LogFouineur

V1.x

Installation Guide

User Guide

REF : LogFouineur/JLP/2018/04

Author : Jean-Louis PASTUREL

Description of Successive Versions

VERSION	DATES	State	Author
V1.0	31/April/2018	Initialisation Version	JL PASTUREL

Table des matières

Advertisement :
English is not my native language, so in the document, you will find a lot of syntax and grammar mistakes, awkwardness, difficulties to understand some sentences …
Please let me know at jean-louis.pasturel-wrong-reply@orange.fr
(Remove -wrong-reply to the mail address)

1 Introduction

LogFouineur etymology : “une fouine” in French is a weasel

“fouiner” means : to nose around/about , to snoop .

1.1 Context

Important :
To use this tool, you must know basics on Pattern Matching with regular expression (Perl regex for example). At the end of the document, I give some examples of regex that are currently used in **LogFouineur**.
The general mechanism used in this tool is to parse in 2 phases :

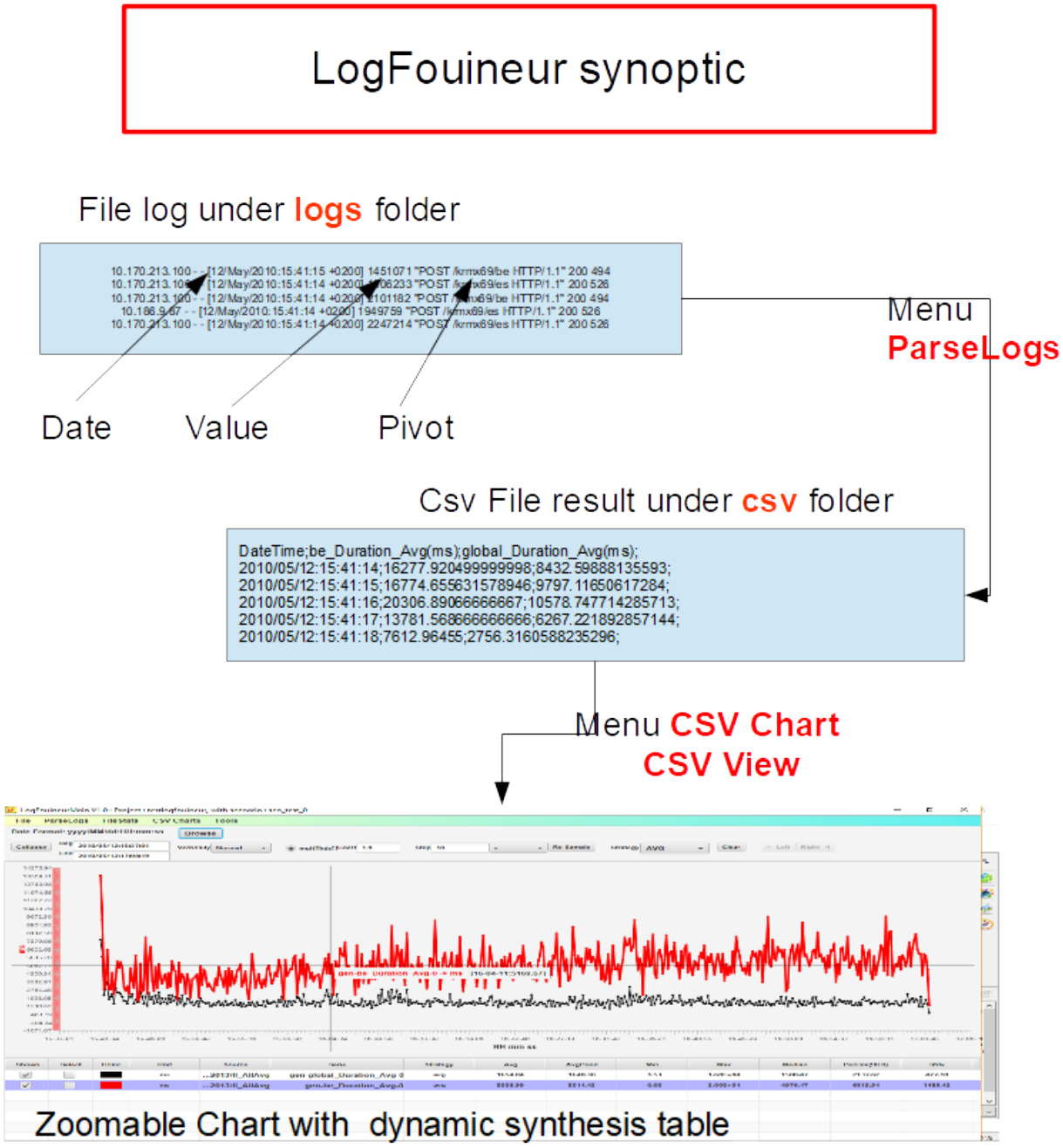
- First phase : first regex extracts from a source that contains the interesting information in a result
- Second phase, **if necessary** : second regex extracts, from precedent result, the final information

This mechanism can handle almost all cases .

The product **LogFouineur** is a kind of workbench that groups several tools :
– parsing dated logs (system logs, Web servers, WAS, application ...) and converting them into csv files
– visualisation of csv files, or direct visualisation with certain types of logs (JVM GC logs for example).
– Utilities tools => date <=> dateInMillis, aggregation of files, regex expression tester ...

The tool **LogFouineur 1.0** is developed with Java 9+ Oracle / JFX 9+ Oracle :
LogFouineur Core : Apache 2 License http://www.apache.org/licenses/LICENSE-2.0.html
LMAX Disruptor : Apache 2 License http://www.apache.org/licenses/LICENSE-2.0.html
Java 9+ / JFX 9+ licenses :
http://www.oracle.com/technetwork/java/javase/terms/license/index.html
Important :
The available packaging contains only the libraries that have the following licenses : LGPL, CPL, Apache, BSD

2 Synoptic


LogFouineur synoptic
File log under logs folder
10.170.213.100 - - [12/May/2010:15:41:15 +0200] 1451071 "POST /krmx69/be HTTP/1.1" 200 494
10.170.213.100 - - [12/May/2010:15:41:14 +0200] 1106233 "POST /krmx69/es HTTP/1.1" 200 526
10.170.213.100 - - [12/May/2010:15:41:14 +0200] 2101182 "POST /krmx69/be HTTP/1.1" 200 494
10.186.9.67 - - [12/May/2010:15:41:14 +0200] 1949759 "POST /krmx69/es HTTP/1.1" 200 526
10.170.213.100 - - [12/May/2010:15:41:14 +0200] 2247214 "POST /krmx69/es HTTP/1.1" 200 526
Menu ParseLogs
Date
Value
Pivot
Csv File result under csv folder
DateTime;be_Duration_Avg(ms);global_Duration_Avg(ms);
2010/05/12:15:41:14;16277.9204999999998;8432.59888135593;
2010/05/12:15:41:15;16774.655631578946;9797.11650617284;
2010/05/12:15:41:16;20306.89066666667;10578.747714285713;
2010/05/12:15:41:17;13781.568666666666;6267.221892857144;
2010/05/12:15:41:18;7612.96455;2756.3160588235296;
Menu CSV Chart
CSV View
Zoomable Chart with dynamic synthesis table

3 Installation

3.1 Packaging

Le packaging is done in a form of zip file **LogFouineur.zip** which the root is **logfouineur .**

3.2 Installation of LogFouineur

3.2.1 Requirements

An **Oracle JDK /Java FX 9.0+** must be installed on your desktop. (not tested with OpenJDK and OpenJFX)

3.2.2 Create a deployment directory

For all the document, we suppose that you use a Windows System (there is no difficulty to adapt for a Linux box) and that the installation directory is **c:\opt** .
It **is not mandatory** to create a directory **logfouineur**.

3.2.3 De-compaction

After having downloaded **LogFouineur.zip** in **c:\opt** le de-compact-it in this directory.

3.2.4 Configuration

The configuration is set in the start script of **LogFouineur** (it may be change with new version, but the principle will be conserved) :
File **c:\opt\logfouineur\scripts\launch.cmd**

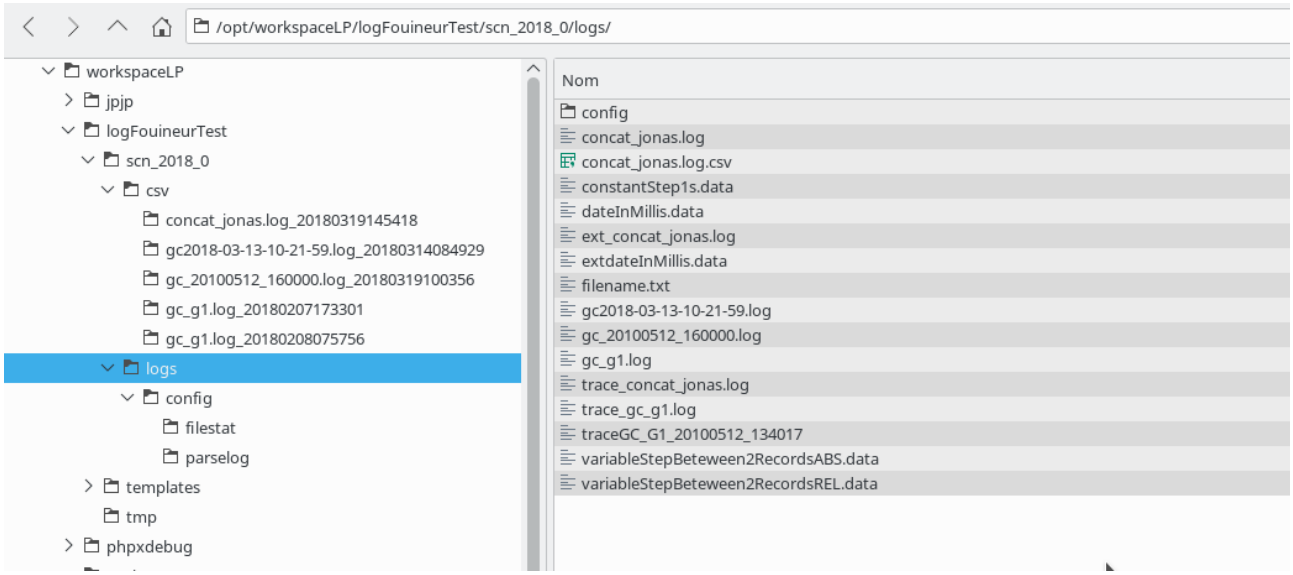
```
set JAVA_HOME=C\Program Files\Java\jdk-9.0.4
set root=C:\opt\logfouineur
set workspace=C:\opt\workspaceLP
"%JAVA_HOME%\bin\java" -Xms1024M -Xmx1024M -Droot=%root% -Dworkspace=%workspace%
--module-path %root%\libs; %root%\libExt -m
org.jlp.logfouineur/org.jlp.logfouineur.ui.LogFouineurMain
```

At the beginning of the file (in bold characters), 3 environment variables must be set, according to your installation.
The Window Environment Variable PATH has to make reachable the executable **java** of the JVM.
Important : The **manual creation** of the directory pointed by **%workspace%** is **mandatory.**
We can after create a link **Windows** on the desktop to launch **LogFouineur** by left click on the mouse**.**

4 User Guide

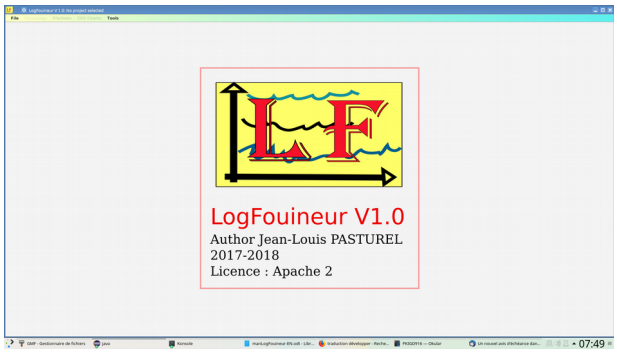
For every feature of **LogFouineur**, we are going to describe the threads of the screens and the parameters to set in the corresponding screens.
The screen-shots shown in this document correspond with the Version 1.x of **LogFouineur** .
The general principle is to organize the **%workspace%** directory by project and by sub-folders.
The sub-folders are the scenarios. These last sub-folders may be created by date, by type of scenario or other sorting keys.
When the scenario sub-folders is created, it is automatically prefixed by the variable (if not yet correctly prefixed) : **logfouineur.prefixscenario** initialised in the file :
config/logfouineur.properties

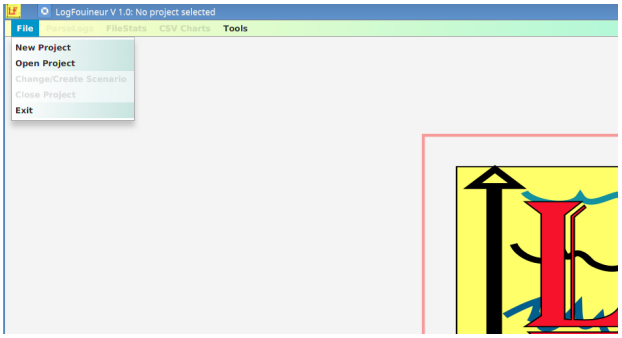
For the rest of the document, these sub-folders will be named “scenario folder”.A typical tree in the workspace folder looks like this:



In the **logs** folder, there are the original access-logs or application logs files and in the **csv** folder, the csv files generated after treatment of logs files.

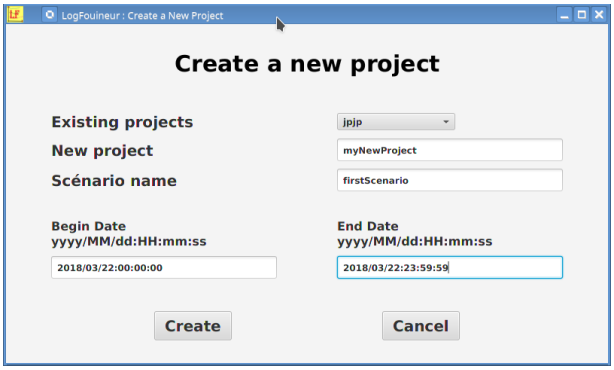
4.1 Menu "Files"





4.1.1 New Project

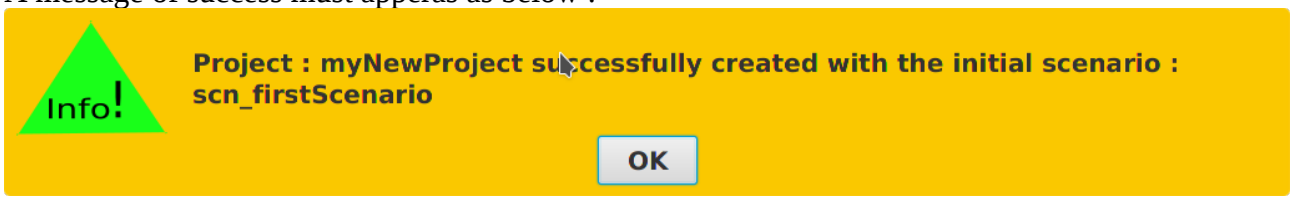
The sub-menu **New Project** allows to create a new project, which name is set in the Dialog Box below:



Fill :

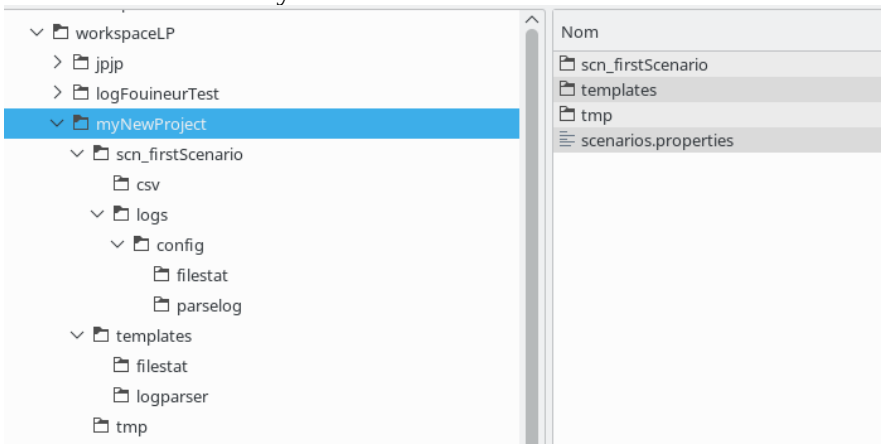
- the name of New Project,
- the name of the first scénario of the project.
- the begin and end date of the first scenario in respect of the date format.

Click Create and a new directory **c:\opt\workspaceLP\myNewProject** is created if **%workspace %** is set to **c:\opt\workspaceLP**
A message of success must apperas as below :



The prefix scn_ of the scenario is automaticly added.
This operation must be the first operation for a new project.

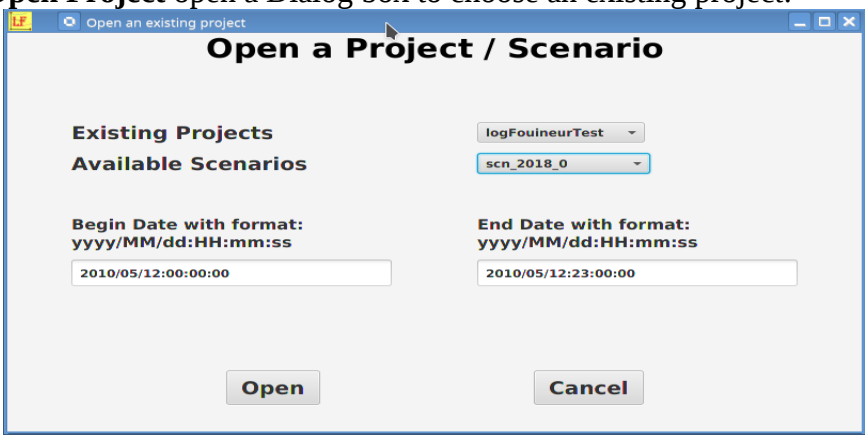
The arborescence may looks like :



In **./myNewProject/scn_firstScenario/logs** directory, we will put the log files that concern the period of the defined scenario .

4.1.2 Open Project

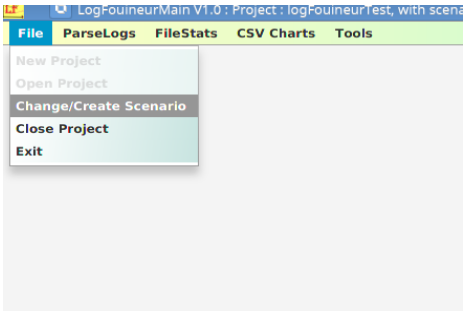
The sub-menu **Open Project** open a Dialog box to choose an existing project:

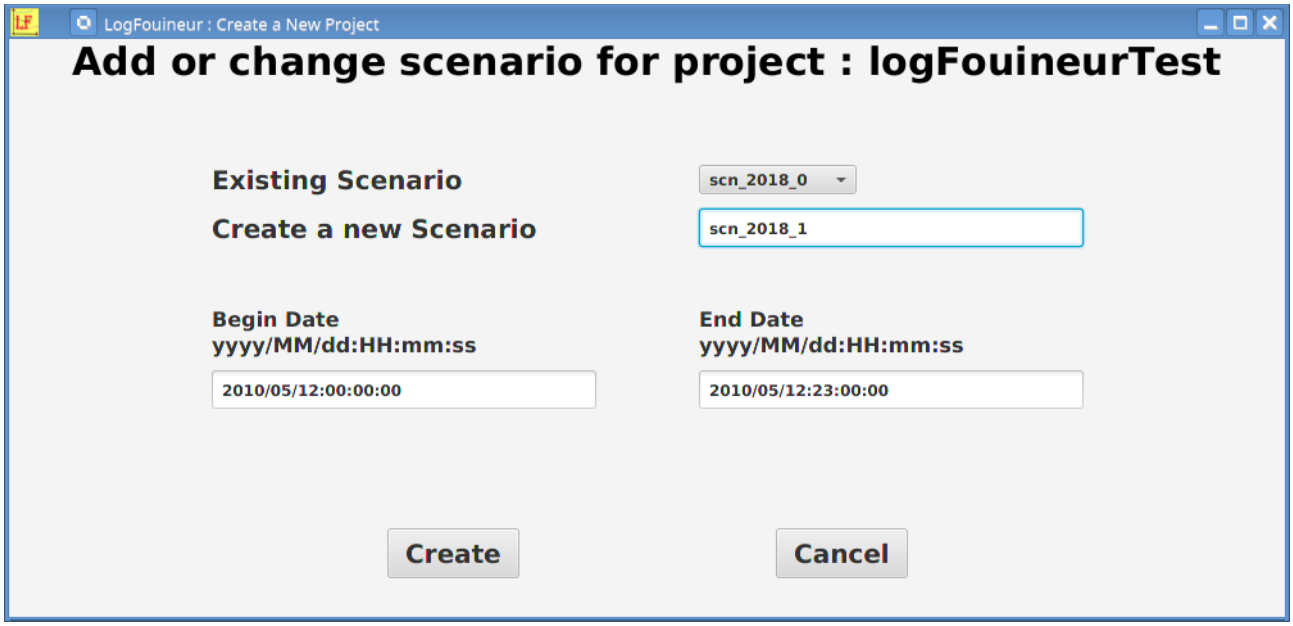


We can choose for example the existing project **logFouineurTest** with the scenario **scn_2018_0**
The textFields for begin / end date are filled in regard of the chosen scenario.
The fields can be updated. It increases performance when parsing large files, which the logs are not all useful.
Note : By creating or opening a project, we enable menus on the MenuBar. Closing the project disables these same menus.

4.1.3 Create/Change Scenario

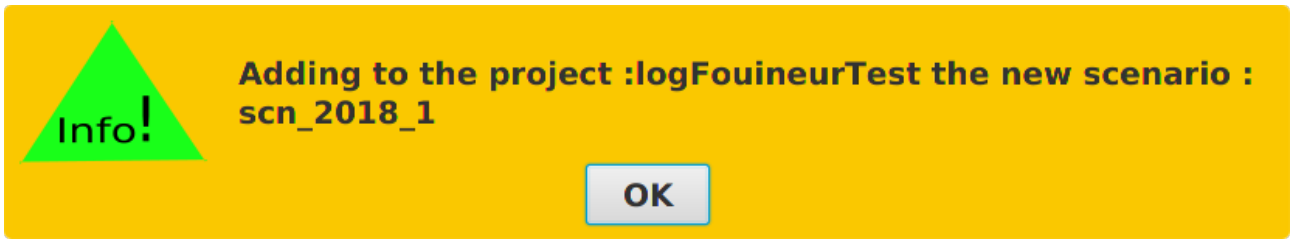
The sub-menu **Create/Change Scenario** allows to structure the directory of the project by changing or adding new scenarios in scenarios or date of tests.
The scenario or test name is what you type in the Dialog Box below.
Remenber : when you create a New Project, you must also create a first scenario.



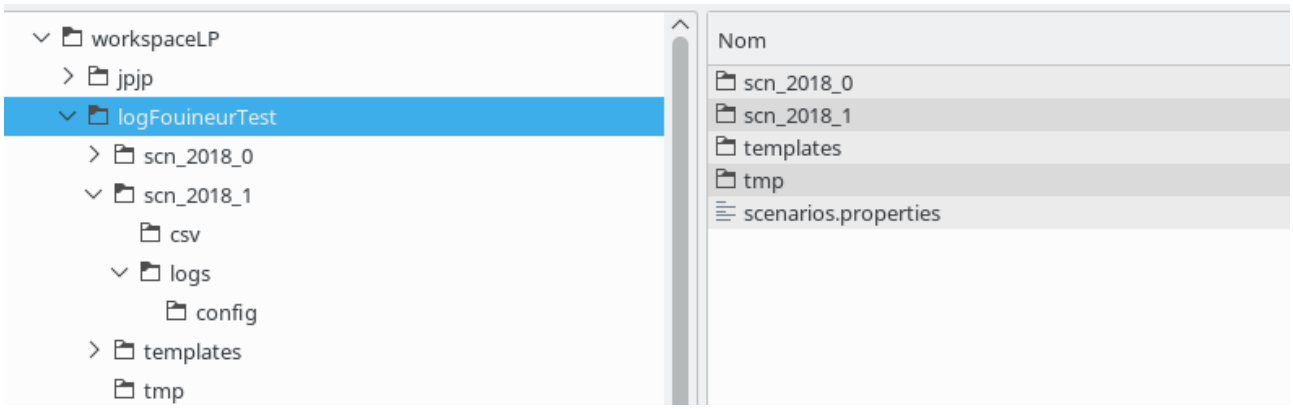


You can either change to existing scenario using the combo-box or create a new scenario by filling the textField Create a new scenario.

In case of new scenarion the new directory is created as shown below :
%workspace%\logFouineurTest\scn_2018_1
The begin / end date for the new scenario must be set in the correspondent TextFields.
For an existing scenario, the fields are automatically filled.
The date fields can be updated. It increases performance when parsing large files, which the the logs are not all useful.



The message indicates the success of creation of a new scenario.



In the sub-directory **logs** of the new folder (**scn_2018_1**), we can put all logs files corresponding to the new scenario .

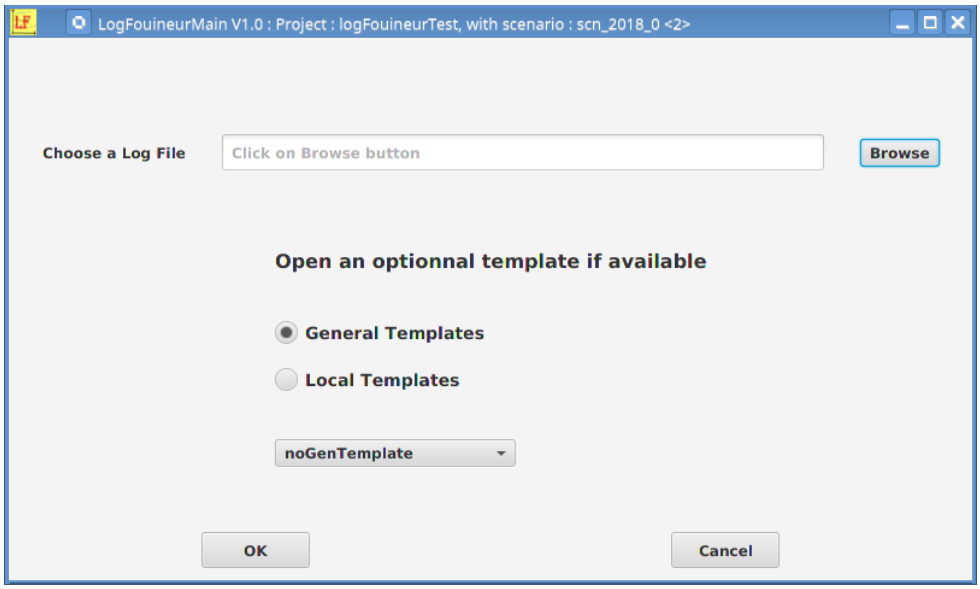
Note : to remove a project, remove the directory of the project .
To remove a scenario from a project, remove the folder of the scenario **and** remove all reference of the scenario in the file **%workspace%/%project%/scenarios.properties** :

```
#Creation of a new scenario
#Thu Mar 22 15:08:06 CET 2018
scn_2018_1.dateEnd=2010/05/12\:23\:00\:00
scn_2018_0.dateEnd=2010/05/12\:23\:00\:00
listScenarios=scn_2018_0 scn_2018_1
scn_2018_1.dateBegin=2010/05/12\:00\:00\:00
scn_2018_0.dateBegin=2010/05/12\:00\:00\:00
```

Deleting scenario **scn_2018_1** means deleting all red/bold characters in scenarios.properties as shown above.

4.2 Menu "ParseLogs"

With this menu, we transform a log file to a csv file that can be graphed with Java Fx (Menu CSV Charts)
The Menu **ParseLogs** (in fact a Button) open a Dialog as shown below :

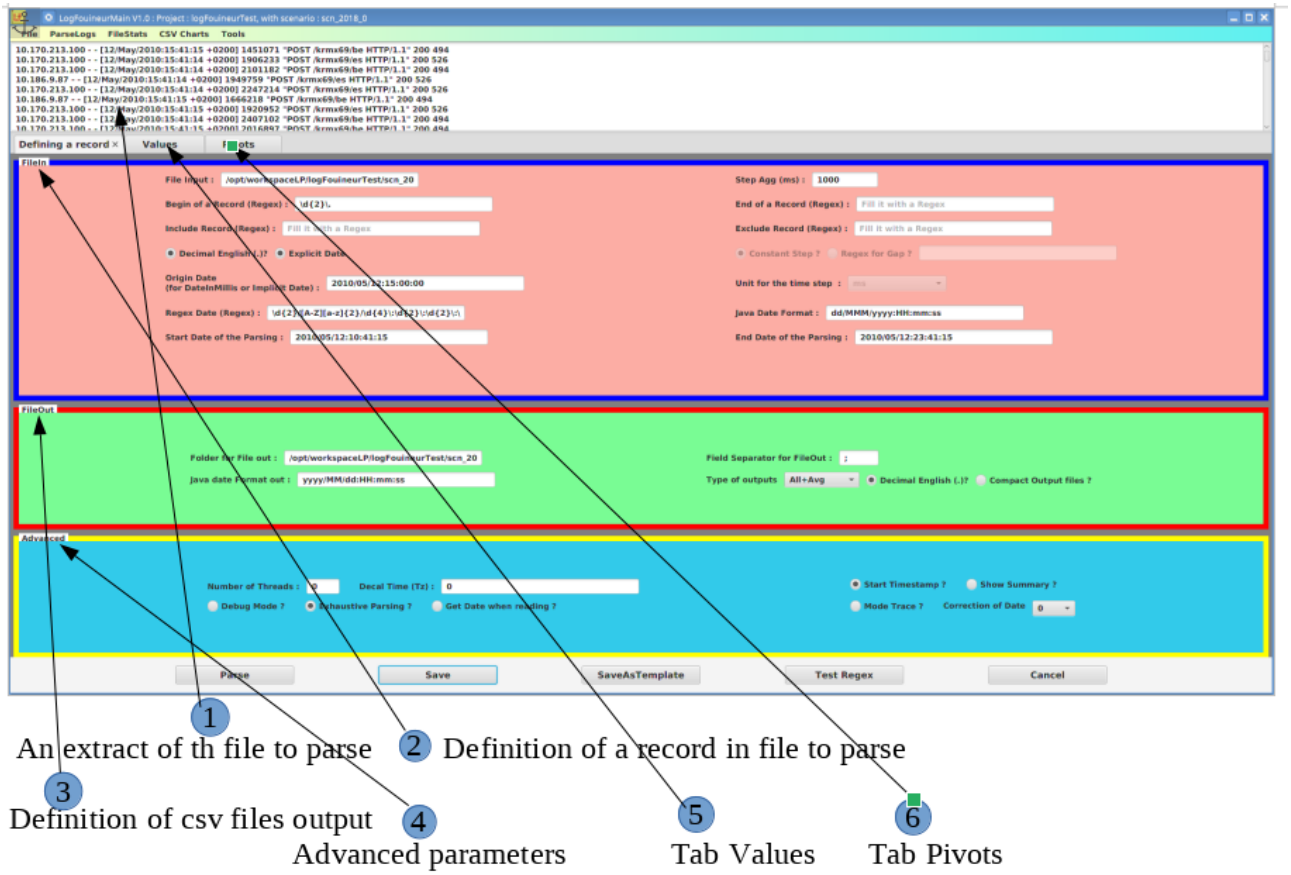


First action to do, click on Browse to choose the file to parse, a FileChooser will open in the directory **%workspace/<Project><scenario>/logs**
You have to choose a file.

After wrads there are 3 possibilities to configure before parsing :

- with no template, (ex-nilho) => ComboBox with **noGenTemplate** or **noLocTemplate**.
- with a local template, only valid for the current project. You must choose a local template in the the ComboBox if any.
- with a general template, shared by all projects General Template selected and a template chosen in the ComboBox

The screens below, show the more general configuration with no template after filling all.
After clicking on OK Button, the screen below appears :



Parsing Start Date / Parsing End Date : set a beginning and an end date for parsing. It is useful when log files are large. See the table below for the format.

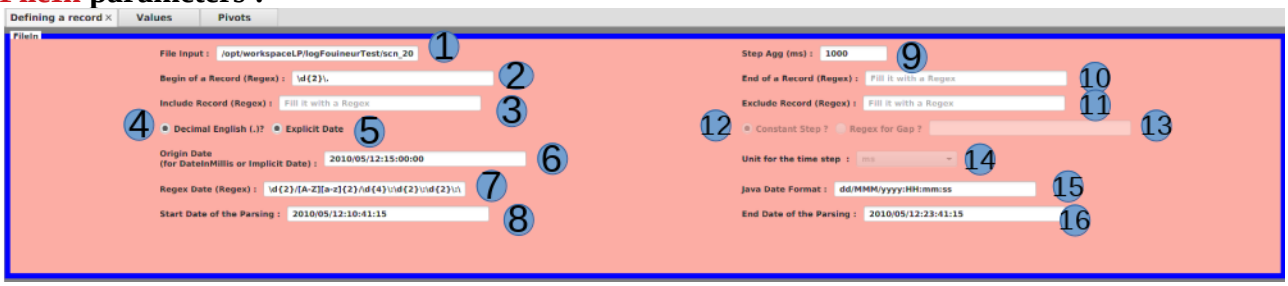
In the TextArea at the top of the Windows, there are the first 20 lines of the log file to parse

In the bottom part, there is a Tabbed Pane with 3 tabs

The first tab has the functions :

- to define what is a record in the source file (Beginning/End of record, Locale …)
- to retrieve the date of the record (explicitly or implicitly).
- to define the format of the output CSV file(Date format, separator)
- Automatic graph or not after parsing

FileIn parameters :



Parameters	Signification	Example
1 → File input	Full path of the source file**.** Automatically filled by **LogFouineur** when chosen	
2 → Begin of a Record (Regex)	Regular expression (Perl Regex) to find the beginning of the record. **Mandatory.**	**\d{4}-\d{2}-\d{2}**
3 → Include Record (Regex)	Regular perl expression that allows to include all records that are interesting.	Exclude feature has priority over Include feature.
4 → Decimal English	Choice of the Locale for File In (for Date and representation of Double)	Depends on file log format.
5 → Explicit Date	If selected, means that every record of the file contains an explicit date. If not (Implicit) the date must be computed.	Selected (as possible). Certainly the most current case of logs.
6 → Origin Date (DateInMillis or Implicit case)	Definition of a beginning date with the Java DataFormat defined by the field **Java Format of Date**	**1970/01/01:00:00:00** (0 of TimeStamp) Used when **Explicit Date** is unchecked, if so an increment must be defined. or when **Regex of Date** is set **to dateInMillis** with explicit dates.
7 → Regex Date (Regex)	Regular expression (perl regex) to catch the date. The date must be in the first selected group. In case of date in timestamp in millis, the pattern of java date format is set to **dateInMillis**.	**^([^:]+[^]+)**
8 → Start Date of the Parsing	The date for the beginning of parsing	Format must be : **yyyy/MM/dd:HH:mm:ss**
9 → Step Agg (ms)	Measurement period of result aggregation in milliseconds.	1000
10 → End of a Record (Regex)	Regular perl expression to catch the end of a record. If this field is empty, it signifies that each line is a record.	If there is no evident end of record pattern, you can set it the same pattern as the pattern of the beginning of the record. LogFouineur takes care not to skip any record.
11 → Exclude Record (Regex)	Regular perl expression that allows to exclude some records that are not interesting.	
12 → Constant Step	Used when **Explicit Date** is unselected.	Selected / unselected

Parameters	Signification	Example
or **Regex for step ?** **(**Exclusive radio Buttons **)**		
13 → Values for choise 12	Define the increment relative to the parameter : - Constant Step Constant step between 2 records - Regex Step , extract in the record can be defined as **ABS**lolute or **REL**ative. The unit is given by **14**	**1000** **(?![^=]=)\d+ REL** **(?![^=]=)\d+ ABS** **(*)** See more explanation after this table.
14 → Unit for Time Step	Unit of the increment. Possible values are **s,ms,micros,nanos, hour, ...**	**ms**
15 → Java Date Format	Select the Java DateFormat of the records of the log file.	**[dd/MMM/yyyy:HH:mm:ss Z]** or **dateInMillis or dateInMillis;<mult>** mult=1000 if date is in seconds The dateInMillis format, compute date, starting with the beginning date filled in the **Start Date (dateInMillis or Implicit case)** parameter
16 → End Date of the Parsing	The date for the end of parsing	Format must be : **yyyy/MM/dd:HH:mm:ss**

FileOut parameters :

Parameters	Signification	Example
1 → Folder for Files Out	Full path of the target directory files**.** Automatically filled by **LogFouineur** when chosen	
2 → Java Date Format out	Format DateFormat Java of the csv generated file	**yyyy/MM/dd:HH:mm:ss**
3 → Field separator for File Out	Separator for the output CSV file	;
4 → Type of Output	You can choose, what csv files are generated. (AVG, Max, Min … are the strategy of compute of the values contained in a period of Aggregation)	**All+Avg** **Avg** **All+Max** **...**
5→ Decimal English	Choice of the Locale for File In (for Date and representation of Double)	Selected (as possible) Recommanded choise
6 → Compacted OutPut Files ?	If selected, a csv file is generated for each **Pivot** (see tab **Pivots**)	

Advanced parameters :

Parameters	Signification	Example
1 → Numbers of Threads	Number of Threads (Disruptor/ Handler) to treat the log file	**0 → CPU dependend (Recommanded Value)**
2 → Decal TimeZone (TZ)	If the format date doesn't contains the TZ zone, you can correct it by adding or subtract **ms**. It can be also used to correct a clock fault in the logs	**+7200000**
3 → Debug Mode	For debugging with a little extract of the log file. There is a log trace in the directory <logfouineur_Home>/logs	**Checked/Unchecked**
4 → Exhaustive Parsing	Useful when 2 pivots have inclusive regex for example GET\s/MyHome/ GET\s/ If not checked, the URLs matching the first regex are not parsed by the second regex	**Checked/Unchecked** If you are sure that regex are all not inclusive, unchecking is faster for parsing.
5 → Get Date When Parsing	Interesting with big log files, when the logs have may recors out of the interesting period. When the file match the period, the unchecked value is faster for parsing.	**Checked/Unchecked**
6 → Start TimeStamp	Not Used	For future feature… perhaps !
7 → Show Summary	If checked, at the end of the parsing, the file allAverages.csv is graphed automatically.	**Checked/Unchecked**
8 → Trace Mode	Create a file, in logs folder, with the extracted records. Can be useful, when the records are spaned in multi lines in the log file.	**Checked/Unchecked**
9 → Correction of date	When the first value is expressed as a duration the 3 strategies are : if 0 => no modification of the date if 1 => add the duration to the date if -1 => subtract the duration to the date	Select 0 or 1 or -1

The analysis of access-logs is based on the utilisation of regular expressions that are described in the JDK Javadoc. Look at Pattern class :

https://docs.oracle.com/javase/9/docs/api/index.html?java/util/regex/Pattern.html

(*) The different forms of increment are :

- **<regular expression><space>REL|ABS**
  - Enter here a regular expression to match the increment. The tool test regexp can help you (see later in this document).
  - A space
  - key Word :
    - **REL** for relative step to the precedent record (first record date : Start Date of Parsing + step)
    - **ABS** absolute to the Start Date of the Parsing
  - the unit is given by the parameter **Unit of Step**

If the key word is absent, **REL** is taken by default.

- **<Constant value of the increment>**
  Specify here the constant value of the increment, the unit is given by the parameter **Unit of Step** . A To be be used when there is no explicit date in the access logs and you know the period of generating every record.

Values tab :

The Value tab is divided in 5 columns :

- **1 → Name Value** : in this column you set the name of the value that you want to follow (the name is free)
- **2 → First Regex / Function** : in this column, you set the mean for extracting the value, there are two ways to extract a value :
  - **regex** (the default shown on the screen-shot above). You put a regex to extract the value. If you are not able to extract the value with this first regexp, you can apply a second regexp(**Second Regex / Parameters**) to extract the final value from the first matching.
  - **function=<nameOfClass>**
    - It is an advanced feature, that allows to write a kind of plug-in to **LogFouineur** (useful when the value can't be extracted directly example : sum or difference of 2 values in a record , computing GC throughput...).**(**)** See below the whole explanation of this feature.
- **3 → Second Regex / Parameters**

    - A regex when needed as explained above or parameters when using plugins/function
- **4 → Unit :**
    - the unit of the value extracted.
- **5 → Scale** :
    - allows to change the scale of the value to match the unit chosen.
- **6 → isDuration** : Only one of the value can be tagged as duration. Used to determine how many request has been active in a period (pseudo parallel requests).

(**) key word function :
It is an advanced topic and it needs to know programming in Java language.

We must write a Java **class** with the **plugins** package, which has at least 2 methods according to the Interface given below :

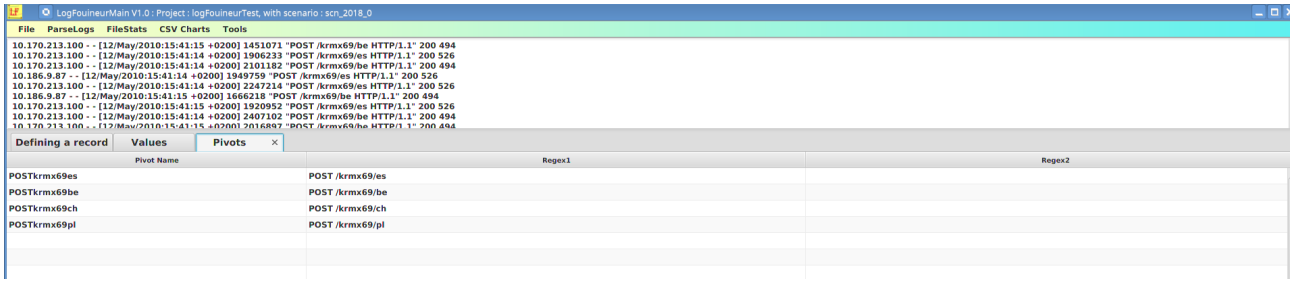
```
package plugins;

public interface IMyPlugins {

	public void initialize(String strRegex2);
	public Double  returnDouble(String line);

}
```

You can add your customs methods, parameters, static parameters to handle counters ...

There are examples in the directory **<LogFouineur_Home>/myPlugins** . The classes (byte-code) must be put in the archive jar **: <LogFouineur_Home_Home>/myPlugins/myPlugins.jar.** This archive jar is also set in the module-path of the script **launch.cmd** (.ksh).

The return value of the method retour is a **Double** (it can be also Double.NaN, this case is treated).
In the column **First Regex / Function ,** the name of the class is set (without the suffix .class) **: function=Compute2Values**
In the field **Second Regex / Parameters**, we can set all the necessary parameters to execute the function. This parameter is a Java Class **String**.The parameters **regex2** are treated by the **initialize** methods according to the needed treatement of the record. The **initialize** method is called at the creation of the instance of the class.
The record / line (Java Class **String**) is passed to the **returnDouble** method.
Tip :
When the name of a class begins by **Mono**, the function must be used in mono-threaded parsing. If one class begins by Mono in the Value Tab, the parsing is automatically mono-threaded (The field Nb of Threads is set automatically to 1)
Look at the Annexe part an example of function.

Pivots Tab :

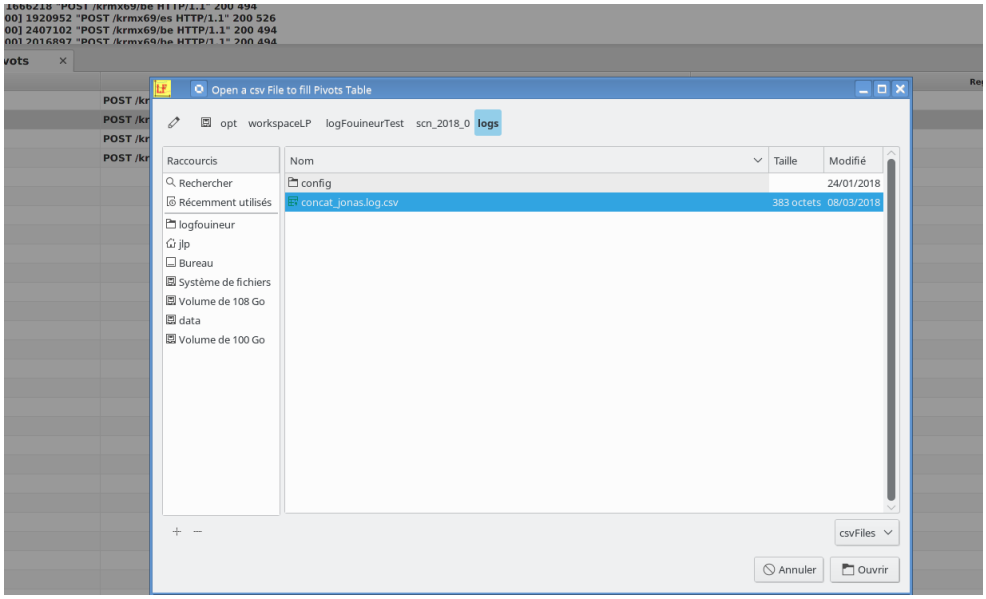


What is a "Pivot" ?
In a record, there are other information, that are not numeric, but useful to sort the values. Example URLs in Apache Access Logs, or other WAS access logs , a **Pivot** can be a specific pattern of URL as shown above.

This tab has 3 columns :

- **Name :**
  - The free name of the Pivot
- **First Regex** : You put a full string value if you can or a regexp to extract the value. If you are not able to extract the value with this first regexp, you can apply a second regex (**Second Regex)**
- **Second Regex** to extract the final Pivot from the first matching.

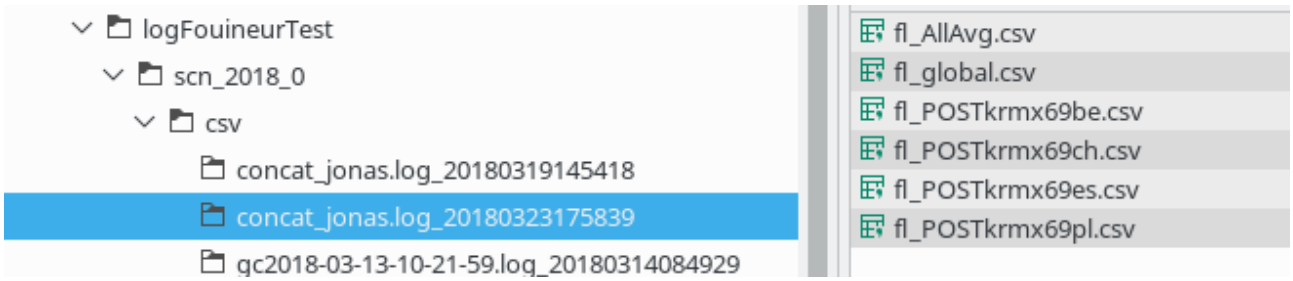
Hint 1 : If beforehand you have parsed this file with **FileStats** (see further in the document) and saved the result table in a csv file in the logs directory, by right clicking in the table Pivot and choosing the csv file saved, the rows of the Pivot tab are automatically filled. After you can suppress, modify, insert other rows.



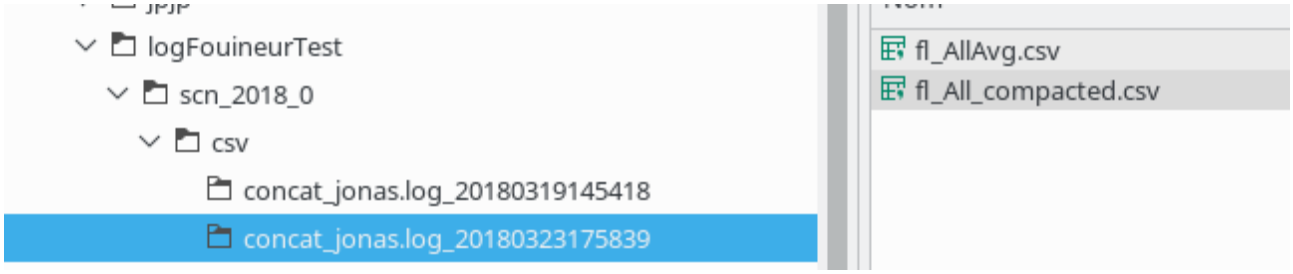
Hint 2 : For the column **First Regex,** if you use full String as possible (without pattern regex as \s\? ...) , the treatment is faster.

Principle of generated files in the csv directory
If you have 1 values in the Tab Value (Duration) and 4 Pivots in the tab Pivot .
Two cases :

not checked.

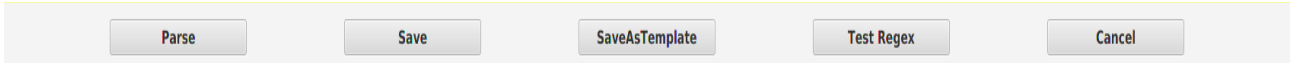


checked



- The compacted mode gives bigger csv files.

Buttons :



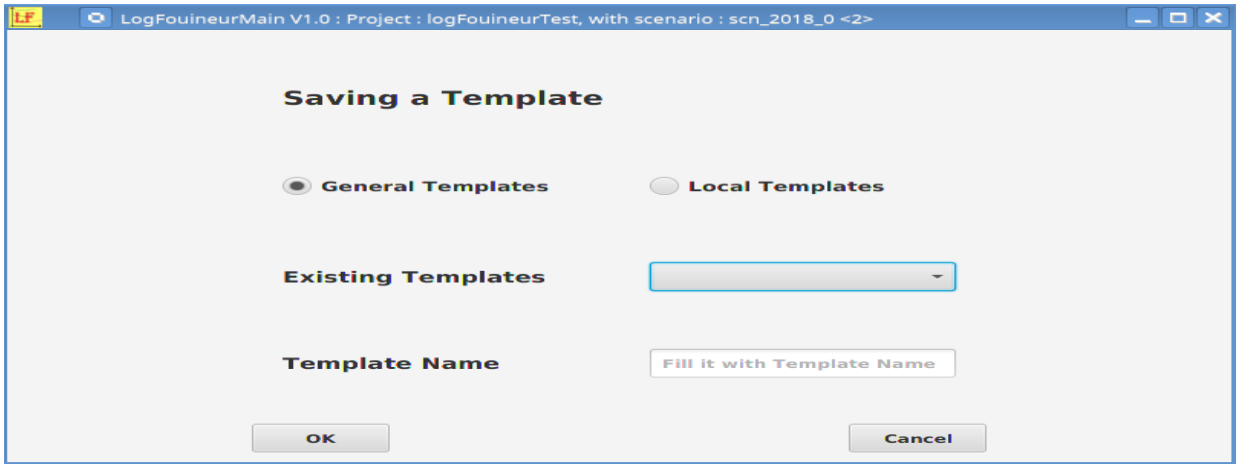
TestRegex:
The button TestRegex allows to test regex in a Dialog box. See the Menu tool for more explanation.

Save :

The button Save saves the configuration and can be be reused for parsing another time

SaveAsTemplate :

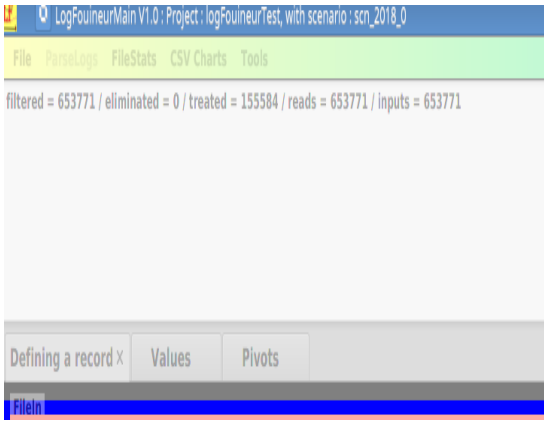
The button **SaveAsTemplate** save the configuration as a template that will be re_used as pattern for the same kind of access log file**.**

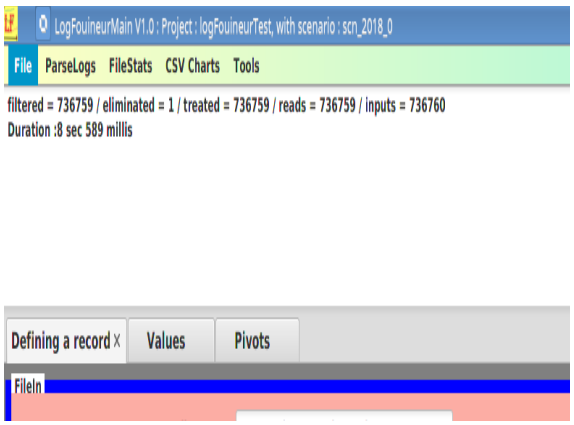


We can save this template on the context of the project (**RadioButton Local Template** checked), or in a General Context (can be used by others projects) when **RadioButton General Template** is checked**.** You give a name, the combo-box lists existing template, to avoid destructions of template. You can replace a template by overwriting it.

Parse Logs :

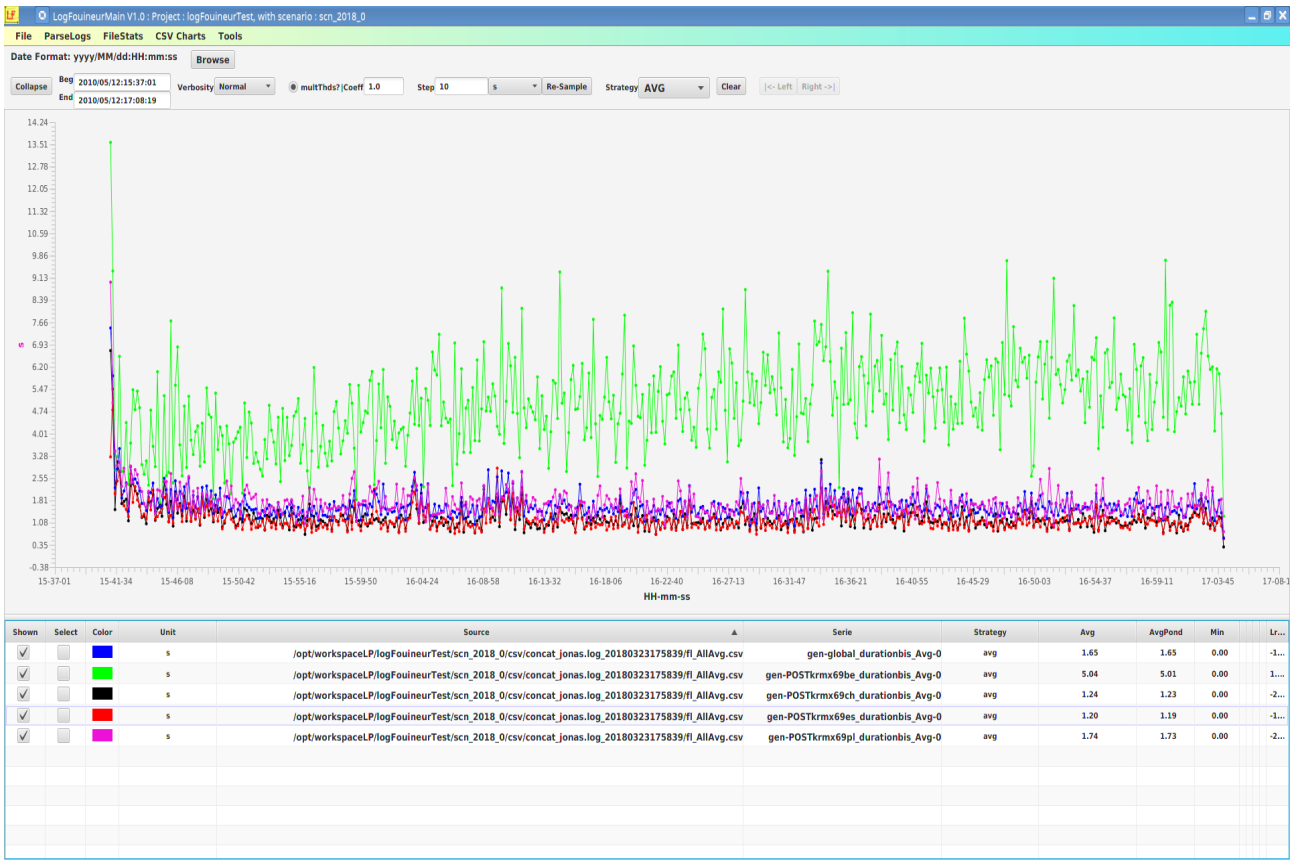
The click on Parse Logs starts the Parsing :





This screen is shown at the end of the parsing, without graphing the **fl_AllAvg.csv** file because the radioButton S**howSummary** was unchecked.

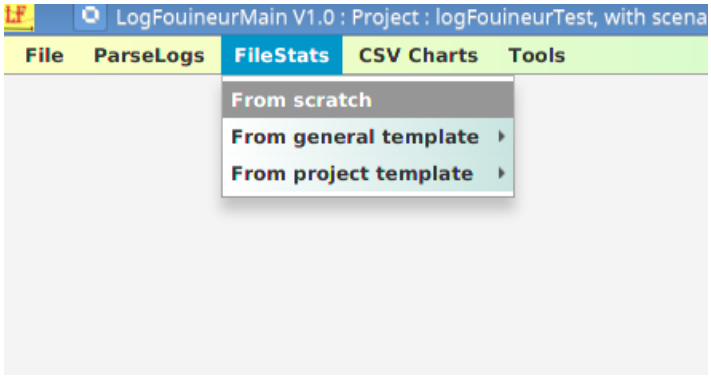
If the RadioButton **ShowSummary after parsing** is checked before launching the parsing, the **fl_AllAvg.csv** is graphed as shown below :



4.3 Menu "FileStats"

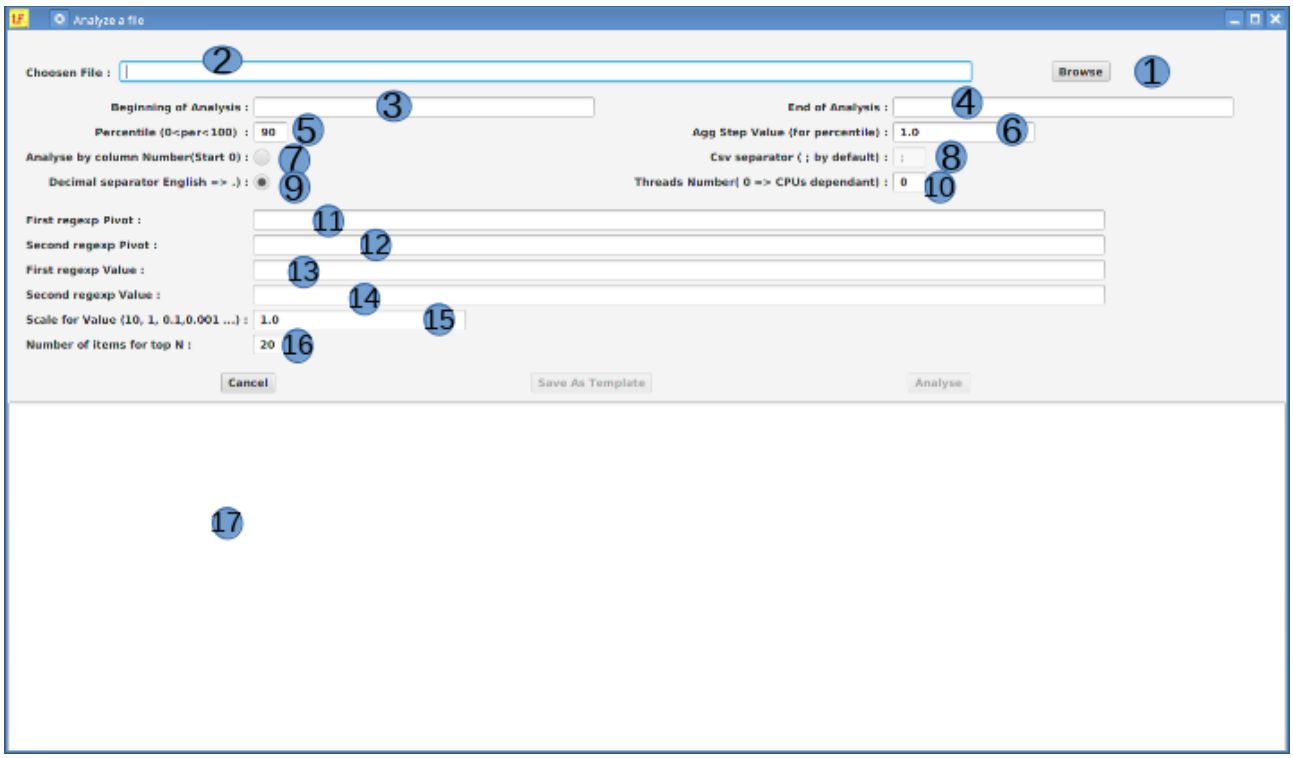
To be used, when the records are mono-line.
Workaround : If records are spanned into several lines, you can first parse the logs with a default duration (regex1 => \d for example et yhe unit , scale doesn't matter) and activating trace mode. You obtain a file with mono-line records.

The feature recognize a certain numbers of date Pattern.If a pattern is not known, this pattern must be added in the file (advanced topic) :
<LogFouineur_Home>/config/logFouineurDates.properties



4.3.1 Sous-Menu "from scratch"

After choosing a log file to parse :
1 → Open a FileChooser
2 → The full path of the choosen file

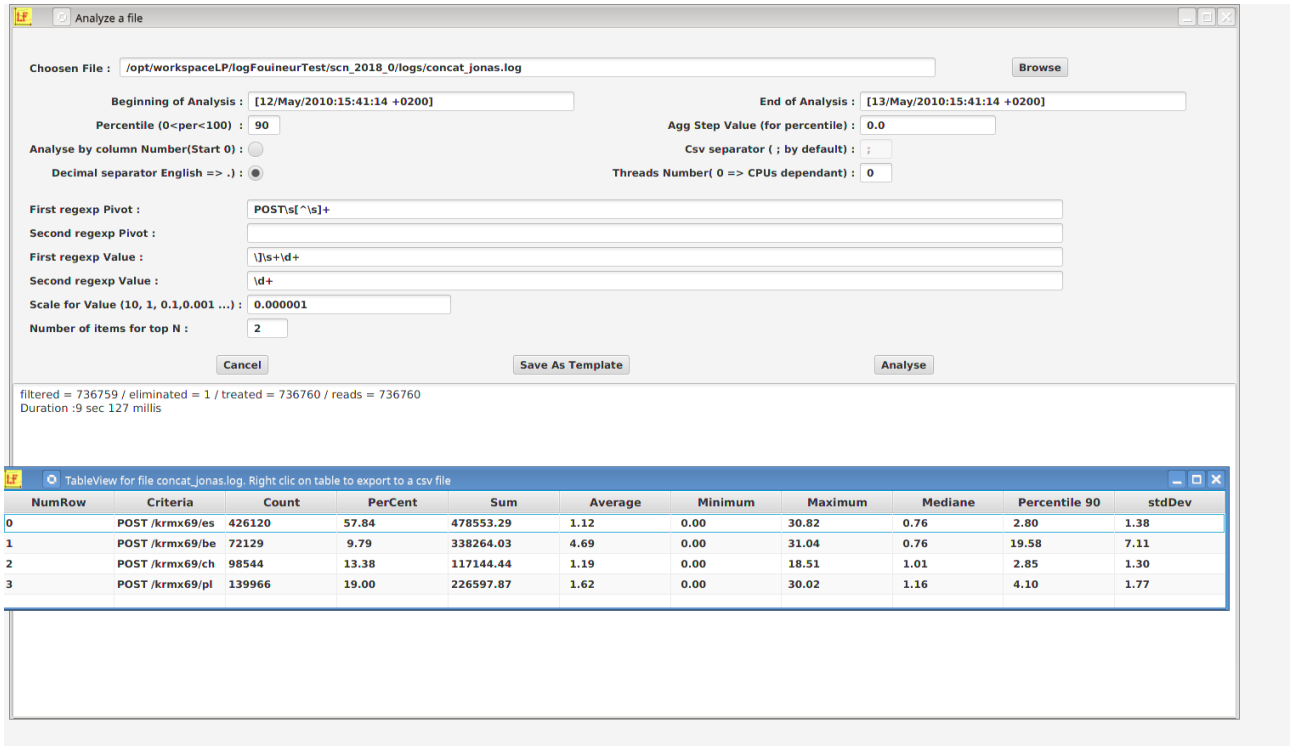


The fields to fill are explained in the table below :

Name of Parameters	Possibles Values	Comments
3→ Beginning Of Analysis	[08/0ct/2010:00:**10:00** +0200]	By default the first date found in the file but can be modified by hand. The format of the date must be respected
4→ End Of Analysis	[26/Jan/2011:**00:30:00** +0100]	By default the last date found in the file but can be modified by hand. The format of the date must be respected
5 → percentile	Number between 0 and 100	The percentile X% : the value which X % of the values are lesser than this value
6 → Agg Step	Number depending on scale value	When there is a huge numbers of values in the source file, a OutOfMemory could be raised during the computing of the Mediane Value and the Percentile value. This parameter allows to aggregate values by step, and so the number of values decreases
7 → Analyse by column number	checked/unchecked	If checked, source file is in csv format.
8 → Csv separator	, or ; or other character	CSV Separator, enabled when Analyse by column number is checked
9 ->Decimal separator English => .?	checked/unchecked	If Checked => English => decimal separator = . If not checked => French => decimal separator = ,
10 → Number of Threads	0 or nb >0	0 → CPU dependant
11 → First regexp for Filter in pivot	Java/perl regex	First Regexp to extract the Pivot
12 → Second regexp for Filter in pivot	Java/perl regex	If necessary, second regexp to extract the Pivot from the first matching
13 → First regexp for Filter in Value	Java/perl regex	First regexp to extract the value
14 → Second regexp for Filter in Value	Java/perl regex	If necessary, second regexp to extract the Value from the first matching
15 → Scale of a value	1, 10, 0.001 ...	To change the scale of the value
16→ Number of items of the topn	10	Defines the Top n that will be shown after the parsing.
17 → Zone of infos		

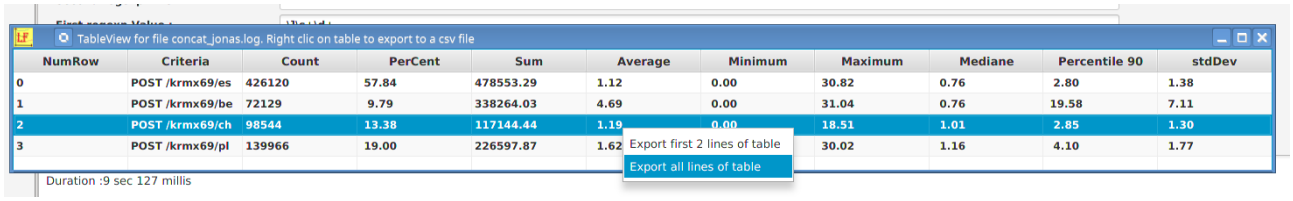
In case of csv file(**Analyse by column number** checked) :
- the field **First regexp for Filter in pivot** is named **index of Column for Pivot (starts to 0)**
- the field **First regexp to extract a Value** is named **index of Column for Value (starts to 0)**

The other fields are unchanged.
After clicking on Analyse button :



By right clicking, in the contain of the table you can

- save the table in a csv file

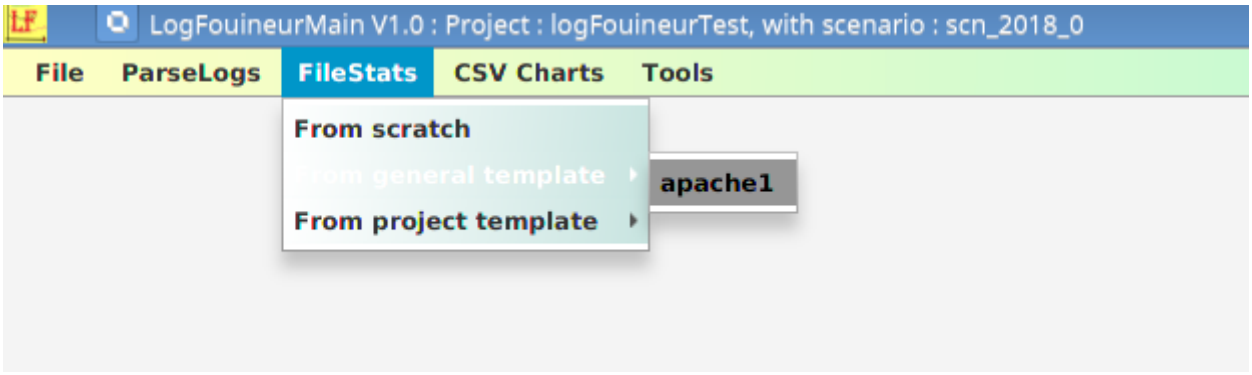


Hint: Saving this file in Csv Format, allows to fill automatically the Pivot Tab of the Menu **ParseLogs (see Menu ParseLogs** more above in the document**)**

Button SaveAsTemplate

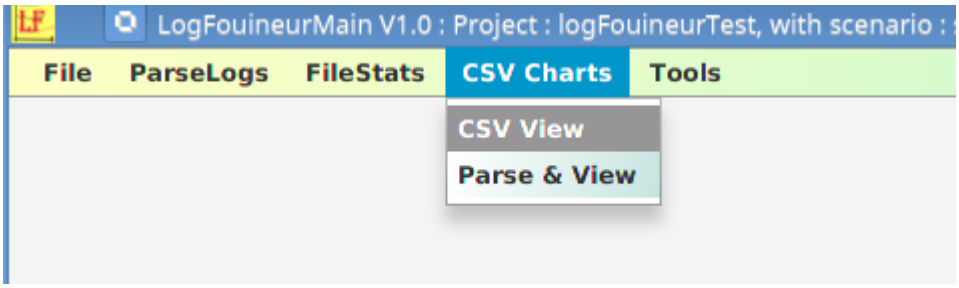
The button **SaveAsTemplate** runs as we have seen above to save a template with **Parselogs**
You can save a General Template shared with all projects or a local template used only by the current project.

4.3.2 Sub-Menu “From general template” and “ From project template ”



Choose a template before opening a log file.
The fields are filled with the values of the template and afterwards choose the file to parse

4.4 Menu "CSV Charts"

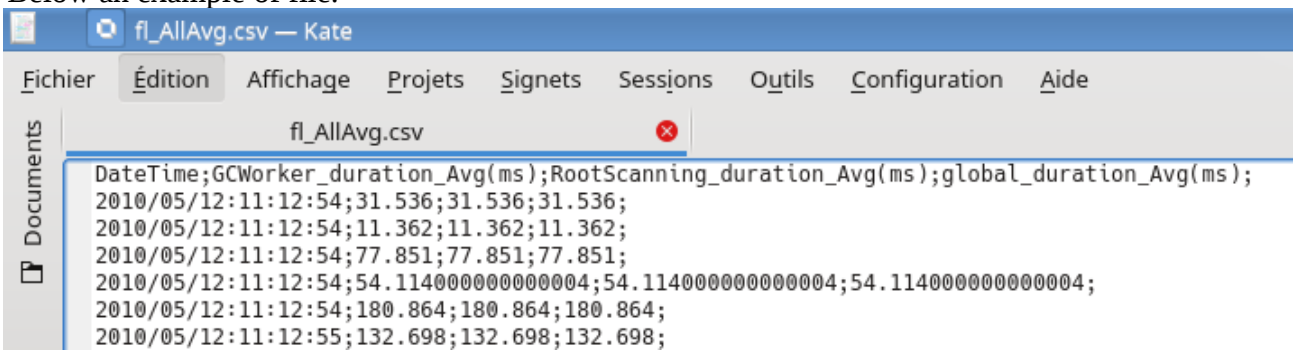


Csv file restrictions :
The only CSV files that CSV View can graph must have the following configuration :

- the first line contains title of columns separated by the csv separator
- the first column must be a date respecting one format described in the file **<LogParser_Home>/config/logFouineurDates.properties. The title must contain the strings date or time.**
- Optionnaly a column may have a title “Pivots” for sorting values .
- The others columns must contains numeric values. This column can be empty (if ; is the csv separator ;; indicates an empty column).

For the values, the title indicates the unit between parenthesis. By default, the unit "**unit**" is taken. Obviously, **LogFouineur** generates correct CSV files.

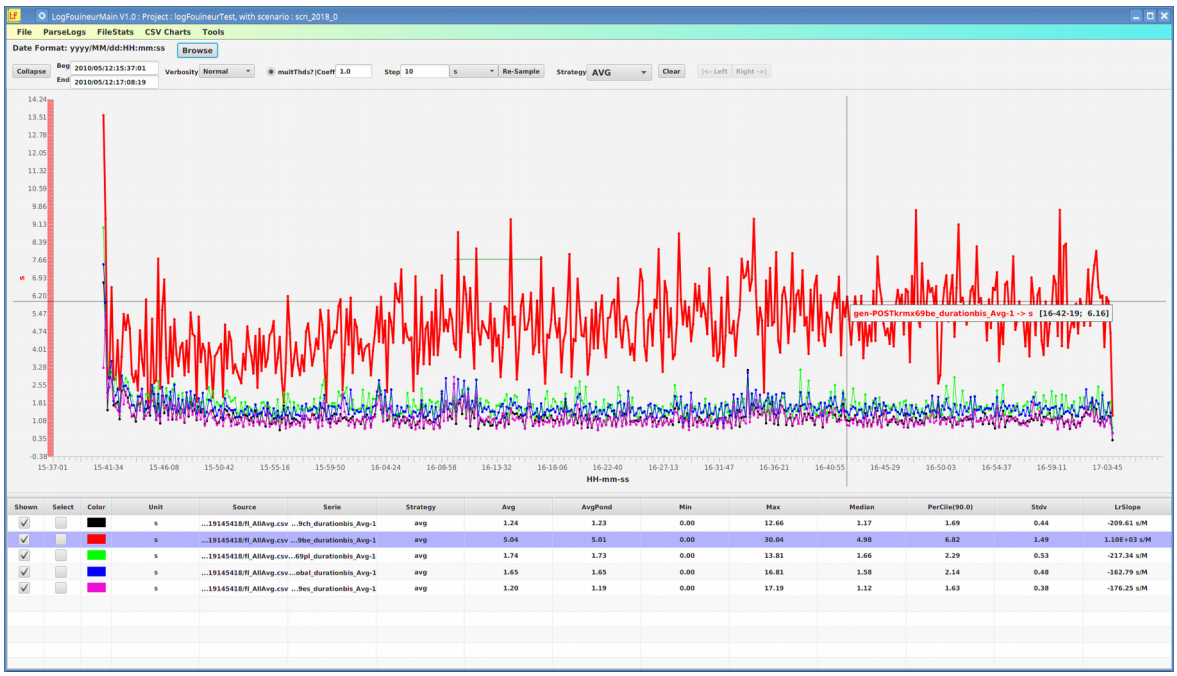
Below an example of file.



4.4.1 Sub-Menus "CSV View"

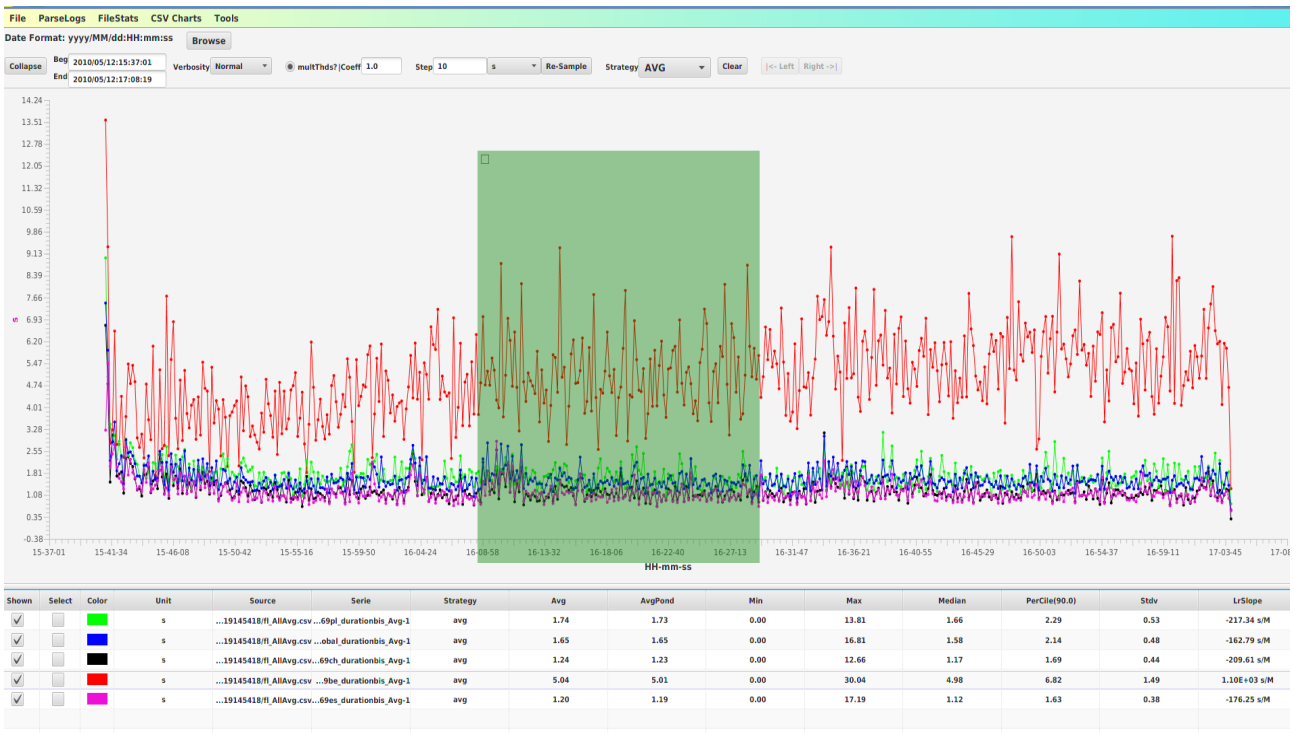
This menu allows to choose csv files, using a FileChooser. You can select multiple files in the same directory.

After choosing a file, in the csv directory of the correspondant scenario, the chart is shown as below :



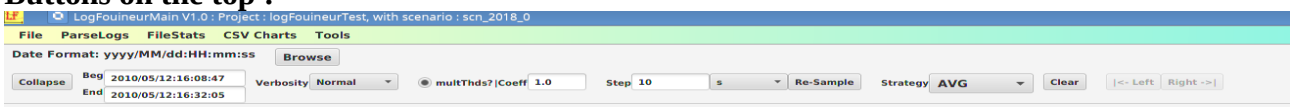
From your System File Explorer, you can also add csv file by DnD from the same or another directory.

The graph must be zoomed by selecting a region
. It can be de-zoomed by dragging a thin rectangle (with no height)



Clicking right on the title of the table allows to add/remove columns.
Clicking right in the contains of the table allows remove selected row series on the table and on the graph.

Buttons on the top :



Collapse/Expand :
Show/hide the table and the graph.
Labels Beg / End :
Shows the time interval of the chart.
Verbosity
To configure the verbosity of the dynamic popup.(Normal, Verbose, Silent)
Radio Button MultiThreads / MonoThread
Parallelism

Step

If pts is chosen in the combobox, indicate the number of points desired for the chart. If a time unit is chosen, indicate the value of the interval according to the chosen unit.

Combo Box unit

see just above.

Sample :

When you change the unit and or the value of step, redraw the charts according to the new choise.

Strategy :

The strategy to aggregate the value, when there are several values in the step defined above.

Clear :

Cleans all CSV file from the Chart and the table

|<-Left :

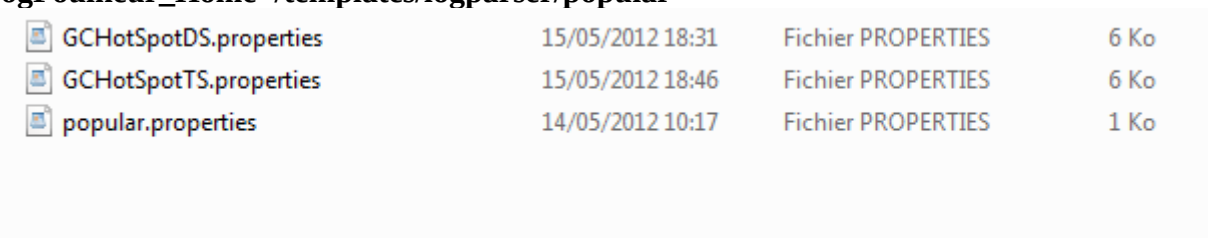
allows, when 2 or more series are “marked” in the table (column “select”), to align these series to the left side of the chart by shifting the beginning of all series to the minimum of the beginning of the marked series.

Right ->|:

allows, when 2 or more series are “marked” in the table (column “select”), to align these series to the right side of the chart by shifting the beginning of all series to the maximum of the beginning of the marked series.

4.4.2 Sub-Menu "Parse & View"

This sub-menu is for parsing popular logs files that have not to be configured by hands.
The principle is to recognise the type of file (JVM GC Logs for examples) and to apply to it a template of scaLogParser.
This templates are located in the directory :
<LogFouineur_Home>/templates/logparser/popular



The files that you meet here are (but it will be extended in the future) :
The files **GCHotSpotDS.properties** and **GCHotSpotTS.properties** are the templates build from the **ParserLogs** menu described more above in this document.

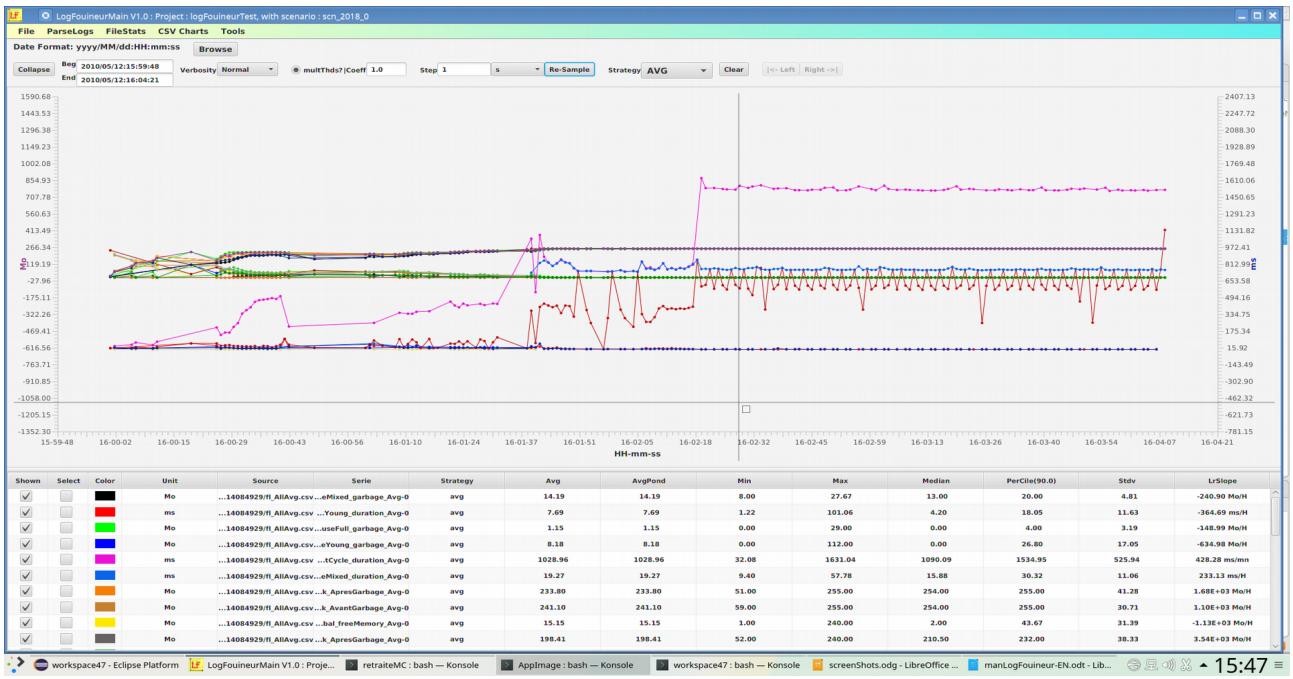
The file **popular.properties** file helps to recognize the “footprint” of the log file. If the file log has no footprint in this file, a popup message is displayed.
popular.properties :

```
popular.list=GCHotSpotDS;GCHotSpotTS


## Empreinte GCHotSpotDS
popular.GCHotSpotDS.debEnr=^(\\d{4}-\\d\\d-\\d\\dT\\d\\d\:\\d\\d\:
\\d\\d\\.\\d\\d\\d(\\+|-)\\d{4}\:\\s+\\d+\.\\d+\:\\s+\\[)
popular.GCHotSpotDS.finEnr=
popular.GCHotSpotDS.isDateExplicit=true
popular.GCHotSpotDS.reg1=(\\[GC|\\[CMS|\\[Full GC)



# Empreinte GCHotSpotTS
popular.GCHotSpotTS.debEnr=^(\\d+\\.\\d+\:\\s+\\[)
popular.GCHotSpotTS.finEnr=
popular.GCHotSpotTS.isDateExplicit=false
popular.GCHotSpotTS.reg1=(\\[GC|\\[CMS|\\[Full GC)
```

popular.list contains the list of templates treated (without the **.properties** suffix)

You choose the file in a FileChooser and , if the footprint of the file is known, the parsing and the charting are done automatically.

After choosing and click on OK, the chart is directly displayed :



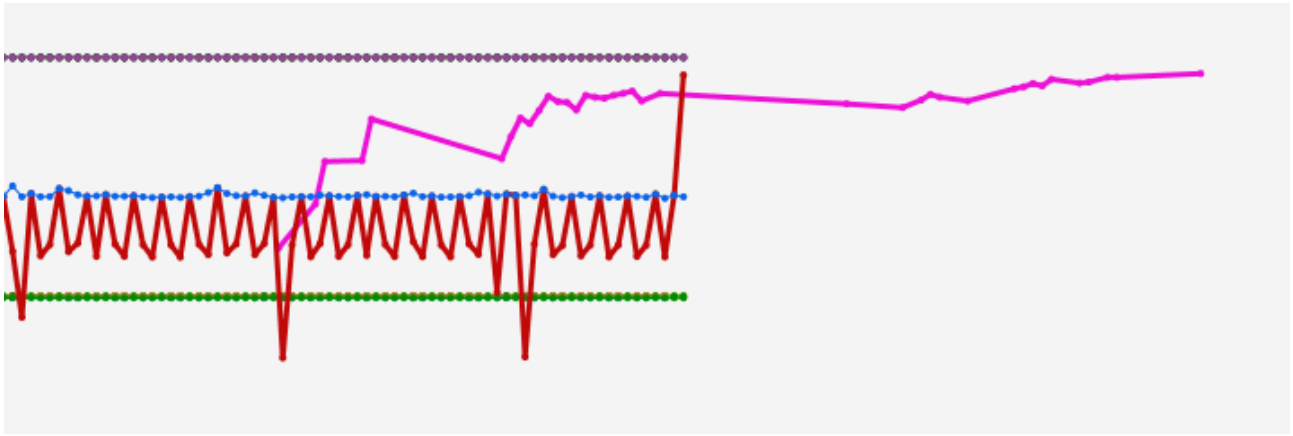
You can go on manipulating the chart as described in the precedent paragraph.

4.4.3 “Funny” feature

When you Crtl+ click on the title of the table , you can add a column “Translate” :

Unit	Translation	Source	Serie	Strategy
Mo	-1 -0.5 0 0.5 1	...084929/fl_AllAvg.csv	...mark_garbage_Avg-0	avg
Mo	-1 -0.5 0 0.5 1	...084929/fl_AllAvg.csv	...ll_freeMemory_Avg-0	avg
Mo	-1 -0.5 0 0.5 1	...084929/fl_AllAvg.csv	...anup_garbage_Avg-0	avg
ms	-1 -0.5 0 0.5 1	...084929/fl_AllAvg.csv	...anup_duration_Avg-0	avg
ms	-1 -0.5 0 0.5 1	...084929/fl_AllAvg.csv	...mark_duration_Avg-0	avg
Mo	-1 -0.5 0 0.5 1	...084929/fl_AllAvg.csv	...ixed_garbage_Avg-0	avg

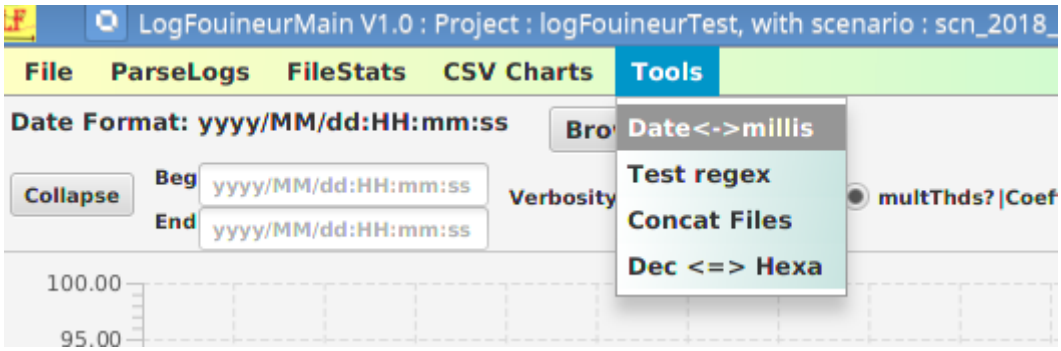
By paying with the cusor, you can translate the series on the the chart above.



By playing in the slider you can translate Time Series to the right or the left . It would be useful to compare the behaviour of two time series where the dates are different (every hours, every day, every week ...).

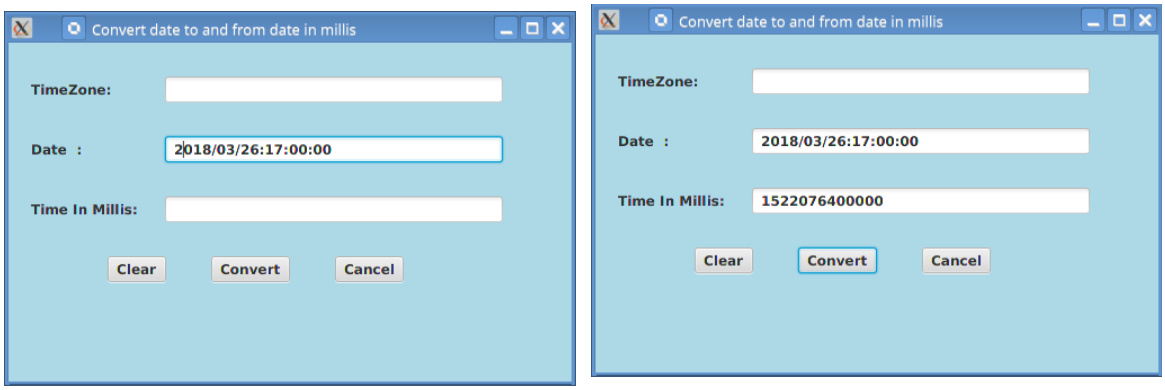
4.5 Menu "Tools"

This menu groups tools that are useful for **LogFouineur**. Other can be added in future versions.



4.5.1 Sub-Menu "Date ↔ millis"

DialogBox that allows date conversions Date in Millis <=> Date in String



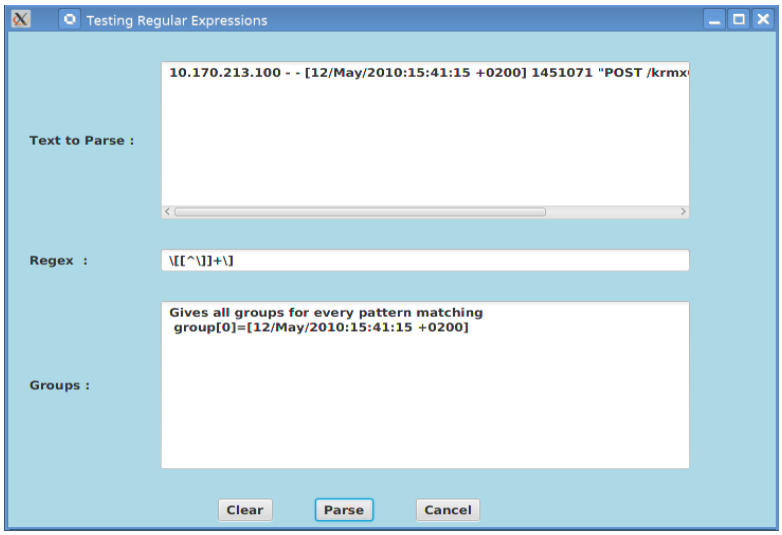
We can :

- not fill any field, and the current date is returned both in milliseconds time and string time
- fill milliseconds Time field and afterwards get the String date
- fill String date field and afterwards get the Milliseconds date
- in the three cases we can modify the Timezone with the format (+|-)dddd => ex : +0200

4.5.2 Sub-Menu "Test regexp"

As seen below, the log parsing uses a lot the mechanism of Regexp Pattern Matching.
Search Pattern in the Java javadoc :

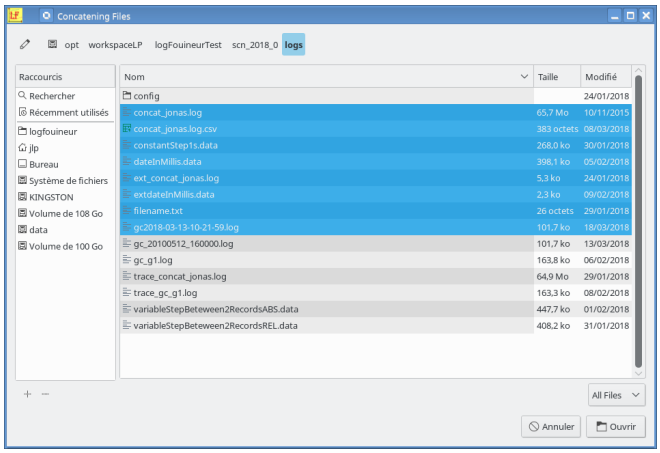
Example :

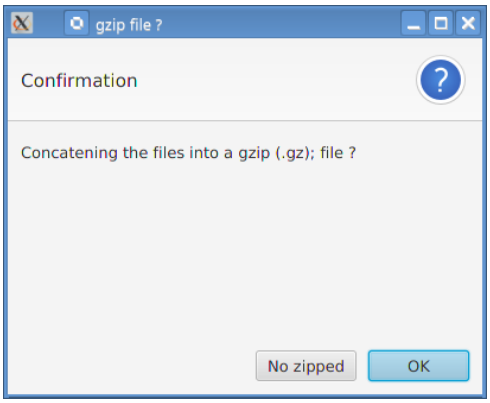


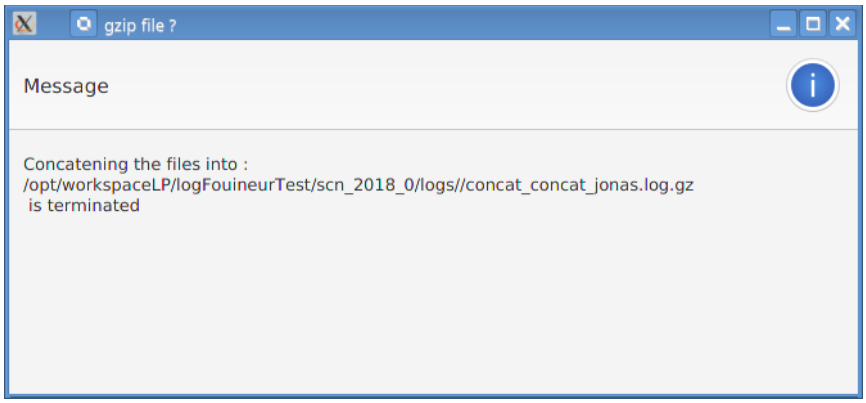
For **LogFouineur**, the interesting result is the result contained in **group[0]**.

4.5.3 Sub-Menu "Concat Files"

This sub-menu allows to concatenate files with the same structure in one file. The result file can be a full text file or a gzip format file.
The files to concate must be in the same directory.
The submenu opens a FileChooser, with the multi select option.

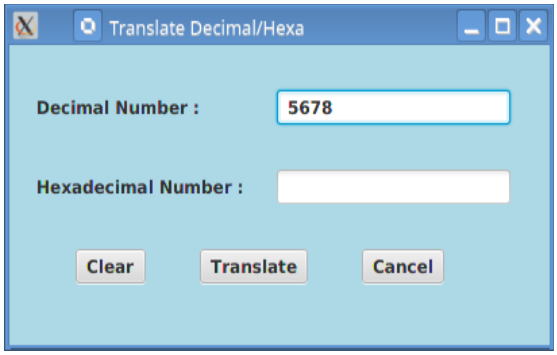






4.5.4 Sub-Menu "Hex <=> Dec"

This tool allow, to convert an hexadecimal number to a decimal number, and the opposite.



5 Annexe

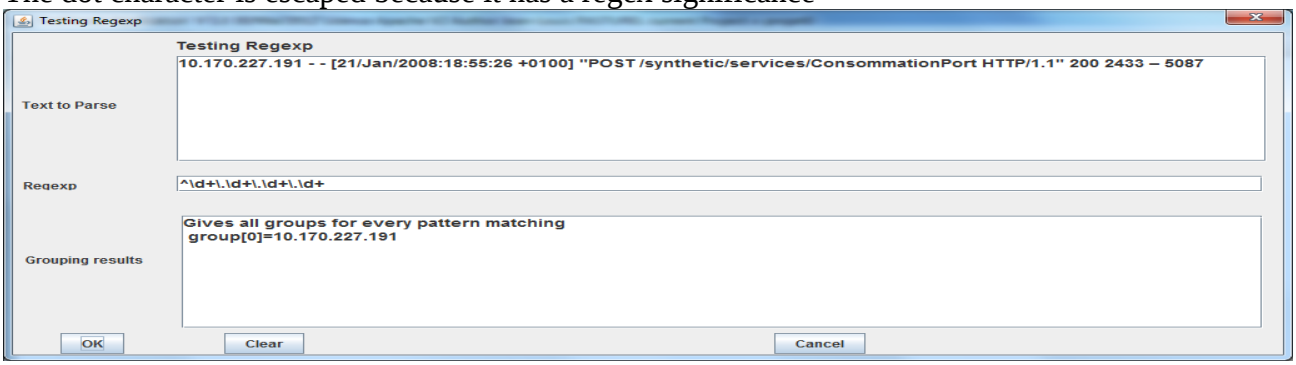
5.1 Examples of Java/Perl regex

Below some examples of regex to extract datas from a line:

```
10.170.227.191 - - [21/Jan/2008:18:55:26 +0100] "POST /synthetic/services/ConsommationPort
HTTP/1.1" 200 2433 – 5087
```

5.1.1 Regexp : ^\d+\.\d+\.\d+\.\d+

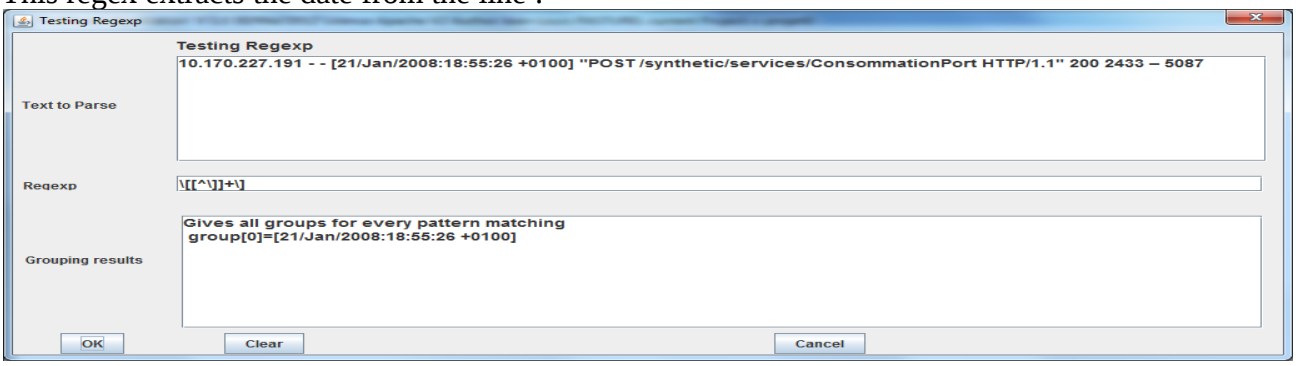
This regexp reads the line from the beginning of the line (character ^) , expects one or several numbers until a dot (**\d+\.**) , this repeated three times, and finally expects one or several more numbers (**\d**+).
The dot character is escaped because it has a regex significance



so you extract the IP address at the beginning of the line.

5.1.2 Regexp : \[[^\]]+\]

This regexp, reads from the first opening square bracket (**\[**) that must be escaped because it has a regex significance , and after it takes all characters that **are not** closing square bracket, note that in this context, the ^ character is the negation operator when it is not the first character of the regex=> (**[^\]]**+) , and the matching is terminated by the closing square bracket (**\]**)
This regex extracts the date from the line :

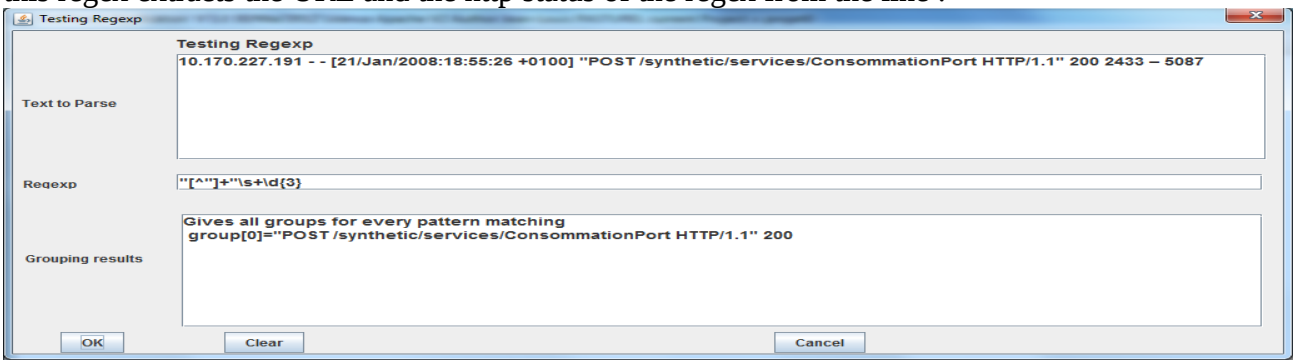


Note : Be care of the difference of signification of the ^ character when it is the first character of the regex, and when it is somewhere else.

5.1.3 Regexp : "[^"]+"\s+\d{3}

That must be read from the first double quotation marks (") and after all characters that are not double quotation marks (**[^"]**+) , after a double quotation mark (") , after one or more space or tab characters (**\s**+), and terminated by 3 digits (**\d{3}**) .

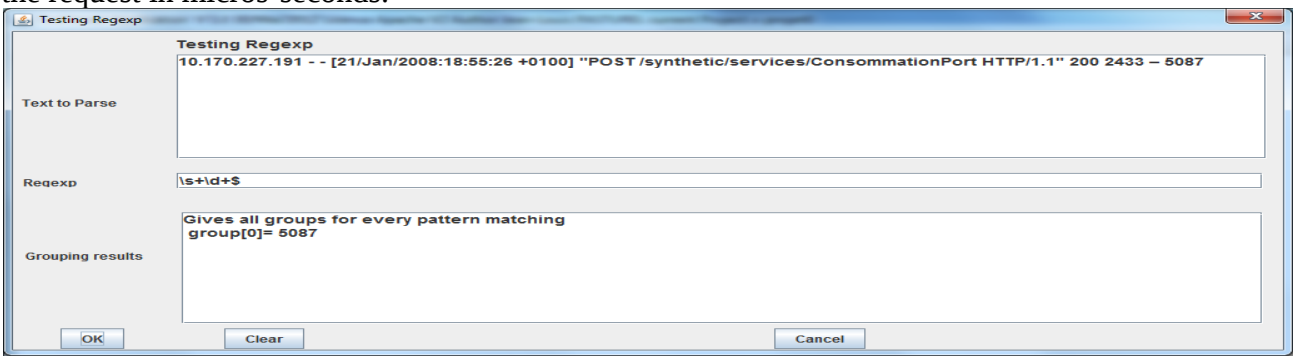
this regex extracts the URL and the http status of the regex from the line :



5.1.4 Regexp : \s+\d+$

This regexp reads beginning from the end of the line (last character is designated by **$**), before the end of the line one or more digits (**\d**+) and before one or more space or tab characters (**\s**+).

this regex extracts the last number in the line, for this Apache logs, it corresponds to the duration of the request in micros-seconds.



5.2 Use of key word “function” in LogFouineur/ParseLogs and FileStats

5.2.1 File to treat

Extract of the file to treat

```
2008/01/27 17:49:40 val1= 10.3 val2=15
2008/01/27 17:49:40 val1= 10.3 val2=15
2008/01/27 17:49:40 val1= 10.3 val2=15
2008/01/27 17:49:40 val1= 10.3 val2=15
```

```
2008/01/27 17:49:41 val1= 10.3 val2=16
2008/01/27 17:49:41 val1= 10.3 val2=16
2008/01/27 17:49:41 val1= 10.3 val2=16
2008/01/27 17:49:42 val1= 10.3 val2=17
2008/01/27 17:49:42 val1= 10.3 val2=17
2008/01/27 17:49:43 val1= 10.3 val2=18
2008/01/27 17:49:46 val1= 10.3 val2=19
2008/01/27 17:49:47 val1= 10.3 val2=19
2008/01/27 17:49:47 val1= 10.3 val2=19
2008/01/27 17:49:48 val1= 10.3 val2=20
2008/01/27 17:49:48 val1= 10.3 val2=16
2008/01/27 17:49:48 val1= 10.3 val2=16
2008/01/27 17:49:48 val1= 10.3 val2=16
```

Note the space after val1= .
The goal of the function is to get the result of the 4 operations (+,-,*,/) between val1 and val2

5.2.2 Classes Java "myPlugins/plugins"

Below the source of **Compute2Values** class :

- **Compute2Values.scala**

```
package plugins;

import java.util.regex.Matcher;
import java.util.regex.Pattern;

// TODO: Auto-generated Javadoc
/**
 * The Class Compute2Values.
 */
public class Compute2Values implements IMyPlugins {
      String regex1_1,regex1_2,regex2_1,regex2_2;
      String op;



      /**
       * Initialize.
       */
      public void initialize(String strRegex2){
      /*    The first character is the separator for splitting
       * Defining a tab of parameters param
       * params[0] is the operation "+","-";"*";"/";"%";"<";">"
            params[1] is the regex1 to extract first number
            params[2] is the regex2 to extract first number
            params[3] is the regex1 to extract second number
            params[4] is the regex2 to extract second number

            */


            /* retrieve separator*/
            String sep = strRegex2.substring(0, 1);

            String[] params = strRegex2.substring(1).split(sep);
```

```
            op=params[0];
            regex1_1=params[1];
            regex1_2=params[2];
            regex2_1=params[3];
            regex2_2=params[4];

      };

      /**
       * Return double.
       *
       * @param params the params
       * @return the double
       */
      public Double returnDouble(String line){
            /**
            line is the record

            for regex a blank character " " means no regex

            */
//

System.out.println("------------------------------------------------------------")
;
//          System.out.println("params.length"+params.length);
//          for (int i =0;i< params.length;i++) {
//                System.out.println("param["+i+"]="+ params[i] );
//          }
//

System.out.println("------------------------------------------------------------\n
");

            Double ret=Double.NaN;
            double firstNumber=Double.NaN;
            double secondNumber=Double.NaN;

            // Extraction FirstNumber
            Matcher match1=Pattern.compile(regex1_1).matcher(line);
            if(match1.find()){
                  String extract1=match1.group();
                  if(regex1_2.trim().length()>0){
                        Matcher
match2=Pattern.compile(regex1_2).matcher(extract1);
                        if(match2.find()){
                              String extract2=match2.group();

      firstNumber=Double.parseDouble(extract2.replaceAll(",", "."));

                        }else return Double.NaN;

                  } else {
                        firstNumber=Double.parseDouble(extract1.replaceAll(",",
```

```
"."));
                  }

            }else return Double.NaN;

            // Extraction SecondNumber
            Matcher match3=Pattern.compile(regex2_1).matcher(line);
            if(match3.find()){
                  String extract1=match3.group();
                  if(regex2_2.trim().length()>0){
                        Matcher
match4=Pattern.compile(regex2_2).matcher(extract1);
                        if(match4.find()){
                              String extract2=match4.group();

      secondNumber=Double.parseDouble(extract2.replaceAll(",", "."));

                        }else return Double.NaN;

                  } else {
                        secondNumber=Double.parseDouble(extract1.replaceAll(",",
"."));
                  }

            }else return Double.NaN;

            // All 2 Number are correct compute now
            switch (op){
            case "+" :
                  ret = firstNumber+secondNumber;
                  break;
            case "-" :
                  ret = firstNumber-secondNumber;
                  break;
            case "*" :
                  ret = firstNumber*secondNumber;
                  break;
            case "/" :
                  if (secondNumber == 0d) ret=Double.NaN;
                  else  ret=firstNumber/secondNumber;

                  break;
            case "%" :
                  if (secondNumber == 0d) ret=Double.NaN;
                  else  ret=firstNumber%secondNumber;

                  break;
            case "<" :
                  ret= Math.min(firstNumber, secondNumber);
                  break;
            case ">" :
                  ret= Math.max(firstNumber, secondNumber);
                  break;
```

```
            }

            return ret;
      }

}
```

Note : If the Name of the Class begins with Mono, this class must be used in Mono- threaded parsing
The Java code of this class is in the directory **<LogFouineur_Home>/myPlugins/plugins.**
The binary of this class is in the jar archive **<LogFouineur_Home>/myPlugins/plugins.jar**