

CSV Comparator Documentation

|  |  |
| --- | --- |
| Key | Value |
| Author: | Reto Scheiwiller |
| Last modified: | 04.03.2016 |
| CSV Comparator Version: | 1.5 |

Table of Contents

[1 Introduction 3](#_Toc415590390)

[1.1 Identifier Column 3](#_Toc415590391)

[1.2 Top Level 3](#_Toc415590392)

[1.3 Default Comparison Mode 4](#_Toc415590393)

[1.4 Manual Comparison Mode 5](#_Toc415590394)

[1.4.1 Compare Definitions 5](#_Toc415590395)

[2 CSVComparatorGUI.jar – User Interface 6](#_Toc415590396)

[3 CSVComparatorCLI.jar – Command Line Interface 7](#_Toc415590397)

[3.1 Arguments 7](#_Toc415590398)

[3.2 Methods for Column Definitions 10](#_Toc415590399)

[3.3 Examples 11](#_Toc415590400)

[3.3.1 Mandatory Input Example 11](#_Toc415590401)

[3.3.2 Using a Configuration File 12](#_Toc415590402)

[3.3.3 Specify identifier column and don’t sort result 13](#_Toc415590403)

[3.3.4 Do Manual Comparison by specifying -column.compareDef 14](#_Toc415590404)

[3.3.5 Compare with top levels 15](#_Toc415590405)

[3.3.6 Print result to standard output 17](#_Toc415590406)

[4 Tips & Tricks 18](#_Toc415590407)

[4.1 Excel 18](#_Toc415590408)

[4.2 Automation with Bash-Script 19](#_Toc415590409)

[4.3 Automation Suite 20](#_Toc415590410)

# Introduction

The CSV Comparator is a data processing tool which takes two files containing comma separated values and compares them. The format of the files should be proper, this means the first line is a header row and each line is considered to have the same number of delimited values as the header line.  
  
There are some arguments which can be used to set input and output formats, as well how the data should be processed.

Because the list of arguments provided to the jar can get huge, the argument “-config.file” provides the possibility to store the arguments in a file.  
  
There are two “modes” which you can use to process the input, default and manual comparison. Please find more information in the specific sections.

The following subsections describe some important concepts and terms which are used in this document, they are important to understand how the CSV Comparator.jar works. For examples, please refer to the “Examples” section.

## Identifier Column

The identifier column is the column which contains the value which is considered as the identifier of a row. If in both files a row has the same identifier the comparator will compare these rows to each other.

All the identifier values will be merged into one column, which is a list of unique values. All the other columns will be applied to this merged column and will be printed so in the result.

If in one file two or more rows have the same identifying value, by default a number in braces “[1]” is added to the values to make them unique.

When using the command line interface, by default the first column in both files is considered as the identifier column, to set it manually use the argument “-column.identifier”.

## Top Level

The top level could be used if the files contain records with sub records, for example:

|  |  |  |
| --- | --- | --- |
| **Level** | **Name** | **Value** |
| Top | Warehouse A | 8 |
| Sub | Apple | 5 |
| Sub | Pear | 3 |
| Top | Warehouse B | 8008 |
| Sub | Apple | 7 |
| Sub | Grapes | 8000 |
| Sub | Pear | 1 |

The top level definition for this example would look like this:

-topLevel=Level,Level,Top

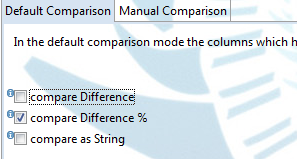
This will consider the column “Level” from both files, and will consider all rows which contains “Top” in this column as a top level. The CSVComparator.jar will split both files in sub parts which contains a top level and all rows till the next top level. This sub parts will be compared to each other, if the identifying value in the top level row is equals.  
If the identifier column in the above example is the column “Name”, and in both files is a top level with “Warehouse A”, this top levels with its sub rows will be compared. Top levels which are not in both files will be written at the end of the result, so the data is complete.

## Default Comparison Mode

In the default mode the CSVComparator.jar will take from both files the columns which have the same header and compare them. Columns which are not found in both files are ignored.

**User Interface:**

On the user interface yo will find the following tab were you can trigger a defaut comparison:



**Command Line**  
On the command line, default comparison mode will be used when no “-column.compareDef” argument is provided. To specify how the columns should be compared, use the following arguments:

“-result.comparediff”, “-result.comparediff%” and “-result.comparestring”.

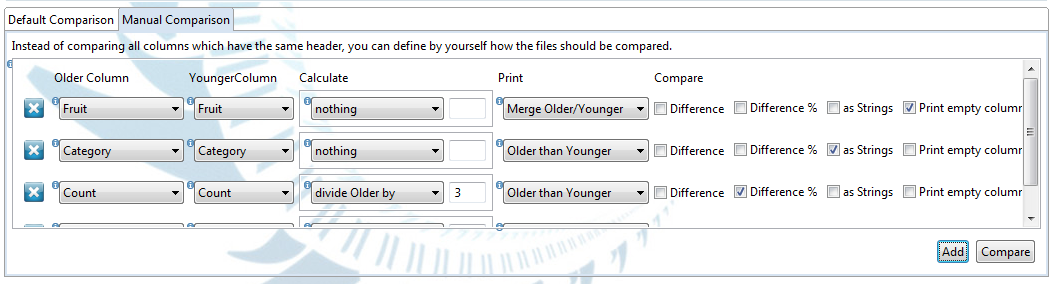
By default, for each compared column pair a column with change in percentage is

printed(-result.comparediff%=true).

if you set all three of the above mentioned arguments to true, for each compared column pair 3 additional columns will be printed.

## Manual Comparison Mode

If the identifier column is not printed by specifying a compare definition with specifying “print Merged”, it will be automatically printed as the first column in the result.

**User Interface**  
To do a manual comparison change to the tab ”Manual Comparison”, add compare definitions(see section “1.4.1 Compare Definitions”) and click compare.  


**Command Line**

The manual comparison is used when at least one “-column.compareDef” argument is provided to the jar. In this mode, the CSVComparator.jar will only do what is defined by the “-column.compareDef”.

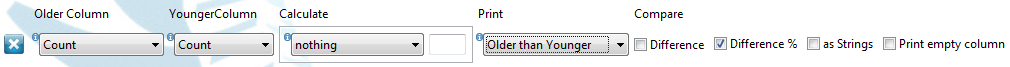
The arguments “-result.comparediff”, “-result.comparediff%” and “-result.comparestring” will be ignored, all other arguments and their default values are also used in this mode.

Compare Definitions

Compare definitions are used to manually configure how the comparison should be done. For one comparison, multiple compare definitions can be specified, they will be processed in their order.

A column definition consists of the column header of the older and the younger file, as well some methods which specify what should be done with these columns, for example:

**User Interface**A compare definition in the user interface:



**Command Line**

This will take the column “Count” from both files, print the older column to the result, print the younger column to the result and compare the difference between the columns.

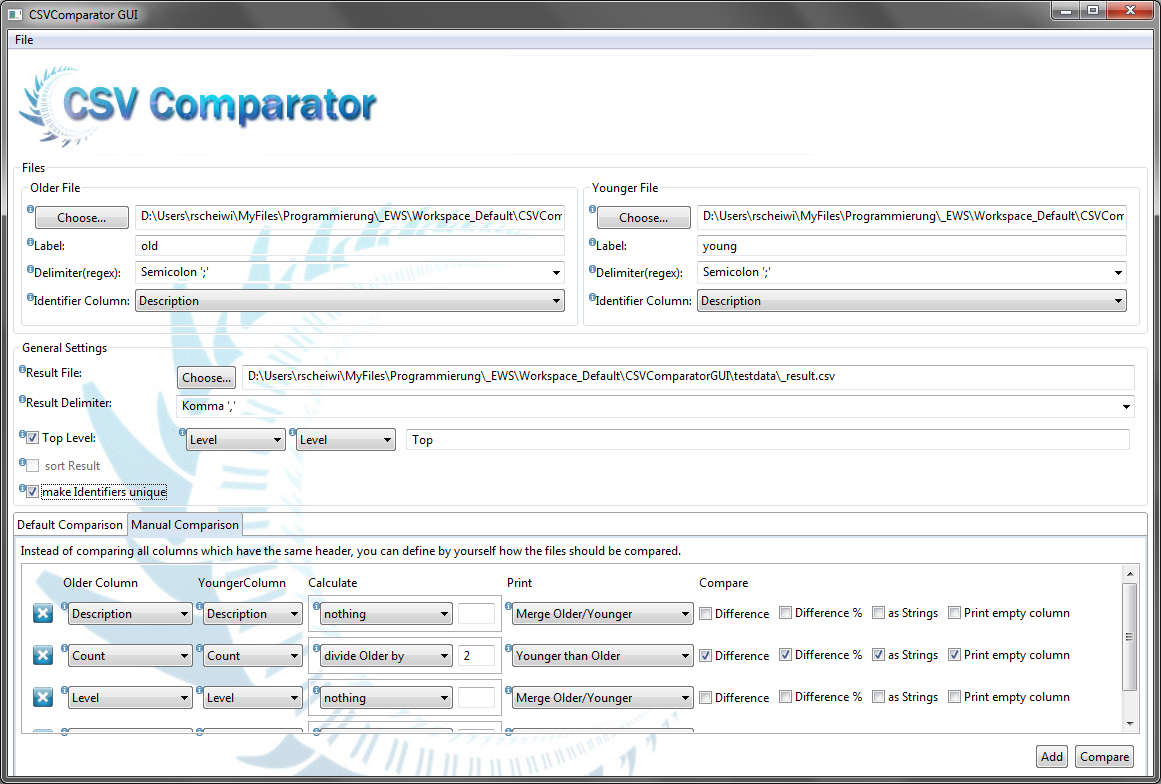
-column.compareDef =Count,Count,printOlder(),printYounger(),compareDifference%()

# CSVComparatorGUI.jar – User Interface

(required java version: 7.0)

The following screenshot shows the user interface. The documentation of the fields can be accessed by hover over the small info icons “”.  
  
The user interface allows to choose two CSV files and compare them. Currently the result is only written to a file and will not be displayed in the application.

Through the menu “File” you can save and load configuration files, which can also be loaded by the CSVComparatorCLI.jar by using the argument “-config.file”. Also it allows you generate a bash script, which can be used to automate the comparison.



# CSVComparatorCLI.jar – Command Line Interface

(required java version: 6.0)

## Arguments

If no arguments or wrong arguments are provided through the command line, a list of available arguments will be printed.

| Argument | Description |
| --- | --- |
| -config.file | **Syntax:** -config.file={filepath}  The path to a config-file. The config-file can include all the arguments defined in this list delimited by newline. Also lines starting with “#” are considered as comments, also blank lines are allowed. |
| -config.loglevel.console | **Syntax:** config.loglevel.console={ALL|TRACE|DEBUG|INFO|WARN|ERROR|FATAL|OFF}  **Default:** “INFO”  Log level(log4j2) printed to the standard output |
| -config.loglevel.file | **Syntax:** config.loglevel.file={ALL|TRACE|DEBUG|INFO|WARN|ERROR|FATAL|OFF}  **Default:** “DEBUG”  Log level(log4j2) printed to the logfile. |
| -older.label | **Syntax:** -older.label={label}  **Default:** “older”  The label which is appended to the older columns in the results. |
| -older.file | **Syntax:** -older.file={filepath}  The path to the older csv-file, which is used in the comparison. |
| -older.delimiter | **Syntax:** -older.delimiter={delimiter}  **Default:** “,”  The delimiter which is used in the older file to delimit the data. |
| -older.quotes | **Syntax:** -older.quotes={true|false}  **Default:** false  **Description:** Toggles handling of quoted values, including \\ and \" inside quotation. (does not work with \" outside of quotation) |
| -younger.label | **Syntax:** -younger.label={label}  **Default:** “younger”  The label which is appended to the younger columns in the results. |
| -younger.file | **Syntax:** -younger.file={filepath}  The path to the younger csv-file, which is used in the comparison. |
| -younger.delimiter | **Syntax:** -younger.delimiter={delimiter}  **Default:** “,”  The delimiter which is used in the younger file to delimit the data. |
| -younger.quotes | **Syntax:** - younger.quotes={true|false}  **Default:** false  **Description:** Toggles handling of quoted values, including \\ and \" inside quotation. (does not work with \" outside of quotation) |
| -result.file | **Syntax:** -result.file={filepath}  **Default:** “./CSVComparatorResult.csv”  The path to the result file, were the compared data should be written. |
| -result.delimiter | **Syntax:** -result.delimiter={delimiter}  **Default:** “,”  The delimiter which is used in the result to delimit the data. |
| -result.quotes | **Syntax:** -result.quotes={true|false}  **Default:** false  **Description:** Toggles writing result with quotes. |
| -result.comparediff | **Syntax:** -result.comparediff={true|false}  **Default:** “false”  **Dependencies:** Takes no effect if “-column.compareDef” is used.  If true, for all columns the difference will be calculated. |
| -result.comparediff% | **Syntax:** -result.comparediff%={true|false}  **Default:** “true”  **Dependencies:** Takes no effect if “-column.compareDef” is used.  If true, for all columns the difference in percentage will be calculated. |
| -result.comparestring | **Syntax:** -result.comparestring={true|false}  **Default:** “false”  **Dependencies:** Takes no effect if “-column.compareDef” is used.  If true, for all columns the values will be compared as strings. If the strings are equal the result is “EQ”, otherwise the result is “NOTEQ”. |
| -result.sort | **Syntax:** -result.sort={true|false}  **Default:** “true”  If true, the result will be sorted by the natural order of the identifier column. If false, the rows from both files are merged by alternately taking one row of each file.  (see argument “-column.identifier” for more information). |
| -result.stdout | **Syntax:** -result.stdout={true|false|onerror} **Default:** “false”If true, the result will be written to the standard output instead to a file. 'onerror' will print the result to standard output in case an error occured during writing the file. If you plan to pipe the output to another tool, set'-config.loglevel.console=OFF' so nothing else will be written to the standard output. |
| -column.identifier | **Syntax:** -column.identifier={olderColumnHeader},{youngerColumnHeader}  **Default:** First column in both files.  Set the identifier column for the older and the younger File. If this argument is not used, the first column in both files is considered as the identifier column. |
| -column.identifier.makeunique | **Syntax:** -column.identifier.makeunique={true|false}  **Default:** “true”  If true, the identifiers in both files are made unique, if one idenitifier is used multiple times.  If this is turned off, only the first row with the identifier is used, all other rows are ignored. |
| -column.toplevel | **Syntax:** -column.toplevel={olderColumnHeader},{youngerColumnHeader},{Value}  Set the topLevel column for the older and the younger File. The value is the string which is used to determine if a row is a topLevel. |
| -column.compareDef | **Syntax:** -column.compareDef={olderColumnHeader}, {youngerColumnHeader},{method},…  Set a column definition for the manual comparison. You can specify as many methods as you want delimited by comma. For a list of methods you can use, refer to the section “Methods for Column Definitions”. This argument can be used zero to many times. |

## Methods for Column Definitions

|  |  |
| --- | --- |
| Method | Description |
| printOlder() | This method prints the older column, which is applied to the merged identifier column, to the result. |
| printYounger() | This method prints the older column, which is applied to the merged identifier column, to the result. |
| printMerged() | This method merges the values from the older and younger columns and prints it to the result. If both columns contains a value and the values are not equal, the values will be merged as “<olderValue>/<youngerValue>”. This method is useful when you want to print a column containing values which are related to the identifier column and serve as additional information. |
| printSeparator() | Print a column to the result which doesn’t contain values. |
| compareDifference%() | Compares the difference in percentage between the columns and print a result column. |
| compareDifference() | Compares the difference between the columns and print a result column. |
| compareAsStrings() | Compares the values of the columns as strings and print a result column.  **“EQ”:** the values are equal **“NOTEQ”:** the values are equal |
| divideYoungerBy(value) | Divides all values of the younger column by the provided value. All methods which are called after this method will use the newly calculated values. |
| multiplyYoungerBy(value) | Multiplies all values of the younger column by the provided value. All methods which are called after this method will use the newly calculated values. |
| divideOlderBy(value) | Divides all values of the older column by the provided value. All methods which are called after this method will use the newly calculated values. |
| multiplyOlderBy(value) | Multiplies all values of the older Column by the provided value. All methods which are called after this method will use the newly calculated values. |

## Examples

Mandatory Input Example

Only the two arguments “-older.file” and “-younger.File” are mandatory as input, for the other arguments there are default values or they are optional. Please consider the following defaults:

* For both files the delimiter is comma “,”
* The label appended to the columns are
  + for the olderFile: “older”
  + for the youngerFile: “younger”
* The result are written to “C:/temp/CVSComparator/compare\_result.csv”
* the first column is considered as identifier column.
* The difference percentage is calculated

**Execute:** The execution from the command line will look like this:

java -jar CSVComparator.jar –older.file="./testdata/fruits\_older.csv" –younger.file="./testdata/fruits\_younger.csv"

**Input:** Contents of fruits\_older.csv, delimited by “,“:

|  |  |
| --- | --- |
| Fruit | Count |
| Apple | 10 |
| Pear | 7 |
| Banana | 5 |
| Orange | 20 |

Contents of fruits\_younger.csv, delimited by “,”:

|  |  |
| --- | --- |
| Fruit | Count |
| Apple | 8 |
| Pear | 17 |
| Banana | 5 |
| Lemon | 16 |

**Result:** Contents of “C:/temp/CVSComparator/compare\_result.csv”:

|  |  |  |  |
| --- | --- | --- | --- |
| Fruit | Count(older) | Count(younger) | Count Diff% |
| Apple | 10 | 8 | -0.199999999999999 |
| Banana | 5 | 5 | 0 |
| Lemon | - | 16 | - |
| Orange | 20 | - | - |
| Pear | 7 | 17 | 1.428571429 |

**Result Explanations:**

* The “Fruit” column was used as identifier, the values from both files were merged.
* All columns which are not used as identifier are compared, in this case, only the column Count
* The labels are added to the column headers to be able to identify the column
* “Count Diff%” contains the comparison
* The result is sorted by the natural order of the identifier column
* The values for “Lemon” and “Orange” are not compared, because they are not in both files

Using a Configuration File

**Execute:** The execution from the command line will look like this:

java -jar CSVComparator.jar –config.file="./examples/\_example\_config\_file.dat"

**Config-File:**

Contents of “./examples/\_example\_config\_file.dat”:

# All lines starting with “#” are considered as comments, blank lines are allowed

# Each argument has to be on a separate line.

# Older File Configuration

-older.file=./testdata/fruits\_older.csv

-older.label=01.2013

-older.delimiter=,

# Younger File Configuration

-younger.file=./testdata/fruits\_younger.csv

-younger.label=02.2013

-younger.delimiter=,

# Result File Configuration

-result.file=./result/compared\_fruits\_ConfigFile.csv

-result.delimiter=;

# Output Configuration

-result.comparediff=true

-result.comparediff%=true

-result.comparestring=true

**Input:** Contents of fruits\_older.csv, delimited by “,“:

|  |  |
| --- | --- |
| Fruit | Count |
| Apple | 10 |
| Pear | 7 |
| Banana | 5 |
| Orange | 20 |

Contents of fruits\_younger.csv, delimited by “,”:

|  |  |
| --- | --- |
| Fruit | Count |
| Apple | 8 |
| Pear | 17 |
| Banana | 5 |
| Lemon | 16 |

**Result:** Contents of “./result/compared\_fruits.csv”, delimited by “;”:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Fruit | Count(01.2013) | Count(02.2013) | Count Diff | Count Diff% | Count EQ? |
| Apple | 10 | 8 | -2 | -0.2 | NOTEQ |
| Banana | 5 | 5 | 0 | 0 | EQ |
| Lemon | - | 16 | - | - | - |
| Orange | 20 | - | - | - | - |
| Pear | 7 | 17 | 10 | 1.428571429 | NOTEQ |

**Result Explanations:**

* The result is written with semicolon “;” as delimiter.
* The defined labels are used
* For column “Count”, the difference, the difference in percentage and a string comparison is done.

Specify identifier column and don’t sort result

If the identifier Column in your file is not the first column which is used by default, or the identifier column has a different name in the compared files, use the argument “-column.identifier” to choose it manually.  
  
With the argument “-result.sort” you can turn off sorting the result.

**Execute:** The execution from the command line will look like this:

java -jar CSVComparator.jar -configFile="./examples/\_example\_identifierColumn\_sortResult.dat"

**Config-File:**

Contents of “./examples/\_example\_identifier column\_sortResult.dat”:

-older.file=./testdata/fruits\_identifierColumn\_older.csv

-younger.file=./testdata/fruits\_identifierColumn\_younger.csv

-result.file=./result/compared\_fruits\_identifierColumn.csv

-result.delimiter=;

-column.identifier=Fruit,SweetFood

-result.sort=false

-column.compareDef=Count,Count,printOlder(),printYounger(),compareDifference%()

-config.debug=true

**Input:** Contents of “fruits\_identifier column\_older.csv”, delimited by “,“:

|  |  |
| --- | --- |
| Count | Fruit |
| 10 | Apple |
| 7 | Pear |
| 5 | Banana |
| 20 | Orange |

Contents of fruits\_identifier column\_younger.csv, delimited by “,”:

|  |  |
| --- | --- |
| Count | SweetFood |
| 8 | Apple |
| 17 | Pear |
| 5 | Banana |
| 16 | Lemon |

**Result:** Contents of “./result/compared\_fruits.csv”, delimited by “,”:

|  |  |  |  |
| --- | --- | --- | --- |
| Count(older) | Count(younger) | Count Diff% | SweetFood |
| 10 | 8 | -0.2 | Apple |
| 7 | 17 | 1.428571429 | Pear |
| 5 | 5 | 0 | Banana |
| 20 | - | - | Orange |
| - | 16 | - | Lemon |

**Result Explanations:**

* The identifier column is printed at the same position as in the younger file, at the end.
* The name of the identifier column is the one from the younger column.
* The rows are not sorted, they are alternately taken from both files.

Do Manual Comparison by specifying -column.compareDef

**Config-File:** Contents of “./examples/\_example\_columnDef.dat”:

-older.file=./testdata/fruits\_columnDef\_older.csv

-younger.file=./testdata/fruits\_columnDef\_younger.csv

-result.file=./result/compared\_fruits\_columnDef.csv

-result.delimiter=;

-column.compareDef=Count,Count,printOlder(),printYounger(),compareDifference%()

-column.compareDef=Category,Category,printOlder(),printYounger(),compareAsStrings()

-config.debug=true

**Input:** Contents of “fruits\_ columnDef \_older.csv”, delimited by “,“:

|  |  |  |
| --- | --- | --- |
| Fruit | Category | Count |
| Apple | A | 10 |
| Apple | B | 12 |
| Pear | B | 7 |
| Banana | C | 5 |
| Orange | D | 20 |

Contents of fruits\_ columnDef \_younger.csv, delimited by “,”:

|  |  |  |
| --- | --- | --- |
| Fruit | Category | Count |
| Apple | A | 8 |
| Pear | B | 17 |
| Banana | A | 5 |
| Banana | B | 8008 |
| Lemon | A | 16 |

**Result:** Contents of “./result/compared\_fruits.csv”, delimited by “,”:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Fruit | Count (older) | Count (younger) | Count  Diff% | Category (older) | Category (younger) | Category  EQ? |
| Apple [1] | 10 | 8 | -0.2 | A | A | EQ |
| Apple [2] | 12 | - | - | B | - | - |
| Banana [1] | 5 | 5 | 0 | C | A | NOTEQ |
| Banana [2] | - | 8008 | - | - | B | - |
| Lemon | - | 16 | - | - | A | - |
| Orange | 20 | - | - | D | - | - |
| Pear | 7 | 17 | 1.428571429 | B | B | EQ |

**Result Explanations:**

* The identifier column is the first column of both files(Default).
* The first “-column.compareDef” option prints first the two “Count” columns, and then a result column which contains the difference in percentage.
* The second “-column.compareDef” option prints first the two “Category” columns, and then a result column which compares the values as strings (**EQ**=equal, **NOTEQ**=not equal).

Compare with top levels

**Config-File:** Contents of “./examples/\_example\_topLevel.dat”:

-config.loglevel.console=INFO

-config.loglevel.file=DEBUG

-older.file=.\testdata\fruits\_topLevel\_older.csv

-older.label=old

-older.delimiter=;

-younger.file=D:.\testdata\fruits\_topLevel\_younger.csv

-younger.label=young

-younger.delimiter=;

-result.file=.\result\compared\_fruits\_topLevel.csv

-result.sort=false

-column.toplevel=Level,Level,Top

-column.compareDef=Count,Count,printOlder(),printYounger(),compareDifference%()

**Input:** Contents of “fruits\_ topLevel\_older.csv”:

|  |  |  |
| --- | --- | --- |
| Description | Count | Level |
| Warehouse A | **54** | **Top** |
| Orange | 20 | Sub |
| Apple | 10 | Sub |
| Apple | 12 | Sub |
| Pear | 7 | Sub |
| Banana | 5 | Sub |
| Warehouse B | **62** | **Top** |
| Orange | 15 | Sub |
| Apple | 14 | Sub |
| PassionFruit | 16 | Sub |
| Pear | 7 | Sub |
| Banana | 10 | Sub |
| Warehouse C | **16157** | **Top** |
| Plum | 0 | Sub |
| Durian | 2 | Sub |
| Cranberry | 10000 | Sub |
| Strawberry | 600 | Sub |
| Blackberry | 5555 | Sub |

**Input:** Contents of “fruits\_ topLevel \_younger.csv”:

|  |  |  |
| --- | --- | --- |
| Description | Count | Level |
| Warehouse A | **92** | **Top** |
| Orange | 5 | Sub |
| Apple | 77 | Sub |
| Pear | 2 | Sub |
| Banana | 8 | Sub |
| Warehouse D | **238** | **Top** |
| PassionFruit | 16 | Sub |
| Mango | 2 | Sub |
| Grapes | 220 | Sub |
| Warehouse C | **13646** | **Top** |
| Melon | 0 | Sub |
| Durian | 2 | Sub |
| Cranberry | 12345 | Sub |
| Strawberry | 300 | Sub |
| Blackberry | 999 | Sub |
| Warehouse B | **62** | **Top** |
| Orange | 17 | Sub |
| Apple | 12 | Sub |
| PassionFruit | 18 | Sub |
| Pear | 9 | Sub |
| Banana | 6 | Sub |

Please find the result on the next page.

**Result:** Contents of “./result/compared\_fruits\_topLevel.csv”:

|  |  |  |  |
| --- | --- | --- | --- |
| Description | Count(old) | Count(young) | Count Diff% |
| Warehouse A | **54** | **92** | **0.703703704** |
| Orange | 20 | 5 | -0.75 |
| Apple [1] | 10 | 77 | 6.7 |
| Apple [2] | 12 | - | - |
| Pear | 7 | 2 | -0.714285714 |
| Banana | 5 | 8 | 0.6 |
| Warehouse B | **62** | **62** | **0** |
| Orange | 15 | 17 | 0.133333333 |
| Apple | 14 | 12 | -0.142857143 |
| PassionFruit | 16 | 18 | 0.125 |
| Pear | 7 | 9 | 0.285714286 |
| Banana | 10 | 6 | -0.4 |
| Warehouse C | **16157** | **13646** | **-0.155412515** |
| Plum | 0 | - | - |
| Melon | - | 0 | - |
| Durian | 2 | 2 | 0 |
| Cranberry | 10000 | 12345 | 0.2345 |
| Strawberry | 600 | 300 | -0.5 |
| Blackberry | 5555 | 999 | -0.820162016 |
| Warehouse D | **-** | **238** | **-** |
| PassionFruit | - | 16 | - |
| Mango | - | 2 | - |
| Grapes | - | 220 | - |

**Result Explanations:**

* The data is splitted into “blocks” with a top level. All the rows below a top level till the next are considered as a block. When there is a block with the same top level in both files, these blocks are compared to each other. In this example Warehouse A, B and C.
* Warehouse D is only found in the younger file and could not be compared. Because all data of both files should be present in the result file to have a complete picture, all blocks which couldn’t be compared are added at the end of the result file.

Print result to standard output

If you want to print the result to the standard output for direct viewing or to pipe it to another tool, use the argument “-result.stdout=true” and turn off the logging to the standard output with “-config.loglevel.console=OFF”.(Logs will still be written to the logfile)

**Config-File:** Contents of “./examples/\_example\_stdout.dat”:

-older.file=./testdata/fruits\_older.csv

-older.delimiter=,

-younger.file=./testdata/fruits\_younger.csv

-younger.delimiter=,

-result.stdout=true

-result.delimiter=;

-config.loglevel.console=OFF

# Tips & Tricks

## Excel

Delimiting with List separator

If you will open the results in Excel, choosing the default list separator as the delimiter will cause that Excel will split up the values in cells automatically when opening the file.

To find or define the default list separator go to the following location:

* Control Panel >> Reqion and Language >> Formats >> Additional Settings >> List Separator

Percentage Formatting

To format percentage values in Excel with +/- signs and red/green color highlighting for easier analysis, you can go to “Right Click Cells >>Format Cells >> Custom ” and paste this definition into the field “type”:

[Red]+0.0%;[Green]-0.0%;0.0%

Recording Macros

If you have to analyze the same compared data again and again, you can record a macro under   
“View >> Macros >> Record Macro …”, also the stop of the recording appears in the same location.   
With “View Macros” you can see you recorded macros and run them.  
  
Also it is possible to make your own buttons in the menubar to execute your macros. Go to   
“Right Click menubar >> Customize the Ribbon…” to do that.

If you store your macros in a separate file on a shared location, you can also export this customized menubars and share it with other people.

## Automation with Bash-Script

You can automate the call to the jar-file with a bash script like this:

#!/bin/bash

##################################################################################

# Author: Reto Scheiwiller

#

# Compare Files

# ------------------------------------

# This script will compare the CSV-Files set in the Variables.

# This script takes no arguments, please set the variables in the script.

#

##################################################################################

# Check if there are no arguments present

**if** **[** "$#" **-**ne 0 **]**

**then**

###########################################################

# Print Help

###########################################################

**echo** "compare\_CSV.sh takes no arguments, set the variables in the script."

**exit** 1

**else**

###########################################################

# Set Configuration

###########################################################

##### Java Environment Configration #####

JAVA**=**'/cygdrive/c/Program Files (x86)/Java/jre1.6.0/bin/java'

CSVCOMPARE\_JAR**=**'C:\Programs\CSVComparator\CSVComparator.jar'

#older File

OLDER\_FILE**=**'C:\Temp\older\_file.csv'

OLDER\_LABEL**=**'OLD'

OLDER\_DELIMITER**=**';'

#newer File

YOUNGER\_FILE**=**'C:\Temp\younger\_file.csv'

YOUNGER\_LABEL**=**'YOUNG'

YOUNGER\_DELIMITER**=**';'

#Result Config

RESULT\_FILE**=**" C:\Temp\compare\_result.csv"

RESULT\_DELIMITER**=**';'

###########################################################

# Compare Files

###########################################################

"$JAVA" **-**jar **$CSVCOMPARE\_JAR\**

"-older.file=$OLDER\_FILE"**\**

"-older.delimiter=$OLDER\_DELIMITER"**\**

"-older.label=$OLDER\_LABEL"**\**

"-younger.File=$YOUNGER\_FILE"**\**

"-younger.delimiter=$YOUNGER\_DELIMITER"**\**

"-younger.label=$YOUNGER\_LABEL"**\**

"-result.file=$RESULT\_FILE"**\**

"-result.delimiter=$RESULT\_DELIMITER"

**fi**

## Automation Suite

If you have data which frequently have to be compared to each other, you can build a small bash environment to automate this process.



**Repository 1: automation.cfg**



**Repository 2: automation.cfg**



**configuration.sh**



**collect\_Data.sh**



**compare\_CSV.sh**



**compare\_CSV\_noVersions.sh**



**Temp**



**other Scripts…**



**other Resources…**