The CSV-Sorter program

Manual for version 0.92-beta (2014/07/09)

Documentation 2014/07/09

http://T-F-S.github.io/csvsorter/

Thomas F. Sturm[1]

Contents

[1]Prof. Dr. Dr. Thomas F. Sturm, Institut für Mathematik und Informatik, Universität der Bundeswehr München, D-85577 Neubiberg, Germany; email: thomas.sturm@unibw.de

1 Introduction

The **CSV-Sorter** program serves to sort CSV[2] files.

It is a Java command-line program which processes one CSV input file to one CSV output file controlled by an XML configuration file. Depending on the configuration, **CSV-Sorter** can deal with different formats, separators, delimiters, various sorting presets, header and no header files.

The **CSV-Sorter** program was developed as external sorting program for the `csvsimple` LaTeX package, see `www.ctan.org/tex-archive/macros/latex/contrib/csvsimple`. But it can be used for any CSV sorting task.

[2]CSV file: file with comma separated values.

2 License

CSV-Sorter is licensed under the New BSD License[3]. The New BSD License has been verified as a GPL-compatible free software license by the Free Software Foundation[4], and has been vetted as an open source license by the Open Source Initiative[5].

[3]http://opensource.org/licenses/BSD-3-Clause
[4]http://www.fsf.org/
[5]http://www.opensource.org/

3 Installation

3.1 Basic Installation

CSV-Sorter is written in Java and needs to have a Java virtual machine (version 1.6 or higher) installed. With some luck, you already have Java installed on your system. To check this, open a command console i. e. some terminal on Linux or executing `cmd.exe` on Windows.

If executing `> java -version` on the command-line displays some Java version information, you have Java installed. Otherwise, you need to install a Java Runtime Environment to proceed.

The **CSV-Sorter** is an executable Jar-archive named `csvsorter.jar`. You can build `csvsorter.jar` from the sources, but this is usually not necessary.

Change with your command console to the directory, where you saved `csvsorter.jar`. Execute the following on the command-line in this directory:

```
> java -jar csvsorter.jar
```

You should see something like
`This is CSV-Sorter` followed by
`Configuration file is missing.` and further text. This means that you are ready to use **CSV-Sorter** in the current directory. If you get some Java errors, your installed virtual machine is probably too old and you have to update it to use **CSV-Sorter** .

3.2 Further Installation

The further steps are optional and provide an easier use for **CSV-Sorter** .

Instead of calling `> java -jar csvsorter.jar` ***options***, you are recommended to create a shortcut batch/script to access **CSV-Sorter** easily from everywhere on your system.

On Windows, this would be something like:

csvsorter.cmd

```
@echo off
java -jar "c:\SOME PATH STRING\csvsorter.jar" %*
```

Replace `c:\SOME PATH STRING\csvsorter.jar` by the appropriate path for `csvsorter.jar`.

Put this `csvsorter.cmd` to a directory which is part of your system path. If you do not know where to put this file, create a directory `c:\bat`, put the file into it, and *add* the new directory to your path variable. A similar script file is recommended for Linux users.

After these steps, **CSV-Sorter** is accessible system-wide by

```
> csvsorter options
```

4 Usage

Depending on the installation done in the last section, **CSV-Sorter** is called by

```
> java -jar csvsorter.jar options
```

or

```
> csvsorter options
```

(used in the following)

where *options* are the following:

`-c` *configuration xml file*

This mandatory configuration file controls the processing of the input CSV file. See the following examples and Section 5 for details.

`-l` *logfile*

This file is used to write logging messages. If it is not specified or the configuration is faulty, `csvsorter.log` is used instead.

`-i` *input csv file*

The CSV file to process.

`-o` *output csv file*

The CSV file to write after processing. It is an error to set the output file equal to the input file.

`-x` *input=output csv file*

The CSV file to process and to write after processing (overwriting).

- The presence of a configuration file is mandatory.
- Note that the configuration file may contain the rest of the options.
- Command-line options override configuration file settings.

4.1 Example 1

songcontest.csv (input)

```
Title,Artist,Country,Televote,Juryvote
The green hills of the shire,Frodo and Friends,New Zealand,76,45
La la la la la,The Singers,United Kingdom,25,62
One and two and three,Hansi Unterober,Germany,47,35
Paris e Calais,Chantal and Pascal,France,47,41
The bell rings,Baab,Sweden,87,24
Rarara,Channel Rats,Grand Fenwick,12,14
```

This CSV file is to be sorted by the first column. Since the input file has a header line, the name `Title` for the first column can be used inside the configuration file.

titlesort.xml (configuration)

```
<?xml version="1.0" encoding="UTF-8"?>
<csv>
  <sortlines>
    <column name="Title" order="ascending" type="string"/>
  </sortlines>
</csv>
```

Processing:

```
> csvsorter -c titlesort.xml -i songcontest.csv -o songcontest_sorted.csv
```

Output file:

songcontest_sorted.csv (output)

```
Title,Artist,Country,Televote,Juryvote
La la la la la,The Singers,United Kingdom,25,62
One and two and three,Hansi Unterober,Germany,47,35
Paris e Calais,Chantal and Pascal,France,47,41
Rarara,Channel Rats,Grand Fenwick,12,14
The bell rings,Baab,Sweden,87,24
The green hills of the shire,Frodo and Friends,New Zealand,76,45
```

Note that the configuration file can be used for any further examples independent from the number of columns or the position of the `Title` column.

4.2 Example 2

songcontest.csv

```
Title,Artist,Country,Televote,Juryvote
The green hills of the shire,Frodo and Friends,New Zealand,76,45
La la la la la,The Singers,United Kingdom,25,62
One and two and three,Hansi Unterober,Germany,47,35
Paris e Calais,Chantal and Pascal,France,47,41
The bell rings,Baab,Sweden,87,24
Rarara,Channel Rats,Grand Fenwick,12,14
```

This CSV file is to be sorted by the `Televote` numbers in descending order. If candidates have the same Televote values, the `Country` is used for sorting:

televotesort.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<csv>
  <sortlines>
    <column name="Televote" order="descending" type="integer"/>
    <column name="Country" order="ascending" type="string"/>
  </sortlines>
</csv>
```

Processing:

```
> csvsorter -c televotesort.xml -i songcontest.csv -o songcontest_sorted.csv
```

Output file:

songcontest_sorted.csv

```
Title,Artist,Country,Televote,Juryvote
The bell rings,Baab,Sweden,87,24
The green hills of the shire,Frodo and Friends,New Zealand,76,45
Paris e Calais,Chantal and Pascal,France,47,41
One and two and three,Hansi Unterober,Germany,47,35
La la la la la,The Singers,United Kingdom,25,62
Rarara,Channel Rats,Grand Fenwick,12,14
```

Note that the configuration file can be used for any further examples independent from the number of columns or the position of the `Televote` and `Country` columns.

4.3 Example 3

songcontest.csv

```
Title,Artist,Country,Televote,Juryvote
The green hills of the shire,Frodo and Friends,New Zealand,76,45
La la la la la,The Singers,United Kingdom,25,62
One and two and three,Hansi Unterober,Germany,47,35
Paris e Calais,Chantal and Pascal,France,47,41
The bell rings,Baab,Sweden,87,24
Rarara,Channel Rats,Grand Fenwick,12,14
```

This CSV file is to be sorted by the sum of the `Televote` and `Juryvote` numbers in descending order. If candidates have the same values, the `Country` is used for sorting:

sumsort.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<csv>
  <sortlines>
    <sum order="descending" type="integer">
      <column name="Televote"/>
      <column name="Juryvote"/>
    </sum>
    <column name="Country" order="ascending" type="string"/>
  </sortlines>
</csv>
```

Processing:

```
> csvsorter -c sumsort.xml -i songcontest.csv -o songcontest_sorted.csv
```

Output file:

songcontest_sorted.csv

```
Title,Artist,Country,Televote,Juryvote
The green hills of the shire,Frodo and Friends,New Zealand,76,45
The bell rings,Baab,Sweden,87,24
Paris e Calais,Chantal and Pascal,France,47,41
La la la la la,The Singers,United Kingdom,25,62
One and two and three,Hansi Unterober,Germany,47,35
Rarara,Channel Rats,Grand Fenwick,12,14
```

Note that the configuration file can be used for any further examples independent from the number of columns or the position of the `Televote`, `Juryvote`, and `Country` columns.

4.4 Example 4

songcontest.csv

```
Title,Artist,Country,Televote,Juryvote
The green hills of the shire,Frodo and Friends,New Zealand,76,45
La la la la la,The Singers,United Kingdom,25,62
One and two and three,Hansi Unterober,Germany,47,35
Paris e Calais,Chantal and Pascal,France,47,41
The bell rings,Baab,Sweden,87,24
Rarara,Channel Rats,Grand Fenwick,12,14
```

This CSV file is to be sorted by the sum of the `Televote` and `Juryvote` numbers in descending order. If candidates have the same values, the `Title` is used for sorting. The output should contain only `Juryvote`, `Juryvote`, and `Title`. Further, we need double quotes as brackets and semicolons as delimiters:

filtersort.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<csv>
  <outDelimiter sign=";"/>
  <outBracket leftsymbol="doublequote" rightsymbol="doublequote" />
  <sortlines>
    <sum order="descending" type="integer">
      <column name="Televote"/>
      <column name="Juryvote"/>
    </sum>
    <column name="Title" order="ascending" type="string"/>
  </sortlines>
  <outColumns>
    <column name="Televote"/>
    <column name="Juryvote"/>
    <column name="Title"/>
  </outColumns>
</csv>
```

Processing:

```
> csvsorter -c filtersort.xml -i songcontest.csv -o songcontest_sorted.csv
```

Output file:

songcontest_sorted.csv

```
"Televote";"Juryvote";"Title"
"76";"45";"The green hills of the shire"
"87";"24";"The bell rings"
"47";"41";"Paris e Calais"
"25";"62";"La la la la la"
"47";"35";"One and two and three"
"12";"14";"Rarara"
```

4.5 Example 5

gradetab.csv

```
name        givenname        matriculation        gender        grade
Maier        Hans        12345        m        1.0
Huber        Anna        23456        f        2.3
Weissbaeck        Werner        34567        m        5.0
```

This CSV file is not to be sorted, but just to be reformatted. The input uses tabulator signs as delimiters (invisible here). The output should use commas and add additional curly braces:

format.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<csv>
  <delimiter signsymbol="tab"/>
  <outDelimiter sign=","/>
  <outBracket left="{" right="}" />
</csv>
```

Processing:

```
> csvsorter -c format.xml -i gradetab.csv -o gradetab_sorted.csv
```

Output file:

gradetab_sorted.csv

```
{name},{givenname},{matriculation},{gender},{grade}
{Maier},{Hans},{12345},{m},{1.0}
{Huber},{Anna},{23456},{f},{2.3}
{Weissbaeck},{Werner},{34567},{m},{5.0}
```

5 Configuration

The **CSV-Sorter** program is controlled by an XML configuration file according to the following template.

Configuration file template

```
<?xml version="1.0" encoding="UTF-8"?>
<csv>
  <noHeader/>
  <delimiter sign=";"/>
  <bracket leftsymbol="doublequote" rightsymbol="doublequote" empty="false" />
  <outDelimiter signsymbol="comma"/>
  <outBracket left="{" right="}" empty="false"/>
  <transform/>
  <contentToLaTeX/>
  <charset in="InputCharset" out="OutputCharset"/>
  <language iso="de"/>
  <sortlines>
    <column name="Country" order="ascending" type="string"/>
    <column name="Points" order="ascending" type="integer"/>
    <column number="2" order="descending" type="string"/>
    <sum order="ascending" type="integer">
      <column name="Value"/>
      <column number="7"/>
    </sum>
  </sortlines>
  <outColumns>
    <column name="Points"/>
    <column name="Country"/>
  </outColumns>
  <input file="InputFile"/>
  <output file="OutputFile"/>
  <log file="LogFile"/>
</csv>
```

Nearly all tag elements are optional and there is no specific order of appearance. The document element `<csv>` is mandatory.

5.1 `<noheader>`

If this element is present, the CSV file(s) do not have a header line. Note that in this case the columns can be addressed by number only. If this element is not present, the first line of the CSV file is interpreted as header line and its contents can be used to address columns by names.

5.2 `<delimiter>`

This element defines the delimiter sign for the input file. If it is not present, the comma is the default delimiter. The actual delimiter sign is defined by one of two feasible attributes of the element.

`sign=","` The value of the attribute is the actual delimiter sign.

`signsymbol="comma"` The value of the attribute is a symbolic description of the actual delimiter sign. See Table 1 on page 12 for a list of feasible symbol names.

1: Symbolic sign names

Description	Sign
`braceleft`	"{" curly brace left
`braceright`	"}" curly brace right
`comma`	"," comma
`doublequote`	""" double quote
`pipe`	"|" pipe
`semicolon`	";" semicolon
`singlequote`	"'" single quote
`tab`	tabulator sign

5.3 `<bracket>`

This element defines the bracket signs for entries of the input file. If it is not present, double quotes are uses as default brackets. The actual bracket signs are defined by the following feasible attributes of the element.

`left="{"` The value of the attribute is the actual bracket sign.

`leftsymbol="doublequote"` The value of the attribute is a symbolic description of the actual bracket sign. See Table 1 on page 12 for a list of feasible symbol names.

`right="}"` The value of the attribute is the actual bracket sign.

`rightsymbol="doublequote"` The value of the attribute is a symbolic description of the actual bracket sign. See Table 1 on page 12 for a list of feasible symbol names.

`empty="true"` If the value of the attribute is `true`, no input brackets are used at all. Setting `left` and `right` to an empty string is *not* equivalent to this (actually, this would be ignored)!

If the brackets are not set empty, brackets still are not mandatory to be used in the input file. But if an opening bracket is found in the input file, there has to be a matching closing bracket.

5.4 `<outDelimiter>`

This element defines the delimiter sign for the output file. If it is not present, the input delimiter sign is used for the output also. The actual delimiter sign is defined by one of two feasible attributes of the element.

`sign=","` The value of the attribute is the actual delimiter sign.

`signsymbol="comma"` The value of the attribute is a symbolic description of the actual delimiter sign. See Table 1 on page 12 for a list of feasible symbol names.

5.5 `<outBracket>`

This element defines the bracket signs for entries of the output file. If it is not present, the input bracket signs are used as default brackets. The actual bracket signs are defined by the following feasible attributes of the element.

`left="{"` The value of the attribute is the actual bracket sign.

`leftsymbol="doublequote"` The value of the attribute is a symbolic description of the actual bracket sign. See Table 1 on page 12 for a list of feasible symbol names.

`right="}"` The value of the attribute is the actual bracket sign.

`rightsymbol="doublequote"` The value of the attribute is a symbolic description of the actual bracket sign. See Table 1 on page 12 for a list of feasible symbol names.

`empty="true"` If the value of the attribute is `true`, no output brackets are used at all. Setting `left` and `right` to an empty string is *not* equivalent to this (actually, this would be ignored)!

If the lines of the output file are not needed to be transformed and this element is not present, the output file is written with the same lines as the input file (even with missing brackets).

5.6 `<transform>`

If this element is present, the input lines are always transformed to output lines. Actually, output brackets are always set.

5.7 `<contentToLaTeX>`

If this element is present, the content text is processed to be more LaTeX friendly. Especially, `"\"` is replaced by `"\textbackslash{}"`, `"&"` is replaced by `"\&{}"`, etc. You should not process a file twice with this setting!

5.8 `<charset>`

This element defines the character set for the input and output files. If this element is not present, the default character set of the current Java virtual machine is used depending upon the locale and character set of the underlying operating system.

`in="windows-1252"` This defines the character set for the input file.

`out="UTF-8"` This defines the character set for the output file. If an input character set is given but no output character set, then the input character set is used as output character also.

Feasible charset names are listed in the IANA Charset Registry[6], but not all of them will be implemented in the current Java virtual machine. Of interest may be `"US-ASCII"`, `"UTF-8"` (Unicode), `"windows-1252"` (Windows Western Latin, e. g. German), `"IBM850"` (DOS-Latin-1).

[6] `http://www.iana.org/assignments/character-sets/character-sets.xhtml`

5.9 `<language>`

This element defines the language used for sorting, e. g. for proper observance of German umlauts, etc. If this element is not present, the default locale of the current Java virtual machine is used depending upon the locale of the underlying operating system.

`iso="de"` The value of the attribute is the actual ISO 639 alpha-2 or alpha-3 language code, e. g. `"de"` (German), `"en"` (English), `"fr"` (French).

2: Data types for columns

Type	Comment
`integer`	integer value between -2^{31} and $2^{31}-1$
`long`	long value between -2^{63} and $2^{63}-1$
`double`	double precision floating point value (not localized!)
`date`	date localized by `<language>`
`string`	text localized by `<language>`

5.10 `<sortlines>`

This element defines the actual sorting presets. It may contain a list of `<column>` and/or `<sum>` elements for a hierarchical sorting specification. The first sub-element has the highest priority.

5.10.1 `<column>`

Defines a sorting rule according to one column. The column has to be denoted by `name` or `number` and to be set to a `type` of data.

`name="NAME"` Denotes the column by a name given by the header line. The value is not case sensitive.

`number="1"` Denotes the column by number (started at 1).

`order="ascending"` Sets the sorting rule to be `"ascending"` (default) or `"descending"`.

`type="integer"` Sets the data type. See Table 2 on page 15 for a list of feasible data types. If this attribute is not present, `"string"` is used as data type.

`default="VALUE"` Sets a default value for the column content. It is used for sorting, if the actual column content cannot be parsed according to the given data type. If this attribute is not present, lines with unparsable content are ignored.

5.10.2 `<sum>`

Defines a sorting rule according to the summarized value of columns. The columns a denoted by a list of embedded `<column>` elements. Further, the sum has the following attributes:

`order="ascending"` Sets the sorting rule to be `"ascending"` (default) or `"descending"`.

`type="integer"` Sets the mandatory data type. Feasible data types are `"integer"`, `"long"`, and `"double"` only.

`default="VALUE"` Sets a default value for the sum. It is used for sorting, if the actual sum cannot be computed for any reasons. If this attribute is not present, lines with uncomputable sums are ignored.

5.10.2.1 `<column>` Defines one column of the sum. The column has to be denoted by `name` or `number`.

`name="NAME"` Denotes the column by a name given by the header line.

`number="1"` Denotes the column by number (started at 1).

5.11 `<outColumns>`

Defines a set of columns to be used for the output file. The columns a denoted by a list of embedded `<column>` elements.

5.11.1 `<column>`

Defines one column of the output. The column has to be denoted by `name` or `number`.

`name="NAME"` Denotes the column by a name given by the header line.

`number="1"` Denotes the column by number (started at 1).

5.12 `<input>`

This element defines the input file. The mandatory attribute of the element is:

`file="FILENAME"` The value of the attribute is the actual file name.

`overwrite="true"` If this attribute is present and its value equals `true`, the input file is allowed to be overwritten. Note that you have to specify the output file nonetheless. Command-line options may change this setting.

The appropriate command-line option overwrites this value.

5.13 `<output>`

This element defines the output file. The mandatory attribute of the element is:

`file="FILENAME"` The value of the attribute is the actual file name.

The appropriate command-line option overwrites this value.

5.14 `<log>`

This element defines the log file. The mandatory attribute of the element is:

`file="FILENAME"` The value of the attribute is the actual file name.

The appropriate command-line option overwrites this value.

6 Hints, Tricks, and Troubleshooting

6.1 Hierarchical brackets

The line scanning algorithm tries to identify the columns of the CSV file on a best-effort base. Therefore, it always uses the configured `<delimiter>` *and* `<bracket>`.

- Brackets can be omitted in the input file, but if an opening bracket was used, there has to be a matching closing bracket.
- Everything between a delimiter and an opening bracket is considered a whitespace! Analogously, everything between a closing bracket and a delimiter is considered a whitespace!

  With standard setting,

  ```
  ..., bla " My text" bla,...
  ```

  is interpreted as

  ```
  ...," My text",...
  ```

- Inside a bracket pair, a delimiter sign is interpreted as normal text.

  ```
  ...,"one,two",...
  ```

  Here, the single column content is `one,two`.

- Brackets are interpreted hierarchically, i. e. you can have brackets inside brackets. Note that *all* opening brackets need to have matching closing brackets. If the left bracket sign is identical to the right bracket sign (as in the standard case), detection is done on a best-effort base.

  ```
  ...,"one "two", and "three"",...
  ```

  Here, the single column content is `one "two", and "three"`.

- With the standard settings, the following line is faulty, even if brackets are omitted:

  ```
  ...,Fl\"ache,...
  ```

  To circumvent the problem, you should configure other bracket signs or empty brackets, even if you do not use brackets directly in the input file.

7 Version History

Version 0.92-beta (2014/07/09)

fixed	Data loss, if input and output file are the same, corrected.
fixed	Descending string sorting corrected.
changed	Input and output files are checked to be different.
changed	’type’ attribute is not mandatory any more (set to ’string’ if not present).
new	Input=output file with new ’x’ command-line option or ’overwrite’ attribute.
new	Console messages added for error cases.
new	’default’ attributes for columns and sums added.
new	Data type ’long’ added.

Version 0.91 beta (2014/07/05)

changed	Hierarchical bracket algorithm improved.
changed	Speed optimization for brackets (about 50 percent).
new	Empty input and output brackets implemented.

Version 0.90 beta (2014/06/30)

new	First public release.

Version unpublished (2008/12)

new	Unpublished private version(s).