The \texttt{fcolumn} package\footnote{This file has version number v1.3, last revised 2021/01/14.}

Edgar Olthof
\texttt{edgar <dot> olthof <at> inter <dot> nl <dot> net}

Printed January 14, 2021

\begin{abstract}
In financial reports, text and currency amounts are regularly put in one table, e.g., a year balance or a profit-and-loss overview. This package provides the settings for automatically typesetting and checking such columns, including the sum line (preceded by a rule of the correct width), using the specifier \texttt{f}.
\end{abstract}

1 \textbf{Introduction}

The package \texttt{fcolumn} provides the macros for an extra tabular specifier that makes creating financial tables easy. The column specifier \texttt{f} itself is rather simple; it is the predefined version of a generic column \texttt{F}. The generic version expects four arguments: 1) grouping character of the integer part on output, 2) decimal mark used on output, 3) compact additional information on input/output characteristics, and 4) anything, but primarily used for providing formatting information, see below.

The \texttt{f}-column in the current version of the package is defined for the continental European standard: 
\begin{verbatim}
\newcolumntype{f}{F{.}{,}{3,2}{}}
\end{verbatim} 
This means that a number like 12345,67 will be typeset as 12.345,67. People in the Anglo-saxon world would rather code 
\begin{verbatim}
\newcolumntype{f}{F{,}{.}{3,2}{}}
\end{verbatim} 
for the same input, yielding 12,345.67 as output for the number given above. The default value for \texttt{#3} is \texttt{3,2}, indicating that grouping of the integer part is by three digits, that a comma is used in the TeX-source to indicate the decimal separator, and that the decimal part consists of two digits. If however, in your country or company grouping is done with a thinspace every four digits, that the separator in the source should be the character \texttt{p}, and there are three digits after the decimal mark—that happens to be a \texttt{\cdot}, then simply specify 
\begin{verbatim}
\newcolumntype{f}{F{\,}{\cdot}{4p3}{}}
\end{verbatim} 
in that case. The input could be 12345p678 then, yielding 12 345·678 as output.

By default two digits are used for the decimal part, so if you really want no decimal digits (in that case of course also skipping the decimal mark) you have to explicitly specify \texttt{x,0}. If you want no grouping character, specify \texttt{0,x}. 
As the fourth parameter you can insert anything just before the typesetting of an amount in a column takes place. Its purpose is to add additional formatting information, e.g., \texttt{\textcolor{red}} to have the contents of a column coloured red, but it can be misused, so use with care. And it can’t do all!

This package requires and loads the \texttt{array} package [1]. To show where and how the \texttt{F}-column is used, let’s look at some typical financial information as shown in Table 1 and how this is entered in \LaTeX{} (Table 2). All the work was done by

**Table 1**: Example Table.

<table>
<thead>
<tr>
<th>properties</th>
<th>31 dec 2014</th>
<th>debts</th>
<th>31 dec 2014</th>
</tr>
</thead>
<tbody>
<tr>
<td>house</td>
<td>200,000.00</td>
<td>equity capital</td>
<td>50,000.00</td>
</tr>
<tr>
<td>bank account</td>
<td>-603.23</td>
<td>mortgage</td>
<td>150,000.00</td>
</tr>
<tr>
<td>savings</td>
<td>28,000.00</td>
<td>cash</td>
<td>145,85</td>
</tr>
<tr>
<td>cash</td>
<td>227,542.62</td>
<td>profit</td>
<td>27,542.62</td>
</tr>
</tbody>
</table>

**Table 2**: Verbatim version of Example Table 1.

\begin{table}[htb]
\caption{Example Table.}
\label{tab:ex1}
\begin{tabular}{@{}lflf@{}}
\multicolumn{4c}{\bfseries Balance sheet}\%
\hline
properties & \texttt{\leeg{31 dec 2014}} & debts & \texttt{\leeg{31 dec 2014}}\%
\hline
house & 200,000.00 & equity capital & 50,000.00 \%
bank account & -603.23 & mortgage & 150,000.00 \%
savings & 28,000.00 & cash & 145,85 \%
\texttt{\sumline}
cash & 227,542.62 & profit & 27,542.62 \%
\texttt{\bottomrule}
\end{tabular}
\end{table}

the column specifier “f” (for “finance”). In this case it constructs the \texttt{\sumline}, typesets the numbers, calculates the totals, determines the widths of the sumrules, and checks whether the two columns are in balance; if not, the user is warned via a \texttt{\PackageWarning}. Of course for nice settings the \texttt{booktabs} package [2] was used, but that is not the point here.

This package is heavily inspired by the \texttt{dcolumn} package by David Carlisle [3], some constructions are more or less copied from that package. Version 1.3 (this version) incorporates the idea of Christian Hoff of providing additional (formatting) information per column. A rather contrived example is given in Table 3,
combining colour and fonts. How this is entered in \LaTeX is shown in Table 4.

Table 3: Example Table with column formatting.

<table>
<thead>
<tr>
<th>properties</th>
<th>31 dec 2014</th>
<th>debts</th>
<th>31 dec 2014</th>
</tr>
</thead>
<tbody>
<tr>
<td>house</td>
<td>200,000.00</td>
<td>equity capital</td>
<td>50,000.00</td>
</tr>
<tr>
<td>bank account</td>
<td>−603.23</td>
<td>mortgage</td>
<td>150,000.00</td>
</tr>
<tr>
<td>savings</td>
<td>28,000.00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>cash</td>
<td>145.85</td>
<td>profit</td>
<td>27,542.62</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>227,542.62</td>
</tr>
</tbody>
</table>

The font changing commands like \texttt{\mathsf} and \texttt{\mathbf} act on an argument, hence

Table 4: Almost verbatim version of Example Table 3.

\newcolumntype{q}[1]{F{.}{,}{3,2}{#1}}
\begin{table}[htb]
\caption{Example Table with column formatting.}
\label{tab:ex3}
\begin{tabular}{@{}lq{\color{red}\mathsf,{}\mathbf}lq{\color{green}}@{}}
\multicolumn{4}{c}{\bfseries Balance sheet}\n...
...(same financial contents as in Table 1)
...
\end{tabular}
\end{table}

require braces, but these are already provided internally for this purpose. For that reason this type of commands must be given last, without braces (and if you don’t specify a font changing command, these extra internal braces are just redundant). The argument to the new colume type may consist of two parts, separated by a comma. In that case, the part to the left of the comma is applied to the whole column and the right part is applied in addition to the result. The example in Table 3 shows this: the bold font is only used in the \texttt{\textbackslash sumline} and the effect of \texttt{\mathsf} is annullled by the brace pair before \texttt{\textbackslash mathbf}. You may even change the colour for the \texttt{\textbackslash sumline}, e.g., blue by \texttt{\textbackslash color\{blue\}\textbackslash mathbf}, since colour settings do not require braces and the last setting overrides a previous one. If you want bold font for the whole column, leave out the comma (and the \texttt{\textbackslash mathsf{}}).

Note that changes in font size, e.g., \texttt{\textbackslash huge} in \texttt{\textbackslash huge\textbackslash color\{red\}\textbackslash mathbf} as parameter to column type \texttt{q} are ignored by \LaTeX, since the formatting information is used in math environment, which has its own way of handling this. This isn’t bad, as size changes in one column, without overall changes to the table look terrible. If you want something huge, make a \texttt{\textbackslash huge} table.

This package now also works with \texttt{longtable} [4], provided \texttt{longtable} is loaded before \texttt{fcolumn}; it checks for that. For a change, the raw formatting of the mul-
The tipage table is shown first, in Table 5. For more information on how to handle `\endhead` and related commands, see the documentation of `longtable` [4].

**Table 5:** Almost verbatim version of Example Table 6.

```latex
\begin{longtable}{l l l l}
\caption{\label{tab:ex6}Table showing compatibility of fcolumn and longtable.}\%
\multicolumn{4}{c}{\bfseries Balance sheet}\%
\toprule
properties & leeg{31 dec 2014} & debts & leeg{31 dec 2014}\%
\midrule
\endfirsthead
\caption[\textit{(continued from previous page)}]{\textit{(continued from previous page)}}\%
\midrule
debts & leeg{31 dec 2014} & properties & leeg{31 dec 2014} & debts & leeg{31 dec 2014}\%
\midrule
\endhead
\caption[\textit{(Table continues on next page)}]{\textit{(Table continues on next page)}}\%
\midrule
\multicolumn{4}{r@{}}{\textit{(Table continues on next page)}}\%
\endfoot
\bottomrule
\endlastfoot
\end{longtable}
```

...(somewhat altered financial contents as in Table 1)

And here is the result (it’s ugly, but it had to be that long to demonstrate the page break). Not shown here are the new `fcolumn` formatting possibilities (like new fonts and/or colours), but it has been checked they do work in combination with `longtable`.

**Table 6:** Table showing compatibility of fcolumn and longtable.

<table>
<thead>
<tr>
<th>properties</th>
<th>31 dec 2014</th>
<th>debts</th>
<th>31 dec 2014</th>
</tr>
</thead>
<tbody>
<tr>
<td>house</td>
<td>200,000.00</td>
<td>equity capital</td>
<td>50,000.00</td>
</tr>
<tr>
<td>bank account</td>
<td>−603.23</td>
<td>mortgage</td>
<td>150,000.00</td>
</tr>
<tr>
<td>savings 1</td>
<td>2,800.00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>savings 2</td>
<td>2,800.00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>savings 3</td>
<td>2,800.00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>savings 4</td>
<td>2,800.00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>savings 5</td>
<td>2,800.00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>savings 6</td>
<td>2,800.00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>savings 7</td>
<td>2,800.00</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(Table continues on next page)
Table 6: (continued from previous page)

<table>
<thead>
<tr>
<th>properties</th>
<th>31 dec 2014</th>
<th>debts</th>
<th>31 dec 2014</th>
</tr>
</thead>
<tbody>
<tr>
<td>savings 8</td>
<td>2,800.00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>savings 9</td>
<td>2,800.00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>savings 10</td>
<td>2,800.00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>cash</td>
<td>145.85</td>
<td>profit</td>
<td>27,542.62</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>227,542.62</td>
</tr>
</tbody>
</table>

2 Commands

The user only needs to know six commands or constructions. These six are given here.

F

In the tabular the column specifier F can be given with arguments, or the predefined version f, where the four arguments of F are \{.\}, \{,\}, \{3,2\}, and \{\}. If you want g to be your own definition like the curious one given in Section 1, then specify \texttt{\newcolumntype{g}{F{.}{,}{3,2}{}}} prior to using g in a tabular.

Entries in an F-column are, from that moment on, treated as numbers unless explicitly escaped by \leeg, see below. The numbers are typeset according to the template the user gives with his/her F-column. The “middle” character of #3 is an important switch: it does more than just setting the input decimal mark. By default the input grouping character is the dot, except when the dot is specified as input decimal mark; in that case the comma is acting as input grouping character. With this convention the continental Europe and Anglo-saxon part of the world is served. And using input grouping markers is optional.

\sumline

The numbers in an F-column are typeset as a financial amount, but the real benefit comes with the \texttt{\sumline}. It does three things:

1) It calculates the total of the column so far and the maximum width encountered so far, including the width of the total;
2) It generates a rule with width calculated in the first item;
3) It checks the columns that are supposed to balance whether or not they actually do. If so, nothing happens. If not, a \texttt{PackageWarning} is given that column i and j do not balance, where i and j are the relevant columns. This is only done if the total number of F-columns is even, e.g., if there are six F-columns, then 1 is checked against 4, 2 against 5, and 3 against 6. If the number of F-columns is odd then anything could be possible in that table and nothing is assumed about structure within the table. This behaviour can be overridden, see below.

By default the vertical separation between the rule and the total is 2pt, but this can be changed by the optional argument to \texttt{\sumline}. Give, e.g., \texttt{\sumline[10pt]}, in case you want this spacing to be 10pt. And you may even give two options, like in \texttt{\sumline[10pt][5pt]}, in which the second option is the
extra space below the summary row. In fact that second option is the option to \ that is implicit in \sumline.

\resetsumline Suppose you want to typeset one tabular with the profit-and-loss of many projects individually. The layout of those tabulars is the same and it were nice if all columns were aligned. This can be done by making it one big tabular with a fresh start for each project. The macro \resetsumline is used for that: it resets all totals and all column widths, see for example Table 7. Note that the rules in the first and

<table>
<thead>
<tr>
<th>Table 7: Example: multiple projects.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Project 1</strong></td>
</tr>
<tr>
<td>expense</td>
</tr>
<tr>
<td>food</td>
</tr>
<tr>
<td>drinks</td>
</tr>
<tr>
<td>music</td>
</tr>
<tr>
<td>profit</td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Project 2</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>expense</td>
</tr>
<tr>
<td>food</td>
</tr>
<tr>
<td>drinks</td>
</tr>
<tr>
<td>music</td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

third F-columns of project 1 cover 1,200,00 whereas in project 2 those rules are narrower since they only cover 430,00; still the columns are aligned. The verbatim way of setting up Table 7 is given in Table 6.

\leeg If an F-column should be empty then simply leave it empty. If however it should not be empty but the entry should be treated as text—even if it is a number—, this can be done with \leeg. It expects an argument and this argument is typeset in the column. The common case is where p.m. (pro memoria) is entered. In contrast to v1.1.2 of this package, now even an empty F-column followed by \ is allowed.

\checkfcolumns The automatic column balance check can also be done manually. If F-columns 1 and 4 should balance and you want them to be checked, then simply say \checkfcolumns{1}{4}. With more than nine F-columns you may be forced to say something like \checkfcolumns{10}{12}. If \checkfcolumns is used, the automatic check is disabled. Multiple \checkfcolumns are supported; if F-columns 1, 2, and 3 should balance, you specify \checkfcolumns{1}{2}{3} and \checkfcolumns{12}{23}. There is no explicit command to disable all checking, but \checkfcolumns{1}{1} obviously also serves that purpose.

\ifstrict@accounting In the rare occasion that a negative number occurs in a financial table, the sign of that number can be an explicit minus sign (–) or the number is coloured red,
or it is typeset between parentheses, and there may be even other ways. By default (for aesthetic reasons) \texttt{fcolumn} typesets it with a minus sign, but strict accounting prescribes that the number should be put between parentheses. The latter can be accomplished by setting \texttt{\textbackslash strict\textbackslash(accountingtrue}. But since this contains a non-letter, it is also possible to invoke \texttt{fcolumn} with the option \texttt{strict}, i.e., \texttt{\usepackage[strict]{fcolumn}}, which sets this flag.

3 The macros

Here follows the actual code.

3.1 Option

\texttt{option strict} There is one option. If set, strict accounting rules are used in display.

\begin{verbatim}
1 \newif\ifstrict\textbackslashaccounting \strict\textbackslash(accountingfalse
2 \DeclareOption{strict}{\strict\textbackslash(accountingtrue}
\end{verbatim}
3 \ProcessOptions

## 3.2 Definitions

column F

The column specifier \texttt{F} is the generic one, and \texttt{f} is the default (continental European) one for easy use. Note that the definition of the column type \texttt{f} does not use private macros (no @), so overriding its definition is easy for a user.

\begin{verbatim}
4 \newcolumntype{F}[4]{>{\b@fi{#1}{#2}{#3}{#4}}r<{\e@fi}}
5 \newcolumntype{f}{F{.}{,}{3,2}{}}
\end{verbatim}

\texttt{\FCsc@l} and \texttt{\FCtc@l}

Two \langle count \rangle s are defined, that both start at zero: the \langle count \rangle \texttt{\FCsc@l}, that keeps track at which F-column the tabular is working on and the \langle count \rangle \texttt{\FCtc@l}, that records the number of F-columns that were encountered so far. Later in the package the code can be found for generating a new \langle count \rangle and a new \langle dimen \rangle if the number of requested F-columns is larger than currently available. This is of course the case when an F-column is used for the first time.

\begin{verbatim}
6 \newcount\FCsc@l \FCsc@l=0 \newcount\FCtc@l \FCtc@l=0
\end{verbatim}

\texttt{\geldm@cro}

The macro \texttt{\geldm@cro} takes a number and by default interprets this as an amount expressed in cents (dollar cents, euro cents, centen, pfennige, centimes, kopecks, groszy) and typesets it as the amount in entire currency units (dollars, euros, guldens, marke, francs, rubles, zloty) with comma as decimal separator and the dot as grouping character (thousand separator if the first part of \texttt{#1} is 3). As explained, this can be changed. It uses two private booleans: \texttt{\withs@p} and \texttt{\strict@ccounting}. The latter is used to typeset negative numbers between parentheses. By default it doesn’t do this: a minus sign is used.

\begin{verbatim}
7 \newif\ifwiths@p
Actually \texttt{\geldm@cro} is only a wrapper around \texttt{\g@ldm@cro}.
8 \def\geldm@cro#1#2{\withs@pfalse\afterassignment\g@ldm@cro\count@#1\relax{#2}}
\end{verbatim}

\texttt{\g@ldm@cro}

After setting the environment for formatting, this macro starts by looking at the sign of \texttt{#2}: if it is negative, it prints the correct indicator (a parenthesis or a minus sign), assigns the absolute value of \texttt{#2} to \texttt{\count2} and goes on. Note that \texttt{\geldm@cro} and therefore \texttt{\g@ldm@cro} are always used within $\$, so it is really a minus sign that is printed, not a hyphen. All calculations are done with \texttt{\count0}, \texttt{\count1}, etc. i.e., without F-column-specific \langle count \rangle s because it is all done locally. Leaving the tabular environment will restore their values. This is also true for the effect of \texttt{\FCform@t}, so that formatting information is local to this column. The reason for inserting the opening brace between \texttt{\FCform@t} and \texttt{\ifnum} (and the accompanying closing brace after \texttt{\fi}) is to facilitate the possible use of \texttt{\mathbf} or any other font changing command as the last item in \texttt{\FCform@t}.

\begin{verbatim}
10 \def\g@ldm@cro#1\relax#2{\FCform@t{\ifnum#2<0 \ifstrict@ccounting (\else
11 \fi \count2=-#2 \else\count2=#2
12 \fi
\end{verbatim}
Calculate the entire currency units: this is the result of $x/a$ as integer division, with $a = 10^n$ and $n$ the part of #1 after the separator (if any). Here the first character of #1 is discarded, so the separator in #1 is not strict: you could also specify 3.2 instead of 3,2 (or even 3p2).

\begin{verbatim}
\count4=\ifx\relax#1\relax 2 \else \@gobble#1\relax\fi
\count3=0
\loop\ifnum\count3<\count4\divide\count2 by 10 \advance\count3 by \@ne\repeat

Note that \count3 now equals \count4: this going up-and-down will be used more often, it saves several assignments. The value in \count2 is then output by \g@ldens using the separation given (and stored in \count@).
\g@ldens{\the\count@}%
\end{verbatim}

If there is a decimal part...

\begin{verbatim}
\ifnum\count3>0\decim@lmark
\loop\ifnum\count3>0
\divide\count2 by 10 \advance\count3 by \m@ne\repeat
\ifnum#2>0 \count2=-#2\else\count2=#2 \fi
\loop\ifnum\count3>0
\divide\count2 by 10 \advance\count3 by \@ne\repeat
\ifnum#2>0 \advance\count2 by #2 \else \advance\count2 by -#2 \fi
\zerop@d{\number\count3}{\number\count2}%
\fi
\end{verbatim}

If the negative number is indicated by putting it between parentheses, then the closing parenthesis should stick out of the column, otherwise the alignment of this entry in the column is wrong. This is done by an \rlap and therefore does not influence the column width. For the last column this means that this parenthesis may even stick out of the table. I don’t like this, therefore I chose to put \strict@ccountingfalse. Change if you like, by setting the option strict.

If overflow was detected, an exclamation mark is output to the right of the value that caused this. This of course ruins the appearance of the table, but in this case that serves a clear goal: there’s something wrong and you should know.

\begin{verbatim}
\ifx\FCs@gn\m@ne \ifnum#2<0 \ifstrict@ccounting
\rlap{)}~!\else\rlap{~!}\fi\else\ifstrict@ccounting
\rlap{)}\phantom{~!}\else\rlap{~!}\fi\fi
\else \ifnum#2<0 \ifstrict@ccounting\rlap{)}\fi\else\rlap{~!}\fi\fi
\end{verbatim}
Here the whole currency units are dealt with. The macro \texttt{\gldens} is used recursively, therefore the double braces; this allows to use \texttt{\count0} locally. This also implies that tail recursion is not possible here, but that is not very important, as the largest number (which is $2^{31} - 1$) will only cause a threefold recursion using the default 3,2 (ninefold when using 1,0, but who does that?). The largest amount this package can deal with is therefore 2,147,483,647 (using 3,0). For most people this is probably more than enough if the currency is euros or dollars. And otherwise make clear that you use a currency unit of k\text{€} (or even M\text{€} for the very rich). The author is thinking of ways to use two counters for each number. The maximum then becomes $2^{63} - 1$. Even expressed in cents this would lead to a maximum of slightly more than 92.2 P\text{€}; about 100 times the current world economy [5]. Yet another method is to use Heiko Oberdiek’s package \texttt{bigintcalc}: then only memory restrictions apply. This, however, requires a major rewrite of \texttt{fcolumn}. For now, version 1.3 sticks to the moderate amounts.

There is no straightforward interpretation of \#1 being zero or negative, therefore this is used as an indicator that no grouping character should be used.

\begin{verbatim}
def\gldens#1{{\count3=\count2 \count0=#1
First divide by 10\textsuperscript{n}, where n is \#1.
\ifnum\count0<1 \count0=3 \fi
\loop \ifnum\count0>0 \divide\count2 by 10 \advance\count0 by \m@ne
Here is the recursive part,
\ifnum\count2>0 \gldens{#1}\fi
and then reconstruct the rest of the number.
\count0=#1
\ifnum\count0<1 \count0=3 \fi
\loop \ifnum\count0>0 \multiply\count2 by 10 \advance\count0 by \m@ne
\repeat
\count2=-\count2
\advance\count2 by \count3 \duzendprint{#1}}}
\end{verbatim}

The macro \texttt{\duzendprint} takes care for correctly printing the separator and possible trailing zeros. The former, however, is only done if \#1 is larger than zero.

\begin{verbatim}
def\duzendprint#1{{\ifwiths@p\ifnum#1>0 \sep@rator\fi
\zerop@d{#1}{\number\count2}\
\else\zerop@d1{\number\count2}\fi\global\withs@ptrue}
\end{verbatim}

The macro \texttt{\zerop@d} uses at least \#1 digits for printing the number \#2, padding with zeros when necessary. Note: \#1 being zero or negative is a flag that it should be interpreted as 3. A bit ugly, but it works, since the related code knows about this.

It is done within an extra pair of braces, so that \texttt{\count0} and \texttt{\count1} can be used without disturbing their values in other macros.

\begin{verbatim}
def\zerop@d#1#2{{\count0=1 \count1=#2
\end{verbatim}
First determine the number of digits of \#2 (expressed in the decimal system). This number is in \count0 and is at least 1.

\loop
\divide \count1 by 10
\ifnum\count1>0
\advance\count0 by \@ne
\repeat

If \#1 is positive, the number of zeros to be padded is max(0, \#1 - \count0) (the second argument can be negative), so a simple loop suffices. If it is zero or negative, this is a signal that it should be interpreted as 3 (and no separator will be output).

\ifnum\#1>0
\loop
\ifnum\count0<\#1 \relax
0
\advance\count0 by \@ne
\repeat
\fi
\else
\advance\count0 by -3
\loop
\ifnum\count0<0
0
\advance\count0 by \@ne
\repeat
\fi
\fi

\zetg@ld
This macro takes care for several things: it increases the subtotal for a given F-column, it checks whether or not that subtotal has overflown, it records the largest width of the entries in that column and it typesets \#1 via \geldm@cro.

\def\zetg@ld#1#2{\count0=#2 \relax \let\FCs@gn=\@ne
First it checks whether there is a risk of overflow in this step. If \textit{A} and \textit{B} are two \TeX-registers and \textit{B} is to be added to \textit{A}, overflow will not occur if one is (or both are) zero or if \textit{A} and \textit{B} have different signs. Otherwise, be careful. Note that \TeX does not check for overflow when performing an \texttt{\advance} (done in section 1238 of Ref. \cite{6}), in contrast to \texttt{\multiply}, see section 105.

\ifnum\count0<0
\ifnum\csname FCtot\romannumeral\FCsc@l\endcsname<0
\let\FCs@gn=\m@ne
\fi
\fi
\ifnum\count0>0
\ifnum\csname FCtot\romannumeral\FCsc@l\endcsname>0
\let\FCs@gn=\m@ne
\fi
\fi
\global\advance\csname FCtot\romannumeral\FCsc@l\endcsname by \count0
\ifx\FCs@gn\m@ne
They had the same sign: risk of overflow. Record the sign of \count0 (and of the original total of this column; they were the same) in \texttt{\FCs@gn}. Table 9 shows what can go wrong if the numbers are too large: in the left F-column the sumline is incorrect and the number that caused the overflow is indicated by an exclamation mark. In the middle F-column, overflow occurs twice and because this is once positive, once negative here, cancellation of errors occurs and the sumline is correct in the end. Nevertheless, it is advised to swap the two items that caused the overflow, as shown in the right F-column.
Table 9: Examples on overflow.

<table>
<thead>
<tr>
<th>Projects</th>
<th>income</th>
<th>31 dec 2014</th>
<th>31 dec 2015</th>
<th>31 dec 2016</th>
</tr>
</thead>
<tbody>
<tr>
<td>item 1</td>
<td>20.000.000,00</td>
<td>20.000.000,00</td>
<td>20.000.000,00</td>
<td></td>
</tr>
<tr>
<td>item 2</td>
<td>10.000.000,00</td>
<td>2.000.000,00</td>
<td>-2.000.000,00</td>
<td></td>
</tr>
<tr>
<td>item 3</td>
<td>5.000.000,00</td>
<td>-2.000.000,00</td>
<td>2.000.000,00</td>
<td></td>
</tr>
<tr>
<td></td>
<td>-7,949.672,962</td>
<td>20.000.000,00</td>
<td>20.000.000,00</td>
<td></td>
</tr>
</tbody>
</table>

Since the absolute value of \texttt{\textbackslash FC\textbackslash sign} is unity, no overflow will occur in the multiplication step below.

\begin{verbatim}
\ifnum\count0>0 \let\FC\textbackslash sign\@ne \fi
\count0=\csname FC\textbackslash total@\romannumeral\FC\textbackslash scale\endcsname
\multiply\count0 by \FC\textbackslash sign
\ifnum\count0<0 \let\FC\textbackslash sign=\m@ne \fi
\PackageError{fcolumn}{Register overflow}{Overflow occurred \number\FC\textbackslash scale. You can press <enter> now and I’ll proceed,\MessageBreak but check your table. The offending entry is indicated with an exclamation\MessageBreak mark in the output.}%
\else\let\FC\textbackslash sign=\@ne \fi
\fi
\end{verbatim}

The value of \texttt{\textbackslash FC\textbackslash sign} is used in \texttt{\textbackslash geld\textbackslash m@cro} below.

\begin{verbatim}
\setbox0=\hbox{$\textbackslash geld\textbackslash m@cro{#1}{#2}$}\
\ifdim\wd0>\csname FC\textbackslash width@\romannumeral\FC\textbackslash scale\endcsname\global\csname FC\textbackslash width@\romannumeral\FC\textbackslash scale\endcsname=\wd0\fi\unhbox0
\end{verbatim}

The \texttt{\textbackslash count}s \texttt{\textbackslash FC\textbackslash scale} captures the part to the left of the decimal mark, \texttt{\textbackslash FC\textbackslash r} that to the right.

\begin{verbatim}
\newcount\FC\textbackslash scale \newcount\FC\textbackslash r
Some auxiliary definitions for capturing compacted information.
\def\setucc@de#1#2\relax{\uccode'~='#1 }
\def\assignform@t#1,#2,#3\end{\def\FC\textbackslash form@t{#1}\def\FC\textbackslash form@t{tt{#1#2}}}
\texttt{\textbackslash b@fi} The macro \texttt{\textbackslash b@fi} provides the beginning of the financial column. It will be inserted in the column to capture the number entered by the user. The separator and decimal mark are within a math environment, so you can indeed specify \texttt{,} instead of \texttt{\thinspace}, but there is an extra brace around, so it doesn’t affect the spacing between the digits (trick copied from \texttt{dcolumn}, Ref. [3]).
\end{verbatim}

\begin{verbatim}
\def\b@fi#1#2#3\end{\def\sep@rator{{#1}}}%
\end{verbatim}

An intermediate macro \texttt{\textbackslash sep@xt} to extract the first character of \texttt{#1}, which in most cases will be the only character.

\begin{verbatim}
\def\sep@xt#1#2#3#4\end{\def\sep@rator{{#1}}}%
\end{verbatim}
The value specified by the user is then captured by \texttt{FC@l} and this is done in a special way: \texttt{FC@l} is assigned globally within \texttt{box0}. Why? To use it as scribbling paper to examine what the user entered, without dumping it into the horizontal list.

There are four parts to an F-column entry, all parts optional, making 16 combinations. The sequence is (in the Backus–Naur notation of Ref. [7]): \langle \text{sign} \rangle \langle \text{integer constant} \rangle \langle \text{decimal mark} \rangle \langle \text{integer constant} \rangle. Here \langle \text{sign} \rangle is a plus or minus character with category code 12, \langle \text{integer constant} \rangle is a sequence of zero or more (decimal) \langle \text{digit} \rangle s, and \langle \text{decimal mark} \rangle is the middle part of \#3, i.e., the comma in 3,2 or the period in 3.2. If the \langle \text{decimal mark} \rangle is absent with no space characters between the two \langle \text{integer constant} \rangle terms, these merge, making four redundant entries. One of the combinations is \langle \text{empty} \rangle, a sequence of exactly zero non-space tokens: this is the only combination that doesn’t put anything in an F-column—and was the most difficult part to handle.

The minus sign must be captured separately, because in an entry like \texttt{-0,07} the 7 cents are negative, but this cannot be seen from the part to the left of the decimal mark, since \texttt{-0} is 0 in \TeX{} (in fact in most computer languages, but not in MIX [8]), so \texttt{\texttt{if}num\texttt{-0}<0} yields \texttt{false}. \texttt{FCs@gn} is a general purpose flag. Its first use is to capture the sign.

Do the scan inside a box and inside math mode. Start with defining all characters that may appear as the first one in an F-column as active.

For the input decimal mark something extra is needed: if it is the first character in an F-column (like in \texttt{,07}), it should also restore the \texttt{mathcodes} of the digits. Checking whether or not it is the first is easy, since in that case the \texttt{mathcodes} of the decimal digits is still \texttt{"8000}. The assignment to \texttt{FC@r} starts with 1, so that appended digits get captured correctly, even if they start with 0. Postprocessing of \texttt{FC@r} is done in \texttt{\texttt{e@fi}}. The input decimal mark switches itself off as active character, so at most one input decimal separator is allowed (N.B.: this makes sense).
The input grouping character effectively expands to “nothing, i.e., ignore” in a complicated way: it ignores the character and resumes scanning the number. The test prior to that action is needed if the grouping character is the first character encountered in the \texttt{F}-column. Which part to continue with depends on whether or not an input decimal mark was encountered; that can be checked by looking at its \texttt{\mathcode}.

The input grouping character is the dot \texttt{.}, except when that character was already chosen as input decimal mark. In that case, the grouping character will be the comma. This is easy to check because the \texttt{\uccode} of \texttt{`} is still preserved.

The \texttt{\expandafter} below is necessary because the global assignment should act after the \texttt{\fi}.

Now actually activate all these codes. The first is simple, but after that, one can’t say "8000 anymore because 0 acts as active. But copying \texttt{\mathcodes} still works.

These three remain active until the $\texttt{\e@fi}$ is encountered. The following ones will, except in the (empty) case, have their activeness turned off at some time.

If the digits are still active then either nothing was entered or only characters that did not deactivate the digits were entered. In either case the output should be ⟨empty⟩. To flag this outside the group that started with the opening $\texttt{\b@fi}$, \texttt{\FC@r} is set globally to a negative value. This doesn’t harm, because it didn’t
contain relevant information anyway. Outside the group, the sign of \texttt{FC@r} can
then be tested. This is a slight misuse of this \texttt{count}, but now it’s documented. In
effect, \texttt{FC@r} can only be -1, 1, or at least 10, so the comparison \texttt{ifnum FC@r>0}
does not mis 0.

\texttt{\def\e@fi{\ifnum\mathcode\0=\mathcode'- \global\FC@r=\m@ne\fi}\egroup}

\texttt{\ifnum\FC@r>0}

If there was no decimal mark or if there was a decimal mark but no decimal
digit, \texttt{FC@r} will still be 1, which doesn’t parse well with \texttt{secd@xt}, so a zero is
appended, i.e., yielding 10.

\texttt{\ifnum\FC@r<10 \FC@r=10 \fi}

Next is a loop for bringing the decimal part in the correct way to the integer part.
The loop is performed the number of decimal digits to be printed (the 2 in 3,2
of the default setting). This also means that if you provided more decimal digits
than this, the excess digit(s) will not be handled and a \texttt{PackageWarning} will be
given, showing these excess digits. This is truncation, not rounding!

\texttt{\def\i@ts##1##2{\count0=##2}}
\texttt{\afterassignment\i@ts \count@\sp@l}
\texttt{\loop\ifnum\count0>0 \multiply\FC@l by 10
\expandafter\secd@xt\number\FC@r\end \advance\count0 by \m@ne
\repeat}
\texttt{\ifnum\FC@r>10}
\texttt{\def\tw@l##1##2\relax{##2}}
\texttt{\PackageWarning{fcolumn}{Excess digit\ifnum\FC@r>100 s\fi \space
\expandafter\tw@l\number\FC@r\relax'' in decimal part
\MessageBreak ignored}}
\texttt{\fi}

Don’t forget to correct for the sign (once this is done, \texttt{FCs@gn} is free again and
can and will be used for other purposes). Then output the result.

\texttt{\ifx\FCs@gn\m@ne\relax \FC@l=\FC@l \fi}
\texttt{\zetg@ld\sp@l\FC@l}\%}
\texttt{\fi}

\texttt{\secd@xt} The second digit from the left is needed from a decimal number. The macro
\texttt{\secd@xt} extracts that digit, provided that the number has at least two digits,
but that is guaranteed by \texttt{\@fi}. That second digit is then added to \texttt{FC@l}. A
new number is assigned to \texttt{FC@r}, that consists of the digits of \#1\#3. If \#3 was
empty, a zero is appended. In this way \texttt{FC@r} is prepared for insertion in the
next invocation of \texttt{secd@xt}. In iterating: 1234 yields 134, yields 14, yields 10,
stays 10, etc.

\texttt{\def\secd@xt#1#2#3\end{\advance\FC@l by #2}
\texttt{\FC@r=#1#3 \ifnum\FC@r<10 \multiply\FC@r by 10 \fi}}

\texttt{\restore@thcodes} As shown above, once the first digit, or sign, or decimal separator, or grouping
character is scanned, the decimal digits should lose their activeness. That is
done here rather blunt, since the actual \texttt{mathcode} is not important—as long as
it is not \#8000—because the digits are not used for typesetting (and even if they
were; it’s inside \box0, whose contents will be discarded). When the $ in \e@fi is encountered, the digits get back their original \mathcodes so that the actual typesetting in \zetg@ld is correct again.

\def\restore@mathcodes{\mathcode\0=0 \mathcode\1=0 \mathcode\2=0 \mathcode\3=0 \mathcode\4=0 \mathcode\5=0 \mathcode\6=0 \mathcode\7=0 \mathcode\8=0 \mathcode\9=0 }

### 3.3 Adaptations to existing macros

The definition of \@array had to be extended slightly because it should also include \@mksumline (acting on the same #2 as \@mkpream gets). This change is transparant: it only adds functionality and if you don’t use that, you won’t notice the difference. It starts by just copying the original definition from v2.4k of the array package [1].

\def\@array[#1]#2{\@tempdima \ht \strutbox \advance \@tempdima by\extrarowheight \setbox \@arstrutbox \hbox{\vrule \@height \arraystretch \@tempdima \@depth \arraystretch \dp \strutbox \@width \z@} \def\ialign{\everycr{\noalign{\global\FCsc@l=0 }}\tabskip\z@skip\halign} Here comes the first change: after each \\ (or \cr for that matter) the \(count\) \FCsc@l should be reset. This is easiest done with \everycr, but \everycr is put to {} by \ialign, so that definition should change. The resetting should be done globally.

\def\ialign{\everycr{\noalign{\global\FCsc@l=0 }}}\tabskip\z@skip\ialign\halign

Then the definition is picked up again.

\begingroup \@mkpream{#2} \xdef\@preamble{\noexpand \ialign \noexpand \halign} \begingroup \@mksumline{#2} \endgroup As a side product of \@mksumline also the \(count\)s for the totals and \(dimen\)s for the widths of the columns are created. The columns should start fresh, i.e., totals are 0 and widths are 0 pt.

\res@tsumline
From here on it is just the old definition of $\text{array.sty}$.

\@arrayleft
\if #1t\vtop \else \if#1b\vbox \else \vcenter fi fi
\bgroup
\let \@sharp ##
\let \protect \relax
\lineskip \z@
\baselineskip \z@
\m@th
\let\\@arraycr \let\tabularnewline\\ \let\par\@empty \@preamble}

Because $\text{@array}$ was changed here and it is this version that should be used, $\text{@@array}$ should be \let equal to $\text{@array}$ again.

\let\@@array=\@array

Much of the techniques here are repeated in $\text{\LT@array}$.

### 3.4 The sumline, close to a postamble

The construction of the sumline is much easier than that of the preamble for several reasons. It may be safely assumed that the preamble specifier is grammatically correct because it has already been screened by $\text{@mkpream}$. Furthermore, most entries will simply add nothing to $\text{@s@ml@ne}$, e.g., $\&$, $!$, and $|$ can be fully ignored. Ampersands are only inserted by $c$, $l$, $r$, $p$, $m$, and $b$. So a specifier like $@{}lf1f0@{}$ will yield the sumline $&\a&&\a\backslash$, (where $\a$ is a macro that prints the desired result of the column, see below). Had the specifier been $l|f||@{}l|f$, then the same sumline must be constructed: all difficulties are already picked up and solved in the creation of the preamble.

In reality the sumline must be constructed from the expanded form of the specifier, so $@{}lf0@{}$ will expand as $@{}l>{\b@fi{.}{,}{3,2}{}}r<{\e@fi}@{}$. The rules for constructing the sumline are now very simple:

- add an ampersand when $c$, $l$, $r$, $p$, $m$, or $b$ is found, unless it is the first one (this is the same as in the preamble);
- add a $\a$ when $<{\e@fi}$ is found;
- ignore everything else;
- close with a $\\$.

(In reality also the column check is inserted just before the $\\$, see $\text{\aut@check}$.)

To discriminate, a special version of $\text{@testpach}$ could be written, but that is not necessary: $\text{@testpach}$ can do all the work, although much of it will be discarded. Here speed is sacrificed for space and this can be afforded because the creation of the sumline is done only once per $\text{\tabular}$ or $\text{\longtable}$.

The start is copied from $\text{@mkpream}$.

\def\@mksumline#1{\gdef@s@ml@ne{}\@lastchclass 4 \@firstamptrue
\global\FCsc@l=0
\let\prr@sult=\relax

At first the column number is reset and the actual code for what was called $\a$ above is made inactive.

\global\FCsc@1=0
\let\prr@sult=\relax
Then \@mkpream is picked up again.

Next is the loop over all tokens in the expanded form of the specifier. The change with respect to \@mkpream is that the body of the loop is now only dealing with F-classes 0, 2, and 10. What to do in those cases is of course different from what to do when constructing the preamble, so special definitions are created, see below.

And the macro is finished by applying the \aut@check and appending the \ to the sumline. Note that the \aut@check is performed in the last column, but since it does not put anything in the horizontal list—it only writes to screen and transcript file—, this is harmless.

Macro \@addtosumline, as its name already suggests, adds something to the sumline, like its counterpart \@addtopreamble did to the preamble.

Class f10 for the sumline creation is a stripped down version of \@classx: add an ampersand unless it is the first. It deals with the specifiers b, m, p, c, l, and r.

If both tests yield true, i.e., we encountered a <{e@fi} where we expect one to find, then add the macro to typeset everything.
But we're not done yet: in the following lines of code the appropriate \langle count\rangle s and \langle dimen\rangle s are created, if necessary. Note that \texttt{FCsc@l} was set to 0 in the beginning of \texttt{\@mksumline}, so it is well-defined when \texttt{\@classfii} is used.

\begin{verbatim}
204 \global\advance\texttt{FCsc@l} by \texttt{\@ne}
205 \ifnum\texttt{FCsc@l} > \texttt{FCtc@l}

 Apparently the number of requested columns is larger than the currently available number of relevant \langle count\rangle s and \langle dimen\rangle s, so new ones should be created. What is checked here is merely the existence of \texttt{FCtot@<some roman numeral>}. If it already exists—although it may not even be a \langle count\rangle; that cannot be checked—it is not created by \texttt{fcolumn} and an error is given. In case it is a \langle count\rangle you're just lucky, and you could ignore that error, although any change to this \langle count\rangle is global anyway, so things will be overwritten. In the case it is not a \langle count\rangle, things will go haywire and you'll soon find out. The remedy then is to rename your \langle count\rangle prior to \texttt{fcolumn} to avoid this name clash.

\begin{verbatim}
206 \expandafter\ifx\csname FCtot@\romannumeral\texttt{FCsc@l}\endcsname\relax
207 \expandafter\newcount\csname FCtot@\romannumeral\texttt{FCsc@l}\endcsname
208 \else
209 \PackageError{fcolumn}{Name clash for \langle count\rangle}{\expandafter\csname 
210 FCtot@\romannumeral\texttt{FCsc@l}\endcsname is already defined and it may 
211 not even be a \langle count\rangle. If you're\MessageBreak sure it is a \langle count\rangle, 
212 you can press \texttt{<enter>} now and I'll proceed, but things\MessageBreak 
213 will get overwritten.}%
214 \fi
215 \expandafter\ifx\csname FCwd@\romannumeral\texttt{FCsc@l}\endcsname\relax
216 \expandafter\newdimen\csname FCwd@\romannumeral\texttt{FCsc@l}\endcsname
217 \else
218 \PackageError{fcolumn}{Name clash for \langle dimen\rangle}{\expandafter\csname 
219 FCwd@\romannumeral\texttt{FCsc@l}\endcsname is already defined and it may 
220 not even be a \langle dimen\rangle. If you're\MessageBreak sure it is a \langle dimen\rangle, 
221 you can press \texttt{<enter>} now and I'll proceed, but things\MessageBreak 
222 will get overwritten.}%
223 \fi
224 \fi
225 \fi
226 \fi
227 \fi}
\end{verbatim}

And the same is applicable for the \langle dimen\rangle: in case of a name clash you have to rename your \langle dimen\rangle prior to \texttt{fcolumn}.

\begin{verbatim}
215 \expandafter\ifx\csname FCwd@\romannumeral\texttt{FCsc@l}\endcsname\relax
216 \expandafter\newdimen\csname FCwd@\romannumeral\texttt{FCsc@l}\endcsname
217 \else
218 \PackageError{fcolumn}{Name clash for \langle dimen\rangle}{\expandafter\csname 
219 FCwd@\romannumeral\texttt{FCsc@l}\endcsname is already defined and it may 
220 not even be a \langle dimen\rangle. If you're\MessageBreak sure it is a \langle dimen\rangle, 
221 you can press \texttt{<enter>} now and I'll proceed, but things\MessageBreak 
222 will get overwritten.}%
223 \fi
224 \fi
225 \fi
226 \fi
227 \fi}
\end{verbatim}

If the creation was successful, the \langle count\rangle \texttt{FCtc@l} should be increased.

\begin{verbatim}
217 \global\texttt{FCtc@l}=\texttt{FCsc@l}
218 \else
219 \PackageError{fcolumn}{Name clash for \langle dimen\rangle}{\expandafter\csname 
220 FCwd@\romannumeral\texttt{FCsc@l}\endcsname is already defined and it may 
221 not even be a \langle dimen\rangle. If you're\MessageBreak sure it is a \langle dimen\rangle, 
222 you can press \texttt{<enter>} now and I'll proceed, but things\MessageBreak 
223 will get overwritten.}%
224 \fi
225 \fi
226 \fi
227 \fi}
\end{verbatim}

Once created, it is not necessary to initialise them here because that is done later in one go.

\texttt{\sumline} The command for the sumline has one optional argument: the separation between the rule and the total. By default this is 2 pt, but the user may specify \texttt{\sumline[10pt]} if that separation needs to be 10 pt. The assignment needs to be
global, because it is done in the first column of the tabular, but is valid for the whole line.

\newdimen\s@mlinesep
\def\sumline{\@ifnextchar[\s@mline{\s@mline[2pt]}}
\def\s@mline[#1]{\global\s@mlinesep=#1 \s@mline}

In the introduction it was stated that \sumline has two options, but in reality that second option is the option to \ that is issued by \s@mline.

\prr@sult

The macro \prr@sult actually puts the information together. It starts like \leeg.

\def\prr@sult{$\egroup \let\e@fi=\relax \let\FCform@t=\FCform@tt

Then the information for the last line is computed. It is not sufficient to calculate the width of the result (in points) to use that as the width of the rule separating the individual entries and the result. It may be that the sum is larger (in points) than any of the entries, e.g., when the result of 6+6 (using specifier 3,2) is typeset. The width of the rule should be equal to the width of \hbox\$12\{,\}00\$ then. On the other hand the width of the rule when summing 24 and –24 should be that of \hbox\$-24\{,\}00\$ (or \hbox\$24\{,\}00\$, see above), not the width of the result \hbox\$0\{,\}00\$. Therefore the maximum of all entry widths, including the result, was calculated.

\setbox0=\hbox{$\geldm@cro{\sp@l}{\number\csname FCtot@\romannumeral\FCsc@l\endcsname}$}\
\ifdim\wd0>\csname FCwd@\romannumeral\FCsc@l\endcsname\fi
\vbox{\hrule width \csname FCwd@\romannumeral\FCsc@l\endcsname\vskip\s@mlinesep
\hbox to \csname FCwd@\romannumeral\FCsc@l\endcsname{\hfil\unhbox0}}}\

3.5 Other checks

\leeg

This macro is used to overrule the default behaviour of the pair \b@fi and \e@fi. It starts with ending the groups in the same way that \e@fi would normally do. Then the effect of \e@fi (that is still in the preamble) is annihilated by \letting it to be \relax. This \let is only local to the current column. Then the argument to \leeg is treated in a similar way as \e@fi would do with a typeset number.

Since the user may from time to time also need a column entry other than a number in the table, e.g., \leeg{p.m.}, this definition is without at-sign. By defining \leeg in this way, instead of \multicolumn1r{} (which contains \omit), the default spacing in the column is retained. It has its normal effect on the column width, but doesn’t alter the width of the sumrule.

\def\leeg#1{$\egroup \let\e@fi=\relax #1}

Note that anything may be given as argument to \leeg, so in principle it can also be used to cheat: \leeg{0,03} will insert 0,03 in the table but it doesn’t increase the totals of that column by 3 (assuming 3,2 coding for the separations). But you won’t cheat, won’t you? It may affect the width, so be careful: don’t insert the unabridged version of Romeo and Juliet [9] here.
Since all changes to the totals and widths of the columns are global, they have to be reset actively at the start of a tabular or array. That is an action by itself, but it may occur more often, on request of the user, therefore a special macro is defined. A side effect of this macro is that FCsc01 is reset to 0. This is an advantage: it should be zero at the beginning of a line in the table (for other lines this is done by the \\).

\resetsumline To reset a sumline within a table, it should be done within a \noalign.

\aut@check If the number of F-columns is even, it is assumed that they are part of two sets of columns of which each column of the first set should balance the appropriate column of the second set. If on the other hand the number of columns is odd, then at least one column has nothing to balance against and no checking occurs. It is correct to check for oddness of FCsc01 since this \aut@check is only performed in the last column of the tabular: the value of FCsc01 now equals the number of columns used in the current tabular (and may differ from FCtc01).

The output is only to screen and the transcript file; it doesn’t change the appearance of your document, so in case the assumption is wrong you can safely ignore the result and go on. The (count)0 and 1 are used here and this can be done because any content of those (count)0s from previous calculations has become irrelevant at this moment.

If the list FC@chklist is empty, the list for the automatic check is generated (which will remain empty if FCsc01 is odd).

\resetsumline{\noalign{\resetsumline}}

\aut@check If the number of F-columns is even, it is assumed that they are part of two sets of columns of which each column of the first set should balance the appropriate column of the second set. If on the other hand the number of columns is odd, then at least one column has nothing to balance against and no checking occurs. It is correct to check for oddness of FCsc01 since this \aut@check is only performed in the last column of the tabular: the value of FCsc01 now equals the number of columns used in the current tabular (and may differ from FCtc01).

The output is only to screen and the transcript file; it doesn’t change the appearance of your document, so in case the assumption is wrong you can safely ignore the result and go on. The (count)0 and 1 are used here and this can be done because any content of those (count)0s from previous calculations has become irrelevant at this moment.

If the list FC@chklist is empty, the list for the automatic check is generated (which will remain empty if FCsc01 is odd).
When \texttt{\aut@check} is finished, \texttt{\FC@chklist} is empty again, i.e., well prepared for the next time it is used. This also means that the default behaviour kicks in again: if that's not what you want, you should specify the appropriate \texttt{\checkfcolumns} lines again.

This function eats the first two numbers off \texttt{\FC@chklist}.

\begin{verbatim}
\def\fre@t#1,#2;#3\end{
\count0=#1 \count1=#2 \xdef\FC@chklist{#3}}
\end{verbatim}

But the assumptions for \texttt{\aut@check} may be wrong, therefore manual control on this checking is also made possible here. The macro \texttt{\checkfcolumns} provides a way to the user to check that the appropriate columns are balanced (as it should in a balance). Arguments \texttt{#1} and \texttt{#2} are the F-column numbers to compare. It is the responsibility of the user to provide the correct numbers here, otherwise bogus output is generated. If this manual check is inserted, the automatic check will not be performed.

\begin{verbatim}
\def\checkfcolumns#1#2classed{\xdef\FC@chklist{#1,#2;\ \def\FC@chklist{\end}}
\end{verbatim}

\section*{3.6 Support for package \texttt{longtable}}

Package \texttt{longtable} is used for tables that may span multiple pages. \texttt{fcolumn} and \texttt{longtable} work together as long as \texttt{longtable} is loaded first, so that \texttt{fcolumn} can adapt one definition of \texttt{longtable}. If \texttt{longtable} is not loaded, that definition is just that: a definition. If \texttt{longtable} is loaded before \texttt{fcolumn}, it is a redefinition, and it should be. For that reason the user is warned (not loading \texttt{longtable} is not an error if you don’t use it) if \texttt{fcolumn} is loaded without prior loading of \texttt{longtable}.

\begin{verbatim}
\ifx\longtable\@undefined\PackageWarningNoLine{fcolumn}{fcolumn is loaded without longtable. That's OK, but \MessageBreak if you want to load longtable as well, make sure \MessageBreak it is done before loading fcolumn}\fi
\end{verbatim}

And here is the only definition of \texttt{longtable} that needs to be adapted a bit, to make \texttt{fcolumn} work with that package.

\begin{verbatim}
\def\LT@array[#1]#2{\refstepcounter{table}\stepcounter{LT@tables}\percent
\if l#1 \LTleft\z@\LTright\fill\percent
\else\if r#1 \LTleft\fill\LTright\z@\percent
\else\if c#1 \LTleft\fill\LTright\fill\percent
\fi\fi\fi
\let\LT@mcol\multicolumn\let\LT@@tabarray\@tabarray\let\LT@@hl\hline\percent
\def\@tabarray{\let\hline\LT@@hl \LT@@tabarray}\percent
\let\\LT@tabularcr\\let\tabularnewline\percent}
\end{verbatim}
Until this line it was just the code for \LT@array from package \texttt{longtable}. The five lines of the next chunk are new to \LT@array. Their purpose is the same as in \texttt{array} above. That's it!

\textbf{Acknowledgement}

Thanks to Karl Berry for valuable comments regarding the consistency of the installation procedure of this version. Frank Mittelbach gave various useful suggestions for improving the input parsing as well as hints to make the package more \LaTeX-like. He also challenged me to make fcolumn compatible with \texttt{longtable}. Christian Hoff's request on column formatting triggered many happy hours of coding.
References

[1] Frank Mittelbach and David Carlisle. A new implementation of \LaTeX’s \texttt{tabular} and \texttt{array} environment.
[2] Simon Fear. Publication quality tables in \LaTeX.

Change History

v0.1
  General: First working version. . . . 1

v1.0
  General: Three-argument version is working properly. . . . 1

v1.1
  General: Automatic checking of column balance performed when number of F-columns is even (behaviour can be overridden). Empty entries are now recognised and correctly treated as such, except for the one ended by the double backslash. Not serious; workaround possible. Furthermore optimisation of code: minimised the number of private counts and resetting of column counter done in a nicer way. . . . . 1

v1.1.1
  General: Installation procedure changed from \texttt{.ins-in-.dtx} to separate \texttt{.ins} and \texttt{.dtx} after discussion with Karl Berry as well as some minor code improvements. . . . . 1

v1.1.2
  General: Some inconsistencies between explanatory text and actual code removed. . . . . 1

v1.2
  General: Input parsing changed after comment from Frank Mittelbach. He (Frank) also gave various suggestions for improving robustness or user friendliness of this package. This version is only backwards compatible when zero decimal digits were and are specified as modifier. . . . . 1

v1.3
  General: Christian Hoff requested the possibility to provide extra formatting information to a column, e.g., colouring. That was implemented by having an extra parameter to the generic F-column. The current solution is not very robust, as font and/or size change in math environment are very tricky, but providing colour information works, see the example in the main text. This version is now compatible with package longtable. Version 1.3 is backwards compatible to 1.2: it only adds functionality. . . . 1
### Index

Numbers written in italic refer to the page where the corresponding entry is described; numbers underlined refer to the code line of the definition; numbers in roman refer to the code lines where the entry is used.

#### Symbols

<table>
<thead>
<tr>
<th>Command</th>
<th>Page(s)</th>
<th>Page(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>@array</td>
<td>183</td>
<td></td>
</tr>
<tr>
<td>@addtosumline</td>
<td></td>
<td>106, 197, 203</td>
</tr>
<tr>
<td>@array</td>
<td>157</td>
<td></td>
</tr>
<tr>
<td>@arraycr</td>
<td>182, 295</td>
<td></td>
</tr>
<tr>
<td>@arrayleft</td>
<td>175</td>
<td></td>
</tr>
<tr>
<td>@chclass</td>
<td>193, 194</td>
<td></td>
</tr>
<tr>
<td>@chnum</td>
<td>198</td>
<td></td>
</tr>
<tr>
<td>@classfii</td>
<td>193, 199</td>
<td></td>
</tr>
<tr>
<td>@classfx</td>
<td>194, 197, 198</td>
<td></td>
</tr>
<tr>
<td>@classfz</td>
<td>198</td>
<td></td>
</tr>
<tr>
<td>@empty</td>
<td>182, 247, 259, 314</td>
<td></td>
</tr>
<tr>
<td>@endpbox</td>
<td>288, 292</td>
<td></td>
</tr>
<tr>
<td>@ifnextchar</td>
<td></td>
<td>229</td>
</tr>
<tr>
<td>@lastchclass</td>
<td></td>
<td>184, 194, 199</td>
</tr>
<tr>
<td>@mksumline</td>
<td></td>
<td></td>
</tr>
<tr>
<td>@startpbox</td>
<td>291, 292, 295</td>
<td></td>
</tr>
<tr>
<td>@tabacol</td>
<td></td>
<td>290</td>
</tr>
<tr>
<td>@tabarray</td>
<td>281, 282</td>
<td></td>
</tr>
<tr>
<td>@tabclassiv</td>
<td></td>
<td>290</td>
</tr>
<tr>
<td>@tabclassz</td>
<td></td>
<td>290</td>
</tr>
<tr>
<td>@tabularcr</td>
<td></td>
<td></td>
</tr>
<tr>
<td>@temptokena</td>
<td></td>
<td>187, 191</td>
</tr>
<tr>
<td>@testpach</td>
<td></td>
<td></td>
</tr>
<tr>
<td>@undefined</td>
<td></td>
<td>309</td>
</tr>
<tr>
<td>@empty</td>
<td></td>
<td>288, 292</td>
</tr>
<tr>
<td>@ifnextchar</td>
<td></td>
<td>229</td>
</tr>
<tr>
<td>@lastchclass</td>
<td></td>
<td>184, 194, 199</td>
</tr>
<tr>
<td>@mksumline</td>
<td></td>
<td>172, 184, 309</td>
</tr>
<tr>
<td>@startpbox</td>
<td>291, 292, 295</td>
<td></td>
</tr>
<tr>
<td>@tabacol</td>
<td></td>
<td>290</td>
</tr>
<tr>
<td>@tabarray</td>
<td>281, 282</td>
<td></td>
</tr>
<tr>
<td>@tabclassiv</td>
<td></td>
<td>290</td>
</tr>
<tr>
<td>@tabclassz</td>
<td></td>
<td>290</td>
</tr>
<tr>
<td>@tabularcr</td>
<td></td>
<td></td>
</tr>
<tr>
<td>@undefined</td>
<td></td>
<td>309</td>
</tr>
<tr>
<td>@empty</td>
<td></td>
<td></td>
</tr>
<tr>
<td>@ifnextchar</td>
<td></td>
<td>229</td>
</tr>
<tr>
<td>@lastchclass</td>
<td></td>
<td>184, 194, 199</td>
</tr>
<tr>
<td>@mksumline</td>
<td></td>
<td>172, 184, 309</td>
</tr>
<tr>
<td>@startpbox</td>
<td>291, 292, 295</td>
<td></td>
</tr>
<tr>
<td>@tabacol</td>
<td></td>
<td>290</td>
</tr>
<tr>
<td>@tabarray</td>
<td>281, 282</td>
<td></td>
</tr>
<tr>
<td>@tabclassiv</td>
<td></td>
<td>290</td>
</tr>
<tr>
<td>@tabclassz</td>
<td></td>
<td>290</td>
</tr>
<tr>
<td>@tabularcr</td>
<td></td>
<td></td>
</tr>
<tr>
<td>@undefined</td>
<td></td>
<td>309</td>
</tr>
<tr>
<td>@empty</td>
<td></td>
<td></td>
</tr>
<tr>
<td>@ifnextchar</td>
<td></td>
<td>229</td>
</tr>
<tr>
<td>@lastchclass</td>
<td></td>
<td>184, 194, 199</td>
</tr>
<tr>
<td>@mksumline</td>
<td></td>
<td>172, 184, 309</td>
</tr>
<tr>
<td>@startpbox</td>
<td>291, 292, 295</td>
<td></td>
</tr>
<tr>
<td>@tabacol</td>
<td></td>
<td>290</td>
</tr>
<tr>
<td>@tabarray</td>
<td>281, 282</td>
<td></td>
</tr>
<tr>
<td>@tabclassiv</td>
<td></td>
<td>290</td>
</tr>
<tr>
<td>@tabclassz</td>
<td></td>
<td>290</td>
</tr>
<tr>
<td>@tabularcr</td>
<td></td>
<td></td>
</tr>
<tr>
<td>@undefined</td>
<td></td>
<td>309</td>
</tr>
<tr>
<td>@empty</td>
<td></td>
<td></td>
</tr>
<tr>
<td>@ifnextchar</td>
<td></td>
<td>229</td>
</tr>
</tbody>
</table>

#### Numbers

<table>
<thead>
<tr>
<th>Number</th>
<th>Page(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>\0</td>
<td></td>
</tr>
<tr>
<td>\1</td>
<td></td>
</tr>
<tr>
<td>\2</td>
<td></td>
</tr>
<tr>
<td>\3</td>
<td></td>
</tr>
<tr>
<td>\4</td>
<td></td>
</tr>
<tr>
<td>\5</td>
<td></td>
</tr>
<tr>
<td>\6</td>
<td></td>
</tr>
<tr>
<td>\7</td>
<td></td>
</tr>
<tr>
<td>\8</td>
<td></td>
</tr>
</tbody>
</table>

25
<table>
<thead>
<tr>
<th>Command</th>
<th>Page(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>\strict@countingfalse</code></td>
<td>1</td>
</tr>
<tr>
<td><code>\strict@countingtrue</code></td>
<td>2</td>
</tr>
<tr>
<td><code>\sumline</code></td>
<td>1, 228</td>
</tr>
<tr>
<td><code>\tabcolsep</code></td>
<td>294</td>
</tr>
<tr>
<td><code>\tw@1</code></td>
<td>144, 146</td>
</tr>
<tr>
<td><code>\withs@pfalse</code></td>
<td>8</td>
</tr>
<tr>
<td><code>\withs@ptrue</code></td>
<td>50</td>
</tr>
<tr>
<td><code>\uppercase</code></td>
<td>100–109, 112, 116, 121, 124</td>
</tr>
<tr>
<td><code>\ucode</code></td>
<td>92, 100–109, 115, 120, 123</td>
</tr>
<tr>
<td><code>\t@stm</code></td>
<td>200, 202</td>
</tr>
<tr>
<td><code>\t@stn</code></td>
<td>201, 202</td>
</tr>
<tr>
<td><code>\z@p@d</code></td>
<td>30, 49, 50, 51</td>
</tr>
<tr>
<td><code>\zetg@d</code></td>
<td>62, 150</td>
</tr>
</tbody>
</table>