parameters:

-file

Specifies the sound file to play. The specified sound file must fit into available physical memory and be playable by an installed waveform-audio device driver. If no path is specified, ew::PlaySoundFile searches the following directories for sound files: the current directory; the Windows directory; the Windows system directory; directories listed in the PATH environment variable; and the list of directories mapped in a network. If it cannot find the specified sound, ew::PlaySoundFile uses the default system event sound entry instead. This parameter is mandatory.

-sync

Can be 0 or 1.

If 1, the playback will be synchronous. ew::PlaySoundFile returns after the sound event completes.

If 0, the playback will be asynchronous and ew::PlaySoundFile returns immediately. This parameter is optional. If not given, 1 is assumed.

Example:

```
ew::PlaySoundFile -file crow.wav -sync 0
```

5.2.1.2. ew::Beep

Synopsis:

```
ew::Beep ?-frequency frequency? ?-duration duration?
```

Description:

The ew::Beep function generates simple tones on the speaker. The operation is synchronous; it does not return control to its caller until the sound finishes. See the **SoundDemo** script included within the *ELMwatcher* distribution. You will find it in the subdirectory “\EMKA\ELMwatcher\tcl\EWscripts\SoundDemo” from ELMwatcher’s installation directory.

Parameters:

-frequency

Frequency of the sound, in hertz. This parameter must be in the range **37** through **32767**. This parameter is optional. If not given, **1000** is assumed.

-duration

Duration of the sound, in milliseconds. This parameter is optional. If not given, **200** is assumed.

Example:

```
ew::Beep -frequency 1500 -duration 500
```

5.2.1.3. ew::CreateProcess

Synopsis:

```
ew::CreateProcess commandline
```

Description:

The ew::CreateProcess function treats its argument as the specification of a command line to execute as a new process. TCL's build-in exec command can be used too.

Parameters:

commandline

The command line to execute. This can be, for example, the path and name of a file of type .exe or .bat. Start parameters can be specified. The first white-space – delimited token of the command line specifies the module name. If you are using a long file name that contains a space, use quoted strings to indicate where the file name ends and the arguments begin. If the file name does not contain an extension, .exe is appended. Therefore, if the file name extension is .com, this parameter must include the .com extension. If the file name ends in a period (.) with no extension, or if the file name contains a path, .exe is not appended. If the file name does not contain a directory path, the system searches for the executable file in the following sequence:

1. The directory from which *ELMwatcher* is loaded.
2. The current directory for the parent process.
3. The 32-bit Windows system directory. Windows Me/98: The Windows system directory.
4. The 16-bit Windows system directory.
5. The Windows directory.
6. The directories that are listed in the PATH environment variable.

Example:

```
ew::CreateProcess notepad
```

5.2.1.4. ew::ConnectMailServer

Synopsis:

```
ew::ConnectMailServer -server server  
                       ?-port port? ?-timeout timeout?  
                       ?-user user? ?-password password?
```

Description:

The ew::ConnectMailServer function establishes a connection to a Mail Server.

To send an E-mail, a connection to a Mail Server has to be established first by calling **ew::ConnectMailServer**. Since this must be done only once, a **startup script** can be used (see also paragraph 4.2.1.6).

Then, the **ew::SendMail** function can be called as often as needed. The **ew::SendMail** function is usually called from the **notification script**.

Finally, **ew::DisconnectMailServer** should be called to shut down the connection to the Mail Server. This can be done using a **finalization script**.

See the **EMailDemo** scripts included within the *ELMwatcher* distribution. You will find them in the subdirectory “\EMKA\ELMwatcher\tcl\EWscripts\EMailDemo” from *ELMwatcher*’s installation directory.

Parameters:

-server

The name or the IP address of the computer running the Mail Server. This parameter is mandatory.

-port

The port number of the Mail Server. This parameter is optional. If not specified, Port 25 is used.

-timeout

Determines, how long (in seconds) *ELMwatcher* waits for response from the Mail Server. This parameter is optional. If not specified, the waiting period is 30 seconds.

-user

If the Mail Server requires authentication, the user name must be specified by this argument. This parameter is optional.

-password

If the Mail Server requires authentication, the password must be specified by this argument. This parameter is optional.

Example:

```
ew::ConnectMailServer \  
  -server MYMAILSERVER \  
  -port 25 \  
  -user MyAuthenticationUserName \  
  -password MyAuthenticationPassword \  
  -timeout 20
```

;#mandatory
;#optional
;#optional
;#optional
;#optional

5.2.1.5. `ew::DisconnectMailServer`

Synopsis:

```
ew::DisconnectMailServer
```

Description:

The `ew::DisconnectMailServer` function closes the connection to a Mail Server.

To send an E-mail, a connection to a Mail Server has to be established first by calling `ew::ConnectMailServer`. Since this must be done only once, a **startup script** can be used (see also paragraph 4.2.1.6).

Then, the `ew::SendMail` function can be called as often as needed. The `ew::SendMail` function is usually called from the **notification script**.

Finally, `ew::DisconnectMailServer` should be called to shut down the connection to the Mail Server. This can be done using a **finalization script**.

See the **EMailDemo** scripts included within the *ELMwatcher* distribution. You will find them in the subdirectory "`\EMKA\ELMwatcher\tcl\EWscripts\EMailDemo`" from *ELMwatcher's* installation directory.

Example:

```
ew::DisconnectMailServer
```

5.2.1.6. ew::SendMail

Synopsis:

```
ew::SendMail -fromaddress fromaddress -toaddress toaddress  
             ?-fromname fromname? ?-toname toname?  
             ?-ccname ccname? ?-ccaddress ccaddress?  
             ?-subject subject?  
             ?-body body?  
             ?-attachment attachment?
```

Description:

The ew::SendMail sends an E-mail.

To send an E-mail, a connection to a Mail Server has to be established first by calling **ew::ConnectMailServer**. Since this must be done only once, a **startup script** can be used (see also paragraph 4.2.1.6).

Then, the **ew::SendMail** function can be called as often as needed. The **ew::SendMail** function is usually called from the **notification script**.

Finally, **ew::DisconnectMailServer** should be called to shut down the connection to the Mail Server. This can be done using a **finalization script**.

See the **EMailDemo** scripts included within the *ELMwatcher* distribution. You will find them in the subdirectory “\EMKA\ELMwatcher\tcl\EWscripts\EMailDemo” from *ELMwatcher*’s installation directory.

Parameters:

-fromaddress

The E-mail address of the sender. This parameter is mandatory.

-toaddress

The E-mail address of the recipient. This parameter is mandatory.

-fromname

The shown name of the sender. This parameter is optional, but recommended.

-toname

The shown name of the recipient. This parameter is optional, but recommended.

-ccname

The shown name of a carbon copy recipient. This parameter is optional. It can be notated more than once to specify several carbon copy recipients. If at least one ccname parameter is given, the number of ccname parameters must match the number of ccaddress parameters.

-ccaddress

The E-mail address of a carbon copy recipient. This parameter is optional. It can be notated more than once to specify several carbon copy recipients. If no ccname parameter is given, the ccaddress is used as shown name too.

-subject

The subject line of the E-mail. This parameter is optional, but recommended.

-body

The body text of the E-mail. This parameter is optional.

-attachment

A file name can be specified. The file will be sent as attachment. This parameter is optional. It can be notated more than once to specify several attachments.

Example:

```
ew::SendMail \  
-fromname "My Username" \  
-fromaddress myaddress@myemail.com \  
-toname "Someone Else" \  
-toaddress someoneelse@somewhereelse.com \  
-ccname "A CCd User" \  
-ccaddress "ccuser@ccdaddress.com" \  
-ccname "A second CCd User" \  
-ccaddress "ccuser2@ccd2address.com" \  
-subject "An important message ..." \  
-body "The following happened: ..." \  
-attachment "somepath1\\somefile1" \  
-attachment "somepath2\\somefile2"
```

5.2.1.7. ew::GetBrowser

Synopsis:

```
ew::GetBrowser
```

Description:

The ew::GetBrowser function determines the path of the default Web browser of the operating system. Can be used to start the Web browser from a script.

Example:

```
set DefaultBrowser [ew::GetBrowser]  
ew::CreateProcess [format "%s %s" $DefaultBrowser index.htm]
```

5.2.1.8. ew::SnmpGetRequest

Synopsis:

```
ew::SnmpGetRequest -agent agent -oid oid  
?-community community? ?-timeout timeout?
```

Description:

The ew::SnmpGetRequest function allows for reading of SNMP variables from any SNMP agent.

If this function succeeds then the value of the SNMP variable of the first specified OID is returned. If more than one variable has been read from the agent then the **ew::SnmpGetResult** function may be called then to retrieve the values of the remaining variables. The **ew::SnmpGetResultCount** may be used to get the number of successfully read variables.

If **ew::SnmpGetRequest** fails, an error message is returned informing about the reason. The functions **ew::SnmpGetErrorMsg**, **ew::SnmpGetErrorStatus** and **ew::SnmpGetErrorIndex** may also be used to get further error information.

Parameters:

-agent

The name or the IP address of the SNMP agent. This may be any SNMP device. This parameter is mandatory.

-oid

The Object Identifier (OID) of the SNMP variable to read the value from. The OID must be given in dotted notation with a leading dot – see example below. This parameter is mandatory. It's possible to apply this parameter more than once by using different OIDs to read out several variables at once. In this case the **ew::SnmpGetResult** function must be used to retrieve all of the values.

-community

The community string used for the read operation. This parameter is optional. If not specified, the string "public" is used. If the agent is an ELM system, a community string in form of "username:password" may be used to access the system with an registered user account.

-timeout

Determines, how long (in milliseconds) *ELMwatcher* waits for response from the SNMP agent. The value must be in the range **50** through **5000**. This parameter is optional. If not specified, the waiting period is 1000 milliseconds.

Example:

```
ew::SnmpGetRequest \  
-agent elm-system.dyndns.org \  
-oid .1.3.6.1.2.1.1.1.0 \  
-oid .1.3.6.1.2.1.1.2.0 \  
-community public \  
-timeout 1500
```

;#mandatory
;#at least one OID must be given
;#further OIDs are optional
;#optional
;#optional

5.2.1.9. ew::SnmpGetResultCount

Synopsis:

```
ew::SnmpGetResultCount
```

Description:

The ew::SnmpGetResultCount function returns the number of successfully read variables from the last ew::SNMPGetRequest call.
The ew::SnmpGetResult function may be used to retrieve the values.

Parameters:

none

5.2.1.10. ew::SnmpGetResult

Synopsis:

```
ew::SnmpGetResult ?-number number? ?-type?
```

Description:

The ew::SnmpGetResult function returns the value or the datatype of a certain variable read by the last ew::SNMPGetRequest call.

Parameters:

-number

The number of the variable. This parameter must be in range 1...n where n corresponds to the number of OIDs requested at the last successful ew::SnmpGetRequest call. If there is no variable available for the specified number then the function fails. The highest available variable number can be obtained by an ew::SnmpGetResultCount function call.
This parameter is optional. If not specified, the value for the first variable is returned.

-type

This parameter is optional. If specified, the datatype of the variable is returned instead of the value.

Examples:

```
ew::SnmpGetResult ;#return value of first variable previously read

ew::SnmpGetResult \  
-number 2 ;#return value of the second variable previously read

ew::SnmpGetResult \  
-number 2 \  
-type ;#return datatype of the second variable previously read
```

5.2.1.11. ew::SnmpGetErrorMsg

Synopsis:

`ew::SnmpGetErrorMsg`

Description:

The `ew::SnmpGetErrorMsg` function returns an error information text if the last `ew::SNMPGetRequest` has failed. Otherwise an empty text is returned.

Parameters:

none

5.2.1.12. ew::SnmpGetErrorStatus

Synopsis:

`ew::SnmpGetErrorStatus`

Description:

The `ew::SnmpGetErrorStatus` function returns the error status field of the SNMP response frame returned by the agent due to the last `ew::SNMPGetRequest` call. The error status is set by the agent to a value different from 0 if the request could not be processed successfully by the agent. If the last `ew::SNMPGetRequest` call was successful, 0 is returned.
If there wasn't any response frame received from the agent (if timeout has occurred) then -1 is returned.

Parameters:

none

5.2.1.13. ew::SnmpGetErrorIndex

Synopsis:

`ew::SnmpGetErrorIndex`

Description:

The `ew::SnmpGetErrorIndex` function returns the error index field of the SNMP response frame of the last `ew::SNMPGetRequest` call. The error index field corresponds to the number of the variable the agent could not process and starts with 0 for the first variable. If there wasn't any response frame received (if timeout has occurred) then -1 is returned. If the last `ew::SNMPGetRequest` call was successful, 0 is returned.

Parameters:

none

The following variables are provided by *ELMwatcher*:

5.2.1.14. ew::Library

Description:

The directory path to the *ELMwatcher* script library. This is the initial default directory of the TCL interpreter. *ELMwatcher* sample scripts are located there. You may store own scripts under this path but this is not a must. Default is the subdirectory “\EMKA\ELMwatcher\tcl\EWscripts” from *ELMwatcher*’s installation directory.

5.2.1.15. ew::Phase

Description:

This variable indicates the context of execution of a script. Using ew::Phase allows for writing a single script file which contains different code segments for notification, startup and finalization phases. Such a script file can be specified within the **TCL** tab of the **Preferences** window as notification, startup, finalization and timer script file.

Possible values are:

`startup, notification, finalization, interactive, timer`

Example:

```
switch $ew::Phase {  
  startup { puts "This script was called after startup of ELMwatcher" }  
  notification { puts "This script was called on a notification" }  
  finalization { puts "This script was called before closing ELMwatcher" }  
  interactive { puts "This script was started interactively from the console" }  
  timer { puts "This script is called periodically by a timer event" }  
}
```

5.2.1.16. ew::Trap_AgentIPAddress

Description:

Contains the IP address of the ELM system which has sent the last notification.

Example:

```
switch $ew::Trap_AgentIPAddress {
  192.168.1.1 {                               ;# If the IP address is 192.168.1.1 then
    ew::PlaySoundFile -file crow.wav         ;# play sound file crow.wav
  }
  192.168.1.2 {                               ;# If the IP address is 192.168.1.2 and:
    if {$ew::Trap_ELMAlarmLine == 1} {      ;# if alarm line is 1 then
      ew::Beep -frequency 2000 -duration 300 ;# tone of 2000 Hz for 300 ms
    } else {                                  ;# but, if alarm line is not 1, then
      ew::Beep -frequency 1000 -duration 600 ;# tone of 1000 Hz for 600 ms
    }
  }
}
```

5.2.1.17. ew::Trap_Community

Description:

Contains the SNMP Community of the ELM system which has sent the last received notification.

Example:

```
if { $ew::Trap_Community == "public" } {
  puts "The community is: public"
}
```

5.2.1.18. ew::Trap_Enterprise

Description:

Contains the enterprise OID number of the last received notification.

Example:

```
puts [ format "The enterprise OID is: %s" $ew::Trap_Enterprise ]
```

5.2.1.19. ew::Trap_EventDescription

Description:

Contains a description of the last received notification (for example: "Alarm line 1 is active.").

Example:

```
puts [ format "Trap content is: %s" $ew::Trap_EventDescription ]
```

5.2.1.20. ew::Trap_ReceivedAt

Description:

Contains date and time at which the last notification was received.

Example:

```
puts [ format "The notification was received at: %s" $ew::Trap_ReceivedAt ]
```

5.2.1.21. ew::Trap_Generic

Description:

Contains the generic trap type of the last received notification. RFC1157 (SNMP) defines the following generic trap types:

coldStart	0
warmStart	1
linkDown	2
linkUp	3
authenticationFailure	4
egpNeighborLoss	5
enterpriseSpecific	6

Example:

```
puts [ format "The generic trap type is: %s" $ew::Trap_Generic ]
```

5.2.1.22. ew::Trap_Specific

Description:

Contains the specific trap type of the last received notification according to RFC1157 (SNMP). See the ELM-MIB2 specification for trap-type numbers and there meaning.

Example:

```
puts [ format "The specific trap type is: %s" $ew::Trap_Specific ]
```

5.2.1.23. ew::Trap_GroupNr

Description:

Contains the group number which *ELMwatcher* has assigned to the last received trap.

Example:

```
puts [ format "The notification belongs to group: %s" $ew::Trap_GroupNr ]
```

5.2.1.24. ew::Trap_Nr

Description:

Contains the trap number which *ELMwatcher* has assigned to the last received trap.

Example:

```
puts [ format "%s notifications received" $ew::Trap_Nr ]
```

5.2.1.25. ew::Trap_TimeStamp

Description:

Contains the time elapsed between the last (re)initialization of the ELM system and the generation of the trap.

Example:

```
puts [ format "The TimeStamp of the trap source is: %s" $ew::Trap_TimeStamp ]
```

5.2.1.26. ew::Trap_ELMAAlarmLine

(obsolete; MIB1-traps only)

Description:

Contains the alarm line which causes the trap.

Example:

```
puts [ format "The alarm line is: %s" $ew::Trap_ELMAAlarmLine ]
```

5.2.1.27. ew::Trap_ELMAAlarmSources

Description:

Contains status information about the alarm sources and can have one the following numbers:

-1

Unknown: The alarm causes have not been read by ELMwatcher. The option '**Find out and show alarm causes**' at the **Options / Preferences / Display** dialog is turned off.

0

OK: The alarm causes have been successfully read from the ELM system.

1

Access denied: The alarm causes could not be read from the ELM system due to an insufficient permission (login level).

2

Read error: The read operation failed.

3

The trap was not an alarm related trap so alarm causes have not been read by ELMwatcher.

5.2.1.28. ew::Trap_ELMAAlarmHandles

Description:

Contains a string with a list of handles or latches which caused the alarm. Every handle or latch is indicated by its number followed by the letter 'A' if the handle/latch is in 'Active' alarm status; or followed by the letter 'L' if the handle/latch is in 'Latched' alarm status. An 'A' usually means that the handle or latch is still opened. The letter 'L' means that the handle or latch is already closed again. The Handles/Latches are separated by a space character. Example: "1A 2L 6A" means handles 1 and 6 are 'active' (is open now); handle 2 is 'latched' (was open but is closed now).

Note: The option '**Find out and show alarm causes**' at the **Options / Preferences / Display** dialog must be turned on for reading the alarm causes. ELMwatcher must have permission for read access by the ELM system (at least login level 1 is needed).

5.2.1.29. ew::Trap_ELMAAlarmUpSensors

Description:

Contains a string with a list of sensors which caused the alarm due to crossing the upper set point. Every sensor is indicated by its number followed by the letter 'A' if the sensor-threshold is in 'Active' alarm status; or followed by the letter 'L' if the sensor-threshold is in 'Latched' alarm status. An 'A' usually means that the upper threshold is still exceeded. The letter 'L' means that the sensor-value is already below the upper threshold again. The sensors are separated by a space character.

Example: "1A 2L 6A" means the upper set points of sensors 1 and 6 are crossed; the upper set point of sensor 2 was crossed in the past but the value is in a normal range now.

Note: The option '**Find out and show alarm causes**' at the **Options / Preferences / Display** dialog must be turned on for reading the alarm causes. ELMwatcher must have permission for read access by the ELM system (at least login level 1 is needed).

5.2.1.30. ew::Trap_ELMAAlarmLoSensors

Description:

Contains a string with a list of sensors which caused the alarm due to crossing the lower set point. Every sensor is indicated by its number followed by the letter 'A' if the sensor-threshold is in 'Active' alarm status; or followed by the letter 'L' if the sensor-threshold is in 'Latched' alarm status. An 'A' usually means that the sensor-value is still below the lower threshold. The letter 'L' means that the sensor-value is already above the lower threshold again. The sensors are separated by a space character.

Example: "1A 2L 6A" means the lower set points of sensors 1 and 6 are crossed; the lower set point of sensor 2 was crossed in the past but the value is in a normal range now.

Note: The option '**Find out and show alarm causes**' at the **Options / Preferences / Display** dialog must be turned on for reading the alarm causes. ELMwatcher must have permission for read access by the ELM system (at least login level 1 is needed).

5.2.1.31. ew::Trap_TCN

(MIB2-traps only)

Description:

Contains the type-referred component-number (e.g. handle number, sensor number or alarm line number) if the trap is caused by the component in question.

Example:

```
puts [ format "The component number of the trap is: %s" $ew::Trap_TCN ]
```

5.2.1.32. ew::Trap_Description

(MIB2-traps only)

Description:

Contains the original trap description as contained in the trap-frame.
(The text of the ew::Trap_EventDescription variable, which contains also a description, is build by ELMwatcher and may contain a translated text depending on the current user interface language setting.)

Example:

```
puts [ format "Original trap description: %s" $ew::Trap_Description ]
```

5.2.2. HTMLPageDemo - an example of a notification TCL script

The following is an example script file included within the *ELMwatcher* distribution. You can find it in the subdirectory “\EMKA\ELMwatcher\tcl\EWscripts\HTMLPageDemo” from *ELMwatcher*'s installation directory. It creates a HTML web page which contains a list of the last received notifications. The HTML file is updated on every incoming notification. Furthermore, the script starts a Web Browser and loads the HTML file into the browser. The browser will reload the file every 10 seconds showing you always an up to date list. No startup/finalization scripts are required. The file path of this file must be set as the **Notification script file** within *ELMwatcher*'s **TCL** tab of the **Preferences** window (see paragraph 4.2.1.6).

```
#####  
#ELMwatcher HTMLPageDemo.tcl sample  
#  
#EMKA electronic AG  
#http://www.emka-electronic.com  
#H. Fischer, 2004/01/09  
#  
#This ist an example of a TCL script that can be used by the EMKA ELMwatcher trap receiver.  
#It creates a list of the last received ELM-notifications and writes it out as  
#a HTML file. Furthermore, the script starts a Web Browser and loads the  
#HTML file into the browser. The browser will reload the file every 10 seconds.  
#  
#No startup/finalization scripts are required. Set the file path of this file as  
#notification script file within ELMwatcher's TCL Preferences.  
#  
#The script can be easely configured by setting of some variables located at the top of  
#the code section (see below).  
#Change the HTMLfilename variable if a different file name/path is wished.  
#The value of the MaxNotifications variable determines, how many traps will be shown.  
#Set the StartBrowser variable to a value different from "1" if the automatic browser start  
#is not wished.  
#  
#####
```

```
package require ELMwatcher 1.0
```

```
#####
```

```
set HTMLfilename "C:/ELMTraps.html" ;#The location should be changed (different from root)  
#set HTMLfilename "C:/Inetpub/wwwroot/ELM/Trap.html" ;#A sample location where a Web Server may  
;provide the file
```

```
set MaxNotifications 20 ;# max. notifications shown in the table
```

```
set StartBrowser 1 ;# determines, whether a browser shall be started (1) or not (other value)
```

```
#####
```

```
#Change to the HTMLPageDemo directory if not already done so.  
#If no path for the HTML file is given, the file will be written into this directory.  
if { [string first "HTMLPageDemo" [pwd]] < 0 } {  
    cd HTMLPageDemo  
}
```

```
#Create the list variable if it does not already exist  
if { [info exists TrapList] == 0 } {  
    set TrapList ""  
}
```

```
#Insert informations of the current trap at the top of the list  
set TrapList [linsert $TrapList 0 \  
    $ew::Trap_ReceivedAt \  
    [info hostname] \  
    $ew::Trap_AgentIPAddress \  
    $ew::Trap_Community \  
    $ew::Trap_Nr \  
    $ew::Trap_GroupNr \  
    $ew::Trap_TimeStamp \  
    $ew::Trap_EventDescription]
```

```
#Check, if the limit of the list is reached, and if so, delete the oldest entry  
set MaxListElements [expr $MaxNotifications * 8] ;# there are 8 values per trap  
if { [llength $TrapList] > $MaxListElements } {  
    set TrapList [lrange $TrapList 0 [expr $MaxListElements - 1]]  
}
```

```
#Open the HTML file for writing.  
set HTMLfile [open $HTMLfilename w ]
```

```
#Write HTML tag and header section.  
puts $HTMLfile "<html>\
```



```
<head>\
  <meta http-equiv=\"refresh\" content=\"10\">\
  <meta http-equiv=\"content-type\" content=\"text/html; charset=ISO-8859-1\">\
  <title>Notification received by ELMwatcher</title>\
</head>"

#Begin writing the body section, a heading line, and the heading row of the trap table.
puts $HTMLfile [format "\
  <body bgcolor=\"#FFFFFF0\">\
  <h2 align=\"center\"><b>ELMwatcher informs about the last %s notifications\
received from ELM systems</b></h2>\
  <table border=\"1\" cellspacing=\"1\" width=\"100%s\">\
    <tr>\
      <th>Received at</th>\
      <th>On computer</th>\
      <th>from IP address</th>\
      <th>Community</font></span></th>\
      <th>Trap number</th>\
      <th>Group number</th>\
      <th>Timestamp</th>\
      <th>Description</th>\
    </tr>" $MaxNotifications "%"]

#Write the content of the TrapList list variable into the HTML trap table.
set ListIndex 0
for {set i 0} { ($ListIndex < [llength $TrapList]) } {incr i} {
  puts $HTMLfile "<tr>" ;#write start of table row tag
  for {set j 0} { $j < 8 } {incr j} {
    puts $HTMLfile "<td align=\"center\">" ;#write start table cell tag
    puts $HTMLfile [lindex $TrapList $ListIndex]
    incr ListIndex
    puts $HTMLfile "</td>" ;#write end table cell tag
  }
  puts $HTMLfile "</tr>" ;#write end of table row tag
}

#Complete the table, the body section and the HTML document.
puts $HTMLfile "</table>\
  <p>generated by ELMwatcher's HTMLPageDemo-TCL-script</p>\
  </body>\
  </html>"

#Flush data from file buffers into the HTML file and close the file.
flush $HTMLfile
close $HTMLfile

#Get the default browser path and start the browser if this is required
if {[info exists DefaultBrowser] == 0} && ($StartBrowser == 1) {
  set DefaultBrowser [ew::GetBrowser]
  ew::CreateProcess [format "%s %s" $DefaultBrowser $HTMLfilename]
}

#End
#####
```



The generated HTML file looks like the following example:

ELMwatcher informs about the last 20 notifications received from ELM systems

Received at	On computer	from IP address	Community	Trap number	Group number	Timestamp	Description
2004/06/11 14:03:46	LT3-XP	172.16.120.210	public	36	3	901978	Alarm line 2 is active.
2004/06/11 14:03:46	LT3-XP	172.16.120.210	public	35	2	901974	Alarm line 1 is active.
2004/06/11 14:03:43	LT3-XP	172.16.200.1	public	34	1	141370	Alarm line 1 is active.
2004/06/11 14:02:46	LT3-XP	172.16.120.210	public	33	3	895880	Alarm line 2 is active.
2004/06/11 14:02:46	LT3-XP	172.16.120.210	public	32	2	895880	Alarm line 1 is active.
2004/06/11 14:02:43	LT3-XP	172.16.200.1	public	31	1	135275	Alarm line 1 is active.
2004/06/11 14:01:45	LT3-XP	172.16.120.210	public	30	3	889785	Alarm line 2 is active.
2004/06/11 14:01:45	LT3-XP	172.16.120.210	public	29	2	889785	Alarm line 1 is active.
2004/06/11 14:01:41	LT3-XP	172.16.200.1	public	28	1	129181	Alarm line 1 is active.
2004/06/11 14:00:43	LT3-XP	172.16.120.210	public	27	3	883691	Alarm line 2 is active.
2004/06/11 14:00:43	LT3-XP	172.16.120.210	public	26	2	883691	Alarm line 1 is active.
2004/06/11 14:00:40	LT3-XP	172.16.200.1	public	25	1	123086	Alarm line 1 is active.
2004/06/11 13:59:42	LT3-XP	172.16.120.210	public	24	2	877596	Alarm line 1 is active.
2004/06/11 13:59:42	LT3-XP	172.16.120.210	public	23	3	877596	Alarm line 2 is active.
2004/06/11 13:59:39	LT3-XP	172.16.200.1	public	22	1	116992	Alarm line 1 is active.
2004/06/11 13:58:43	LT3-XP	172.16.120.210	public	21	3	871502	Alarm line 2 is active.
2004/06/11 13:58:43	LT3-XP	172.16.120.210	public	20	2	871502	Alarm line 1 is active.
2004/06/11 13:58:40	LT3-XP	172.16.200.1	public	19	1	110897	Alarm line 1 is active.
2004/06/11 13:57:44	LT3-XP	172.16.120.210	public	18	3	865407	Alarm line 2 is active.
2004/06/11 13:57:44	LT3-XP	172.16.120.210	public	17	2	865407	Alarm line 1 is active.

generated by ELMwatcher's HTMLPageDemo-TCL-script