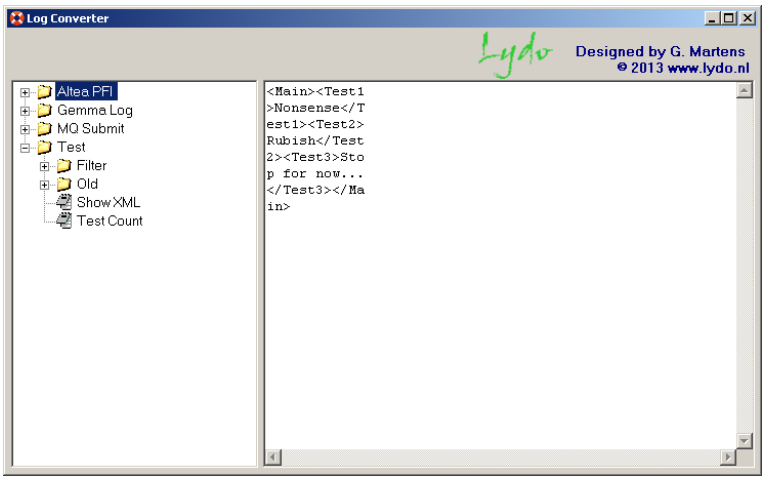
Log Converter 2.0

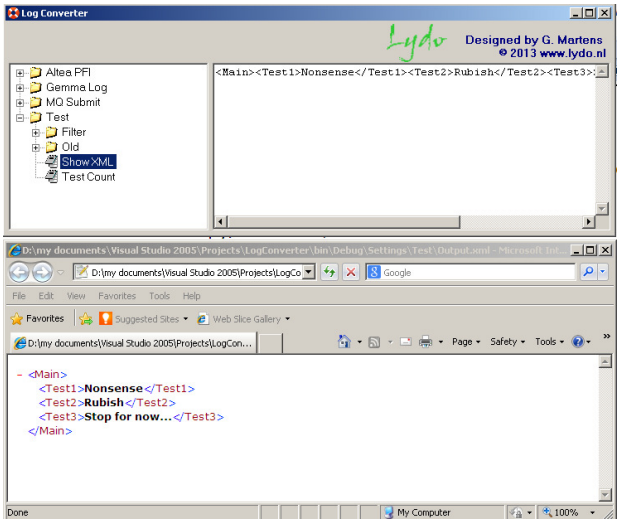
Log converter is a small program to help you analyzing your log files. It can filter text, count events, replace text, create (text) reports, save output and open other programs.



The left treeview contain the actions that can be done on input. The actions are actually text files (and can be separated in different folders). These text files contain all the commands and variables to adjust the input.

The textbox on the right contains the input (and later the output) to perform an action on. Input can be set by one of the following methods:

- Copy/Paste text;
- Drag a (text) file in the textbox;
- Use the [INPUT] command;
- Use the [LOADINPUT] command.



As an example, pressing the “ShowXML” action will remove all linefeeds, save the content and open the new file in Internet Explorer to show a decent XML file.

In the following pages a few examples are explained. A list of all possible commands and variables can be found at the end of this document.

With ALT-L, a log file is written from the last action. The log is cleared with the start of each action.

Example 1: "ShowXML"

Some log tools show useful information, however that data is distorted by the layout of the log tool. This example assumes that data is copied from a text field with linefeeds at fixed distances.

```
<Main><Test1
>Nonsense</T
est1><Test2>
Rubbish</Test
2><Test3>Sto
p for now...
</Test3></Ma
in>
```

The action file "ShowXML.txt" contains the following text:

```
'Load the input file as a courtesy, normally this will be a copy paste action
[LOADINPUT]#BASEPATH#\Settings\Test\Distorted XML.log[/LOADINPUT]

'Now show the content via a Messagebox
[MSGOK]This is the content of the file:#CRLF##CRLF##INPUT#[/MSGOK]

'Remove the Linefeed and carriage returns from the input
[SEARCHONTEXT]#LF#[/SEARCHONTEXT]
[REPLACEWITHTEXT][/REPLACEWITHTEXT]
[SEARCHONTEXT]#CR#[/SEARCHONTEXT]
[REPLACEWITHTEXT][/REPLACEWITHTEXT]

'This part is disregarded, it will normally place linefeeds and carriage returns between two
tags
[REM]
[SEARCHONTEXT]><[/SEARCHONTEXT]
[REPLACEWITHTEXT]>#CRLF#<[/REPLACEWITHTEXT]
[/REM]

'Initialize the filedialogs and ask for the filename
[FILEINITDIR]#BASEPATH#\Settings\Test[/FILEINITDIR]
[FILETYPE]XML[/FILETYPE]
[FILEINITNAME]Output.xml[/FILEINITNAME]
[FILENAME]Save as XML![/FILENAME]

'Save the content to the previous specified filename
[SAVECONTENT]#FILENAME#[/SAVECONTENT]

'The filename with quotes are provided in the arguments
'The file is then shown in iexplorer
[RUNARGS]"#FILENAME#"[/RUNARGS]
[RUNCOMMAND]C:\Program Files\Internet Explorer\iexplore.exe[/RUNCOMMAND]
```

First note that commands have always a fixed format:
`[COMMAND]command variable[/COMMAND]`
Note the brackets, capital letters and closing tag. The reason for using this layout is that variables can also contain (large) xml parts. Take care that the closing tag of the command corresponds correctly with the starting tag. Embedded commands are not allowed.

Also some internal variables can be used. Those variables are enclosed with '#'. Before a command is run, the #VARIABLE# text is replaced with the corresponding value. A list of all possible commands and variables can be found at the end of this document.

Now take a look at the pseudo code that will perform the action "ShowXML".

Normally the input is copied from the log tool and pasted in the textbox. For convenience, a command is used to load that input from a file:

```
[LOADINPUT]#BASEPATH#\Settings\Test\Distorted XML.log[/LOADINPUT]
```

#BASEPATH# corresponds with the folder where the “Log Converter.exe” is located.

For demonstration purposes, a message box pops up with the original content:

```
[MSGOK]This is the content of the file:#CRLF##CRLF##INPUT#[/MSGOK]
```

#INPUT# returns the inputvalue, #CRLF# returns a linefeed.

To replace text, first the SEARCHONTEXT command has to be used. With the REPLACEWITHTEXT command does the actual replacements.

```
[SEARCHONTEXT]#LF#[/SEARCHONTEXT]
[REPLACEWITHTEXT][/REPLACEWITHTEXT]
```

In this case, all linefeeds (CHR(10)) are replaced with empty space (“”). The content is now:

```
<Main><Test1>Nonsense</Test1><Test2>Rubbish</Test2><Test3>Stop for now...</Test3></Main>
```

To obtain a filename from a user the FILENAME command can be used to show a file dialog. The other commands are not required but can be used to initialize the file dialog

```
[FILEINITDIR]#BASEPATH#\Settings\Test[/FILEINITDIR]
[FILETYPE]XML[/FILETYPE]
[FILEINITNAME]Output.xml[/FILEINITNAME]
[FILENAME]Save as XML![/FILENAME]
```

Because of the word “Save” in the FILENAME command, a Save File dialog will be shown, otherwise an Open File dialog is shown. When cancel is pressed in the dialog, no further commands are handled.

The content is saved to the previous determined filename:

```
[SAVECONTENT]#FILENAME#[/SAVECONTENT]
```

Now we would like to open the XML file in another program:

```
[RUNARGS]"#FILENAME#"[/RUNARGS]
[RUNCOMMAND]C:\Program Files\Internet Explorer\iexplore.exe[/RUNCOMMAND]
```

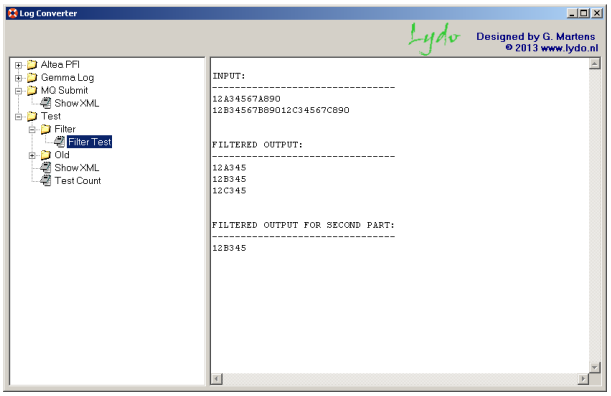
RUNCOMMAND contains the program, RUNARGS contain the arguments to use that program.

To use the default program to open the file, the following code can be used:

```
[RUNCOMMAND]#FILENAME#[/RUNCOMMAND]
```

Example 2: "Filter Test"

This is a simple example to show the Filter function. As simple as this example is, the possibilities are great. Returning certain messages, data or summaries in a 140MB file is now a matter of seconds…



Take another look at the pseudo code:
Note that values can be stored in internal variables with the commands A, B and C. These values can later be used with the variables #A#, #B# and #C#.

Filtering is done with a start value, an end value and (optional) a value to separate the found parts. With FILTEROUT, all the found parts are removed (and replaced with the filterseperator). With FILTER, all the found parts are returned (with filterseperator between them).

The OUTPUT command returns the found values in a nice format.

```
'Store input in variable A and use that as
input
[A]12A34567A890
12B34567B89012C34567C890[/A]
[INPUT]#A#[/INPUT]
[MSGOK]Input is obtained from the [INPUT]
command: #CRLF##A#[/MSGOK]

'Set the filter values and filter out all
those found parts
[FILTERSTART]1[/FILTERSTART]
[FILTEREND]45[/FILTEREND]
[FILTERSEPARATOR]-->[/FILTERSEPARATOR]
[FILTEROUT]0[/FILTEROUT]

'Provide a message box with the found
information
[MSGOK]All Text starting with '1' and ending
with '45' is filtered out. #CRLF#A seperator
'-->' is placed between the parts:

#INPUT#[/MSGOK]

'Now ask if the filter in must be shown,
Cancel will abort all following commands
[MSGOKCANCEL]That was FILTEROUT, would you
like to try FILTER?[/MSGOKCANCEL]

'Use the input again as set earlier in
variable A
[INPUT]#A#[/INPUT]

'Now filter in and store it in variable B
[FILTERSTART]1[/FILTERSTART]
[FILTEREND]45[/FILTEREND]
[FILTERSEPARATOR]#CRLF#[/FILTERSEPARATOR]
[FILTER][/FILTER]
[B]#INPUT#[/B]

'Use the standard input again and give back
the second part that has been found during
filtering
'Store it in variable C
[INPUT]#A#[/INPUT]
[FILTER]2[/FILTER]
[C]#INPUT#[/C]

'Now output all the determined variables in a
nice format
[OUTPUT]
INPUT:
---------------------------------
#A#


FILTERED OUTPUT:
---------------------------------
#B#


FILTERED OUTPUT FOR SECOND PART:
---------------------------------
#C#
[/OUTPUT]
```

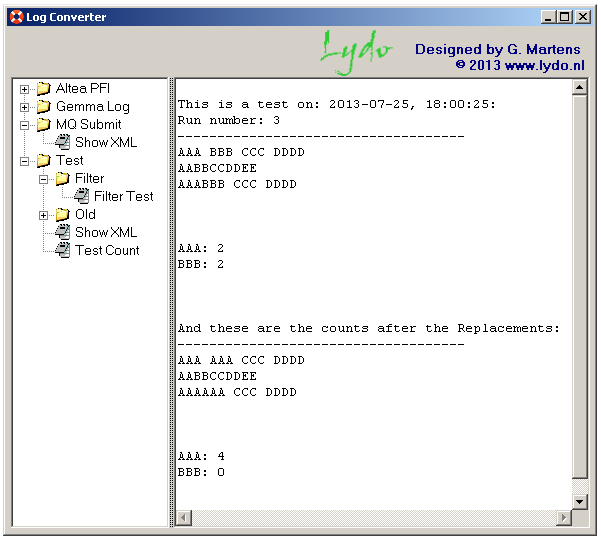
Example 3: "Test Count"

This example shows how to create a summary of a log.

Copy the following text as input and past it in the textbox:

AAA BBB CCC DDDD
AABBCCDDEE
AAABBB CCC DDDD

DoubleClick on "Test Count"



Most things in the code have been used in the previous examples, but some things are added...

This command will open an input box and store the value in variable #A#

```
[A]{INPUTBOX|#Run nr:#}[/A]
```


```
'Just a remark, but use this for your input
as copy/paste
[REM] This is input for the test
AAA BBB CCC DDDD
AABBCCDDEE
AAABBB CCC DDDD
[/REM]

[MSGOK]Just copy the text from this file in
the textbox:
AAA BBB CCC DDDD
AABBCCDDEE
AAABBB CCC DDDD
[/MSGOK]

'Initialize the filedialogs and ask for a
filename
[FILEINITDIR]#BASEPATH#\Settings\Test[/FILEIN
ITDIR]
[FILETYPE]TXT[/FILETYPE]
[FILEINITNAME]Output.txt[/FILEINITNAME]
[FILENAME]Open a file![/FILENAME]
[MSGOK]No file is not actually opened, it
just gives a value back:
#CRLF##FILENAME#[/MSGOK]

'Ask for something with an inputbox and store
that value in a variable 'A' for later...
[A]{INPUTBOX|#Run nr:#}[/A]

'Set the output text. All actions are done on
the original content
'Note that a count is done on certain text...
[OUTPUT]
This is a test on: #YYYY#-#MM#-#DD#, #TIME#:
Run number: #A#
-------------------------------------
#INPUT#

AAA: {COUNT|#AAA#}
BBB: {COUNT|#BBB#}


[/OUTPUT]

'Ask for confirmation, a CANCEL will stop
further processing
[MSGOKCANCEL]Continue?[/MSGOKCANCEL]

'Replace all BBB with AAA in the content.
[SEARCHONTEXT]BBB[/SEARCHONTEXT]
[REPLACEWITHTEXT]AAA[/REPLACEWITHTEXT]

'Add info to the previous output
[ADD2OUTPUT]
And these are the counts after the
Replacements:
-------------------------------------
#INPUT#

AAA: {COUNT|#AAA#}
BBB: {COUNT|#BBB#}

[/ADD2OUTPUT]
```

The output is determined in the OUTPUT command. However, the content is still available at the background (can be obtained with #INPUT#). In this case more replacements (or filters if required) can be made and added to the already existing output with the command ADD2OUTPUT.

The following pseudo code will count the defined text (in this case “AAA”) and return that value to the output.

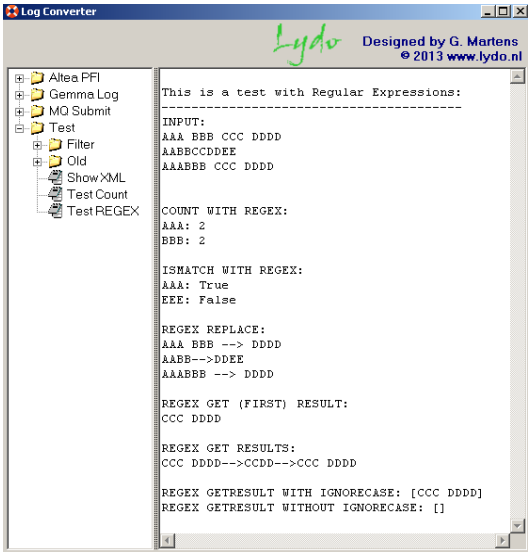
```
{COUNT|#AAA#}
```

Note that this text can be several lines, as long as it is preceded with “{COUNT|#” and closed with “#}”.

Example 4: "Test Regex"

This example shows how to make use of the Regular Expressions functionality.
For more info on regular expressions and their patterns, please check internet (Google on "regular expressions" or "vb.net regex")

DoubleClick on "Test Regex"



Regular Expressions use patterns to make matches in a string. These patterns are provided via the applicable commands. Results from these commands are either saved in the INPUT or in a variable called REGEXRESULT.

The example shows the use of the commands with very simple patterns, the use of patterns are not a part of this manual. Check internet to use patterns in a more advanced way.

- [REGEXGETCOUNT]PATTERN[REGEXGETCOUNT] returns the amount of matches in REGEXRSULT.
- [REGEXGETISMATCH]PATTERN[/REGEXGETISMATCH] returns True/False for matches found in REGEXRSULT.
- [REGEXREPLACEMENT]ReplacementString[/REGEXREPLACEMENT] sets the replacementstring.
- [REGEXREPLACE]PATTERN[/REGEXREPLACE] replaces the found matches with the replacementstring and returns the string in REGEXRSULT.
- [REGEXREPLACE-INPUT]PATTERN[/REGEXREPLACE-INPUT] replaces the found matches with the replacementstring and returns the string in INPUT.
- [REGEXGETRESULT]PATTERN[/REGEXGETRESULT] returns the first match of a pattern on the input in REGEXRSULT.
- [REGEXGETRESULT-INPUT]PATTERN[/REGEXGETRESULT-INPUT] returns the first match of a pattern on the input in INPUT.
- [REGEXSEPERATOR]SEPERATOR[/REGEXSEPERATOR] sets the seperator string.
- [REGEXGETRESULTS]PATTERN[/REGEXGETRESULTS] (note the extra 's' on the end) returns all the matches of a pattern on the input in REGEXRSULT. The matches are seperated from each other by the sepererator string.
- [REGEXGETRESULTS-INPUT]PATTERN[/REGEXGETRESULTS-INPUT] Idem, but now the result is stored in INPUT.

- [REGEXOPTIONS]OPTIONS[/REGEXOPTIONS] sets the value for the settings on which Regex works. Check internet (google on 'vb.net RegexOptions Enumeration') for more information on the options. The values can be:

0: None	8: Compiled	128: NOT TO BE USED
1: Ignorecase	16: Singleline	256: ECMAScript
2: Multiline	32: IgnorePatternWhitespace	512: CultureInvariant
4: ExplicitCapture	64: RightToLeft	

```
'Use this string as testinput
[INPUT]AAA BBB CCC DDDD
AABBCCDDEE
AAABBB CCC DDDD[/INPUT]

'Now provide the header and input message as output
[OUTPUT]
This is a test with Regular Expressions:
----------------------------------------
INPUT:
#INPUT#

[/OUTPUT]

'Count the matches that comply with the regex pattern, in this case 'AAA'
'Result is returned via variable REGEXRESULT
[REGEXGETCOUNT]AAA[/REGEXGETCOUNT]
[ADD2OUTPUT]
COUNT WITH REGEX:
AAA: #REGEXRESULT#[/ADD2OUTPUT]
'now with another pattern, in this case 'bbb', note that ignoreCase is default on
[REGEXGETCOUNT]bbb[/REGEXGETCOUNT]
[ADD2OUTPUT]
BBB: #REGEXRESULT#
[/ADD2OUTPUT]

'Now check if matches are found on a pattern and return a boolean
'Result is returned via variable REGEXRESULT
[REGEXGETISMATCH]AAA[/REGEXGETISMATCH]
[ADD2OUTPUT]
ISMATCH WITH REGEX:
AAA: #REGEXRESULT#[/ADD2OUTPUT]
[REGEXGETISMATCH]EEE[/REGEXGETISMATCH]
[ADD2OUTPUT]
EEE: #REGEXRESULT#
[/ADD2OUTPUT]
```

TO BE CONTINUED ON NEXT PAGE

```
'This part is for a replacement
'The replacement string is set first, then the Replace command is given with a pattern.
'Result is returned via variable REGEXRESULT
'If the result must be returned to the input, then use REGEXREPLACE-INPUT, not
REGEXREPLACE
[REGEXREPLACEMENT]-->[/REGEXREPLACEMENT]
[REGEXREPLACE]\bCCC\b|CC[/REGEXREPLACE]
[ADD2OUTPUT]
REGEX REPLACE:
#REGEXRESULT#
[/ADD2OUTPUT]

'This part returns the (first) match that complies with a provided pattern.
'Result is returned via variable REGEXRESULT
'If the result must be returned to the input, then use REGEXGETRESULT-INPUT, not
REGEXGETRESULT
[REGEXGETRESULT]C.*D[/REGEXGETRESULT]
[ADD2OUTPUT]
REGEX GET (FIRST) RESULT:
#REGEXRESULT#
[/ADD2OUTPUT]

'This part returns all the matches that complies with a provided pattern.
'The matches are seperated from each other with a provided seperator
'Result is returned via variable REGEXRESULT
'If the result must be returned to the input, then use REGEXGETRESULTS-INPUT, not
REGEXGETRESULTS
[REGEXSEPERATOR]-->[/REGEXSEPERATOR]
[REGEXGETRESULTS]C.*D[/REGEXGETRESULTS]
[ADD2OUTPUT]
REGEX GET RESULTS:
#REGEXRESULT#
[/ADD2OUTPUT]

'Options can be set for different settings:
'  0: None
'  1: Ignorecase
'  2: Multiline
'  4: ExplicitCapture
'  8: Compiled
' 16: Singleline
' 32: IgnorePatternWhitespace
' 64: RightToLeft
'128: NOT TO BE USED
'256: ECMAScript
'512: CultureInvariant
'This example shows the differences for ignorecase
[REGEXOPTIONS]1[/REGEXOPTIONS]
[REGEXGETRESULT]c.*d[/REGEXGETRESULT]
[ADD2OUTPUT]
REGEX GETRESULT WITH IGNORECASE: [#REGEXRESULT#]
[/ADD2OUTPUT]
[REGEXOPTIONS]0[/REGEXOPTIONS]
[REGEXRESULT]TEST[/REGEXRESULT]
[REGEXGETRESULT]c.*d[/REGEXGETRESULT]
[ADD2OUTPUT]REGEX GETRESULT WITHOUT IGNORECASE: [#REGEXRESULT#]
[/ADD2OUTPUT]
```

Example 5: "Loop & Tabs"

This example shows the usage of a loop and tabsettings.

The default tabinterval is 8. This value can be changed or specific tabstops can be entered. When both set, the last one will override the previous setting.

The tabsetting however is a local program value. For exporting or printing functionality it is sometimes required to change all tabs with the correct amount of spaces. The TAB2SPACES command will do that.

```
[TABINTERVAL]8[/TABINTERVAL]
[TABSTOPS]4;15;25[/TABSTOPS]

[OUTPUT]
Demo with TABS and LOOP:

         1         2         3         4
12345678901234567890123456789012345678901234
----------------------------------------
[/OUTPUT]

[LOOPVAR]1;2;3;4;AAAV;  AB; CD ;99;Test;10;11 [/LOOPVAR]
[LOOP]
[ADD2OUTPUT]#LVC##TAB##LV##TAB#Test #LVC##CRLF#[/ADD2OUTPUT]
[/LOOP]

[TAB2SPACES][/TAB2SPACES]
```

```
Demo with TABS and LOOP:

         1         2         3         4
12345678901234567890123456789012345678901234
----------------------------------------
1   1          Test 1
2   2          Test 2
3   3          Test 3
4   4          Test 4
5   AAAV       Test 5
6     AB       Test 6
7    CD        Test 7
8   99         Test 8
9   Test       Test 9
10  10         Test 10
11  11         Test 11
```

When a lot of actions are repeated, a loop can be helpful. First, all loop variables are set via the LOOPVAR command. Their position in the loop is shown via the LoopVarCounter variable #LVC#. All commands within a LOOP command are repeated for the amount of variables set previously. The variables within the loop are stored in the #LV# variable.

The example shows that for every variable a line is written, containing:

`Counter → TAB → Loop variable → TAB→"Test "→Counter→Carriage return/Linefeed`

The tabsstops are at position 4 and 15, so the text after a tab begins at 5 and 16.

Example 6: "GetPart & ProgressBar"

These examples show the usage of a Progress Bar and the GetPart functionality.

The GetPart variable returns the specific text on a location in the input.

```
'Use this string as testinput
[INPUT]AAA BBB CCC DDDD
AABBCCDDEE     FFFF
AAABBB CCC DDDD[/INPUT]

'Now provide the header and input message as output
[OUTPUT]#INPUT#

----------------------------------------
This is a test with GetPart:
----------------------------------------
# -1 ; 6 ; 4 #               :{GETPART|# -1 ; 6 ; 4 #}
# 0 ; 42 ; 5 #               :{GETPART|# 0 ; 42 ; 5 #}
# 2  ; 6 ; 3 ; Default #     :{GETPART|# 2 ; 6 ; 3 ; Default #}
# 1 ; 10 ; 4 ; Default #     :{GETPART|# 1 ; 10 ; 4 ; Default #}
# 0 ; 30 ; 3 ; Default #     :{GETPART|# 0 ; 30 ; 3 ; Default #}
# 2 ; 12 ; 3 ; Default #     :{GETPART|# 2 ; 12 ; 3 ; Default #}
[/OUTPUT]
```

The variable inputs are Line, Position, Length and optional a default text when the trimmed return value is empty. If the provide Line is 0 or less, then the absolute position is used. If a line is 1 or more, then that the column is used as position for that specific line. With version 2.0, position indicators are provided with the logging tool for easy use.

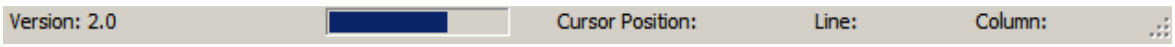
```
AAA BBB CCC DDDD
AABBCCDDEE     FFFF
AAABBB CCC DDDD

----------------------------------------
This is a test with GetPart:
----------------------------------------
# -1 ; 6 ; 4 #               :BB C
# 0 ; 42 ; 5 #               :ABBB
# 2  ; 6 ; 3 ; Default #     :CDD
# 1 ; 10 ; 4 ; Default #     :CC D
# 0 ; 30 ; 3 ; Default #     :Default
# 2 ; 12 ; 3 ; Default #     :Default
```

Also an optional progressbar is added which can be controlled with the following commands:

```
[PROGRESSMIN]0[/PROGRESSMIN]
[PROGRESSMAX]100[/PROGRESSMAX]
[PROGRESSBAR]50[/PROGRESSBAR]
[PROGRESSADD]10[/PROGRESSADD]
```

ProgressMin is the minimum value (should be 0 or higher) and is per default 0.
ProgressMax is the maximum value and is per default 100.
ProgressBar sets the value of the progressbar. If less then 0, the Progressbar will not be shown.
ProgressAdd adds the value to the already existing value



TIPS

- When creating an action file, make it step by step. Make sure that every command does the intention.
- When you want to make extensive reports, do not be afraid to store temporary data in a new file. For example:
    1. Read file
    2. Filter out something
    3. Replace something
    4. Store in File “temp1.txt”
    5. Read file again
    6. Do something else
    7. Create output summary
    8. Store in File “temp2.txt”
    9. Read file “temp1.txt” and store in var A
    10. Read file “temp2.txt” and store in var B
    11. Create output report which includes variables #A# and #B#
- When importing a very large file, the textbox view content is truncated to increase the performance. If you edit the textbox content, then that truncated content will be used…. You should not use the textbox to edit large files anyway…
- For the same reason the program works with two variables “Content” and “Output”
    - When input is placed, the Content variable is filled and so is the Output variable as long as it is within size limits.
    - Changing the Textbox will both change the Content and Output variable.
    - Filtering, replacements, counts, etc. are done on the Content only. Only when all commands are performed, the (changed) Content is copied to the Output.
    - The Command [OUTPUT] will only change the output variable, not the content variable
    - Saving the content: use [SAVECONTENT]
    - Saving the output: use [SAVEOUTPUT]
- All file handling is done in memory. No checking is done on memory size and availability, if the input file is too large, an exception will occur. Close all other applications if more memory is required. On a standard laptop (2GB) I have tested 250MB log files successfully, 600MB was too much…
- Learn Regular Expressions! They provide some powerful tooling.

Variables

Variables can be used with commands. The string is replaced with the actual value before a command is run.

#BASEPATH#	Returns the application folder so that relative paths can be used.
#MM#	Returns the current month in two digits (e.g. '02')
#MMM#	Returns the current month in three char word (e.g. 'Feb')
#MMMM#	Returns the current month as a word (e.g. 'February')
#YYYY#	Returns the current year in 4 digits (e.g. '2013')
#YY#	Returns the current year in 2 digits (e.g. ' 13')
#DD#	Returns the current day in 2 digits (e.g. '05')
#D#	Returns the current day in 1 or 2 digits (e.g. '5' or '31')
#TIME#	Returns the current time in hour:min:sec
#FILENAME#	Returns the filename as selected previous via the command "FILENAME".
#LF#	Returns the char for a linefeed.
#CR#	Returns the char for a carriage return
#CRLF#	Returns the string for a carriage return/linefeed
#A#	Returns the value as previous set via the command "A"
#B#	Returns the value as previous set via the command "B"
#C#	Returns the value as previous set via the command "C"
#LV#	Returns the variable for a Loop
{COUNT|#SearchText#}	Counts the specified text in the content and returns that value
{INPUTBOX|#Description Text#}	Shows an input box with the provided description text and returns the input value
{GETPART|#Line ; Position ; Length ; Optional Default Text#}	Gets the string from the Input for a certain Line and Position and Length. If the Line is 0 or less, the position is the absolute position and not the position within a Line. If the trimmed returned value is empty, an Default Text is returned when provided.
#INPUT#	Works with the A, B, C, MSGOK, MSGOKCANCEL, OUTPUT and ADD2OUTPUT commands. It returns the input
#REGEXRESULT#	Default place where the Regular Expressions functions store there output.

Commands

A	Sets the variable #A#
B	Sets the variable #B#
C	Sets the variable #C#
FILETYPE	Sets the default file type in the file dialogs. Options are "TXT" (default), "XML", "LOG". Other values will refer to "All files".
FILEINITDIR	Sets the initial directory for the file dialog.
FILEINITNAME	Sets the initial filename for the file dialog.
FILENAME	Opens a file dialog with a specified title. When cancel is pressed, it will cancel the rest of the batch file. When the command input includes the word "SAVE", a save dialog is used instead of an open dialog. The provided filename is stored in #FILENAME# Examples: [FILENAME]Save the output?[/FILENAME] opens a Save dialog [FILENAME]Output Safe?[/FILENAME] opens an Open dialog
LOADINPUT	Loads the provided file in content
INPUT	Stores provided text in content
FILTERSTART	Uses the provided text as the start for filtering
FILTEREND	Uses the provided text as the end for filtering
FILTERSEPARATOR	Uses the provided text as a separator between filtered text
FILTEROUT	Searches for text beginning with Filterstart and ending with Filterend and removes that text from the content The variable is either empty or a number. Suppose 5 parts are found, normally all 5 would be deleted. The number will only delete that part (or the last part when the number is greater than the found parts).
FILTER	Searches for text beginning with Filterstart and ending with Filterend and keeps only that text from the content. The variable is either empty or a number. Suppose 5 parts are found, normally all 5 would be shown (with the separator in between). The number will only return that part (or the last part when the number is greater than the found parts).
MSGOK	Shows a message box with the provided text
MSGOKCANCEL	Shows a message box with the provided text. When cancel is pressed, it will cancel the rest of the batch file.
SEARCHONTEXT	Uses this input as the text to search in the content
REPLACEWITHTEXT	Replaces all "SEARCHONTEXT" in the content with the provided string.
SAVEOUTPUT	Saves the output text to the provided file.
SAVECONTENT	Saves the content text to the provided file.

LOOPVAR	To be used before a LOOP. All commands within a LOOP will be repeated for each variable as defined in LOOPVAR. The values are separated with ";". Within the Loop the values can be obtained with #LV#.
LOOP	Repeats every command within a Loop for all variables as set previously by LOOPVAR.
RUNARGS	Sets arguments (if required) for RUNCOMMAND.
RUNCOMMAND	Starts the provided process. Example: [RUNCOMMAND]D:\Test.XML[/RUNCOMMAND] opens the file in de default windows editor. Another option is to open the xml file with Internet Explorer: [RUNARGS]"D:\Test.XML"[/RUNARGS] [RUNCOMMAND]C:\Program Files\Internet Explorer\iexplore.exe[/RUNCOMMAND]
OUTPUT	Uses the provided text as output.
ADD2OUTPUT	Adds the provided text to the current output.
TABINTERVAL	Sets the tab interval in characters, default is 8.
TABSTOPS	Sets the tab stops (character position), separated by ";"
TAB2SPACES	Changes tabs to spaces.
REGEXGETCOUNT	Returns the amount of matches in REGEXRSULT, based on the provided pattern.
REGEXGETISMATCH	Returns True/False for matches found in REGEXRSULT, based on the provided pattern.
REGEXREPLACEMENT	Sets the replacementstring for the commands REGEXREPLACE and REGEXREPLACE-INPUT.
REGEXREPLACE	Replaces the found matches, based on the provided pattern, with the replacementstring and returns the string in REGEXRSULT.
REGEXREPLACE-INPUT	Replaces the found matches, based on the provided pattern, with the replacementstring and returns the string in INPUT.
REGEXGETRESULT	Returns the first match, based on the provided pattern, of a pattern on the input in REGEXRSULT.
REGEXGETRESULT-INPUT	Returns the first match, based on the provided pattern, of a pattern on the input in INPUT.
REGEXSEPERATOR	Sets the seperator string for the REGEXGETRESULTS and REGEXGETRESULTS-INPUT commands.
REGEXGETRESULTS	Returns all the matches, based on the provided pattern, on the input in REGEXRSULT. The matches are seperated from each other by the sepererator string.
REGEXGETRESULTS-INPUT	Returns all the matches, based on the provided pattern, on the input in INPUT. The matches are seperated from each other by the sepererator string.

REGEXOPTIONS	Sets the value for the settings on which Regex works. Check internet (google on 'vb.net RegexOptions Enumeration') for more information on the options. Values: 0: None 1: Ignorecase 2: Multiline 4: ExplicitCapture 8: Compiled 16: Singleline 32: IgnorePatternWhitespace 64: RightToLeft 128: NOT TO BE USED 256: ECMAScript 512: CultureInvariant
PROGRESSMIN	Sets the minimum value for a progressbar. Minimum value is 0. Default value is 0.
PROGRESSMAX	Sets the maximum value for a progressbar. Value must be above the minimum value. Default value is 100.
PROGRESSBAR	Sets the value of the progressbar. If the value is less then 0, the progressbar will be hidden.
PROGRESSADD	Adds the provided value to the already existing value of the progressbar.
RUNTOOLFILE	Reserved for future use with *.CMD Files.
RUNTOOL	Reserved for future use with *.CMD Files.

Commands between [REM] and [/REM] are disregarded.