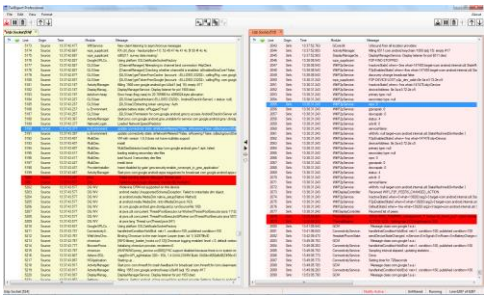
TailExpert help

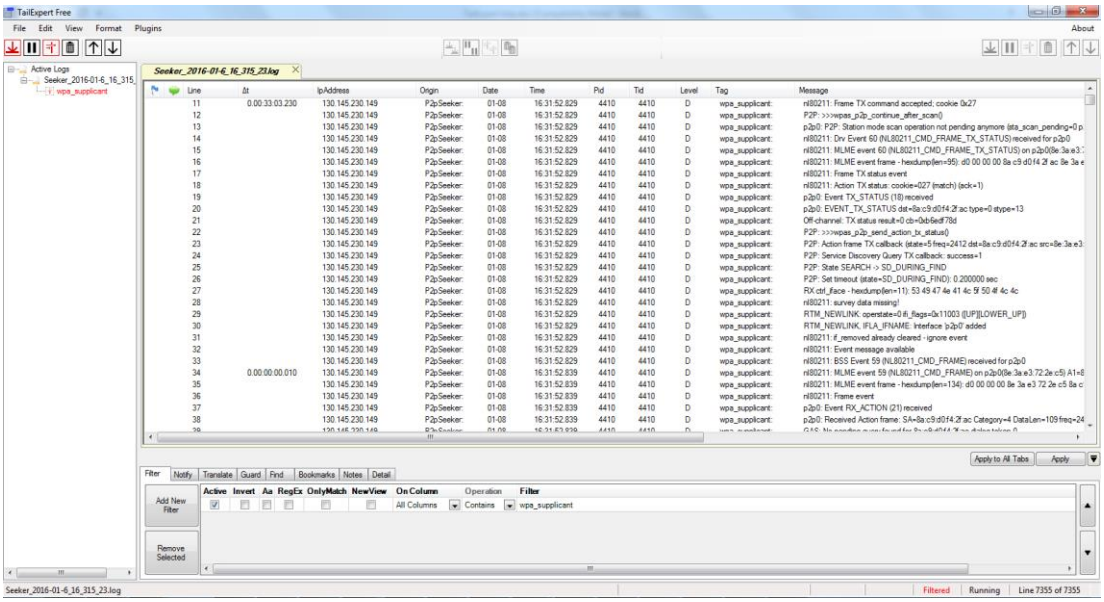
The TailExpert program is a utility that provides the ability to watch and analyse logfiles while they grow.
Features supported:



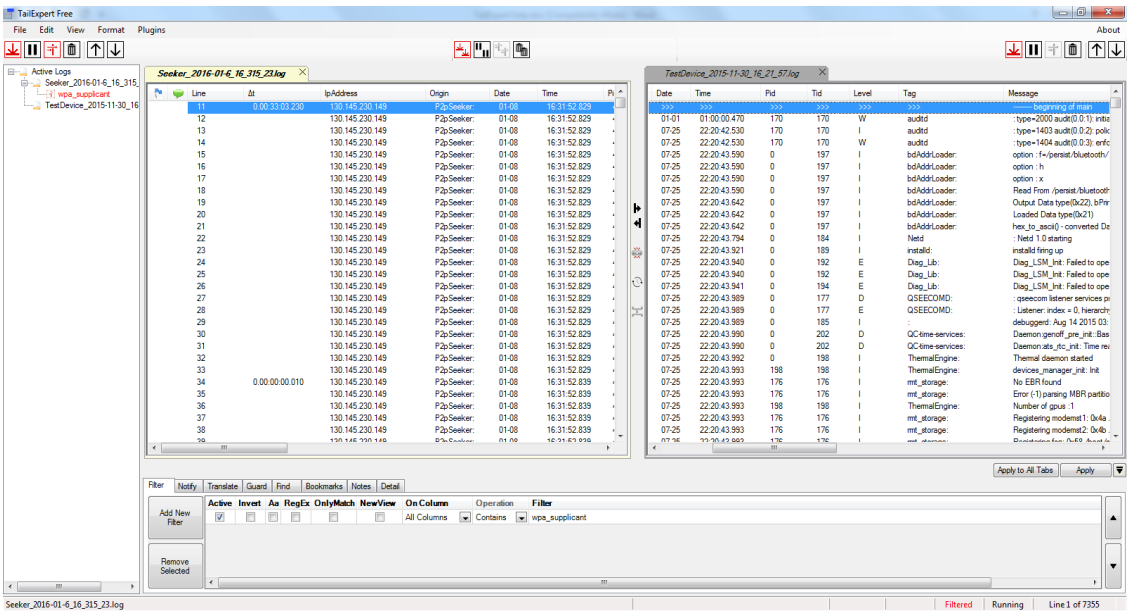
- Watch multiple files, eventlogs
- Builtin UDP network frontend (Syslog)
- Builtin Rs232 frontend
- Builtin Sftp browser
- Run lua script as a frontend
- Advanced messages filtering
- Advanced message notifications
- Place guards
- Translation of messages
- Free definable view formats, columnize loggings
- Compare logs
- Side-by-side viewing
- Advanced log analysis via embedded lua scripting
- Place bookmarks
- Place notes
- Measure timestamp delta
- Automatically sync timestamps of side by side views

Introduction

TailExpert is your tool of choice when you want to analyze your log files both offline or “in flight” just when log messages arrive. TailExpert is capable of keeping track of multiple logs at the same time even when arriving from different origin e.g. file, syslog, eventlog, Rs232. While watching the logs, all the logmessages are stored complete internally and ready to be displayed in a logview according to your preference. At any time you can pause the logviewer, keeping the incoming messages getting stored in the background. By using filters one can remove unimportant logmessages from a logview which makes analysis more comfortable. Logmessages can be made even more noticeable by adding notifications that highlight logmessages that contain words and even more complex matches by using regular expressions. TailExpert can also be used as a detection tool when notifications are added that trigger an audible alarm or sent an email when a match has occurred. This enables finding matches during overnight runs without the need to keep your watch. TailExpert also provides the option to compare two logs giving you an instant insight in changes during subsequent runs.



The program interface is consists of one or two logview windows which can hold several tabs containing a log. Left to the main window one can find the filetree viewer. The filetree viewer shows the logs currently loaded, the icons in front of the logname indicate the origin of the log viewed. TailExpert can view logs from 8 types of origin, file, adb logcat, eventlog, udp, serial, outputdbgstring, clipboard and by running a lua script. When filters have been defined for the log, the filters will show up in the filetree as siblings of the log. Clicking on the filter in the filetree activates the filter for the corresponding log (behavior of the quick filter icon can be defined in the preferences).



Underneath the logview there is the feature panel. The feature panel contains tabs for the filter, notify, translate, guard, find, bookmarks, comments and details functions. From the view menu it is possible to hide both the filetree and featurepanel hence to enlarge the number of loglines/characters of the logs viewed. At the bottom of the window there is a status panel that will show you the selected logname, error/notification messages, an indicator whether a filter/translation or notification is active and whether the tail engine is running or paused. At the right corner of the status panel the current selected line and the total number of lines is shown.

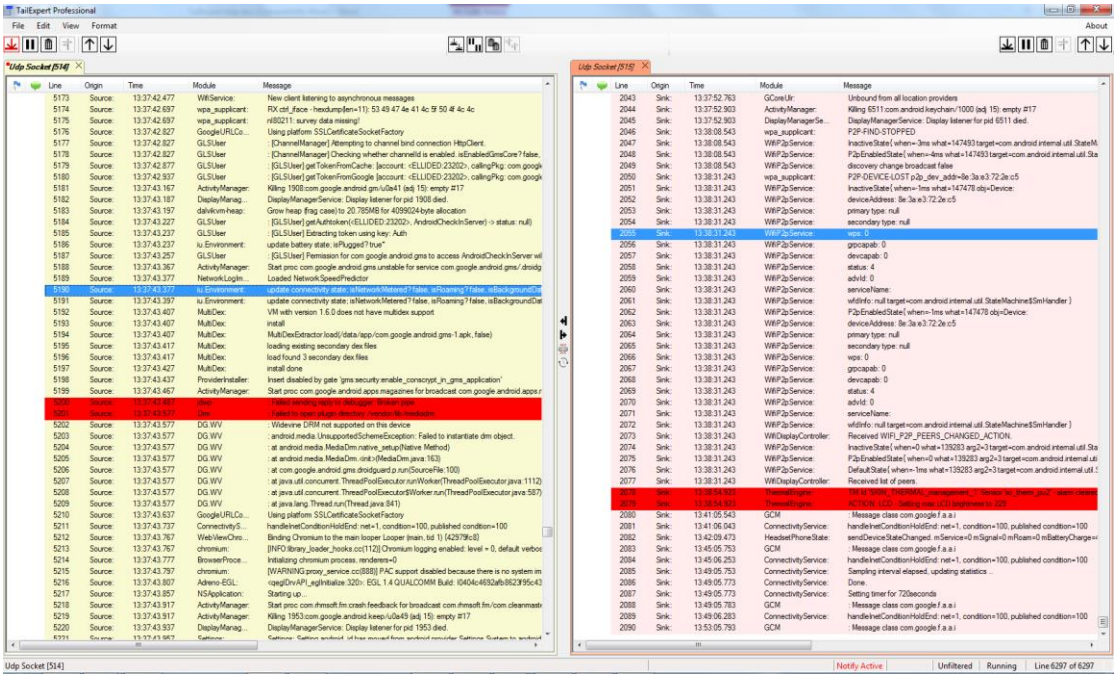
Usage

Toolbar buttons

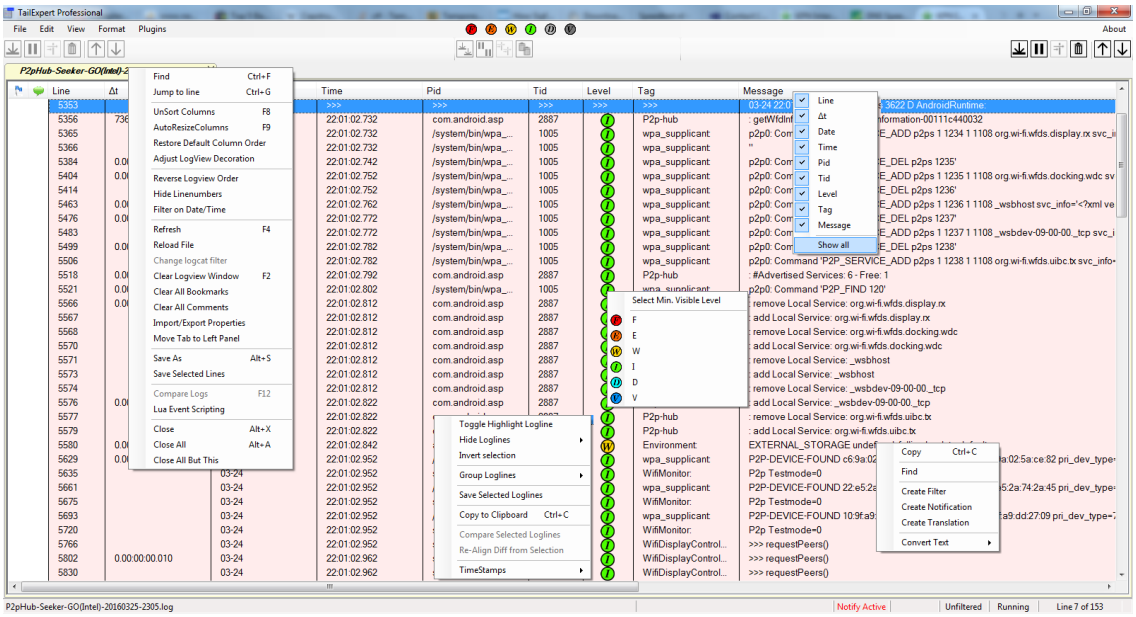
The toolbar buttons give quick access to the basic functions, stop/start scroll, pause/run, filters on/off, clear logview and jump to bookmark up/down. The left and right panel each have their own set of toolbar buttons which will be visible when the corresponding panel is also visible. When both panels are visible an extra set of toolbar buttons will become visible, with which you can perform the mentioned actions on all panels at once.

Main view pane

The main view pane can be split into two halves enabling you to view two logs side by side. Each view pane has his own set of toolbar buttons for stop scroll, pause/start tail, clear log, filter on/off, and got bookmark. The splitterbar is equipped with four buttons, two buttons to collapse left or right view pane and a button to link the scrollbars of the tow view panes and a button to sync the timestamps of the two logs. Easy start your log examination by dragging your log files into TailExpert or opening them from the menu. When you open a log file for the second time TailExpert will load all your filters, notifications, guards … so you can start watching your logs just the way you adjusted it. Save your views into a session just before you go home and next day just drag the session file into TailExpert and you can continue just where you left off. When hoovering the mouse over a logmessage and pressing <control> and right mouse button simultaneously, editmode is activated. In Edit mode the content of the column is put in selection mode which makes it possible to copy the message to clipboard or quickly define a filter for the copied text. Default the word underneath the mouse pointer is selected and the editmode context menu is shown.



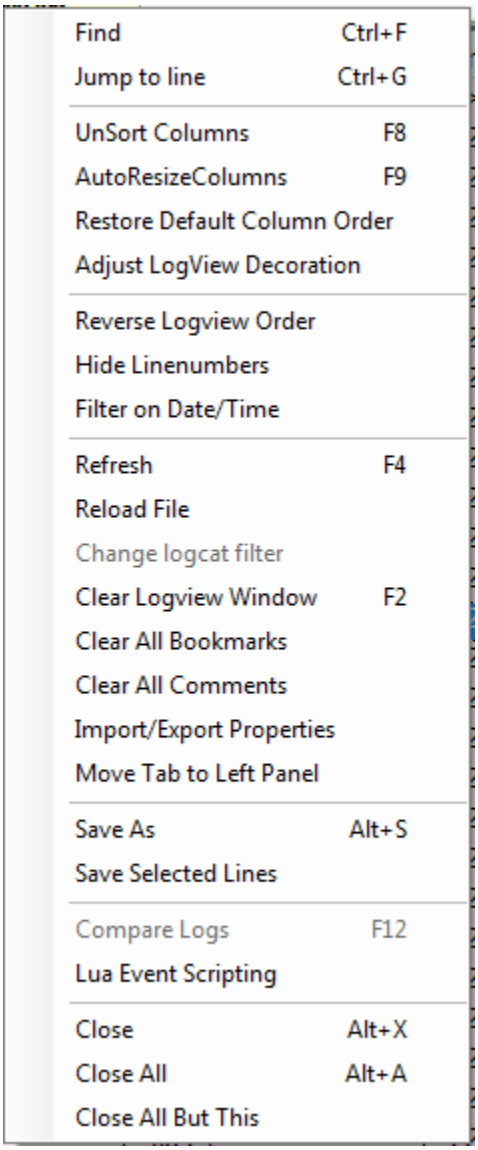
Context menu's



The logview aera of TailExpert contains five different context menus:

LogTab Context menu

The logview context menu appears when hitting the right mouse button in the logview tab. The logview context menu provides access to the following functions:



Find

Search in log messages for keywords (see appendix A for regular expression syntax)

Jump to line

Jump to a line number

Unsort columns

Double clicking a column of the Logview header will sort the messages according to the selected column. This option is provided to unsort the log messages.

AutoResize columns

Resize the column width according to the content of the cells

Restore Default Column Order

Restore to the default column order when columns have been dragged

Adjust Logview decoration

Change the color scheme and font of the logview

Reverse Logview Order

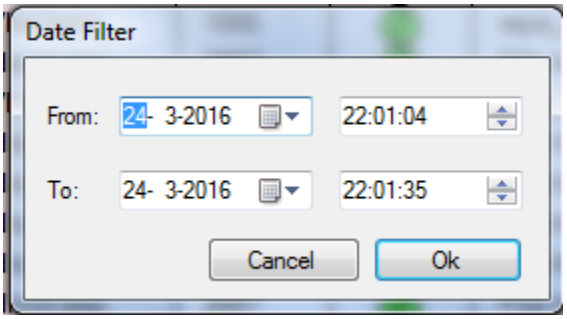
Place newest loglines on top

Show Linenumbers

Option to show/Hide linenumbers

Filter on Date/Time

Option that is available when a logformat is active that supports date/timestamp parsing and enables one to filter the logmessages based on a date/time span.



Refresh

Refresh Logview pane

Reload File

Reload the a log file from disk discarding column sorting and line highlighting

Change LogCat Filter

Restart Logcat capture using a new defined logcat filter. The loglines already captured are preserved in the logview.

Clear window

Clear logview window

Clear All bookmarks

Clear all bookmarks placed

Clear All Comments

Clear all comments placed

Import/Export Properties

Import or export the logview specific settings line filters, notifications etc. and share these over other logviews.

Move Tab to Other Panel

Move this logview to another panel so that two logviews can be viewed side by side

Save As

Save logview to a file, only the visible information will be saved to file. Only the visible columns will be saved and any filters applied will also be in effect when saving to file.

Save Selected to file

Save selected loglines to a file.

Copy to clipboard

Copy selected log messages to the clipboard for further use.

Lua Event Scripting

This option is very powerful and gives the ability to run a lua script upon each log message received. Lua is compact and highly efficient scripting language, which can be used as a sophisticated log analyzer. TailExpert provides a new logview pane to the Lua script for output, but it is also supported from lua to use .NET components to create new complex windows or dialogs like listboxes, checkboxes etc. to display analysis results, an example is provided in the lua directory under the installation directory. Lua can be set to passthrough all messages (default) however it can also be set that messages are capured by lua and then your script is responsible to forward messages to the originating logview.

Lua scripting is available to both file watch and syslog type of watch sessions.

Compare Tabs

See Chapter Compare Logs.

Close

Close active tab

Close All

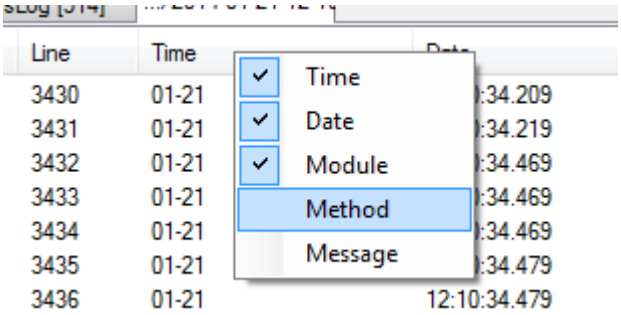
Close all tabs

Close All But This

Close all other tabs except the active one

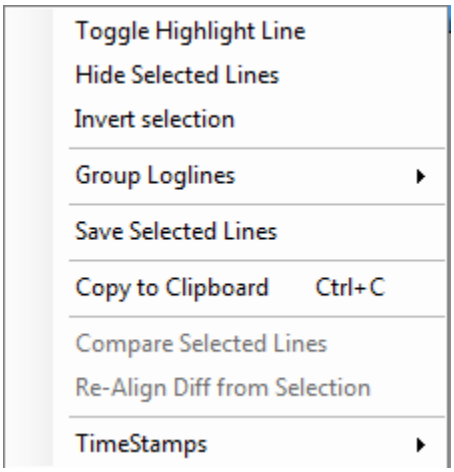
LogView header menu

The logview header menu can be accessed through a right click on the logview header and provides the ability to hide or show columns when a multicolum format is selected



LogMessage Context menu

This menu appears when hitting the right mouse button when hoovering over a logmessage. The logmessage context menu provides access to the following functions:



Toggle Highlight

Using this option the selected line(s) in the logview are highlighted

Hide Selected Lines

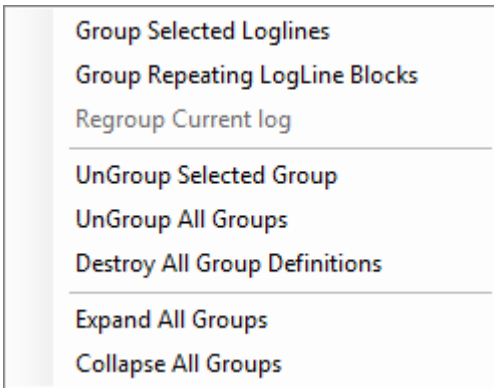
Use this option to hide lines that clutter the log

Invert Selection

Use this option to invert to selection of loglines

Group Loglines

Use this option to group loglines together in a group. The grouped loglines will be collapsed into a groupline which can be decorated with an identifier and a color setting.



Group Repeating LogLine Blocks

Use Goup Repeating Logline Blocks to group all repeating occurrences of the selected block of logmessages in the complete log.

Save Selected Lines

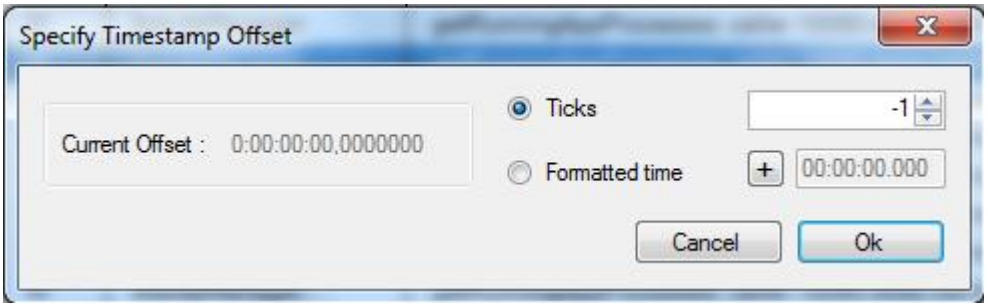
Use this option to save the selected loglines to a file

Copy to Clipboard

Use this option to copy the selected loglines to the clipboard

TimeStamps

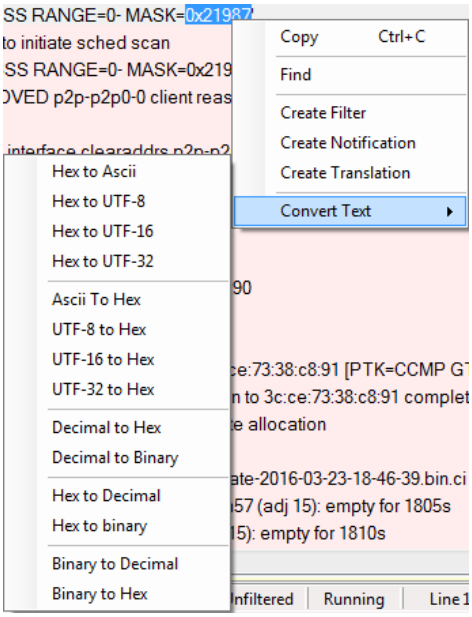
Use this option to update the timestamps of the selected logview with a time offset. This option is only enabled when the logformat selected in the view has Timestamp parsing enabled. When enabled you can manually specify an offset o apply to the logview timestamps of purpose to manually sync timestamps of two or more logs.



When a time offset is applied to a logview the tab will get the Δt identification icon.

EditMode Context Menu

The EditMode context menu can be accessed when <ctrl> right clicking in the text area and gives access to the following functions:



Copy

Copy selected text to the clipboard

Find

Copies selected text to find panel and jumps to the find panel

Create Filter

Creates a default configured filter with selected text and jumps to filter panel

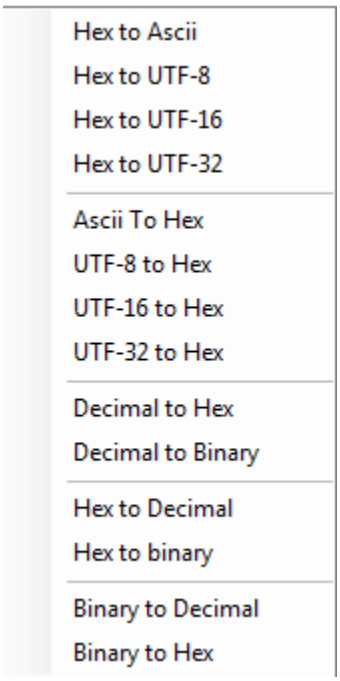
Create Notification

Creates a default configured notification with selected text and jumps to notification panel

Create Translation

Creates a default configured translation with selected text and jumps to translation panel

Convert Text



This option opens the conversion menu with the following options:

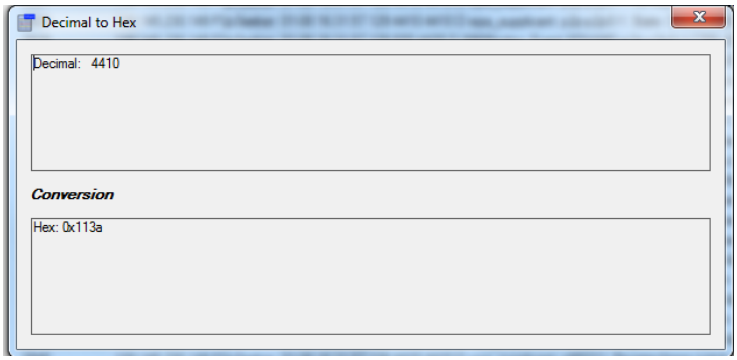
Hex to Ascii/UTF-8/UTF-16/UTF-32: Convert hex numbers to corresponding asci characters.

Ascii/UTF-8/UTF-16/UTF-32 to Hex: Convert a string to Hex numbers

Decimal to Hexadecimal/Binary: Convert a decimal number to a hexadecimal or binary representation

Hexadecimal to Decimal/Binary: Convert a decimal number to a hexadecimal or binary representation

Binary to Decimal/Hexadecimal: Convert a binary number to a decimal or hexadecimal representation



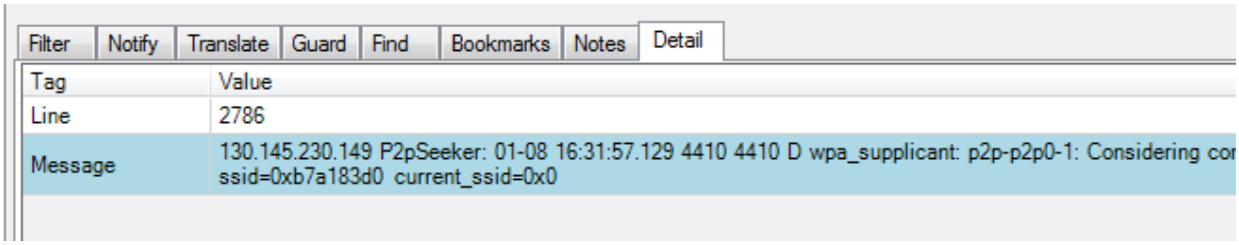
Logview window functions

Rename tab title

Double clicking the tab enables the tab title edit mode where the tab can be renamed to a more readable name.

Logmessage tooltip

When a logmessage does not fit the column, hoovering the logmessage wil show a tooltip text with the complete message. When you want to inspect the logmessage even in more detail, you can use the details panel. The details panel shows the complete message where you have the ability to copy parts of the logmessage to clipboard.



Doubleclick logmessage

When doubleclicking a logmessage the logmessage will be marked highlighted using the highlight colors specified in the preference menu. When bookmark on highlight is checked the logline will also be bookmarked and added to the bookmark collection for quick lookup.

Bookmark column

Bookmarks can also be placed by clicking in the bookmark column.

Comments column

When clicking in the comments column a dialog will popup that enables you to add a comment to the corresponding logmessage. This comment is saved along the logmessage when the log is saved to file and the comment is added when the log is printed on paper.

Multiview window

When more than one log is opened, one has the option to put two opened logs side by side. When dragging a tab a second panel will become visible and one can drop the tab into the opened panel. When both panels are opened extra options become available.

Splitterpanel buttons

These buttons can be used to collapse/restore the left or right view pane.

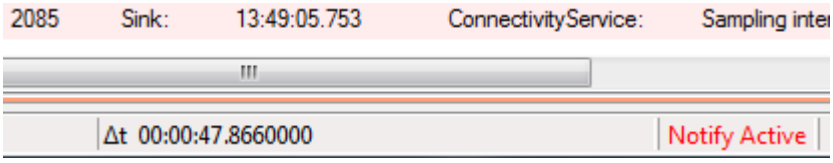
Link the scrollbars of left and right view panel

Sync the timestamps of left and right view panels (only available when the logformat has timestamp parsing enabled)

Sync two logviews based on their timestamps, this option helps you to reveil the interaction of two logs.

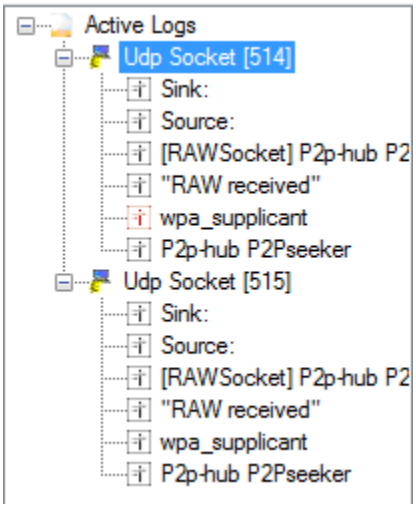
Timestamp delta calculations

When timestamp parsing is enabled on both view panes TailExpert is able to show the time delta between the selected row in the left and right pane. After selecting the second row the delta time is displayed in the status bar for 8 seconds before it is dismissed.



TreeView Quick Filters

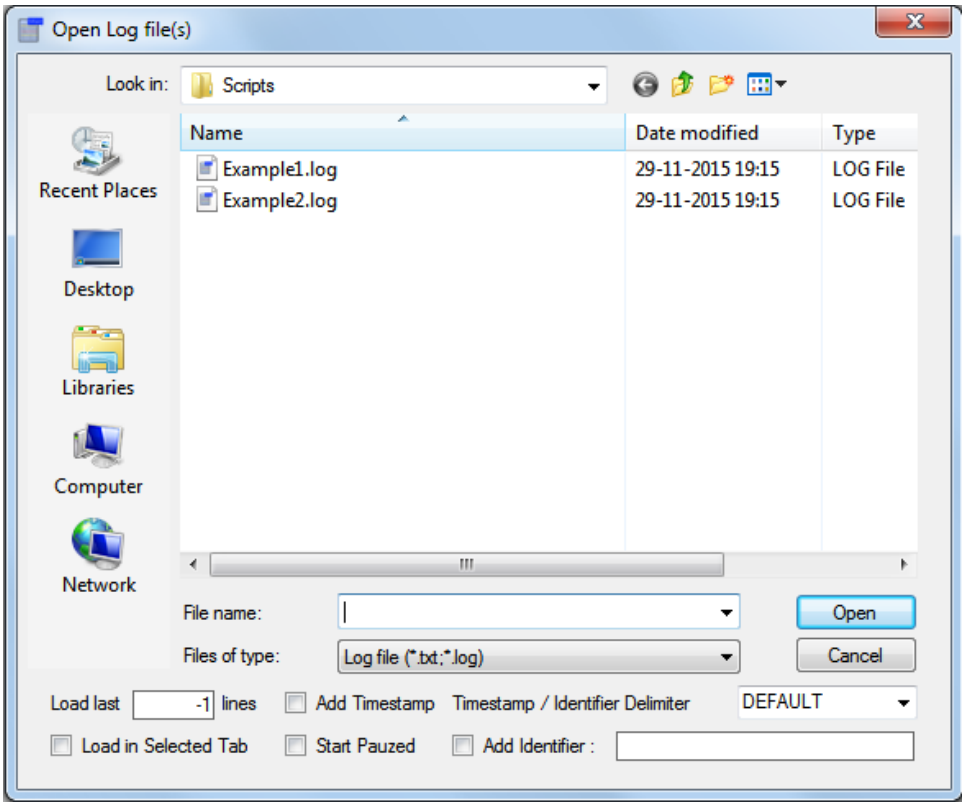
When you open a log in TailExpert, it will be added to the active logs in the treeview pane. When you already have defined filters for this log, these filters will be added to the logname node for quick access. By clicking on the filter you can activate (or toggle activation depending on settings) the filter. To remove the filter(s) you just click on the logname node and TailExpert will remove all filters and reload the log.



File menu

Open File

Open a file or multiple files you wish to follow in the viewer. A new tab will be created which will show incoming log lines. The file will be watched for changes, lines will be appended to the bottom of the logview pane. When the file has been loaded before and filters, notifications and/or a logformat has been defined for this file, these filters, notifications and the logformat will be restored. When e new file is opened default the plain text logformat is selected.

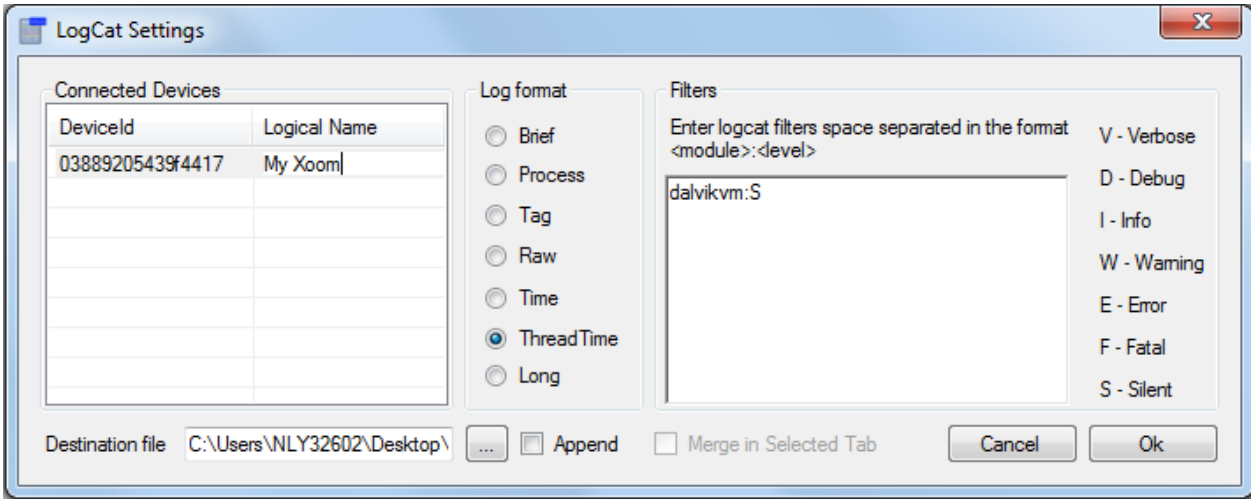


In the openfile dialog extra options are visible:

- Load last x lines: With this option you can specify the number of lines history to load, -1 means that the complete file will be loaded.
- Add Timestamp: add a timestamp in front of each logmessage loaded from the file
- Timestamp delimiter: character to add behind timestamp to ease logformat columnization
- Start paused: when checked the tail engine is paused so new messages will not be read (yet)
- Load in Selected tab: Enables you to merge two or more logs of any origin into one view.
- Add Identifier: Add a column to your log with the specified identifier for logmessage identification

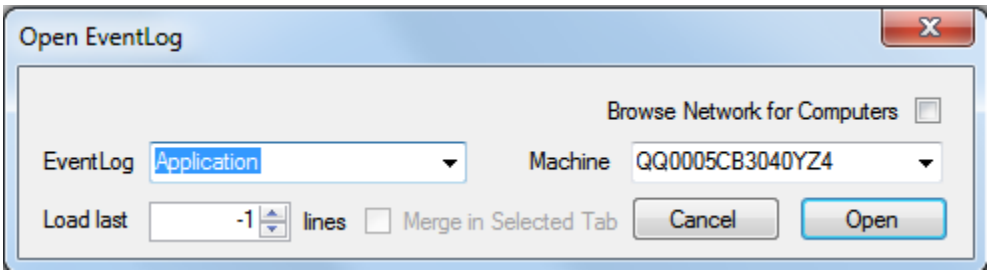
Open Adb logcat

Open logcat log through the builtin adb interface. Newly connected devices will get listed with their corresponding serial number, one can give the connected device a human readable name by editing the Logical Name column. The filters set here will be active during the complete session and cannot be changed during a session, use this filter only when you want to pre-filter the log to avoid extreme large logs. For analysis purposes it is better to use the filters mechanism available in TailExpert.



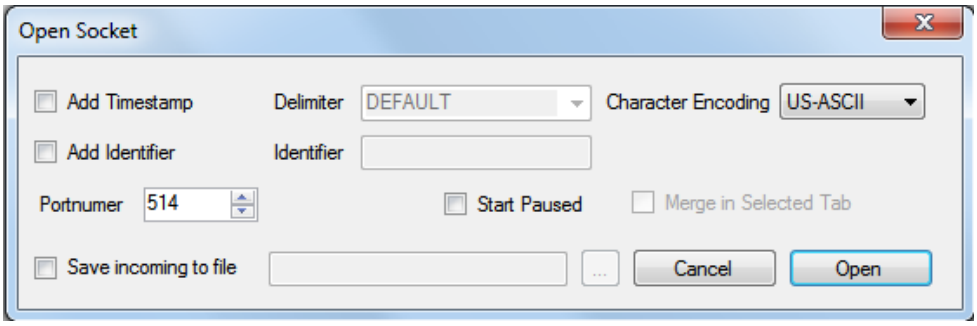
Open Eventlog

Open and watch an eventlog on your local system or on a remote system. When there's already a log opened in the viewer the option Merge in Selected Tab is enabled, which will redirect the incoming logmessages to the selected logview. The number of lines to load from history is default set to -1 what means that all available logentries will be loaded, set this number to a number of choice to limit the history retrieved.



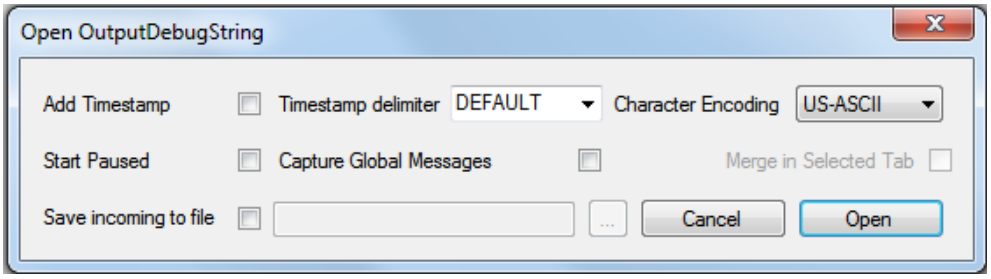
Open UDP Socket

Open a UDP socket to receive log messages, specify port to use, character encoding, optional add a timestamp to incoming messages. Default port is set to 514 the Syslog (UNIX) port. Depending on the source of the log, the logmessages will contain a timestamp. If the source of the log does not contain timestamps, TailExpert can add a timestamp. Check the “Save incoming to file” to save the retrieved logmessage to a file.



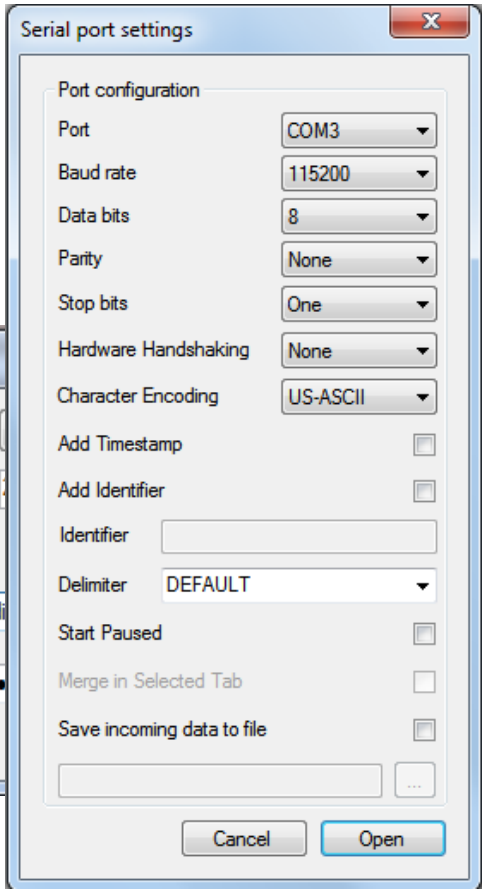
Open OutputDebugString

OutputDebugString is a somewhat outdated windows logsystem, but still used by some programs. Open an OutputDebugString receiver to catch windows output debug string messages. Here you also have the option to add a timestamp to the logmessages and to save the incoming logmessages to a file.



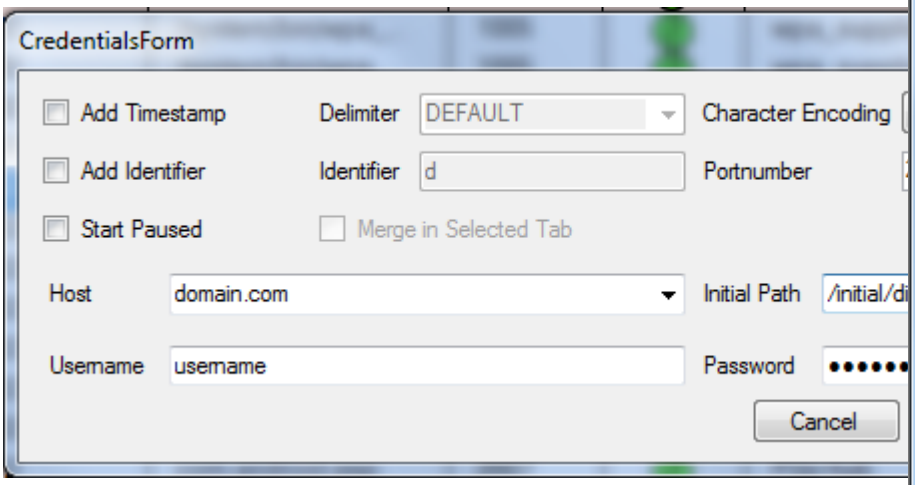
Open Serialport

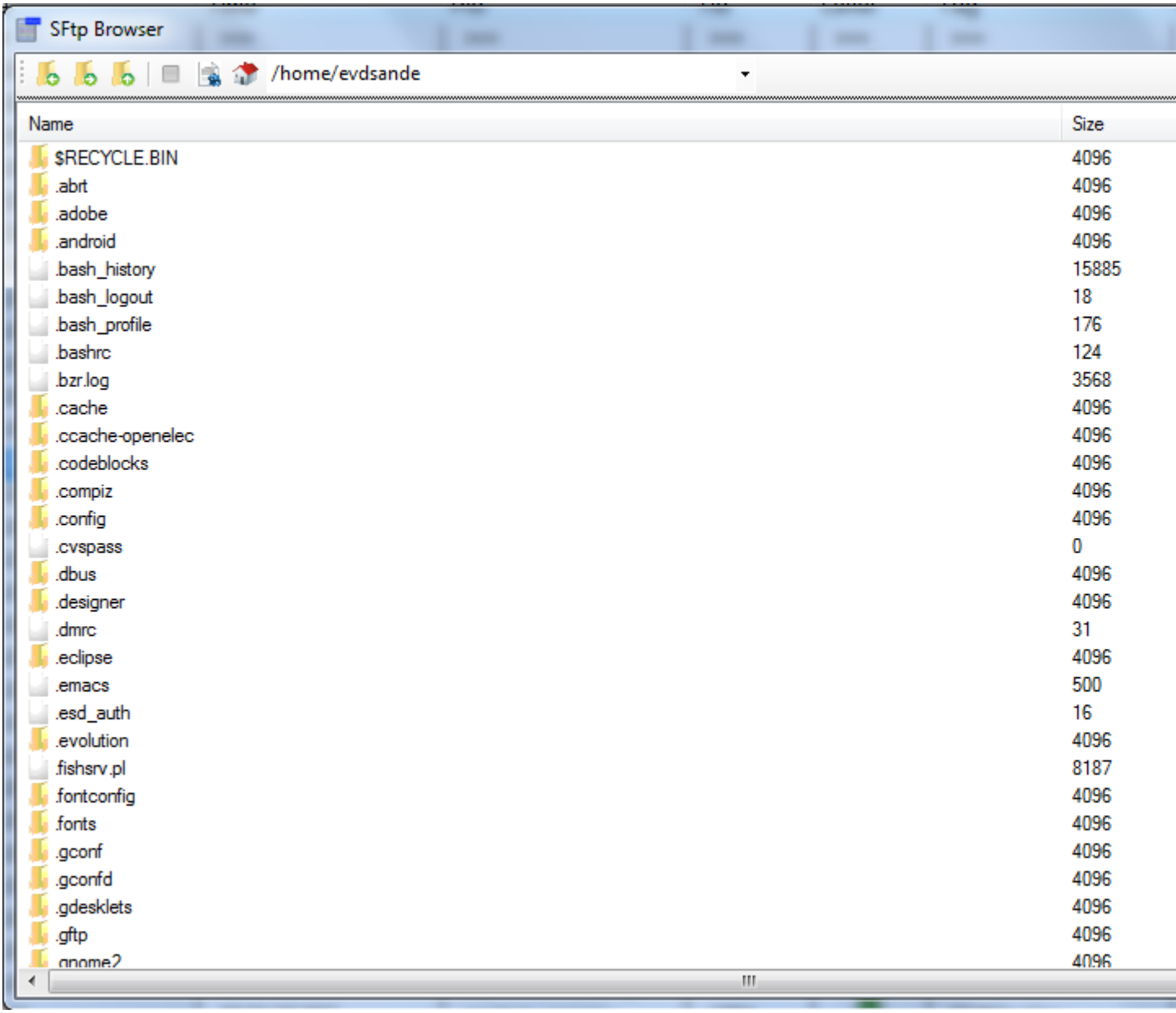
Open a serial port to receive log messages, specify port and port settings and character encoding, optional add a timestamp to incoming messages or add an identifier to your logmessages. There's also an option to save the incoming logmessages to a file.



Open Sftp logfile

Open a logfile through the built-in sftpbrowser.




SFtp Browser
/home/evdsande
Name
Size
$RECYCLE.BIN 4096
.abrt 4096
.adobe 4096
.android 4096
.bash_history 15885
.bash_logout 18
.bash_profile 176
.bashrc 124
.bzr.log 3568
.cache 4096
.ccache-openelec 4096
.codeblocks 4096
.compiz 4096
.config 4096
.cvspass 0
.dbus 4096
.designer 4096
.dmrc 31
.eclipse 4096
.emacs 500
.esd_auth 16
.evolution 4096
.fishsrv.pl 8187
.fontconfig 4096
.fonts 4096
.gconf 4096
.gconfd 4096
.gdesklets 4096
.gftp 4096
.gnome2 4096

Open From Clipboard

Open a new tab using the clipboard content.

Run Luascript

Run a luascript which can open a new tab and perform the log gathering.

Save as

Save displayed log messages to a file. If the logview has an active filter on it, note that filtered (hidden) messages will not be saved, the same accounts for hidden columns. Note that defined filters and notifications as well as the logformat is not saved alongside with the saved logmessages. These preferences need to be set again when the logmessages are loaded in TailExpert. TailExpert however provides the option to copy filters, notifications and logformat from other tabs.

Save Selected Lines

Save selected logmessages to a file, if the logview has an active filter on it, note that filtered (hidden) messages will not be saved, the same accounts for hidden columns.

Close

Close the active tab.

Recent Files

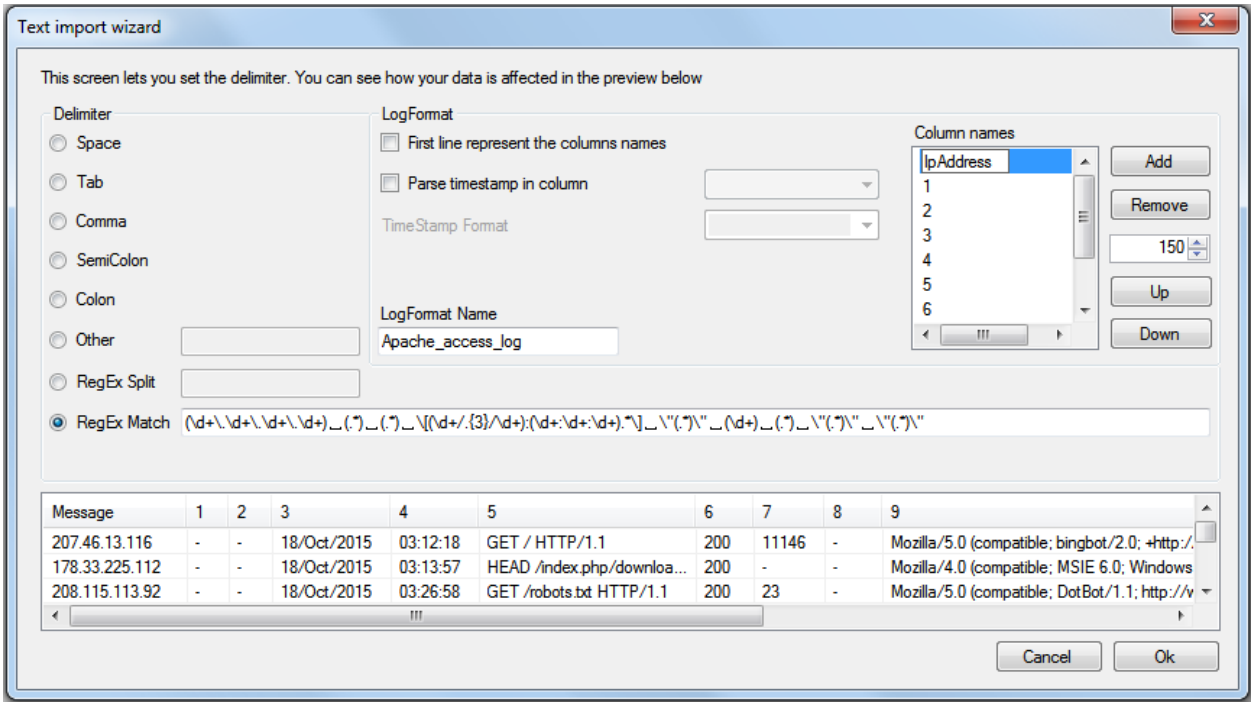
Shows files used recently, new files are automatically added to the list and the oldest one is removed.

Clear Recent Files

Clear the list of recent used files.

Logformat Import Wizard

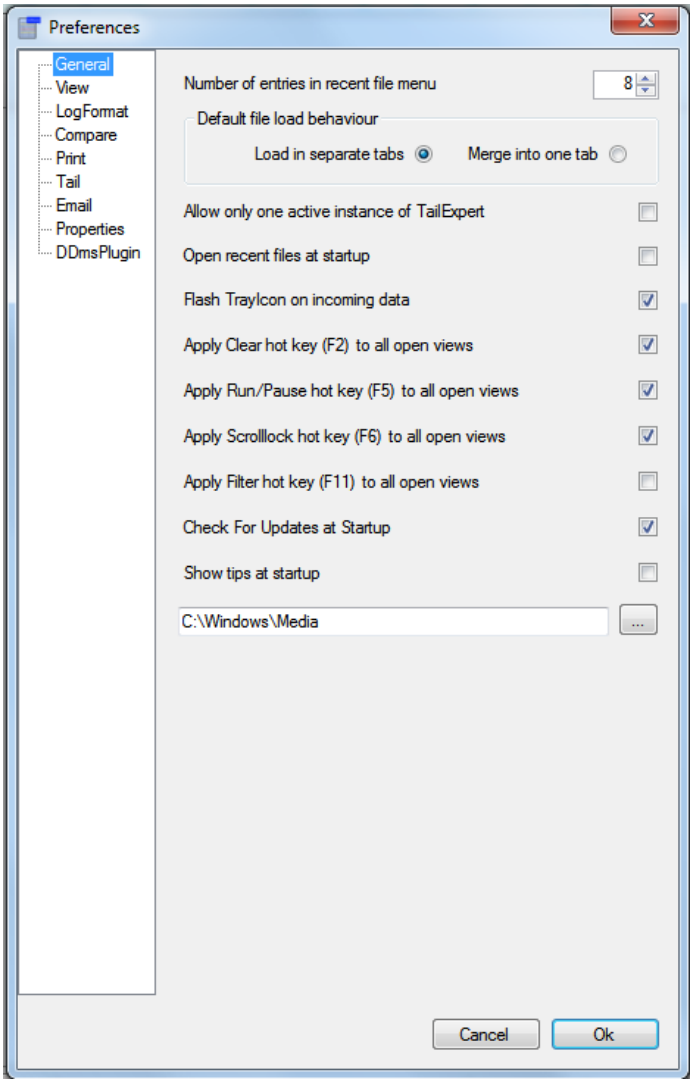
Wizard that help to set the logformat for a logfile. Select the delimiter to use to separate subjects in the logmessage. The table at the bottom of the dialog will show the formatted logmessages as an example. Next to a number of preconfigured delimiters the wizard supports also supports regular expressions. Regula expressions can be used as a split expressions or as a match expression. In the figure below a sample is shown of a match expression to format the apache access_log format. When the first row of the logfile contains the columnnames activate the checkbox to collect the column names otherwise add the required columnnames in the box to the right. If your logs contain timestamp data you can decide to let TailExpert interpret the timestamps (this function has a speed penalty) which will enable TailExpert to automatically sync the timestamps of two logviews when messages arrive.



Preferences

The preferences dialog provides access to configuration parameters for the TailExpert program.

General settings



Default load behavior: Here you can specify if the default behavior should be that every load action will load the log into the opened logview, or that each load results in the creation of a new tab.

Number of recent files menu entries: specify the number of recent files entries you like to have in the file menu.

Check for updates at startup : specify whether you want TailExpert to look for updates when it is started (requires an internet connection).

Allow only one instance: When checked starting a new instance of TailExpert will direct you to the existing instance

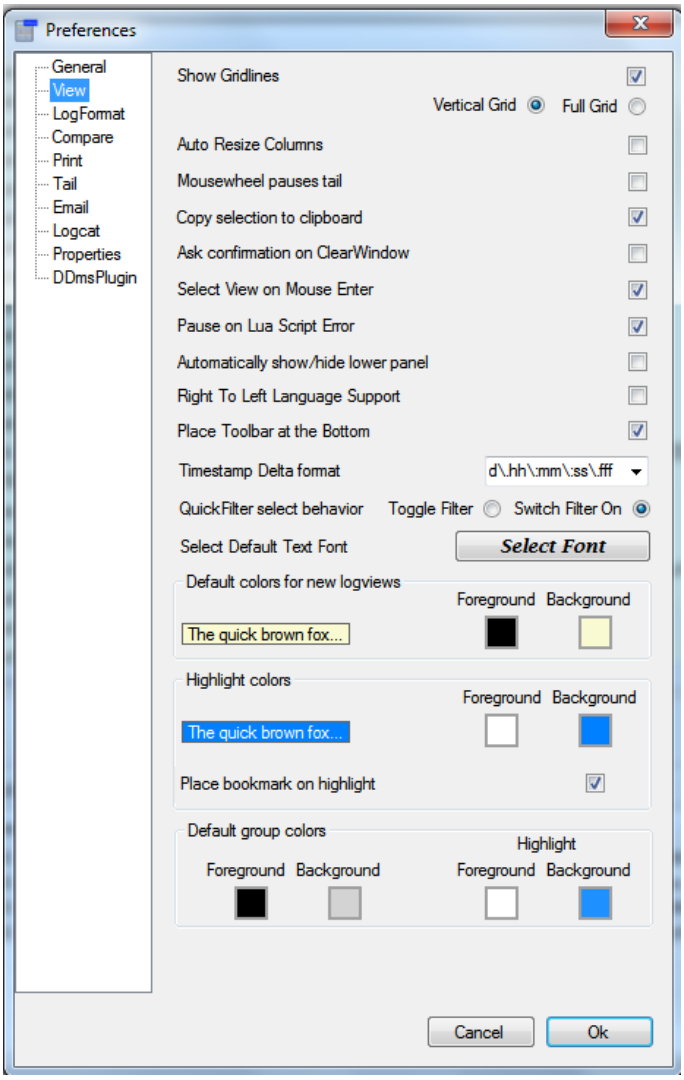
Open Recent Files at startup: When checked all files in the recent file list get loaded at startup

Apply Clear hot key (F2) to all open views: When checked, the F2 Clear window function will apply to all open views

Apply Run/Pause hot key (F5) to all open views: When checked, the F5 Run/Pause window function will apply to all open views

Apply Scrollock hot key (F6) to all open views: When checked, the F6 Scrollock window function will apply to all open views

Apply Filter On/Off hot key (F11) to all open views: When checked, the F11 Filter On/Off window function will apply to all open views



View settings

Default file load behavior: Select here if you want Tailexpert to merge logs into one logview.

Auto Resize Columns: In this pane you can specify whether the logview should automatically resize the columns whenever a logmessage is added to the logview.

Mousewheel pauzes tail: When this option is checked, using the mouse scroll will put the tail in pause so you can view the logmessages without the logview to scroll to end when a new message arrives.

Copy selection to clipboard: When this option is checked, every selection is copied to clipboard automatically.

Ask confirmation on ClearWindow: When the option is checked, when you select clear window a confirmation is asked

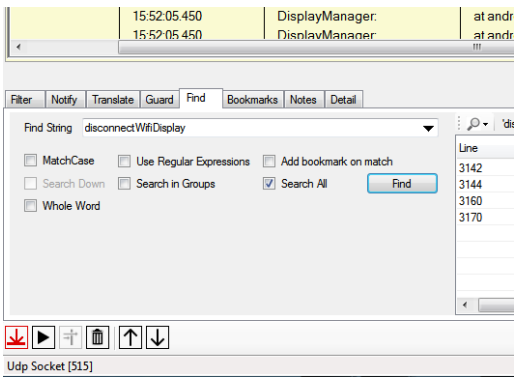
Select View on Mouse enter: When this option is checked and you have two logs in a side by side view, the view is selected when the mouse enters the view area

Pause on Lua Script Error: When this option is selected, the tail engine is paused when a lua error occurs, to easy script debugging

Automatically show/hide lower panel: When this option is selected, moving the mouse near to be bottom of the form shows the lower panel, moving the mouse out of the panel, hides the panel.

Right to Left language support: When this item is checked the logmessages are shown from right to left.

Place Toolbar at the bottom: Locate the toolbar at the bottom of the window.



Quick Filter select behavior: Select Toggle filter of Switch filter on according to your wish

Select Default Text font: specify default text font to use in the logview, applied to new logviews, note that each logview can override these settings by using the Adjust Logview decoration option in the tab context menu

Default Colors: specify default text colors to use in the logview, applied to new logviews, note that each logview can override these settings by using the Adjust LogView Decoration option in the tab context menu

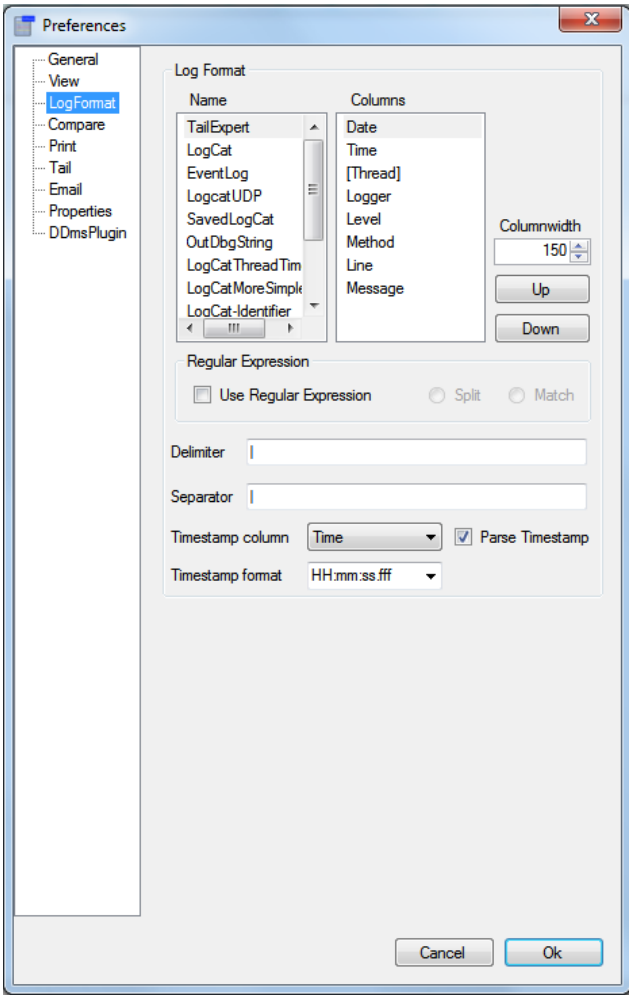
Highlight Color: specify the colors used when highlighting a specific logline by double clicking a logline

Highlight Color: specify foreground and background color to use with the highlight current line function

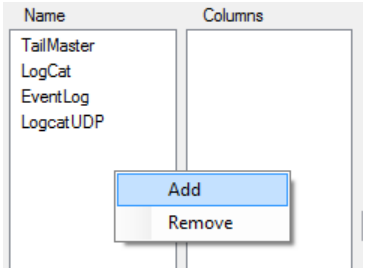
Place bookmark on highlight: When this option is checked, when double clicking a log line a bookmark is also added

LogFormat

In this preference screen logformats can be defined or altered.

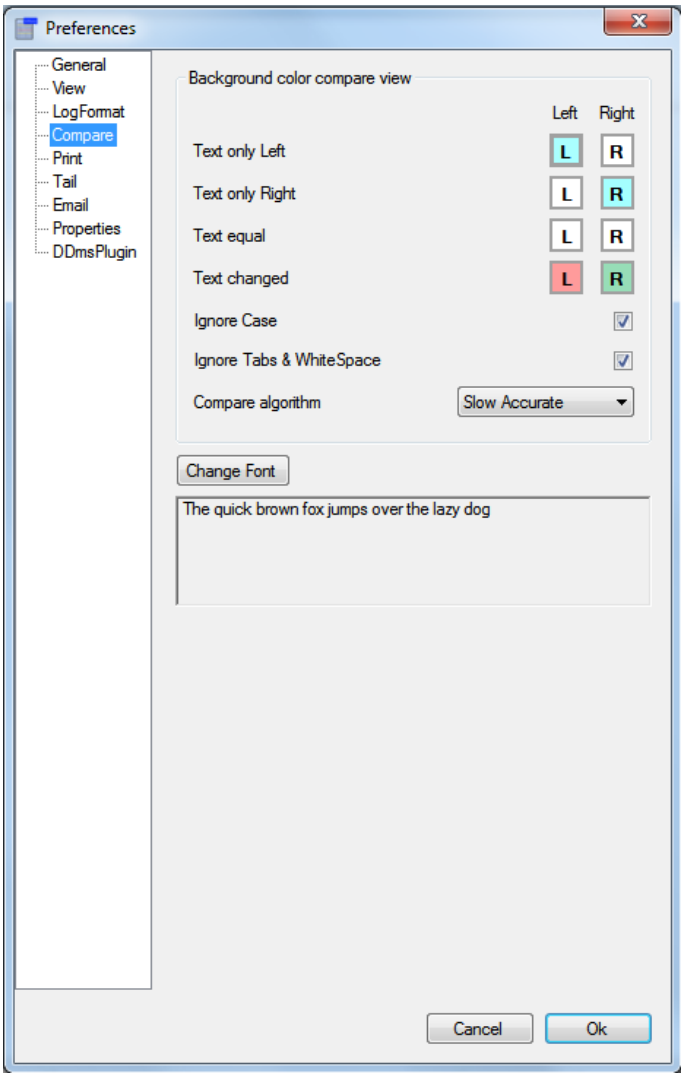


To add a new logformat right click on the corresponding listview and click add, then enter the new name and add your columns in the column listview. Specify the delimiter which is used to separate words in a logmessage and the separator which separates column text when log is exported or printed. For each column the default column width can be adjusted and columns can be manually sorted.



Compare

This preference screen show the options for the compare view. Display colors and font can be adjusted and whether the compare tool should ignore Case and/or Tabs and Whitespaces.



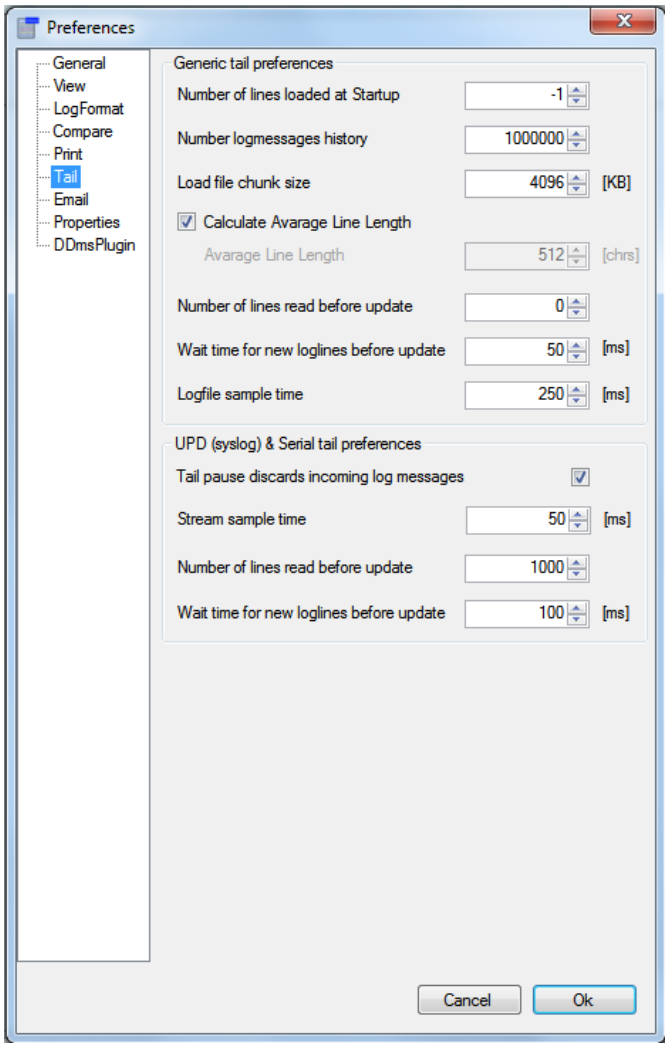
Print

Select printer font

Tail

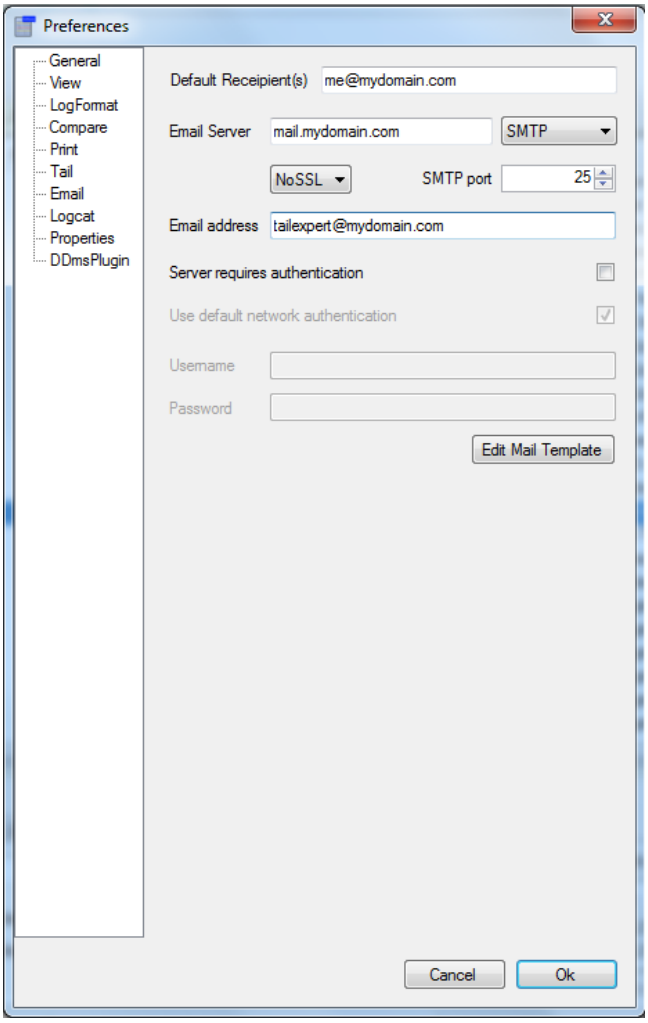
For the tail background process the following configuration parameters are available:

- **Number of lines loaded at startup**: Applicable for Open File, this specifies how much history is loaded from the file on which TailExpert will be active.
- **Number of Scroll Lines**: Maximum number of lines to show in the logview pane, specifying 0 here will result in no limit.
- **Load file chunk size**: Size of internal buffer which is used to load a logfile. Can be used to optimize file load time.
- **Calculate Average Line Length**: Use an algorithm to calculate the average number of characters per line to optimize the load of logfiles.
- **Number of Lines read before update**: Number of lines read from the logfile or socket interface before the logview pane is updated. This option make the screen update smoother, and lowers system load.
- **Wait time for new loglines before update**: Timeout to wait on new input logmessages before pushing messages to the logview pane.
- **Logfile sample time**: Sample time in which the file is checked on changes



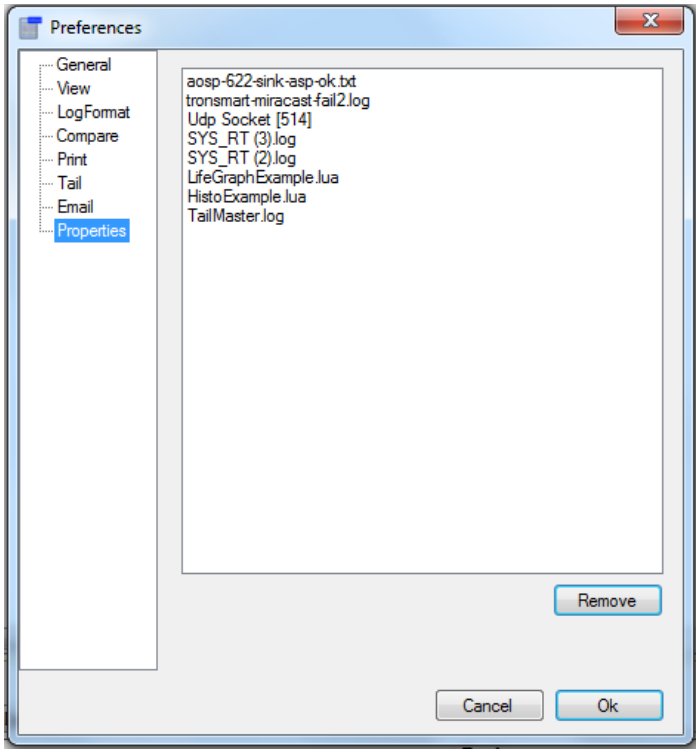
Email

In the email pane you can specify the email service you want to use with TailExpert. TailExpert is capable of using 2 different kinds of email services: Outlook and SMTP. To be able to use outlook, outlook has to be installed on the system.



Properties

In the properties preference pane you can clean up your saved session data. Select the entries you no longer need and hit the remove button.



Ddms plugin

When the Ddms plugin is selected during installation, the Ddms plugin preference menu will become accessible. Here you need to specify the android SDK location from where Ddms will be executed.

Print

Print the log to a printer, here also only the visible logmessages are printed as well as only the visible columns.

Print Selected

Print the selected loglines to a printer, here also only the visible logmessages are printed as well as only the visible columns.

Preview

Preview the printer output.

Settings

Printer settings.

Edit menu

Find

Find text in log messages, find function, normally, will start at the bottom and search up in history. Use match case of regular expressions to narrow the search or select search down to search from top to bottom. Note that the find function will search for text only in the visible loglines but will take also hidden columns into account.

Jump to line

Jump to line number provided

Unsort columns

Undo selected sort of columns, log messages will be shown in the order as they where received or read from file.

AutoResizeColumns

Resize columns to fit contents, hidden columns will be made visible

Toggle Highlight Line

Mark a logmessage with a color for better log understanding

Hide Linenumbers

Hide the column with linenumbers

Hide selected loglines

Hide logmessages that clutter your view for better understanding of the log. When the view is refreshed hidden loglines will re-appear.

Refresh

Refresh logview pane, log messages will be reread from the internal cache.

Clear window

Clear all messages in the logview pane, internal cache is not cleared. To reread the internal cache and re-show the logmessages use refresh.

Clear bookmarks

Clear all bookmarks placed

Copy to clipboard

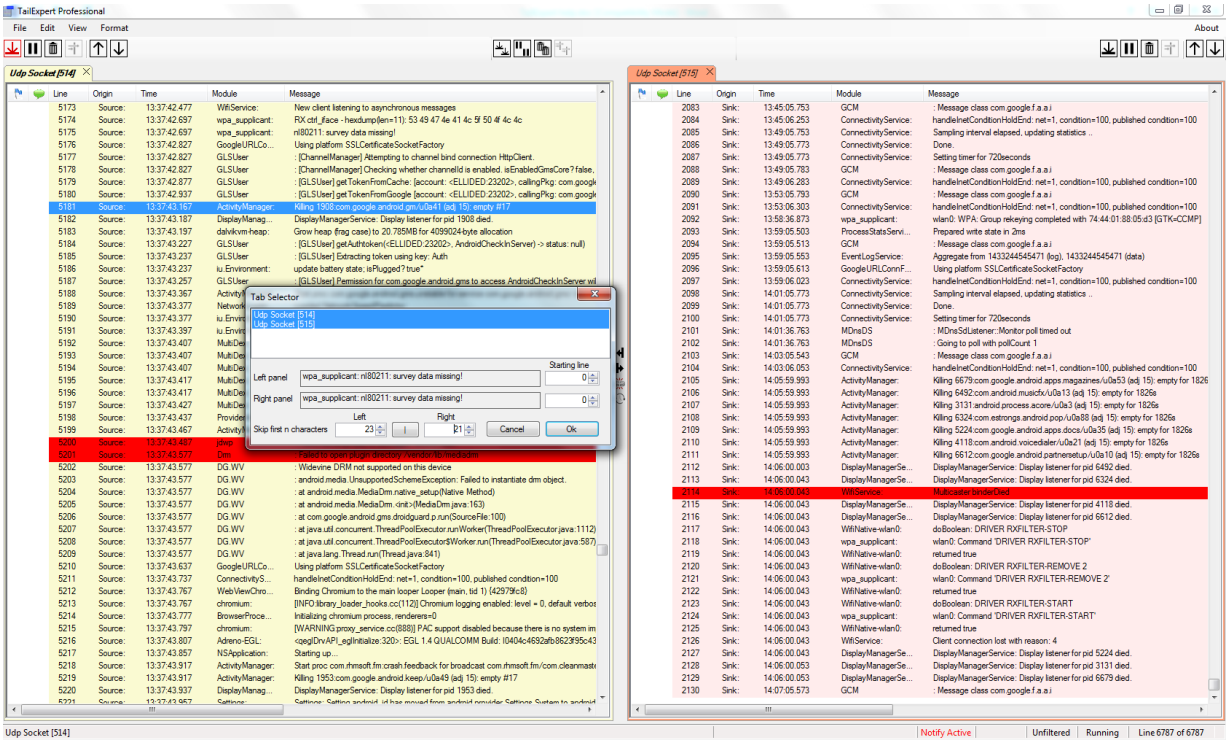
Copy selected logmessages to the clipboard

Lua Event scripting

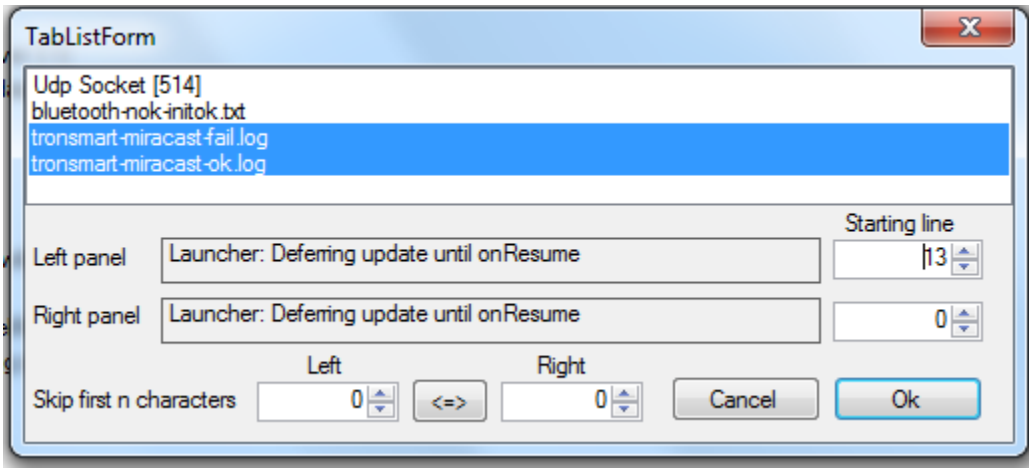
With this option you can run a lua script on every logmessage that is captured. This enables you to create even more sophisticated filters or you can even create an analysis script that does automatic log analysis. Combining this with the ability to use notifications, you can deploy TailExpert even in an automated test environment.

Compare Logs

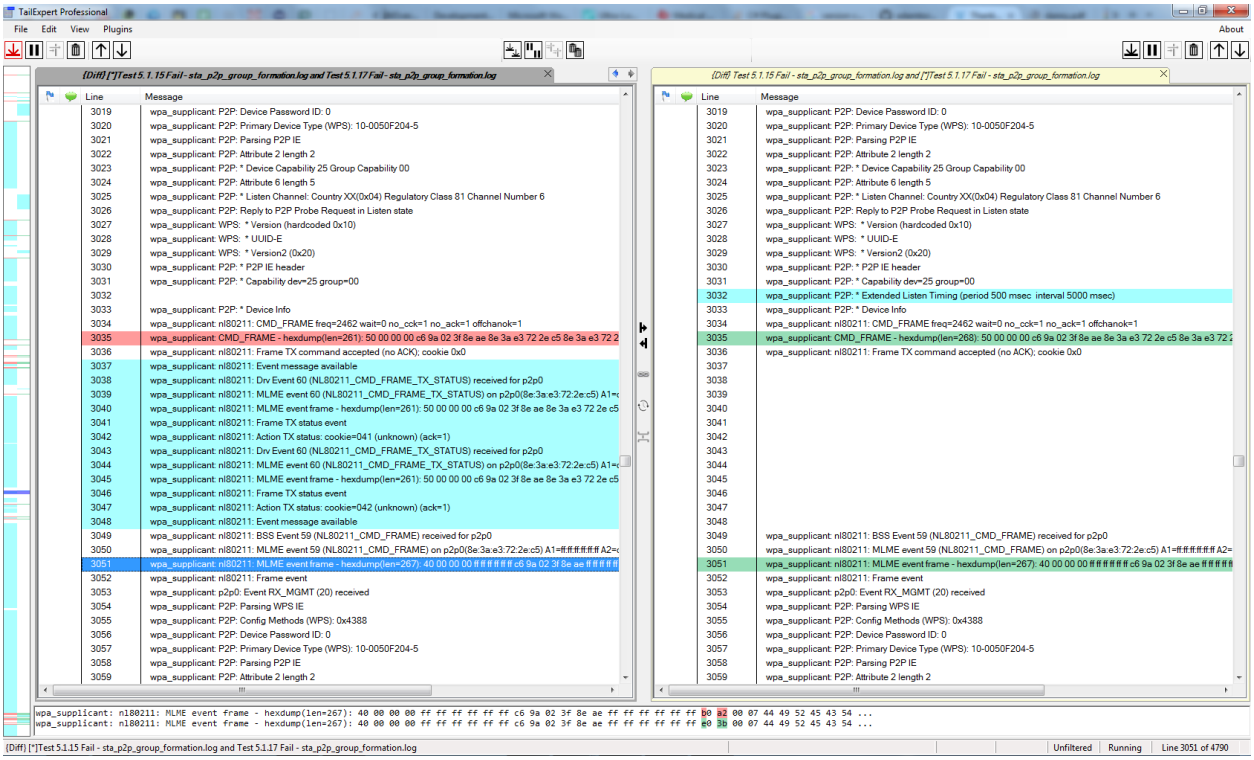
This option makes it possible to compare two logviews, a new logview window is opened with both logs side by side indicating the differences.



The following dialog lets you select two logs from the logs opened in the logviewer to compare and provides you the ability to remove a number of characters from beginning of the loglines to avoid mismatches on date/time entries. Use the starting line to search for the best line to start the comparison.



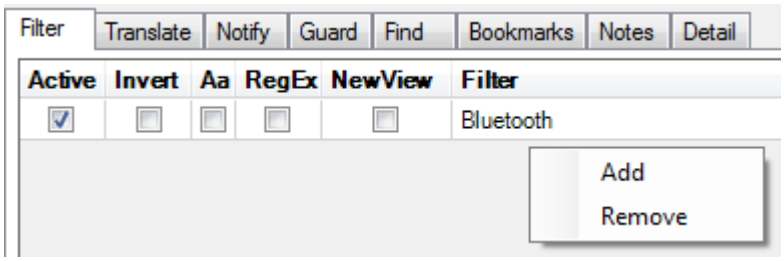
Example Compare View



View menu

Filter

This option will show the filter tab, here you can specify filters. Filters can be words separated by spaces or sentences which need to be quoted “”, the filter will match when any of the words/sentences occur in the logmessage. To match all words/sentences the words/sentences need to be proceeded by a + sign, this “AND” function can also be accomplished by defining and activating multiple filters. A ^ sign in front of the word/sentence will negate the match. Use right mouse button in the panel to show context menu where you can add or remove filters. The filter will only be activated when the checkbox in the first column is checked. The filter will be applied to the logview, only when you hit the Apply button on the right. (see appendix A for regular expression syntax).



Examples:

error warning will match all logmessages that contain the words *error* **OR** *warning*

error +warning	will match logmessages that contain both words *error* **AND** *warning*
^info	will match all logmessages that do not contain the word ***info***
^info +^debug	*will match* all logmessages that do not contain the word ***info*** nor the word ***debug***
"This is a text"	will match all logmessages that contain the sentence ***"This is a text"***

When the checkbox RegEx is selected a regular expression is expected.

Working with numbers

TailExpert can also filter on numbers found in the logmessage by means of the ==, !=, <, >, ≥, ≤, <>, ><, ≥≤, ≤≥ operators. When the number to compare is placed inside a lengthy logmessage, which may even contain other numbers, we need first to locate the number of interest. This can be done by specifying first the substring preceding the number as is shown in the example below.

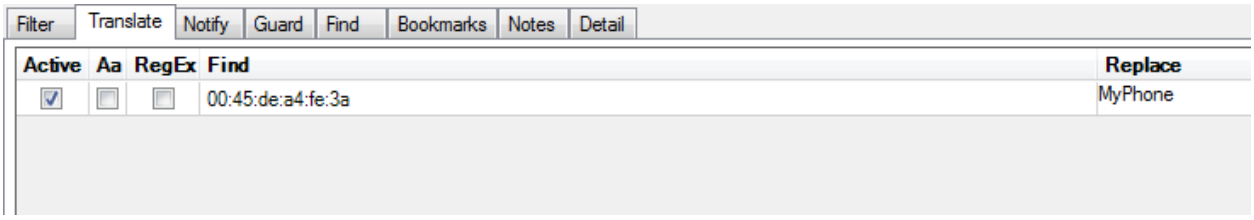
```
06-16 13:17:57.287 798 798 W PackageParser: No actions in intent filter at
/system/priv-app/Contacts.apk Binary XML file line #375
```

Let's say we want to capture all log messages that contain the text *"Binary XML file line #"* and a line number between 300 and 400. We can then specify the filter using the <> operation and we enter the string ***"Binary XML file line #" 300-400*** into the filter field.

Note that combining multiple filters statements in one filter cannot be applied when using the numerical compare operators.

Translate message

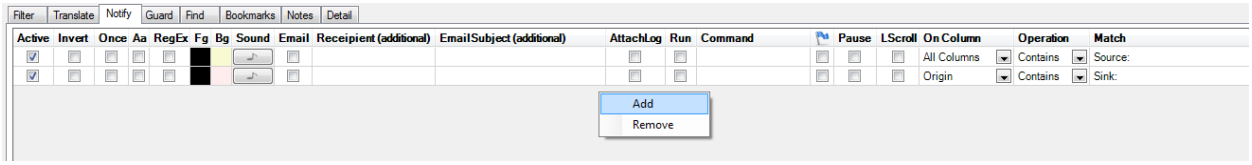
This option will show the Translate tab, here you can specify translation filters. Translation filters enable you to replace words of parts of the logmessage with a specified word or line. This can be helpful if the logmessages contain a large amount of information that can clutter you view. Please read the chapter on filters for the filter format. (see appendix A for regular expression syntax)



Notifications

This option will show the notifications tab, here you can specify notification filters. Notification filters can be applied to the whole logmessage or when using a logformat on the defined columns. When a specific column is selected arithmetic comparisons are possible however when a arithmetic operation is selected the filter may only contain one condition per row. For a non-arithmetic operation (contains) the

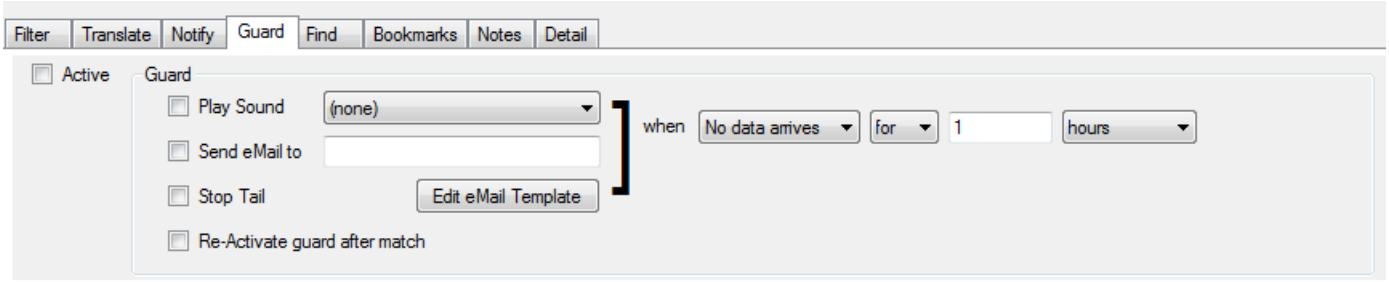
syntax defined in the previous chapter applies. If a logmessage has a match, the notification filter will be activated and the action executed. A notification can be highlighting of the matching line, playing a sound or send an email or all options at once. Use right mouse button to show context menu to add or remove filters. The notification will only be activated when the checkbox in the first column is checked. The notification will be applied to the logview, only when you hit the Apply button on the right. Please read the chapter on filters for the find text format. (see appendix A for regular expression syntax)



Enter text to match in the Find editbox and select a background and foreground color by clicking in the color box.

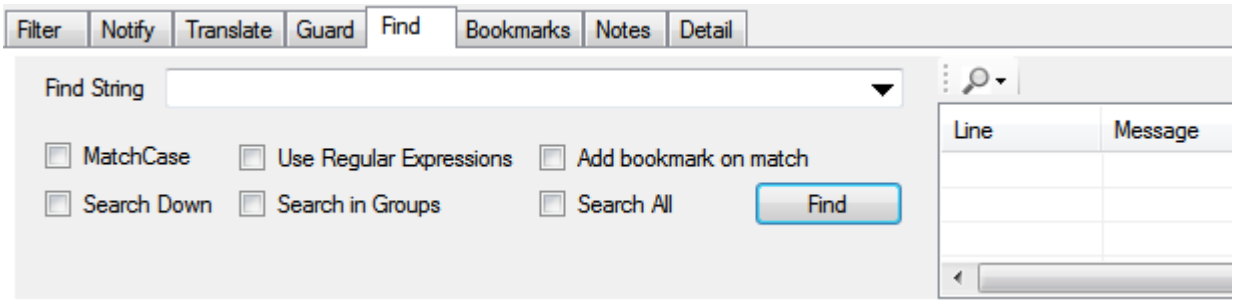
Guard

This option shows the Guard tab, here you can specify a guard that is fired when the specified condition is matched. For example the no logmessages are received within a certain timeframe a sound can be played or an email can be sent to a specified recipient.



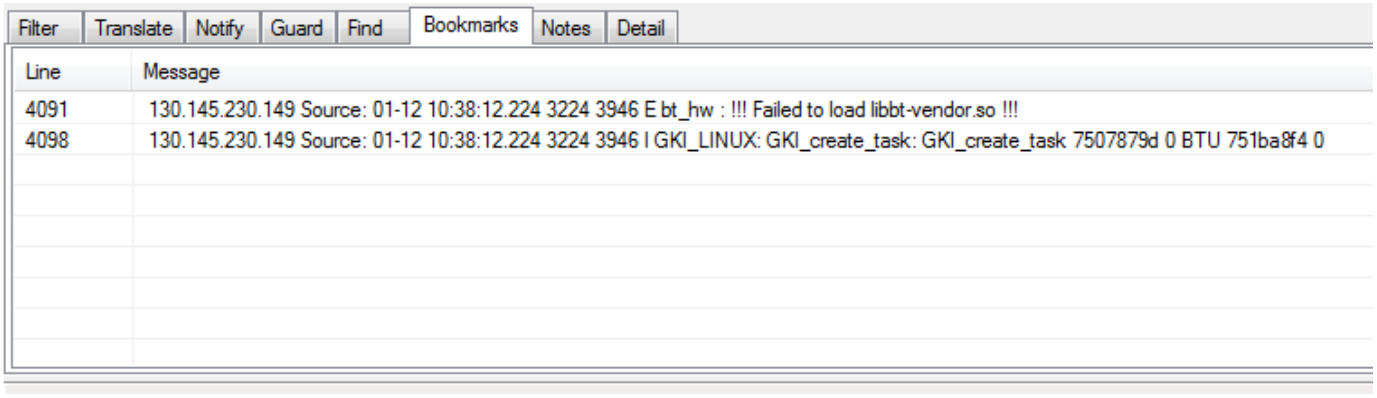
Find

Search occurrences of text in logmessages, the default behavior is that search is started from bottom up (newest lines first). (see appendix A for regular expression syntax)



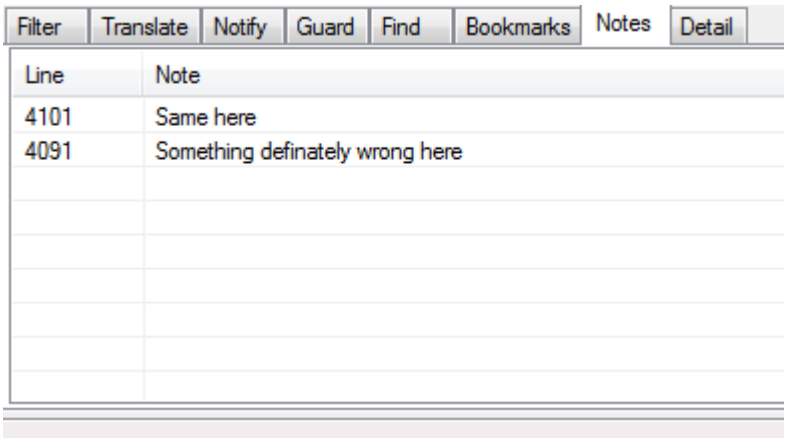
Bookmarks

This view shows all bookmarks placed in the logview. Double clicking on a bookmark takes you to the corresponding message.



Notes

This view shows all bookmarks placed in the logview. Double clicking on a bookmark takes you to the corresponding message.



Setup Lua scripting

TailExpert is able to run a specific Lua script on each log message received. This enables complex log analysis. Example lua scripts are provided in the lua directory of the installation path, as a quick start. Or can be found at the TailExpert support forum.

Hide/ Show Panel

Hide/Show Filter/Highlight/Translate/Format panel

Hide/Show filetree

Hide/Show the filetree panel

Format menu

Plaintext

Display test as one line without columns

Errorlog

Use the format specified by ‘Errorlog’ to separate log messages into columns

Dbglog

Use the format specified by ‘Dbglog’ to separate log messages into columns

Logcat

Use the format specified by ‘Logcat’ to separate log messages into columns

…

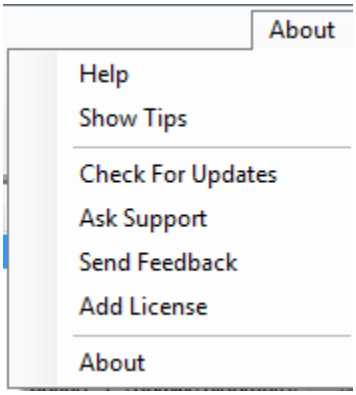
New log formats will be added to this menu dynamically.

Plugins menu

When a plugin is selected during installation the plugins menu becomes visible. Here you can select the plugin you want to activate.

About menu

The about menu gives access to the following functions:



Help: Shows the bundeled documentation of TailExpert

Show Tips: Shows a tips dialog which give an overview of the more advanced features of TailExpert

Check for updates: This will activate a check for available updates. When a new update is detected the program will prompt you to ask whether you want to install this new version.

Ask Support: This option will direct you to the TailExpert support site.

Send feedback: This option will prompt you with an form on which you can rate the program and write comments, comments on the program are very well appreciated and help us to make TailExpert even better.

Add License: Here you can add you TailExpert license you obtained from Neware Enterprise Solutions and unleash the real power of TailExpert.

About: Shows some information of the program such as the version number.

Lua Scripting

Using Lua scripts within TailExpert can be very powerfull. The Lua scripting engine in TailExpert not only provides you the standard lua library functionality but the Lua scripting engine has also access to the full .NET runtime environment by means of the integration of luainterface. Besides that LogTail also adds two graphing libraries which one can use to paint graphs based upon matches detected in the logmessages. Additionaly TailEpert also provides a number of functions to the scripting engine to automate some tasks line setting filters, notifications etc. imagine that you have a system that throws enormous amounts of logmessages and when the part your interrested in arrives your logfile is so large that analysing it is painfull. In this case you could automate this job by adding a lua script that filters all logmessages until an event arrives then switches the filter off and stops the log tail when the interresting part has been captured. Or you want to capture a sequence of events. Or you just want to capture a temperature logged and see this is a graph, lua scripts can do it for you.

Using Tailexpert internal functions

TailExpert exposes to following functions to lua scripts to control the application.

PrintLine*(string line) - print logmessage to active tab*
PrintLines*(LuaTable table) - print multiple logmessages to active tab*
PrintStatusLine*(string line) - print message to status bar*
LogMessage*(int severity, string message) - print a logmessage to TailExpert log file (defaults to C:\Temp\TailExpert.log)*
string GetDateTimeString*(string format) - get the current date/time formatted in given format (see .NET documentation on format sepcifiers)*
AddPluginMenuEntry*(ToolStripMenuItem item) - add a menu entry in the plugins menu*
RemovePluginMenuEntry*(ToolStripMenuItem item) - remove specified menu entry from plugins menu*
string AddFilter*(string tabName, string find, string column, string operation, bool invert, bool matchCase, bool regEx) - add filter to specified tab, returns a guid that identifies filter*
RemoveFilter*(string tabName, string guid) - remove filter specified with guid from specified tab*
ActivateFilter*(string tabName, string guid, bool active) - activate filter specified with guid on specified tab*
string AddTranslate*(string tabName, string find, string column, string operation, bool invert, bool matchCase, bool regEx) - add translation to specified tab, returns a guid that identifies translation*
RemoveTranslate*(string tabName, string guid) - remove translation specified with guid from specified tab*
ActivateTranslate*(string tabName, string guid, bool active) - activate translation specified with guid on specified tab*
string AddNotification*(string tabName, string find, string column, string operation, bool invert, bool matchCase, bool regEx) - add notification to specified tab, returns a guid that identifies notification*
RemoveNotification*(string tabName, string guid) - remove notification specified with guid from specified tab*
ActivateNotification*(string tabName, string guid, bool active) - activate notification specified with guid on specified tab*
Pause*(string tabName, boolean pause, boolean discard) - pause/run log, discard: discard incoming messages when paused*
LockScroll*(string tabName, boolean lockScroll) - lock scroll*
int GetLastIndex*(string tabName) - get last rowindex from specified tab*

SelectLogLines*(string tabName, int begin, int end) - select logmessages from specified tab from begin to end*
SaveSelectedLogLines*(string tabName, string filename, string encoding, bool lineNumbers) - save selected loglines from specified tab to file*
SaveLog*(string tabName, string filename, string encoding, bool lineNumbers) - save complete log from specified tab to file*
OpenFileTab*(string filename, bool addTimeStamp, bool pauzed) - open a file tab on specified filename*
OpenEventLogTab*(string eventLogName, bool addTimeStamp, bool pauzed) - open tab on specified event log*
OpenSysLogTab*(string tabName, int port, bool addTimeStamp, bool pauzed, bool saveToFile, string filename) - open tab on specified UDP port*
OpenSerialLogTab*(string tabName, string portName, string baud, string bits, string stop, string parity, string handshake, bool addTimeStamp, bool pauzed, bool saveToFile, string filename) - open tab on specified serial port*
string GetActiveTab*() - get the active (visible) tab*
SelectTab*(String tabName) - select the active (visible) tab*
CloseTab*(string tabName) - close selected tab*
List<string> GetTabNames*() - get list op opened tabs*
ClearTab*(string tabName) - clear tab contents*
ReloadTab*(string tabName) - reload tab from file*
RefreshTab*(string tabName) - refresh tab content from internal cache*

In the example code below some of the mentioned functions are used, a logfile is opened and a filter gets installed on this logfile. In the OnEnter() function a new tab is openend in the logview using the OpenFileTab function and next a filter is added using the AddFilter() function.

Code:

```
--[[
   This is an example script for the lua engine in TailExpert

   Purpose:
   This script will add a filter to an already opened tab and activate it
]]--
require("Config")

-- Import WIndows Assemblies
import("System")
import("System.Windows.Forms")
import("System.Drawing")
import("System.Drawing.Drawing2D")

function click()
   local filter = AddFilter("TailExpert.log", "Add Filter", ALLCOLUMNS, CONTAINS, false, false, false)
   ActivateFilter("TailExpert.log", filter, true)
end

--[[
Required predefined functions: Are called from TailExpert, adapt to your needs
]]--

function OnEvaluate(loglines)
   -- Do not put LogMessage() here this will overflow your logs as
   -- Evaluate is called every tail cycle
   -- If you need log make sure it's not output every cycle
--    if ScriptInitialized == true then
--    end
end

function OnEnter()
  ScriptInitialized = true
  LogMessage(INFO, "InstallFilter.lua script OnEnter()")

   OpenFileTab("c:\\temp\\TailExpert.log", false, false)
   dropDownItem = ToolStripMenuItem()
   dropDownItem.Text = "TestPlugin"
   dropDownItem.Click:Add(click)
   AddPluginMenuEntry(dropDownItem)
   -- tests the functions above
   ScriptInitialized = true
end

function OnExit()
  LogMessage(INFO, "InstallFilter.lua script OnExit()")
   ScriptInitialized = false  -- prevent Evaluate to be called

   if (dropDownItem ~= nil) then
     RemovePluginMenuEntry(dropDownItem)
   end
end
```

Using the ZedGraph library

The TailExpert installation also includes a copy of the zedGraph library (http://sourceforge.net/projects/zedgraph) . ZedGraph is a class library, user control, and web control for .net, written in C#, for drawing 2D Line, Bar, and Pie Charts. It features full, detailed customization capabilities. All functions of zedGraph are avalaible to the script engine of TailExpert, which makes it possible to visualize your logfile data in graphs while your logfile grows. You could use this for example to count the number of warnings and error of some kind and directly show them in a bargraph next to the log being examined. The bargraph can be updated when new loglines arrive. The example below shows a script the displays the function usage of TailExpert from the TailExpert logfile in C:\Temp\TailExpert.log in a bargraph, when you select the functions listed in the bargraph, you will see the bargraph change.

Code:

```
--[[
   This is an example script for the lua engine in TailExpert

   Purpose:
   This script demonstrates the use of the ZedGraph library available within TailExpert.
   It will plot a histogram based upon occurrences of actions detected in the TailExpert
   log file.
]]--
require("Config")

-- Import WIndows Assemblies
import("System")
import("System.Windows.Forms")
import("System.Drawing")
import("System.Drawing.Drawing2D")
import("ZedGraph")

local form = Form()
form.Height = 600
form.Width = 800
form.Text = "Output window of HistoExample.lua"
form.HelpButton=false
form.MaximizeBox=true
form.MinimizeBox=true

local graph = ZedGraphControl()
graph.Dock = DockStyle.Fill
myPane = graph.GraphPane

-- Set the titles and axis labels
myPane.Title.Text = "Bar Graph Example"
myPane.XAxis.Title.Text = "Categories"
myPane.YAxis.Title.Text = "Nr of Hits"

local label = String[5]
local dbl_arr = Double[5]

for i=0, 4,1 do
   dbl_arr[i] = 0.0
end

label[0] = "Pause"
```

```
label[1] = "Scroll"
label[2] = "Filter"
label[3] = "Compare"
label[4] = "Bookmarks"

-- Create the three BarItems, change the fill properties so the angle is at 90
-- degrees for horizontal bars
bar = myPane:AddBar( "Function usage", nil, dbl_arr, Color.Red )
bar.Bar.Fill = Fill( Color.Red, Color.White, Color.Red, 0 )

-- Set BarBase to the YAxis for horizontal bars
myPane.BarSettings.Base = BarBase.X
-- Make the bars stack instead of cluster
myPane.BarSettings.Type = BarType.Stack

-- Fill the axis background with a color gradient
myPane.Chart.Fill = Fill( Color.White, Color.LightGoldenrodYellow, 45.0 )
myPane.XAxis.Scale.TextLabels = label
myPane.XAxis.Type = AxisType.Text
graph:AxisChange()

form.Controls:Add(graph)
form:Show()

--[[
Script starts here
]]--

local filename = 'c:\\temp\\TailExpert.log'
local file = nill

local linenr = 0
function ReadFile(f)
    -- Do not put message here as it will make a recurring effect when you have log enabled and load TailExpert.log
    outputBuffer = {}
    if ScriptInitialized == true then
        if feof(f) ~= nill then
            while feof(f) ~= true do
                local line = f:read("*line")
                if line ~= nill then
                    PutLogLine( line )
                    if string.match(line, "ButtonPause_Click") ~= nil then
                        dbl_arr[0] = dbl_arr[0] + 1.0
                    end
                    if string.match(line, "ButtonScroll_Click") ~= nil then
                        dbl_arr[1] = dbl_arr[1] + 1.0
                    end
                    if string.match(line, "ButtonFilter_Click") ~= nil then
                        dbl_arr[2] = dbl_arr[2] + 1.0
                    end
                    if string.match(line, "compareTabsToolStripMenuItem_Click") ~= nil then
                        dbl_arr[3] = dbl_arr[3] + 1.0
                    end
                    if string.match(line, "NextBookmark_Click") ~= nil then
                        dbl_arr[4] = dbl_arr[4] + 1.0
                    end
                    if string.match(line, "PrevBookmark_Click") ~= nil then
                        dbl_arr[4] = dbl_arr[4] + 1.0
                    end
                end
            -- Create new pointPairList
            pointPairList = PointPairList()
```

```
            pointPairList:Add(nil, dbl_arr)
            curveItem = myPane.CurveList[0]
            curveItem.Points = pointPairList
            graph:AxisChange()
            graph:Refresh()
              end
          end
            end
end

--[[
Required predefined functions: Are called from TailExpert, adapt to your needs
]]--

function OnEvaluate(loglines)
   -- Do not put LogMessage() here this will overflow your logs as
   -- Evaluate is called every tail cycle
   -- If you need log make sure it's not output every cycle
   if ScriptInitialized == true then
      ReadFile(file)
   end
end

function OnEnter()
     LogMessage(INFO, "HistoExample.lua script OnEnter()")
     ScriptInitialized = true

   -- tests the functions above
   if file_exists(filename) then
      file = assert(io.open(filename, "r"))
   end
   file:seek("set", 0)
   ScriptInitialized = true
end

function OnExit()
     LogMessage(INFO, "HistoExample.lua script OnExit()")
     ScriptInitialized = false  -- prevent Evaluate to be called
     graph:Dispose()
     form:Close()
     file:close()
end
```

The engine used to enable .NET access from lua is luaInterface. Lua interface translates lua call to C# calls in .NET components and vise versa.

LuaInterface

The following documentation is borrowed from the luainterface documentation and describes the calling syntax. Examples showed here are just shown as an illustration and cannot be directly run in the lua engine of TailExpert. Use the examples supplied with TailExpert to embed C# methods in your lua scripts for TailExpert. The examples provided with TailExpert do not require you to '`require 'CLRPackage'`' as the scripting environment of TailExpert already takes care for that.

LuaInterface is a way for lua programs to access the CLR (Common Language Runtime), otherwise known as .NET in the Windows world and Mono on Linux. You can also use LuaInterface to easily add Lua scripting abilities to C# programs, for instance, but that is outside the scope of this tutorial and is discussed fully in the LuaInterface documentation.

Hello, World, Take One

Here is a very simple program which uses the CLR classes to write out the value of the square root of two:

```
require 'CLRPackage'

import "System"

Console = luanet.import_type "System.Console"
Math = luanet.import_type "System.Math"

Console.WriteLine("sqrt(2) is {0}",Math.Sqrt(2))
```

The first task for any LuaInterface program is to load any assemblies needed, and the second task is to import the types. In this case, the `Console` and `Math` classes are loaded explicitly, and we call their static methods `WriteLine` and `Sqrt`. LuaInterface will convert Lua numbers and strings into their .NET equivalents for us.

Like all hello programs, it is basically silly; the equivalent pure Lua program is of course just this one-liner:

```
print("sqrt(2) is "..math.sqrt(2))
```

But these techniques will be used to bring all the considerable power of the .NET framework into Lua programs, like the ability to easily write good-looking GUI applications, examine running processes, etc.

Hello, World, Take Two

If you allow for the usual 'public static main' stuff, the equivalent C# program is also a one-liner, thanks to the `using` statement. This brings a whole namespace into global scope, so if there's a `using System;` at the top, then any member of that namespace becomes directly available. It would be very convenient to have this available for LuaInterface programs as well. It turns out that this is not difficult at all, using a short utility library:

```
require 'CLRPackage'
import "System"

Console.WriteLine("sqrt(2) is {0}",Math.Sqrt(2))
```

Here the `import` function works very much like C#'s `using` statement. This program brings in the `System.IO` namespace, so now useful things like `Directory` and `Path` effectively become global.

```
require 'CLRPackage'
import "System"
import "System.IO"
Console.WriteLine("we are at {0}",Directory.GetCurrentDirectory())
```

Dealing with Arrays and .NET Collections

Array values are very common argument types, and are often returned by functions. LuaInterface provides a convenient index notation for accessing the values, but note that the index runs from zero to `Length-1`!

Creating an array of strings is easy. Use the type name followed by the size in square brackets - this is consistent with the C# syntax.

```
ss = String[3]
ss[0] = "one"
ss[1] = "two"
ss[2] = "three"
```

However, note that you can only initialize values in this simple way if you are dealing with objects and not numbers. Then you have to use `Array.SetValue` explicitly.

```
d = Double[4]
d[0] = 1.0
```

Will result in a: `System.InvalidCastException: Unable to cast object of type 'System.Double[]' to type 'System.Object[]'.`

You will need to do:

```
d:SetValue(0,1.0)
```

There are no implicit conversions of Lua tables into .NET types, but it is easy to do manually. Here is a useful function for creating an array of doubles from a table:

```
function make_double_array (tbl)
    local arr = Double[#tbl]
    for i,v in ipairs(tbl) do
        arr:SetValue(v,i-1)
    end
    return arr
end
```

Other collection types are treated similarly; generally, if the object is indexable, Lua will be able to index it naturally as well.

There is no direct support in Lua for enumerables, as there is in C# with the `foreach` statement, but it is not difficult to write a function that will allow us to iterate through any such collection using a Lua `for` statement:

```
function enum(o)
   local e = o:GetEnumerator()
   return function()
      if e:MoveNext() then
        return e.Current
```

```
     end
   end
end

for v in enum(args) do print(v) end
```

Dealing with Exceptions

There are some .NET methods which will throw exceptions on error. These are converted into regular Lua errors by LuaInterface, so you need to execute these methods in a *protected call* to avoid your program crashing. For instance, `load_assembly` usually loads assemblies from the Global Assembly Cache (GAC) but it will work with your own assemblies if you end them with an explicit '.dll'.

However, it will not load an assembly given an arbitrary path. This is not difficult to get around, but the function we need is one of those that throw exceptions. Consider this script:

```
-- load.lua
require "CLRPackage"
import "System.Reflection"

function get_assembly_name(name)
    local res
    local suc,err = pcall(function()
        res = AssemblyName.GetAssemblyName(name)
    end)
    if suc then
        return res
    else
        return nil,err
    end
end

print(get_assembly_name(arg[1]))
```

`GetAssemblyName` returns an object which you can pass to `Assembly.Load`, but we have to call it using `pcall`. `pcall` will usually just return `true`, but if an error occured it returns `false,error`. In effect, this is Lua's `try..catch` mechanism. The variable err will contain the exception thrown and the message from the exception in string format.

Showing a Form

Showing a form is very straightforward, once you have brought in the necessary assembly, and the Form type itself.

```
-- form1.wlua
require 'CLRPackage'
import("System.Windows.Forms")
form = Form()
form.Text = "Hello, World!"
form:ShowDialog()
```

Since this is a GUI script, I have given it a .wlua extension. The convention is that .lua is associated with the console version lua.exe, and .wlua is associated with the GUI version wlua.exe. Then, if you run this script from Windows Explorer you will not get the dreaded black console window appearing as well. (Also, in Lua for Windows .wlua scripts can be properly run and debugged from SciTE.)

Since explicitly bringing in all the required types can be tedious, I will now switch to using CLRPackage/import for the next example:

```
-- form2.wlua
require 'CLRPackage'
import "System.Windows.Forms"
import "System.Drawing"

form = Form()
form.Text = "Hello, World!"
button = Button()
button.Text = "Click Me!"
button.Location = Point(20,20)
button.Click:Add(function()
    MessageBox.Show("We wuz clicked!",arg[0],MessageBoxButtons.OK)
end)

form.Controls:Add(button)
form:ShowDialog()
```

Windows.Forms applications are conceptually very simple; one creates controls of various types, sets their properties, and adds them to the control array of the form, which then can be shown. One attaches functions to the *events* of the objects to get feedback from the user or the system.

Unlike IUP, controls are usually given an explicit position. However, controls can be docked to the sides of their *container*:

```
-- form3.wlua
require 'CLRPackage'
import "System.Windows.Forms"
import "System.Drawing"

button = Button()
button.Text = "Click!"
button.Dock = DockStyle.Top
edit = RichTextBox()
edit.Dock = DockStyle.Fill

form = Form()
form.Text = "Hello, World!"
form.Controls:Add(edit)
form.Controls:Add(button)
form:ShowDialog()
```

In this example, a text box is made to fill all of the form, and then a button is added which is to be docked to the top side of the form. (The creation order of the text box and the button is not important, but they do have to be added in this order.)

More Useful Utilities

The `CLRForm` module contains some very useful utilties for doing Windows Forms programming in Lua. `lconsole` is built using `CLRForm`, so all of these utilities can be tested in this environment.

Prompting User for Data

A common need is to ask the user for a single string:

```
require "CLRForm"
if PromptForString("MyApp","Give your name","") then
    MessageBox.Show("Name!",arg[0],MessageBoxButtons.OK)
end
```

People however get irritated by being asked tiny questions, so in true bureaucratic style we would like her to fill in a form. This is a very common thing in GUI applications, and generally requires far too much coding. Some of that coding may be generated by the wizards that lurk in your IDE, but sometimes the ideal amount of code is zero. Consider this:

```
-- auto1.wlua
require "CLRForm"

data = {
    firstname = "",
    lastname = "",
    age = 0,
    title = "",
    phone = "",
    email = ""
}

form = AutoVarDialog { Text = "Please Supply Details", Object = data;
    "First Name:","firstname",
    "Last Name:","lastname",
    "Age:","age",
    "Title:","title",
    "Phone number:","phone",
    "E-mail Address:","email"
}

if form:ShowDialogOK() then
   print 'ok'
end

os.exit(0)
```

The call to `AutoVarDialog` automatically generates a dialog based on a template, which maps descriptive labels to the actual fieldnames. This kind of trick is possible in highly dynamic languages like Lua, where the actual type at runtime of any object is easy to determine. So by default we will try to edit numbers and strings with text boxes, and boolean values with checkboxes. At this level, some input

validation is already possible; `age` was a number, so whatever gets typed into the 'Age:' box must be convertable to a number and the user will not be allowed to proceed successfully until this is fixed.

But the general problem of validation remains. It's a bad idea to let bogus data into your system, and it should be caught as soon as possible. Also, fields like 'Title' are profoundly confusing. People will put in what their interpretation tells them, rather than your interpretation. So the next example shows an extra (optional) validation field that can follow the field name:

```
-- auto1.wlua
require "CLRForm"

data = {
    firstname = "steve",
    lastname = "donovan",
    age = 16,
    title = "Mr",
    phone = "+27116481212",
    email = "steve.j.donovan@gmail.com"
}

form = AutoVarDialog { Text = "Please Supply Details", Object = data;
    "First Name:","firstname",NonBlank,
    "Last Name:","lastname",NonBlank,
    "Age:","age",Range(16,120),
    "Title:","title",{"Mr","Ms","Dr","Prof"},
    "Phone number:","phone",Match ('^%+%d+$',"Not a valid phone no."),
    "E-mail Address:","email",Match ("%S+@%S+","Not valid email")
}

if form:ShowDialogOK() then
   print 'ok'
end

os.exit(0)
```

Notice that by making 'title' to be explicitly a list of items, we can now deduce that a drop-down list is the appropriate way to present the choice to the user. As for 'age', numerical values nearly always have a valid range. Text fields require more complicated validation - here phone numbers must be entered in international format, and email addresses must have a '@' somewhere. (They can still be utterly bogus, but at least they are well-formed rubbish ;))

Here is a more complete example, showing off file entry fields and booleans.

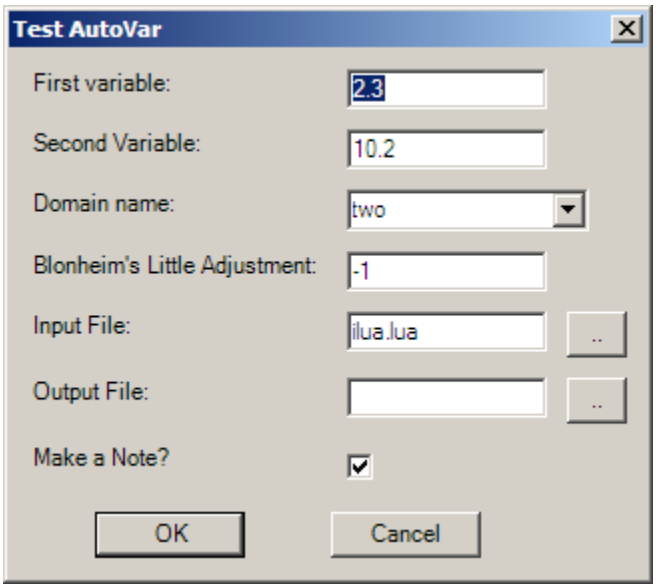
```
-- autoform.wlua
require "CLRForm"

tbl = {
    x = 2.3,
    y = 10.2,
    z = "two",
    t = -1.0,
    file = "c:\\lang\\lua\\ilua.lua",
    outfile = "",
```

```
    res = true,
}

form = AutoVarDialog { Text = "Test AutoVar", Object = tbl;
    "First variable:","x", Range(0,4),
    "Second Variable:","y",
    "Domain name:","z", {"one","two","three"; Editable=true},
    "Blonheim's Little Adjustment:","t",
    "Input File:","file",FileIn "Lua (*.lua)|C# (*.cs)",
    "Output File:","outfile",FileOut "Text (*.txt)",
    "Make a Note?","res",
}

if form:ShowDialogOK() then
    print(tbl.x,tbl.z,tbl.res,tbl.file)
end
```



Stream Layout

Generally, Windows Forms controls are positioned absolutely *within* their parent control. Other frameworks use explicit *layout* strategies, which can be very useful. For instance, the `AutoVarDialog` form uses automatic control layout. `CLRForm` defines a useful `StreamLayout` class, which works rather like writing to a file; each new control is placed after the last, until there's an explicit request for a 'new line'.

This example attaches a rough-and-ready toolbar to a form. A `Panel` control is a useful tool in composing forms; by default it does not have a border. It is easy to attach it to the top of the form with `DockStyle.Top`. The `StreamLayout` is used to put the controls in a nice row:

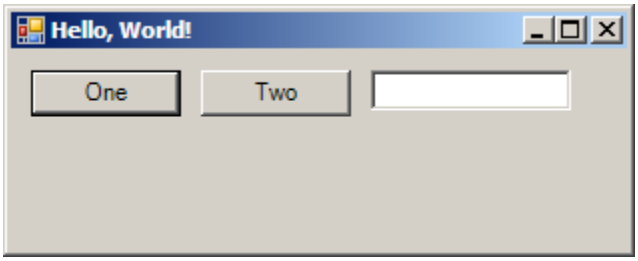
```
-- layout1.wlua
require 'CLRForm'

panel = Panel()
```

```
layout = StreamLayout(panel)
b = Button()
b.Text = "One"
layout:Add(b)
b = Button()
b.Text = "Two"
layout:Add(b)
t = TextBox()
layout:Add(t)
layout:Finish()

form = Form()
form.Text = "Hello, World!"
panel.Dock = DockStyle.Top

form.Controls:Add(panel)
form:ShowDialog()
```



Menus and Form Classes

Object-oriented style is particularly appropriate for GUI applications. Although Lua does not have a formal concept of 'class', it is not difficult to add a class mechanism. Then our forms can be self-contained objects, as is obligatory in C#. The classes described here are not proper derived classes of the .NET Form class, rather they use *delegation*; if our form object cannot find a method or property within itself, it calls the delegate object, which *is* a Form object.

```
-- menuform.lua
require "CLRForm"

MenuForm = class()

function MenuForm:_init ()
    self.form = Form()
    self.name = "Dolly"
    -- this method can only be called once we've set up our own fields!
    self:delegate(self.form)
    self.Text = "A Simple Form Class"
    self.FormBorderStyle = FormBorderStyle.FixedDialog
    self.MaximizeBox = false
    self.MinimizeBox = false
    self.ContextMenu = popup_menu {
        "First",method(self,self.first),
        "Second",method(self,self.second),
    }
end
```

```
function MenuForm:first()
    local name = PromptForString(arg[0],"Give the dog a name",self. name)
    if name then self.name = name end
end

function MenuForm:second()
    ShowMessageBox (self.name)
end

form = MenuForm()
form:ShowDialog()
```

Notice the useful `popup_menu` function, which takes some of the tedium out of defining menus. (A more complete example can be found in the source for `lconsole.lua`, where shortcuts are defined, etc.) The little function method has a simple definition; it creates a callback function which actually does the function call, passing the object as the first parameter (the *self* object):

```
function method (obj,fun)
    return function()
        fun(obj)
    end
end
```

Main menus are also easy to construct; here is how `lconsole.wlua` does its menu:

```
local menu = main_menu {
    "File",{
        "Load Lua(CtrlO)",load_lua,
        "Save Session(CtrlS)",save_session,
        "Save As Text",save_text,
        "E&xit(CtrlX)",function() os.exit(0) end,
    },
    "Run",{
        "Save and Go(F5)",save_and_go,
        "Create Function",function() fun() end,
        "Delete Item",delete_list_item,
        "Clear Code Pane",clear_code,
    },
    "History", {
        "Last(AltUpArrow)", function() get_history(true) end,
        "Previous(AltDownArrow)", function() get_history(false) end
    }
}
```

If an item string contains '(...)', then it's interpreted as a name from the `System.Windows.Forms.Shortcut` enumeration; see the .NET documentation for all available constants.

A useful Form Constructor

`LuaForm` makes creating form objects easier. You feed `LuaForm` with a table; the array-like part of the table contains the controls, and the map-like part contains any properties you wish to assign to the control. If there is only one control to be added, then it's assumed that we want it to fill the whole client area of the form.

```
> txt = RichTextBox()
> f = LuaForm{txt;Text="hello"}
> f:Show()
```

If multiple controls are added, then one can provide their dock styles:

```
> txt = RichTextBox()
> list = ListBox()
> f = LuaForm{'Fill',txt,'Left',list; Text = "Test!"}
> f:Show()
```

Appendix A

C# Regular Expressions Cheat Sheet

Cheat sheet for C# regular expressions meta characters, operators, quantifiers etc.

Character	Description
****	Marks the next character as either a special character or escapes a literal. For example, "n" matches the character "n". "\n" matches a newline character. The sequence "\\" matches "\" and "\(" matches "(". Note: double quotes may be escaped by doubling them: "<a href=""...>"
^	Depending on whether the MultiLine option is set, matches the position before the first character in a line, or the first character in the string.
$	Depending on whether the MultiLine option is set, matches the position after the last character in a line, or the last character in the string.
*****	Matches the preceding character zero or more times. For example, "zo*" matches either "z" or "zoo".
+	Matches the preceding character one or more times. For example, "zo+" matches "zoo" but not "z".
?	Matches the preceding character zero or one time. For example, "a?ve?" matches the "ve" in "never".
.	Matches any single character except a newline character.
(pattern)	Matches *pattern* and remembers the match. The matched substring can be retrieved from the resulting **Matches** collection, using Item **[0]...[n]**. To match parentheses characters ( ), use "\(" or "\)".
(?<name>pattern)	Matches *pattern* and gives the match a name.
(?:pattern)	A non-capturing group
(?=...)	A positive lookahead

(?!...)	A negative lookahead
(?<=...)	A positive lookbehind .
(?<!...)	A negative lookbehind .
x|y	Matches either *x* or *y*. For example, "z|wood" matches "z" or "wood". "(z|w)oo" matches "zoo" or "wood".
{*n*}	*n* is a non-negative integer. Matches exactly *n* times. For example, "o{2}" does not match the "o" in "Bob," but matches the first two o's in "foooood".
{*n*,}	*n* is a non-negative integer. Matches at least *n* times. For example, "o{2,}" does not match the "o" in "Bob" and matches all the o's in "foooood." "o{1,}" is equivalent to "o+". "o{0,}" is equivalent to "o*".
{*n*,*m*}	*m* and *n* are non-negative integers. Matches at least *n* and at most *m* times. For example, "o{1,3}" matches the first three o's in "fooooood." "o{0,1}" is equivalent to "o?".
[*xyz*]	A character set. Matches any one of the enclosed characters. For example, "[abc]" matches the "a" in "plain".
[^*xyz*]	A negative character set. Matches any character not enclosed. For example, "[^abc]" matches the "p" in "plain".
[*a-z*]	A range of characters. Matches any character in the specified range. For example, "[a-z]" matches any lowercase alphabetic character in the range "a" through "z".
[^*m-z*]	A negative range characters. Matches any character not in the specified range. For example, "[m-z]" matches any character not in the range "m" through "z".
\b	Matches a word boundary, that is, the position between a word and a space. For example, "er\b" matches the "er" in "never" but not the "er" in "verb".
\B	Matches a non-word boundary. "ea*r\B" matches the "ear" in "never early".
\d	Matches a digit character. Equivalent to [0-9].
\D	Matches a non-digit character. Equivalent to [^0-9].
\f	Matches a form-feed character.

\k	A back-reference to a named group.
\n	Matches a newline character.
\r	Matches a carriage return character.
\s	Matches any white space including space, tab, form-feed, etc. Equivalent to "[\f\n\r\t\v]".
\S	Matches any nonwhite space character. Equivalent to "[^ \f\n\r\t\v]".
\t	Matches a tab character.
\v	Matches a vertical tab character.
\w	Matches any word character including underscore. Equivalent to "[A-Za-z0-9_]".
\W	Matches any non-word character. Equivalent to "[^A-Za-z0-9_]".
\num	Matches *num*, where *num* is a positive integer. A reference back to remembered matches. For example, "(.)\1" matches two consecutive identical characters.
n	Matches *n*, where *n* is an octal escape value. Octal escape values must be 1, 2, or 3 digits long. For example, "\11" and "\011" both match a tab character. "\0011" is the equivalent of "\001" & "1". Octal escape values must not exceed 256. If they do, only the first two digits comprise the expression. Allows ASCII codes to be used in regular expressions.
\x*n*	Matches *n*, where *n* is a hexadecimal escape value. Hexadecimal escape values must be exactly two digits long. For example, "\x41" matches "A". "\x041" is equivalent to "\x04" & "1". Allows ASCII codes to be used in regular expressions.
\u*n*	Matches a Unicode character expressed in hexadecimal notation with exactly four numeric digits. "\u0200" matches a space character.
\A	Matches the position before the first character in a string. Not affected by the MultiLine setting
\Z	Matches the position after the last character of a string. Not affected by the MultiLine setting.
\G	Specifies that the matches must be consecutive, without any intervening non-matching characters.