The csvsimple-legacy package

Manual for version 2.5.1pre1 (2023/10/17)

Thomas F. Sturm[1]

https://www.ctan.org/pkg/csvsimple

https://github.com/T-F-S/csvsimple

Abstract

csvsimple(-legacy) provides a simple LaTeX interface for the processing of files with comma separated values (CSV). csvsimple-legacy relies heavily on the key value syntax from pgfkeys which results in an easy way of usage. Filtering and table generation is especially supported. Since the package is considered as a lightweight tool, there is no support for data sorting or data base storage.

Actually, csvsimple-legacy is identical to the old version 1.22 (2021/06/07) of csvsimple. It is superseded by csvsimple-l3, a LaTeX3 implementation of csvsimple which is a *nearly* drop-in for the erstwhile implementation.

- If you are a new user or an experienced user of csvsimple creating a new document, you are encouraged to turn to csvsimple-l3, see «The csvsimple-l3 package»
- If you used csvsimple before version 2.00 in one or many documents, there is *no need* to change anything. Loading csvsimple without options loads csvsimple-legacy. csvsimple-legacy will be maintained to stay functional as it is for the sake of compatibility to old documents.
- Differences between csvsimple-legacy and csvsimple-l3 are discussed in «The csvsimple package».

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

Contents

1 Introduction

The `csvsimple-legacy` package is applied to the processing of CSV[2] files. This processing is controlled by key value assignments according to the syntax of `pgfkeys`. Sample applications of the package are tabular lists, serial letters, and charts.

An alternative to `csvsimple-legacy` is the `datatool` package which provides considerably more functions and allows sorting of data by LaTeX. `csvsimple-legacy` has a different approach for the user interface and is deliberately restricted to some basic functions with fast processing speed.

Mind the following restrictions:

- Sorting is not supported directly but can be done with external tools, see Section 3.9 on page 24.
- Values are expected to be comma separated, but the package provides support for other separators, see Section 3.7 on page 22.
- Values are expected to be either not quoted or quoted with curly braces `{}` of TeX groups. Other quotes like doublequotes are not supported directly, but can be achieved with external tools, see Section 5.6 on page 42.
- Every data line is expected to contain the same amount of values. Unfeasible data lines are silently ignored by default, but this can be configured, see Section 3.3 on page 17.

1.1 Loading the Package

The package `csvsimple-legacy` loads the packages `pgfkeys`, `etoolbox`, and `ifthen`. `csvsimple-legacy` itself is loaded with *one* of the following alternatives inside the preamble:

```
\usepackage{csvsimple}
   % or alternatively (not simultaneously!)
\usepackage[legacy]{csvsimple}
   % or alternatively (not simultaneously!)
\usepackage{csvsimple-legacy}
```

Not automatically loaded, but used for many examples are the packages `longtable` and `booktabs`.

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

1.2 First Steps

Every line of a processable CSV file has to contain an identical amount of comma[3] separated values. The curly braces `{}` of TEX groups can be used to mask a block which may contain commas not to be processed as separators.

The first line of such a CSV file is usually but not necessarily a header line which contains the identifiers for each column.

CSV file «grade.csv»

```
name,givenname,matriculation,gender,grade
Maier,Hans,12345,m,1.0
Huber,Anna,23456,f,2.3
Weißbäck,Werner,34567,m,5.0
Bauer,Maria,19202,f,3.3
```

The most simple way to display a CSV file in tabular form is the processing with the `\csvautotabular` → P. 9 command.

```
\csvautotabular{grade.csv}
```

name	givenname	matriculation	gender	grade
Maier	Hans	12345	m	1.0
Huber	Anna	23456	f	2.3
Weißbäck	Werner	34567	m	5.0
Bauer	Maria	19202	f	3.3

Typically, one would use `\csvreader` → P. 8 instead of `\csvautotabular` to gain full control over the interpretation of the included data.

In the following example, the entries of the header line are automatically assigned to TEX macros which may be used deliberately.

```
\begin{tabular}{|l|c|}\hline%
\bfseries Person & \bfseries Matr.~No.
\csvreader[head to column names]{grade.csv}{}%
{\\\givenname\ \name & \matriculation}%
  \\\hline
\end{tabular}
```

Person	**Matr. No.**
Hans Maier	12345
Anna Huber	23456
Werner Weißbäck	34567
Maria Bauer	19202

[3] See `/csv/separator` → P. 22 for other separators than comma.

`\csvreader` is controlled by a plenty of options. For example, for table applications line breaks are easily inserted by `/csv/late after line` → P. 14. This defines a macro execution just before the following line. Additionally, the assignment of columns to TEX macros is shown in a non automated way.

```
\begin{tabular}{|r|l|c|}\hline%
& Person & Matr.~No.\\\hline\hline
\csvreader[late after line=\\\hline]%
  {grade.csv}{name=\name,givenname=\firstname,matriculation=\matnumber}%
  {\thecsvrow & \firstname~\name & \matnumber}%
\end{tabular}
```

	Person	Matr. No.
1	Hans Maier	12345
2	Anna Huber	23456
3	Werner Weißbäck	34567
4	Maria Bauer	19202

An even more comfortable and preferrable way to create a table is setting appropriate option keys. Note, that this gives you the possibility to create a `pgfkeys` style which contains the whole table creation.

```
\csvreader[tabular=|r|l|c|,
    table head=\hline & Person & Matr.~No.\\\hline\hline,
    late after line=\\\hline]%
  {grade.csv}{name=\name,givenname=\firstname,matriculation=\matnumber}%
  {\thecsvrow & \firstname~\name & \matnumber}%
```

	Person	Matr. No.
1	Hans Maier	12345
2	Anna Huber	23456
3	Werner Weißbäck	34567
4	Maria Bauer	19202

The next example shows such a style definition with the convenience macro `\csvstyle` → P. 11. Here, we see again the automated assignment of header entries to column names by `/csv/head to column names` → P. 16. For this, the header entries have to be without spaces and special characters. But you can always assign entries to canonical macro names by hand like in the examples above. Here, we also add a `/csv/head to column names prefix` → P. 16 to avoid macro name clashes.

```
\csvstyle{myTableStyle}{tabular=|r|l|c|,
    table head=\hline & Person & Matr.~No.\\\hline\hline,
    late after line=\\\hline,
    head to column names,
    head to column names prefix=MY,
    }

\csvreader[myTableStyle]{grade.csv}{}%
  {\thecsvrow & \MYgivenname~\MYname & \MYmatriculation}%
```

	Person	Matr. No.
1	Hans Maier	12345
2	Anna Huber	23456
3	Werner Weißbäck	34567
4	Maria Bauer	19202

Another way to address columns is to use their roman numbers. The direct addressing is done by `\csvcoli`, `\csvcolii`, `\csvcoliii`, ...:

```
\csvreader[tabular=|r|l|c|,
    table head=\hline & Person & Matr.~No.\\\hline\hline,
    late after line=\\\hline]%
  {grade.csv}{}%
  {\thecsvrow & \csvcolii~\csvcoli & \csvcoliii}%
```

	Person	Matr. No.
1	Hans Maier	12345
2	Anna Huber	23456
3	Werner Weißbäck	34567
4	Maria Bauer	19202

And yet another method to assign macros to columns is to use arabic numbers for the assignment:

```
\csvreader[tabular=|r|l|c|,
    table head=\hline & Person & Matr.~No.\\\hline\hline,
    late after line=\\\hline]%
  {grade.csv}{1=\name,2=\firstname,3=\matnumber}%
  {\thecsvrow & \firstname~\name & \matnumber}%
```

	Person	Matr. No.
1	Hans Maier	12345
2	Anna Huber	23456
3	Werner Weißbäck	34567
4	Maria Bauer	19202

For recurring applications, the `pgfkeys` syntax allows to create own styles for a consistent and centralized design. The following example is easily modified to obtain more or less option settings.

```
\csvset{myStudentList/.style={%
  tabular=|r|l|c|,
  table head=\hline & Person & #1\\\hline\hline,
  late after line=\\\hline,
  column names={name=\name,givenname=\firstname}
  }}

\csvreader[myStudentList={Matr.~No.}]{grade.csv}{matriculation=\matnumber}%
{\thecsvrow & \firstname~\name & \matnumber}%
\hfill%
\csvreader[myStudentList={Grade}]{grade.csv}{grade=\grade}%
{\thecsvrow & \firstname~\name & \grade}%
```

	Person	Matr. No.
1	Hans Maier	12345
2	Anna Huber	23456
3	Werner Weißbäck	34567
4	Maria Bauer	19202

	Person	Grade
1	Hans Maier	1.0
2	Anna Huber	2.3
3	Werner Weißbäck	5.0
4	Maria Bauer	3.3

Alternatively, column names can be set by \csvnames → P. 11 and style definitions by \csvstyle → P. 11. With this, the last example is rewritten as follows:

```
\csvnames{myNames}{1=\name,2=\firstname,3=\matnumber,5=\grade}
\csvstyle{myStudentList}{tabular=|r|l|c|,
  table head=\hline & Person & #1\\\hline\hline,
  late after line=\\\hline, myNames}

\csvreader[myStudentList={Matr.~No.}]{grade.csv}{}%
{\thecsvrow & \firstname~\name & \matnumber}%
\hfill%
\csvreader[myStudentList={Grade}]{grade.csv}{}%
{\thecsvrow & \firstname~\name & \grade}%
```

	Person	Matr. No.
1	Hans Maier	12345
2	Anna Huber	23456
3	Werner Weißbäck	34567
4	Maria Bauer	19202

	Person	Grade
1	Hans Maier	1.0
2	Anna Huber	2.3
3	Werner Weißbäck	5.0
4	Maria Bauer	3.3

The data lines of a CSV file can also be filtered. In the following example, a certificate is printed only for students with grade unequal to 5.0.

```
\csvreader[filter not strcmp={\grade}{5.0}]%
  {grade.csv}{1=\name,2=\firstname,3=\matnumber,4=\gender,5=\grade}%
  {\begin{center}\Large\bfseries Certificate in Mathematics\end{center}
  \large\ifcsvstrcmp{\gender}{f}{Ms.}{Mr.}
  \firstname~\name, matriculation number \matnumber, has passed the test
  in mathematics with grade \grade.\par\ldots\par
  }%
```

Certificate in Mathematics

Mr. Hans Maier, matriculation number 12345, has passed the test in mathematics with grade 1.0.

. . .

Certificate in Mathematics

Ms. Anna Huber, matriculation number 23456, has passed the test in mathematics with grade 2.3.

. . .

Certificate in Mathematics

Ms. Maria Bauer, matriculation number 19202, has passed the test in mathematics with grade 3.3.

. . .

2 Macros for the Processing of CSV Files

`\csvreader[`⟨*options*⟩`]{`⟨*file name*⟩`}{`⟨*assignments*⟩`}{`⟨*command list*⟩`}`

`\csvreader` reads the file denoted by ⟨*file name*⟩ line by line. Every line of the file has to contain an identical amount of comma separated values. The curly braces `{}` of TEX groups can be used to mask a block which may contain commas not to be processed as separators.

The first line of such a CSV file is by default but not necessarily processed as a header line which contains the identifiers for each column. The entries of this line can be used to give ⟨*assignments*⟩ to TEX macros to address the columns. The number of entries of this first line determines the accepted number of entries for all following lines. Every line which contains a higher or lower number of entries is ignored during standard processing.

The ⟨*assignments*⟩ are given by key value pairs ⟨*name*⟩=⟨*macro*⟩. Here, ⟨*name*⟩ is an entry from the header line *or* the arabic number of the addressed column. ⟨*macro*⟩ is some TEX macro which gets the content of the addressed column.

The ⟨*command list*⟩ is executed for every accepted data line. Inside the ⟨*command list*⟩ is applicable:

- `\thecsvrow` or the counter `csvrow` which contains the number of the current data line (starting with 1).
- `\csvcoli`, `\csvcolii`, `\csvcoliii`, ..., which contain the contents of the column entries of the current data line. Alternatively can be used:
- ⟨*macro*⟩ from the ⟨*assignments*⟩ to have a logical addressing of a column entry.

Note, that the ⟨*command list*⟩ is allowed to contain `\par` and that all macro definitions are made global to be used for table applications.

The processing of the given CSV file can be controlled by various ⟨*options*⟩ given as key value list. The feasible option keys are described in section 3 from page 14.

```
\csvreader[tabular=|r|l|l|, table head=\hline, table foot=\hline]{grade.csv}%
  {name=\name,givenname=\firstname,grade=\grade}%
  {\grade & \firstname~\name & \csvcoliii}
```

1.0	Hans Maier	12345
2.3	Anna Huber	23456
5.0	Werner Weißbäck	34567
3.3	Maria Bauer	19202

Mainly, the `\csvreader` command consists of a `\csvloop` macro with following parameters:
`\csvloop{`⟨*options*⟩`, file=`⟨*file name*⟩`, column names=`⟨*assignments*⟩`,`
`command=`⟨*command list*⟩`}`
Therefore, the application of the keys `/csv/file` → P. 23 and `/csv/command` → P. 14 is useless for `\csvreader`.

`\csvloop{`⟨*options*⟩`}`

Usually, `\csvreader` may be preferred instead of `\csvloop`. `\csvreader` is based on `\csvloop` which takes a mandatory list of ⟨*options*⟩ in key value syntax. This list of ⟨*options*⟩ controls the total processing. Especially, it has to contain the CSV file name.

```
\csvloop{file={grade.csv}, head to column names, command=\name,
  before reading={List of students:\ },
  late after line={{,}\ }, late after last line=.}
```

List of students: Maier, Huber, Weißbäck, Bauer.

The following `\csvauto...` commands are intended for quick data overview with limited formatting potential. See Subsection 3.5 on page 20 for the general table options in combination with `\csvreader` → P. 8 and `\csvloop` → P. 8.

`\csvautotabular[`⟨*options*⟩`]{`⟨*file name*⟩`}`

`\csvautotabular` is an abbreviation for the application of the option key `/csv/autotabular` → P. 20 together with other ⟨*options*⟩ to `\csvloop` → P. 8. This macro reads the whole CSV file denoted by ⟨*file name*⟩ with an automated formatting.

```
\csvautotabular{grade.csv}
```

name	givenname	matriculation	gender	grade
Maier	Hans	12345	m	1.0
Huber	Anna	23456	f	2.3
Weißbäck	Werner	34567	m	5.0
Bauer	Maria	19202	f	3.3

```
\csvautotabular[filter equal={\csvcoliv}{f}]{grade.csv}
```

name	givenname	matriculation	gender	grade
Huber	Anna	23456	f	2.3
Bauer	Maria	19202	f	3.3

`\csvautolongtable[`⟨*options*⟩`]{`⟨*file name*⟩`}`

`csvautolongtable` is an abbreviation for the application of the option key `/csv/autolongtable` → P. 20 together with other ⟨*options*⟩ to `\csvloop` → P. 8. This macro reads the whole CSV file denoted by ⟨*file name*⟩ with an automated formatting. For application, the package `longtable` is required which has to be loaded in the preamble.

```
\csvautolongtable{grade.csv}
```

name	givenname	matriculation	gender	grade
Maier	Hans	12345	m	1.0
Huber	Anna	23456	f	2.3
Weißbäck	Werner	34567	m	5.0
Bauer	Maria	19202	f	3.3

`\csvautobooktabular`[⟨*options*⟩]{⟨*file name*⟩}

`\csvautobooktabular` is an abbreviation for the application of the option key `/csv/autobooktabular` → P. 20 together with other ⟨*options*⟩ to `\csvloop` → P. 8. This macro reads the whole CSV file denoted by ⟨*file name*⟩ with an automated formatting. For application, the package `booktabs` is required which has to be loaded in the preamble.

```
\csvautobooktabular{grade.csv}
```

name	givenname	matriculation	gender	grade
Maier	Hans	12345	m	1.0
Huber	Anna	23456	f	2.3
Weißbäck	Werner	34567	m	5.0
Bauer	Maria	19202	f	3.3

`\csvautobooklongtable`[⟨*options*⟩]{⟨*file name*⟩}

`csvautobooklongtable` is an abbreviation for the application of the option key `/csv/autobooklongtable` → P. 20 together with other ⟨*options*⟩ to `\csvloop` → P. 8. This macro reads the whole CSV file denoted by ⟨*file name*⟩ with an automated formatting. For application, the packages `booktabs` and `longtable` are required which have to be loaded in the preamble.

```
\csvautobooklongtable{grade.csv}
```

name	givenname	matriculation	gender	grade
Maier	Hans	12345	m	1.0
Huber	Anna	23456	f	2.3
Weißbäck	Werner	34567	m	5.0
Bauer	Maria	19202	f	3.3

`\csvset{`⟨*options*⟩`}`

Sets ⟨*options*⟩ for every following `\csvreader` → P. 8 and `\csvloop` → P. 8. For example, this command may be used for style definitions.

```
\csvset{grade list/.style=
    {column names={name=\name,givenname=\firstname,grade=\grade}},
  passed/.style={filter not strcmp={\grade}{5.0}} }

The following students passed the test in mathematics:
\csvreader[grade list,passed]{grade.csv}{}{\firstname\ \name\ (\grade); }%
```

The following students passed the test in mathematics: Hans Maier (1.0); Anna Huber (2.3); Maria Bauer (3.3);

`\csvstyle{`⟨*key*⟩`}{`⟨*options*⟩`}`

Abbreviation for `\csvset{`⟨*key*⟩`/.style={`⟨*options*⟩`}}` to define a new style.

`\csvnames{`⟨*key*⟩`}{`⟨*assignments*⟩`}`

Abbreviation for `\csvset{`⟨*key*⟩`/.style={column names={`⟨*assignments*⟩`}}}` to define additional ⟨*assignments*⟩ of macros to columns.

```
\csvnames{grade list}{name=\name,givenname=\firstname,grade=\grade}
\csvstyle{passed}{filter not strcmp={\grade}{5.0}}

The following students passed the test in mathematics:
\csvreader[grade list,passed]{grade.csv}{}{\firstname\ \name\ (\grade); }%
```

The following students passed the test in mathematics: Hans Maier (1.0); Anna Huber (2.3); Maria Bauer (3.3);

`\csvheadset{`⟨*assignments*⟩`}`

For some special cases, this command can be used to change the ⟨*assignments*⟩ of macros to columns during execution of `\csvreader` → P. 8 and `\csvloop` → P. 8.

```
\csvreader{grade.csv}{}%
  { \csvheadset{name=\n} \fbox{\n}
    \csvheadset{givenname=\n} \ldots\ \fbox{\n}  }%
```

Maier … Hans Huber … Anna Weißbäck … Werner Bauer … Maria

`\csviffirstrow{⟨then macros⟩}{⟨else macros⟩}`

Inside the command list of `\csvreader` → P. 8, the ⟨*then macros*⟩ are executed for the first data line, and the ⟨*else macros*⟩ are executed for all following lines.

```
\csvreader[tabbing, head to column names, table head=\hspace*{3cm}\=\kill]%
  {grade.csv}{}%
  {\givenname~\name \> (\csviffirstrow{first entry!!}{following entry})}
```

Hans Maier	(first entry!!)
Anna Huber	(following entry)
Werner Weißbäck	(following entry)
Maria Bauer	(following entry)

`\csvifoddrow{⟨then macros⟩}{⟨else macros⟩}`

Inside the command list of `\csvreader` → P. 8, the ⟨*then macros*⟩ are executed for odd-numbered data lines, and the ⟨*else macros*⟩ are executed for even-numbered lines.

```
\csvreader[head to column names,tabular=|l|l|l|l|,
  table head=\hline\bfseries \# & \bfseries Name & \bfseries Grade\\\hline,
  table foot=\hline]{grade.csv}{}{%
    \csvifoddrow{\slshape\thecsvrow & \slshape\name, \givenname & \slshape\grade}%
    {\bfseries\thecsvrow & \bfseries\name, \givenname & \bfseries\grade}}
```

#	Name	Grade
1	*Maier, Hans*	*1.0*
2	**Huber, Anna**	**2.3**
3	*Weißbäck, Werner*	*5.0*
4	**Bauer, Maria**	**3.3**

The `\csvifoddrow` macro may be used for striped tables:

```
% This example needs the xcolor package
\csvreader[head to column names,tabular=rlcc,
  table head=\hline\rowcolor{red!50!black}\color{white}\# & \color{white}Person
    & \color{white}Matr.~No. & \color{white}Grade,
  late after head=\\\hline\rowcolor{yellow!50},
  late after line=\csvifoddrow{\\\rowcolor{yellow!50}}{\\\rowcolor{red!25}}]%
  {grade.csv}{}%
  {\thecsvrow & \givenname~\name & \matriculation & \grade}%
```

#	Person	Matr. No.	Grade
1	Hans Maier	12345	1.0
2	Anna Huber	23456	2.3
3	Werner Weißbäck	34567	5.0
4	Maria Bauer	19202	3.3

Alternatively, `\rowcolors` from the `xcolor` package can be used for this purpose:

```
% This example needs the xcolor package
\csvreader[tabular=rlcc, before table=\rowcolors{2}{red!25}{yellow!50},
  table head=\hline\rowcolor{red!50!black}\color{white}\# & \color{white}Person
    & \color{white}Matr.~No. & \color{white}Grade\\\hline,
  head to column names]{grade.csv}{}%
  {\thecsvrow & \givenname~\name & \matriculation & \grade}%
```

#	Person	Matr. No.	Grade
1	Hans Maier	12345	1.0
2	Anna Huber	23456	2.3
3	Werner Weißbäck	34567	5.0
4	Maria Bauer	19202	3.3

\csvfilteraccept

All following consistent data lines will be accepted and processed. This command overwrites all previous filter settings and may be used inside /csv/full filter → P. 19 to implement an own filtering rule together with \csvfilterreject.

```
\csvreader[autotabular,
  full filter=\ifcsvstrcmp{\csvcoliv}{m}{\csvfilteraccept}{\csvfilterreject}
  ]{grade.csv}{}{\csvlinetotablerow}%
```

name	givenname	matriculation	gender	grade
Maier	Hans	12345	m	1.0
Weißbäck	Werner	34567	m	5.0

\csvfilterreject

All following data lines will be ignored. This command overwrites all previous filter settings.

\csvline

This macro contains the current and unprocessed data line.

```
\csvreader[no head, tabbing, table head=\textit{line XX:}\=\kill]%
  {grade.csv}{}{\textit{line \thecsvrow:} \> \csvline}%
```

line 1: name,givenname,matriculation,gender,grade
line 2: Maier,Hans,12345,m,1.0
line 3: Huber,Anna,23456,f,2.3
line 4: Weißbäck,Werner,34567,m,5.0
line 5: Bauer,Maria,19202,f,3.3

\thecsvrow

Typesets the current data line number. This is the current number of accepted data lines without the header line. The LaTeX counter csvrow can be addressed directly in the usual way, e. g. by \roman{csvrow}.

\thecsvinputline

Typesets the current file line number. This is the current number of all data lines including the header line. The LaTeX counter csvinputline can be addressed directly in the usual way, e. g. by \roman{csvinputline}.

```
\csvreader[no head, filter test=\ifnumequal{\thecsvinputline}{3}]%
  {grade.csv}{}%
  {The line with number \thecsvinputline\ contains: \csvline}%
```

The line with number 3 contains: Huber,Anna,23456,f,2.3

U 2016-07-01

\csvlinetotablerow

Typesets the current processed data line with & between the entries.

3 Option Keys

For the ⟨*options*⟩ in `\csvreader` →P. 8 respectively `\csvloop` →P. 8 the following `pgf` keys can be applied. The key tree path `/csv/` is not to be used inside these macros.

3.1 Command Definition

`/csv/before reading`=⟨*code*⟩ (no default, initially empty)

Sets the ⟨*code*⟩ to be executed before the CSV file is processed.

`/csv/after head`=⟨*code*⟩ (no default, initially empty)

Sets the ⟨*code*⟩ to be executed after the header line is read.

`/csv/before filter`=⟨*code*⟩ (no default, initially empty)

Sets the ⟨*code*⟩ to be executed after reading and consistency checking of a data line. They are executed before any filter condition is checked, see `/csv/filter` →P. 19. Also see `/csv/full filter` →P. 19.

`/csv/late after head`=⟨*code*⟩ (no default, initially empty)

Sets the ⟨*code*⟩ to be executed after reading and disassembling of the first accepted data line. They are executed before further processing of this line.

`/csv/late after line`=⟨*code*⟩ (no default, initially empty)

Sets the ⟨*code*⟩ to be executed after reading and disassembling of the next accepted data line (after `/csv/before filter`). They are executed before further processing of this next line. `late after line` overwrites `late after first line` and `late after last line`. Note that table options like `/csv/tabular` →P. 20 set this key to `\\` automatically.

`/csv/late after first line`=⟨*code*⟩ (no default, initially empty)

Sets the ⟨*code*⟩ to be executed after reading and disassembling of the second accepted data line instead of `/csv/late after line`. This key has to be set after `late after line`.

`/csv/late after last line`=⟨*code*⟩ (no default, initially empty)

Sets the ⟨*code*⟩ to be executed after processing of the last accepted data line instead of `/csv/late after line`. This key has to be set after `late after line`.

`/csv/before line`=⟨*code*⟩ (no default, initially empty)

Sets the ⟨*code*⟩ to be executed after `/csv/late after line` and before `/csv/command`. `before line` overwrites `before first line`.

`/csv/before first line`=⟨*code*⟩ (no default, initially empty)

Sets the ⟨*code*⟩ to be executed instead of `/csv/before line` for the first accepted data line. This key has to be set after `before line`.

`/csv/command`=⟨*code*⟩ (no default, initially `\csvline`)

Sets the ⟨*code*⟩ to be executed for every accepted data line. They are executed between `/csv/before line` and `/csv/after line`.

`/csv/after line`=⟨*code*⟩ (no default, initially empty)

Sets the ⟨*code*⟩ to be executed for every accepted data line after `/csv/command`. `after line` overwrites `after first line`.

`/csv/after first line`=⟨*code*⟩ (no default, initially empty)

Sets the ⟨*code*⟩ to be executed instead of `/csv/after line` for the first accepted data line. This key has to be set after `after line`.

`/csv/after reading`=⟨*code*⟩ (no default, initially empty)

Sets the ⟨*code*⟩ to be executed after the CSV file is processed.

```
\csvreader[
  before reading        = \meta{before reading}\\,
  after head            = \meta{after head},
  before filter         = \\\meta{before filter},
  late after head       = \meta{late after head},
  late after line       = \meta{late after line},
  late after first line = \meta{late after first line},
  late after last line  = \\\meta{late after last line},
  before line           = \meta{before line},
  before first line     = \meta{before first line},
  after line            = \meta{after line},
  after first line      = \meta{after first line},
  after reading         = \\\meta{after reading}
    ]{grade.csv}{name=\name}{\textbf{\name}}%
```

⟨*before reading*⟩
⟨*after head*⟩
⟨*before filter*⟩⟨*late after head*⟩⟨*before first line*⟩**Maier**⟨*after first line*⟩
⟨*before filter*⟩⟨*late after first line*⟩⟨*before line*⟩**Huber**⟨*after line*⟩
⟨*before filter*⟩⟨*late after line*⟩⟨*before line*⟩**Weißbäck**⟨*after line*⟩
⟨*before filter*⟩⟨*late after line*⟩⟨*before line*⟩**Bauer**⟨*after line*⟩
⟨*late after last line*⟩
⟨*after reading*⟩

Additional command definition keys are provided for the supported tables, see Section 3.5 from page 20.

3.2 Header Processing and Column Name Assignment

`/csv/head=true|false` (default `true`, initially `true`)

If this key is set, the first line of the CSV file is treated as a header line which can be used for column name assignments.

`/csv/no head` (no value)

Abbreviation for `head=false`, i. e. the first line of the CSV file is treated as data line. Note that this option cannot be used in combination with `\csvautotabular` → P. 9, `/csv/autotabular` → P. 20, and similar automated commands/options. See Section 5.5 on page 40 for assistance.

`/csv/column names=`⟨*assignments*⟩ (no default, initially empty)

Adds some new ⟨*assignments*⟩ of macros to columns in key value syntax. Existing assignments are kept.

`/csv/column names reset` (no value)

Clears all assignments of macros to columns.

`/csv/head to column names=true|false` (default `true`, initially `false`)

If this key is set, the entries of the header line are used automatically as macro names for the columns. This option can be used only, if the header entries do not contain spaces and special characters to be used as feasible LaTeX macro names. Note that the macro definition is *global* and may therefore override existing macros for the rest of the document. Adding `/csv/head to column names prefix` may help to avoid unwanted overrides.

N 2019-07-16 `/csv/head to column names prefix=`⟨*text*⟩ (no default, initially empty)

The given ⟨*text*⟩ is prefixed to the name of all macros generated by `/csv/head to column names`. For example, if you use the settings

```
head to column names,
head to column names prefix=MY,
```

a header entry `section` will generate the corresponding macro `\MYsection` instead of destroying the standard LaTeX `\section` macro.

3.3 Consistency Check

`/csv/check column count=true|false` (default `true`, initially `true`)

This key defines, wether the number of entries in a data line is checked against an expected value or not.
If `true`, every non consistent line is ignored without announcement.
If `false`, every line is accepted and may produce an error during further processing.

`/csv/no check column count` (no value)

Abbreviation for `check column count=false`.

`/csv/column count=`⟨*number*⟩ (no default)

Sets the ⟨*number*⟩ of feasible entries per data line. This setting is only useful in connection with `/csv/no head` → P. 16, since ⟨*number*⟩ would be replaced by the number of entries in the header line otherwise.

`/csv/on column count error=`⟨*code*⟩ (no default, initially empty)

⟨*code*⟩ to be executed for unfeasible data lines.

`/csv/warn on column count error` (style, no value)

Display of a warning for unfeasible data lines.

3.4 Filtering

N 2016-07-01 **/csv/filter test=**⟨*condition*⟩ (no default)

Only data lines which fulfill a logical ⟨*condition*⟩ are accepted. For the ⟨*condition*⟩, every single test normally employed like

```
\iftest{some testing}{true}{false}
```

can be used as

```
filter test=\iftest{some testing},
```

For `\iftest`, tests from the `etoolbox` package like `\ifnumcomp`, `\ifdimgreater`, etc. and from Section 4 on page 29 can be used.

```
\csvreader[head to column names,tabular=llll,
  table head=\toprule & \bfseries Name & \bfseries Matr & \bfseries Grade\\\midrule,
  table foot=\bottomrule,
  %>> list only matriculation numbers greater than 20000 <<
  filter test=\ifnumgreater{\matriculation}{20000},
  ]{grade.csv}{}{%
    \thecsvrow & \slshape\name, \givenname & \matriculation & \grade}
```

	Name	Matr	Grade
1	*Huber, Anna*	23456	2.3
2	*Weißbäck, Werner*	34567	5.0

/csv/filter strcmp={⟨*stringA*⟩}{⟨*stringB*⟩} (style, no default)

Only lines where ⟨*stringA*⟩ and ⟨*stringB*⟩ are equal after expansion are accepted. The implementation is done with `\ifcsvstrcmp` → P. 29.

/csv/filter not strcmp={⟨*stringA*⟩}{⟨*stringB*⟩} (style, no default)

Only lines where ⟨*stringA*⟩ and ⟨*stringB*⟩ are not equal after expansion are accepted. The implementation is done with `\ifcsvnotstrcmp` → P. 29.

N 2016-07-01 **/csv/filter expr=**⟨*condition*⟩ (no default)

Only data lines which fulfill a logical ⟨*condition*⟩ are accepted. For the ⟨*condition*⟩, every boolean expression from the `etoolbox` package is feasible. To preprocess the data line before testing the ⟨*condition*⟩, the option key `/csv/before filter` → P. 14 can be used.

```
\csvreader[head to column names,tabular=llll,
  table head=\toprule & \bfseries Name & \bfseries Matr & \bfseries Grade\\\midrule,
  table foot=\bottomrule,
  %>> list only matriculation numbers greater than 20000
  %   and grade less than 4.0 <<
  filter expr={     test{\ifnumgreater{\matriculation}{20000}}
                and test{\ifdimless{\grade pt}{4.0pt}}          },
  ]{grade.csv}{}{%
    \thecsvrow & \slshape\name, \givenname & \matriculation & \grade}
```

	Name	Matr	Grade
1	*Huber, Anna*	23456	2.3

N 2016-07-01 **/csv/filter ifthen=⟨*condition*⟩** (no default)

Only data lines which fulfill a logical ⟨*condition*⟩ are accepted. For the ⟨*condition*⟩, every term from the `ifthen` package is feasible. To preprocess the data line before testing the ⟨*condition*⟩, the option key /csv/before filter → P. 14 can be used.

```
\csvreader[head to column names,tabular=llll,
  table head=\toprule & \bfseries Name & \bfseries Matr & \bfseries Grade\\\midrule,
  table foot=\bottomrule,
  %>> list only female persons <<
  filter ifthen=\equal{\gender}{f},
  ]{grade.csv}{}{%
    \thecsvrow & \slshape\name, \givenname & \matriculation & \grade}
```

	Name	Matr	Grade
1	*Huber, Anna*	23456	2.3
2	*Bauer, Maria*	19202	3.3

/csv/filter=⟨*condition*⟩ (no default)

Alias for /csv/filter ifthen.

/csv/filter equal={⟨*stringA*⟩}{⟨*stringB*⟩} (style, no default)

Only lines where ⟨*stringA*⟩ and ⟨*stringB*⟩ are equal after expansion are accepted. The implementation is done with the `ifthen` package.

/csv/filter not equal={⟨*stringA*⟩}{⟨*stringB*⟩} (style, no default)

Only lines where ⟨*stringA*⟩ and ⟨*stringB*⟩ are not equal after expansion are accepted. The implementation is done with the `ifthen` package.

/csv/no filter (no value, initially set)

Clears a set filter.

/csv/filter accept all (no value, initially set)

Alias for `no filter`. All consistent data lines are accepted.

/csv/filter reject all (no value)

All data line are ignored.

N 2016-07-01 **/csv/full filter=⟨*code*⟩** (no default)

Technically, this key is an alias for /csv/before filter → P. 14. Philosophically, /csv/before filter → P. 14 computes something before a filter condition is set, but /csv/full filter should implement the full filtering. Especially, \csvfilteraccept → P. 13 or \csvfilterreject → P. 13 *should* be set inside the ⟨*code*⟩.

```
\csvreader[head to column names,tabular=llll,
  table head=\toprule & \bfseries Name & \bfseries Matr & \bfseries Grade\\\midrule,
  table foot=\bottomrule,
  %>> list only matriculation numbers greater than 20000
  %   and grade less than 4.0 <<
  full filter=\ifnumgreater{\matriculation}{20000}
                {\ifdimless{\grade pt}{4.0pt}{\csvfilteraccept}{\csvfilterreject}}
                {\csvfilterreject},
  ]{grade.csv}{}{%
    \thecsvrow & \slshape\name, \givenname & \matriculation & \grade}
```

	Name	Matr	Grade
1	*Huber, Anna*	23456	2.3

3.5 Table Support

`/csv/tabular`=⟨*table format*⟩ (style, no default)

Surrounds the CSV processing with `\begin{tabular}{`⟨*table format*⟩`}` at begin and with `\end{tabular}` at end. Additionally, the commands defined by the key values of `/csv/before table`, `/csv/table head`, `/csv/table foot`, and `/csv/after table` are executed at the appropriate places.

`/csv/centered tabular`=⟨*table format*⟩ (style, no default)

Like `/csv/tabular` but inside an additional `center` environment.

`/csv/longtable`=⟨*table format*⟩ (style, no default)

Like `/csv/tabular` but for the `longtable` environment. This requires the package `longtable` (not loaded automatically).

`/csv/tabbing` (style, no value)

Like `/csv/tabular` but for the `tabbing` environment.

`/csv/centered tabbing` (style, no value)

Like `/csv/tabbing` but inside an additional `center` environment.

`/csv/no table` (style, no value)

Deactivates `tabular`, `longtable`, and `tabbing`.

`/csv/before table`=⟨*code*⟩ (no default, initially empty)

Sets the ⟨*code*⟩ to be executed before `\begin{tabular}` or before `\begin{longtable}` or before `\begin{tabbing}`, respectively.

`/csv/table head`=⟨*code*⟩ (no default, initially empty)

Sets the ⟨*code*⟩ to be executed after `\begin{tabular}` or after `\begin{longtable}` or after `\begin{tabbing}`, respectively.

`/csv/table foot`=⟨*code*⟩ (no default, initially empty)

Sets the ⟨*code*⟩ to be executed before `\end{tabular}` or before `\end{longtable}` or before `\end{tabbing}`, respectively.

`/csv/after table`=⟨*code*⟩ (no default, initially empty)

Sets the ⟨*code*⟩ to be executed after `\end{tabular}` or after `\end{longtable}` or after `\end{tabbing}`, respectively.

The following `auto` options are the counterparts for the respective quick overview commands like `\csvautotabular` → P. 9. They are listed for completeness, but are unlikely to be used directly.

`/csv/autotabular`=⟨*file name*⟩ (no default)

Reads the whole CSV file denoted ⟨*file name*⟩ with an automated formatting.

`/csv/autolongtable`=⟨*file name*⟩ (no default)

Reads the whole CSV file denoted ⟨*file name*⟩ with an automated formatting using the required `longtable` package.

`/csv/autobooktabular`=⟨*file name*⟩ (no default)

Reads the whole CSV file denoted ⟨*file name*⟩ with an automated formatting using the required `booktabs` package.

`/csv/autobooklongtable`=⟨*file name*⟩ (no default)

Reads the whole CSV file denoted ⟨*file name*⟩ with an automated formatting using the required `booktabs` and `longtable` packages.

3.6 Special Characters

Be default, the CSV content is treated like normal LaTeX text, see Subsection 5.3 on page 36. But, TeX special characters of the CSV content may also be interpreted as normal characters, if one or more of the following options are used.

`/csv/respect tab=true|false` (default `true`, initially `false`)

If this key is set, every tabulator sign inside the CSV content is a normal character.

`/csv/respect percent=true|false` (default `true`, initially `false`)

If this key is set, every percent sign `"%"` inside the CSV content is a normal character.

`/csv/respect sharp=true|false` (default `true`, initially `false`)

If this key is set, every sharp sign `"#"` inside the CSV content is a normal character.

`/csv/respect dollar=true|false` (default `true`, initially `false`)

If this key is set, every dollar sign `"$"` inside the CSV content is a normal character.

`/csv/respect and=true|false` (default `true`, initially `false`)

If this key is set, every and sign `"&"` inside the CSV content is a normal character.

`/csv/respect backslash=true|false` (default `true`, initially `false`)

If this key is set, every backslash sign `"\"` inside the CSV content is a normal character.

`/csv/respect underscore=true|false` (default `true`, initially `false`)

If this key is set, every underscore sign `"_"` inside the CSV content is a normal character.

`/csv/respect tilde=true|false` (default `true`, initially `false`)

If this key is set, every tilde sign `"~"` inside the CSV content is a normal character.

`/csv/respect circumflex=true|false` (default `true`, initially `false`)

If this key is set, every circumflex sign `"^"` inside the CSV content is a normal character.

`/csv/respect leftbrace=true|false` (default `true`, initially `false`)

If this key is set, every left brace sign `"{"` inside the CSV content is a normal character.

`/csv/respect rightbrace=true|false` (default `true`, initially `false`)

If this key is set, every right brace sign `"}"` inside the CSV content is a normal character.

`/csv/respect all` (style, no value, initially unset)

Set all special characters from above to normal characters. This means a quite verbatim interpretation of the CSV content.

`/csv/respect none` (style, no value, initially set)

Do not change any special character from above to normal character.

3.7 Separators

`/csv/separator`=⟨*sign*⟩ (no default, initially `comma`)

Sets the ⟨*sign*⟩ which is treates as separator between the data values of a data line. Feasible values are:

- `comma`: This is the initial value with ',' as separator.
- `semicolon`: Sets the separator to ';'.

```
% \usepackage{tcolorbox} for tcbverbatimwrite
\begin{tcbverbatimwrite}{testsemi.csv}
  name;givenname;matriculation;gender;grade
  Maier;Hans;12345;m;1.0
  Huber;Anna;23456;f;2.3
  Weißbäck;Werner;34567;m;5.0
\end{tcbverbatimwrite}

\csvautobooktabular[separator=semicolon]{testsemi.csv}
```

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

- `pipe`: Sets the separator to '|'.

```
% \usepackage{tcolorbox} for tcbverbatimwrite
\begin{tcbverbatimwrite}{pipe.csv}
  name|givenname|matriculation|gender|grade
  Maier|Hans|12345|m|1.0
  Huber|Anna|23456|f|2.3
  Weißbäck|Werner|34567|m|5.0
\end{tcbverbatimwrite}

\csvautobooktabular[separator=pipe]{pipe.csv}
```

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

- `tab`: Sets the separator to the tabulator sign. Automatically, `/csv/respect tab` → P. 21 is set also.

3.8 Miscellaneous

/csv/every csv (style, initially empty)

A style definition which is used for every following CSV file. This definition can be overwritten with user code.

```
% Sets a warning message for unfeasible data lines.
\csvset{every csv/.style={warn on column count error}}
% Alternatively:
\csvstyle{every csv}{warn on column count error}
```

/csv/default (style)

A style definition which is used for every following CSV file which resets all settings to default values[4]. This key should not be used or changed by the user if there is not a really good reason (and you know what you do).

/csv/file=⟨*file name*⟩ (no default, initially `unknown.csv`)

Sets the ⟨*file name*⟩ of the CSV file to be processed.

/csv/preprocessed file=⟨*file name*⟩ (no default, initially `\jobname_sorted.csv`)

Sets the ⟨*file name*⟩ of the CSV file which is the output of a preprocessor.

/csv/preprocessor=⟨*macro*⟩ (no default)

Defines a preprocessor for the given CSV file. The ⟨*macro*⟩ has to have two mandatory arguments. The first argument is the original CSV file which is set by /csv/file. The second argument is the preprocessed CSV file which is set by /csv/preprocessed file.

Typically, the ⟨*macro*⟩ may call an external program which preprocesses the original CSV file (e. g. sorting the file) and creates the preprocessed CSV file. The later file is used by \csvreader → P. 8 or \csvloop → P. 8.

```
\newcommand{\mySortTool}[2]{%
  % call to an external program to sort file #1 with resulting file #2
}

\csvreader[%
  preprocessed file=\jobname_sorted.csv,
  preprocessor=\mySortTool,
  ]{some.csv}{}{%
  % do something
}
```

See Subsection 3.9 on page 24 for a concrete sorting preprocessing implemented with an external tool.

/csv/no preprocessing (style, no value, initially set)

Clears any preprocessing, i. e. preprocessing is switched of.

[4] `default` is used because of the global nature of most settings.

3.9 Sorting

TEX/LATEX was not born under a sorting planet. `csvsimple-legacy` provides no sorting of data lines by LATEX-methods since sorting can be done much faster and much better by external tools.

First, one should consider the appropriate *place* for sorting:

- CSV files may be sorted by a tool *before* the LATEX document is processed at all. If the CSV data is not likely to change, this is the most efficient method.
- CSV files may be sorted by a tool every time before the LATEX document is compiled. This could be automated by a shell script or some processing tool like `arara`.
- CSV files may be sorted on-the-fly by a tool during compilation of a LATEX document. This is the most elegant but not the most efficient way.

The first two methods are decoupled from anything concerning `csvsimple-legacy`. For the third method, the `/csv/preprocessor` → P. 23 option is made for. This allows to access an external tool for sorting. *Which tool* is your choice.

CSV-Sorter was written as a companion tool for `csvsimple`. It is an open source Java command-line tool for sorting CSV files, available at
http://T-F-S.github.io/csvsorter/ or https://github.com/T-F-S/csvsorter

It can be used for all three sorting approaches described above. There is special support for on-the-fly sorting with **CSV-Sorter** using the following options.

1. **To use the sorting options, you have to install CSV-Sorter before! `csvsimple` v1.12 or newer needs CSV-Sorter v0.94 of newer!**
2. **You have to give permission to call external tools during compilation, i. e. the command-line options for `latex` have to include `-shell-escape`.**

`/csv/csvsorter command=`⟨*system command*⟩ (no default, initially `csvsorter`)

The ⟨*system command*⟩ specifies the system call for **CSV-Sorter** (without the options). If **CSV-Sorter** was completely installed following its documentation, there is nothing to change here. If the `csvsorter.jar` file is inside the same directory as the LATEX source file, you may configure:

```
\csvset{csvsorter command=java -jar csvsorter.jar}
```

`/csv/csvsorter configpath=`⟨*path*⟩ (no default, initially `.`)

Sorting with **CSV-Sorter** is done using XML configuration files. If these files are not stored inside the same directory as the LATEX source file, a ⟨*path*⟩ to access them can be configured:

```
\csvset{csvsorter configpath=xmlfiles}
```

Here, the configuration files would be stored in a subdirectory named `xmlfiles`.

`/csv/csvsorter log=`⟨*file name*⟩ (no default, initially `csvsorter.log`)

Sets the log file of **CSV-Sorter** to the given ⟨*file name*⟩.

```
\csvset{csvsorter log=outdir/csvsorter.log}
```

Here, the log file is written to a subdirectory named `outdir`.

`/csv/csvsorter token=`⟨*file name*⟩ (no default, initially `\jobname.csvtoken`)

Sets ⟨*file name*⟩ as token file. This is an auxiliary file which communicates the success of **CSV-Sorter** to `csvsimple`.

```
\csvset{csvsorter log=outdir/\jobname.csvtoken}
```

Here, the token file is written to a subdirectory named `outdir`.

`/csv/sort by=`⟨*file name*⟩ (style, initially unset)

The ⟨*file name*⟩ denotes an XML configuration file for **CSV-Sorter**. Setting this option inside `\csvreader` →P. 8 or `\csvloop` →P. 8 will issue a system call to **CSV-Sorter**.

- **CSV-Sorter** uses the given CSV file as input file.
- **CSV-Sorter** uses ⟨*file name*⟩ as configuration file.
- The output CSV file is denoted by `/csv/preprocessed file` →P. 23 which is by default `\jobname_sorted.csv`. This output file is this actual file processed by `\csvreader` →P. 8 or `\csvloop` →P. 8.
- **CSV-Sorter** also generates a log file denoted by `/csv/csvsorter log` →P. 24 which is by default `csvsorter.log`.

First example: To sort our example `grade.csv` file according to `name` and `givenname`, we use the following XML configuration file. Since **CSV-Sorter** uses double quotes as default brackets for column values, we remove bracket recognition to avoid a clash with the escaped umlauts of the example CSV file.

Configuration file «`namesort.xml`»

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

```
% \usepackage{booktabs}
\csvreader[sort by=namesort.xml,
  head to column names,
  tabular=>{\color{red}}lllll,
  table head=\toprule Name & Given Name & Matriculation & Gender & Grade\\\midrule,
  table foot=\bottomrule]
  {grade.csv}{}{\csvlinetotablerow}
```

Name	Given Name	Matriculation	Gender	Grade
Bauer	Maria	19202	f	3.3
Huber	Anna	23456	f	2.3
Maier	Hans	12345	m	1.0
Weißbäck	Werner	34567	m	5.0

Second example: To sort our example `grade.csv` file according to `grade`, we use the following XML configuration file. Further, persons with the same `grade` are sorted by `name` and `givenname`. Since **CSV-Sorter** uses double quotes as default brackets for column values, we remove bracket recognition to avoid a clash with the escaped umlauts of the example CSV file.

Configuration file «gradesort.xml»

```
<?xml version="1.0" encoding="UTF-8"?>
<csv>
  <bracket empty="true" />
  <sortlines>
    <column name="grade" order="ascending" type="double"/>
    <column name="name" order="ascending" type="string"/>
    <column name="givenname" order="ascending" type="string"/>
  </sortlines>
</csv>
```

```
% \usepackage{booktabs}
\csvreader[sort by=gradesort.xml,
  head to column names,
  tabular=llll>{\color{red}}l,
  table head=\toprule Name & Given Name & Matriculation & Gender & Grade\\\midrule,
  table foot=\bottomrule]
  {grade.csv}{}{\csvlinetotablerow}
```

Name	Given Name	Matriculation	Gender	Grade
Maier	Hans	12345	m	1.0
Huber	Anna	23456	f	2.3
Bauer	Maria	19202	f	3.3
Weißbäck	Werner	34567	m	5.0

Third example: To generate a matriculation/grade list, we sort our example `grade.csv` file using the following XML configuration file. Again, since **CSV-Sorter** uses double quotes as default brackets for column values, we remove bracket recognition to avoid a clash with the escaped umlauts of the example CSV file.

Configuration file «matriculationsort.xml»

```
<?xml version="1.0" encoding="UTF-8"?>
<csv>
  <bracket empty="true" />
  <sortlines>
    <column name="matriculation" order="ascending" type="integer"/>
  </sortlines>
</csv>
```

```
% \usepackage{booktabs}
\csvreader[sort by=matriculationsort.xml,
  head to column names,
  tabular=>{\color{red}}ll,
  table head=\toprule Matriculation & Grade\\\midrule,
  table foot=\bottomrule]
  {grade.csv}{}{\matriculation & \grade}
```

Matriculation	Grade
12345	1.0
19202	3.3
23456	2.3
34567	5.0

`/csv/new sorting rule={`⟨*name*⟩`}{`⟨*file name*⟩`}` (style, initially unset)

This is a convenience option to generate a new shortcut for often used `/csv/sort by` → P. 25 applications. It also adds a more semantic touch. The new shortcut option is `sort by` ⟨*name*⟩ which expands to `sort by={`⟨*file name*⟩`}`.

Consider the following example:

```
\csvautotabular[sort by=namesort.xml]{grade.csv}
```

name	givenname	matriculation	gender	grade
Bauer	Maria	19202	f	3.3
Huber	Anna	23456	f	2.3
Maier	Hans	12345	m	1.0
Weißbäck	Werner	34567	m	5.0

A good place for setting up a new sorting rule would be inside the preamble:

```
\csvset{new sorting rule={name}{namesort.xml}}
```

Now, we can use the new rule:

```
\csvautotabular[sort by name]{grade.csv}
```

name	givenname	matriculation	gender	grade
Bauer	Maria	19202	f	3.3
Huber	Anna	23456	f	2.3
Maier	Hans	12345	m	1.0
Weißbäck	Werner	34567	m	5.0

4 String Tests

The following string tests are complementing the string tests from the `etoolbox` package. They all do the same, i.e., comparing expanded strings for equality.

- `\ifcsvstrcmp` is the most efficient method, because it uses native compiler string comparison (if available).
- `\ifcsvstrequal` does not rely on a compiler. It also is the fallback implementation for `\ifcsvstrcmp`, if there is no native comparison method.
- `\ifcsvprostrequal` is possibly more failsafe than the other two string tests. It may be used, if strings contain dirty things like `\textbf{A}`.

N 2016-07-01 **\ifcsvstrcmp**{⟨*stringA*⟩}{⟨*stringB*⟩}{⟨*true*⟩}{⟨*false*⟩}

Compares two strings and executes ⟨*true*⟩ if they are equal, and ⟨*false*⟩ otherwise. The comparison is done using `\pdfstrcmp`, if compilation is done with pdfLaTeX. The comparison is done using `\pdf@strcmp`, if the package `pdftexcmds` is loaded and compilation is done with luaLaTeX or XeLaTeX. Otherwise, `\ifcsvstrcmp` is identical to `\ifcsvstrequal`. This command cannot be used inside the preamble.

N 2016-07-01 **\ifcsvnotstrcmp**{⟨*stringA*⟩}{⟨*stringB*⟩}{⟨*true*⟩}{⟨*false*⟩}

Compares two strings and executes ⟨*true*⟩ if they are *not* equal, and ⟨*false*⟩ otherwise. The implementation uses `\ifcsvstrcmp`.

N 2016-07-01 **\ifcsvstrequal**{⟨*stringA*⟩}{⟨*stringB*⟩}{⟨*true*⟩}{⟨*false*⟩}

Compares two strings and executes ⟨*true*⟩ if they are equal, and ⟨*false*⟩ otherwise. The strings are expanded with `\edef` in the test.

N 2016-07-01 **\ifcsvprostrequal**{⟨*stringA*⟩}{⟨*stringB*⟩}{⟨*true*⟩}{⟨*false*⟩}

Compares two strings and executes ⟨*true*⟩ if they are equal, and ⟨*false*⟩ otherwise. The strings are expanded with `\protected@edef` in the test, i.e. parts of the strings which are protected stay unexpanded.

5 Examples

5.1 A Serial Letter

In this example, a serial letter is to be written to all persons with addresses from the following CSV file. Deliberately, the file content is not given in very pretty format.

CSV file «address.csv»

```
name,givenname,gender,degree,street,zip,location,bonus
Maier,Hans,m,,Am Bachweg 17,10010,Hopfingen,20
    % next line with a comma in curly braces
Huber,Erna,f,Dr.,{Moosstraße 32, Hinterschlag},10020,Örtingstetten,30
Weißbäck,Werner,m,Prof. Dr.,Brauallee 10,10030,Klingenbach,40
    % this line is ignored %
  Siebener ,  Franz,m,   ,  Blaumeisenweg 12  , 10040 ,  Pardauz , 50
    % preceding and trailing spaces in entries are removed %
Schmitt,Anton,m,,{\AE{}lfred-Esplanade, T\ae{}g 37}, 10050,\OE{}resung,60
```

Firstly, we survey the file content quickly using `\csvautotabular`. As can be seen, unfeasible lines are ignored automatically.

```
\tiny\csvautotabular{address.csv}
```

name	givenname	gender	degree	street	zip	location	bonus
Maier	Hans	m		Am Bachweg 17	10010	Hopfingen	20
Huber	Erna	f	Dr.	Moosstraße 32, Hinterschlag	10020	Örtingstetten	30
Weißbäck	Werner	m	Prof. Dr.	Brauallee 10	10030	Klingenbach	40
Siebener	Franz	m		Blaumeisenweg 12	10040	Pardauz	50
Schmitt	Anton	m		Ælfred-Esplanade, Tæg 37	10050	Œresung	60

Now, we create the serial letter where every feasible data line produces an own page. Here, we simulate the page by a `tcolorbox` (from the package `tcolorbox`). For the gender specific salutations, an auxiliary macro `\ifmale` is introduced.

```
% this example requires the tcolorbox package
\newcommand{\ifmale}[2]{\ifcsvstrcmp{\gender}{m}{#1}{#2}}

\csvreader[head to column names]{address.csv}{}{%
\begin{tcolorbox}[colframe=DarkGray,colback=White,arc=0mm,width=(\linewidth-2pt)/2,
      equal height group=letter,before=,after=\hfill,fonttitle=\bfseries,
      adjusted title={Letter to \name}]
  \ifcsvstrcmp{\degree}{}{\ifmale{Mr.}{Ms.}}{\degree}~\givenname~\name\\
  \street\\\zip~\location
  \tcblower
  {\itshape Dear \ifmale{Sir}{Madam},}\\
  we are pleased to announce you a bonus value of \bonus\%{}
  which will be delivered to \location\ soon.\\\ldots
\end{tcolorbox}}
```

Letter to Maier

Mr. Hans Maier
Am Bachweg 17
10010 Hopfingen

Dear Sir,
we are pleased to announce you a bonus value of 20% which will be delivered to Hopfingen soon.
...

Letter to Huber

Dr. Erna Huber
Moosstraße 32, Hinterschlag
10020 Örtingstetten

Dear Madam,
we are pleased to announce you a bonus value of 30% which will be delivered to Örtingstetten soon.
...

Letter to Weißbäck

Prof. Dr. Werner Weißbäck
Brauallee 10
10030 Klingenbach

Dear Sir,
we are pleased to announce you a bonus value of 40% which will be delivered to Klingenbach soon.
...

Letter to Siebener

Mr. Franz Siebener
Blaumeisenweg 12
10040 Pardauz

Dear Sir,
we are pleased to announce you a bonus value of 50% which will be delivered to Pardauz soon.
...

Letter to Schmitt

Mr. Anton Schmitt
Ælfred-Esplanade, Tæg 37
10050 Œresung

Dear Sir,
we are pleased to announce you a bonus value of 60% which will be delivered to Œresung soon.
...

5.2 A Graphical Presentation

For this example, we use some artificial statistical data given by a CSV file.

CSV file «data.csv»

```
land,group,amount
Bayern,A,1700
Baden-Württemberg,A,2300
Sachsen,B,1520
Thüringen,A,1900
Hessen,B,2100
```

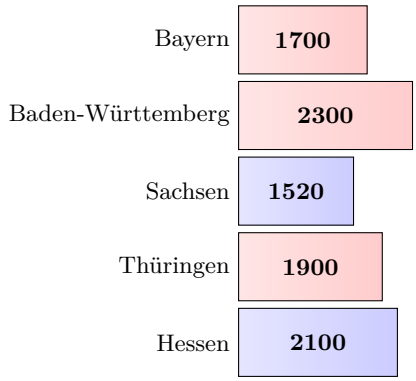
Firstly, we survey the file content using `\csvautobooktabular`.

```
% needs the booktabs package
\csvautobooktabular{data.csv}
```

land	group	amount
Bayern	A	1700
Baden-Württemberg	A	2300
Sachsen	B	1520
Thüringen	A	1900
Hessen	B	2100

The amount values are presented in the following diagram by bars where the group classification is given using different colors.

```
% This example requires the package tikz
\begin{tikzpicture}[Group/A/.style={left color=red!10,right color=red!20},
                    Group/B/.style={left color=blue!10,right color=blue!20}]
\csvreader[head to column names]{data.csv}{}{%
  \begin{scope}[yshift=-\thecsvrow cm]
  \path [draw,Group/\group] (0,-0.45)
    rectangle node[font=\bfseries] {\amount} (\amount/1000,0.45);
  \node[left] at (0,0) {\land};
  \end{scope}  }
\end{tikzpicture}
```



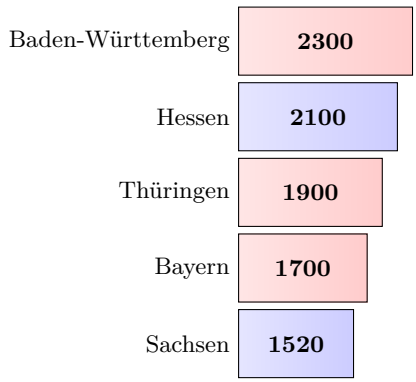
It would be nice to sort the bars by length, i. e. to sort the CSV file by the `amount` column. If the **CSV-Sorter** program is properly installed, see Subsection 3.9 on page 24, this can be done with the following configuration file for **CSV-Sorter**:

Configuration file «amountsort.xml»

```
<?xml version="1.0" encoding="UTF-8"?>
<csv>
  <bracket empty="true" />
  <sortlines>
    <column name="amount" order="descending" type="double"/>
    <column name="land" order="ascending" type="string"/>
  </sortlines>
</csv>
```

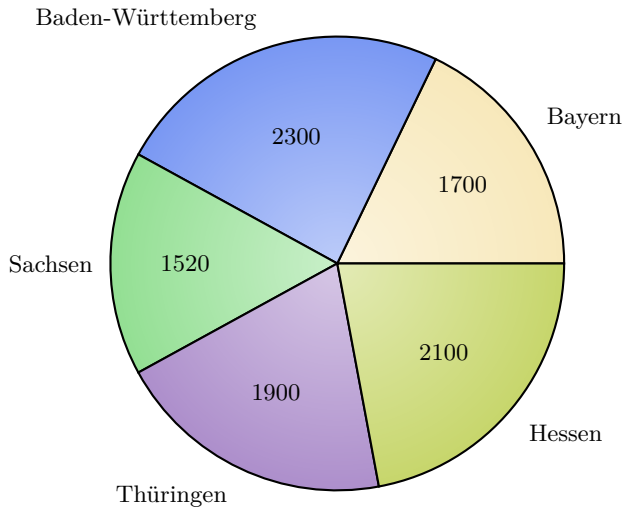
Now, we just have to add an option `sort by=amountsort.xml`:

```
% This example requires the package tikz
% Also, the CSV-Sorter tool has to be installed
\begin{tikzpicture}[Group/A/.style={left color=red!10,right color=red!20},
                    Group/B/.style={left color=blue!10,right color=blue!20}]
\csvreader[head to column names,sort by=amountsort.xml]{data.csv}{}{%
  \begin{scope}[yshift=-\thecsvrow cm]
  \path [draw,Group/\group] (0,-0.45)
    rectangle node[font=\bfseries] {\amount} (\amount/1000,0.45);
  \node[left] at (0,0) {\land};
  \end{scope}  }
\end{tikzpicture}
```



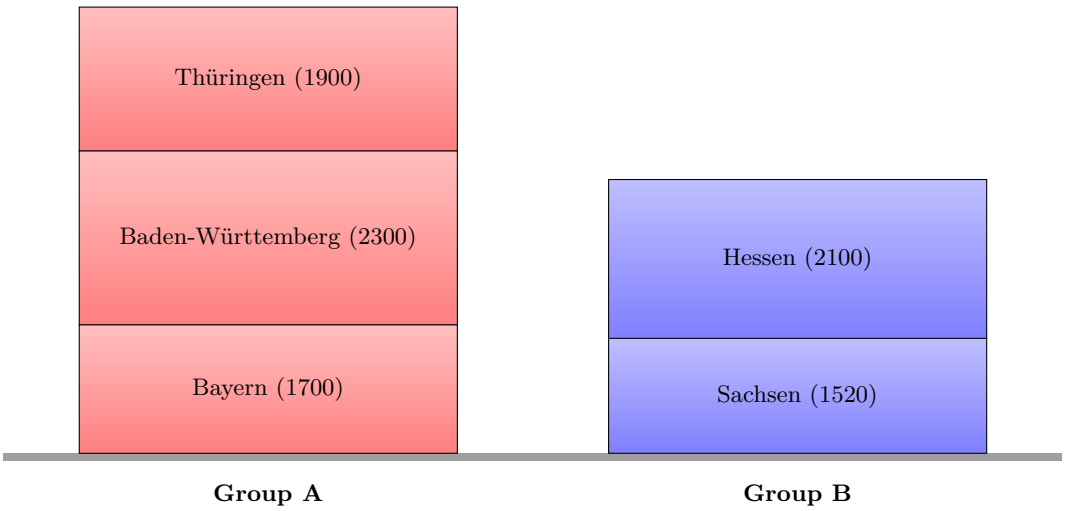
Next, we create a pie chart by calling `\csvreader` twice. In the first step, the total sum of amounts is computed, and in the second step the slices are drawn.

```
% Modified example from www.texample.net for pie charts
% This example needs the packages tikz, xcolor, calc
\definecolorseries{myseries}{rgb}{step}[rgb]{.95,.85,.55}{.17,.47,.37}
\resetcolorseries{myseries}%

% a pie slice
\newcommand{\slice}[4]{
  \pgfmathsetmacro{\midangle}{0.5*#1+0.5*#2}
  \begin{scope}
    \clip (0,0) -- (#1:1) arc (#1:#2:1) -- cycle;
    \colorlet{SliceColor}{myseries!!+}%
    \fill[inner color=SliceColor!30,outer color=SliceColor!60] (0,0) circle (1cm);
  \end{scope}
  \draw[thick] (0,0) -- (#1:1) arc (#1:#2:1) -- cycle;
  \node[label=\midangle:#4] at (\midangle:1) {};
  \pgfmathsetmacro{\temp}{min((#2-#1-10)/110*(-0.3),0)}
  \pgfmathsetmacro{\innerpos}{max(\temp,-0.5) + 0.8}
  \node at (\midangle:\innerpos) {#3};
}

% sum of amounts
\csvreader[before reading=\def\mysum{0}]{data.csv}{amount=\amount}{%
  \pgfmathsetmacro{\mysum}{\mysum+\amount}%
}

% drawing of the pie chart
\begin{tikzpicture}[scale=3]%
\def\mya{0}\def\myb{0}
\csvreader[head to column names]{data.csv}{}{%
  \let\mya\myb
  \pgfmathsetmacro{\myb}{\myb+\amount}
  \slice{\mya/\mysum*360}{\myb/\mysum*360}{\amount}{\land}
}
\end{tikzpicture}%
```



Finally, the filter option is demonstrated by separating the groups A and B. Every item is piled upon the appropriate stack.

```
\newcommand{\drawGroup}[2]{%
  \def\mya{0}\def\myb{0}
  \node[below=3mm] at (2.5,0) {\bfseries Group #1};
  \csvreader[head to column names,filter equal={\group}{#1}]{data.csv}{}{%
    \let\mya\myb
    \pgfmathsetmacro{\myb}{\myb+\amount}
    \path[draw,top color=#2!25,bottom color=#2!50]
      (0,\mya/1000) rectangle node{\land\ (\amount)} (5,\myb/1000);
}}

\begin{tikzpicture}
  \fill[gray!75] (-1,0) rectangle (13,-0.1);
  \drawGroup{A}{red}
  \begin{scope}[xshift=7cm]
  \drawGroup{B}{blue}
  \end{scope}
\end{tikzpicture}
```



5.3 Macro code inside the data

If needed, the data file may contain macro code. Note that the first character of a data line is not allowed to be the backslash '\'.

CSV file «macrodata.csv»

```
type,description,content
M,A nice \textbf{formula},          $\displaystyle \int\frac{1}{x} = \ln|x|+c$
G,A \textcolor{red}{colored} ball, {\tikz \shadedraw [shading=ball] (0,0) circle (.5cm);}
M,\textbf{Another} formula,         $\displaystyle \lim\limits_{n\to\infty} \frac{1}{n}=0$
```

Firstly, we survey the file content using `\csvautobooktabular`.

```
\csvautobooktabular{macrodata.csv}
```

type	description	content
M	A nice **formula**	$\displaystyle \int \frac{1}{x} = \ln \mathtt{x} + c$
G	A colored ball	
M	**Another** formula	$\displaystyle \lim_{n\to\infty} \frac{1}{n} = 0$

```
\csvstyle{my enumerate}{head to column names,
  before reading=\begin{enumerate},after reading=\end{enumerate}}

\csvreader[my enumerate]{macrodata.csv}{}{%
  \item \description:\par\content}

\bigskip
Now, formulas only:
\csvreader[my enumerate,filter equal={\type}{M}]{macrodata.csv}{}{%
  \item \description:\qquad\content}
```

1. A nice **formula**:
   $\displaystyle \int \frac{1}{x} = \ln \mathtt{x} + c$
2. A colored ball:
   
3. **Another** formula:
   $\displaystyle \lim_{n\to\infty} \frac{1}{n} = 0$

Now, formulas only:

1. A nice **formula**: $\displaystyle \int \frac{1}{x} = \ln \mathtt{x} + c$
2. **Another** formula: $\displaystyle \lim_{n\to\infty} \frac{1}{n} = 0$

5.4 Tables with Number Formatting

We consider a file with numerical data which should be pretty-printed.

CSV file «data_numbers.csv»

```
month,    dogs, cats
January,  12.50,12.3e5
February, 3.32, 8.7e3
March,    43,   3.1e6
April,    0.33, 21.2e4
May,      5.12, 3.45e6
June,     6.44, 6.66e6
July,     123.2,7.3e7
August,   12.3, 5.3e4
September,2.3,  4.4e4
October,  6.5,  6.5e6
November, 0.55, 5.5e5
December, 2.2,  3.3e3
```

The `siunitx` package provides a new column type `S` which can align material using a number of different strategies. The following example demonstrates the application with CSV reading. The package documentation of `siunitx` contains a huge amount of formatting options.

```
% \usepackage{siunitx,array,booktabs}
\csvloop{
  file=data_numbers.csv,
  head to column names,
  before reading=\centering\sisetup{table-number-alignment=center},
  tabular={lSS[table-format=2.2e1]@{}c},
  table head=\toprule\textbf{Month} & \textbf{Dogs} & \textbf{Cats} &\\\midrule,
  command=\month & \dogs & \cats &,
  table foot=\bottomrule}
```

Month	Dogs	Cats
January	12.50	12.3×10^{5}
February	3.32	8.7×10^{3}
March	43	3.1×10^{6}
April	0.33	21.2×10^{4}
May	5.12	3.45×10^{6}
June	6.44	6.66×10^{6}
July	123.2	7.3×10^{7}
August	12.3	5.3×10^{4}
September	2.3	4.4×10^{4}
October	6.5	6.5×10^{6}
November	0.55	5.5×10^{5}
December	2.2	3.3×10^{3}

Special care is needed, if the *first* or the *last* column is to be formatted with the column type `S`. The number detection of `siunitx` is disturbed by the line reading code of `csvsimple-legacy` which actually is present at the first and last column. To avoid this problem, the content of the first and last column could be formatted not by the table format definition, but by using a suitable `\tablenum` formatting directly, see `siunitx`.

Another and very nifty workaround suggested by Enrico Gregorio is to add an invisible dummy column with `c@{}` as first column and `@{}c` as last column:

```
% \usepackage{siunitx,array,booktabs}
\csvloop{
  file=data_numbers.csv,
  head to column names,
  before reading=\centering\sisetup{table-number-alignment=center},
  tabular={c@{}S[table-format=2.2e1]S@{}c},
  table head= & \textbf{Cats} & \textbf{Dogs} & \\\midrule,
  command= & \cats & \dogs &,
  table foot=\bottomrule}
```

Cats	Dogs
12.3×10^{5}	12.50
8.7×10^{3}	3.32
3.1×10^{6}	43
21.2×10^{4}	0.33
3.45×10^{6}	5.12
6.66×10^{6}	6.44
7.3×10^{7}	123.2
5.3×10^{4}	12.3
4.4×10^{4}	2.3
6.5×10^{6}	6.5
5.5×10^{5}	0.55
3.3×10^{3}	2.2

Now, the preceding table shall be sorted by the *cats* values. If the **CSV-Sorter** program is properly installed, see Subsection 3.9 on page 24, this can be done with the following configuration file for **CSV-Sorter**:

Configuration file «catsort.xml»

```
<?xml version="1.0" encoding="UTF-8"?>
<csv>
  <bracket empty="true" />
  <sortlines>
    <column name="cats" order="ascending" type="double"/>
  </sortlines>
</csv>
```

Now, we just have to add an option `sort by=catsort.xml`:

```
% \usepackage{siunitx,array,booktabs}
% Also, the CSV-Sorter tool has to be installed
\csvloop{
  file=data_numbers.csv,
  sort by=catsort.xml,
  head to column names,
  before reading=\centering\sisetup{table-number-alignment=center},
  tabular={lSS[table-format=2.2e1]@{}c},
  table head=\toprule\textbf{Month} & \textbf{Dogs} & \textbf{Cats} & \\\midrule,
  command=\month & \dogs & \cats &,
  table foot=\bottomrule}
```

Month	Dogs	Cats
December	2.2	3.3×10^{3}
February	3.32	8.7×10^{3}
September	2.3	4.4×10^{4}
August	12.3	5.3×10^{4}
April	0.33	21.2×10^{4}
November	0.55	5.5×10^{5}
January	12.50	12.3×10^{5}
March	43	3.1×10^{6}
May	5.12	3.45×10^{6}
October	6.5	6.5×10^{6}
June	6.44	6.66×10^{6}
July	123.2	7.3×10^{7}

5.5 CSV data without header line

CSV files with a header line are more semantic than files without header, but it's no problem to work with headless files.

For this example, we use again some artificial statistical data given by a CSV file but this time without header.

CSV file «data_headless.csv»

```
Bayern,A,1700
Baden-Württemberg,A,2300
Sachsen,B,1520
Thüringen,A,1900
Hessen,B,2100
```

Note that you cannot use the `/csv/no head` → P. 16 option for the auto tabular commands. If no options are given, the first line is interpreted as header line which gives an unpleasant result:

```
\csvautobooktabular{data_headless.csv}
```

Bayern	A	1700
Baden-Württemberg	A	2300
Sachsen	B	1520
Thüringen	A	1900
Hessen	B	2100

To get the expected result, one can redefine `/csv/table head` → P. 20 using `\csvlinetotablerow` → P. 13 which holds the first line data for the `\csvauto...` commands:

```
\csvautobooktabular[table head=\toprule\csvlinetotablerow\\]{data_headless.csv}
```

Bayern	A	1700
Baden-Württemberg	A	2300
Sachsen	B	1520
Thüringen	A	1900
Hessen	B	2100

This example can be extended to insert a table head for this headless data:

```
\csvautobooktabular[table head=\toprule\bfseries Land & \bfseries Group
  & \bfseries Amount\\\midrule\csvlinetotablerow\\]{data_headless.csv}
```

Land	**Group**	**Amount**
Bayern	A	1700
Baden-Württemberg	A	2300
Sachsen	B	1520
Thüringen	A	1900
Hessen	B	2100

For the normal \csvreader → P. 8 command, the /csv/no head → P. 16 option should be applied. Of course, we cannot use /csv/head to column names → P. 16 because there is no head, but the columns can be addressed by their numbers:

```
\csvreader[no head,
  tabular=lr,
  table head=\toprule\bfseries Land & \bfseries Amount\\\midrule,
  table foot=\bottomrule]
  {data_headless.csv}
  {1=\land,3=\amount}
  {\land & \amount}
```

Land	Amount
Bayern	1700
Baden-Württemberg	2300
Sachsen	1520
Thüringen	1900
Hessen	2100

5.6 Imported CSV data

If data is imported from other applications, there is not always a choice to format in comma separated values with curly brackets.

Consider the following example data file:

CSV file «imported.csv»

```
"name";"address";"email"
"Frank Smith";"Yellow Road 123, Brimblsby";"frank.smith@organization.org"
"Mary May";"Blue Alley 2a, London";"mmay@maybe.uk"
"Hans Meier";"Hauptstraße 32, Berlin";"hans.meier@corporation.de"
```

If the **CSV-Sorter** program is properly installed, see Subsection 3.9 on page 24, this can be transformed on-the-fly with the following configuration file for **CSV-Sorter**:

Configuration file «transform.xml»

```
<?xml version="1.0" encoding="UTF-8"?>
<csv>
  <bracket leftsymbol="doublequote" rightsymbol="doublequote" />
  <delimiter signsymbol="semicolon" />
  <outBracket leftsymbol="braceleft" rightsymbol="braceright" />
  <outDelimiter signsymbol="comma" />
</csv>
```

Now, we just have to add an option `sort by=transform.xml` to transform the input data. Here, we actually do not sort.

```
% \usepackage{booktabs,array}
% Also, the CSV-Sorter tool has to be installed
\newcommand{\Header}[1]{\normalfont\bfseries #1}

\csvreader[
  sort by=transform.xml,
  tabular=>{\itshape}ll>{\ttfamily}l,
  table head=\toprule\Header{Name} & \Header{Address} & \Header{email}\\\midrule,
  table foot=\bottomrule]
  {imported.csv}{}{\csvlinetotablerow}
```

Name	Address	email
Frank Smith	Yellow Road 123, Brimblsby	`frank.smith@organization.org`
Mary May	Blue Alley 2a, London	`mmay@maybe.uk`
Hans Meier	Hauptstraße 32, Berlin	`hans.meier@corporation.de`

The file which is generated on-the-fly and which is actually read by `csvsimple-legacy` is the following:

```
{name},{address},{email}
{Frank Smith},{Yellow Road 123, Brimblsby},{frank.smith@organization.org}
{Mary May},{Blue Alley 2a, London},{mmay@maybe.uk}
{Hans Meier},{Hauptstraße 32, Berlin},{hans.meier@corporation.de}
```

5.7 Encoding

If the CSV file has a different encoding than the LaTeX source file, then special care is needed.

- The most obvious treatment is to change the encoding of the CSV file or the LaTeX source file to match the other one (every good editor supports such a conversion). This is the easiest choice, if there a no good reasons against such a step. E.g., unfortunately, several tools under Windows need the CSV file to be `cp1252` encoded while the LaTeX source file may need to be `utf8` encoded.
- The `inputenc` package allows to switch the encoding inside the document, say from `utf8` to `cp1252`. Just be aware that you should only use pure ASCII for additional texts inside the switched region.

```
% !TeX encoding=UTF-8
% ....
\usepackage[utf8]{inputenc}
% ....
\begin{document}
% ....
\inputencoding{latin1}% only use ASCII from here, e.g. "Uberschrift
\csvreader[%...
  ]{data_cp1252.csv}{%...
  }{% ....
  }
\inputencoding{utf8}
% ....
\end{document}
```

- As a variant to the last method, the encoding switch can be done using options from `csvsimple-legacy`:

```
% !TeX encoding=UTF-8
% ....
\usepackage[utf8]{inputenc}
% ....
\begin{document}
% ....
% only use ASCII from here, e.g. "Uberschrift
\csvreader[%...
  before reading=\inputencoding{latin1},
  after reading=\inputencoding{utf8},
  ]{data_cp1252.csv}{%...
  }{% ....
  }
% ....
\end{document}
```

- If the **CSV-Sorter** program is properly installed, see Subsection 3.9 on page 24, the CSV file can be re-encoded on-the-fly with the following configuration file for **CSV-Sorter**:

Configuration file «encoding.xml»

```
<?xml version="1.0" encoding="UTF-8"?>
<csv>
  <noHeader/>
  <bracket empty="true"/>
  <charset in="windows-1252" out="UTF-8"/>
</csv>
```

```
% !TeX encoding=UTF-8
% ....
\usepackage[utf8]{inputenc}
% ....
\begin{document}
% ....
\csvreader[%...
  sort by=encoding.xml,
  ]{data_cp1252.csv}{%...
  }{% ....
  }
% ....
\end{document}
```

Index