csvtools v1.24 : A LaTeX 2ε Package Providing Access to Data Saved in a CSV File

Nicola Talbot

3rd July 2007

Contents

1 Introduction 2

2 Mail Merging and Similar Applications 3

3 Converting data in a CSV file into a tabular environment 4

4 Converting CSV file into longtable environment 7

5 Associated Counters 8

6 Cross-Referencing 9

7 Saving Entries 12

8 Pie Charts (csvpie.sty) 14

9 Sorting Data (csvsort.sty) 16

10 The csvtools.pl Perl Script 18
  10.1 Notes 19

11 Bugs/Drawbacks/“Features” 20

12 Contact Details 21

Index 21

List of Examples

1 Mail Merging 3

2 Multiple Figures 4

3 Mail Merging using `\field` 4

4 Aligning Data from a CSV file 5

5 Adding Lines **5**

6 Added Complexity **6**

7 Using a longtable environment **7**

8 Stripy Table **8**

9 More Mail Merging **8**

10 Labelling within `\applyCSVfile` **9**

11 Labelling within `\applyCSVfile` **9**

12 Labelling within `\CSVtotabular` **10**

13 Labelling within `\CSVtotabular` **10**

14 Saving Entries **12**

15 A Pie Chart **15**

16 csvtools.pl — Aligning Data **19**

1 Introduction

The csvtools package allows you to repeatedly perform a set of LaTeX commands on data in each row of a comma separated variable (CSV) file. This can be used for mail merging, generating tables etc.

As from version 1.2, you can specify a different separator. To change the separator, use the command:

`\setcsvseparator`

`\setcsvseparator{`⟨*separator*⟩`}`

For example, if your data is separated by colons instead of commas, do:

```
\setcsvseparator{:}
```

If your separator occurs within an entry, the entry must be enclosed in double quotes, for example:

```
Name,Address,Telephone
A.N. Other,"1 The Street,The Town",0123456789
```

Be careful of TeX special characters occuring within a CSV file, for example:

```
Name,Address,Telephone
Jack \& Jill,"2 The Street,The Town",0123456789
```

2 Mail Merging and Similar Applications

`\applyCSVfile`

`\applyCSVfile`[⟨*n*⟩]{⟨*filename*⟩}{⟨*text*⟩}
`\applyCSVfile*`[⟨*n*⟩]{⟨*filename*⟩}{⟨*text*⟩}

Letters can be generated using data given in each line from ⟨*filename*⟩. If the CSV file contains a header row, the unstarred version of `\applyCSVfile` should be used, otherwise the starred version `\applyCSVfile*` should be used. The optional argument ⟨*n*⟩ specifies on which line the actual data (not header line) starts. The unstarred version defaults to line 2 (the header row is always assumed to be on line 1) and the starred version defaults to 1.

`\insert...`

With the unstarred version, the entries in the header row are used to generate commands of the form `\insert`⟨*identifier*⟩[1] to access corresponding elements in the row currently being processed. For example, suppose the first line of the CSV file looks like:

```
Name,Address,Time,Date
```

`\insertbyname`

then the commands `\insertName`, `\insertAddress`, `\insertTime` and `\insertDate` are created, allowing you to use the entries in the first, second, third and fourth columns of the current row. If the header text contains non-alphabetical characters, e.g. `Full Name`, then you will need to use `\insertbyname`{⟨*text*⟩}, e.g. `\insertbyname{Full Name}`.

`\field`

Alternatively, you can use the `\field`{⟨*col*⟩} command, where ⟨*col*⟩ is the column number of the entry, so `\field{1}` indicates the first entry in the current row and `\field{2}` indicates the second entry in the current row.

Example 1 (Mail Merging)

Suppose there is a file called `details.csv` that has the following contents:

```
Name,Address,Time,Date
Miss A. Person,1 The Road\\The Town\\AB1 2XY,15.00,4th May 2004
Mr A. N. Other,2 The Road\\The Town\\AB1 2XY,15.30,11th May 2004
```

then the following code can be used to generate a letter for each person in the CSV file[2]:

```
\applyCSVfile{details.csv}{%
\begin{letter}{\insertName\\\insertAddress}
\opening{Dear \insertName}

You are invited to an interview at \insertTime\ on the \insertDate.

\closing{Yours Sincerely}
\end{letter}}
```

Note that you could also use `\insertbyname{Name}` etc instead of `\insertName` etc. Also note that you need to specify the file extension when specifying the filename.

---

[1] See Note 1 in Section 11

[2] Remeber to use a letter type of class file

Example 2 (Multiple Figures)

Suppose `sample3.csv` looks like:

```
File,Caption
circle.ps,A Circle
rectangle.ps,A Rectangle
triangle.ps,A Triangle
```

Assuming that the files `circle.ps`, `rectangle.ps` and `triangle.ps` exist, then the following code will generate a figure for each graphics file[3]:

```
\applyCSVfile{sample3.csv}{
\begin{figure}
\centerline{\includegraphics{\insertFile}}
\caption{\insertCaption}
\end{figure}}
```

Note that in this example, you can't use `\insertbyname{File}`. (See Note 3 in Section 11.)

Example 3 (Mail Merging using `\field`)

Suppose there is a file called `details.csv` that has the following contents:

```
Miss A. Person,1 The Road\\The Town\\AB1 2XY,15.00,4th May 2004
Mr A. N. Other,2 The Road\\The Town\\AB1 2XY,15.30,11th May 2004
```

In this case the data has no header file, so the starred version of `\applyCSVfile` must be used. Since there is no header file, you must use `\field` to access the entries:

```
\applyCSVfile*{details.csv}{%
\begin{letter}{\field{1}\\\field{2}}
\opening{Dear \field{1}}

You are invited to an interview at \field{3}\ on the \field{4}.

\closing{Yours Sincerely}
\end{letter}}
```

---

3 Converting data in a CSV file into a tabular environment

`\CSVtotabular` `\CSVtotabular`{⟨*filename*⟩}{⟨*col-align*⟩}{⟨*first*⟩}{⟨*middle*⟩}{⟨*last*⟩}

⟨*filename*⟩ is the name of the CSV file which must have a header row on line 1, ⟨*col-align*⟩ is the column alignment argument that gets passed to the tabular environment, ⟨*first*⟩ is the code for the first line, ⟨*middle*⟩ is the code for the middle lines and ⟨*last*⟩ is the code for the last line. This is best demonstrated with an example.

---

[3] The graphics or graphicx package will be needed.

Table 1: Example 4

Name	**Assignment 1**	**Assignment 2**	**Total**
A. Smith	80	70	150
B. Jones	60	80	140
J. Doe	85	75	160
	75	75	150

Example 4 (Aligning Data from a CSV file)

Suppose the file `sample.csv` looks like:

```
Name,Assignment 1,Assignment 2,Total
A. Smith,80,70,150
B. Jones,60,80,140
J. Doe,85,75,160
,75,75,150
```

then the following code can be used to align the data:

```
\CSVtotabular{sample.csv}{lccc}{%
\bfseries Name &
\bfseries Assignment 1&
\bfseries Assignment 2&
\bfseries Total\\}{%
\insertName &
\insertbyname{Assignment 1} &
\insertbyname{Assignment 2} &
\insertTotal\\}{%
 &
\insertbyname{Assignment 1} &
\insertbyname{Assignment 2} &
\insertTotal}
```

The result of this code is shown in Table 1[4].

`\ifnextrowlast` `\ifnextrowlast{`⟨*last-code*⟩`}{`⟨*not-last-code*⟩`}`

The command `\ifnextrowlast` can be used to vary what happens on the last but one row. The following example illustrates this by placing `\hline\hline` after the penultimate row.

Example 5 (Adding Lines)

```
\CSVtotabular{sample.csv}{|l|ccc|}{%
\hline\bfseries Name &
\bfseries Assignment 1&
\bfseries Assignment 2&
\bfseries Total\\\hline\hline}{%
\insertName &
\insertbyname{Assignment 1} &
```

---

[4] Note that `\CSVtotabular` only puts the data in a `tabular` environment not in a table

Table 2: Example 5

Name	**Assignment 1**	**Assignment 2**	**Total**
A. Smith	80	70	150
B. Jones	60	80	140
J. Doe	85	75	160
	75	75	150

```
\insertbyname{Assignment 2} &
\insertTotal
\ifnextrowlast{\\\hline\hline}{\\}}{%
 &
\insertbyname{Assignment 1} &
\insertbyname{Assignment 2} &
\insertTotal\\\hline}
```

This result of this code is shown in Table 2.

Example 6 (Added Complexity)

In this example, `\multicolumn` is used to override the column specifier for the first column in the last row.

```
\CSVtotabular{sample2.csv}{|l|ccc|}{%
\hline\bfseries Name &
\bfseries Assignment 1 &
\bfseries Assignment 2 &
\bfseries Total\\\hline\hline
}{%
\insertName &
\insertbyname{Assignment 1} &
\insertbyname{Assignment 2} &
\insertTotal
\ifnextrowlast{\\\hline\multicolumn{1}{l|}{}}{\\}
}{%
 &
\insertbyname{Assignment 1} &
\insertbyname{Assignment 2} &
\insertTotal\\\cline{2-4}
}
```

Notice that instead of placing `\multicolumn{1}{l|}{}` at the start of the final argument, it is instead placed in the first argument to `\ifnextrowlast`[5]. The result of this code is shown in Table 3.

---

[5] See Note 4 in Section 11

Table 3: Example 6

Name	Assignment 1	Assignment 2	Total
A. Smith	80	70	150
B. Jones	60	80	140
J. Doe	85	75	160
	75	75	150

4 Converting CSV file into longtable environment

`\CSVtolongtable`

The command `\CSVtolongtable` works in the same way as `\CSVtotabular` but creates a longtable environment instead of a tabular environment.

Example 7 (Using a longtable environment)

Suppose the CSV file in the previous example, contains, say, 100 entries. This will no longer fit onto one page, so it would be better to use CSVtolongtable instead. For example:

```
\CSVtolongtable{sample.csv}{|l|ccc|}{%
\caption{Student Marks}\label{tab:students}\\
\hline
\bfseries Name &
\bfseries Assignment 1 &
\bfseries Assignment 2 &
\bfseries Total\\\hline
\endfirsthead
\caption[]{Student Marks}\\
\hline
\bfseries Name &
\bfseries Assignment 1 &
\bfseries Assignment 2 &
\bfseries Total\\\hline
\endhead
\hline
\multicolumn{3}{r}{\em Continued on next page}
\endfoot
\hline
\endlastfoot}{%
\insertName &
\insertbyname{Assignment 1} &
\insertbyname{Assignment 2} &
\insertTotal
\ifnextrowlast{\\\hline\hline}{\\}}{%
 & \insertbyname{Assignment 1} &
 \insertbyname{Assignment 2} &
\insertTotal\\}
```

5 Associated Counters

`csvlinenum`
`csvrownumber`

Within the `\CSVtotabular`, `\CSVtolongtable` and `\applyCSVfile` commands, there are two counters, `csvlinenum` and `csvrownumber`. The former, `csvlinenum`, is the current line number in the CSV file, whereas the latter, `csvrownumber`, is the current data row. Of the two counters, `csvrownumber` is likely to be the most useful.

Example 8 (Stripy Table)

David Carlisle's colortbl package defines the command `\rowcolor` which enables you to specify the row colour. Suppose you want a stripy table[6], this can be achieved as follows:

```
\CSVtotabular{sample2.csv}{lccc}{%
\rowcolor{green}\bfseries Name &
\bfseries Assignment 1 &
\bfseries Assignment 2 &
\bfseries Total\\\rowcolor{blue}
}{%
\insertName &
\insertbyname{Assignment 1} &
\insertbyname{Assignment 2} &
\insertTotal
\ifthenelse{\isodd{\value{csvrownumber}}}{%
\\\rowcolor{green}}{\\\rowcolor{blue}}
}{%
 &
\insertbyname{Assignment 1} &
\insertbyname{Assignment 2} &
\insertTotal
}
```

The resulting table is illustrated in Table 4.

Table 4: Stripy Table Example

Name	**Assignment 1**	**Assignment 2**	**Total**
A. Smith	80	70	150
B. Jones	60	80	140
J. Doe	85	75	160
	75	75	150

Example 9 (More Mail Merging)

This is an example of mail merging where the letter reference is generated from the value of `csvrownumber`. The CSV file is as used in Example 1 on page 3.

---

[6] This is designed as an example of how to use the package, not incouragement to produce garish tables!

```
\applyCSVfile{details.csv}{%
\begin{letter}{\insertName\\\insertAddress}
\opening{Dear \insertName}

\textbf{Ref : } interview.\thecsvrownumber

You are invited to an interview at \insertTime\ on the \insertDate.

\closing{Yours Sincerely}
\end{letter}}
```

6 Cross-Referencing

Labels can be generated using the standard `\label` command, but you will need some way to make each label unique. Example 10 does this by using `\thecsvrownumber`, whereas Example 11 uses `\insert`⟨*identifier*⟩.

Example 10 (Labelling within `\applyCSVfile`)

Example 2 on page 3 can be modified to label each figure:

```
\applyCSVfile{sample3.csv}{
\begin{figure}
\centerline{\includegraphics{\insertFile}}
\caption{\insertCaption}
\label{fig:pic\thecsvrownumber}
\end{figure}}
```

This example uses `\label{fig:pic\thecsvrownumber}`, so the first figure generated by this `\applyCSVfile` command will have the label `fig:pic1`, the second `fig:pic2` etc.

Example 11 (Labelling within `\applyCSVfile`)

Modifying the previous example, we now have:

```
\applyCSVfile{sample3.csv}{
\begin{figure}
\centerline{\includegraphics{\insertFile}}
\caption{\insertCaption}
\label{fig:\insertFile}
\end{figure}}
```

The labels for each figure are now: `fig:circle.ps`, `fig:rectangle.ps` and `fig:triangle.ps`, respectively.

Example 12 (Labelling within `\CSVtotabular`)

This example is slightly more complicated. The CSV file, `data.csv` looks like:

```
Incubation Temperature,Incubation Time,Time to Growth
40,120,40
40,90,60
35,180,20
```

The following code generates a table using the data with an additional column that generates the experiment number. (See note 8.)

```
\begin{table}
\caption{Time to Growth Experiments}
\label{tab:exp}
\vspace{10pt}
\centering
\CSVtotabular{data.csv}{cccc}{%
 % Header Row
\bfseries Experiment &
\bfseries \begin{tabular}{c}Incubation\\Temperature\end{tabular} &
\bfseries \begin{tabular}{c}Incubation\\Time\end{tabular} &
\bfseries \begin{tabular}{c}Time\\to\\Growth\end{tabular}\\}{%
 % Middle Rows
\label{exp:\insertbyname{Incubation Temperature}:\insertbyname{Incubation Time}}
\thecsvrownumber &
\insertbyname{Incubation Temperature} &
\insertbyname{Incubation Time} &
\insertbyname{Time to Growth} \\}{%
 % Final Row
\label{exp:\insertbyname{Incubation Temperature}:\insertbyname{Incubation Time}}
\thecsvrownumber &
\insertbyname{Incubation Temperature} &
\insertbyname{Incubation Time} &
\insertbyname{Time to Growth}}
\par
\end{table}

It can be seen from Table~\ref{tab:exp}, that
Experiment~\ref{exp:35:180} had the shortest time to growth.
```

In this example, each experiment has the corresponding label `exp:`⟨*Incubation Temperature*⟩`:`⟨*Incubation Time*⟩ so the first experiment has label `exp:40:120`, the second experiment has the label `exp:40:90` and the third experiment has the label `exp:35:180`.

Table 5 shows the resulting table for this example.

The following example is more refined in that it takes advantage of the fact that the time to growth data consists of integers only, so the experiment with the maximum growth can be determined by LaTeX.

Example 13 (Labelling within `\CSVtotabular`)

```
\newcounter{maxgrowth}
\newcounter{incT} % incubation temperature
```

Table 5: Time to Growth Experiments

Experiment	**Incubation Temperature**	**Incubation Time**	**Time to Growth**
1	40	120	40
2	40	90	60
3	35	180	20

```
\newcounter{inct} % incubation time

\begin{table}
\caption{Time to Growth Experiments}
\label{tab:exp}
\vspace{10pt}
\centering
\CSVtotabular{data.csv}{cccc}{%
 % Header row
\bfseries Experiment &
\bfseries \begin{tabular}{c}Incubation\\Temperature\end{tabular} &
\bfseries \begin{tabular}{c}Incubation\\Time\end{tabular} &
\bfseries \begin{tabular}{c}Time\\to\\Growth\end{tabular}\\}{%
 % Middle rows
\label{exp:\insertbyname{Incubation Temperature}:\insertbyname{Incubation Time}}
\thecsvrownumber &
\insertbyname{Incubation Temperature} &
\insertbyname{Incubation Time} &
\insertbyname{Time to Growth}%
\ifthenelse{\value{maxgrowth}<\insertbyname{Time to Growth}}{%
\setcounter{maxgrowth}{\insertbyname{Time to Growth}}%
\setcounter{incT}{\insertbyname{Incubation Temperature}}%
\setcounter{inct}{\insertbyname{Incubation Time}}}{}%
\\}{%
 % Last row
\label{exp:\insertbyname{Incubation Temperature}:\insertbyname{Incubation Time}}
\thecsvrownumber &
\insertbyname{Incubation Temperature} &
\insertbyname{Incubation Time} &
\insertbyname{Time to Growth}%
\ifthenelse{\value{maxgrowth}<\insertbyname{Time to Growth}}{%
\setcounter{maxgrowth}{\insertbyname{Time to Growth}}%
\setcounter{incT}{\insertbyname{Incubation Temperature}}%
\setcounter{inct}{\insertbyname{Incubation Time}}}{}%
}
\par
\end{table}

As can be seen from Table~\ref{tab:exp},
Experiment~\ref{exp:\theincT:\theinct}
had the maximum time to growth, with
incubation time \theinct,
incubation temperature \theincT\ and
```

```
time to growth, \themaxgrowth.
```

---

7 Saving Entries

Entries can be saved using the command:

`\csvSaveEntry` \csvSaveEntry[⟨*counter*⟩]{⟨*identifier*⟩}[⟨*empty text*⟩]

where ⟨*counter*⟩ is a LaTeX counter, by default `csvrownumber`, and ⟨*identifier*⟩ is the header entry. The entry can then be used with the command:

`\csvGetEntry` \csvGetEntry{⟨*counter*⟩}{⟨*identifier*⟩}

The final optional argument ⟨*empty text*⟩ to `\csvSaveEntry` is the text to use if the entry is blank. For example, `\csvSaveEntry{Time}[MISSING DATA]` will print MISSING DATA if the Time field is blank.

The following example illustrates the use of these commands.

Example 14 (Saving Entries)

This example illustrates how you can use one CSV file to access data in other CSV files. This example has several CSV files:

File `index.csv`:

```
File,Temperature,NaCl,pH
exp25a.csv,25,4.7,0.5
exp25b.csv,25,4.8,1.5
exp30a.csv,30,5.12,4.5
```

File `exp25a.csv`:

```
Time,Logcount
0,3.75
23,3.9
45,4.0
```

File `exp25b.csv`:

```
Time,Logcount
0,3.6
60,3.8
120,4.0
```

File `exp30a.csv`:

```
Time,Logcount
0,3.73
23,3.67
60,4.9
```

It is not possible to nest `\CSVtotabular`, `\CSVtolongtable` and `\applyCSVfile`, so if you need to go through `index.csv` and use each file named in there, you can first go through `index.csv` storing the information using `\csvSaveEntry` as follows:

```
\newcounter{maxexperiments}
\applyCSVfile{sample5.csv}{%
\stepcounter{maxexperiments}
\csvSaveEntry{File}
\csvSaveEntry{Temperature}
\csvSaveEntry{NaCl}
\csvSaveEntry{pH}
}
```

The counter `maxexperiments` simply counts the number of entries in `index.csv`. The entries can now be used to generate a table for each file listed in `index.csv` (the `\whiledo` command is defined in the ifthen package):

```
\newcounter{experiment}
\whiledo{\value{experiment}<\value{maxexperiments}}{%
\stepcounter{experiment}
\begin{table}
\caption{Temperature = \protect\csvGetEntry{experiment}{Temperature},
NaCl = \protect\csvGetEntry{experiment}{NaCl},
pH = \protect\csvGetEntry{experiment}{pH}}
\vspace{10pt}
\centering
\CSVtotabular{\csvGetEntry{experiment}{File}}{ll}{%
Time & Log Count\\}{%
\insertTime & \insertLogcount\\}{%
\insertTime & \insertLogcount}

\end{table}
}
```

Note that `\csvGetEntry` needs to be `\protect`ed within the `\caption` command.

This example can be modified if, say, you only want the tables where the temperature is 25:

```
\setcounter{experiment}{0}
\whiledo{\value{experiment}<\value{maxexperiments}}{%
\stepcounter{experiment}
\ifthenelse{\equal{\csvGetEntry{experiment}{Temperature}}{25}}{%
\begin{table}
\caption{Temperature = \protect\csvGetEntry{experiment}{Temperature},
NaCl = \protect\csvGetEntry{experiment}{NaCl},
pH = \protect\csvGetEntry{experiment}{pH}}
\vspace{10pt}
\centering
\CSVtotabular{\csvGetEntry{experiment}{File}}{ll}{%
Time & Log Count\\}{%
\insertTime & \insertLogcount\\}{%
\insertTime & \insertLogcount}\par
\end{table}}{}
}
```

8 Pie Charts (csvpie.sty)

If you want to create a pie chart from data stored in a CSV file, you can use the csvpie package, distributed with the csvtools package. A basic pie chart can be created using the command:

`\csvpiechart` `\csvpiechart[⟨options⟩]{⟨variable⟩}{⟨filename⟩}`

where ⟨*filename*⟩ is the name of the CSV file containing the data, and ⟨*variable*⟩ is the command indicating the entry that contains the value for the given segment. The starred version of `\csvpiechart` should be used if the CSV file has no header row.

`\csvpieinnerlabel` `\csvpieouterlabel` The pie charts have "inner" labels on the segment, and "outer" labels outside the chart. The labels are given by the commands `\csvpieinnerlabel` and `\csvpieouterlabel`. The default definitions are:

```
\newcommand{\csvpieouterlabel}{\field{1}}
\newcommand{\csvpieinnerlabel}{\field{2}\%}
```

This assumes that the second column contains the data, and the first column contains a description, but can be redefined as necessary.

The pie chart display can be modified using the optional argument to `\csvpiechart`. This argument should be a ⟨*key*⟩=⟨*value*⟩ list. The available keys are as follows:

start This should be an integer specifying the starting angle of the first segment. This is 0 by default.

total This should be an integer specifying the sum of all the segment values. This is 100 by default.

radius This should be a length specifying the radius of the pie chart. (Default: 2cm)

inner This should be a fraction specifying the relative distance along the radius to start the inner label. (Default: 0.25)

outer This should be a fraction specifying the relative distance along the radius to start the outer label. (Default: 1.25)

cutaway This should be a comma-separated list of numbers corresponding to the segments that should be cut away from the rest of the pie chart. Since the value may contain commas, the value should always be enclosed in braces. Ranges may also be used. If a range is used, all the segments in the given range are kept together, so, for example, `cutaway={1,2}` will separate the first two segments from the pie chart, and the two segments will also be separated from each other, whereas `cutaway={1-2}` will separate the first two segments from the pie chart, but will keep the two segments together.

offset This should be a fraction specifying the relative distance along the radius to shift the cut away segments. (Default: 0.1)

firstrow This should be the number of the first row containing the actual data. This is equivalent to the optional argument of `\applyCSVfile` or `\applyCSVfile*`.

Note that TEX performs integer arithmetic. Although the CSV file may contain decimal numbers, rounding will occur when constructing the pie charts.

`\csvpiesegmentcol` The colours for the pie chart segments can be set using the command:

`\csvpiesegmentcol{⟨n⟩}{⟨colour⟩}`

where ⟨*n*⟩ is the segment number, and ⟨*colour*⟩ is a defined colour name. For example, if you want the first segment in the pie chart to be yellow, do:

```
\csvpiesegmentcol{1}{yellow}
```

There are 8 predefined segment colours, if your pie chart has more than 8 segments, you will need to specify the remainder.

`\csvpiesegcolname` You can obtain the colour name for a given segment using:

`\csvpiesegcolname{⟨n⟩}`

where ⟨*n*⟩ is the segment number. The `\csvpiechart` command uses `\applyCSVfile`, so the `csvrownumber` counter can be used. This means that you can change the text colour of the outer label to match the segment. For example:

```
\renewcommand{\csvpieouterlabel}{%
\color{\csvpiesegcolname{\value{csvrownumber}}}\field{2}}
```

Note that `\value` must be used since ⟨*n*⟩ has to be a number.

If you want grey pie charts, either use the `monochrome` package option:

```
\usepackage[monochrome]{csvpie}
```

`\colorpiechartfalse` `\colorpiecharttrue` or use the command `\colorpiechartfalse` prior to using `\csvpiechart`. To switch back to colour pie charts, use `\colorpiecharttrue`.

Example 15 (A Pie Chart)

Given a CSV file (called `fruit.csv`) containing:

```
Name,Value
Apples,20
Pears,15
"lemons,limes",30.5
Peaches,24.5
Cherries,10
```

Then the value for each segment is given by the second column, so ⟨*variable*⟩ should be `\field{2}` or `\insertValue`. The pie charts shown in Figure 1 can be created using:

```
 % Change the way the labels are displayed
\renewcommand{\csvpieinnerlabel}{\sffamily\insertValue\%}
\renewcommand{\csvpieoutlabel}{%
\color{\csvpiesegcolname{\value{csvrownumber}}}\sffamily\insertName}

\begin{figure}
\begin{center}
\begin{tabular}{cc}
```

```
\csvpiechart[start=45,cutaway={1,2}]{\insertValue}{fruit.csv} &
\csvpiechart[start=45,cutaway={1-2}]{\insertValue}{fruit.csv} \\
(a) & (b)
\end{tabular}
\end{center}
\caption{Pie Chart Example (a) cutaway=\{1,2\} (b) cutaway=\{1-2\}}
\end{figure}
```

The inner and outer labels have been redefined to use a sans-serif font, and the outer label is in the same colour as its corresponding segment. Both pie charts have a starting angle of 45°, and theyhave the first two segments cutaway, but in (a) the first two segments are separated from each other, whereas in (b), the first two segments are joined, although separated from the rest of the pie chart.

If the CSV file has no header row, the starred version should be used, e.g.:

```
\csvpiechart*[cutaway={1-2}]{\field{2}}{fruit.csv}
```

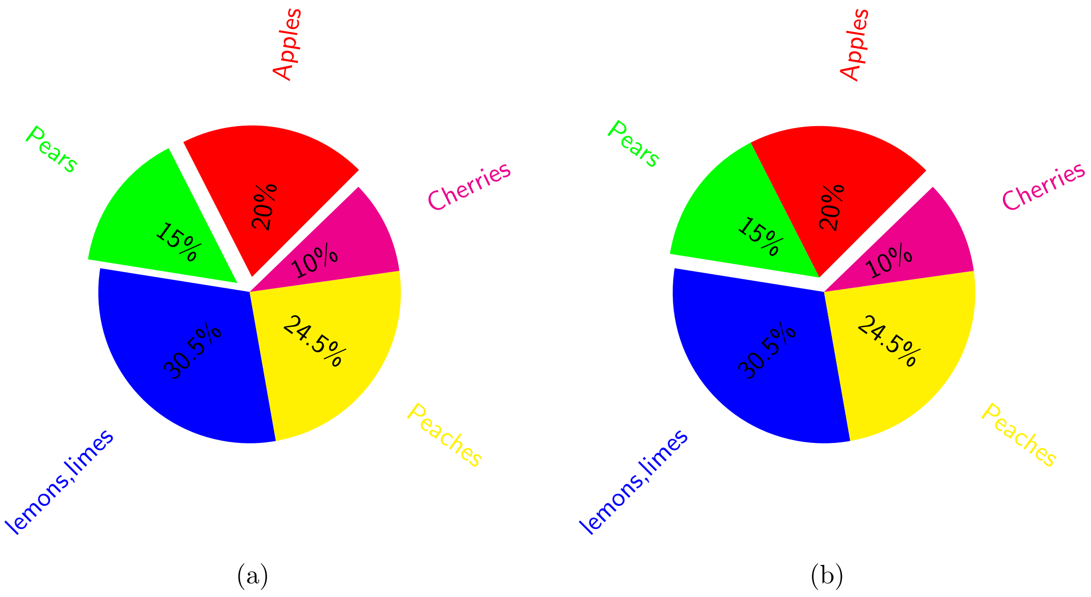

Figure 1: Pie Chart Example (a) cutaway={1,2} (b) cutaway={1-2}

---

9 Sorting Data (csvsort.sty)

The csvsort package (which forms part of the csvtools bundle) provides analogous commands to those provided by csvtools, but the data is first sorted. The csvsort package needs to be loaded separately in order to access the necessary commands. The package options should be a list of key=value pairs, where the available keys are:

verbose Verbose mode. This is a boolean key. If set, the comparisons performed by the insertion sort code are printed to the screen. (Default: `verbose=true`.)

sort This key specifies how to sort the data. It may take one of the following values:

- `alphabetical ascending` (or just `alphabetical`)
- `alphabetical descending`
- `numerical ascending` (or just `numerical`)
- `numerical descending`

(Default: `sort=alphabetical ascending`)

variable The sort variable. (Default: `sort=\field{1}`)

sfirstdataline The line on which the data starts in a data file without a header row. (Default: `sfirstdataline=1`.)

firstdataline The line on which the data starts in a data file with a header row. (Default: `firstdataline=2`.)

Note that the csvsort package requires the xfor package and Éamonn McManus' `compare.tex` file. The csvsort package uses an insertion sort method to sort the data, so large amounts of data may slow processing time. The following commands are provided by csvsort:

`\sortapplyCSVfile`

`\sortapplyCSVfile[`⟨*options*⟩`]{`⟨*filename*⟩`}{`⟨*text*⟩`}`
`\sortapplyCSVfile*[`⟨*options*⟩`]{`⟨*filename*⟩`}{`⟨*text*⟩`}`

These commands are analogous to `\applyCSVfile` and `\applyCSVfile*`, except that the data is first sorted. The optional argument is a key=value list. The keys are the same as those used in the package options, described above. These options only apply to the given instance of the command, whereas the package options apply to all csvsort commands, unless overridden in ⟨*options*⟩. Example, suppose you have a file called `unsorted.csv` which looks like:

```
First Name,Surname,Age
Zephram,Lang,60
Fred,Lang,10
Barney,Langley,25
Jane,Brown,5
Adam,Smith,24
Bert,Jones,18
```

Then

```
\sortapplyCSVfile[sort=alphabetical,variable=\insertSurname]{unsorted.csv}{%
\insertSurname, \insertbyname{First Name}. Age: \insertAge\par}
```

will produce the following output:

```
Brown, Jane. Age:5
Jones, Bart. Age:18
Lang, Zephram. Age:60
Lang, Fred. Age:10
```

Langley, Barney. Age:25

Smith, Adam. Age: 24

Note that the data has only been sorted according to the surname. To sort first by surname, then by first name, you can do something like:

```
\sortapplyCSVfile[sort=alphabetical,
variable={\insertSurname,\insertbyname{First Name}}]{unsorted.csv}{%
\insertSurname, \insertbyname{First Name}. Age: \insertAge\par}
```

As with `\applyCSVfile`, you must use `\field` if you use the starred version:

```
\sortapplyCSVfile*[sort=alphabetical,
variable={\field{2},\field{1}}]{unsorted.csv}{%
\field{2}, \field{1}. Age: \field{3}\par}
```

The commands:

`\sortCSVtotabular` `\sortCSVtotabular[`⟨*options*⟩`]{`⟨*filename*⟩`}{`⟨*col-spec*⟩`}{`⟨*first row*⟩`}{`⟨*all but last row*⟩`}{`⟨*last row*⟩`}`

`\sortCSVtolongtable` `\sortCSVtolongtable[`⟨*options*⟩`]{`⟨*filename*⟩`}{`⟨*col-spec*⟩`}{`⟨*first row*⟩`}{`⟨*all but last row*⟩`}{`⟨*last row*⟩`}`

Are analogous to `\CSVtotabular` and `\CSVtolongtable`, where, again, ⟨*options*⟩ is a list of key=value pairs, the same as `\sortapplyCSVfile`. Using the same example data as above, the following command will sort the data according to age (in numerical order) and place in a tabular environment:

```
\sortCSVtotabular[sort=numerical,variable=\insertAge]{unsorted.csv}{llr}{%
\bfseries Surname & \bfseries First Name & \bfseries Age\\}{%
\insertSurname & \insertbyname{First Name} & \insertAge\\}{%
\insertSurname & \insertbyname{First Name} & \insertAge}
```

Note that the counter `csvlinenum` has no meaning in the commands provided by the csvsort package. The `csvrownumber` counter corresponds to the sorted data row.

10 The csvtools.pl Perl Script

Suppose you have several large CSV files, and you have included the information into your document using `\applyCSVfile`, `\CSVtolongtable`, `\CSVtotabular` or `\csvpiechart`, which has made life so much easier for you, but you are now required by a journal to submit your source code in a single `.tex` file. They don't want all your CSV files, so what do you do? If you have Perl installed on your system you can use the `csvtools.pl` Perl script. This has the following syntax:

`csvtools.pl` ⟨*in-file*⟩ ⟨*out-file*⟩

where ⟨*in-file*⟩ is the name of your file that contains the `\applyCSVfile`, `\CSVtotabular` etc commands, and ⟨*out-file*⟩ is a new file which will be created by `csvtools.pl`. This new file will be the same as ⟨*in-file*⟩ except that all occurances of `\applyCSVfile`, `\CSVtolongtable`, `\CSVtotabular` and `\csvpiechart` will be replaced by the relevant data extracted from the named CSV files.

Example 16 (csvtools.pl — Aligning Data)

Suppose the file `mydoc.tex` contains the code given in Example 4, with the associated CSV file `sample.csv` also given in that example. Then if you do:

```
csvtools.pl mydoc.tex mydocnew.tex
```

the file `mydocnew.tex` will be created which will be identical to `mydoc.tex` except the lines containing the code `\CSVtotabular{sample.csv}{lccc}{...}{...}{...}` will be replaced with the lines:

```
% \CSVtotabular{sample.csv}... converted using csvtools.pl
%>> START INSERT
\begin{tabular}{lccc}
\bfseries Name &
\bfseries Assignment 1 &
\bfseries Assignment 2 &
\bfseries Total\\
A. Smith&80&70&150\\
B. Jones&60&80&140\\
J. Doe&85&75&160\\
 &75&75&150
\end{tabular}%<< END INSERT
```

Similarly, `csvtools.pl` will substitute all occurrances of `\CSVtolongtable`, `\applyCSVfile` and `\csvpiechart`.

10.1 Notes

1. If perl is located in a directory other than `/usr/bin/` you will need to edit the first line of `csvtools.pl` as appropriate. You can find the location using the command:

   ```
   which perl
   ```

2. If you can't directly execute a Perl script, you can do:

   `perl csvtools.pl` ⟨*in-file*⟩ ⟨*out-file*⟩

3. You must first LaTeX your document before using `csvtools.pl` as it checks the log file for any counters that have been defined.

4. `csvtools.pl` only knows about a very limited set of LaTeX commands. It should be able to understand:

   ```
   \CSVtotabular{\csvGetEntry{experiment}{File}}{ll}{...
   ```

   (see Example 14), but it won't be able to understand, say,

   ```
   \newcommand{\filename}{\csvGetEntry{experiment}{File}}
   \CSVtotabular{\filename}{ll}{...
   ```

   It can pick up on `\addtocounter`, `\stepcounter`, `\refstepcounter` and `\setcounter` but only if they are used explicitly in the named `.tex` file. (It ignores any files that have been included using `\input`, `\include` etc.)

5. This Perl script has only been tested under Linux, but it ought to work under other systems.

11 Bugs/Drawbacks/"Features"

1. The package doesn't check to see whether `\insert`⟨*identifier*⟩ exists, otherwise you would not be able to use multiple CSV files with the same headers, as in Example 14. Therefore it is recommended that you check to make sure that the command does not already exist. For example, the TEX commands `\insert` and `\insertpenalties` already exist, so a blank header or a header named `penalties` would cause problems. (These two will now cause an error as from version 1.1, but it's something bear in mind.)

2. Note also that `\insertbyname` doesn't check if you've given a valid label, so if no text appears, check you've spelt it correctly, checking punctuation, spaces and case.

3. Note that in Example 2, replacing line 3 with:

   ```
   \centerline{\includegraphics{\insertbyname{File}}}
   ```

   will cause an error, as `\insertbyname{File}` doesn't get fully expanded by the time it gets passed to `\includegraphics`, and will prevent `\includegraphics` from finding the file. It is possible to get around this using TEX's `\edef` command:

   ```
   \edef\psfilename{\insertbyname{File}}
   \centerline{\includegraphics{\psfilename}}
   ```

4. You can't have commands like `\hline`, `\cline` and `\multicolumn` in the first column of the ⟨*middle*⟩ or ⟨*last*⟩ code of `\CSVtotabular` or `\CSVtolongtable`. If you do, it will generate a `misplaced \noalign` error, instead you need to put it at the end of the ⟨*first*⟩ or ⟨*middle*⟩ code. (See Example 6.)

5. You can't have nested `\applyCSVfile`, `\CSVtolongtable` and `\CSVtotabular` commands. Nor can you have `\csvpiechart` within one of these commands (See Example 14)

6. If the CSV file has a header row, it must be on the first line.

7. It is possible for TEX to run out of memory if you use `\csvSaveEntry` on a large file.

8. In version 1.0, there was an inconsistency with `csvrownumber` within `\applyCSVfile` and `\CSVtotabular`. In the former it excluded the header row, whereas the latter included it. This has been changed in version 1.1 so that within `\applyCSVfile`, `\CSVtotabular` and `\CSVtolongtable`, `csvrownumber` refers to the data row (excluding header row.) I hope this doesn't cause problems, but it makes more sense that they should be consistent. So if you have no blank lines in your CSV file, `csvrownumber` should always be 1 more than `csvlinenumber`.

12 Contact Details

Dr Nicola Talbot
School of Computing Sciences
University of East Anglia
Norwich. NR4 7TJ. England.

http://theoval.cmp.uea.ac.uk/~nlct/

Index

A

\applyCSVfile 3

C

\colorpiechartfalse 15
\colorpiecharttrue . 15
\csvGetEntry 12
\csvlinenum 8
\csvpiechart 14
\csvpieinnerlabel . 14
\csvpieouterlabel . 14
\csvpiesegcolname . 15
\csvpiesegmentcol . 15
\csvrownumber 8
\csvSaveEntry 12
\CSVtolongtable 7
\CSVtotabular 4

F

\field 3

I

\ifnextrowlast 5
\insert... 3
\insertbyname 3

S

\setcsvseparator . . . 2
\sortapplyCSVfile . 17
\sortCSVtolongtable 18
\sortCSVtotabular . 18