The `csvsimple` (`csvsimple-l3`) package

Manual for version 3.0.0 (2026/07/01)

Thomas F. Sturm[1]

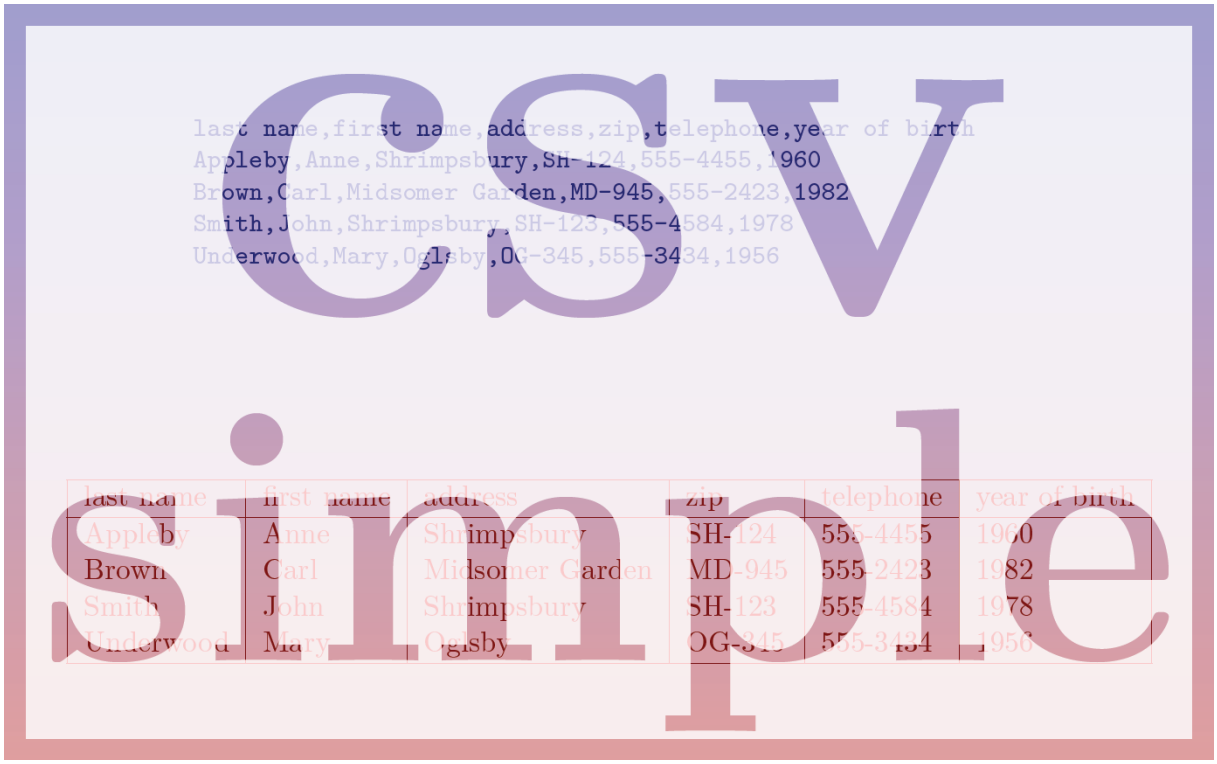
https://www.ctan.org/pkg/csvsimple

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

Abstract

`csvsimple(-l3)` provides a simple LaTeX interface for processing files with comma-separated values (CSV). `csvsimple(-l3)` makes extensive use of key–value syntax, resulting in a straightforward interface.

Filtering and table generation are particularly supported. As the package is intended to be lightweight, it does not provide support for data sorting or database storage.



[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` (`csvsimple-l3`) package is applied to the processing of CSV[2] files. This processing is controlled by key–value assignments according to the syntax of `l3keys`. Sample applications of the package are tabular lists, serial letters, and charts.

An alternative to `csvsimple` is the `datatool`→CTAN package which provides considerably more functions and allows sorting of data by LaTeX. `csvsimple` 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 4.10 on page 37.
- Values are expected to be comma-separated, but the package provides support for other separators, see Section 4.8 on page 34.
- Values are expected to be either not quoted or quoted with curly braces `{}` of TeX groups. Other quotes like double-quotes are not supported directly, but can be achieved with external tools, see Section 7.7 on page 62. For approximate patching see Section 6 on page 45.
- 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 4.3 on page 21.

1.1 Loading the Package

`csvsimple-l3` (or simply `csvsimple`) can be loaded in the preamble using *one* of the following alternatives:

```
\usepackage{csvsimple}
   % or alternatively (not simultaneously!)
\usepackage[l3]{csvsimple}
   % or alternatively (not simultaneously!)
\usepackage{csvsimple-l3}
```

The packages `longtable`→CTAN, `tabularray`→CTAN, `booktabs`→CTAN, `ifthen`→CTAN, and `etoolbox`→CTAN are used in many examples, but are not loaded automatically.

1.2 Alternative Option

As an alternative to `l3`, the package provides a `legacy` option that loads the LaTeX 2_ε version of `csvsimple`. This is the *superseded* version, identical to `csvsimple` version 1.22.

Documents based on that earlier version *do not need to be modified* and will remain compilable in the future if `csvsimple-legacy` is used.

`csvsimple-legacy` can be loaded in the preamble using *one* of the following alternatives:

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

The remainder of this documentation covers `csvsimple-l3` only. If you choose `csvsimple-legacy`, refer instead to "csvsimple-legacy.pdf".

For upgrading from `csvsimple-legacy` to `csvsimple-l3`, see Section 8 on page 67.

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

1.3 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. 14 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. 9 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 csvsim/separator → P. 34 for other separators than comma.

`\csvreader` is controlled by a plenty of options. For example, for table applications line breaks are easily inserted by `csvsim/late after line` → P. 18. 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 preferable way to create a table is setting appropriate option keys. Note, that this gives you the possibility to create a meta key (called style here) which contains the whole table creation using `\csvstyle` → P. 10 or `\keys_define:nn` from `l3keys`.

```
\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. 10. Here, we see again the automated assignment of header entries to column names by `csvsim/head to column names` → P. 20. For this, the header entries have to be without spaces and special characters. But you can always assign entries to canonical macro names manually like in the examples above. Here, we also add a `csvsim/head to column names prefix` → P. 20 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 `l3keys` syntax allows to create own meta options (styles) for a consistent and centralized design. The following example is easily modified to obtain more or less option settings.

```
\csvstyle{myStudentList}{%
    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. 10 and style definitions by \csvstyle → P. 10. 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\IfCsvsimStrEqualTF{\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 as comma-separated list of 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*⟩, from `\csvnames` → P. 10, from `csvsim/column names` → P. 20, or from `csvsim/head to column names` → P. 20 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 4 from page 18.

```
\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` → P. 10 macro with following parameters:

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

Therefore, the application of the keys `csvsim/file` → P. 36 and `csvsim/command` → P. 19 is useless for `\csvreader`.

`\csvloop{`⟨*options*⟩`}`

Usually, `\csvreader` → P. 9 may be preferred instead of `\csvloop`. `\csvreader` → P. 9 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.

U 2021-06-25 `\csvset{`⟨*options*⟩`}`

Sets ⟨*options*⟩ for every following `\csvreader` → P. 9 and `\csvloop`. Note that most options are set to default values at the begin of these commands and therefore cannot be defined reasonable by `\csvset`. But it may be used for options like `csvsim/csvsorter command` → P. 37 to give global settings. Also see `csvsim/every csv` → P. 36.

`\csvstyle{`⟨*key*⟩`}{`⟨*options*⟩`}`

Defines a new `l3keys` meta key to call other keys. It is used to make abbreviations for convenient key set applications. The new ⟨*key*⟩ can take one parameter. The name `\csvstyle` originates from an old version of `csvsimple` which used `pgfkeys` instead of `l3keys`.

```
\csvstyle{grade list}{
    column names = {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);

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

Abbreviation for `\csvstyle{`⟨*key*⟩`}{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);

U 2021-06-28 `\ifcsvoddrow{⟨then macros⟩}{⟨else macros⟩}`

Inside the command list of `\csvreader` → P. 9, the ⟨*then macros*⟩ are executed for odd-numbered data lines, and the ⟨*else macros*⟩ are executed for even-numbered lines. `\ifcsvoddrow` is expandable.

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

The deprecated, but still available alias for this command is `\csvifoddrow`.

U 2021-06-28 `\ifcsvfirstrow{⟨then macros⟩}{⟨else macros⟩}`

Inside the command list of `\csvreader` → P. 9, the ⟨*then macros*⟩ are executed for the first data line, and the ⟨*else macros*⟩ are executed for all following lines. `\ifcsvfirstrow` is expandable.

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

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

The deprecated, but still available alias for this command is `\csviffirstrow`.

`\csvfilteraccept`

All following consistent data lines will be accepted and processed. This command overwrites all previous filter settings and may be used inside `csvsim/full filter` → P. 26 to implement an own filtering rule together with `\csvfilterreject`.

```
\csvreader[
    autotabular,
    full filter = \IfCsvsimStrEqualTF{\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

U 2022-01-11 **\csvlinetotablerow**

Typesets the current processed data line with & between the entries. This macro is *expandable*.

U 2021-06-25 **\thecsvrow**
N 2021-06-25 **\g_csvsim_row_int**

Typesets the current data line number. This is the current number of accepted data lines without the header line. Despite of the name, there is no associated LaTeX counter `csvrow`, but \thecsvrow accesses the LaTeX3 integer \g_csvsim_row_int.

N 2021-06-25 **\thecsvcolumncount**
N 2021-06-25 **\g_csvsim_columncount_int**

Typesets the number of columns of the current CSV file. This number is either computed from the first valid line (header or data) or given by csvsim/column count → P. 21. Despite of the name, there is no associated LaTeX counter `csvcolumncount`, but \thecsvcolumncount accesses the LaTeX3 integer \g_csvsim_columncount_int.

```
\csvreader{grade.csv}{}{%
The last file consists of \thecsvcolumncount{} columns and
\thecsvrow{} accepted data lines. The total number of lines
is \thecsvinputline{}.
```

The last file consists of 5 columns and 4 accepted data lines. The total number of lines is 6.

U 2021-06-25 **\thecsvinputline**
N 2021-06-25 **\g_csvsim_inputline_int**

Typesets the current file line number. This is the current number of all data lines including the header line and all lines filtered out. Despite of the name, there is no associated LaTeX counter `csvinputline`, but \thecsvinputline accesses the LaTeX3 integer \g_csvsim_inputline_int.

```
\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

3 Macros for Automatic Survey Tables

The following `\csvauto...` commands are intended for quick data overview with *limited* formatting potential. The most customizable `\csvauto...` commands are `\csvautotabularray` → P. 16 and friends.

For full control see Subsection 4.6 on page 29 for the general table options in combination with `\csvreader` → P. 9 and `\csvloop` → P. 10.

`\csvautotabular[`⟨*options*⟩`]{`⟨*file name*⟩`}`
N 2021-06-25 `\csvautotabular*[`⟨*options*⟩`]{`⟨*file name*⟩`}`

`\csvautotabular` or `\csvautotabular*` is an abbreviation for the application of the option key `csvsim/autotabular` → P. 32 or `csvsim/autotabular*` → P. 32 together with other ⟨*options*⟩ to `\csvloop` → P. 10. This macro reads the whole CSV file denoted by ⟨*file name*⟩ with an automated formatting. The star variant treats the first line as data line and not as header line.

```
\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*⟩`}`
N 2021-06-25 `\csvautolongtable*[`⟨*options*⟩`]{`⟨*file name*⟩`}`

`\csvautolongtable` or `\csvautolongtable*` is an abbreviation for the application of the option key `csvsim/autolongtable` → P. 32 or `csvsim/autolongtable*` → P. 32 together with other ⟨*options*⟩ to `\csvloop` → P. 10. This macro reads the whole CSV file denoted by ⟨*file name*⟩ with an automated formatting. For application, the package `longtable` → CTAN is required which has to be loaded in the preamble. The star variant treats the first line as data line and not as header line.

```
\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*⟩}

N 2021-06-25 \csvautobooktabular*[⟨*options*⟩]{⟨*file name*⟩}

\csvautobooktabular or \csvautobooktabular* is an abbreviation for the application of the option key csvsim/autobooktabular → P. 32 or csvsim/autobooktabular* → P. 32 together with other ⟨*options*⟩ to \csvloop → P. 10. This macro reads the whole CSV file denoted by ⟨*file name*⟩ with an automated formatting. For application, the package booktabs → CTAN is required which has to be loaded in the preamble. The star variant treats the first line as data line and not as header line.

```
\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*⟩}

N 2021-06-25 \csvautobooklongtable*[⟨*options*⟩]{⟨*file name*⟩}

\csvautobooklongtable or \csvautobooklongtable* is an abbreviation for the application of the option key csvsim/autobooklongtable → P. 32 or csvsim/autobooklongtable* → P. 32 together with other ⟨*options*⟩ to \csvloop → P. 10. This macro reads the whole CSV file denoted by ⟨*file name*⟩ with an automated formatting. For application, the packages booktabs → CTAN and longtable → CTAN are required which have to be loaded in the preamble. The star variant treats the first line as data line and not as header line.

```
\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

N 2023-10-13
U 2023-10-17

\csvautotabularray[⟨*options*⟩]{⟨*file name*⟩}[⟨*taboptions 1*⟩][⟨*taboptions 2*⟩]
\csvautotabularray*[⟨*options*⟩]{⟨*file name*⟩}[⟨*taboptions 1*⟩][⟨*taboptions 2*⟩]
\csvautolongtabularray[⟨*options*⟩]{⟨*file name*⟩}[⟨*taboptions 1*⟩][⟨*taboptions 2*⟩]
\csvautolongtabularray*[⟨*options*⟩]{⟨*file name*⟩}[⟨*taboptions 1*⟩][⟨*taboptions 2*⟩]

These macros are abbreviations for the application of the option keys csvsim/autotabularray → P. 32, csvsim/autotabularray* → P. 32, csvsim/autolongtabularray → P. 32, or csvsim/autolongtabularray* → P. 32 together with other ⟨*options*⟩ to \csvloop → P. 10. These macros read the whole CSV file denoted by ⟨*file name*⟩ with an automated formatting. For application, the package tabularray → CTAN is required which has to be loaded in the preamble. \csvautotabularray uses the tblr environment and \csvautolongtabularray uses the longtblr environment. The star variants treat the first line as data line and not as header line.
Options to the table environments from tabularray → CTAN may be given by either setting csvsim/generic table options → P. 31 or using ⟨*taboptions 1*⟩ and ⟨*taboptions 2*⟩.
The default setting is

```
generic table options =
  { {
    row{1}     = {font=\bfseries,preto=\MakeUppercase},
    hline{1,Z} = {0.08em},
    hline{2}   = {0.05em},
  } }
```

For the star variants, the default setting is

```
generic table options =
  { {
    hline{1,Z} = {0.08em},
  } }
```

Examples:

```
\csvautotabularray{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

```
\csvautotabularray[table centered,
    generic table options =
    {{
      row{odd}   = {red!85!gray!7},
      row{1}     = {bg=red!85!gray, fg=white,
                    font=\bfseries, preto=\MakeUppercase},
    }}
  ] {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

Alternatively to csvsim/generic table options → P. 31 (and overruling this option), one may give options to **tblr** or **longtblr** using ⟨*taboptions 1*⟩ and ⟨*taboptions 2*⟩. If ⟨*taboptions 2*⟩ is *not present*, then ⟨*taboptions 1*⟩ is used as mandatory argument (tabularray → CTAN inner specification). Otherwise, ⟨*taboptions 1*⟩ is used as optional argument (tabularray → CTAN outer specification) and ⟨*taboptions 2*⟩ as mandatory argument (tabularray → CTAN inner specification).

```
\csvautotabularray[table centered]
  {grade.csv}
  [
    row{odd}    = {red!85!gray!7},
    row{1}      = {bg=red!85!gray, fg=white,
                   font=\bfseries, preto=\MakeUppercase},
  ]
```

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

```
\csvautotabularray[table centered]
  {grade.csv}
  [
    tall,
    caption      = {My table},
    remark{Note} = {My remark},
  ]
  [
    row{odd}    = {red!85!gray!7},
    row{1}      = {bg=red!85!gray, fg=white,
                   font=\bfseries, preto=\MakeUppercase},
  ]
```

Table 3: My table

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

Note: My remark

4 Option Keys

For the ⟨*options*⟩ in `\csvreader` → P. 9 respectively `\csvloop` → P. 10 the following `l3keys` keys can be applied. The ⟨*module*⟩ name `csvsim` is not to be used inside these macros.

4.1 Command Definition

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

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

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

Sets the ⟨*code*⟩ to be executed after the header line is read. `\thecsvcolumncount` → P. 13 and header entries are available.

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

Sets the ⟨*code*⟩ to be executed after reading and consistency checking of a data line. It is executed before any filter condition is checked, see e.g. `csvsim/filter ifthen` → P. 26 and also see `csvsim/full filter` → P. 26. No additions to the input stream should be given here. All line entries are available.

N 2021-07-06 `csvsim/after filter`=⟨*code*⟩ (no default, initially empty)

Sets the ⟨*code*⟩ to be executed for an accepted line after `csvsim/late after line` and before `csvsim/before line`. All line entries are available. No additions to the input stream should be given here. ⟨*code*⟩ may contain processing of data content to generate new values.

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

Sets the ⟨*code*⟩ to be executed after reading and disassembling of the first accepted data line. These operations are executed before further processing of this line. ⟨*code*⟩ should not refer to any data content, but may be something like \\.

`csvsim/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 `csvsim/before filter`). These operations are executed before further processing of this line. ⟨*code*⟩ should not refer to any data content, but may be something like \\. `csvsim/late after line` overwrites `csvsim/late after first line` and `csvsim/late after last line`. Note that table options like `csvsim/tabular` → P. 29 set this key to \\ automatically.

`csvsim/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 `csvsim/late after line`. ⟨*code*⟩ should not refer to any data content. This key has to be set after `csvsim/late after line`.

`csvsim/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 `csvsim/late after line`. ⟨*code*⟩ should not refer to any data content. This key has to be set after `csvsim/late after line`.

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

Sets the ⟨*code*⟩ to be executed after `csvsim/after filter` and before `csvsim/command` → P. 19. All line entries are available. `csvsim/before line` overwrites `csvsim/before first line`.

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

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

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

Sets the ⟨*code*⟩ to be executed for every accepted data line. It is executed between `csvsim/before line` → P. 18 and `csvsim/after line`. `csvsim/command` describes the main processing of the line entries. `\csvreader` → P. 9 sets `csvsim/command` as mandatory parameter.

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

Sets the ⟨*code*⟩ to be executed for every accepted data line after `csvsim/command`. All line entries are still available. `csvsim/after line` overwrites `csvsim/after first line`.

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

Sets the ⟨*code*⟩ to be executed instead of `csvsim/after line` for the first accepted data line. All line entries are still available. This key has to be set after `csvsim/after line`.

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

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

The following example illustrates the sequence of command execution. Note that `csvsim/command` is set by the mandatory last parameter of `\csvreader` → P. 9.

```
\csvreader[
  before reading        = \meta{before reading}\\,
  after head            = \meta{after head},
  before filter         = \\\meta{before filter},
  after filter          = \meta{after 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*⟩⟨*after filter*⟩⟨*before first line*⟩**Maier**⟨*after first line*⟩
⟨*before filter*⟩⟨*late after first line*⟩⟨*after filter*⟩⟨*before line*⟩**Huber**⟨*after line*⟩
⟨*before filter*⟩⟨*late after line*⟩⟨*after filter*⟩⟨*before line*⟩**Weißbäck**⟨*after line*⟩
⟨*before filter*⟩⟨*late after line*⟩⟨*after filter*⟩⟨*before line*⟩**Bauer**⟨*after line*⟩
⟨*late after last line*⟩
⟨*after reading*⟩

Additional command definition keys are provided for the supported tables, see Section 4.6 from page 29.

4.2 Header Processing and Column Name Assignment

`csvsim/doc updated/head=true|false` (default `true`, initially `true`)

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

`csvsim/no head` (no value)

Abbreviation for `head=false`, i. e. the first non-empty line of the CSV file is treated as data line. Note that this option cannot be used in combination with the `\csvauto...` commands like `\csvautotabular` → P. 14, etc. Instead, there are *star* variants like `\csvautotabular*` → P. 14 to process files without header line. See Section 7.5 on page 57 for examples.

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

Adds some new ⟨*assignments*⟩ of macros to columns in key–value syntax. Existing assignments are kept.
The ⟨*assignments*⟩ are given as comma-separated list of 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.

```
column names = {name=\surname, givenname=\firstname, grade=\grade}
```

`csvsim/column names reset` (no value)

Clears all assignments of macros to columns.

`csvsim/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 `csvsim/head to column names prefix` may help to avoid unwanted overrides.

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

The given ⟨*text*⟩ is prefixed to the name of all macros generated by `csvsim/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.

N 2022-02-01 `csvsim/column names detection=true|false` (default `true`, initially `true`)

If this key is set, the header line is detected for names which can be used for `csvsim/column names` and `csvsim/head to column names`. Otherwise, these options are not functional.
This key can and should be set to `false`, if the header line contains macros or characters not allowed inside LATEX control sequences, because otherwise compilation error are to be expected.

4.3 Consistency Check

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

This key defines, whether 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.

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

Abbreviation for `check column count=false`.

U 2021-06-24 `csvsim/column count=`⟨*number*⟩ (no default, initially `0`)

Sets the ⟨*number*⟩ of feasible entries per data line. If `csvsim/column count` is set to `0`, the number of entries of the first non-empty line determines the column count (automatic detection).
This setting is only useful in connection with `csvsim/no head` → P. 20, since ⟨*number*⟩ would be replaced by the number of entries in the header line otherwise.

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

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

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

Display of a warning for unfeasible data lines.

4.4 Filtering

Applying a *filter* means that data lines are only processed / displayed, if they fulfill a given *condition*.

The following string compare filters `csvsim/filter strcmp` and `csvsim/filter equal` are identical by logic, but differ in implementation.

U 2022-10-21 **csvsim/filter strcmp**={⟨*stringA*⟩}{⟨*stringB*⟩} (no default)

Only lines where ⟨*stringA*⟩ and ⟨*stringB*⟩ are equal after expansion are accepted. The implementation is done with `\str_if_eq_p:ee`.

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

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

U 2022-10-21 **csvsim/filter not strcmp**={⟨*stringA*⟩}{⟨*stringB*⟩} (no default)

Only lines where ⟨*stringA*⟩ and ⟨*stringB*⟩ are not equal after expansion are accepted. The implementation is done with `\str_if_eq_p:ee`.

csvsim/filter equal={⟨*stringA*⟩}{⟨*stringB*⟩} (no default)

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

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

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

N 2021-06-25 U 2022-10-21 **csvsim/filter fp**={⟨*floating point comparison*⟩} (no default)

Only data lines which fulfill a LaTeX3 ⟨*floating point comparison*⟩ are accepted. The evaluation is done using `\fp_compare_p:n`.

```
% \usepackage{booktabs}
\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 fp  = {  \matriculation > 20000  &&  \grade < 4.0  },
  ]{grade.csv}{}{%
    \thecsvrow & \slshape\name, \givenname & \matriculation & \grade
  }
```

	Name	Matr	Grade
1	*Huber, Anna*	23456	2.3

N 2021-06-25
U 2022-10-21

`csvsim/filter bool={`⟨*boolean expression*⟩`}` (no default)

Only data lines which fulfill a LaTeX3 ⟨*boolean expression*⟩ are accepted. Note that such an ⟨*boolean expression*⟩ needs expl3 code. To preprocess the data line before testing the ⟨*boolean expression*⟩, the option key `csvsim/before filter` → P. 18 can be used.

```
% For convenience, we save the filter
\ExplSyntaxOn
%>> list only matriculation numbers greater than 20000, list only men <<
\csvstyle{myfilter}
  {
    filter~bool =
      {
        \int_compare_p:n { \matriculation > 20000 } &&
        \str_if_eq_p:ee  { \gender }{ m }
      }
  }
\ExplSyntaxOff

\csvreader[
    head to column names,
    tabular    = llll,
    table head = \toprule & \bfseries Name & \bfseries Matr & \bfseries Grade\\\midrule,
    table foot = \bottomrule,
    myfilter
  ]{grade.csv}{}{%
    \thecsvrow & \slshape\name, \givenname & \matriculation & \grade
  }
```

	Name	Matr	Grade
1	*Weißbäck, Werner*	34567	5.0

N 2021-06-25

`\csvfilterbool{`⟨*key*⟩`}{`⟨*boolean expression*⟩`}`

Defines a new `l3keys` meta key which applies `csvsim/filter bool` with the given ⟨*boolean expression*⟩.

```
% For convenience, we save the filter
\ExplSyntaxOn
%>> list only matriculation numbers greater than 20000, list only men <<
\csvfilterbool{myfilter}
  {
    \int_compare_p:n { \matriculation > 20000 } &&
    \str_if_eq_p:ee  { \gender }{ m }
  }
\ExplSyntaxOff

\csvreader[
    head to column names,
    tabular    = llll,
    table head = \toprule & \bfseries Name & \bfseries Matr & \bfseries Grade\\\midrule,
    table foot = \bottomrule,
    myfilter
  ]{grade.csv}{}{%
    \thecsvrow & \slshape\name, \givenname & \matriculation & \grade
  }
```

	Name	Matr	Grade
1	*Weißbäck, Werner*	34567	5.0

The following filter options are *appendable* to the expl3 based filter options:

- `csvsim/filter strcmp` → P. 22
- `csvsim/filter not strcmp` → P. 22
- `csvsim/filter fp` → P. 22
- `csvsim/filter bool` → P. 23

N 2022-10-21 **csvsim/and filter strcmp**={⟨*stringA*⟩}{⟨*stringB*⟩} (no default)
N 2022-10-21 **csvsim/or filter strcmp**={⟨*stringA*⟩}{⟨*stringB*⟩} (no default)

Like `csvsim/filter strcmp` → P. 22, but appended to a required existing expl3 based filter with *and* (`&&`) resp. *or* (`||`).

```
\csvreader[
    head to column names,
    tabular    = llll,
    table head = \toprule & \bfseries Name & \bfseries Matr & \bfseries Grade\\\midrule,
    table foot = \bottomrule,
    filter fp  = {\matriculation>20000},
    and filter strcmp = {\gender}{m},
  ]{grade.csv}{}{%
    \thecsvrow & \slshape\name, \givenname & \matriculation & \grade
  }
```

	Name	Matr	Grade
1	*Weißbäck, Werner*	34567	5.0

N 2022-10-21 **csvsim/and filter not strcmp**={⟨*stringA*⟩}{⟨*stringB*⟩} (no default)
N 2022-10-21 **csvsim/or filter not strcmp**={⟨*stringA*⟩}{⟨*stringB*⟩} (no default)

Like `csvsim/filter not strcmp` → P. 22, but appended to a required existing expl3 based filter with *and* (`&&`) resp. *or* (`||`).

N 2022-10-21 **csvsim/and filter fp**={⟨*floating point comparison*⟩} (style, no default)
N 2022-10-21 **csvsim/or filter fp**={⟨*floating point comparison*⟩} (style, no default)

Like `csvsim/filter fp` → P. 22, but appended to a required existing expl3 based filter with *and* (`&&`) resp. *or* (`||`).

```
% \usepackage{booktabs}
\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 fp      = { \matriculation > 20000 },
    and filter fp  = { \grade < 4.0  },
  ]{grade.csv}{}{%
    \thecsvrow & \slshape\name, \givenname & \matriculation & \grade
  }
```

	Name	Matr	Grade
1	*Huber, Anna*	23456	2.3

N 2022-10-21 **csvsim/and filter bool**={⟨*boolean expression*⟩} (style, no default)
N 2022-10-21 **csvsim/or filter bool**={⟨*boolean expression*⟩} (style, no default)

Like `csvsim/filter bool` → P. 23, but appended to a required existing expl3 based filter with *and* (`&&`) resp. *or* (`||`).

N 2016-07-01 **csvsim/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 → CTAN package like `\ifnumcomp`, `\ifdimgreater`, etc. and from Section 5 on page 44 can be used. Also, arbitrary own macros fulfilling this signature can be applied.

```
% \usepackage{etoolbox,booktabs}
\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

N 2016-07-01 **csvsim/filter expr=**⟨*boolean expression*⟩ (no default)

Only data lines which fulfill a ⟨*boolean expression*⟩ are accepted. Every ⟨*boolean expression*⟩ from the etoolbox → CTAN package is feasible (package loading required!). To preprocess the data line before testing the ⟨*boolean expression*⟩, the option key csvsim/before filter → P. 18 can be used.

```
% \usepackage{etoolbox,booktabs}
\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 **csvsim/filter ifthen=**⟨*boolean expression*⟩ (no default)

Only data lines which fulfill a ⟨*boolean expression*⟩ are accepted. For the ⟨*boolean expression*⟩, every term from the ifthen[→ CTAN] package is feasible (package loading required!). To preprocess the data line before testing the ⟨*boolean expression*⟩, the option key csvsim/before filter[→ P. 18] can be used.

```
% \usepackage{ifthen,booktabs}
\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

csvsim/no filter (no value, initially set)

Clears a set filter.

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

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

csvsim/filter reject all (no value)

All data line are ignored.

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

Technically, this key is an alias for csvsim/before filter[→ P. 18]. Philosophically, csvsim/before filter[→ P. 18] computes something before a filter condition is set, but csvsim/full filter should implement the full filtering. Especially, \csvfilteraccept[→ P. 12] or \csvfilterreject[→ P. 12] *should* be set inside the ⟨*code*⟩.

```
% \usepackage{etoolbox,booktabs}
\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

4.5 Line Range

Applying a *line range* means to select certain line numbers to be displayed. These line numbers are not necessarily line numbers of the input file, see `\thecsvinputline` → P. 13, but line numbers of type `\thecsvrow` → P. 13.

For example, if a *filter* was applied, see Section 4.4 on page 22, and 42 lines are accepted, a *range* could select the first 20 of them or line 10 to 30 of the accepted lines.

N 2021-06-29
U 2022-09-21

`csvsim/range`={⟨*range1*⟩,⟨*range2*⟩,⟨*range3*⟩,...} (no default, initially empty)

Defines a comma-separated list of line ranges. If a line number `\thecsvrow` → P. 13 satisfies one or more of the given ⟨*range1*⟩, ⟨*range2*⟩, ..., the corresponding line is processed and displayed. If `csvsim/range` is set to empty, all lines are accepted.
Every ⟨*range*⟩ can corresponds to one of the following variants:

⟨*a*⟩-⟨*b*⟩	meaning line numbers ⟨*a*⟩ to ⟨*b*⟩.
⟨*a*⟩-	meaning line numbers ⟨*a*⟩ to `\c_max_int`=2 147 483 647.
-⟨*b*⟩	meaning line numbers 1 to ⟨*b*⟩.
-	meaning line numbers 1 to 2 147 483 647 (inefficient; don't use).
⟨*a*⟩	meaning line numbers ⟨*a*⟩ to ⟨*a*⟩ (i.e. only ⟨*a*⟩).
⟨*a*⟩+⟨*d*⟩	meaning line numbers ⟨*a*⟩ to ⟨*a*⟩+⟨*d*⟩−1.
⟨*a*⟩+	meaning line numbers ⟨*a*⟩ to ⟨*a*⟩ (i.e. only ⟨*a*⟩).
+⟨*d*⟩	meaning line numbers 1 to ⟨*d*⟩.
+	meaning line numbers 1 to 1 (i.e. only 1; weird).

```
% \usepackage{booktabs}
\csvreader[
    head to column names,
    range      = 2-3,
    tabular    = llll,
    table head = \toprule & \bfseries Name & \bfseries Matr & \bfseries Grade\\\midrule,
    table foot = \bottomrule,
  ]{grade.csv}{}{%
    \thecsvrow & \slshape\name, \givenname & \matriculation & \grade
  }
```

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

```
% \usepackage{booktabs}
\csvreader[
    head to column names,
    range      = 3-,
    tabular    = llll,
    table head = \toprule & \bfseries Name & \bfseries Matr & \bfseries Grade\\\midrule,
    table foot = \bottomrule,
  ]{grade.csv}{}{%
    \thecsvrow & \slshape\name, \givenname & \matriculation & \grade
  }
```

	Name	Matr	Grade
3	*Weißbäck, Werner*	34567	5.0
4	*Bauer, Maria*	19202	3.3

```
% \usepackage{booktabs}
\csvreader[
    head to column names,
    range      = 2+2,
    tabular    = llll,
    table head = \toprule & \bfseries Name & \bfseries Matr & \bfseries Grade\\\midrule,
    table foot = \bottomrule,
  ]{grade.csv}{}{%
    \thecsvrow & \slshape\name, \givenname & \matriculation & \grade
  }
```

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

```
% \usepackage{booktabs}
\csvreader[
    head to column names,
    range      = {2,4},
    tabular    = llll,
    table head = \toprule & \bfseries Name & \bfseries Matr & \bfseries Grade\\\midrule,
    table foot = \bottomrule,
  ]{grade.csv}{}{%
    \thecsvrow & \slshape\name, \givenname & \matriculation & \grade
  }
```

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

To select the last n lines, you have to know or count the line numbers first. The following example displays the last three line numbers:

```
% \usepackage{booktabs}
\csvreader{grade.csv}{}{}%   count line numbers
\csvreader[
    head to column names,
    range      = {\thecsvrow-2}-,
    tabular    = llll,
    table head = \toprule & \bfseries Name & \bfseries Matr & \bfseries Grade\\\midrule,
    table foot = \bottomrule,
  ]{grade.csv}{}{%
    \thecsvrow & \slshape\name, \givenname & \matriculation & \grade
  }
```

	Name	Matr	Grade
2	*Huber, Anna*	23456	2.3
3	*Weißbäck, Werner*	34567	5.0
4	*Bauer, Maria*	19202	3.3

4.6 Table Support

4.6.1 Predefined Tables

`csvsim/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 `csvsim/before table` →P. 30, `csvsim/table head` →P. 30, `csvsim/table foot` →P. 30, and `csvsim/after table` →P. 30 are executed at the appropriate places. `csvsim/late after line` →P. 18 is set to `\\`.

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

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

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

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

`csvsim/tabbing` (style, no value)

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

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

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

N 2021-07-06 `csvsim/tabularray`=⟨*table format*⟩ (style, no default)

Like `csvsim/tabular` but for the `tblr` environment. This requires the package `tabularray` →CTAN (not loaded automatically). This also sets `csvsim/collect data` →P. 42 since this kind of table needs collected content, see Section 4.11 on page 42. Note that `csvsim/after reading` →P. 19 is set to use the collected data immediately. See Section 7.6 on page 59 for examples.

N 2021-07-23 `csvsim/long tabularray`=⟨*table format*⟩ (style, no default)

Like `csvsim/tabularray` but using the `longtblr` environment from the package `tabularray` →CTAN (not loaded automatically).

N 2021-07-06 `csvsim/centered tabularray`=⟨*table format*⟩ (style, no default)

Like `csvsim/tabularray` but inside an additional `center` environment.

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

Deactivates `tabular`-like environments activated by `csvsim/tabular`, `csvsim/longtable`, etc. Note that not all settings of `csvsim/tabularray` are reverted.

4.6.2 Additional Options for Tables

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

Sets the ⟨*code*⟩ to be executed before the begin of `tabular`-like environments, i.e. immediately before `\begin{tabular}`, etc.

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

Sets the ⟨*code*⟩ to be executed after the begin of `tabular`-like environments, i.e. immediately after `\begin{tabular}`, etc.

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

Sets the ⟨*code*⟩ to be executed before the end of `tabular`-like environments, i.e. immediately before `\end{tabular}`, etc.

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

Sets the ⟨*code*⟩ to be executed after the end of `tabular`-like environments, i.e. immediately after `\end{tabular}`, etc.

N 2021-09-09 `csvsim/table centered=true|false` (default `true`, initially `false`)

If `true`, the table is put inside an additional `center` environment. This environment begins before `csvsim/before table` and ends after `csvsim/after table`. The predefined `tabular`-like environments from Section Section 4.6.1 on the preceding page use this option internally, i.e. `centered tabular=ccc` is identical to `tabular=ccc, table centered`.

4.6.3 Generic Tables

In Section Section 4.6.1 on page 29, several `tabular`-like environments are described with predefined keys. The following keys allow to use further `tabular`-like environments with configurable names and options.

N 2021-09-09 **`csvsim/generic table`**=⟨*name*⟩ (no default, initially empty)

Surrounds the CSV processing with `\begin{`⟨*name*⟩`}` at begin and with `\end{`⟨*name*⟩`}` at end. Additionally, the commands defined by the key–values of `csvsim/before table` →P. 30, `csvsim/table head` →P. 30, `csvsim/table foot` →P. 30, and `csvsim/after table` →P. 30 are executed at the appropriate places. `csvsim/late after line` →P. 18 is set to `\\`.
If the environment ⟨*name*⟩ takes options, these have to be set using `csvsim/generic table options`.

```
% The `tabular` environment would be used like the following example
...
generic table         = tabular,
generic table options = {{ccllrr}},
...
```

N 2021-09-09
U 2023-12-18 **`csvsim/generic collected table`**=⟨*name*⟩ (no default, initially empty)

Like `csvsim/generic table` but for environments which need collected content, e.g. `tblr` from package `tabularray` →CTAN, see Section 4.11 on page 42. Note that `csvsim/consume collected data` →P. 43 is set to `true` to use the collected data immediately.

```
% The `tblr` environment from package `tabularray` would be used
% like the following example
...
generic collected table = tblr,
generic table options   = {{rowsep=1mm, colsep=5mm}},
...
```

N 2021-09-09 **`csvsim/generic table options`**={⟨*code*⟩} (no default, initially empty)

Places ⟨*code*⟩ immediately after `\begin{`⟨*name*⟩`}` set up with `csvsim/generic table` or `csvsim/generic collected table`. ⟨*code*⟩ may contain any parameters the environment ⟨*name*⟩ needs to have. **You are strongly advised to use an extra pair of curly brackets {⟨*code*⟩} around ⟨*code*⟩**, because the outer pair of braces is removed during option processing, see examples above.

```
% Environment without parameters:
generic table options =,
% Environment with a mandatory parameter:
generic table options = {{parameter}},
% Environment with an optional and a mandatory parameter:
generic table options = {[optional]{mandatory}},
% Environment with two mandatory parameters:
generic table options = {{mandatory 1}{mandatory 2}},
```

4.6.4 General Survey Tables

The following `auto` options are the counterparts for the respective quick overview commands like `\csvautotabular` → P. 14, see Section 3. They are listed for completeness, but are unlikely to be used directly.

U 2022-02-01 **csvsim/autotabular**=⟨*file name*⟩ (no default)
csvsim/autotabular*=⟨*file name*⟩ (no default)

Reads the whole CSV file denoted ⟨*file name*⟩ with an automated formatting. The star variant treats the first line as data line and not as header line.

U 2022-02-01 **csvsim/autolongtable**=⟨*file name*⟩ (no default)
csvsim/autolongtable*=⟨*file name*⟩ (no default)

Reads the whole CSV file denoted ⟨*file name*⟩ with an automated formatting using the required `longtable` package. The star variant treats the first line as data line and not as header line.

U 2022-02-01 **csvsim/autobooktabular**=⟨*file name*⟩ (no default)
csvsim/autobooktabular*=⟨*file name*⟩ (no default)

Reads the whole CSV file denoted ⟨*file name*⟩ with an automated formatting using the required `booktabs` package. The star variant treats the first line as data line and not as header line.

U 2022-02-01 **csvsim/autobooklongtable**=⟨*file name*⟩ (no default)
csvsim/autobooklongtable*=⟨*file name*⟩ (no default)

Reads the whole CSV file denoted ⟨*file name*⟩ with an automated formatting using the required `booktabs` and `longtable` packages. The star variant treats the first line as data line and not as header line.

N 2023-10-13 **csvsim/autotabularray**=⟨*file name*⟩ (no default)
N 2023-10-13 **csvsim/autotabularray***=⟨*file name*⟩ (no default)
N 2023-10-13 **csvsim/autolongtabularray**=⟨*file name*⟩ (no default)
N 2023-10-13 **csvsim/autolongtabularray***=⟨*file name*⟩ (no default)

Reads the whole CSV file denoted ⟨*file name*⟩ with an automated formatting using the required `tabularray` package. `csvsim/autotabularray` uses the `tblr` environment and `csvsim/autolongtabularray` uses the `longtblr` environment. The star variants treat the first line as data line and not as header line.

4.7 Special Characters

Be default, the CSV content is treated like normal LaTeX text, see Subsection 7.3 on page 52. For example, % can be used to start an in-line comment. But, TeX special characters of the CSV content may also be interpreted as normal characters (`\catcode` 12, other), if one or more of the following options are used.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

`csvsim/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.

`csvsim/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.

`csvsim/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.

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

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

4.8 Separators

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

Sets the ⟨*sign*⟩ which is treated 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, `csvsim/respect tab` → P. 33 is set also.

N 2023-05-08

- `space`: Sets the separator to space(s).

```
% \usepackage{tcolorbox} for tcbverbatimwrite
\begin{tcbverbatimwrite}{space.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=space]{space.csv}
```

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

Note that leading spaces are ignored and multiple spaces are treated as one space. To denote an empty data cell insert `{}`, e.g. `1␣{}␣3`.

4.9 Miscellaneous

csvsim/every csv (style, initially empty)

A meta key (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.
\csvstyle{every csv}{warn on column count error}
```

csvsim/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).

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

Sets the ⟨*file name*⟩ of the CSV file to be processed. `\csvreader` → P. 9 sets this option by a mandatory parameter.

csvsim/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.

csvsim/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 `csvsim/file`. The second argument is the preprocessed CSV file which is set by `csvsim/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. 9 or `\csvloop` → P. 10.

```
\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 4.10 on page 37 for a concrete sorting preprocessing implemented with an external tool.

csvsim/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.

4.10 Sorting

TEX/LATEX was not born under a sorting planet. `csvsimple-l3` provides no sorting of data lines by LATEX-methods[5] 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-l3`. For the third method, the `csvsim/preprocessor` → P. 36 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
https://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!**
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`.**

`csvsim/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}
```

`csvsim/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`.

`csvsim/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`.

[5] Nevertheless, Section 7.9 on page 65 provides an example of data storage and sorting using LATEX3 methods.

`csvsim/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`.

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

The ⟨*file name*⟩ denotes an XML configuration file for **CSV-Sorter**. Setting this option inside `\csvreader` → P. 9 or `\csvloop` → P. 10 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 `csvsim/preprocessed file` → P. 36 which is by default `\jobname_sorted.csv`. This output file is this actual file processed by `\csvreader` → P. 9 or `\csvloop` → P. 10.
- **CSV-Sorter** also generates a log file denoted by `csvsim/csvsorter log` → P. 37 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[
    head to column names,
    sort by    = namesort.xml,
    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[
    head to column names,
    sort by    = gradesort.xml,
    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
<?xml version="1.0" encoding="UTF-8"?>
<csv>
  <bracket empty="true" />
  <sortlines>
    <column name="matriculation" order="ascending" type="integer"/>
  </sortlines>
</csv>
```

```
% \usepackage{booktabs}
\csvreader[
    head to column names,
    sort by    = matriculationsort.xml,
    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

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

This is a convenience option to generate a new shortcut for often used `csvsim/sort by` → P. 38 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

N 2021-06-28 `\csvsortingrule{⟨name⟩}{⟨file name⟩}`

Identical in function to `csvsim/new sorting rule`, see above. A good place for setting up a new sorting rule would be inside the preamble:

```
\csvsortingrule{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

4.11 Data Collection

`csvsimple-l3` reads and processes a CSV file line by line. Accordingly, the TEX input stream is filled line by line. Although this is an efficient procedure, for some applications like tables with the `tabularray`→CTAN package, collecting the data from the CSV file into a macro is needed. This macro can be given to the target application for further processing.

N 2021-07-06
U 2023-10-17

`csvsim/collect data=true|false` (default `true`, initially `false`)

`csvsimple-l3` provides limited and experimental support to collect the input data from the CSV file plus user additions into a macro named `\csvdatacollection`→P. 43. Setting `csvsim/collect data` adds the contents of the following keys to `\csvdatacollection`→P. 43:

- `csvsim/after head`→P. 18
- `csvsim/after first line`→P. 19
- `csvsim/after line`→P. 19
- `csvsim/before first line`→P. 18
- `csvsim/before line`→P. 18
- `csvsim/late after first line`→P. 18
- `csvsim/late after head`→P. 18
- `csvsim/late after last line`→P. 18
- `csvsim/late after line`→P. 18

Also, the *expanded* content of

- `csvsim/command`→P. 19

is added to `\csvdatacollection`→P. 43 (depending on `csvsim/consume collected data`→P. 43 and `csvsim/data collection`→P. 43). Note that for `csvsim/command`→P. 19 special care has to be taken *what* should be protected from expansion and *what not*. Observe the following hints for `csvsim/command`→P. 19:

- For data macros like `\csvcoli` use `\csvexpval\csvcoli` to add the *value* of this macro to `\csvdatacollection`→P. 43. This is optional, if `\csvcoli` contains numbers or text without active characters, but essential, if it contains macros.
- `\csvlinetotablerow`→P. 13 is to be used *without* `\csvexpval`.
- For macros like `\textbf` use `\csvexpnot\textbf` to *prevent* expansion.
- Using computations or not expandable conditionals may likely cause compilation errors.

```
\csvreader[
    collect data,
    head to column names,
    late after line=\\,
    late after last line=,
  ]{grade.csv}{}{%
    \thecsvrow. \csvexpval\givenname\ \csvexpnot\textbf{\csvexpval\name}
  }
Collected data:\par
\csvdatacollection
```

Collected data:
1. Hans **Maier**
2. Anna **Huber**
3. Werner **Weißbäck**
4. Maria **Bauer**

Note that data collection is *limited* to some special cases and does not allow to save all possible content. Table options like `csvsim/longtable`→P. 29 are generally not supported with the important exception of `csvsim/tabularray`→P. 29 which uses `csvsim/collect data` automatically.
See Section 7.6 on page 59 for examples.

N 2023-12-18 **csvsim/consume collected data=true|false** (default **true**, initially **false**)

If set to **false**, the collected data of a CSV file processed with csvsim/collect data → P. 42 is saved into \csvdatacollection.
Otherwise, if set to **true**, the collected data is not saved, but directly used after reading the CSV file, see csvsim/generic collected table → P. 31. After usage, the collected data is cleared, i.e. \csvdatacollection is emptied.

N 2021-07-06
U 2024-05-16 **csvsim/data collection=**⟨*macro*⟩ (no default, initially \csvdatacollection)

Sets the collection macro to an alternative for \csvdatacollection.

```
data collection = \myData,    %  instead of \csvdatacollection
```

Note that until version 2.6.0 (2024/01/19), csvsim/data collection was not reset to the default \csvdatacollection for following CSV files, but it is now.

N 2021-07-06 **\csvdatacollection**

Macro which contains the collected data of a CSV file processed with csvsim/collect data → P. 42. This macro name can be changed by setting csvsim/data collection.

N 2021-07-06
U 2023-12-17 **\csvexpval**⟨*macro*⟩

Recovers the content of the given ⟨*macro*⟩ and prevents further expansion. This is a wrapper for **\exp_not:o**. Alternatively, **\expandonce** from etoolbox → CTAN could be used.

N 2021-07-06 **\csvexpnot**⟨*macro*⟩

Prevents the expansion of the given ⟨*macro*⟩. This is a wrapper for **\exp_not:N**. Alternatively, **\noexpand** could be used.

The following macros can only be used inside keys which are *not* collected to \csvdatacollection, e.g. inside csvsim/after filter → P. 18.

N 2021-07-06
U 2023-12-17 **\csvcollectn{**⟨*code*⟩**}**

Appends the given ⟨*code*⟩ to \csvdatacollection.
This corresponds to **\tl_build_gput_right:Nn**.

N 2021-07-06
U 2023-12-18 **\csvcollecte{**⟨*code*⟩**}**
\csvcollectx{⟨*code*⟩**}**

Appends the expansion of the given ⟨*code*⟩ to \csvdatacollection.
This corresponds to **\tl_build_gput_right:Ne**.
\csvcollectx is an alias for \csvcollecte and is kept for backward compatibility.

N 2021-07-06
U 2023-12-17 **\csvcollectV**⟨*macro*⟩

Appends the content of the given ⟨*macro*⟩ to \csvdatacollection.
This corresponds to **\tl_build_gput_right:Ne** and **\exp_not:o** for ⟨*macro*⟩.

5 String and Number Tests

The following string and number tests are, to some extent, provided for backward compatibility. Mainly, they are wrappers for corresponding `expl3` conditionals. Therefore, you are encouraged to use the following CamelCase macros like `\IfCsvsimStrEqualTF` which provide by their name insight to the underlying `expl3` functions. The lowercase variants are kept for backward compatibility.

N 2016-07-01
U 2023-12-19

`\IfCsvsimStrEqualTF{`⟨*string A*⟩`}{`⟨*string B*⟩`}{`⟨*true*⟩`}{`⟨*false*⟩`}`
`\ifcsvstrcmp{`⟨*string A*⟩`}{`⟨*string B*⟩`}{`⟨*true*⟩`}{`⟨*false*⟩`}`

Compares two strings and executes ⟨*true*⟩ if they are equal, and ⟨*false*⟩ otherwise. The comparison is done using `\str_if_eq:eeTF`. `\IfCsvsimStrEqualTF` is expandable. Typically, this is the preferred function for many use cases.

N 2016-07-01
U 2021-06-28

`\ifcsvnotstrcmp{`⟨*string A*⟩`}{`⟨*string B*⟩`}{`⟨*true*⟩`}{`⟨*false*⟩`}`

Compares two strings and executes ⟨*true*⟩ if they are *not* equal, and ⟨*false*⟩ otherwise. The implementation uses `\IfCsvsimStrEqualTF`. `\ifcsvnotstrcmp` is expandable. Consider using `\IfCsvsimStrEqualTF` alternatively.

N 2016-07-01
U 2023-12-19

`\IfCsvsimTlEqualTF{`⟨*token list A*⟩`}{`⟨*token list B*⟩`}{`⟨*true*⟩`}{`⟨*false*⟩`}`
`\ifcsvstrequal{`⟨*token list A*⟩`}{`⟨*token list B*⟩`}{`⟨*true*⟩`}{`⟨*false*⟩`}`

Compares two token lists and executes ⟨*true*⟩ if they are equal, and ⟨*false*⟩ otherwise. The comparison is done using `\tl_if_eq:eeTF`. `\IfCsvsimTlEqualTF` is not expandable. If you have no special reason for using a token list comparison, where characters and category codes of those characters are compared, you may rather choose `\IfCsvsimStrEqualTF`.

N 2016-07-01
U 2023-12-19

`\IfCsvsimTlProtectedEqualTF{`⟨*token list A*⟩`}{`⟨*token list B*⟩`}{`⟨*true*⟩`}{`⟨*false*⟩`}`
`\ifcsvprostrequal{`⟨*token list A*⟩`}{`⟨*token list B*⟩`}{`⟨*true*⟩`}{`⟨*false*⟩`}`

Compares two token lists and executes ⟨*true*⟩ if they are equal, and ⟨*false*⟩ otherwise. The token lists are expanded with `\protected@edef` in the test, i.e. parts of the token lists which are protected stay unexpanded. The comparison is done using `\tl_if_eq:NNTF`. `\IfCsvsimTlProtectedEqualTF` is not expandable.

N 2021-06-28
U 2023-12-19

`\IfCsvsimFpCompareTF{`⟨*floating point comparison*⟩`}{`⟨*true*⟩`}{`⟨*false*⟩`}`
`\ifcsvfpcmp{`⟨*floating point comparison*⟩`}{`⟨*true*⟩`}{`⟨*false*⟩`}`

Evaluates the given ⟨*floating point comparison*⟩ and executes ⟨*true*⟩ or ⟨*false*⟩ appropriately. The evaluation is done using `\fp_compare:nTF`.
Basically, a ⟨*floating point comparison*⟩ consists of ⟨*fp expr*$_1$⟩ ⟨*relation*⟩ ⟨*fp expr*$_2$⟩, like $x < y$, but `\fp_compare:nTF` even allows a chain of comparisons. `\IfCsvsimFpCompareTF` is expandable.

N 2021-06-28
U 2023-12-19

`\IfCsvsimIntCompareTF{`⟨*integer comparison*⟩`}{`⟨*true*⟩`}{`⟨*false*⟩`}`
`\ifcsvintcmp{`⟨*integer comparison*⟩`}{`⟨*true*⟩`}{`⟨*false*⟩`}`

Evaluates the given ⟨*integer comparison*⟩ and executes ⟨*true*⟩ or ⟨*false*⟩ appropriately. The evaluation is done using `\int_compare:nTF`.
Basically, a ⟨*integer comparison*⟩ consists of ⟨*int expr*$_1$⟩ ⟨*relation*⟩ ⟨*int expr*$_2$⟩, like $x < y$, but `\int_compare:nTF` even allows a chain of comparisons. `\IfCsvsimIntCompareTF` is expandable.

6 Hooks

The following hook(s) are present following LaTeX's hook management.

N 2023-05-08 **csvsimple/csvline** This hook adds code after reading a line into `\csvline` → P. 12 and before processing this line. The token list `\csvline` → P. 12 may be manipulated with a global assignment.

The following example replaces every `"..."` by `{...}` to approximate double-quote processing within LaTeX. Still, masking of double-quotes or nesting will not work.

```
\AddToHook{csvsimple/csvline}
  {
    \tl_set_eq:NN \l_tmpa_tl \csvline
    \regex_replace_all:nnN { "([^"]+)" } { {\1} } \l_tmpa_tl
    \tl_gset_eq:NN \csvline \l_tmpa_tl
  }
```

7 Examples

7.1 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]{\IfCsvsimStrEqualTF{\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}]
  \IfCsvsimStrEqualTF{\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.
...

7.2 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
```

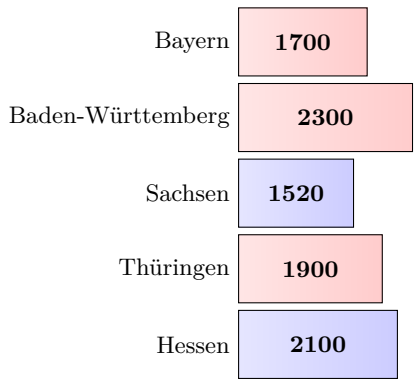
Firstly, we survey the file content using `\csvautobooktabular`.

```
% 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}
```



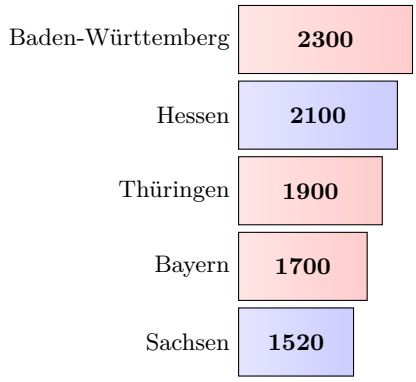
It would be nice to sort the bars by length, i. e. to sort the CSV file by the `amount` column. If the **CSV-Sorter** program is properly installed, see Subsection 4.10 on page 37, this can be done with the following configuration file for **CSV-Sorter**:

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>
```

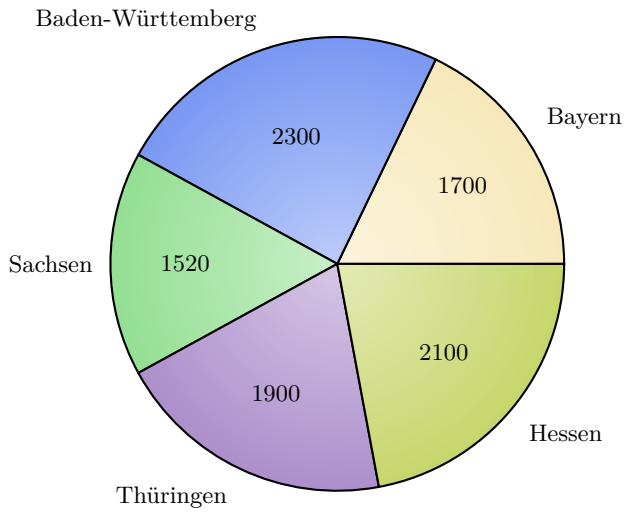
Now, we just have to add an option `sort by=amountsort.xml`:

```
% 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}
```



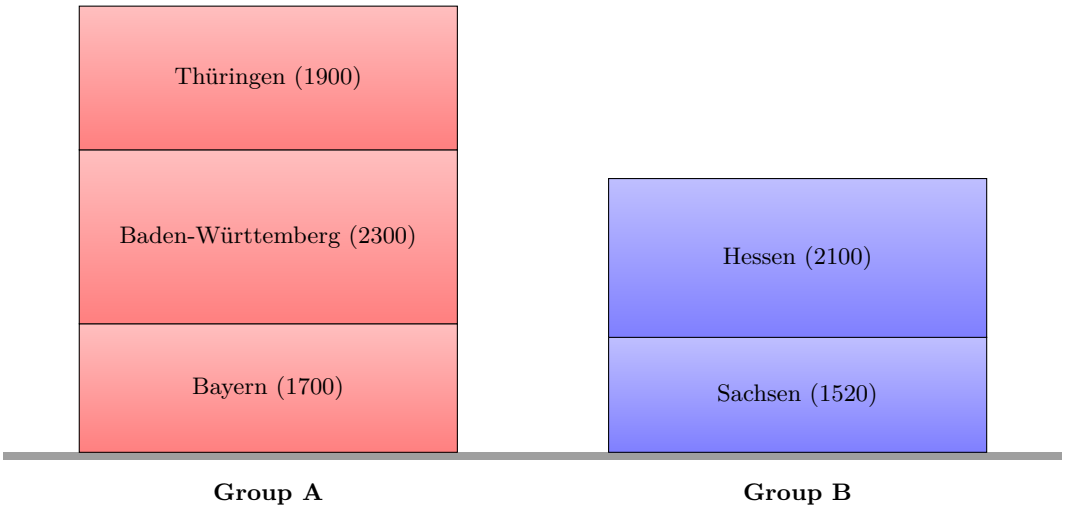
Next, we create a pie chart by calling `\csvreader` twice. In the first step, the total sum of amounts is computed, and in the second step the slices are drawn.

```
% 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}
```



7.3 Macro code inside the data

If needed, the data file may contain macro code.

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 strcmp={\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: $\displaystyle \lim_{n\to\infty} \frac{1}{n} = 0$

7.4 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 CTAN package provides a huge amount of formatting options for numbers. A good and robust way to apply formatting by siunitx CTAN inside tables generated by `csvsimple-l3` is the `\tablenum` macro from siunitx CTAN.

```
% \usepackage{siunitx,array,booktabs}
\csvreader[
    head to column names,
    before reading = \begin{center}\sisetup{table-number-alignment=center},
    tabular        = cc,
    table head     = \toprule \textbf{Cats} & \textbf{Dogs} \\\midrule,
    table foot     = \bottomrule,
    after reading  = \end{center}
  ]{data_numbers.csv}{}{%
    \tablenum[table-format=2.2e1]{\cats} & \tablenum{\dogs}
  }
```

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

It is also possible to create on-the-fly tables using calculations of the given data. The following example shows cat values bisected and dog values doubled.

```
% \usepackage{siunitx,array,booktabs,xfp}
\csvreader[
    head to column names,
    before reading = \begin{center}\sisetup{table-number-alignment=center},
    tabular        = cccc,
    table head     = \toprule \textbf{Cats} & \textbf{Dogs}
                     & \textbf{Halfcats} & \textbf{Doubledogs} \\\midrule,
    table foot     = \bottomrule,
    after reading  = \end{center}
  ]{data_numbers.csv}{}{%
    \tablenum[table-format=2.2e1]{\cats} & \tablenum{\dogs}
      & \tablenum[exponent-mode=scientific, round-precision=3,
          round-mode=places, table-format=1.3e1]{\fpeval{\cats/2}}
      & \tablenum{\fpeval{\dogs*2}}
  }
```

Cats	Dogs	Halfcats	Doubledogs
12.3×10^{5}	12.50	6.150×10^{5}	25
8.7×10^{3}	3.32	4.350×10^{3}	6.64
3.1×10^{6}	43	1.550×10^{6}	86
21.2×10^{4}	0.33	1.060×10^{5}	0.66
3.45×10^{6}	5.12	1.725×10^{6}	10.24
6.66×10^{6}	6.44	3.330×10^{6}	12.88
7.3×10^{7}	123.2	3.650×10^{7}	246.4
5.3×10^{4}	12.3	2.650×10^{4}	24.6
4.4×10^{4}	2.3	2.200×10^{4}	4.6
6.5×10^{6}	6.5	3.250×10^{6}	13
5.5×10^{5}	0.55	2.750×10^{5}	1.1
3.3×10^{3}	2.2	1.650×10^{3}	4.4

The `siunitx` package also provides a new column type `S` which can align material using a number of different strategies. 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-l3` which actually is present at the first and last column. To avoid this problem, the utilization of `\tablenum` is appropriate, see above. Alternatively, a 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}
\csvreader[
    head to column names,
    before reading = \begin{center}\sisetup{table-number-alignment=center},
    tabular        = {c@{}S[table-format=2.2e1]S@{}c},
    table head     = \toprule & \textbf{Cats} & \textbf{Dogs} & \\\midrule,
    table foot     = \bottomrule,
    after reading  = \end{center}
  ]{data_numbers.csv}{}{%
    & \cats & \dogs &
  }
```

	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 4.10 on page 37, 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
\csvreader[
    head to column names,
    sort by        = catsort.xml,
    before reading = \begin{center}\sisetup{table-number-alignment=center},
    tabular        = lcc,
    table head     = \toprule \textbf{Month} & \textbf{Dogs} & \textbf{Cats} \\\midrule,
    table foot     = \bottomrule,
    after reading  = \end{center}
  ]{data_numbers.csv}{}{%
    \month & \tablenum{\dogs} & \tablenum[table-format=2.2e1]{\cats}
  }
```

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}

7.5 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 `csvsim/no head` → P. 20 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, the *star* versions of the auto tabular commands can be used.

```
\csvautobooktabular*{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
  ]{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. 9 command, the csvsim/no head → P. 20 option should be applied. Of course, we cannot use csvsim/head to column names → P. 20 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

7.6 Tables with tabularray

The tabularray[→ CTAN] package gives extended control for generating tables. csvsim/tabularray[→ P. 29] and csvsim/centered tabularray[→ P. 29] support such tables. A distinctiveness is that for tabularray[→ CTAN] data from a CSV file has to be *collected* first (into a macro) and applied afterwards. The process is hidden from the user view, but has to be taken into account when csvsim/command[→ P. 19] is set up, see Section 4.11 on page 42.

The following examples uses `data.csv` from Section 7.2 on page 48.

```
% \usepackage{tabularray,siunitx,xfp}
\csvreader[
    head to column names,
    centered tabularray =
      {
        rowsep = 1mm,
        colsep = 5mm,
        rows   = {blue7},
        hlines = {2pt, white},
        vlines = {2pt, white},
        row{1} = {bg=azure3, fg=white, font=\bfseries\large, 8mm},
      },
    table head = {\SetCell[c=4]{c} Important Data Table \\},
  ]{data.csv}{}{
       \IfCsvsimStrEqualTF{\group}{A}{\csvexpnot\SetRow{brown7}}{}
       \csvexpnot\SetCell{bg=purple7}
       \csvexpval\land
     & \csvexpval\group
     & \csvexpval\amount
     & \tablenum[exponent-mode=scientific, round-precision=3,
         round-mode=places, table-format=1.3e1]{\fpeval{pi*\amount}}
  }
```

Important Data Table			
Bayern	A	1700	5.341×10^3
Baden-Württemberg	A	2300	7.226×10^3
Sachsen	B	1520	4.775×10^3
Thüringen	A	1900	5.969×10^3
Hessen	B	2100	6.597×10^3

Note in the example above that

- csvsim/table head[→ P. 30] is *collected* unexpanded, i.e. `\SetCell` has not to be protected. On the other hand, CSV data could not be used here.
- csvsim/command[→ P. 19] is *collected* expanded. This is identical to the mandatory last argument of `\csvreader`[→ P. 9].
  - Therefore, expansion of `\SetRow`, `\SetCell`, etc. is prevented by `\csvexpnot`[→ P. 43].
  - The *values* (content) of `\land`, `\group`, etc. are recovered by `\csvexpval`[→ P. 43].
  - `\IfCsvsimStrEqualTF` and `\fpeval` are *expandable* and therefore the results of these commands are *collected*.
  - `\tablenum` from siunitx[→ CTAN] is a robust command and therefore needs no protection. If you are not sure, if a command is robust or not, it does not hurt add the prefix `\csvexpnot`[→ P. 43], i.e. use `\csvexpnot\tablenum`.

Filters and line ranges can be used for tabularray[→ CTAN] and all data collections without restriction:

```
% \usepackage{tabularray}
Display group `A` only:\par
\csvreader[
    head to column names,
    filter strcmp = {\group}{A},
    centered tabularray =
      {
        rowsep = 1mm,
        colsep = 5mm,
        column{1} = {r, fg=yellow5, colsep=2pt},
        column{2} = {r, yellow8!10, font=\bfseries},
        column{3} = {l, yellow8},
        hlines    = {2pt, white},
      },
  ]{data.csv}{}{
       \thecsvrow
    & \csvexpval\land
    & \csvexpval\amount
  }
```

Display group ‘A‘ only:

1	**Bayern**	1700
2	**Baden-Württemberg**	2300
3	**Thüringen**	1900

```
% \usepackage{tabularray}
Display data from line 3 on:\par
\csvreader[
    head to column names,
    range = 3-,
    centered tabularray =
      {
        rowsep = 1mm,
        colsep = 5mm,
        column{1} = {r, fg=violet5, colsep=2pt},
        column{2} = {r, violet8!10, font=\bfseries},
        column{3} = {l, violet8},
        hlines    = {2pt, white},
      },
  ]{data.csv}{}{
       \thecsvrow
    & \csvexpval\land
    & \csvexpval\amount
  }
```

Display data from line 3 on:

3	**Sachsen**	1520
4	**Thüringen**	1900
5	**Hessen**	2100

The following example uses \csvautotabularray → P. 16 to display the whole table. Note that the tabularray → CTAN options are given as last optional argument.

```
% \usepackage{tabularray}
\csvautotabularray[table centered]{data.csv}
  [
    row{odd}    = {blue!85!gray!7},
    row{1}      = {blue!50!gray!25, font=\bfseries, preto=\MakeUppercase},
    hline{1,Z} = {0.1em, blue!50!black},
    hline{2}    = {blue!50!black}
  ]
```

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

7.7 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 4.10 on page 37, 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-l3` 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}
```

7.8 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-l3`:

```
% !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 4.10 on page 37, 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}
```

7.9 Storing Data in L3 Property Lists

Instead of processing data from CSV files immediately, it can be stored in LaTeX3 data structures for later use.

As an example, the data from the frequently used file "grade.csv" introduced in Section 1.3 on page 4 is stored in L3 property lists.

1. As a first step, the property lists are created, for example `\l__mytest_matnumber_prop`. The property keys are stored in the sequence `\l__mytest_key_seq`.
2. In the second step, the CSV file is read and the storage setup is applied. Note that the actual file reading takes place outside the expl3 context.
3. As a continuation of the example, the stored data is processed. Here, the key sequence is sorted according to the matriculation numbers.
4. As a final step, the stored and sorted data is used to construct a tabularray[→ CTAN] table. Note that the complete table is first assembled in `\l__mytest_table_tl`, and that this token list is used at the very end of the code.

```
% \usepackage{ninecolors,tabularray}

% Step 1: setup storing data to property lists using the line number as key
\ExplSyntaxOn
\csvstyle{ myproperties }
  {
    before~reading =
      {
        \tl_clear_new:N  \l__mytest_key_tl
        \seq_clear_new:N \l__mytest_key_seq
        \prop_clear_new_linked:N \l__mytest_firstname_prop
        \prop_clear_new_linked:N \l__mytest_name_prop
        \prop_clear_new_linked:N \l__mytest_matnumber_prop
      },
    column~names =
      {
        1 = \name,
        2 = \firstname,
        3 = \matnumber
      },
    command =
      {
        \tl_set:Ne \l__mytest_key_tl { \int_use:N \g_csvsim_row_int }
        \seq_put_right:NV \l__mytest_key_seq    \l__mytest_key_tl
        \prop_put:NVV \l__mytest_firstname_prop \l__mytest_key_tl \firstname
        \prop_put:NVV \l__mytest_name_prop      \l__mytest_key_tl \name
        \prop_put:NVV \l__mytest_matnumber_prop \l__mytest_key_tl \matnumber
      }
  }
\ExplSyntaxOff

% Step 2: read an store the csv file
\csvloop
  {
    file = grade.csv,
    myproperties
  }

% Step 3: sort key sequence by matnumber
\ExplSyntaxOn
\seq_sort:Nn \l__mytest_key_seq
  {
    \int_compare:nNnTF
```

```
      { \prop_item:Nn \l__mytest_matnumber_prop { #1 } }
      >
      { \prop_item:Nn \l__mytest_matnumber_prop { #2 } }
      { \sort_return_swapped: }
      { \sort_return_same: }
  }

% Step 4: data output as tabularray table
\par
\tl_clear_new:N \l__mytest_table_tl
\tl_build_begin:N \l__mytest_table_tl
\tl_build_put_right:Nn \l__mytest_table_tl
  {
    \begin{tblr}
      {
        colspec    = { rcllXr },
        row{even}  = { bg=azure9 },
        row{1}     = { bg=azure1, fg=white, font=\bfseries\large, ht=8mm },
        row{2}     = { bg=azure2, fg=white, font=\bfseries },
        cell{1}{1} = { c=6 }{ halign=c },
      }
    My~sorted~example~table \\
      & Matr. No. & Name & First~Name & Etc. & CSV~line\\
  }
\seq_map_indexed_inline:Nn \l__mytest_key_seq
  {
    \tl_build_put_right:Nn \l__mytest_table_tl { #1 & }
    \tl_build_put_right:Ne \l__mytest_table_tl
      { \prop_item:Nn \l__mytest_matnumber_prop { #2 } & }
    \tl_build_put_right:Ne \l__mytest_table_tl
      { \prop_item:Nn \l__mytest_name_prop { #2 } & }
    \tl_build_put_right:Ne \l__mytest_table_tl
      { \prop_item:Nn \l__mytest_firstname_prop { #2 } & }
    \tl_build_put_right:Nn \l__mytest_table_tl { \ldots & }
    \tl_build_put_right:Nn \l__mytest_table_tl { #2 \\ }
  }
\tl_build_put_right:Nn \l__mytest_table_tl
  {
    \end{tblr}
  }
\tl_build_end:N \l__mytest_table_tl
\l__mytest_table_tl
\ExplSyntaxOff
```

My sorted example table

	Matr.No.	Name	First Name	Etc.	CSV line
1	12345	Maier	Hans	...	1
2	19202	Bauer	Maria	...	4
3	23456	Huber	Anna	...	2
4	34567	Weißbäck	Werner	...	3

8 Differences between csvsimple-l3 and csvsimple-legacy

This section is intended for users who know csvsimple before version 2.00.

csvsimple-l3 (aka csvsimple as described in this document) is a *nearly* drop-in replacement for csvsimple-legacy. Although old documents have no *need* to be changed, adopting the new LaTeX3 version for existing documents should impose not too much effort. Actually, it depends on how intense pgfkeys specific styles were used.

That brings us to the differences between the two packages and a more precise understanding what *nearly* drop-in replacement means. The following enumeration does not list new features of csvsimple-l3 (if any), but takes an upgrade point of view.

- Any patches or additions using undocumented internals of csvsimple-legacy will stop to function, because csvsimple-l3 has a completely new implementation.
- csvsimple-l3 is programmed using the L3 programming layer. No additional packages are loaded or needed with exception of several options which allow to access methods from ifthen, etoolbox, longtable, etc. On the other hand, csvsimple-legacy is programmed in LaTeX 2_ε with dirty tricks from here and there.
- The most significant change of the user interface is that the key value engine of csvsimple-legacy is pgfkeys (root /csv/) while csvsimple-l3 uses l3keys (root csvsim/). Names and usage of the keys are *unchanged*. But, if you made own pgfkeys *styles* using the pgfkeys style handler, these *styles* have to be adapted to .meta keys of the L3 programming layer. The good news is that styles made with \csvstyle → P. 10 become .meta keys automatically.
- The macro \csvheadset is removed. It is not supportable by the new implementation. I never used it and I forgot why I ever wrote it – I hope the same is true for you. If not, csvsimple-legacy can be used for documents which needs it.
- Option /csv/filter is removed. Instead, csvsim/filter ifthen → P. 26 can be used (also true with /csv/filter ifthen for the old version).
- The deprecated options /csv/nofilter and /csv/nohead are removed. They were not documented any more since years. Obviously, use csvsim/no filter → P. 26 and csvsim/no head → P. 20 instead.
- Compilation problems are to be expected, if an S column of the siunitx package is used as first or last column. Documents neglecting this rule successfully for csvsimple-legacy, may fail to compile with csvsimple-l3.
- The LaTeX counters csvinputline and csvrow are replaced by LaTeX3 integers \g_csvsim_inputline_int → P. 13 and \g_csvsim_row_int → P. 13, but accessors \thecsvinputline → P. 13 and \thecsvrow → P. 13 are still valid.
- The packages pgfrcs, pgfkeys, ifthen, etoolbox, and shellesc are not included anymore (include manually, if needed).
- \csviffirstrow and \csvifoddrow are deprecated and replaced by \ifcsvfirstrow → P. 12 \ifcsvoddrow → P. 11 which are more consistent in nomenclature.
- For csvsimple-l3, data lines are allowed to begin with an backslash.
- Assigned macros like \myname for e.g. the third column contain not \csvcoliii anymore, but are equal to the content of \csvcoliii now.
- Character code changes with csvsim/respect percent → P. 33 etc. and the tabulator as separator should work for csvsimple-l3 as expected in every situation (not always worked

for `csvsimple-legacy`).

- A drawback of `csvsimple-l3` against `csvsimple-legacy` is a higher compilation time. This may vary by used compiler. An example document of 5061 pages using a CSV file with 166 992 lines took about 28 seconds with `csvsimple-legacy` and about 51 seconds with `csvsimple-l3` on my machine (just a singular observation, no scientific analysis at all).

Index