A new implementation of LaTeX's tabular and array environment*

Frank Mittelbach David Carlisle†

Printed November 29, 1999

Abstract

This article describes an extended implementation of the LaTeX array– and tabular–environments. The special merits of this implementation are further options to format columns and the fact that fragile LaTeX–commands don't have to be `\protect`'ed any more within those environments.

The major part of the code for this package dates back to 1988—so does some of its documentation.

1 Introduction

This new implementation of the array– and tabular–environments is part of a larger project in which we are trying to improve the LaTeX-code in some aspects and to make LaTeX even easier to handle.

The reader should be familiar with the general structure of the environments mentioned above. Further information can be found in [3] and [1]. The additional options which can be used in the preamble as well as those which now have a slightly different meaning are described in table 1.

`\extrarowheight` Additionally we introduce a new parameter called `\extrarowheight`. If it takes a positive length, the value of the parameter is added to the normal height of every row of the table, while the depth will remain the same. This is important for tables with horizontal lines because those lines normally touch the capital letters. For example, we used `\setlength{\extrarowheight}{1pt}` in table 1.

We will discuss a few examples using the new preamble options before dealing with the implementation.

- If you want to use a special font (for example `\bfseries`) in a flushed left column, this can be done with `>{\bfseries}l`. You do not have to begin every entry of the column with `\bfseries` any more.
- In columns which have been generated with `p`, `m` or `b`, the default value of `\parindent` is 0pt. This can be changed with `>{\setlength{\parindent}{1cm}}p`.
- The `>`– and `<`–options were originally developed for the following application: `>{$}c<{$}` generates a column in math mode in a tabular–environment. If you use this type of a preamble in an array–environment, you get a column in LR mode because the additional $'s cancel the existing $'s.

*This file has version number v2.3m, last revised 1998/05/13.

†David kindly agreed on the inclusion of the `\newcolumntype` implementation, formerly in `newarray.sty` into this package

Unchanged options	
`l`	Left adjusted column.
`c`	Centered adjusted column.
`r`	Right adjusted column.
`p{width}`	Equivalent to `\parbox[t]{width}`.
`@{decl.}`	Suppresses inter-column space and inserts `decl.` instead.
New options	
`m{width}`	Defines a column of width `width`. Every entry will be centered in proportion to the rest of the line. It is somewhat like `\parbox{width}`.
`b{width}`	Coincides with `\parbox[b]{width}`.
`>{decl.}`	Can be used before an `l`, `r`, `c`, `p`, `m` or a `b` option. It inserts `decl.` directly in front of the entry of the column.
`<{decl.}`	Can be used after an `l`, `r`, `c`, `p{..}`, `m{..}` or a `b{..}` option. It inserts `decl.` right after the entry of the column.
`|`	Inserts a vertical line. The distance between two columns will be enlarged by the width of the line in contrast to the original definition of LaTeX.
`!{decl.}`	Can be used anywhere and corresponds with the `|` option. The difference is that `decl.` is inserted instead of a vertical line, so this option doesn't suppress the normally inserted space between columns in contrast to `@{...}`.

Table 1: The preamble options.

- One can also think of more complex applications. A problem which has been mentioned several times in TeXhax can be solved with `>{\centerdots}c<{\endcenterdots}`. To center decimals at their decimal points you (only?) have to define the following macros:

```
{\catcode`\.\active\gdef.{\egroup\setbox2\hbox\bgroup}}
\def\centerdots{\catcode`\.\active\setbox0\hbox\bgroup}
\def\endcenterdots{\egroup\ifvoid2 \setbox2\hbox{0}\fi
   \ifdim \wd0>\wd2 \setbox2\hbox to\wd0{\unhbox2\hfill}\else
     \setbox0\hbox to\wd2{\hfill\unhbox0}\fi
   \catcode`\.12 \box0.\box2}
```

  Warning: The code is bad, it doesn't work with more than one dot in a cell and doesn't work when the tabular is used in the argument of some other command. A much better version is provided in the `dcolumn.sty` by David Carlisle.

- Using `c!{\hspace{1cm}}c` you get space between two columns which is enlarged by one centimeter, while `c@{\hspace{1cm}}c` gives you exactly one centimeter space between two columns.

1.1 Defining new column specifiers

`\newcolumntype` Whilst it is handy to be able to type

`>{`⟨*some declarations*⟩`}{c}<{`⟨*some more declarations*⟩`}`

if you have a one-off column in a table, it is rather inconvenient if you often use columns of this form. The new version allows you to define a new column specifier, say `x`, which

will expand to the primitives column specifiers.[1] Thus we may define

```
\newcolumntype{x}{>{⟨some declarations⟩}{c}<{⟨some more declarations⟩}}
```

One can then use the `x` column specifier in the preamble arguments of all `array` or `tabular` environments in which you want columns of this form.

It is common to need math-mode and LR-mode columns in the same alignment. If we define:

```
\newcolumntype{C}{>{$}c<{$}}
\newcolumntype{L}{>{$}l<{$}}
\newcolumntype{R}{>{$}r<{$}}
```

Then we can use `C` to get centred LR-mode in an `array`, or centred math-mode in a `tabular`.

The example given above for 'centred decimal points' could be assigned to a `d` specifier with the following command.

```
\newcolumntype{d}{>{\centerdots}c<{\endcenterdots}}
```

The above solution always centres the dot in the column. This does not look too good if the column consists of large numbers, but to only a few decimal places. An alternative definition of a `d` column is

```
\newcolumntype{d}[1]{>{\rightdots{#1}}r<{\endrightdots}}
```

where the appropriate macros in this case are:[2]

```
\def\coldot{.}% Or if you prefer, \def\coldot{\cdot}
{\catcode`\.=\active
  \gdef.{$\egroup\setbox2=\hbox to \dimen0 \bgroup$\coldot}}
\def\rightdots#1{%
  \setbox0=\hbox{$1$}\dimen0=#1\wd0
  \setbox0=\hbox{$\coldot$}\advance\dimen0 \wd0
  \setbox2=\hbox to \dimen0 {}%
  \setbox0=\hbox\bgroup\mathcode`\.="8000 $}
\def\endrightdots{$\hfil\egroup\box0\box2}
```

Note that `\newcolumntype` takes the same optional argument as `\newcommand` which declares the number of arguments of the column specifier being defined. Now we can specify `d{2}` in our preamble for a column of figures to at most two decimal places.

A rather different use of the `\newcolumntype` system takes advantage of the fact that the replacement text in the `\newcolumntype` command may refer to more than one column. Suppose that a document contains a lot of `tabular` environments that require the same preamble, but you wish to experiment with different preambles. Lamport's original definition allowed you to do the following (although it was probably a mis-use of the system).

```
\newcommand{\X}{clr}
\begin{tabular}{\X} ...
```

`array.sty` takes great care **not** to expand the preamble, and so the above does not work with the new scheme. With the new version this functionality is returned:

```
\newcolumntype{X}{clr}
\begin{tabular}{X} ...
```

The replacement text in a `\newcolumntype` command may refer to any of the primitives of `array.sty` see table 1 on page 2, or to any new letters defined in other `\newcolumntype` commands.

`\showcols` A list of all the currently active `\newcolumntype` definitions is sent to the terminal and log file if the `\showcols` command is given.

[1] This command was named `\newcolumn` in the `newarray.sty`. At the moment `\newcolumn` is still supported (but gives a warning). In later releases it will vanish.

[2] The package `dcolumn.sty` contains more robust macros based on these ideas.

1.2 Special variations of `\hline`

The family of `tabular` environments allows vertical positioning with respect to the baseline of the text in which the environment appears. By default the environment appears centered, but this can be changed to align with the first or last line in the environment by supplying a `t` or `b` value to the optional position argument. However, this does not work when the first or last element in the environment is a `\hline` command—in that case the environment is aligned at the horizontal rule.

Here is an example:

Tables with no hline commands used versus tables

with some hline commands

used.

```
Tables
\begin{tabular}[t]{l}
 with no\\ hline \\ commands \\ used
\end{tabular} versus tables
\begin{tabular}[t]{|l|}
 \hline
  with some \\ hline \\ commands \\
 \hline
\end{tabular} used.
```

`\firsthline` `\lasthline`

Using `\firsthline` and `\lasthline` will cure the problem, and the tables will align properly as long as their first or last line does not contain extremely large objects.

Tables with no line commands used versus tables

with some line commands

used.

```
Tables
\begin{tabular}[t]{l}
  with no\\ line \\ commands \\ used
\end{tabular} versus tables
\begin{tabular}[t]{|l|}
 \firsthline
  with some \\ line   \\ commands \\
 \lasthline
\end{tabular} used.
```

`\extratabsurround` The implementation of these two commands contains an extra dimension, which is called `\extratabsurround`, to add some additional space at the top and the bottom of such an environment. This is useful if such tables are nested.

2 Final Comments

2.1 Handling of rules

There are two possible approaches to the handling of horizontal and vertical rules in tables:

1. rules can be placed into the available space without enlarging the table, or
2. rules can be placed between columns or rows thereby enlarging the table.

`array.sty` implements the second possibility while the default implementation in the LaTeX kernel implements the first concept. Both concepts have their merrits but one has to be aware of the individual implications.

- With standard LaTeX adding rules to a table will not affect the width or height of the table (unless double rules are used), e.g., changing a preamble from `lll` to `l|l|l` does not affect the document other than adding rules to the table. In contrast, with `array.sty` a table that just fit the `\textwidth` might now produce an overfull box.

- With standard LaTeX modifying the width of rules could result in ugly looking tables because without adjusting the `\tabcolsep`, etc. the space between rule and column could get too small (or too large). In fact even overprinting of text is possible. In contrast, with `array.sty` modifying any such length usually works well as the actual visual white space (from `\tabcolsep`, etc.) does not depend on the width of the rules.

- With standard LaTeX boxed tabulars actually have strange corners because the horizontal rules end in the middle of the vertical ones. This looks very unpleasant when a large `\arrayrulewidth` is chosen. In that case a simple table like

```
\setlength{\arrayrulewidth}{5pt}
\begin{tabular}{|l|}
  \hline  A \\  \hline
\end{tabular}
```

  will produce something like



2.2 Comparisons with older versions of `array.sty`

There are some differences in the way version 2.1 treats incorrect input, even if the source file does not appear to use any of the extra features of the new version.

- A preamble of the form `{wx*{0}{abc}yz}` was treated by versions prior to 2.1 as `{wx}`. Version 2.1 treats it as `{wxyz}`

- An incorrect positional argument such as `[Q]` was treated as `[c]` by `array.sty`, but is now treated as `[t]`.

- A preamble such as `{cc*{2}}` with an error in a $*$-form will generate different errors in the new version. In both cases the error message is not particularly helpful to the casual user.

- Repeated `<` or `>` constructions generated an error in earlier versions, but are now allowed in this package. `>{`⟨*decs1*⟩`}>{`⟨*decs2*⟩`}` is treated the same as `>{`⟨*decs2*⟩⟨*decs1*⟩`}`.

- The `\extracolsep` command does not work with the old versions of `array.sty`, see the comments in `array.bug`. With version 2.1 `\extracolsep` may again be used in `@`-expressions as in standard LaTeX, and also in `!`-expressions (but see the note below).

2.3 Bugs and Features

- Error messages generated when parsing the column specification refer to the preamble argument **after** it has been re-written by the `\newcolumntype` system, not to the preamble entered by the user. This seems inevitable with any system based on pre-processing and so is classed as a **feature**.

- The treatment of multiple `<` or `>` declarations may seem strange at first. Earlier implementations treated `>{`⟨*decs1*⟩`}>{`⟨*decs2*⟩`}` the same as `>{`⟨*decs1*⟩⟨*decs2*⟩`}`. However this did not give the user the opportunity of overriding the settings of a `\newcolumntype` defined using these declarations. For example, suppose in an `array` environment we use a `C` column defined as above. The `C` specifies a centred text column, however `>{\bfseries}C`, which re-writes to

`>{\bfseries}>{$}c<{$}` would not specify a bold column as might be expected, as the preamble would essentially expand to `\hfil$\bfseries$#$ $\hfil` and so the column entry would not be in the scope of the `\bfseries`! The present version switches the order of repeated declarations, and so the above example now produces a preamble of the form `\hfil$ $\bfseries#$ $\hfil`, and the dollars cancel each other out without limiting the scope of the `\bfseries`.

- The use of `\extracolsep` has been subject to the following two restrictions. There must be at most one `\extracolsep` command per `@`, or `!` expression and the command must be directly entered into the `@` expression, not as part of a macro definition. Thus `\newcommand{\ef}{\extracolsep{\fill}}...@{\ef}` does not work with this package. However you can use something like `\newcolumntype{e}{@{\extracolsep{\f` instead.

- As noted by the LaTeX book, for the purpose of `\multicolumn` each column with the exception of the first one consists of the entry and the *following* inter-column material. This means that in a tabular with the preamble `|l|l|l|l|` input such as `\multicolumn{2}{|c|}` in anything other than the first column is incorrect.

  In the standard array/tabular implementation this error is not so noticeable as that version contains negative spacing so that each `|` takes up no horizontal space. But since in this package the vertical lines take up their natural width one sees two lines if two are specified.

References

[1] M. GOOSSENS, F. MITTELBACH and A. SAMARIN. The LaTeX Companion. Addison-Wesley, Reading, Massachusetts, 1994.

[2] D. E. KNUTH. The TeXbook (Computers & Typesetting Volume A). Addison-Wesley, Reading, Massachusetts, 1986.

[3] L. LAMPORT. LaTeX — A Document Preparation System. Addison-Wesley, Reading, Massachusetts, 1986.