The csvsimple package

Version 1.05 (2012/03/12)

Thomas F. Sturm[1]

Abstract

csvsimple provides a simple LaTeX interface for the processing of files with comma separated values (CSV). csvsimple relies heavily on the key value syntax from pgfkeys which results (hopefully) 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.

Contents

1 Introduction

The csvsimple package is applied to the processing of CSV[2] files. This processing is controlled by key value assignments according to the syntax of pgfkeys [3]. Sample applications of the package are tabular lists, serial letters, and charts.

An alternative to csvsimple is the datatool package [2] which provides considerably more functions like exchange of separator and delimiter symbols or sorting of data. csvsimple has a different approach for the user interface and is deliberately restricted to some basic functions with fast processing speed.

[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

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

1.1 Loading the Package

The package `csvsimple` loads the packages `pgfkeys` [3] and `ifthen` [1]. `csvsimple` itself is loaded in the usual manner in the preamble:

```
\usepackage{csvsimple}
```

1.2 First Steps

Every line of a processable CSV 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 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
Wei\ss{}b\"{a}ck,Werner,34567,m,5.0
```

The most simple way to display a CSV file in tabular form is the processing with the `\csvautotabular` → P. 7 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

Typically, one would use `\csvreader` → P. 6 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

`\csvreader` is controlled by a plenty of options. For example, for table applications line breaks are easily inserted by `/csv/late after line` → P. 10. 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

An even more comfortable 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

The next example shows such a style definition with the convenience macro `\csvstyle` → P. 7. Here, we see again the automated assignment of header entries to column names by `/csv/head to column names` → P. 11. 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.

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

\csvreader[myTableStyle]{grade.csv}{}%
  {\thecsvrow & \givenname~\name & \matriculation}%
```

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

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

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

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

	Person	Grade
1	Hans Maier	1.0
2	Anna Huber	2.3
3	Werner Weißbäck	5.0

Alternatively, column names can be set by \csvnames→ P. 7 and style definitions by \csvstyle→ P. 7. 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

	Person	Grade
1	Hans Maier	1.0
2	Anna Huber	2.3
3	Werner Weißbäck	5.0

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 equal={\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\ifthenelse{\equal{\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.

. . .

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 10.

```
\csvreader[tabular=|r|l|l|, table head=\hline, late after line=\\,
          late after last line=\\\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

Mainly, the `\csvreader` command consists of a `\csvloop` → P. 6 macro with following parameters:

`\csvloop{`⟨*options*⟩`, file=`⟨*file name*⟩`, column names=`⟨*assignments*⟩`,`
`command=`⟨*command list*⟩`}`

Therefore, the application of the keys `/csv/file` → P. 14 and `/csv/command` → P. 10 is useless for `\csvreader`.

`\csvloop{`⟨*options*⟩`}`

Usually, `\csvreader` → P. 6 may be preferred instead of `\csvloop`. `\csvreader` → P. 6 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}, column names={name=\name}, command=\name,
  before reading={List of students:\ },
  late after line={,}\ }, late after last line=.}
```

List of students: Maier, Huber, Weißbäck.

\csvautotabular{⟨*file name*⟩}

\csvautotabular is an abbreviation for the application of the option key /csv/autotabular →P. 13. 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

\csvautolongtable{⟨*file name*⟩}

csvautolongtable is an abbreviation for the application of the option key /csv/autolongtable →P. 13. 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.

\csvset{⟨*options*⟩}

Sets ⟨*options*⟩ for every following \csvreader →P. 6 and \csvloop →P. 6. 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 equal={\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);

\csvstyle{⟨*Stilname*⟩}{⟨*options*⟩}

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

\csvnames{⟨*Stilname*⟩}{⟨*Zuweisungsliste*⟩}

Abbreviation for \csvset{⟨*style name*⟩/.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 equal={\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);

\csvheadset{⟨*assignments*⟩}

For some special cases, this command can be used to change the ⟨*assignments*⟩ of macros to columns during execution of \csvreader →P. 6 and \csvloop →P. 6.

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

Maier … Hans Huber … Anna Weißbäck … Werner

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

Inside the command list of \csvreader → P. 6, 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)

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

Inside the command list of \csvreader → P. 6, 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,
  late after line=\\, late after last line=\\\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*

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

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,
  late after line=\\,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

\csvfilteraccept

All following consistent data lines will be accepted and processed. This command overwrites all previous filter settings and may be used inside /csv/before filter → P. 10 to implement an own filtering rule together with \csvfilterreject.

```
\csvreader[autotabular,
  before filter=\ifthenelse{\equal{\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

\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 equal={\thecsvinputline}{3}]%
  {grade.csv}{}%
  {The line with number \thecsvinputline\ contains: \csvline}%
```

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

\csvlinetotablerow

Typesets the current processed data line with & between the entries. Most users will never apply this command.

3 Option Keys

For the ⟨*options*⟩ in `\csvreader` →P. 6 respectively `\csvloop` →P. 6 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`=⟨*macros*⟩ (no default, initially empty)
Sets the ⟨*macros*⟩ to be executed before the CSV file is processed.

`/csv/after head`=⟨*macros*⟩ (no default, initially empty)
Sets the ⟨*macros*⟩ to be executed after the header line is read.

`/csv/before filter`=⟨*macros*⟩ (no default, initially empty)
Sets the ⟨*macros*⟩ 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. 12.

`/csv/late after head`=⟨*macros*⟩ (no default, initially empty)
Sets the ⟨*macros*⟩ 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`=⟨*macros*⟩ (no default, initially empty)
Sets the ⟨*macros*⟩ to be executed after reading and disassembling of the next accepted data line (after `/csv/before filter` →P. 10). They are executed before further processing of this next line. `late after line` overwrites `late after first line` and `late after last line`.

`/csv/late after first line`=⟨*macros*⟩ (no default, initially empty)
Sets the ⟨*macros*⟩ to be executed after reading and disassembling of the second accepted data line instead of `/csv/late after line` →P. 10. This key has to be set after `late after line`.

`/csv/late after last line`=⟨*macros*⟩ (no default, initially empty)
Sets the ⟨*macros*⟩ to be executed after processing of the last accepted data line instead of `/csv/late after line` →P. 10. This key has to be set after `late after line`.

`/csv/before line`=⟨*macros*⟩ (no default, initially empty)
Sets the ⟨*macros*⟩ to be executed after `/csv/late after line` →P. 10 and before `/csv/command` →P. 10. `before line` overwrites `before first line`.

`/csv/before first line`=⟨*macros*⟩ (no default, initially empty)
Sets the ⟨*macros*⟩ to be executed instead of `/csv/before line` →P. 10 for the first accepted data line. This key has to be set after `before line`.

`/csv/command`=⟨*macros*⟩ (no default, initially `\csvline`)
Sets the ⟨*macros*⟩ to be executed for every accepted data line. They are executed between `/csv/before line` →P. 10 and `/csv/after line` →P. 10.

`/csv/after line`=⟨*macros*⟩ (no default, initially empty)
Sets the ⟨*macros*⟩ to be executed for every accepted data line after `/csv/command` →P. 10. `after line` overwrites `after first line`.

`/csv/after first line`=⟨*macros*⟩ (no default, initially empty)
Sets the ⟨*macros*⟩ to be executed instead of `/csv/after line` →P. 10 for the first accepted data line. This key has to be set after `after line`.

`/csv/after reading`=⟨*macros*⟩ (no default, initially empty)
Sets the ⟨*macros*⟩ 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*⟩
⟨*late after last line*⟩
⟨*after reading*⟩

Additional command definition keys are provided for the supported tables, see section 3.4 from page 13.

3.2 Header Processing and Column Name Assignment

/csv/head=⟨*boolean value*⟩ (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.

/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=⟨*boolean value*⟩ (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.

3.3 Consistency Check and Filtering

/csv/check column count=⟨*boolean value*⟩ (default `true`, initially `true`)
This key defines, if the number of entries in a data line is checked against an expected value.
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. 11, since ⟨*number*⟩ would be replaced by the number of entries in the header line otherwise.

/csv/on column count error=⟨*macros*⟩ (no default, initially empty)
⟨*macros*⟩ to be executed for unfeasible data lines.

/csv/warn on column count error (style, no value)
Display of a warning for unfeasible data lines.

/csv/filter=⟨*condition*⟩ (no default)
Only data lines which fulfill a logical ⟨*condition*⟩ are accepted. For the ⟨*condition*⟩, every term from the `ifthen` package [1] is feasible. To preprocess the data line before testing the ⟨*condition*⟩, the option key `/csv/before filter` → P. 10 can be used.

/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.

/csv/filter equal=`{`⟨*string A*⟩`}{`⟨*string B*⟩`}` (style, no default)
Only lines where ⟨*string A*⟩ and ⟨*string B*⟩ are equal after expansion are accepted.

/csv/filter not equal=`{`⟨*string A*⟩`}{`⟨*string B*⟩`}` (style, no default)
Only lines where ⟨*string A*⟩ and ⟨*string B*⟩ are not equal after expansion are accepted.

3.4 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^{→ P. 13}, /csv/table head^{→ P. 13}, /csv/table foot^{→ P. 13}, and /csv/after table^{→ P. 13} are executed at the appropriate places.

/csv/centered tabular=⟨*table format*⟩ (style, no default)
Like /csv/tabular^{→ P. 13} but inside an additional `center` environment.

/csv/longtable=⟨*table format*⟩ (style, no default)
Like /csv/tabular^{→ P. 13} but for the `longtable` environment. This requires the package `longtable` (not loaded automatically).

/csv/tabbing (style, no value)
Like /csv/tabular^{→ P. 13} but for the `tabbing` environment.

/csv/centered tabbing (style, no value)
Like /csv/tabbing^{→ P. 13} but inside an additional `center` environment.

/csv/no table (style, no value)
Deactivates `tabular`, `longtable`, and `tabbing`.

/csv/before table=⟨*macros*⟩ (no default, initially empty)
Sets the ⟨*macros*⟩ to be executed before `\begin{tabular}` or before `\begin{longtable}` or before `\begin{tabbing}`, respectively.

/csv/table head=⟨*macros*⟩ (no default, initially empty)
Sets the ⟨*macros*⟩ to be executed after `\begin{tabular}` or after `\begin{longtable}` or after `\begin{tabbing}`, respectively.

/csv/table foot=⟨*macros*⟩ (no default, initially empty)
Sets the ⟨*macros*⟩ to be executed before `\end{tabular}` or before `\end{longtable}` or before `\end{tabbing}`, respectively.

/csv/after table=⟨*macros*⟩ (no default, initially empty)
Sets the ⟨*macros*⟩ to be executed after `\end{tabular}` or after `\end{longtable}` or after `\end{tabbing}`, respectively.

/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.

3.5 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[3]. 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 unused)

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 → P. 14. The second argument is the preprocessed CSV file which is set by /csv/preprocessed file → P. 14.

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. 6 or \csvloop → P. 6.

```
\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
}
```

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

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

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

4 Examples

4.1 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.,{Moosstra\ss{}e 32, Hinterschlag},10020,\"{O}rtingstetten,30
Wei\ss{}b\"{a}ck,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]{\ifthenelse{\equal{\gender}{m}}{#1}{#2}}

\csvreader[head to column names]{address.csv}{}{%
\begin{tcolorbox}[colframe=DarkGray,colback=White,arc=0mm,width=(\linewidth-2pt)/2,
                  before=,after=\hfill,title={Letter to \name},fonttitle=\bfseries]
  \ifthenelse{\equal{\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.
...

4.2 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-W\"{u}rttemberg,A,2300
Sachsen,B,1520
Th\"{u}ringen,A,1900
Hessen,B,2100
```

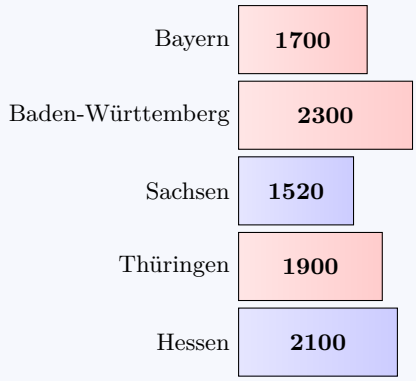
Firstly, we survey the file content using `\csvautotabular`.

```
\csvautotabular{data.csv}
```

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

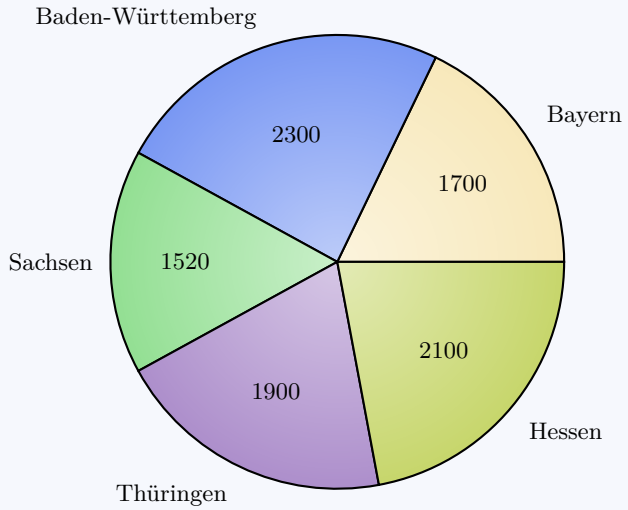
The amount values are presented in the following diagram by bars where the group classification is given using different colors.

```
% 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}
```



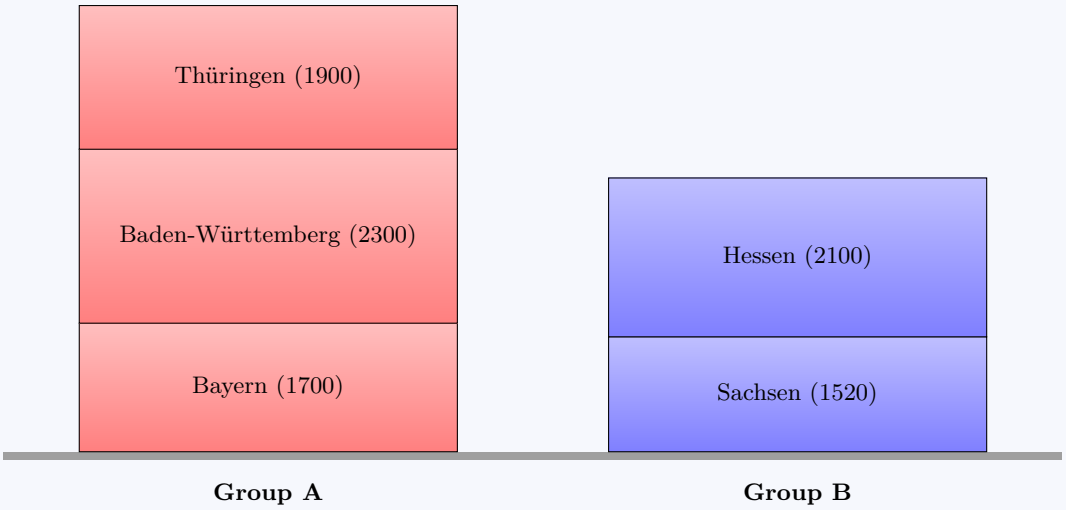
Next, we create a pie chart by calling `\csvreader` twice. In the first step, the total sum of amounts is computed, and in the second step the slices are drawn.

```
% 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}
```



References

[1] David Carlisle. *The ifthen package*. May 26, 2001. http://mirror.ctan.org/macros/latex/base/.

[2] Nicola L. C. Talbot. *datatool v 2.03: Databases and data manipulation*. Nov. 15, 2009. http://mirror.ctan.org/macros/latex/contrib/datatool/datatool.pdf.

[3] Till Tantau. *The TikZ and PGF Packages. Manual for version 2.10*. Oct. 25, 2010. http://mirror.ctan.org/graphics/pgf/base/doc/generic/pgf/pgfmanual.pdf.

Index