

# The `afterpage` package\*

David Carlisle  
carlisle@cs.man.ac.uk

1995/10/27

This package implements a command, `\afterpage`, that causes the commands specified in its argument to be expanded after the current page is output.<sup>1</sup>

1. Sometimes L<sup>A</sup>T<sub>E</sub>X's float positioning mechanism gets overloaded, and all floating `figures` and `tables` drift to the end of the document. One may flush out all the unprocessed floats by issuing a `\clearpage` command, but this has the effect of making the current page end prematurely. Now you can issue `\afterpage{\clearpage}` and the current page will be filled up with text as usual, but then a `\clearpage` command will flush out all the floats before the next text page begins.
2. An earlier mechanism to help with float placement was the optional argument `[H]` (meaning **HERE!**) which was originally added to the standard floating environments by `here.sty`, and is now provided by `float.sty`. However some `[H]` users have commented that they did not really mean 'Here!' They actually wanted 'Somewhere close'. This can now be achieved by `\afterpage{\clearpage\begin{figure}[H] ... \end{figure}}`. This ensures that the figure is at the top of the next page. (The `\clearpage` stops any other figures drifting past the `[H]` figure.)
3. Floating longtables. `longtable.sty` provides the `longtable` environment, a multi-page version of `tabular`. Many `longtable` users have told me that it is difficult to set the text surrounding the long table, and that they wanted a 'floating' version. As, presumably, `longtables` are long, they are probably too large to hold in memory, and float in the way that the `table` environment is floated, however if the table is in a separate file, say `ltfile.tex`, you can now use one of:  
`\afterpage{\clearpage\input{ltfile}}`  
`\afterpage{\clearpage\input{ltfile}\clearpage}`.  
The first form lets text appear on the same page as the end of the longtable, the second ensures that the surrounding text starts again on a new page.

---

\*This file has version number v1.08, last revised 1995/10/27.

<sup>1</sup>This is really a pre-release, to see whether people like the idea of a command like this. This implementation is *not* particularly robust. This implementation does not work in two column mode, and can get 'confused' by L<sup>A</sup>T<sub>E</sub>X's floating environments.

# A new implementation of L<sup>A</sup>T<sub>E</sub>X's `tabular` and `array` environment\*

Frank Mittelbach      David Carlisle<sup>†</sup>

Printed March 10, 2004

## Abstract

This article describes an extended implementation of the L<sup>A</sup>T<sub>E</sub>X `array`- and `tabular`-environments. The special merits of this implementation are further options to format columns and the fact that fragile L<sup>A</sup>T<sub>E</sub>X-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 L<sup>A</sup>T<sub>E</sub>X-code in some aspects and to make L<sup>A</sup>T<sub>E</sub>X 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`1. You do not have to begin every entry of the column with `\bfseries` any more.

---

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

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

Unchanged options	
<code>l</code>	Left adjusted column.
<code>c</code>	Centered adjusted column.
<code>r</code>	Right adjusted column.
<code>p{width}</code>	Equivalent to <code>\parbox[t]{width}</code> .
<code>@{decl.}</code>	Suppresses inter-column space and inserts <code>decl.</code> instead.
New options	
<code>m{width}</code>	Defines a column of width <code>width</code> . Every entry will be centered in proportion to the rest of the line. It is somewhat like <code>\parbox{width}</code> .
<code>b{width}</code>	Coincides with <code>\parbox[b]{width}</code> .
<code>&gt;{decl.}</code>	Can be used before an <code>l</code> , <code>r</code> , <code>c</code> , <code>p</code> , <code>m</code> or a <code>b</code> option. It inserts <code>decl.</code> directly in front of the entry of the column.
<code>&lt;{decl.}</code>	Can be used after an <code>l</code> , <code>r</code> , <code>c</code> , <code>p{..}</code> , <code>m{..}</code> or a <code>b{..}</code> option. It inserts <code>decl.</code> right after the entry of the column.
<code> </code>	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 L <sup>A</sup> T <sub>E</sub> X.
<code>!{decl.}</code>	Can be used anywhere and corresponds with the <code> </code> option. The difference is that <code>decl.</code> is inserted instead of a vertical line, so this option doesn't suppress the normally inserted space between columns in contrast to <code>@{...}</code> .

Table 1: The preamble options.

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

```

{\catcode'\.\active\gdef.\{ \egroup\setbox2\hbox\bgroup}
\def\centerdotes{\catcode'\.\active\setbox0\hbox\bgroup}
\def\endcenterdotes{\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

`\newcolumnstype` 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.<sup>1</sup> Thus we may define

```
\newcolumnstype{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:

```
\newcolumnstype{C}{>{\$}c<{\$}}
\newcolumnstype{L}{>{\$}l<{\$}}
\newcolumnstype{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.

```
\newcolumnstype{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

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

where the appropriate macros in this case are:<sup>2</sup>

```
\def\coldot{.}% Or if you prefer, \def\coldot{\cldot}
{\catcode'\.=\active
```

<sup>1</sup>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.

<sup>2</sup>The package `dcolumn.sty` contains more robust macros based on these ideas.

```

\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{\hfил\egroup\box0\box2}

```

Note that `\newcolumn` 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 `\newcolumn` system takes advantage of the fact that the replacement text in the `\newcolumn` 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:

```

\newcolumn{X}{clr}
\begin{tabular}{X}...

```

The replacement text in a `\newcolumn` 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 `\newcolumn` commands.

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

## 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
tables		<code>\begin{tabular}[t]{1}</code>
<div style="border: 1px solid black; padding: 2px; display: inline-block;">with some hline commands</div>	used.	<code>with no\\ hline \\ commands \\ used</code>
		<code>\end{tabular} versus tables</code>
		<code>\begin{tabular}[t]{ 1 }</code>
		<code>\hline</code>
		<code>with some \\ hline \\ commands \\</code>
		<code>\hline</code>
		<code>\end{tabular} used.</code>

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

Tables with no line commands used	versus	Tables
tables		<code>\begin{tabular}[t]{1}</code>
<div style="border: 1px solid black; padding: 2px; display: inline-block;">with some line commands</div>	used.	<code>with no\\ line \\ commands \\ used</code>
		<code>\end{tabular} versus tables</code>
		<code>\begin{tabular}[t]{ 1 }</code>
		<code>\firstline</code>
		<code>with some \\ line \\ commands \\</code>
		<code>\lastline</code>
		<code>\end{tabular} used.</code>

`\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 merits 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 `l11` to `l|1|1` 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 L<sup>A</sup>T<sub>E</sub>X 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 L<sup>A</sup>T<sub>E</sub>X 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. `>{\langle decs1 \rangle}>{\langle decs2 \rangle}` is treated the same as `>{\langle decs2 \rangle}\langle decs1 \rangle`.
- 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 L<sup>A</sup>T<sub>E</sub>X, 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 `>\langle decs1 \rangle >\langle decs2 \rangle` the same as `>\langle decs1 \rangle \langle decs2 \rangle`. 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{\fill}}}` 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 `|1|1|1|1|` 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.

## 3 The documentation driver file

The first bit of code contains the documentation driver file for  $\TeX$ , i.e., the file that will produce the documentation you are currently reading. It will be extracted from this file by the `docstrip` program.

```
1 \*driver
2 \NeedsTeXFormat{LaTeX2e}[1995/12/01]
3 \documentclass{ltxdoc}
```



# The **bm** package<sup>\*†</sup>

David Carlisle with support by Frank Mittelbach

1999/07/05

## 1 Introduction

This package defines commands to access bold math symbols. The basic command is `\bm` which may be used to make the math expression in its argument be typeset using bold fonts.

The syntax of `\bm` is:

```
\bm{<math expression>}
```

So `\alpha \not= \bm{\alpha}` produces  $\alpha \neq \boldsymbol{\alpha}$ .

`\bm` goes to some trouble to preserve the spacing, so that for instance `\bm<` is a bold `<` but with the correct `\mathrel` spacing that `TeX` gives to `<`. The calculations that `TeX` needs to do for `\bm` can be quite involved and so a definition form is provided.

```
\DeclareBoldMathCommand[<math version>]{<cmd>}{<math expression>}
```

Defines `\cmd` to be the bold form of the math expression. The `<math version>` defaults to ‘bold’ (i.e., `\boldmath`).

For relatively simple expressions, the resulting definitions are very efficient, for instance after:

```
\DeclareBoldMathCommand\balpha{\alpha}
```

`\balpha` is a single ‘mathchardef’ token producing a bold alpha, and so is just as fast to execute as `\alpha`.

The above command is mainly intended for use in packages. For occasional use in `LATeX` documents, and for compatibility with the plain `TeX` support for the mathtime fonts, a ‘user-level’ version, `\bmdefine` is provided that is equivalent to: `\DeclareBoldMathCommand[bold]`.

If there is a ‘heavy’ math version defined (usually accessed by a user-command `\heavymath`) then a similar command `\hm` is defined which access these ‘ultra bold’ fonts. Currently this is probably only useful with the ‘mathtime plus’ font collection. Definitions of commands that use these fonts may be made by specifying the optional argument ‘heavy’ to `\DeclareBoldMathCommand`. Again an abbreviation, `\hmdefine`, is provided, equivalent to:

```
\DeclareBoldMathCommand[heavy].
```

---

<sup>\*</sup>This file has version number v1.0g, last revised 1999/07/05.

<sup>†</sup>Development of this package was commissioned by Y&Y.

The command names (but not the implementation) are taken from Michael Spivak's macros to support the mathtime fonts for plain TeX. In those original macros, the syntax for `\bmdefine` was `\bmdefine\balpha{\bm\alpha}` (with a nested `\bm`). This syntax also works with this package.

## 2 Font allocation

In order to access bold fonts in the simplest and quickest possible manner, the package normally allocates symbol fonts for bold (and possibly heavy) fonts into the 'normal' math version. By default it allocates at most four fonts for `\bm` and at most three fonts for `\hm`. This means that if the mathtime plus font set is being used, seven additional symbol fonts will be used, in addition to the basic four that L<sup>A</sup>T<sub>E</sub>X already declares. The mathtime package also declares an extra symbol font, bringing the total to twelve. The maximum number of symbol *and* math alphabet fonts that can be used in a math version is sixteen. So the above allocation scheme does not leave room for many extra math symbols (such as the AMS symbols) or math alphabets (such as `\mathit`).

Before loading the `bm` package you may define `\bmmax` and `\hmmax` to be suitable values, for instance you may want to set `\newcommand\hmmax{0}` if you will not be using `\hm` much, but you do have a heavy math version defined.

Even if `\bmmax` is set to zero, `\bm` will still access the correct bold fonts (by accessing the fonts via `\boldmath`) but this method is slower, and does not work with delimiters. Delimiters can only be made bold if the bold font has been allocated.

Conversely if you have a non standard font set that makes available extra math delimiters and accents in bold and medium weights you may want to *increase* `\bmmax` so that fonts are allocated for your font set.

## 3 Features

In most cases this package should work in a fairly self explanatory way, but there are some things that might not be obvious.

### 3.1 Interaction with Math Alphabet Commands

As mentioned above, `\bm` goes to some trouble to try to make a command that is just like its argument, but using a bold font. This does not always produce the effect that you might expect.

```
$1 g \bm{g}$
$2 \mathrm{g \bm{g}}$
$3 {g} \bm{{g}}$
$4 \mathrm{{g} \bm{{g}}}$
$5 \mathrm{g} \bm{\mathrm{g}}$
```

produces the following:

*1gg 2gg 3gg 4gg 5gg*

In math mode ‘g’ is effectively a command that produces the letter ‘g’ from the ‘letters’ alphabet, unless a Math Alphabet command is in effect, in which case the ‘g’ comes from the specified alphabet. `\bm{g}` makes an equivalent command, but which defaults to a bold letter alphabet. So in the first example `\bm{g}` is bold math italic, but in the second example the `\mathrm` applies to both `g` and `\bm{g}` in the same way, and so they are both roman.

`\bm` only inspects the ‘top level’ definition of a command, for more complicated expressions, and anything inside a `{ }` group, `\bm` forces bold fonts by essentially the same (slow) technique used by the AMS `\boldsymbol` command (but `\bm` still takes more care of the spacing). So the third example produces identical output to the first (but `TEX` takes more time producing it).

In the fourth example the `\mathrm{\bm{g}}` is essentially equivalent to `\mathrm{\mbox{\boldmath$g$}}`. Currently math alphabet settings are not passed down to ‘nested’ math lists, and so in this example, the `\mathrm` has no effect, and a bold math italic ***g*** is obtained.

Similarly the last example is equivalent to `\mbox{\boldmath$\mathrm{g}$}` and so in this case, one obtains a bold roman **g**.

### 3.2 Delimiters

`TEX` can treat character tokens in two<sup>1</sup> ways. If there is a preceding `\left` or `\right` it can treat them as a delimiter, otherwise it can treat them as a standard character. For example `\left<\right>` produces  $\langle \rangle$ , which is totally different from `<>`, which produces `<>`.

`TEX` can only do this for character tokens. Commands such as `\langle` do not act in this way. This means that `\bm` has to decide whether to treat a character as a delimiter or not. The rule it uses is, it makes a delimiter command for a character if the previous token in the argument was `\left` or `\right`. So `\left\bm{<}` does not work, but `\bm{\left<}` does.

### 3.3 Command Arguments

Normally if a command takes arguments the full command, including any arguments, should be included in `\bm`.

So `\bm{\overbrace{abc}}` (producing  $\overbrace{abc}$ ) not `\bm{\overbrace}{abc}`. If you do not include all the arguments you will typically get the error message:  
Runaway argument?

! Forbidden control sequence found while scanning use of ...

However commands defined in terms of the `TEX` accent and radical primitives *may* be used without their arguments. So `\bm{\hat}{a}` produces  $\hat{a}$ , a bold accent over a non-bold *a* (compare  $\hat{a}$ ) whereas `\bm{\hat{a}}` makes both the *a* and the accent bold,  $\hat{a}$ . Similarly, although the `LATEX` command `\sqrt` must be used with

---

<sup>1</sup>Well more than two really.

its arguments, `\sqrtsign` may be used as in `\bm\sqrtsign{abc}` to produce  $\sqrt{abc}$  rather than  $\sqrt{abc}$  or  $\sqrt{abc}$

If you really need to make a command with arguments use bold fonts without making all of the arguments bold, you can explicitly reset the math version in the argument, eg:

`\sqrt{xyz}`    `\bm{\sqrt{xyz}}`    `\bm{\sqrt{\mbox{\unboldmath$xyz$}}}`  
 $\sqrt{xyz}$                        $\sqrt{xyz}$                        $\sqrt{xyz}$

### 3.4 Bold fonts

This package interrogates the font allocations of the bold and heavy math versions, to determine which bold fonts are available. This means that it is best to load the package *after* any packages that define new symbol fonts, or (like the `mathtime` package) completely change the symbol font allocations.

If no bold font appears to be available for a particular symbol, `\bm` will use ‘poor man’s bold’ that is, overprinting the same character in slightly offset positions to give an appearance of boldness.

In the standard Computer Modern font set, there is no bold ‘large symbols’ font. In the ‘`mathptm`’ and (standard) `mathtime` font sets there are no bold math fonts. In the ‘`mathtime plus`’ font set there are suitable fonts for bold and heavy math setting, and so `\bm` and `\hm` work well. Similarly in the basic Lucida New Math font set there are no bold math fonts, so `\bm` will use ‘poor man’s bold’. However if the Lucida Expert set is used, Then `\bm` will detect, and use the bold math fonts that are available.

As discussed above, one may set `\bmmax` higher or lower than its default value of four to control the font allocation system. Finer control may be gained by explicitly declaring bold symbol fonts. Suppose you have a symbol font ‘`xyz`’ that is available in medium and bold weights, then you would declare this to `LATEX` via:

```
\DeclareSymbolFont{extras} {OMS}{xyz}{m}{n}
\SetSymbolFont{extras}{bold}{OMS}{xyz}{bx}{n}
```

At this point the symbols will be available in the normal math version, and their bold variants in `\boldmath`. If you also declare:

```
\DeclareSymbolFont{boldextras}{OMS}{xyz}{bx}{n}
```

That is, declare a symbol font whose name is formed by prefixing ‘bold’ (or ‘heavy’) to an existing symbol font, then `\bm` (or `\hm`) will use this font directly, rather than accessing the ‘`extras`’ symbol font via `\boldmath`.

### 3.5 Strange failures

In order to get the correct spacing, `\bm` has to ‘investigate’ the definition of the commands in its argument. It is possible that some strange constructions could ‘confuse’ this investigation. If this happens then `LATEX` will almost certainly stop with a strange error. This should not happen with any of the math symbols defined in the base `LATEX` or AMS distributions, or any commands defined in terms of those symbols using normal `LATEX` math constructs. However if some command does fail to work inside `\bm` you should always be able to surround it with an extra set of

braces `\bm{\{ \cmd}}` rather than `\bm{\cmd}`. `\bm` will not then attempt to set the correct spacing, so you may need to set it explicitly, for instance, for a relation, `\bm{\mathrel{\cmd}}`.

### 3.6 AMS package `amsbsy`

The `\bm` command shares some functionality with the `\boldsymbol` command from the AMS L<sup>A</sup>T<sub>E</sub>X collection. To aid in moving documents between these two packages, this package defines `\boldsymbol` and `\heavysymbol` as alternative names for `\bm` and `\hm`.

## 4 Implementation

The commands `\bm` and `\hm` work by defining a number of additional symbol fonts corresponding to the standard ones ‘operators’, ‘letters’, ‘symbols’, and ‘largesymbols’. The names for these symbol fonts are produced by prefixing the usual name with ‘bold’ or ‘heavy’.

For maximum flexibility we get the font definitions by looking in the corresponding math versions, i.e., into `\mv@bold` and if defined into `\mv@heavy`.

```
1 <*package>
```

<code>\bm@table</code>	The table, <code>\bm@table</code> , (which is locally <code>\let</code> to either the bold or heavy version)
<code>\bm@boldtable</code>	defines, for each <i>&lt;math group&gt;</i> ( <i>&lt;fam&gt;</i> ), the ‘offset’ to the bold version of the
<code>\bm@heavytable</code>	specified symbol font. If there is no bold symbol font defined, the offset will be
	set to zero if there is a bold font assigned to this slot in the bold math version, or
	−1 if the font in the bold math version is the same as the one in the normal math
	version. In this case a ‘poor man’s bold’ system of overprinting is used to achieve
	boldness where this is possible.

The settings are made at the time this package is read, and so it is best to load this package late, after any font loading packages have been loaded. Symbol fonts loaded after this package will get the offset of zero, so they will still be made bold by `\bm` as long as an appropriate font is declared for the bold math version.

`\bm@boldtable` and `\bm@heavytable` are set up using very similar code, which is temporarily defined to `\bm`, to save wasting a `csname`. Similarly `\bm@pmb...` (which will be defined later) are used as scratch macros.

The general plan. Run through the fonts allocated to the normal math version. Ignore *<math alphabet>* allocations<sup>2</sup> but for each math symbol font, look in the math version specified by `#1` (bold or heavy). If the font there is different, then allocate a new symbol font in the normal math version to access that bold font and place the numerical difference between the allocations of the bold and normal font into the table being built (`\bm@boldtable`, if `#1` is bold). If the symbol allocation is already greater than `\bmmax` do not allocate a new symbol font, but rather set the offset in the table to zero. `\bm` will detect this, and use `\boldmath` on its

---

<sup>2</sup>For now?

# The `calc` package

## Infix notation arithmetic in $\text{\LaTeX}^*$

Kresten Krab Thorup, Frank Jensen (and Chris Rowley)

1998/07/07

### Abstract

The `calc` package reimplements the  $\text{\LaTeX}$  commands `\setcounter`, `\addtocounter`, `\setlength`, and `\addtolength`. Instead of a simple value, these commands now accept an infix notation expression.

## 1 Introduction

Arithmetic in  $\text{\TeX}$  is done using low-level operations such as `\advance` and `\multiply`. This may be acceptable when developing a macro package, but it is not an acceptable interface for the end-user.

This package introduces proper infix notation arithmetic which is much more familiar to most people. The infix notation is more readable and easier to modify than the alternative: a sequence of assignment and arithmetic instructions. One of the arithmetic instructions (`\divide`) does not even have an equivalent in standard  $\text{\LaTeX}$ .

The infix expressions can be used in arguments to macros (the `calc` package doesn't employ category code changes to achieve its goals)<sup>1</sup>.

## 2 Informal description

Standard  $\text{\LaTeX}$  provides the following set of commands to manipulate counters and lengths [2, pages 194 and 216].

`\setcounter{ctr}{num}` sets the value of the counter *ctr* equal to (the value of) *num*. (Fragile)

`\addtocounter{ctr}{num}` increments the value of the counter *ctr* by (the value of) *num*. (Fragile)

---

\*We thank Frank Mittelbach for his valuable comments and suggestions which have greatly improved this package.

<sup>1</sup>However, it therefore assumes that the category codes of the special characters, such as `(*/)` in its syntax do not change.

`\setlength{cmd}{len}` sets the value of the length command *cmd* equal to (the value of) *len*. (Robust)

`\addtolength{cmd}{len}` sets the value of the length command *cmd* equal to its current value plus (the value of) *len*. (Robust)

(The `\setcounter` and `\addtocounter` commands have global effect, while the `\setlength` and `\addtolength` commands obey the normal scoping rules.) In standard L<sup>A</sup>T<sub>E</sub>X, the arguments to these commands must be simple values. The `calc` package extends these commands to accept infix notation expressions, denoting values of appropriate types. Using the `calc` package, *num* is replaced by  $\langle$ integer expression $\rangle$ , and *len* is replaced by  $\langle$ glue expression $\rangle$ . The formal syntax of  $\langle$ integer expression $\rangle$  and  $\langle$ glue expression $\rangle$  is given below.

In addition to these commands to explicitly set a length, many L<sup>A</sup>T<sub>E</sub>X commands take a length argument. After loading this package, most of these commands will accept a  $\langle$ glue expression $\rangle$ . This includes the optional width argument of `\makebox`, the width argument of `\parbox`, `minipage`, and a `tbluar` p-column, and many similar constructions. (This package does not redefine any of these commands, but they are defined by default to read their arguments by `\setlength` and so automatically benefit from the enhanced `\setlength` command provided by this package.)

In the following, we shall use standard T<sub>E</sub>X terminology. The correspondence between T<sub>E</sub>X and L<sup>A</sup>T<sub>E</sub>X terminology is as follows: L<sup>A</sup>T<sub>E</sub>X counters correspond to T<sub>E</sub>X's count registers; they hold quantities of type  $\langle$ number $\rangle$ . L<sup>A</sup>T<sub>E</sub>X length commands correspond to T<sub>E</sub>X's `dimen` (for rigid lengths) and `skip` (for rubber lengths) registers; they hold quantities of types  $\langle$ dimen $\rangle$  and  $\langle$ glue $\rangle$ , respectively.

T<sub>E</sub>X gives us primitive operations to perform arithmetic on registers as follows:

- addition and subtraction on all types of quantities without restrictions;
- multiplication and division by an *integer* can be performed on a register of any type;
- multiplication by a *real* number (i.e., a number with a fractional part) can be performed on a register of any type, but the stretch and shrink components of a glue quantity are discarded.

The `calc` package uses these T<sub>E</sub>X primitives but provides a more user-friendly notation for expressing the arithmetic.

An expression is formed of numerical quantities (such as explicit constants and L<sup>A</sup>T<sub>E</sub>X counters and length commands) and binary operators (the tokens '+', '-', '\*', and '/' with their usual meaning) using the familiar infix notation; parentheses may be used to override the usual precedences (that multiplication/division have higher precedence than addition/subtraction).

Expressions must be properly typed. This means, e.g., that a `dimen` expression must be a sum of `dimen` terms: i.e., you cannot say '2cm+4' but '2cm+4pt' is valid.

In a `dimen` term, the dimension part must come first; the same holds for glue terms. Also, multiplication and division by non-integer quantities require a special syntax; see below.

Evaluation of subexpressions at the same level of precedence proceeds from left to right. Consider a dimen term such as “`4cm*3*4`”. First, the value of the factor `4cm` is assigned to a dimen register, then this register is multiplied by 3 (using `\multiply`), and, finally, the register is multiplied by 4 (again using `\multiply`). This also explains why the dimension part (i.e., the part with the unit designation) must come first;  $\TeX$  simply doesn’t allow untyped constants to be assigned to a dimen register.

The `calc` package also allows multiplication and division by real numbers. However, a special syntax is required: you must use `\real{⟨decimal constant⟩}`<sup>2</sup> or `\ratio{⟨dimen expression⟩}{⟨dimen expression⟩}` to denote a real value to be used for multiplication/division. The first form has the obvious meaning, and the second form denotes the number obtained by dividing the value of the first expression by the value of the second expression.

A later addition to the package (in June 1998) allows an additional method of specifying a factor of type dimen by setting some text (in LR-mode) and measuring its dimensions: these are denoted as follows.

```
\widthof{⟨text⟩} \heightof{⟨text⟩} \depthof{⟨text⟩}
```

These calculate the natural sizes of the `⟨text⟩` in exactly the same way as is done for the commands `\settoheight` etc. on Page 216 of the manual [2].

Note that there is a small difference in the usage of these two methods of accessing text dimensions. After `\settoheight{⟨text⟩}{Some text}` you can use:

```
\setlength{⟨parskip⟩}{0.68⟨textwd⟩}
```

whereas using the more direct access to the width of the text requires the longer form for multiplication, thus:

```
\setlength{⟨parskip⟩}{\widthof{Some text} * \real{0.68}}
```

$\TeX$  discards the stretch and shrink components of glue when glue is multiplied by a real number. So, for example,

```
\setlength{⟨parskip⟩}{3pt plus 3pt * \real{1.5}}
```

will set the paragraph separation to 4.5pt with no stretch or shrink. (Incidentally, note how spaces can be used to enhance readability.)

When  $\TeX$  performs arithmetic on integers, any fractional part of the results are discarded. For example,

```
\setcounter{x}{7/2}
\setcounter{y}{3*\real{1.6}}
\setcounter{z}{3*\real{1.7}}
```

will assign the value 3 to the counter `x`, the value 4 to `y`, and the value 5 to `z`. This truncation also applies to *intermediate* results in the sequential computation of a composite expression; thus, the following command

```
\setcounter{x}{3 * \real{1.6} * \real{1.7}}
```

---

<sup>2</sup>Actually, instead of `⟨decimal constant⟩`, the more general `⟨optional signs⟩⟨factor⟩` can be used. However, that doesn’t add any extra expressive power to the language of infix expressions.



will assign 6 to  $x$ .

As an example of the use of `\ratio`, consider the problem of scaling a figure to occupy the full width (i.e., `\textwidth`) of the body of a page. Assume that the original dimensions of the figure are given by the `dimen` (length) variables, `\Xsize` and `\Ysize`. The height of the scaled figure can then be expressed by

```
\setlength{\newYsize}{\Ysize*\ratio{\textwidth}{\Xsize}}
```

### 3 Formal syntax

The syntax is described by the following set of rules. Note that the definitions of `<number>`, `<dimen>`, `<glue>`, `<decimal constant>`, and `<plus or minus>` are as in Chapter 24 of The `TEXbook` [1]; and `<text>` is LR-mode material, as in the manual [2]. We use *type* as a meta-variable, standing for ‘integer’, ‘dimen’, and ‘glue’.<sup>3</sup>

```
<type expression>  → <type term>
                   | <type expression><plus or minus><type term>

<type term>        → <type factor>
                   | <type term><multiply or divide><integer factor>
                   | <type term><multiply or divide><real number>

<type factor>      → <type> | <text dimen factor> | (12<type expression>)12

<integer>          → <number>

<text dimen factor> → <text dimen command>{<text>}

<text dimen command> → \widthof | \heightof | \depthof

<multiply or divide> → *12 | /12

<real number>      → \ratio{<dimen expression>}{<dimen expression>}
                   | \real{<decimal constant>}
```

Note that during most of the parsing of `calc` expressions, no expansion happens; thus the above syntax must be explicit<sup>4</sup>.

### 4 The evaluation scheme

In this section, we shall for simplicity consider only expressions containing ‘+’ (addition) and ‘\*’ (multiplication) operators. It is trivial to add subtraction and division.

An expression  $E$  is a sum of terms:  $T_1 + \dots + T_n$ ; a term is a product of factors:  $F_1 * \dots * F_m$ ; a factor is either a simple numeric quantity  $f$  (like `<number>`) as described in the `TEXbook`, or a parenthesized expression ( $E'$ ).

<sup>3</sup>This version of the `calc` package doesn’t support evaluation of `muglue` expressions.

<sup>4</sup>Two exceptions to this are: the first token is expanded one-level (thus the whole expression can be put into a macro); wherever a `<decimal constant>` or `<type>` is expected.

Since the  $\text{\TeX}$  engine can only execute arithmetic operations in a machine-code like manner, we have to find a way to translate the infix notation into this ‘instruction set’.

Our goal is to design a translation scheme that translates  $X$  (an expression, a term, or a factor) into a sequence of  $\text{\TeX}$  instructions that does the following [Invariance Property]: correctly evaluates  $X$ , leaves the result in a global register  $A$  (using a global assignment), and does not perform global assignments to the scratch register  $B$ ; moreover, the code sequence must be balanced with respect to  $\text{\TeX}$  groups. We shall denote the code sequence corresponding to  $X$  by  $\llbracket X \rrbracket$ .

In the replacement code specified below, we use the following conventions:

- $A$  and  $B$  denote registers; all assignments to  $A$  will be global, and all assignments to  $B$  will be local.
- “ $\leftarrow$ ” means global assignment to the register on the lhs.
- “ $\leftarrow$ ” means local assignment to the register on the lhs.
- “ $\hookrightarrow_{[C]}$ ” means “save the code  $C$  until the current group (scope) ends, then execute it.” This corresponds to the  $\text{\TeX}$ -primitive `\aftergroup`.
- “{” denotes the start of a new group, and “}” denotes the end of a group.

Let us consider an expression  $T_1 + T_2 + \dots + T_n$ . Assuming that  $\llbracket T_k \rrbracket$  ( $1 \leq k \leq n$ ) attains the stated goal, the following code clearly attains the stated goal for their sum:

$$\begin{aligned} \llbracket T_1 + T_2 + \dots + T_n \rrbracket \implies & \{ \llbracket T_1 \rrbracket \} B \leftarrow A \quad \{ \llbracket T_2 \rrbracket \} B \leftarrow B + A \\ & \dots \quad \{ \llbracket T_n \rrbracket \} B \leftarrow B + A \quad A \leftarrow B \end{aligned}$$

Note the extra level of grouping enclosing each of  $\llbracket T_1 \rrbracket$ ,  $\llbracket T_2 \rrbracket$ ,  $\dots$ ,  $\llbracket T_n \rrbracket$ . This will ensure that register  $B$ , used to compute the sum of the terms, is not clobbered by the intermediate computations of the individual terms. Actually, the group enclosing  $\llbracket T_1 \rrbracket$  is unnecessary, but it turns out to be simpler if all terms are treated the same way.

The code sequence “ $\{ \llbracket T_2 \rrbracket \} B \leftarrow B + A$ ” can be translated into the following equivalent code sequence: “ $\{ \hookrightarrow_{[B \leftarrow B + A]} \llbracket T_2 \rrbracket \}$ ”. This observation turns out to be the key to the implementation: The “ $\hookrightarrow_{[B \leftarrow B + A]}$ ” is generated *before*  $T_2$  is translated, at the same time as the ‘+’ operator between  $T_1$  and  $T_2$  is seen.

Now, the specification of the translation scheme is straightforward:

$$\begin{aligned} \llbracket f \rrbracket & \implies A \leftarrow f \\ \llbracket (E') \rrbracket & \implies \llbracket E' \rrbracket \\ \llbracket T_1 + T_2 + \dots + T_n \rrbracket & \implies \{ \hookrightarrow_{[B \leftarrow A]} \llbracket T_1 \rrbracket \} \quad \{ \hookrightarrow_{[B \leftarrow B + A]} \llbracket T_2 \rrbracket \} \\ & \dots \quad \{ \hookrightarrow_{[B \leftarrow B + A]} \llbracket T_n \rrbracket \} \quad A \leftarrow B \\ \llbracket F_1 * F_2 * \dots * F_m \rrbracket & \implies \{ \hookrightarrow_{[B \leftarrow A]} \llbracket F_1 \rrbracket \} \quad \{ \hookrightarrow_{[B \leftarrow B * A]} \llbracket F_2 \rrbracket \} \\ & \dots \quad \{ \hookrightarrow_{[B \leftarrow B * A]} \llbracket F_m \rrbracket \} \quad A \leftarrow B \end{aligned}$$

By structural induction, it is easily seen that the stated property is attained.

By inspection of this translation scheme, we see that we have to generate the following code:

- we must generate “ $\{\hookrightarrow_{[B\leftarrow A]}\{\hookrightarrow_{[B\leftarrow A]}\}$ ” at the left border of an expression (i.e., for each left parenthesis and the implicit left parenthesis at the beginning of the whole expression);
- we must generate “ $\}A \Leftarrow B\}A \Leftarrow B$ ” at the right border of an expression (i.e., each right parenthesis and the implicit right parenthesis at the end of the full expression);
- ‘\*’ is replaced by “ $\}\{\hookrightarrow_{[B\leftarrow B*A]}\}$ ”;
- ‘+’ is replaced by “ $\}A \Leftarrow B\}\{\hookrightarrow_{[B\leftarrow B+A]}\{\hookrightarrow_{[B\leftarrow A]}\}$ ”;
- when we see (expect) a numeric quantity, we insert the assignment code “ $A \Leftarrow$ ” in front of the quantity and let  $\text{\TeX}$  parse it.

## 5 Implementation

For brevity define

$$\langle \text{numeric} \rangle \longrightarrow \langle \text{number} \rangle \mid \langle \text{dimen} \rangle \mid \langle \text{glue} \rangle \mid \langle \text{muglue} \rangle$$

So far we have ignored the question of how to determine the type of register to be used in the code. However, it is easy to see that (1) ‘\*’ always initiates an  $\langle \text{integer factor} \rangle$ , (2) all  $\langle \text{numeric} \rangle$ s in an expression, except those which are part of an  $\langle \text{integer factor} \rangle$ , are of the same type as the whole expression, and all  $\langle \text{numeric} \rangle$ s in an  $\langle \text{integer factor} \rangle$  are  $\langle \text{number} \rangle$ s.

We have to ensure that  $A$  and  $B$  always have an appropriate type for the  $\langle \text{numeric} \rangle$ s they manipulate. We can achieve this by having an instance of  $A$  and  $B$  for each type. Initially,  $A$  and  $B$  refer to registers of the proper type for the whole expression. When an  $\langle \text{integer factor} \rangle$  is expected, we must change  $A$  and  $B$  to refer to integer type registers. We can accomplish this by including instructions to change the type of  $A$  and  $B$  to integer type as part of the replacement code for ‘\*’; if we append such instructions to the replacement code described above, we also ensure that the type-change is local (provided that the type-changing instructions only have local effect). However, note that the instance of  $A$  referred to in  $\hookrightarrow_{[B\leftarrow B*A]}$  is the integer instance of  $A$ .

We shall use  $\backslash \text{begingroup}$  and  $\backslash \text{endgroup}$  for the open-group and close-group characters. This avoids problems with spacing in math (as pointed out to us by Frank Mittelbach).

# The dcolumn package\*

David Carlisle

2001/05/28

## Abstract

This package defines a system for defining columns of entries in an `array` or `tabular` which are to be aligned on a ‘decimal point’.

This package defines `D` to be a column specifier with three arguments.

`D{sep.tex}{sep.dvi}{decimal places}`

`<sep.tex>` should be a single character, this is used as the separator in the `.tex` file. Thus it will usually be ‘.’ or ‘,’.

`<sep.dvi>` is used as the separator in the output, this may be the same as the first argument, but may be any math-mode expression, such as `\cdot`. It should be noted that `dcolumn` always uses math mode for the digits as well as the separator.

`<decimal places>` should be the maximum number of decimal places in the column. If this is negative, any number of decimal places can be used in the column, and all entries will be centred on (the leading edge of) the separator. Note that this can cause a column to be too wide, compare the first two columns in the example below. If this argument is positive, the column uses macros equivalent to `\rightdots \endrightdots` of `array.sty`, otherwise the macros are essentially equivalent to `\centerdots \endcenterdots`.

You may not want to use all three entries in the `array` or `tabular` preamble, so you may define your own preamble specifiers using `\newcolumnmtype`.

For example we may say:

```
\newcolumnmtype{d}[1]{D{.}{\cdot}{#1}}
```

`d` takes a single argument specifying the number of decimal places, and the `.tex` file should use `.`, with `·` being used in the output.

```
\newcolumnmtype{.}{D{.}{.}{-1}}
```

`.` specifies a column of entries to be centred on the `..`

```
\newcolumnmtype{,}{D{,}{,}{2}}
```

`,` specifies takes a column of entries with at most two decimal places after a `.`,

The following table begins `\begin{tabular}{|d{-1}|d{2}|.|,|}`

---

\*This file has version number v1.06, last revised 2001/05/28.

1.2	1.2	1.2	1.2
1.23	1.23	12.5	300,2
1121.2	1121.2	861.20	674,29
184	184	10	69
.4	.4		,4
		.4	

Note that the first column, which had a negative  $\langle decimal\ places \rangle$  argument is wider than the second column, so that the decimal point appears in the middle of the column. Also note that this package deals correctly with entries with no decimal part, no integer part, and blank entries.

If you have table headings (inserted with  $\backslash multicolumn\{1\}\{c\}\{.\}$  to override the D column type) then it may be that neither of the above ‘centred’ or ‘right aligned’ forms is quite what you want.

head	head	head	wide heading	wide heading	wide heading
1.2	1.2	1.2	1.2	1.2	1.2
11212.2	11212.2	11212.2	.4	.4	.4
.4	.4	.4			

In both of these tables the first column is set with  $D\{.\}\{-1\}$  to produce a column centered on the  $.$ , and the second column is set with  $D\{.\}\{1\}$  to produce a right aligned column.

The centered column produces columns that are wider than necessary to fit in the numbers under a heading as it has to ensure that the decimal point is centred. The right aligned column two does not have this drawback, but under a wide heading a column of small right aligned figures looks a bit odd.

In version v1.03 a third possibility is introduced. The third  $\langle decimal\ places \rangle$  argument may specify *both* the number of digits to the left and to the right of the decimal place. The third column in the first table above is set with  $D\{.\}\{5.1\}$  and in the second table,  $D\{.\}\{1.1\}$ , to specify ‘five places to the left and one to the right’ and ‘one place to the left and one to the right’ respectively. (You may use ‘,’ or other tokens, not necessarily ‘.’ in this argument.) The column of figures is then positioned such that a number with the specified numbers of digits is centred in the column.

This notation also enables columns that are centred on the mid-point of the separator, rather than its leading edge; for example  $D\{+\}\{\backslash, \backslash pm\}, \{3, 3\}$  will give nice, symmetric layout of up to three digits on either side of a  $\pm$  sign.

## 1 The Macros

1  $\langle *package \rangle$

First we load `array.sty` if it not already loaded.

2  $\backslash RequirePackage\{array\}$

The basic ideas behind these macros are explained in the documentation for `array.sty`. However they use three tricks which may be useful in other contexts.

# The `delarray` package\*

David Carlisle  
carlisle@cs.man.ac.uk

1994/03/14

## 1 Examples

The addition to `array.sty` added in `delarray.sty` is a system of implicit `\left` `\right` pairs. If you want an array surrounded by parentheses, you can enter:  
`\begin{array}{cc} ...`

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix}$$

Similarly if an environment equivalent to PLAIN `TeX`'s `\cases` could be defined by:

`\begin{array}\{\{1L}\} ...`

$$f(x) = \begin{cases} 0 & \text{if } x = 0 \\ \sin(x)/x & \text{otherwise} \end{cases}$$

Here `L` is supposed to denote a column of left aligned L-R text. It may be defined via: `\newcolumnntype{L}{>{\$}1<{\$}}`, as discussed in `array.sty`. Note that as the delimiters must always be used in pairs, the `'.` must be used to denote a 'null delimiter'.

This feature is especially useful if the `[t]` or `[b]` arguments are also used. In these cases the result is not equivalent to surrounding the environment by `\left... \right`, as can be seen from the following example:

$$\begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix} \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix} \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix} \quad \text{not} \quad \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix} \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix} \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}$$

```
\begin{array}[t]{c} 1\2\3 \end{array}
\begin{array}[c]{c} 1\2\3 \end{array}
\begin{array}[b]{c} 1\2\3 \end{array}
```

---

\*This file has version number v1.01, last revised 1994/03/14.

```

\quad\mbox{not}\quad
\left(\begin{array}[t]{c} 1\2\3 \end{array}\right)
\left(\begin{array}[c]{c} 1\2\3 \end{array}\right)
\left(\begin{array}[b]{c} 1\2\3 \end{array}\right)

```

## 2 The Macros

```

1 <*package>
2 \RequirePackage{array}[1994/02/03]
\@tabarray This macro tests for an optional bracket and then calls up \@array or
\@array[c] (as default).
3 \def\@tabarray{\ifnextchar[{\@array}{\@array[c]}}

\@array This macro tests for an optional delimiter before the left brace of the main pream-
ble argument. If there is no delimiter, \@arrayleft and \@arrayright are made
a no-ops, and \@array is called with the positional argument. Otherwise call
\@delarray.
4 \def\@array[#1]{\ifnextchar\bgroup
5   {\let\@arrayleft\relax\let\@arrayright\relax\@array[#1]}%
6   {\@delarray[#1]}}

\@delarray We now know that we have an array (or tabular) with delimiters.
7 \def\@delarray[#1]#2#3#4{%
The following line is completely redundant but it does catch errors involving del-
imiters before the processing of the alignment begins. A common error is likely
to be omitting the ‘.’ in a \cases-type construction. This causes the first token
of the alignment to be gobbled, possibly causing lots of spurious errors before the
cause of the error, the missing delimiter, is discovered as \@arrayright puts the
alignment and the delimiters together.
8   \setbox\z@\hbox{${\left#2\right#4}$}%
In the case of a ‘c’ argument we do not need to rebox the alignment, so we can
define \@arrayleft and \@arrayright just to insert the delimiters.
9   \if#1c\def\@arrayleft{\left#2}\def\@arrayright{\right#4}%
Otherwise we (should) have a [t] or [b] argument, so first we store the alignment,
without delimiters in box0.
10  \else\def\@arrayleft{\setbox\z@}%
Then after the alignment is finished:
11  \def\@arrayright{%
Calculate the amount the box needs to be lowered (this will be negative in the
case of [b]). A little bit of arithmetic cf. the TeXBook, Appendix G, rule 8. We
calculate the amount this way, rather than just taking the difference between the
depth of box0 and the depth of the box defined below, as the depth of that box

```

# The `enumerate` package\*

David Carlisle

1999/03/05

## Abstract

This package gives the `enumerate` environment an optional argument which determines the style in which the counter is printed.

An occurrence of one of the tokens `A a I i` or `1` produces the value of the counter printed with (respectively) `\Alph \alph \Roman \roman` or `\arabic`.

These letters may be surrounded by any strings involving any other `TEX` expressions, however the tokens `A a I i 1` must be inside a `{ }` group if they are not to be taken as special.

## 1 Examples

EX i. one one one one one one one one one one one one one one	<pre>\begin{enumerate}[EX i.] \item one one one one one one one       one one one one\label{LA} \item two   \begin{enumerate}[\example a)] \item one of two one of two       one of two\label{LB} \item two of two \end{enumerate} \item two of two \end{enumerate}</pre>
EX ii. two  example a) one of two one of two one of two  example b) two of two	<pre>\begin{enumerate}[\example a)] \item one\label{LC} \item two \end{enumerate}</pre>
A-1 one	<pre>\begin{enumerate}[\example a)] \item one\label{LC} \item two \end{enumerate}</pre>
A-2 two	

`\label` and `\ref` may be used as with the standard `enumerate` environment. `\ref` only produces the counter value, not the whole label. `\ref` prints the value

---

\*This file has version number v3.00, last revised 1999/03/05.



in the same style as `\item`, as determined by the presence of one of the tokens `A a I i 1` in the optional argument. In the above example `\ref{LA}`, `\ref{LB}` and `\ref{LC}` produce ‘i’, ‘iia’ and ‘1’ respectively.

## 2 Macros

```

1 <*package>

\@enlab Internal token register used to build up the label command from the optional
argument.
2 \newtoks\@enLab

\@enQmark This just expands to a ‘?’. \ref will produce this, if no counter is printed.
3 \def\@enQmark{?}

The next four macros build up the command that will print the item label.
They each gobble one token or group from the optional argument, and add corre-
sponding tokens to the register \@enLab. They each end with a call to \@enloop,
which starts the processing of the next token.

\@enLabel Add the counter to the label. #2 will be one of the ‘special’ tokens A a I i 1,
and is thrown away. #1 will be a command like \Roman.
4 \def\@enLabel#1#2{%
5   \edef\@enthe{\noexpand#1{\@enumctr}}%
6   \@enLab\expandafter{\the\@enLab\csname the\@enumctr\endcsname}%
7   \@enloop}

\@enSpace Add a space to the label. The tricky bit is to gobble the space token, as you can
\@enSpace not do this with a macro argument.
8 \def\@enSpace{\afterassignment\@enSpace\let\@tempa= }
9 \def\@enSpace{\@enLab\expandafter{\the\@enLab\space}\@enloop}

\@enGroup Add a { } group to the label.
10 \def\@enGroup#1{\@enLab\expandafter{\the\@enLab{#1}}\@enloop}

\@enOther Add anything else to the label
11 \def\@enOther#1{\@enLab\expandafter{\the\@enLab#1}\@enloop}

\@enloop The body of the main loop. Eating tokens this way instead of using \@tfor lets
\@enloop@ you see spaces and all braces. \@tfor would treat a and {a} as special, but not
{{a}}.
12 \def\@enloop{\futurelet\@entemp\@enloop@}
13 \def\@enloop@{%
14   \ifx A\@entemp           \def\@tempa{\@enLab\Alph } \else
15   \ifx a\@entemp           \def\@tempa{\@enLab\alph } \else
16   \ifx i\@entemp           \def\@tempa{\@enLab\roman } \else
17   \ifx I\@entemp           \def\@tempa{\@enLab\Roman } \else

```

# File not found error\*

Frank Mittelbach

March 10, 2004

## 1 Introduction

When  $\LaTeX 2_{\epsilon}$  is unable to find a file it will ask for an alternative file name. However, sometimes the problem is only noticed by  $\TeX$ , and in that case  $\TeX$  insists on getting a valid file name; any other attempt to leave this error loop will fail.<sup>1</sup> Many users try to respond in the same way as to normal error messages, e.g. by typing *<return>*, or **s** or **x**, but  $\TeX$  will interpret this as a file name and will ask again.

To provide a graceful exit out of this loop, we define a number of files which emulate the normal behavior of  $\TeX$  in the error loop as far as possible.

After installing these files the user can respond with **h**, **q**, **r**, **s**, **e**, **x**, and on some systems also with *<return>* to  $\TeX$ 's missing file name question.

## 2 The documentation driver

This code will generate the documentation. Since it is the first piece of code in the file, the documentation can be obtained by simply processing this file with  $\LaTeX 2_{\epsilon}$ .

```
1 <*driver>
2 \documentclass{ltxdoc}
3 \begin{document} \DocInput{fileerr.dtx} \end{document}
4 </driver>
```

## 3 The files

### 3.1 Asking for help with h

When the user types **h** in the file error loop  $\TeX$  will look for the file **h.tex**. In this file we put a message informing the user about the situation (we use  $\text{\^{\^}J}$  to start

---

\*This file has version v1.0e last revised 1997/07/07

<sup>1</sup>On some systems,  $\TeX$  accepts a special character denoting the end of file to return from this loop, e.g. Control-D on UNIX or Control-Z on DOS.

new lines in the message) and then finish with a normal `\errmessage` command thereby bringing up `TeX`'s normal error mechanism.

```
5 <*help>
6 \newlinechar='\^^J
7 \message{!The file name provided could not be found.^^J%
8 Use '<enter>' to continue processing,^^J%
9 'S' to scroll future errors^^J%
10 'R' to run without stopping,^^J%
11 'Q' to run quietly,^^J%
12 or 'X' to terminate TeX}
13 \errmessage{}
14 </help>
```

### 3.2 Scrolling this and further errors with s

For the response `s` we put a message into the file `s.tex` and start `\scrollmode` to scroll further error messages in this run. On systems that allow `.tex` as a file name we can also trap a single *<return>* from the user.

```
15 <+scroll | return | run, batch> \message{File ignored}
16 <+scroll> \scrollmode
17 <+run> \nonstopmode
18 <+batch> \batchmode
```

### 3.3 Exiting the run with x or e

If the user enters `x` or `e` to stop `TeX`, we need to put something into the corresponding file which will force `TeX` to give up. We achieve this by turning off terminal output and then asking `TeX` to stop: first by using the internal `LATeX` name `\@@end`, and if that doesn't work because something other than `LATeX` is used, by trying the `TeX` primitive `\end`.

```
19 <+edit | exit> \batchmode \csname @@end\endcsname \end
```

We end every file with an explicit `\endinput` which prevents the `docstrip` program from putting the character table into the generated files.

```
20 \endinput
```

# A font sampler

Alan Jeffrey

v0.11

## 1 Introduction

This document describes the font test document `fontsmpl.tex` and the accompanying package `fontsmpl.sty`. This produces a test of a font family, printing a sample text, a table of accents, and a sample of commands such as `\pounds`.

It can be used in two ways. The `fontsmpl` package provides the command `\fontsample`, which produces a sample of the current font.

The `fontsmpl` document prompts interactively for a font family (for example ‘`cmr`’) and produces a sample of that family.

## 2 Documentation

This docstrip document has three docstrip options:

- `document` the code for `fontsmpl.tex`.
- `package` the code for `fontsmpl.sty`.
- `driver` this documentation.

The code for the driver is:

```
1 <*driver>
2 \NeedsTeXFormat{LaTeX2e}
3 \documentclass{ltxdoc}
4 \begin{document}
5   \DocInput{fontsmpl.dtx}
6 \end{document}
7 </driver>
```

## 3 Font sample document

The sample document prompts for a family, and uses the `fontsmpl` package. If there is a file `fontsmpl.cfg`, this is loaded.

```
8 <*document>
```

```

 9 \NeedsTeXFormat{LaTeX2e}
10 \documentclass{article}
11 \usepackage{fontsmpl}
12 \makeatletter
13 \InputIfFileExists{fontsmpl.cfg}{\}{}
14 \makeatother
15 \typein[\family]{Please enter a family name (for example 'cmr').}
16 \title{Test of \LaTeX{} font family '\family'}
17 \author{Font sample produced with 'fontsmpl'}
18 \raggedright
19 \begin{document}
20 \maketitle
21 \fontfamily{\family}\selectfont
22 \fontencoding{T1}\selectfont\fontsample
23 \fontencoding{OT1}\selectfont\fontsample
24 \itshape
25 \fontencoding{T1}\selectfont\fontsample
26 \fontencoding{OT1}\selectfont\fontsample
27 \slshape
28 \fontencoding{T1}\selectfont\fontsample
29 \fontencoding{OT1}\selectfont\fontsample
30 \scshape
31 \fontencoding{T1}\selectfont\fontsample
32 \fontencoding{OT1}\selectfont\fontsample
33 \upshape\bfseries
34 \fontencoding{T1}\selectfont\fontsample
35 \fontencoding{OT1}\selectfont\fontsample
36 \itshape
37 \fontencoding{T1}\selectfont\fontsample
38 \fontencoding{OT1}\selectfont\fontsample
39 \slshape
40 \fontencoding{T1}\selectfont\fontsample
41 \fontencoding{OT1}\selectfont\fontsample
42 \scshape
43 \fontencoding{T1}\selectfont\fontsample
44 \fontencoding{OT1}\selectfont\fontsample
45 \end{document}
46 \end{document}

```

## 4 Font sample package

The fontsmpl package is a  $\LaTeX 2_{\epsilon}$  package.

```

47 \*package
48 \NeedsTeXFormat{LaTeX2e}
49 \ProvidesPackage{fontsmpl}[1994/10/29 Font sample package]

```

`\fontsample` The `\fontsample` command prints out a sample text, a selection of glyphs, and a table of accents.

```

50 \newcommand{\fontsample}{%
51   Test of font \f@encoding/\f@family/\f@series/\f@shape.
52   \fontsampletext
53   \fontsampleglyphs
54   \fontsampleaccents
55 }

```

`\fontsampletext` A sample text, taken from Knuth's `testfont.tex`.

```

56 \newcommand{\fontsampletext}{%
57   Some text:
58   \begin{quote}\begin{flushleft}
59     On November 14, 1885, Senator \& Mrs.~Leland Stanford called
60     together at their San Francisco mansion the 24~prominent men who
61     had been chosen as the first trustees of The Leland Stanford
62     Junior University. They handed to the board the Founding Grant
63     of the University, which they had executed three days before.
64     This document---with various amendments, legislative acts, and
65     court decrees---remains as the University's charter. In bold,
66     sweeping language it stipulates that the objectives of the
67     University are ‘‘to qualify students for personal success and
68     direct usefulness in life; and to promote the publick welfare by
69     exercising an influence in behalf of humanity and civilization,
70     teaching the blessings of liberty regulated by law, and
71     inculcating love and reverence for the great principles of
72     government as derived from the inalienable rights of man to life,
73     liberty, and the pursuit of happiness.’’
74   \\
75   (!‘THE DAZED BROWN FOX QUICKLY GAVE 12345--67890 JUMPS!’)
76   \\
77   ?‘But aren’t Kafka’s Schlo\ss\
78   and \AE sop’s \OE uvres often na\"i ve vis-\‘a-vis the
79   d\ae monic ph\oe nix’s official r\^ole in fluffy s\t ouffl\‘es?
80   \\
81
82   \end{flushleft}\end{quote}
83 }

```

`\fontsampleglyphs` A list of sample glyph commands.

```

\fontsampleglyph 84 \newcommand{\fontsampleglyphs}{%
85   Some glyphs:
86   \begin{quote}\begin{flushleft}
87     \fontsampleglyph{\#}
88     \fontsampleglyph{\$}
89     \fontsampleglyph{\%}
90     \fontsampleglyph{\&}
91     \fontsampleglyph{\AA}
92     \fontsampleglyph{\AE}
93     \fontsampleglyph{\DH}
94     \fontsampleglyph{\DJ}
95     \fontsampleglyph{\L}

```

96 \fontsampleglyph{\NG}  
 97 \fontsampleglyph{\OE}  
 98 \fontsampleglyph{\O}  
 99 \fontsampleglyph{\P}  
 100 \fontsampleglyph{\SS}  
 101 \fontsampleglyph{\S}  
 102 \fontsampleglyph{\TH}  
 103 \fontsampleglyph{\\_}  
 104 \fontsampleglyph{\aa}  
 105 \fontsampleglyph{\ae}  
 106 \fontsampleglyph{\copyright}  
 107 \fontsampleglyph{\dag}  
 108 \fontsampleglyph{\ddag}  
 109 \fontsampleglyph{\dh}  
 110 \fontsampleglyph{\dj}  
 111 \fontsampleglyph{\dots}  
 112 \fontsampleglyph{\guillemotleft}  
 113 \fontsampleglyph{\guillemotright}  
 114 \fontsampleglyph{\guilsinglleft}  
 115 \fontsampleglyph{\guilsinglright}  
 116 \fontsampleglyph{\i}  
 117 \fontsampleglyph{\j}  
 118 \fontsampleglyph{\l}  
 119 \fontsampleglyph{\ng}  
 120 \fontsampleglyph{\oe}  
 121 \fontsampleglyph{\o}  
 122 \fontsampleglyph{\pounds}  
 123 \fontsampleglyph{\quotedblbase}  
 124 \fontsampleglyph{\quotesinglbase}  
 125 \fontsampleglyph{\ss}  
 126 \fontsampleglyph{\textasciicircum}  
 127 \fontsampleglyph{\textasciitilde}  
 128 \fontsampleglyph{\textbackslash}  
 129 \fontsampleglyph{\textbar}  
 130 \fontsampleglyph{\textbullet}  
 131 \fontsampleglyph{\textcompwordmark}  
 132 \fontsampleglyph{\textendash}  
 133 \fontsampleglyph{\textendash}  
 134 \fontsampleglyph{\textexclamdown}  
 135 \fontsampleglyph{\textgreater}  
 136 \fontsampleglyph{\texthyphenchar}  
 137 \fontsampleglyph{\textless}  
 138 \fontsampleglyph{\textperiodcentered}  
 139 \fontsampleglyph{\textquestiondown}  
 140 \fontsampleglyph{\textquotedblleft}  
 141 \fontsampleglyph{\textquotedblright}  
 142 \fontsampleglyph{\textquotedbl}  
 143 \fontsampleglyph{\textquoteleft}  
 144 \fontsampleglyph{\textquoteright}  
 145 \fontsampleglyph{\textvisiblespace}

```

146     \fontsampleglyph{\th}
147     \fontsampleglyph{\{ }
148     \fontsampleglyph{\} }
149     \end{flushleft}\end{quote}
150 }
151 \newcommand{\fontsampleglyph}[1]{%
152   \ifx#1\@undefined
153     {\typewriterfont\string#1}~is~undefined
154   \else
155     {\typewriterfont\string#1}~is~'#1'
156   \fi
157 }

```

`\fontsampleaccents` A sample of accents.

```

\fontsampleaccent 158 \newcommand{\fontsampleaccents}{%
159   Some accents:
160   \begin{quote}\begin{flushleft}
161     \fontsampleaccent{"} \\
162     \fontsampleaccent{' } \\
163     \fontsampleaccent{. } \\
164     \fontsampleaccent{=} \\
165     \fontsampleaccent{H} \\
166     \fontsampleaccent{^} \\
167     \fontsampleaccent{' } \\
168     \fontsampleaccent{b} \\
169     \fontsampleaccent{c} \\
170     \fontsampleaccent{d} \\
171     \fontsampleaccent{k} \\
172     \fontsampleaccent{u} \\
173     \fontsampleaccent{v} \\
174     \fontsampleaccent{~}
175   \end{flushleft}\end{quote}
176 }
177 \newcommand{\fontsampleaccent}[1]{%
178   \makebox[1em][r]{\typewriterfont\string#1}
179   \makebox[15em][l]{%
180     #1A#1C#1D#1E#1G#1I#1L#1N%
181     #1O#1R#1S#1T#1U#1Y#1Z%
182     #1a#1c#1d#1e#1g#1i#1l#1n%
183     #1o#1r#1s#1t#1u#1y#1z%
184   }
185 }

```

`\typewriterfont` Not all sites have the T1 typewriter fonts, so we set the typewriter font to be a fixed font.

```

186 \DeclareFixedFont{\typewriterfont}
187   {\encodingdefault}{\ttdefault}{\mddefault}{\updefault}{10}

```

`\TextSymbolUnavailable` Switch off the error message from missing glyphs.



```
188 \def\TextSymbolUnavailable#1{%  
189   \textbf{?}\PackageInfo{fontsmpl}{%  
190     Command \protect#1 undefined in encoding \f@encoding%  
191   }%  
192 }  
193 \end{package}
```

# Footnotes in a multi-column layout\*

Frank Mittelbach  
Electronic Data Systems (Deutschland) GmbH  
Eisenstraße 56 N15  
D-6090 Rüsselsheim

March 10, 2004

## 1 Preface to version 1.1

The new release is a basically unchanged version of the original. I upgraded the macros so that they work with  $\LaTeX 2_\epsilon$  and used some of the additional flexibility introduced therein. For example, the command `\preparefootins` is now automatically called at `\begin{document}`, thus allowing the user to adjust the `\textheight` in the preamble.

It is not surprisingly that I was forced to change some of the macros because they dig deep into  $\LaTeX$ 's output routines. Fortunately this is something normally not necessary when upgrading other  $\LaTeX 2.09$  styles to  $\LaTeX 2_\epsilon$  packages.

I also upgraded the documentation to conform to the  $\LaTeX 2_\epsilon$  terminology, e.g., this is a package since document classes will not know about it. However it is very likely that I have missed some necessary corrections.

## 2 Introduction

The placement of footnotes in a multi-column layout always bothered me. The approach taken by  $\LaTeX$  (i.e., placing the footnotes separately under each column) might be all right if nearly no footnotes are present. But it looks clumsy when both columns contain footnotes, especially when they occupy different amounts of space.

In the multi-column package [5], I used page-wide footnotes at the bottom of the page, but again the

result doesn't look very pleasant since short footnotes produce undesired gaps of white space. Of course, the main goal of this package was a balancing algorithm for columns which would allow switching between different numbers of columns on the same page. With this feature, the natural place for footnotes seems to be the bottom of the page<sup>1</sup> but looking at some of the results it seems best to avoid footnotes in such a layout entirely.

Another possibility is to turn footnotes into endnotes, i.e., printing them at the end of every chapter or the end of the entire document. But I assume everyone who has ever read a book using such a layout will agree with me, that it is a pain to search back and forth, so that the reader is tempted to ignore the endnotes entirely.

When I wrote the article about "Future extensions of  $\TeX$ " [6] I was again dissatisfied with the outcome of the footnotes, and since this article should show certain aspects of high quality typesetting, I decided to give the footnote problem a try and modified the  $\LaTeX$  output routine for this purpose. The layout I used was inspired by the yearbook of the Gutenberg Gesellschaft Mainz [2]. Later on, I found that it is also recommended by Jan White [9]. On the layout of footnotes I also consulted books by Jan Tschichold [8] and Manfred Simoneit [7], books, I would

\*The  $\LaTeX$  package `ftnright.sty` which is described in this article has the version number `v1.1e` dated 2000/04/14.

1. You can not use column footnotes at the bottom, since the number of columns can differ on one page.

recommend to everyone being able to read German texts.

## 2.1 Description of the new layout

The result of this effort is presented in this paper and the reader can judge for himself whether it was successful or not.<sup>2</sup> The main idea for this layout is to assemble the footnotes of all columns on a page and place them all together at the bottom of the right column. Allowing for enough space between footnotes and text, and in addition, setting the footnotes in smaller type<sup>3</sup> I decided that one could omit the footnote separator rule which is used in most publications prepared with  $\text{\TeX}$ .<sup>4</sup> Furthermore, I decided to place the footnote markers<sup>5</sup> at the baseline instead of raising them as superscripts.<sup>6</sup>

All in all, I think this generates a neat layout, and surprisingly enough, the necessary changes to the  $\text{\LaTeX}$  output routine are nevertheless astonishingly simple.

## 2.2 The use of the package

This package might be used together with any other package for  $\text{\LaTeX}$  which does not change the three internals changed by `ftnright.sty`.<sup>7</sup> In most cases, it is best to use this package as the very last package in the preamble to make sure that its settings are not overwritten by other packages.

It is unfortunate that the current  $\text{\LaTeX}$  has nearly no provisions to make such changes without overwriting the internal routines. In the  $\text{\LaTeX}3$  implementation, we will certainly add some hooks that will make such changes more easy.

## 3 The documentation driver

The first bit of code contains the documentation driver file for  $\text{\TeX}$ , i.e., the file that will produce the documentation you are currently reading. It will be extracted from this file by the `docstrip` program. If you don't want to make any changes to the presentation you can alternatively process the `.dtx` file

directly with  $\text{\LaTeX} 2_{\epsilon}$  to obtain the documentation.

```

1 <*driver>
2 \documentclass[twocolumn]{article}
3
4 \usepackage{ftnright}
5 \usepackage{doc}
6 \AtBeginDocument{\MakeShortVerb{\|}}
7
8 \newcommand{\TUB}{\s1 TUGboat\|}
9 \renewcommand\DescribeMacro[1]{\fbox
10     {\PrintDescribeMacro{#1}}}
11 \renewcommand\DescribeEnv[1]{\fbox
12     {\PrintDescribeEnv{#1}}}
13 \renewcommand\PrintMacroName[1]{ }
14
15 \setlength{\parindent}{1em}
16 \setlength{\parskip}
17     {2pt plus1pt minus1pt}
18 \setlength{\headsep}{20pt}
19 \setlength{\columnsep}{1.5pc}
20 \renewcommand{\bottomfraction}{.4}
21
22 \flushbottom
23 \CodelineIndex
24 \RecordChanges           % produce history

```

2. Please note, that this option only changed the placement of footnotes. Since this article also makes use of the `doc` package [1], that assigns tiny numbers to code lines sprinkled throughout the text, the resulting design is not perfect. This package is now a standard part of  $\text{\LaTeX} 2_{\epsilon}$ .

3. The standard layout in *TUGboat* uses the same size for footnotes and text, giving the footnotes, in my opinion, much too much prominence.

4. People who prefer the rule can add it by redefining the command `\footnoterule` [3, p. 156]. Please, note, that this command should occupy no space, so that a negative space should be used to compensate for the width of the rule used.

5. The tiny numbers or symbols, e.g., the '5' in front of this footnote.

6. Of course, this is only done for the mark preceding the footnote text and not the one used within the main text where a raised number or symbol set in smaller type will help to keep the flow of thoughts, uninterrupted.

7. These are the macros `\@startcolumn`, `\@makecol` and `\@outputdblcol` as we will see below. Of course, the package will take only effect with a document class using a `twocolumn` layout (like `ltugboat`) or when the user additionally specifies `twocolumn` as a document class option in the `\documentclass` command.

```

25 \EnableCrossrefs
26
27 \setcounter{IndexColumns}{2}
28 \IndexPrologue{\section{Index}}
29 All numbers denote code lines where
30 the corresponding entry is used,
31 underlined entries point to the
32 definition.}
33
34 \begin{document}
35   \DocInput{ftnright.dtx}
36 \end{document}
37 </driver>

```

## 4 The Implementation

As usual, we start by identifying the current version of this package file in the transcript file.<sup>8</sup> This actually happens at the very top of this file so it is commented out here.

```

\ProvidesPackage{ftnright}[\filedate\space
    LaTeX2e package \fileversion]

```

To implement the layout described, above we have to distinguish between the left and the right column on a page. For this purpose L<sup>A</sup>T<sub>E</sub>X maintains the switch `\if@firstcolumn`. When assembling material for the left (i.e., the first) column, footnotes should take up no space, since they are held over for the second column. In the second column these footnotes are combined with the ones found there and placed a suitable distance from the main text at the bottom of this column.

This means that we have to change certain parameters for the insertion `\footins` when we construct the second column. The right place to do this is in the L<sup>A</sup>T<sub>E</sub>X macro `\@outputdblcol` which we are going to change later on. What settings for the insertion parameters are appropriate? For setting the first column `\count\footins` and `\skip\footins` should both be zero since footnotes are held over while for the second column `\count\footins` should be 1000 and the `\skip\footins` has to be set to the desired separation between main text and footnotes.<sup>9</sup>

We will allow one column of footnotes (i.e., the right column) at most, so that `\dimen\footins` has to equal `\textheight`. In principle, it would be possible to allow for even more footnotes, but this would complicate matters enormously.<sup>10</sup>

Since a document usually starts with a left column, we have to set `\count` and `\skip\footins` on top-level to zero. For this purpose, we define a macro `\preparefootins` which will first save the current value of `\skip\footins` in a safe place. This saved value will be used later for the second column. In this way, it is possible for the user or a designer of a document class to adjust this parameter without fiddling with the code of this package file.

```

38 <*package>
39 \def\preparefootins{%
40   \global\rcol@footinsskip\skip\footins
41   \global\skip\footins\z@
42   \global\count\footins\z@

```

We will also assign `\textheight` to `\dimen\footins` to allow the user to change this parameter in the preamble.

```

43 \global\dimen\footins\textheight}

```

It is necessary to make the assignments above `\global` because we are going to use this macro in the output routine which has an implicit grouping level to keep the changes made by it local.

8. Nico Poppelier suggested omitting the `\typeout` statements in the production version of the files to avoid showing all that unnecessary information to the user. While I accept his criticism as valid, I decided that this information should at least be placed into the transcript file to make it easier to detect problems arising from the use of older versions. This happens now automatically as the command `\ProvidesPackage` will only write to the transcript file.

9. A value of 1000 means that there is a one-to-one relationship between the real size of the footnote and the size finally occupied by the footnote on the current page.

10. It is not possible to make `\dimen\footins` larger than `\textheight` directly, because this would result in a full left column (with text) and more than one column of footnotes. Instead, one has to make footnotes visible to the page generation algorithm again at the moment when a full column of footnotes is assembled, but we still have some space left in the first column. It is a nice enhancement, and, I suppose, it is of some value for preparing publications in certain disciplines, so here is the challenge . . .

# The `hhline` package\*

David Carlisle  
carlisle@cs.man.ac.uk

1994/05/23

## Abstract

`\hhline` produces a line like `\hline`, or a double line like `\hline\hline`, except for its interaction with vertical lines.

## 1 Introduction

The argument to `\hhline` is similar to the preamble of an `array` or `tabular`. It consists of a list of tokens with the following meanings:

- = A double hline the width of a column.
- A single hline the width of a column.
- ~ A column with no hline.
- | A vline which ‘cuts’ through a double (or single) hline.
- : A vline which is broken by a double hline.
- # A double hline segment between two vlines.
- t The top half of a double hline segment.
- b The bottom half of a double hline segment.
- \* `*{3}{==#}` expands to `==#==#==#`, as in the `*`-form for the preamble.

If a double vline is specified (`||` or `::`) then the hlines produced by `\hhline` are broken. To obtain the effect of an hline ‘cutting through’ the double vline, use a `#` or omit the vline specifiers, depending on whether or not you wish the double vline to break.

The tokens `t` and `b` must be used between two vertical rules. `|tb|` produces the same lines as `#`, but is much less efficient. The main use for these are to make constructions like `|t:` (top left corner) and `:b|` (bottom right corner).

If `\hhline` is used to make a single hline, then the argument should only contain the tokens `-`, `~` and `|` (and `*`-expressions).

---

\*This file has version number v2.03, last revised 1994/05/23.

An example using most of these features is:

```

\begin{tabular}{|cc|c|c|}
\hline{t:::t:::t|}
a&b&c&d\
\hhline{!:::|~|~|}
1&2&3&4\
\hhline{#=#~|=#}
i&j&k&l\
\hhline{||--||--||}
w&x&y&z\
\hhline{|b:::b:::b|}
\end{tabular}

```

a	b	c	d
1	2	3	4
i	j	k	l
w	x	y	z

The lines produced by L<sup>A</sup>T<sub>E</sub>X's `\hline` consist of a single (T<sub>E</sub>X primitive) `\hrule`. The lines produced by `\hhline` are made up of lots of small line segments. T<sub>E</sub>X will place these very accurately in the `.dvi` file, but the program that you use to print the `.dvi` file may not line up these segments exactly. (A similar problem can occur with diagonal lines in the `picture` environment.)

If this effect causes a problem, you could try a different driver program, or if this is not possible, increasing `\arrayrulewidth` may help to reduce the effect.

## 2 The Macros

```

1 <{*package}

\HH@box Makes a box containing a double hline segment. The most common case, both
rules of length \doublerulesep will be stored in \box1, this is not initialised until
\hhline is called as the user may change the parameters \doublerulesep and
\arrayrulewidth. The two arguments to \HH@box are the widths (ie lengths) of
the top and bottom rules.
2 \def\HH@box#1#2{\vbox{%
3 \hrule \@height \arrayrulewidth \@width #1
4 \vskip \doublerulesep
5 \hrule \@height \arrayrulewidth \@width #2}}

\HH@add Build up the preamble in the register \toks@.
6 \def\HH@add#1{\toks@\expandafter{\the\toks@#1}}

\HH@xexpast We 'borrow' the version of \@xexpast from Mittelbach's array.sty, as this allows
\HH@xexnoop # to appear in the argument list.
7 \def\HH@xexpast#1*#2#3#4\@{#%
8 \@tempcnta #2
9 \toks@=#1\@temptokena=#3}%
10 \let\the\toks@\relax \let\the\toks@\relax
11 \def\@tempa{\the\toks@}%

```

# The `indentfirst` package\*

David Carlisle  
carlisle@cs.man.ac.uk

1995/11/23

## Abstract

Make the first line of all sections etc., be indented by the usual paragraph indentation. This should work with all the standard document classes.

`\if@afterindent` L<sup>A</sup>T<sub>E</sub>X uses the switch `\if@afterindent` to decide whether to indent after a section heading. We just need to make sure that this is always true.

```
1 <*package>
2 \let\@afterindentfalse\@afterindenttrue
3 \@afterindenttrue
4 </package>
```

---

\*This file has version number v1.03, last revised 1995/11/23.

# Displaying page layout variables

Kent McPherson a.o.\*

2000/09/25

## 1 Introduction

This L<sup>A</sup>T<sub>E</sub>X 2<sub>ε</sub> package is a reimplementation of `layout.sty` by Kent McPherson. It defines the command `\layout` which produces an overview of the layout of the current document. The command `\layout*` recomputes the values it uses to produce the overview.

The figure on the next page shows the output of the `\layout` command for this document.

## 2 The implementation

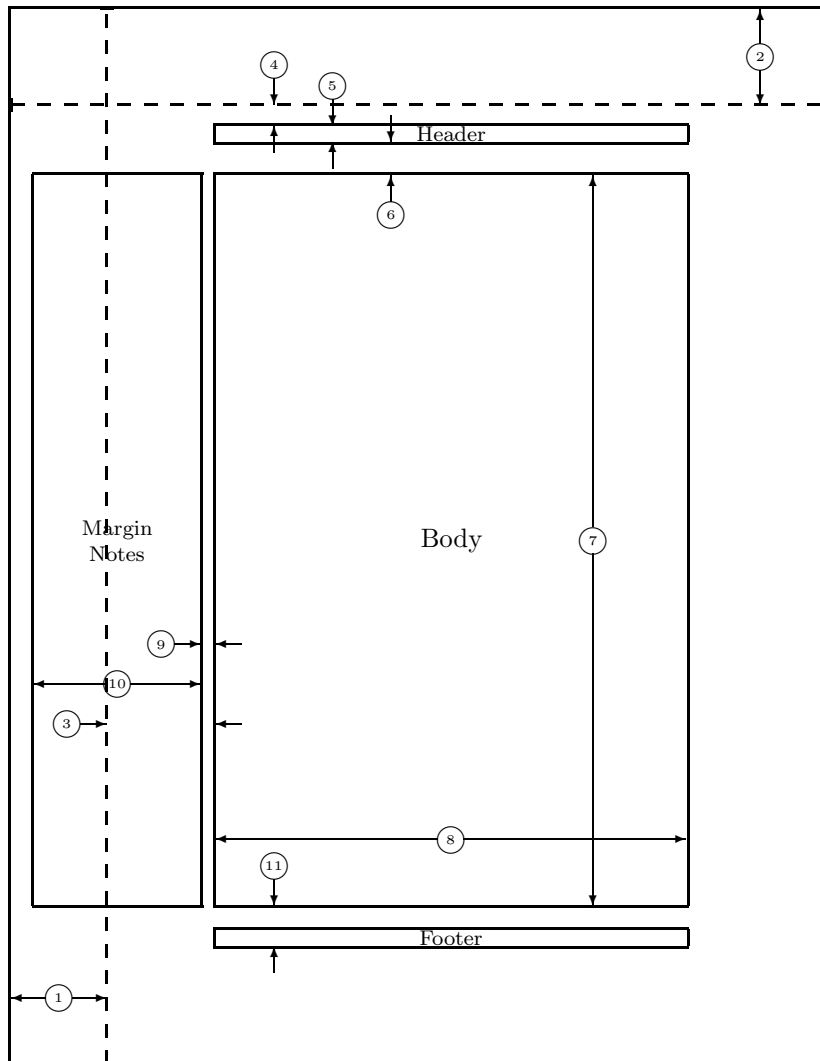
This package prints a figure to illustrate the layout that is implemented by the document class. In the figure several words appear. They are stored in control sequences to be able to select a different language.

```
1 ⟨*package⟩
2 \DeclareOption{dutch}{%
3   \def\Headertext{Kopregel}
4   \def\Bodytext{Broodtekst}
5   \def\Footertext{Voetregel}
6   \def\MarginNotestext{Marge\\Notities}
7   \def\oneinchtext{een inch}
8   \def\notshown{niet getoond}
9 }
10 \DeclareOption{german}{%
11   \def\Headertext{Kopfzeile}
12   \def\Bodytext{Haupttext}
13   \def\Footertext{Fu{\ss}zeile}
14   \def\MarginNotestext{Rand-\\ notizen}
15   \def\oneinchtext{ein Zoll}
16   \def\notshown{ohne Abbildung}
17 }
18 \DeclareOption{ngerman}{\ExecuteOptions{german}}
```

---

\*Converted for L<sup>A</sup>T<sub>E</sub>X 2<sub>ε</sub> by Johannes Braams and modified by Hideo Umeki





- |    |                       |    |                                  |
|----|-----------------------|----|----------------------------------|
| 1  | one inch + \hoffset   | 2  | one inch + \voffset              |
| 3  | \oddsidemargin = 82pt | 4  | \topmargin = 16pt                |
| 5  | \headheight = 12pt    | 6  | \headsep = 25pt                  |
| 7  | \textheight = 550pt   | 8  | \textwidth = 355pt               |
| 9  | \marginparsep = 11pt  | 10 | \marginparwidth = 126pt          |
| 11 | \footskip = 30pt      |    | \marginparpush = 0pt (not shown) |
|    | \hoffset = 0pt        |    | \voffset = 0pt                   |
|    | \paperwidth = 614pt   |    | \paperheight = 794pt             |

# The `longtable` package\*

David Carlisle†

2000/10/22

## Abstract

This package defines the `longtable` environment, a multi-page version of `tabular`.

## List of Tables

1	An optional table caption (used in the list of tables) . . . . .	2
2	A floating table . . . . .	4
3	A difficult <code>\multicolumn</code> combination: pass 1 . . . . .	6
4	A difficult <code>\multicolumn</code> combination: pass 2 . . . . .	6
5	A difficult <code>\multicolumn</code> combination: pass 3 . . . . .	6
6	A difficult <code>\multicolumn</code> combination: pass 4 . . . . .	6
7	A summary of <code>longtable</code> commands . . . . .	9

## 1 Introduction

`longtable` The `longtable` package defines a new environment, `longtable`, which has most of the features of the `tabular` environment, but produces tables which may be broken by  $\TeX$ 's standard page-breaking algorithm. It also shares some features with the `table` environment. In particular it uses the same counter, `table`, and has a similar `\caption` command. Also, the standard `\listoftables` command lists tables produced by either the `table` or `longtable` environments.

The following example uses most of the features of the `longtable` environment. An edited listing of the input for this example appears in Section 8.

**Note:** Various parts of the following table will **not** line up correctly until this document has been run through  $\LaTeX$  several times. This is a characteristic feature of this package, as described below.

---

\*This file has version number v4.10, last revised 2000/10/22.

†The new algorithm for aligning ‘chunks’ of a table used in version 4 of this package was devised coded and documented by David Kastrop, `dak@neuroinformatik.ruhr-uni-bochum.de`.

Table 1: A long table

* This part appears at the top of the table *	
* FIRST	* SECOND *
* longtable columns are specified	* in the *
* same way as in the tabular	* environment. *
* @{*}r lp{1in}@{*}	* in this case. *
* Each row ends with a	* \ command. *
* The \ command has an	* optional *
* argument, just as in	* the *
* tabular	* environment. *
* See the effect of \[10pt]	* ? *
* Lots of lines	* like this. *
* Lots of lines	* like this. *
* Lots of lines	* like this. *
* Lots of lines	* like this. *
* Also \hline may be used,	* as in tabular. *
* That was a \hline	* . *
* That was \hline\hline	* . *
* This is a \multicolumn{2}{ c }	*
* If a page break occurs at a \hline then	* a line is drawn *
* at the bottom of one page and at the	* top of the next. *
* The [t] [b] [c] argument of tabular	* can not be used. *
* The optional argument may be one of	* [l] [r] [c] *
* to specify whether the table should be	* adjusted *
* to the left, right	* or centrally. *
* Lots of lines	* like this. *
* Lots of lines	* like this. *
* Lots of lines	* like this. *
* Lots of lines	* like this. *
* Lots of lines	* like this. *
* Lots of lines	* like this. *
* Lots of lines	* like this. *
* Lots of lines	* like this. *
* Lots of lines	* like this. *
* Lots of lines	* like this. *
* Lots of lines	* like this. *
* Lots of lines	* like this. *
* Lots of lines	* like this. *
* Lots of lines	* like this. *
* Lots of lines	* like this. *
* Lots of lines	* like this. *
* This goes at the	* bottom. *

Table 1: (continued)

* This part appears at the top of every other page *			
	<b>First</b>	<b>Second</b>	
*	Lots of lines	like this.	*
*	Lots of lines	like this.	*
*	Lots of lines	like this.	*
*Some lines may take up a lot of space, like this:		This last*	
		column is a “p”	
		column so this	
		“row” of the	
		table can take	
		up several lines.	
		Note however	
		that T <sub>E</sub> X will	
		never break a	
		page within	
		such a row.	
		Page breaks	
		only occur	
		between rows of	
		the table or at	
		\hline	
		commands.	
*	Lots of lines	like this.	*
*	Lots of lines	like this.	*
*	Lots of lines	like this.	*
*	Lots of lines	like this.	*
*	Lots of lines	like this.	*
*	Lots of lines	like this.	*
*	Lots of lines	like this.	*
*	Lots <sup>1</sup> of lines	like this.	*
*	Lots of lines	like this <sup>2</sup>	*
*	Lots of lines	like this.	*
*	Lots of lines	like this.	*
*	These lines will	appear	*
*	in place of the	usual foot	*
*	at the end	of the table	*

---

<sup>1</sup>This is a footnote.

<sup>2</sup>longtable takes special precautions, so that footnotes may also be used in ‘p’ columns.

A	tabular	environment
within	a floating	table

Table 2: A floating table

## 2 Chunk Size

**LTchunksize** In order to  $\TeX$  multi-page tables, it is necessary to break up the table into smaller chunks, so that  $\TeX$  does not have to keep everything in memory at one time. By default `longtable` uses 20 rows per chunk, but this can be set by the user, with e.g., `\setcounter{LTchunksize}{10}`.<sup>3</sup> These chunks do not affect page breaking, thus if you are using a  $\TeX$  with a lot of memory, you can set `LTchunksize` to be several pages of the table.  $\TeX$  will run faster with a large `LTchunksize`. However, if necessary, `longtable` can work with `LTchunksize` set to 1, in which case the memory taken up is negligible. Note that if you use the commands for setting the table head or foot (see below), the `LTchunksize` must be at least as large as the number of rows in each of the head or foot sections.

This document specifies `\setcounter{LTchunksize}{10}`. If you look at the previous table, after the *first* run of  $\LaTeX$  you will see that various parts of the table do not line up.  $\LaTeX$  will also have printed a warning that the column widths had changed. `longtable` writes information onto the `.aux` file, so that it can line up the different chunks. Prior to version 4 of this package, this information was not used unless a `\setlongtables` command was issued, however, now the information is always used, using a new algorithm<sup>4</sup> and so `\setlongtables` is no longer needed. It is defined (but does nothing) for the benefit of old documents that use it.

## 3 Captions and Headings

At the start of the table one may specify lines which are to appear at the top of every page (under the headline, but before the other lines of the table). The lines are entered as normal, but the last `\` command is replaced by a `\endhead` command. If the first page should have a different heading, then this should be entered in the same way, and terminated with the `\endfirsthead` command. The `LTchunksize` should be at least as large as the number of rows in the heading.

There are also `\endfoot` and `\endlastfoot` commands which are used in the same way (at the *start* of the table) to specify rows (or an `\hline`) to appear at the bottom of each page. In certain situations, you may want to place lines which logically belong in the table body at the end of the `firsthead`, or the beginning of the `lastfoot`. This helps to control which lines appear on the first and last page of the table.

**\caption** The `\caption{...}` command is essentially equivalent to

<sup>3</sup>You can also use the plain  $\TeX$  syntax `\LTchunksize=10`.

<sup>4</sup>Due to David Kastrup.

`\multicolumn{n}{c}{\parbox{\LTcapwidth}{...}}`

where `n` is the number of columns of the table. You may set the width of the caption with a command such as `\setlength{\LTcapwidth}{2in}` in the preamble of your document. The default is 4in. `\caption` also writes the information to produce an entry in the list of tables. As with the `\caption` command in the `figure` and `table` environments, an optional argument specifies the text to appear in the list of tables if this is different from the text to appear in the caption. Thus the caption for table 1 was specified as `\caption[An optional table caption (used in the list of tables)]{A long table\label{long}}`.

You may wish the caption on later pages to be different to that on the first page. In this case put the `\caption` command in the first heading, and put a subsidiary caption in a `\caption[]` command in the main heading. If the optional argument to `\caption` is empty, no entry is made in the list of tables. Alternatively, if you do not want the table number to be printed each time, use the `\caption*` command.

The captions are set based on the code for the article class. If you have re-defined the standard `\@makecaption` command to produce a different format for the captions, you may need to make similar changes to the `longtable` version, `\LT@makecaption`. See the code section for more details.

A more convenient method of customising captions is given by the `caption(2)` package, which provides commands for customising captions, and arranges that the captions in standard environments, and many environments provided by packages (including `longtable`) are modified in a compatible manner.

You may use the `\label` command so that you can cross reference `longtables` with `\ref`. Note however, that the `\label` command should not be used in a heading that may appear more than once. Place it either in the `firsthead`, or in the body of the table. It should not be the `first` command in any entry.

## 4 Multicolumn entries

The `\multicolumn` command may be used in `longtable` in exactly the same way as for `tabular`. So you may want to skip this section, which is rather technical, however coping with `\multicolumn` is one of the main problems for an environment such as `longtable`. The main effect that a user will see is that certain combinations of `\multicolumn` entries will result in a document needing more runs of `LATEX` before the various ‘chunks’ of a table align.

The examples in this section are set with `LTchunksize` set to the minimum value of one, to demonstrate the effects when `\multicolumn` entries occur in different chunks.

Consider Table 3. In the second chunk, `longtable` sees the wide multicolumn entry. At this point it thinks that the first two columns are very narrow. All the width of the multicolumn entry is assumed to be in the third column. (This is a ‘feature’ of `TEX`’s primitive `\halign` command.) `longtable` then passes the information that there is a wide third column to the later chunks, with the result that the first pass over the table is too wide.

Table 3: A difficult `\multicolumn` combination: pass 1

1	2	3
wide multicolumn spanning 1-3		
multicolumn 1-2		3
wide 1	2	3

Table 4: A difficult `\multicolumn` combination: pass 2

1	2	3
wide multicolumn spanning 1-3		
multicolumn 1-2		3
wide 1	2	3

Table 5: A difficult `\multicolumn` combination: pass 3

1	2	3
wide multicolumn spanning 1-3		
multicolumn 1-2		3
wide 1	2	3

Table 6: A difficult `\multicolumn` combination: pass 4

1	2	3
wide multicolumn spanning 1-3		
multicolumn 1-2		3
wide 1	2	3

If the 'saved row' from this first pass was re-inserted into the table on the next pass, the table would line up in two passes, but would be much too wide.

\kill

The solution to this problem used in Versions 1 and 2, was to use a \kill line. If a line is \killed, by using \kill rather than \\ at the end of the line, it is used in calculating column widths, but removed from the final table. Thus entering \killed copies of the last two rows before the wide multicolumn entry would mean that \halign 'saw' the wide entries in the first two columns, and so would not widen the third column by so much to make room for the multicolumn entry.

In Version 3, a new solution was introduced. If the saved row in the .aux file was not being used, longtable used a special 'draft' form of \multicolumn, this modified the definition, so the spanning entry was never considered to be wider than the columns it spanned. So after the first pass, the .aux file stored the widest normal entry for each column, no column was widened due to \spanned columns. By default longtable ignored the .aux file, and so each run of L<sup>A</sup>T<sub>E</sub>X was considered a first pass. Once the \setlongtables declaration was given, the saved row in the .aux file, and the proper definition of \multicolumn were used. If any \multicolumn entry caused one of the columns to be widened, this information could not be passed back to earlier chunks, and so the table would not correctly line up until the third pass. This algorithm always converged in three passes as described above, but in examples such as the ones in Tables 3-6, the final widths were not optimal as the width of column 2, which is determined by a \multicolumn entry was not known when the final width for column 3 was fixed, due to the fact that both \multicolumn commands were switched from 'draft' mode to 'normal' mode at the same time.

Version 4 alleviates the problem considerably. The first pass of the table will indeed have the third column much too wide. However, on the next pass longtable will notice the error and reduce the column width accordingly. If this has to propagate to chunks before the \multicolumn one, an additional pass will, of course, be needed. It is possible to construct tables where this rippling up of the correct widths takes several passes to 'converge' and produce a table with all chunks aligned. However in order to need many passes one needs to construct a table with many overlapping \multicolumn entries, all being wider than the natural widths of the columns they span, and all occurring in different chunks. In the typical case the algorithm will converge after three or four passes, and, the benefits of not needing to edit the document before the final run to add \setlongtables, and the better choice of final column widths in the case of multiple \multicolumn entries will hopefully more than pay for the extra passes that may possibly be needed.

So Table 3 converges after 4 passes, as seen in Table 6.

You can still speed the convergence by introducing judicious \kill lines, if you happen to have constellations like the above.

If you object even to L<sup>A</sup>T<sub>E</sub>X-ing a file twice, you should make the first line of every longtable a \kill line that contains the widest entry to be used in each column. All chunks will then line up on the first pass.



## 5 Adjustment

The optional argument of `longtable` controls the horizontal alignment of the table. The possible options are `[c]`, `[r]` and `[l]`, for centring, right and left adjustment, respectively. Normally centring is the default, but this document specifies

```
\Lleft
\Lright
\setlength\Lleft\parindent
\setlength\Lright\fill
```

in the preamble, which means that the tables are set flush left, but indented by the usual paragraph indentation. Any lengths can be specified for these two parameters, but at least one of them should be a rubber length so that it fills up the width of the page, unless rubber lengths are added between the columns using the `\extracolsep` command. For instance

```
\begin{tabular*}{\textwidth}{@{\extracolsep{...}}...}
```

produces a full width table, to get a similar effect with `longtable` specify

```
\setlength\Lleft{0pt}
\setlength\Lright{0pt}
\begin{longtable}{@{\extracolsep{...}}...}
```

## 6 Changes

This section highlights the major changes since version 2. A more detailed change log may be produced at the end of the code listing if the `ltxdoc.cfg` file specifies

```
\AtBeginDocument{\RecordChanges}
\AtEndDocument{\PrintChanges}
```

Changes made between versions 2 and 3.

- The mechanism for adding the head and foot of the table has been completely rewritten. With this new mechanism, `longtable` does not need to issue a `\clearpage` at the start of the table, and so the table may start half way down a page. Also the `\endlastfoot` command which could not safely be implemented under the old scheme, has been added.
- `longtable` now issues an error if started in the scope of `\twocolumn`, or the `multicols` environment.
- The separate documentation file `longtable.tex` has been merged with the package file, `longtable.dtx` using Mittelbach's `doc` package.
- Support for footnotes has been added. Note however that `\footnote` will not work in the 'head' or 'foot' sections of the table. In order to put a footnote in those sections (e.g., inside a caption), use `\footnotemark` at that point, and `\footnotetext` anywhere in the table *body* that will fall on the same page.

- The treatment of `\multicolumn` has changed, making `\kill` lines unnecessary, at the price of sometimes requiring a third pass through L<sup>A</sup>T<sub>E</sub>X.
- The `\newpage` command now works inside a `longtable`.

Changes made between versions 3 and 4.

- A new algorithm is used for aligning chunks. As well as the widest width in each column, `longtable` remembers which chunk produced this maximum. This allows it to check that the maximum is still achieved in later runs. As `longtable` can now deal with columns shrinking as the file is edited, the `\setlongtables` system is no longer needed and is disabled.
- An extra benefit of the new algorithm's ability to deal with 'shrinking' columns is that it can give better (narrower) column widths in the case of overlapping `\multicolumn` entries in different chunks than the previous algorithm produced.
- The 'draft' multicolumn system has been removed, along with related commands such as `\LTmulticolumn`.
- The disadvantage of the new algorithm is that it can take more passes. The theoretical maximum is approximately twice the length of a 'chain' of columns with overlapping `\multicolumn` entries, although in practice it usually converges as fast as the old version. (Which always converged in three passes once `\setlongtables` was activated.)
- `\*` and `\nopagebreak` commands may be used to control page breaking.

## 7 Summary

Table 7: A summary of `longtable` commands

<b>Parameters</b>		
<code>\LTleft</code>	Glue to the left of the table.	<code>(\fill)</code>
<code>\LTright</code>	Glue to the right of the table.	<code>(\fill)</code>
<code>\LTpre</code>	Glue before the the table.	<code>(\bigskipamount)</code>
<code>\LTpost</code>	Glue after the the table.	<code>(\bigskipamount)</code>
<code>\LTcapwidth</code>	The width of a parbox containing the caption.	<code>(4in)</code>
<code>LTchunksize</code>	The number of rows per chunk.	<code>(20)</code>
<b>Optional arguments to <code>\begin{longtable}</code></b>		
<i>none</i>	Position as specified by <code>\LTleft</code> and <code>\LTright</code> .	
<code>[c]</code>	Centre the table.	
<code>[l]</code>	Place the table flush left.	
<code>[r]</code>	Place the table flush right.	

**Commands to end table rows**

---

<code>\</code>	Specifies the end of a row
<code>\\[<i>&lt;dim&gt;</i>]</code>	Ends row, then adds vertical space (as in the <code>tabular</code> environment).
<code>\\*</code>	The same as <code>\\</code> but disallows a page break after the row.
<code>\tabularnewline</code>	Alternative to <code>\\</code> for use in the scope of <code>\raggedright</code> and similar commands that redefine <code>\\</code> .
<code>\kill</code>	Row is ‘killed’, but is used in calculating widths.
<code>\endhead</code>	Specifies rows to appear at the top of every page.
<code>\endfirsthead</code>	Specifies rows to appear at the top the first page.
<code>\endfoot</code>	Specifies rows to appear at the bottom of every page.
<code>\endlastfoot</code>	Specifies rows to appear at the bottom of the last page.

**longtable caption commands**

---

<code>\caption{<i>&lt;caption&gt;</i>}</code>	Caption ‘Table ?: <i>&lt;caption&gt;</i> ’, and a ‘ <i>&lt;caption&gt;</i> ’ entry in the list of tables.
<code>\caption[<i>&lt;lot&gt;</i>]{<i>&lt;caption&gt;</i>}</code>	Caption ‘Table ?: <i>&lt;caption&gt;</i> ’, and a ‘ <i>&lt;lot&gt;</i> ’ entry in the list of tables.
<code>\caption[]{<i>&lt;caption&gt;</i>}</code>	Caption ‘Table ?: <i>&lt;caption&gt;</i> ’, but no entry in the list of tables.
<code>\caption*{<i>&lt;caption&gt;</i>}</code>	Caption ‘ <i>&lt;caption&gt;</i> ’, but no entry in the list of tables.

**Commands available at the start of a row**

---

<code>\pagebreak</code>	Force a page break.
<code>\pagebreak[<i>&lt;val&gt;</i>]</code>	A ‘hint’ between 0 and 4 of the desirability of a break.
<code>\nopagebreak</code>	Prohibit a page break.
<code>\nopagebreak[<i>&lt;val&gt;</i>]</code>	A ‘hint’ between 0 and 4 of the undesirability of a break.
<code>\newpage</code>	Force a page break.

**Footnote commands available inside longtable**

---

<code>\footnote</code>	Footnotes, but may not be used in the table head & foot.
<code>\footnotemark</code>	Footnotemark, may be used in the table head & foot.
<code>\footnotetext</code>	Footnote text, use in the table body.

**Setlongtables**

---

<code>\setlongtables</code>	Obsolete command. Does nothing now.
-----------------------------	-------------------------------------

## 8 Verbatim highlights from Table 1

```

\begin{longtable}{@{*}r||p{1in}@{*}}
KILLED & LINE!!!! \kill
\caption[An optional table caption ...]{A long table\label{long}}\
\hline\hline
\multicolumn{2}{@{*}c@{*}}%
    {This part appears at the top of the table}\
\textsc{First}&\textsc{Second}\
\hline\hline
\endfirsthead
\caption[]{(continued)}\
\hline\hline
\multicolumn{2}{@{*}c@{*}}%
    {This part appears at the top of every other page}\
\textbf{First}&\textbf{Second}\
\hline\hline
\endhead
\hline
This goes at the&bottom.\
\hline
\endfoot
\hline
These lines will&appear\
in place of the & usual foot\
at the end& of the table\
\hline
\endlastfoot
\env{longtable} columns are specified& in the \
same way as in the \env{tabular}& environment.\
...
\multicolumn{2}{|c|}{This is a ...}\
...
Some lines may take...&
    \raggedleft This last column is a ‘p’ column...
    \tabularnewline
...
Lots of lines& like this.\
...
\hline
Lots\footnote{...} of lines& like this.\
Lots of lines& like this\footnote{...}\
\hline
Lots of lines& like this.\
...
\end{longtable}

```

# An environment for multicolumn output<sup>\*†</sup>

Frank Mittelbach

Email: see top of the source file

Printed March 10, 2004

## Abstract

This article describes the use and the implementation of the `multicols` environment. This environment allows switching between one and multicolumn format on the same page. Footnotes are handled correctly (for the most part), but will be placed at the bottom of the page and not under each column.  $\LaTeX$ 's float mechanism, however, is partly disabled in the current implementation. At the moment only page-wide floats (i.e., star-forms) can be used within the scope of the environment.

## Preface to version 1.5

This new release contains two major changes: `multicols` will now support up to 10 columns and two more tuning possibilities have been added to the balancing routine. The balancing routine now checks the badness of

the resulting columns and rejects solutions that are larger than a certain treshold.

At the same time `multicols` has been upgraded to run under  $\LaTeX 2_{\epsilon}$ .

I apologise for the state of the

code documentation but the work on  $\LaTeX 2_{\epsilon}$  kept me too busy to do a proper job. This will hopefully be corrected in the near future.

## 1 Introduction

Switching between two column and one column layout is possible in  $\LaTeX$ , but every use of `\twocolumn` or `\onecolumn` starts a new page. Moreover, the last page of two column output isn't balanced and this often results in an empty, or nearly empty, right column. When I started to write macros for `doc.sty` (see "The `doc-Option`", *TUGboat* volume

10 #2, pp. 245–273) I thought that it would be nice to place the index on the same page as the bibliography. And balancing the last page would not only look better, it also would save space; provided of course that it is also possible to start the next article on the same page. Rewriting the index environment was comparatively easy, but the next goal, designing an environ-

ment which takes care of footnotes, floats etc., was a harder task. It took me a whole weekend<sup>1</sup> to get together the few lines of code below and there is still a good chance that I missed something after all.

Try it and, hopefully, enjoy it; and *please* direct bug reports and suggestions back to Mainz.

---

<sup>\*</sup>This file has version number v1.5z, last revised 2000/07/10.

<sup>†</sup>Note: This package is released under terms which affect its use in commercial applications. Please see the details at the top of the source file.

<sup>1</sup>I started with the algorithm given in the  $\TeX$ book on page 417. Without this help a weekend would not have been enough.

## 2 The User Interface

To use the environment one simply says

```
\begin{multicols}{\langle number \rangle}
  \langle multicolumn text \rangle
\end{multicols}
```

where  $\langle number \rangle$  is the required number of columns and  $\langle multicolumn text \rangle$  may contain arbitrary L<sup>A</sup>T<sub>E</sub>X commands, except that floats and marginpars are not allowed in the current implementation<sup>2</sup>.

As its first action, the multicols environment measures the current page to determine whether there is enough room for some portion of multicolumn output. This is controlled by the  $\langle dimen \rangle$  variable `\premulticols` which can be changed by the user with ordinary L<sup>A</sup>T<sub>E</sub>X commands. If the space is less than `\premulticols`, a new page is started. Otherwise, a `\vskip` of `\multicolsep` is added.<sup>3</sup>

When the end of the multicols environment is encountered, an analogous mechanism is employed, but now we test whether there is a space larger than `\postmulticols` available. Again we add `\multicolsep` or start a new page.

It is often convenient to spread some text over all columns, just before the multicolumn output, without any page break in between. To achieve this the multicols environment has an optional second argument which can be used for this purpose. For example, the text you are now reading was started with

```
\begin{multicols}{3}
  [\section{The User
    Interface}] ...
```

If such text is unusually

long (or short) the value of `\premulticols` might need adjusting to prevent a bad page break. We therefore provide a third argument which can be used to overwrite the default value of `\premulticols` just for this occasion. So if you want to combine some longer single column text with a multicols environment you could write

```
\begin{multicols}{3}
  [\section{Index}
   This index contains ...]
  [6cm]
  ...
```

The space between columns is controlled by the length parameter `\columnsep`. The width for the individual columns is automatically calculated from this parameter and the current `\linewidth`. In this article a value of 18.0pt was used.

Separation of columns with vertical rules is achieved by setting the parameter `\columnseprule` to some positive value. In this article a value of .4pt was used.

Since narrow columns tend to need adjustments in interline spacing we also provide a  $\langle skip \rangle$  parameter called `\multicolbaselineskip` which is added to the `\baselineskip` parameter inside the multicols environment. Please use this parameter with care or leave it alone; it is intended only for package file designers since even small changes might produce totally unexpected changes to your document.

### 2.1 Balancing columns

Besides the previously mentioned parameters, some others are provided to influence the layout of the columns generated.

Paragraphing in T<sub>E</sub>X is controlled by several parameters. One of the most important is called `\tolerance`: this controls the allowed ‘looseness’ (i.e. the amount of blank space between words). Its default value is 200 (the L<sup>A</sup>T<sub>E</sub>X `\fussy`) which is too small for narrow columns. On the other hand the `\sloppy` declaration (which sets `\tolerance` to 10000 = ∞) is too large, allowing really bad spacing.<sup>4</sup>

We therefore use a `\multicoltolerance` parameter for the `\tolerance` value inside the multicols environment. Its default value is 9999 which is less than infinity but ‘bad’ enough for most paragraphs in a multicolumn environment. Changing its value should be done outside the multicols environment. Since `\tolerance` is set to `\multicoltolerance` at the beginning of every multicols environment one can locally overwrite this default by assigning `\tolerance_□=□\langle desired value \rangle`. There also exists a `\multicolpretolerance` parameter holding the value for `\pretolerance` within a multicols environment. Both parameters are usually used only by package designers.

Generation of multicolumn output can be divided into two parts. In the first part we are collecting material for a page, shipping it out, collecting material for the next page, and so on. As a second step, balancing will

<sup>2</sup>This is dictated by lack of time. To implement floats one has to reimplement the whole L<sup>A</sup>T<sub>E</sub>X output routine.

<sup>3</sup>Actually the added space may be less because we use `\addvspace` (see the L<sup>A</sup>T<sub>E</sub>X manual for further information about this command).

<sup>4</sup>Look at the next paragraph, it was set with the `\sloppy` declaration.

be done when the end of the `multicols` environment is reached. In the first step `TeX` might consider more material whilst finding the final columns than it actually use when shipping out the page. This might cause a problem if a footnote is encountered in the part of the input considered, but not used, on the current page. In this case the footnote might show up on the current page, while the footnotemark corresponding to this footnote might be set on the next one.<sup>5</sup> Therefore the `multicols` environment gives a warning message<sup>6</sup> whenever it is unable to use all the material considered so far.

If you don't use footnotes too often the chances of something actually going wrong are very slim, but if this happens you can help `TeX` by using a `\pagebreak` command in the final document. Another way to influence the behavior of `TeX` in this respect is given by the counter variable `'collectmore'`. If you use the `\setcounter` declaration to set this counter to  $\langle number \rangle$ , `TeX` will consider  $\langle number \rangle$  more (or less) lines before making its final decision. So a value of  $-1$  may solve all your problems at the cost of slightly less optimal columns.

In the second step (balancing columns) we have other bells and whistles. First of all you can say `\raggedcolumns` if you don't want the bottom lines to be aligned. The default is `\flushcolumns`, so `TeX` will normally try to make both the top and bottom baselines of all columns align.

<sup>5</sup>The reason behind this behavior is the asynchronous character of the `TeX page_builder`. However, this could be avoided by defining very complicated output routines which don't use `TeX` primitives like `\insert` but do everything by hand. This is clearly beyond the scope of a weekend problem.

<sup>6</sup>This message will be generated even if there are no footnotes in this part of the text.

Additionally you can set another counter, the `'unbalance'` counter, to some positive  $\langle number \rangle$ . This will make all but the right-most column  $\langle number \rangle$  of lines longer than they would normally have been. 'Lines' in this context refer to normal text lines (i.e. one `\baselineskip` apart); thus, if your columns contain displays, for example, you may need a higher  $\langle number \rangle$  to shift something from one column into another.

Unlike `'collectmore'`, the `'unbalance'` counter is reset to zero at the end of the environment so it only applies to one `multicols` environment.

The two methods may be combined but I suggest using these features only when fine tuning important publications.

Two more general tuning possibilities were added with version 1.5. `TeX` allows to measure the badness of a column in terms of an integer value, where 0 means optimal and any higher value means a certain amount of extra white space. 10000 is considered to be infinitely bad (`TeX` does not distinguish any further). In addition the special value 100000 means overfull (i.e., the column contains more text than could possibly fit into it).

The new release now measures every generated column and ignores solutions where at least one column has a badness being larger than the value of the counter `columnbadness`. The default value for this counter is 10000, thus `TeX` will accept all solutions except those being overfull. By setting the counter to a smaller value you can force the algorithm to search for solutions that do not have columns with a

lot of white space.

However, if the setting is too low, the algorithm may not find any acceptable solution at all and will then finally choose the extreme solution of placing all text into the first column.

Often, when columns are balanced, it is impossible to find a solution that distributes the text evenly over all columns. If that is the case the last column usually has less text than the others. In the earlier releases this text was stretched to produce a column with the same height as all others, sometimes resulting in really ugly looking columns.

In the new release this stretching is only done if the badness of the final column is not larger than the value of the counter `finalcolumnbadness`. The default setting is 9999, thus preventing the stretching for all columns that `TeX` would consider infinitely bad. In that case the final column is allowed to run short which gives a much better result.

And there are two more parameters of some experimental nature, one called `\multicolovershoot` the other `\multicolundershoot`. They control the amount of space a column is allowed to be "too full" or "too short" without affecting the column badness. They are set to 2pt by default.

## 2.2 Not balancing the columns

Although this package was written to solve the problem of balancing columns, I got repeated requests to provide a version where all white space is automatically placed in the last column or columns. Since version

v1.5q this now exists: if you use `multicols*` instead of the usual environment the columns on the last page are not balanced. Of course, this environment only works on top-level, e.g., inside a box one has to balance to determine a column height in absence of a fixed value.

## 2.3 Manually breaking columns

Another request often voiced was: “How to I tell L<sup>A</sup>T<sub>E</sub>X that it should break the first column after this particular line?”. The `\pagebreak` command (which works with the two-column option of L<sup>A</sup>T<sub>E</sub>X) is of no use here since it would end the collection phase of `multicols` and thus all columns on that page. So with version 1.5u the `\columnbreak` command was added. If used within a paragraph it marks the end of the current line as the desired breakpoint. You can observe its effect on the previous page where three lines of text have been artificially forced into the second column (resulting in some white space between paragraphs in the first column).

## 2.4 Floats inside a multicols environment

Within the `multicols` environment the usual star float commands are available but their function is somewhat different as in the two-column mode of standard L<sup>A</sup>T<sub>E</sub>X. Stared floats, e.g., `figure*`, denote page wide floats that are handled in a similar fashion as normal floats outside the `multicols` environment. However, they will never show up on the page where they are encountered. In other words, one can influence their placement by specifying a combination of `t`, `b`, and/or `p` in their optional argument, but

h doesn’t work because the first possible place is the top of the next page. One should also note, that this means that their placement behavior is determined by the values of `\topfraction`, etc. rather than by `\dbl...`

## 2.5 Warnings

Under certain circumstances the use of the `multicols` environment may result in some warnings from T<sub>E</sub>X or L<sup>A</sup>T<sub>E</sub>X. Here is a list of the important ones and the possible cause:

**Underfull \hbox (badness ...)**

As the columns are often very narrow T<sub>E</sub>X wasn’t able to find a good way to break the paragraph. Underfull denotes a loose line but as long the badness values is below 10000 the result is probably acceptable.

**Underfull \vbox ... while \output is active**

If a column contains an character with an unusual depth, for example a ‘(’, in the bottom line then this message may show up. It usually has no significance as long as the value is not more than a few points.

**LaTeX Warning: I moved some lines to the next page**

As mentioned above, `multicols` sometimes screws up the footnote numbering. As a precaution, whenever there is a footnote on a page that where `multicols` had to leave a remainder for the following page this warning appears. Check the footnote numbering on this page. If it turns out that it is wrong you have to manually break the page using `\newpage` or `\pagebreak[...]`.

**Floats and marginpars not allowed inside ‘multicols’ environment!**

This message appears if you try to use the `\marginpar` command or an unstarred version of the `figure` or `table` environment. Such floats will disappear!

## 2.6 Tracing the output

To understand the reasoning behind the decisions T<sub>E</sub>X makes when processing a `multicols` environment, a tracing mechanism is provided. If you set the counter ‘`multicols`’ to a positive  $\langle number \rangle$  you then will get some tracing information on the terminal and in the transcript file:

$\langle number \rangle = 1$ . T<sub>E</sub>X will now tell you, whenever it enters or leaves a `multicols` environment, the number of columns it is working on and its decision about starting a new page before or after the environment.

$\langle number \rangle = 2$ . In this case you also get information from the balancing routine: the heights tried for the left and right-most columns, information about shrinking if the `\raggedcolumns` declaration is in force and the value of the ‘`unbalance`’ counter if positive.

$\langle number \rangle = 3$ . Setting  $\langle number \rangle$  to this value will additionally trace the mark handling algorithm. It will show what marks are found, what marks are considered, etc. To fully understand this information you will probably have to read carefully through the implementation.

$\langle number \rangle \geq 4$ . Setting  $\langle number \rangle$  to such a high value



will additionally place an `\hrule` into your output, separating the part of text which had already been considered

on the previous page from the rest. Clearly this setting should *not* be used for the final output. It will also activate even

more debugging code for mark handling.

## 3 Prefaces to older versions

### 3.1 Preface to version 1.4

Beside fixing some bugs as mentioned in the `multicol.bug` file this new release enhances the `multicols` environment by allowing for balancing in arbitrary contexts. It is now, for example, possible to balance text within a `multicols` or a `minipage` as shown in 2 where a `multicols` environment within a `quote` environment was used. It is now even possible to nest `multicols` environments.

The only restriction to such inner `multicols` environments (nested, or within  $\TeX$ 's internal vertical mode) is that such vari-

ants will produce a box with the balanced material in it, so that they can not be broken across pages or columns.

Additionally I rewrote the algorithm for balancing so that it will now produce slightly better results.

I updated the source documentation but like to apologize in advance for some 'left over' parts that slipped through the revision.

A note to people who like to improve the balancing algorithm of `multicols`: The balancing routine is now placed into

a single macro which is called `\balance@columns`. This means that one can easily try different balancing routines by rewriting this macro. The interface for it is explained in table 1. There are several improvements possible, one can think of integrating the `\badness` function of  $\TeX$ 3, define a faster algorithm for finding the right column height, etc. If somebody thinks he/she has an enhancement I would be pleased to learn about it. But please obey the copyright notice and don't change `multicol.dtx` directly!

### 3.2 Preface to version 1.2

After the article about the `multicols` environment was published in *TUGboat* 10#3, I got numerous requests for these macros. However, I also got a changed version of my style file, together with a letter asking me if I would include the changes to get better paragraphing results in the case of narrow lines. The main differences to my original style option were additional parameters (like `\multicoladjdemerits` to be used for `\adjdemerits`, etc.) which would influence the line breaking algorithm.

But actually resetting such parameters to zero or even worse to a negative value won't give better line breaks inside the `multicols` environment.  $\TeX$ 's line breaking algorithm will only look at those possible line breaks which can be reached without a badness higher than the current value of

`\tolerance` (or `\pretolerance` in the first pass). If this isn't possible, then, as a last resort,  $\TeX$  will produce overfull boxes. All those (and only those) possible break points will be considered and finally the sequence which results in the fewest demerits will be chosen. This means that a value of  $-1000$  for `\adjdemerits` instructs  $\TeX$  to prefer visibly incompatible lines instead of producing better line breaks.

However, with  $\TeX$  3.0 it is possible to get decent line breaks even in small columns by setting `\emergencystretch` to an appropriate value. I implemented a version which is capable of running both in the old and the new  $\TeX$  (actually it will simply ignore the new feature if it is not available). The calculation of `\emergencystretch` is probably incorrect. I made a few tests but

of course one has have much more experience with the new possibilities to achieve the maximum quality.

Version 1.1a had a nice 'feature': the penalty for using the forbidden floats was their ultimate removal from  $\LaTeX$ 's `\@freelist` so that after a few `\marginpars` inside the `multicols` environment floats were disabled forever. (Thanks to Chris Rowley for pointing this out.) I removed this misbehaviour and at the same time decided to allow at least floats spanning all columns, e.g., generated by the `figure*` environment. You can see the new functionality in table 2 which was inserted at this very point. However single column floats are still forbidden and I don't think I will have time to tackle this problem in the near future. As an advice for all who

The macro `\balance@columns` that contains the code for balancing gathered material is a macro without parameters. It assumes that the material for balancing is stored in the box `\mult@box` which is a `\vbox`. It also “knows” about all parameters set up by the `multicols` environment, like `\col@number`, etc. It can also assume that `\@colroom` is the still available space on the current page.

When it finishes it must return the individual columns in boxes suitable for further processing with `\page@sofar`. This means that the left column should be stored in box reg-

ister `\mult@gfirstbox`, the next in register `\mult@firstbox + 2`, ..., only the last one as an exception in register `\mult@grightbox`. Furthermore it has to set up two the macros `\kept@firstmark` and `\kept@botmark` to hold the values for the first and bottom mark as found in the individual columns. There are some helper functions defined in section 5.1 which may be used for this. Getting the marks right “by hand” is non-trivial and it may pay off to first take a look at the documentation and implementation of `\balance@columns` below before trying anew.

Table 1: Interface description for `\balance@columns`

`\setemergencystretch`: This is a hook for people who like to play around. It is supposed to set the `\emergencystretch`  $\langle dimen \rangle$  register provided in the new  $\TeX$  3.0. The first argument is the number of columns and the second one is the current `\hsize`. At the moment the default definition is

$4pt \times \#1$ , i.e. the `\hsize` isn’t used at all. But maybe there are better formulae.

`\set@floatcmds`: This is the hook for the experts who like to implement a full float mechanism for the `multicols` environment. The `@` in the name should signal that this might not be easy.

Table 2: The new commands of `multicol.sty` version 1.2. Both commands might be removed if good solutions to these open problems are found. I hope that these commands will prevent that nearly identical style files derived from this one are floating around.

want to try: wait for  $\TeX$  3.0. It has a few features which will make life much easier in multi-column surroundings. Nevertheless we are working here at the

edge of  $\TeX$ s capabilities, really perfect solutions would need a different approach than it was done in  $\TeX$ s page builder.

The text below is nearly un-

changed, I only added documentation at places where new code was added.

## 4 The Implementation

We are now switching to two-column output to show the abilities of this environment (and bad layout decisions).

### 4.1 The documentation driver file

The next bit of code contains the documentation driver file for  $\TeX$ , i.e., the file that will produce the documentation you are currently reading. It will be extracted from this file by the `docstrip` program. Since this is the first code in this file one can produce the documentation simply by running  $\LaTeX$  on the `.dtx` file.

```
1  $\ast$ driver)
2 \documentclass{ltxdoc}
```

We use the `balancingshow` option when loading `multicols` so that full tracing is produced. This has to

be done before the `doc` package is loaded, since `doc` otherwise requires `multicols` without any options.

```
3 \usepackage{multicol}[1999/05/25]
4 \usepackage{doc}
```

First we set up the page layout suitable for this article.

```
5 \setlength{\textwidth}{39pc}
6 \setlength{\textheight}{54pc}
7 \setlength{\parindent}{1em}
8 \setlength{\parskip}{0pt plus 1pt}
9 \setlength{\oddsidemargin}{0pc}
```

# The `rawfonts` package

Alan Jeffrey

v0.01

## Overview

The `rawfonts`  $\LaTeX 2_{\epsilon}$  package is used to provide emulation of  $\LaTeX 2.09$  documents which used low-level commands such as `\tenrm`. If you say:

```
\usepackage{rawfonts}
```

then over sixty fonts such as `\tenrm` will be loaded into  $\LaTeX$ . This has a large overhead, for example this document uses:

```
8221 words of font info for 30 fonts
```

but when run with the `rawfonts` package it uses:

```
19294 words of font info for 73 fonts
```

That is `rawfonts` can cause your document to load twice as many fonts. (This overhead is why  $\LaTeX 2_{\epsilon}$  does not define `\tenrm` and friends by default.)

If you want to only load a small number of fonts, you can use the `only` option, for example to only load `\tenrm` and `\tensf`:

```
\usepackage[only,tenrm,tensf]{rawfonts}
```

The `rawfonts` package is intended mainly for use with  $\LaTeX 2.09$  documents, and you might want to have the package loaded every time you use  $\LaTeX 2_{\epsilon}$  in compatibility mode. In this case, you should add the line:

```
\RequirePackage{rawfonts}
```

to your `latex209.cfg`  $\LaTeX 2.09$  compatibility configuration file.

# The `showkeys` package\*

David Carlisle

1997/06/12

## 1 Introduction

sec:intro

`showkeys.sty` modifies the `\label`, `\ref`, `\pageref`, `\cite`, and `\bibitem` commands so that the ‘internal’ key is printed. The package tries hard to position these labels so that the formatting of the rest of the document is unchanged. `\label` and `\bibitem` cause the key to appear in a box either in the margin, or in a  $\text{\TeX}$  box of zero width, which may possibly over-print other text. The `\ref`, `\pageref` and `\cite` commands print their arguments in small type, raised just above the line, like this: sec:intro. This package works with the `fleqn` option, the packages in the  $\text{AMS-L}\text{\TeX}$  collection, and the `varioref`, `natbib` and `harvard` packages.

## 2 Package Options

options

Some people have commented that the printing of the `\ref` and `\cite` keys is less useful than the printing of the `\label` keys and so `showkeys` now supports two options that can be given in the `\usepackage` command:

**notref** to stop the redefinition of `\ref` and `\pageref`, and related commands from the `varioref` package.

**notcite** to stop the redefinition of `\cite` and related commands from the `harvard` and `natbib` packages.

So if the package is loaded with `\usepackage[notref]{showkeys}` then `\ref` will have its standard definition, but `\label` will print its key argument (usually in the margin).

If you find the printed keys distracting, but don’t want to use the above options to stop them altogether you may use:

**color** Print the keys in a distinguishing colour. The default value is a light grey.

---

\*This file has version number v3.12, last revised 1997/06/12.

The colours may be changed by redefining the following two colours after the package is loaded. `refkey` (also used for `\cite`) and `labelkey` (also used for `\bibitem`). The defaults are:

```
\definecolor{refkey}{gray}{.75}
\definecolor{labelkey}{gray}{.75}
```

If this option is used the `color` package will be loaded.  
The package accepts two further options.

**final** to suppress the action of this package, for ‘final’ versions.

**draft** the normal behaviour of this package.

Clearly there is not much point in entering the `final` option directly in the `\usepackage` command, as just not loading this package would have the same effect, and execute more quickly, however the `final` option may be useful as it may be used once in the `documentclass` command to affect any number of packages that may be loaded. The `draft` option does not do anything, but is there to honour an informal convention that packages have these options in pairs.

### 3 More Examples

examples

The only other similar package that I could find in the macro index, <sup>DMJ:mi</sup>[3], was <sup>DN:sl</sup>`showlabels.sty`, <sup>anon:sk</sup>[1]. After the first draft of this package was written, I found <sup>anon:sk</sup>[2] on my local installation! I think the current package is more robust than [2], but I thought that `showkeys` was rather a good name, so I have stolen it for this file.

e<sup>1</sup>

1. This has `\label` immediately after `\item`.

e<sup>2</sup>

2. This has the `\label` at the end.

A minipage :- { Within environments like this `minipage`, we cannot use `\marginpar`<sup>1</sup>, so the appearance is slightly different. Here is that `enumerate` environment again:

m&e<sup>1</sup> 1. This has `\label` immediately after `\item`.  
m&e<sup>2</sup> 2. This has the `\label` at the end.

Displayed math (without `equation` counter).

$$0 = 0 \quad \text{\code{disp}}$$

Some text referring to the maths on page <sup>disp</sup>2, and the item <sup>e<sup>1</sup></sup>1.

If `showkeys` thinks that the current environment is going to produce an “equation number”, then it does not show the label where the `\label` command occurs,

<sup>1</sup>Actually `\marginpar` is not used at all in this package now.

but tries to put it in the margin, as shown with equation <sup>eq:xx</sup>1. The package ‘knows’ about the standard `equation` and `eqnarray` environments, and also all the numbered alignment environments offered by the AMS $\LaTeX$  package, `amsmath`.

$$1 = 1 \tag{1} \quad \boxed{\text{eq:xx}}$$

$$2 = 2 \tag{2} \quad \boxed{\text{eqnar:a}}$$

$$3 = 3$$

$$4 = 4 \tag{3} \quad \boxed{\text{eqnar:b}}$$

Within a `figure` environment, the `\label` must not come before the `\caption` command. If you place `\label` inside the argument of `\caption` the label will be shown like this:

Figure 1: Within the caption argument. cap:a

If you place `\label` immediately after the `\caption` command it will be shown like this:

Figure 2: Immediately after the caption argument. cap:b

If you place the `\label` command at some random point after the `\caption` command, it may be shown like:

Figure 3: In vertical mode not immediately after a box.

cap:c

## References

GN:s1 [1] Gil Neiger, *showlabels.sty*, Undated package, similar to this one, but shows labels inline, affecting the formatting of the document.

anon:sk [2] Anonymous, *showkeys.sty*, Package, dated 14 May 1988. Very similar to this one, also uses `\marginpar` in outer vertical mode.

DMJ:mi [3] David M. Jones, *TEX Macro Index*, A catalogue of  $\TeX$  macros, including  $\LaTeX$  packages, available from all good  $\TeX$  archives.

## 4 The Macros

1  $\langle$ \*package $\rangle$

First we handle the options. Normally all related comands are defined to show their ‘keys’. But since v3.03 one can specify:

`notref` to stop the redefinition of `\ref` (and `\pageref`, and related commands from `varioref` package),

# The `somedefs` toolkit package

Alan Jeffrey

v0.03

## Overview

This is an example ‘programmers toolkit’ package, for use by package writers. It allows package writers to provide options which switch definitions on and off. For example, a package `fred` might define a large number of commands, including `\foo` and `\baz`, so:

```
\usepackage{fred}
```

would use a lot of memory, even if `\foo` and `\baz` were the only commands needed. However, if the author of `fred` used the `somedefs` package, then the user would be able to say:

```
\usepackage[only,foo,baz]{fred}
```

and only the commands `\foo` and `\baz` would be defined.

To use the `somedefs` package in your own packages or classes, you say:

```
\RequirePackage{somedefs}
```

You can then use four new commands:

- `\UseAllDefinitions` which says that all the commands in the file should be defined.
- `\UseSomeDefinitions` which says that only the commands specified by `\UseDefinition` should be defined.
- `\UseDefinition{<name>}` which says that the command `\name` should be defined.
- `\ProvidesDefinition{<definition>}` which provides one definition, of the form `\definingcommand{\command}...`

For example, the package `fred` could say:

```

\RequirePackage{somedefs}
\UseAllDefinitions
\DeclareOption{only}{\UseSomeDefinitions}
\DeclareOption*{\UseDefinition{\CurrentOption}}
\ProcessOptions
\ProvidesDefinition{\newcommand{\foo}{...}}
\ProvidesDefinition{\newcommand{\baz}{...}}

```

One of the commands `\UseAllDefinitions` or `\UseSomeDefinitions` should always be used. You may have some commands which need other commands, in which case you have to declare the options by hand. For example, if the command `\bar` needs the command `\foo`, you could say:

```

\DeclareOption{bar}{\UseDefinition{bar}\UseDefinition{foo}}

```

For a longer example of the use of the `somedefs` package, look at the `rawfonts` package.

## Implementation

The driver for the documentation you're now reading.

```

1 <*driver>
2 \documentclass{ltxdoc}
3 \begin{document}
4 \DocInput{somedefs.dtx}
5 \end{document}
6 </driver>

```

This is a L<sup>A</sup>T<sub>E</sub>X 2<sub>ε</sub> package.

```

7 <*package>
8 \NeedsTeXFormat{LaTeX2e}
9 \ProvidesPackage{somedefs}[1994/06/01 Toolkit for optional definitions]

```

<pre> \UseSomeDefinitions \UseAllDefinitions   \UseDefinition \ProvidesDefinition   \@providesdefinition   \@provides@definition \@unprovided@definition </pre>	<p>The package works by having <code>\UseDefinition{&lt;name&gt;}</code> define <code>\name</code> to be <code>\@unprovided@definition</code>. If <code>\UseSomeDefinitions</code> has been called, then <code>\ProvidesDefinition</code> looks to see if <code>\name</code> is <code>\@unprovided@definition</code>. If <code>\UseAllDefinitions</code> has been called, then <code>\ProvidesDefinition</code> does nothing. If neither has been called, then <code>\ProvidesDefinition</code> produces an error message.</p> <pre> 10 \def\UseSomeDefinitions{% 11   \let\ProvidesDefinition\@providesdefinition 12 } 13 \def\UseAllDefinitions{% 14   \let\ProvidesDefinition\@firstofone 15 } 16 \def\UseDefinition#1{% </pre>
---	--



```

17 \expandafter\let\csname#1\endcsname\@unprovided@definition
18 }
19 \def\ProvidesDefinition#1{%
20   \PackageError{somedefs}%
21     {No \noexpand\UseSomeDefinitions or \string\UseAllDefinitions}%
22     {The package which used the 'somedefs' package has an error.}%
23 }
24 \def\@providesdefinition#1{\@provides@definition#1\relax
25   \@provides@definition}
26 \def\@provides@definition#1#2#3\@provides@definition{%
27   \ifx#2\@unprovided@definition
28     #1#2#3%
29   \fi
30 }
31 \def\@unprovided@definition{%
32   \PackageError{somedefs}%
33     {Package 'somedefs' error: this command was never defined}%
34     {You have requested a command which does not exist.}%
35 }
36 \@onlypreamble\UseSomeDefinitions
37 \@onlypreamble\UseAllDefinitions
38 \@onlypreamble\UseDefinition
39 \@onlypreamble\ProvidesDefinition
40 \@onlypreamble\@providesdefinition
41 \@onlypreamble\@provides@definition

That's it!
42 \end{package}

```

# The `tabularx` package\*

David Carlisle

1999/01/07

## Abstract

A new environment, `tabularx`, is defined, which takes the same arguments as `tabular*`, but modifies the widths of certain columns, rather than the inter column space, to set a table with the requested total width. The columns that may stretch are marked with the new token `X` in the preamble argument.

This package requires the `array` package.

## 1 Introduction

This package implements a version of the `tabular` environment in which the widths of certain columns are calculated so that the table is a specified width. Requests for such an environment seem to occur quite regularly in `comp.text.tex`.

`tabularx`     `\begin{tabularx}{\langle width \rangle}{\langle preamble \rangle}`

The arguments of `tabularx` are essentially the same as those of the standard `tabular*` environment. However rather than adding space between the columns to achieve the desired width, it adjusts the widths of some of the columns. The columns which are affected by the `tabularx` environment should be denoted with the letter `X` in the preamble argument. The `X` column specification will be converted to `p{\langle some value \rangle}` once the correct column width has been calculated.

## 2 Examples

The following table is set with `\begin{tabularx}{250pt}{|c|X|c|X|} ....`

Multicolumn entry!		THREE	FOUR
one	The width of this column depends on the width of the table. <sup>1</sup>	three	Column four will act in the same way as column two, with the same width.

---

\*This file has version number v2.07, last revised 1999/01/07.

<sup>1</sup>You can now use `\footnote` inside `tabularx`!

If we change the first line to `\begin{tabularx}{300pt}{|c|X|c|X|}` we get:

	Multicolumn entry!	THREE	FOUR
one	The width of this column depends on the width of the table.	three	Column four will act in the same way as column two, with the same width.

### 3 Differences between `tabularx` and `tabular*`

These two environments take the same arguments, to produce a table of a specified width. The main differences between them are:

- `tabularx` modifies the widths of the *columns*, whereas `tabular*` modifies the widths of the inter-column *spaces*.
- `tabular` and `tabular*` environments may be nested with no restriction, however if one `tabularx` environment occurs inside another, then the inner one *must* be enclosed by `{ }`.
- The body of the `tabularx` environment is in fact the argument to a command, and so certain constructions which are not allowed in command arguments (like `\verb`) may not be used.<sup>2</sup>
- `tabular*` uses a primitive capability of  $\TeX$  to modify the inter column space of an alignment. `tabularx` has to set the table several times as it searches for the best column widths, and is therefore much slower. Also the fact that the body is expanded several times may break certain  $\TeX$  constructs.

## 4 Customising the behaviour of `tabularx`

### 4.1 Terminal output

`\tracingtabularx` If this declaration is made, say in the document preamble, then all following `tabularx` environments will print information about column widths as they repeatedly re-set the tables to find the correct widths.

As an alternative to using the `\tracingtabularx` declaration, either of the options `infoshow` or `debugshow` may be given, either in the `\usepackage` command that loads `tabularx`, or as a global option in the `\documentclass` command.

### 4.2 The environment used to typeset the X columns

By default the X specification is turned into `p{<some value>}`. Such narrow columns often require a special format, this may be achieved using the `>` syntax of `array.sty`. So for example you may give a specification of `>\small}X`.

<sup>2</sup>Since Version 1.02, `\verb` and `\verb*` may be used, but they may treat spaces incorrectly, and the argument can not contain an unmatched `{` or `}`, or a `%` character.

<code>\arraybackslash</code>	Another format which is useful in narrow columns is ragged right, however L <sup>A</sup> T <sub>E</sub> X's <code>\raggedright</code> macro redefines <code>\</code> in a way which conflicts with its use in a tabular or array environments. For this reason this package introduces the command <code>\arraybackslash</code> , this may be used after a <code>\raggedright</code> , <code>\raggedleft</code> or <code>\centering</code> declaration. Thus a <code>tabularx</code> preamble may specify <code>&gt;\raggedright\arraybackslashX</code> .
<code>\newcolumntype</code>	These preamble specifications may of course be saved using the command, <code>\newcolumntype</code> , defined in <code>array.sty</code> . Thus we may say <code>\newcolumntype{Y}{&gt;\small\raggedright\arraybackslashX}</code> and then use <code>Y</code> in the <code>tabularx</code> preamble argument.
<code>\tabularxcolumn</code>	The <code>X</code> columns are set using the <code>p</code> column which corresponds to <code>\parbox[t]</code> . You may want them set using, say, the <code>m</code> column, which corresponds to <code>\parbox[c]</code> . It is not possible to change the column type using the <code>&gt;</code> syntax, so another system is provided. <code>\tabularxcolumn</code> should be defined to be a macro with one argument, which expands to the <code>tabular</code> preamble specification that you want to correspond to <code>X</code> . The argument will be replaced by the calculated width of a column. The default is <code>\newcommand{\tabularxcolumn}[1]{p{#1}}</code> . So we may change this with a command such as: <code>\renewcommand{\tabularxcolumn}[1]{&gt;\small}m{#1}</code>

### 4.3 Column widths

Normally all `X` columns in a single table are set to the same width, however it is possible to make `tabularx` set them to different widths. A preamble argument of `>\hsize=.5\hsizeX>\hsize=1.5\hsizeX` specifies two columns, the second will be three times as wide as the first. However if you want to play games like this you should follow the following two rules.

- Make sure that the sum of the widths of all the `X` columns is unchanged. (In the above example, the new widths still add up to twice the default width, the same as two standard `X` columns.)
- Do not use `\multicolumn` entries which cross any `X` column.

As with most rules, these may be broken if you know what you are doing.

### 4.4 If the algorithm fails...

It may be that the widths of the ‘normal’ columns of the table already total more than the requested total width. `tabularx` refuses to set the `X` columns to a negative width, so in this case you get a warning “`X` Columns too narrow (table too wide)”.

The `X` columns will in this case be set to a width of `1em` and so the table itself will be wider than the requested total width given in the argument to the environment. This behaviour of the package can be customised slightly as noted in the documentation of the code section.

# An Extension of the L<sup>A</sup>T<sub>E</sub>X theorem environment\*

Frank Mittelbach  
Electronic Data Systems  
(Deutschland) GmbH  
Eisenstraße 56  
D-65424 Rüsselsheim  
Federal Republic of Germany

March 10, 2004

## Abstract

The macros described in this paper yield an extension of the L<sup>A</sup>T<sub>E</sub>X theorem mechanism. It is designed to satisfy the different requirements of various journals. Thus, the layout of the “theorems” can be manipulated by determining a “style”. This article describes not only the use, but also the definition, of the necessary macros.

## Preface to version 2.2

For L<sup>A</sup>T<sub>E</sub>X 2<sub>ε</sub> this package did not need any fundamental changes. I only modified the messages generated so that theorem layout styles will show up with the `\listfiles` command and cleaned the section on the New Font Selection Scheme since this is now included in L<sup>A</sup>T<sub>E</sub>X.

## Preface to version 2.1

This version is identical to 2.0g described in *TUGboat* 10#3 except for some internal defaults which are now set depending on the used font selection scheme.

This was done to avoid unpleasant surprises if the new font selection scheme is in force. For further details see section ?? and [?].

## 1 Introduction

For our purposes here, “theorems” are labelled enunciations, often set off from the main text by extra space and a font change. Theorems, corollaries, conjectures,

---

\*This file has version number v2.2c, last revised 1995/11/23.

definitions, and remarks are all instances of “theorems”. The “header” of these structures is composed of a label (such as THEOREM or REMARK) and a number which serializes an item in the sequence of items with the same label.

Shortly after the introduction of L<sup>A</sup>T<sub>E</sub>X at the Fachbereich Mathematik in Mainz, the desire to manipulate the layout of “theorems” arose. In Mainz, the following two conventions came into general use:

1. The number of the theorem is shown in the margin.
2. There is a line break at the end of the theorem header.

Additionally, some journals require different formats which depend on the “sort of theorem”: e.g. often remarks and definitions are set in `\upshape`, while `\itshape` is employed for main theorems.

Confronted with these requirements, a theorem environment was developed in Mainz which allows separate determination of the layout of the “theorems sets”, comparable to `\pagestyle`.

## 2 The user interface

### 2.1 Defining new theorem sets

`\newtheorem` As in the original L<sup>A</sup>T<sub>E</sub>X version, the command `\newtheorem` defines a new “theorem set” or “theorem-like structure”. Two required arguments name the new environment and give the text to be typeset with each instance of the new “set”, while an optional argument determines how the “set” is enumerated:

`\newtheorem{foo}{bar}` The theorem set `foo` (whose name is `bar`) uses its own counter.

`\newtheorem{foo2}[foo]{bar2}` The theorem set `foo2` (printed name `bar2`) uses the same counter as the theorem set `foo`.

`\newtheorem{foo3}{bar3}[section]` The theorem set `foo3` (printed name `bar3`) is enumerated within the counter `section`, i.e. with every new `\section` the enumeration begins again with 1, and the enumeration is composed from the section-number and the theorem counter itself.

`\theoremstyle` Additionally, the command `\theoremstyle` can define the layout of various, or all, theorem sets. It should be noted that any theorem set defined by `\newtheorem` is typeset in the `\theoremstyle` that is current at the time of the definition. Thus, the following

<code>\theoremstyle{break}</code>	<code>\newtheorem{Cor}{Corollary}</code>
<code>\theoremstyle{plain}</code>	<code>\newtheorem{Exa}{Example}[section]</code>

leads to the result that the set `Cor` is formatted in the style `break`, while the set `Exa` and all the following ones are formatted in the style `plain`, unless another

`\theoremstyle` follows. Since the definitions installed by `\newtheorem` are global, one also can limit `\theoremstyle` locally by grouping braces.

`\theorembodyfont` The choice of the font for the theorem body is completely independent of the chosen `\theoremstyle`; this has proven to be very advantageous. For example,

```
{\theorembodyfont{\upshape}          \newtheorem{Rem}{Remark}}
```

defines a theorem set `Rem`, which will be set in `\upshape` in the current layout (which in our example is `plain`). As with `\theoremstyle`, the `\theorembodyfont` chosen is that current at the time of `\newtheorem`. If `\theorembodyfont` is not specified or one defines `\theorembodyfont{}`, then the font used will be that defined by the `\theoremstyle`.

`\theoremheaderfont` It is also possible to customize the font used for the theorem headers. This is, however, a global declaration, and therefore there should be at most one `\theoremheaderfont` declaration in the preamble.<sup>1</sup>

`\theorempreskipamount`  
`\theorempostskipamount` Two additional parameters affect the vertical space around the theorem environments: `\theorempreskipamount` and `\theorempostskipamount` define, respectively, the spacing before and after such an environment. These parameters apply for all theorem sets and can be manipulated with the ordinary length macros. They are rubber lengths, ('skips'), and therefore can contain `plus` and `minus` parts.

Since the definition of theorem sets should—most sensibly—be placed in the preamble, we only allow installation there. It is therefore possible to release the memory used here after `\begin{document}`, in order to make room for other applications.

## 2.2 Existing theorem styles

The following theorem styles exist to date:

**plain** This theorem style emulates the original L<sup>A</sup>T<sub>E</sub>X definition, except that additionally the parameters `\theorem...skipamount` are used.

**break** In this style, the theorem header is followed by a line break.

**marginbreak** The theorem number is set in the margin, and there is a line break as in **break**.

**changebreak** Like **break**, but with header number and text interchanged.

**change** Header number and text are interchanged, without a line break.

**margin** The number is set in the left margin, without a line break.

All styles (except **plain**) select `\slshape` as the default `\theorembodyfont`.

---

<sup>1</sup>If it is actually necessary to have different header fonts, one has to define new theorem styles (substituting the desired font) or specify the information directly in the `\newtheorem` declaration (the unclean variant).

## 2.3 Examples

Given the above theorem sets `Cor`, `Exa` and `Rem`, suppose that the preamble also contains the declarations:

```
\theoremstyle{marginbreak} \newtheorem{Lem}[Cor]{Lemma}
\theoremstyle{change}
\theorembodyfont{\itshape} \newtheorem{Def}[Cor]{Definition}

\theoremheaderfont{\scshape}
```

Then the following are some typical examples of the typeset output resulting from their use.

COROLLARY 1

*This is a sentence typeset in the theorem environment `Cor`.*

EXAMPLE 2.1 *This is a sentence typeset in the theorem environment `Exa`.*

REMARK 1 This is a sentence typeset in the theorem environment `Rem`.

2 LEMMA (BEN USER)

*This is a sentence typeset in the theorem environment `Lem`.*

3 DEFINITION (VERY IMPRESSIVE DEFINITION) *This is a sentence typeset in the theorem environment `Def`.*

The last two examples show the effect of the optional argument to a theorem environment (it is the text typeset in parentheses).

## 3 Special Considerations

Theoremheader and body are implemented as a unit. This means that the `\theoremheaderfont` will inherit characteristics of the `\theorembodyfont` in  $\text{\LaTeX} 2\epsilon$ . Thus, if for example `\theorembodyfont` is `\itshape` and `\theoremheaderfont` is `\bfseries` the font selected for the header will have the characteristics ‘bold extended italic’. If this is not desired one should set the `\theoremheaderfont` to something like

```
\theoremheaderfont{\normalfont\bfseries}
```

i.e. supplying all necessary font informations explicitly.

## 4 Acknowledgements

The publication of this set of macros was only possible with the help of Christina Busse (translating the manuscript into English), Joachim Pense (playing the rôle of typist), Chris Rowley (looking everything over) and many others providing useful suggestions.



# The `trace` package\*

Frank Mittelbach  
L<sup>A</sup>T<sub>E</sub>X3 project  
frank.mittelbach@latex-project.org

2001/07/24

## 1 Introduction

When writing new macros one often finds that they do not work as expected (at least I do :-). If this happens and one can't immediately figure out why there is a problem one has to start doing some serious debugging. T<sub>E</sub>X offers a lot of bells and whistles to control what is being traced but often enough I find myself applying the crude command `\tracingall` which essentially means “give me whatever tracing information is available”.

In fact I normally use  $\epsilon$ -T<sub>E</sub>X in such a case, since that T<sub>E</sub>X extension offers me a number of additional tracing possibilities which I find extremely helpful. The most important ones are `\tracingassigns`, which will show you changes to register values and changes to control sequences when they happen, and `\tracinggroups`, which will tell you what groups are entered or left (very useful if your grouping got out of sync).

So what I really write is

```
\tracingassigns=1\tracinggroups=1\tracingall
```

That in itself is already a nuisance (since it is a mouthful) but there is a worse catch: when using `\tracingall` you do get a awful lot of information and some of it is really useless.

For example, if L<sup>A</sup>T<sub>E</sub>X has to load a new font it enters some internal routines of NFSS which scan font definition tables etc. And 99.9% of the time you are not at all interested in that part of the processing but in the two lines before and the five lines after. However, you have to scan through a few hundred lines of output to find the lines you need.

Another example is the `calc` package. A simple statement like `\setlength \linewidth {1cm}` inside your macro will result in

```
\setlength ->\protect \setlength  
\relax
```

---

\*This file has version number v1.1b, last revised 2001/07/24.

```

\setlength ->\calc@assign@skip

\calc@assign@skip ->\calc@assign@generic \calc@Askip \calc@Bskip

\calc@assign@generic #1#2#3#4->\let \calc@A #1\let \calc@B #2\expandafter \calc
@open \expandafter (#4!\global \calc@A \calc@B \endgroup #3\calc@B
#1<-\calc@Askip
#2<-\calc@Bskip
#3<-\linewidth
#4<-1cm
{\let}
{\let}
{\expandafter}
{\expandafter}

\calc@open (->\beginngroup \aftergroup \calc@initB \beginngroup \aftergroup \calc
@initB \calc@pre@scan
{\beginngroup}
{\aftergroup}
{\beginngroup}
{\aftergroup}

\calc@pre@scan #1->\ifx (#1\expandafter \calc@open \else \ifx \widthof #1\expan
dafter \expandafter \expandafter \calc@textsize \else \calc@numeric \fi \fi #1
#1<-1
{\ifx}
{false}
{\ifx}
{false}

\calc@numeric ->\afterassignment \calc@post@scan \global \calc@A
{\afterassignment}
{\global}
{\fi}
{\fi}

\calc@post@scan #1->\ifx #1!\let \calc@next \endgroup \else \ifx #1+\let \calc@
next \calc@add \else \ifx #1-\let \calc@next \calc@subtract \else \ifx #1*\let
\calc@next \calc@multiplyx \else \ifx #1/\let \calc@next \calc@dividex \else \i
fx #1)\let \calc@next \calc@close \else \calc@error #1\fi \fi \fi \fi \fi \fi \
calc@next
#1<-!
{\ifx}
{true}
{\let}
{\else}
{\endgroup}
{restoring \calc@next=undefined}

\calc@initB ->\calc@B \calc@A
{\skip44}
{\global}
{\endgroup}
{restoring \skip44=0.0pt}

\calc@initB ->\calc@B \calc@A

```

```
{\skip44}  
{\dimen27}
```

Do you still remember what I was talking about?

No? We're trying to find a problem in macro code without having to scan too many uninteresting lines. To make this possible we have to redefine a number of key commands to turn tracing off temporarily in the hope that this will reduce the amount of noise during the trace. For example, if we change one of the `calc` internals slightly, the above tracing output can be reduced to:

```
\setlength ->\protect \setlength  
{\relax}  
  
\setlength ->\calc@assign@skip  
  
\calc@assign@skip ->\calc@assign@generic \calc@Askip \calc@Bskip  
  
\calc@assign@generic #1#2#3#4->\let \calc@A #1\let \calc@B #2\expandafter \calc  
@open \expandafter (#4!\global \calc@A \calc@B \endgroup #3\calc@B  
#1<-\calc@Askip  
#2<-\calc@Bskip  
#3<-\linewidth  
#4<-1cm  
{\let}  
{\let}  
{\expandafter}  
{\expandafter}  
  
\calc@open (->\begingroup \conditionally@traceoff \aftergroup \calc@initB \begi  
ngroup \aftergroup \calc@initB \calc@pre@scan  
{\begingroup}  
  
\conditionally@traceoff ->\tracingrestores \z@ \tracingcommands \z@ \tracingpag  
es \z@ \tracingmacros \z@ \tracingparagraphs \z@  
{\tracingrestores}  
{\tracingcommands}  
{restoring \tracingrestores=1}  
  
\calc@initB ->\calc@B \calc@A  
{\skip44}  
{\dimen27}
```

Still a lot of noise but definitely preferable to the original case.

I redefined those internals that I found most annoyingly noisy. There are probably many others that could be treated in a similar fashion, so if you think you found one worth adding please drop me a short note.

\* \* \*

```
\traceon     The package defines the two macros \traceon and \traceoff to uncondi  
\traceoff   tionally turn tracing on or off, respectively. \traceon is like \tracingall but  
            additionally adds \tracingassigns and \tracinggroups if the  $\epsilon$ -TeX program
```

(in extended mode) is used. And `\traceoff` will turn tracing off again, a command which is already badly missing in plain  $\TeX$ , since it is often not desirable to restrict the tracing using extra groups in the document.

```
\conditionally@traceon
\conditionally@traceoff
```

There are also two internal macros that turn tracing on and off, but only if the user requested tracing in the first place. These are the ones that are used internally within the code below.

Since the package overwrites some internals of other packages you should load it as the last package in your preamble using `\usepackage{trace}`.

The package offers one option (`logonly`) that suppresses terminal output during tracing. This is useful if the  $\TeX$  implementation used gets rather slow when writing a lot of information to the terminal.

## 2 A sample file

The following small test file shows the benefits of the `trace` package. If one uncomments the line loading the package, the amount of tracing data will be drastically reduced. Without the `trace` package we get 6594 lines in the log file; adding the package will reduce this to 1618 lines.

```
\documentclass{article}
\usepackage{calc}
%\usepackage{trace} % uncomment to see difference

\begin{document}
\ifx\traceon\undefined \tracingall \else \traceon \fi

\setlength\linewidth{1cm}

$foo=\bar a$

\small \texttt{\$} \stop
```

## 3 Implementation

This package is for use with  $\LaTeX$  (though something similar could be produced for other formats).

```
1 (*package)
2 \NeedsTeXFormat{LaTeX2e}[1998/12/01]
```

The package has one option that suppresses tracing on the terminal, i.e., if used will not set `\tracingonline` to one. This has been added in version 1.1a since some  $\TeX$  implementations get rather slow when outputting to a terminal.

```
3 \DeclareOption{logonly}
4   {\let\tracingonline@p\z@}
```

The default is showing the tracing information on the terminal.

```
5 \let\tracingonline@p\@ne
6 \ProcessOptions\relax
```

# The `varioref` package\*

Frank Mittelbach

2001/09/04

## Abstract

This package defines the commands `\vref`, `\vpageref`, `\vrefrange`, and `\vpagerefrange` for L<sup>A</sup>T<sub>E</sub>X 2<sub>ε</sub>. `\vref` is similar to `\ref` but adds an additional page reference, like ‘on the facing page’ or ‘on page 27’ whenever the corresponding `\label` is not on the same page. The command `\vpageref` is a variation to `\pageref` with a similar functionality. The `\v...range` commands take two labels as arguments and produce strings which depend on whether or not these labels fall onto a single page or on different pages. Generated strings are customizable so that these commands are usable with various languages.

## 1 Introduction

In many cases it is helpful when referring to a figure or table to put both a `\ref` and a `\pageref` command into the document especially when there are one or more pages between the reference and the object. Therefore some people use a command like

```
\newcommand{\fullref}[1]{\ref{#1} on page~\pageref{#1}}
```

which reduces the number of key strokes, necessary to make such a complete reference. But since one never knows where the referenced object finally falls, using such a device may result in a page reference to the current page which is disturbing and therefore should be avoided.

## 2 The user interface

`\vref` The implementation of `\vref` below produces only a `\ref` when reference and `\label` are on the same page. It will additionally produce one of the strings ‘on the facing page’, ‘on the preceding page’, or ‘on the following page’, if label and reference differ by one and it will produce both `\ref` and `\pageref` when the difference is larger. The word ‘facing’ is used when label and reference both fall onto a double spread. However, if a special page numbering scheme is used instead

---

\*This file has version number v1.3c, last revised 2001/09/04.

of the usual arabic numbering (e.g., `\pagenumbering{roman}`) then there will be no distinction between one or many pages off.

`\vpageref` Sometimes one wants to refer only to page number and again such a reference should normally be suppressed if we are referring to the current page. For this purpose the package defines the `\vpageref` command. It will produce the same strings as `\vref` except that it doesn't start with the `\ref` and except that it will produce the string that is saved in `\reftextcurrent` if label and reference fall onto the same page. By defining `\reftextcurrent` to produce "on this page" or something similar, we can avoid that

```
... see the example \vpageref{ex:foo} which shows ...
```

comes out as "... see the example which shows ...", which could be misleading.

You can put a space in front of `\vpageref` it will be ignored if the command doesn't produce any text at all.

But in fact `\vpageref` allows even more control. It has two optional arguments. With the first one, one can specify the text that should be used if label and reference fall on the same page. This is very helpful if both are near to each other, so that they may or may not be separated by a page break. In such a case we usually know (!) whether the reference is before or after the label so that we can say something like

```
... see the example \vpageref[above]{ex:foo} which shows ...
```

which will then come out as "... see the example above which shows ..." if we are still on the same page, but as "... see the example on the page before which shows ..." (or something similar depending on the settings of the `\reftext..before` commands) when there was a page break in the meantime. One warning however, if you use `\vpageref` with the optional argument to refer to a figure or table, keep in mind that depending on the float placement parameters the float may show up on top of the current page and therefore before the reference even if it came after it in the source file.

But maybe you prefer to say "... see the above example" if example and reference fall onto the same page, i.e., reverse the word order. In fact, in some languages the word order automatically changes in that case. To allow for this variation the second optional argument can be used. It specifies the text preceding the generated reference if object and reference do not fall onto the same page. Thus one would write

```
... see the \vpageref[above example][example]{ex:foo}
which shows ...
```

to achieve the desired effect.

## 2.1 Additions in 1998

`\vrefrange` This command is similar to `\vref` but it takes two mandatory arguments denoting

a range to refer to (e.g., a sequences of figures or a sequence of equations, etc.). So if `fig:a` is your first figure in the sequence and `fig:c` your last you can write

```
... see figures \vrefrange{fig:a}{fig:c} ...
```

which would then be formatted as

```
... see figures 3.4 to 3.6 on pages 23–24 ...
```

or, if they happen to all fall onto the next page, as

```
... see figures 3.4 to 3.6 on the following page ...
```

i.e., the command is deciding what to say depending on where the two labels are placed in relation to each other; it is essentially implemented using `\vpagerefrange` described below. The optional argument the command may take is the text to use in case both labels are placed on the current page.

`\vpagerefrange`

This command is similar to `\vpageref` but takes two mandatory arguments which are two labels denoting a range. If both labels fall onto the same page, the command acts exactly like `\vpageref` (with a single label), otherwise it produces something like “on pages 15–18” (see customization possibilities below). The optional argument it may take is the text to use in case both labels are placed on the current page.

`\vrefpagenum`

This macro is provided to allow the user to write their own small commands which implement functions similar to those provided by the two previous commands. It takes two arguments: the second is a label (i.e., as used in `\label` or `\ref`) and the first is an arbitrary command name (make sure you use our own) that receives the page number related to this label. So if you have two (or more) labels you could retrieve their page numbers, compare them and then decide what to print. For example, the following not very serious definition (also using the `ifthen` package)

```
\newcommand\amusingversion[2]{%
  \vrefpagenum\firstnum{#1}%
  \vrefpagenum\secondnum{#2}%
  the definition%
  \ifthenelse{\equal\firstnum\secondnum}%
    {s of \ref{#1} and \ref{#2} \vpageref{#1}}%
    { of \ref{#1} \vpageref{#1} and of \ref{#2} \vpageref{#2}}%
}
```

```
...\amusingversion{foo}{bar}
```

will print something like

```
... the definitions of 3 and 4 on the previous page
```

in the case both labels are on the same page but something like

```
... the definition of 3 on the next page and of 4 on page 13
```

in case the are on different pages.

## 2.2 Additions in 2001

The user commands `\vref`, `\vpageref`, and `\vpagerefrange` all work by first removing any space on their left and then inserting some space of their own (`\vref`, for example, a nonbreakable space). That seemed like a good idea back then, but it has the disadvantage that you can't use these macros in situations where you definitely do not want any space before the generated text. E.g., in situations like (`\vref{foo} . . .`) you end up with a space after the open parenthesis.

`\vref*`            Since it is too late to change the default behaviour I've added star versions  
`\vpageref*`       of the macros which do not add any space before the generated text (they do  
`\vpagerefrange*` nevertheless remove space at the left).

## 3 Customization

The package supports all options defined by the babel package to translate the fixed strings into other languages than English. (Some languages need updating, however.) It also supports languages currently not in babel; check the section on options later on. You can also modify some or all of the strings by redefining the following commands. Backward references use `\reftextbefore` if the label is on the preceding page but invisible and `\reftextfacebefore` if it is on the facing page (i.e., if the current page number is odd). Similarly `\reftextafter` is used when the label comes on the next page but one has to turn the page and `\reftextfaceafter` if it is on the following but facing page.

In fact, `\reftextface . . .` is used only if the user or the document class specified two-sided printing.

`\reftextbefore`  
`\reftextfacebefore`  
`\reftextafter`  
`\reftextfaceafter`

`\reftextfaraway`    Finally we have `\reftextfaraway` which is used whenever label and reference differ by more than one or when they aren't numeric. This macro is a bit different because it takes one argument, the symbolic reference string so that one can make use of `\pageref` in its replacement text.

`\vreftextvario`      To allow a bit random variation in the generated strings one can use the command `\reftextvario` inside the string macros. It takes two arguments and selects one or the other for printing depending on the number of already seen `\vref` or `\vpageref` commands. As an example see the definitions of `\reftextbefore` etc. on page 7.

### 3.1 Additions in 1998

The commands `\vrefrange` and `\vpagerefrange` produce their text using two macros described below. By redefining them one can modify the results to accommodate special requirements.

They both take two mandatory arguments denoting the first and the last label of the range.

`\reftextpagerange`    This macro produces text that describes the page range of the two labels, e.g., the default for English is “on pages~`\pageref{#1}`--`\pageref{#2}`”.

`\reftextlabelrange`    This macro produces text that describes the range of figures, tables, or whatever the labels refer to, the default for English is “`\ref{#1}` to~`\ref{#2}`”.



## 4 Options

As mentioned above the package supports all standard options offered by the Babel system to customize the strings produced. In addition it offers the option `draft` to turn error messages into warnings during development. The default `final` produces error message when a generated string falls onto a page boundary (see next section).

## 5 A few warnings

Defining commands like the ones described above poses some interesting problems. Suppose, for example, that a generated text like ‘on the next page’ gets broken across pages. If this happens it is very difficult to find an acceptable solution and in fact can even result in a document that will always change from one state to another (i.e., inserting one string, finding that this is wrong, inserting another string on the next run which makes the first string correct again, inserting ...). The current implementation of `varioref` therefore issues an error message whenever the generated text is broken across page boundaries, e.g.,

```
table 5 on the current <page break> page
```

would result in an error, which needs to be resolved by the user by replacing the `\vref` command with an ordinary `\ref` just before the final run. This is not completely satisfactory but in such case no solution really is. During document preparation, while one is still changing the text, such error messages can be turned into warnings by placing a `\vrefwarning` command in the preamble. This is equivalent to specifying “draft” as an option to the package. `\vrefshowerrors` ensures that `varioref` stops when detecting a possible loop. This is the default and equivalent to specifying “final” as an option.

At the end final a warning: every use of `\vref` will internally generate two macro names to keep track of the string positions within the document. As a result you may run out of name space or main memory if you make heavy use of this macro on a small `TEX` installation. For this reason the primitive command `\fullref` is also provided. This command can be used whenever you know for sure that label and reference can’t fall onto nearby pages.

## 6 The documentation driver file

The next bit of code contains the documentation driver file for `TEX`, i.e., the file that will produce the documentation you are currently reading. It will be extracted from this file by the `docstrip` program.

```
1 <*driver>
2 \documentclass{ltxdoc}
3 \usepackage{varioref}
4 \GetFileInfo{varioref.sty}
5 \setlength\hfuzz{1pt} % ignore slight overfulls
```

# A New Implementation of L<sup>A</sup>T<sub>E</sub>X's `verbatim` and `verbatim*` Environments.

Rainer Schöpf  
Zentrum für Datenverarbeitung  
der Universität Mainz  
Anselm-Frantz-von-Bentzel-Weg 12  
D-55099 Mainz  
Federal Republic of Germany  
Internet: `Schoepf@Uni-Mainz.DE`

Bernd Raichle  
Stettener Str. 73  
D-73732 Wäldenbronn  
Federal Republic of Germany  
Internet: `raichle@azu.Informatik.Uni-Stuttgart.DE`

Chris Rowley  
The Open University  
Parsifal College  
Finchley Road  
London NW3 7BG, UK  
Internet: `C.A.Rowley@open.ac.uk`

2001/03/12

## Abstract

This package reimplements the L<sup>A</sup>T<sub>E</sub>X `verbatim` and `verbatim*` environments. In addition it provides a `comment` environment that skips any commands or text between `\begin{comment}` and the next `\end{comment}`. It also defines the command `verbatiminput` to input a whole file verbatim.

## 1 Usage notes

L<sup>A</sup>T<sub>E</sub>X's `verbatim` and `verbatim*` environments have a few features that may give rise to problems. These are:

- Due to the method used to detect the closing `\end{verbatim}` (i.e. macro parameter delimiting) you cannot leave spaces between the `\end` token and `{verbatim}`.
- Since TeX has to read all the text between the `\begin{verbatim}` and the `\end{verbatim}` before it can output anything, long verbatim listings may overflow TeX's memory.

Whereas the first of these points can be considered only a minor nuisance the other one is a real limitation.

This package file contains a reimplementation of the `verbatim` and `verbatim*` environments which overcomes these restrictions. There is, however, one incompatibility between the old and the new implementations of these environments: the old version would treat text on the same line as the `\end{verbatim}` command as if it were on a line by itself.

**This new version will simply ignore it.**

(This is the price one has to pay for the removal of the old `verbatim` environment's size limitations.) It will, however, issue a warning message of the form

```
LaTeX warning: Characters dropped after \end{verbatim*}!
```

This is not a real problem since this text can easily be put on the next line without affecting the output.

This new implementation also solves the second problem mentioned above: it is possible to leave spaces (but *not* begin a new line) between the `\end` and the `{verbatim}` or `{verbatim*}`:

```
\begin {verbatim*}
  test
  test
\end {verbatim*}
```

Additionally we introduce a `comment` environment, with the effect that the text between `\begin{comment}` and `\end{comment}` is simply ignored, regardless of what it looks like. At first sight this seems to be quite different from the purpose of verbatim listing, but actually the implementation of these two concepts turns out to be very similar. Both rely on the fact that the text between `\begin{...}` and `\end{...}` is read by TeX without interpreting any commands or special characters. The remaining difference between `verbatim` and `comment` is only that the text is to be typeset in the first case and to be thrown away in the latter. Note that these environments cannot be nested.

`\verbatiminput` is a command with one argument that inputs a file verbatim, i.e. the command `verbatiminput{xx.yy}` has the same effect as

```
\begin{verbatim}
<Contents of the file xx.yy>
\end{verbatim}
```

This command has also a `*`-variant that prints spaces as `␣`.

## 2 Interfaces for package writers

The `verbatim` environment of L<sup>A</sup>T<sub>E</sub>X 2<sub>ε</sub> does not offer a good interface to programmers. In contrast, this package provides a simple mechanism to implement similar features, the `comment` environment implemented here being an example of what can be done and how.

### 2.1 Simple examples

It is now possible to use the `verbatim` environment to define environments of your own. E.g.,

```
\newenvironment{myverbatim}%
  {\endgraf\noindent MYVERBATIM:%
  \endgraf\verbatim}%
  {\endverbatim}
```

can be used afterwards like the `verbatim` environment, i.e.

```
\begin {myverbatim}
  test
  test
\end {myverbatim}
```

Another way to use it is to write

```
\let\foo=\comment
\let\endfoo=\endcomment
```

and from that point on environment `foo` is the same as the `comment` environment, i.e. everything inside its body is ignored.

You may also add special commands after the `\verbatim` macro is invoked, e.g.

```
\newenvironment{myverbatim}%
  {\verbatim\myspecialverbatimsetup}%
  {\endverbatim}
```

though you may want to learn about the hook `\every@verbatim` at this point. However, there are still a number of restrictions:

1. You must not use the `\begin` or the `\end` command inside a definition, e.g. the following two examples will *not* work:

```
\newenvironment{myverbatim}%
  {\endgraf\noindent MYVERBATIM:%
  \endgraf\begin{verbatim}}%
  {\end{verbatim}}
\newenvironment{fred}
  {\begin{minipage}{30mm}\verbatim}
  {\endverbatim\end{minipage}}
```

If you try these examples, T<sub>E</sub>X will report a “runaway argument” error. More generally, it is not possible to use `\begin... \end` or the related environments in the definition of the new environment. Instead, the correct way to define this environment would be

```
\newenvironment{fred}
{\minipage{30mm}\verbatim}
{\endverbatim\endminipage}
```

2. You *cannot* use the `verbatim` environment inside user defined *commands*; e.g.,

```
\newcommand{\verbatimfile}[1]%
\begin{verbatim}\input{#1}\end{verbatim}
```

does *not* work; nor does

```
\newcommand{\verbatimfile}[1]{\verbatim\input{#1}\endverbatim}
```

3. The name of the newly defined environment must not contain characters with category code other than 11 (letter) or 12 (other), or this will not work.

## 2.2 The interfaces

Let us start with the simple things. Sometimes it may be necessary to use a special typeface for your verbatim text, or perhaps the usual computer modern typewriter shape in a reduced size.

You may select this by redefining the macro `\verbatim@font`. This macro is executed at the beginning of every verbatim text to select the font shape. Do not use it for other purposes; if you find yourself abusing this you may want to read about the `\every@verbatim` hook below.

By default, `\verbatim@font` switches to the typewriter font and disables the ligatures contained therein.

There is a hook (i.e. a token register) called `\every@verbatim` whose contents are inserted into T<sub>E</sub>X’s mouth just before every verbatim text. Please use the `\addto@hook` macro to add something to this hook. It is used as follows:

```
\addto@hook<name of the hook>{\<commands to be added>}
```

After all specific setup, like switching of category codes, has been done, the `\verbatim@start` macro is called. This starts the main loop of the scanning mechanism implemented here. Any other environment that wants to make use of this feature should execute this macro as its last action.

These are the things that concern the start of a verbatim environment. Once this (and other) setup has been done, the code in this package reads and processes characters from the input stream in the following way:

1. Before the first character of an input line is read, it executes the macro `\verbatim@startline`.
2. After some characters have been read, the macro `\verbatim@addtoline` is called with these characters as its only argument. This may happen several times per line (when an `\end` command is present on the line in question).
3. When the end of the line is reached, the macro `\verbatim@processline` is called to process the characters that `\verbatim@addtoline` has accumulated.
4. Finally, there is the macro `\verbatim@finish` that is called just before the environment is ended by a call to the `\end` macro.

To make this clear let us consider the standard `verbatim` environment. In this case the three macros above are defined as follows:

1. `\verbatim@startline` clears the character buffer (a token register).
2. `\verbatim@addtoline` adds its argument to the character buffer.
3. `\verbatim@processline` typesets the characters accumulated in the buffer.

With this it is very simple to implement the `comment` environment: in this case `\verbatim@startline` and `\verbatim@processline` are defined to be no-ops whereas `\verbatim@addtoline` discards its argument.

Let's use this to define a variant of the `verbatim` environment that prints line numbers in the left margin. Assume that this would be done by a counter called `VerbatimLineNo`. Assuming that this counter was initialized properly by the environment, `\verbatim@processline` would be defined in this case as

```
\def\verbatim@processline{%
  \addtocounter{VerbatimLineNo}{1}%
  \leavevmode
  \llap{\theVerbatimLineNo\ \hskip\@totalleftmargin}%
  \the\verbatim@line\par}
```

A further possibility is to define a variant of the `verbatim` environment that boxes and centers the whole verbatim text. Note that the boxed text should be less than a page otherwise you have to change this example.

```
\def\verbatimboxed#1{\begingroup
  \def\verbatim@processline{%
    {\setbox0=\hbox{\the\verbatim@line}%
     \hsize=\wd0
     \the\verbatim@line\par}}%
  \setbox0=\vbox{\parskip=0pt\topsep=0pt\partopsep=0pt
    \verbatiminput{#1}}%
  \begin{center}\fbox{\box0}\end{center}%
\endgroup}
```

As a final nontrivial example we describe the definition of an environment called `verbatimwrite`. It writes all text in its body to a file whose name is given as an argument. We assume that a stream number called `\verbatim@out` has already been reserved by means of the `\newwrite` macro.

Let's begin with the definition of the macro `\verbatimwrite`.

```
\def\verbatimwrite#1{%
```

First we call `\@bsphack` so that this environment does not influence the spacing. Then we open the file and set the category codes of all special characters:

```
\@bsphack
\immediate\openout \verbatim@out #1
\let\do\@makeother\dospecials
\catcode'\^M\active
```

The default definitions of the macros

```
\verbatim@startline
\verbatim@addtoline
\verbatim@finish
```

are also used in this environment. Only the macro `\verbatim@processline` has to be changed before `\verbatim@start` is called:

```
\def\verbatim@processline{%
\immediate\write\verbatim@out{\the\verbatim@line}}%
\verbatim@start}
```

The definition of `\endverbatimwrite` is very simple: we close the stream and call `\@esphack` to get the spacing right.

```
\def\endverbatimwrite{\immediate\closeout\verbatim@out\@esphack}
```

### 3 The implementation

The very first thing we do is to ensure that this file is not read in twice. To this end we check whether the macro `\verbatim@@@` is defined. If so, we just stop reading this file. The 'package' guard here allows most of the code to be excluded when extracting the driver file for testing this package.

```
1 <*package>
2 \NeedsTeXFormat{LaTeX2e}
3 \ProvidesPackage{verbatim}
4 [2001/03/12 v1.5p LaTeX2e package for verbatim enhancements]
5 \ifundefined{verbatim@@@}{\endinput}
```

We use a mechanism similar to the one implemented for the `\comment... \endcomment` macro in  $\mathcal{A}\mathcal{M}\mathcal{S}\text{-}\mathcal{T}\mathcal{E}\mathcal{X}$ : We input one line at a time and check if it contains the `\end{...}` tokens. Then we can decide whether we have reached the end of the verbatim text, or must continue.

# The `xr` package\*

David Carlisle<sup>†</sup>  
carlisle@cs.man.ac.uk

1994/05/28

This package implements a system for eXternal References.

If one document needs to refer to sections of another, say `aaa.tex`, then this package may be loaded in the main file, and the command `\externaldocument{aaa}` given in the preamble.

Then you may use `\ref` and `\pageref` to refer to anything which has been given a `\label` in either `aaa.tex` or the main document. You may declare any number of such external documents.

If any of the external documents, or the main document, use the same `\label` then an error will occur as the label will be multiply defined. To overcome this problem `\externaldocument` has an optional argument. If you declare `\externaldocument[A-]{aaa}` Then all references from `aaa` are prefixed by `A-`. So for instance, if a section of `aaa` had `\label{intro}`, then this could be referenced with `\ref{A-intro}`. The prefix need not be `A-`, it can be any string chosen to ensure that all the labels imported from external files are unique. Note however that if your style declares certain active characters (`:` in French, `"` in German) then these characters can not usually be used in `\label`, and similarly may not be used in the optional argument to `\externaldocument`.

## 1 The macros

1 `\package`

Check for the optional argument.

2 `\def\externaldocument{\@ifnextchar[\XR@{\XR@[]}]}`

Save the optional prefix. Start processing the first aux file.

3 `\def\xR@[#1]#2{%`

4 `\makeatletter`

5 `\def\xR@prefix{#1}%`

6 `\xR@next#2.aux\relax\}}`

Process the next aux file in the list and remove it from the head of the list of files to process.

---

\*This file has version number v5.02, last revised 1994/05/28.

<sup>†</sup>The Author of Versions 1–4 was Jean-Pierre Druchbert



# The `xspace` package\*

David Carlisle

1997/10/13

## Abstract

`\xspace` should be used at the end of a macro designed to be used mainly in text. It adds a space unless the macro is followed by certain punctuation characters.

## 1 Introduction

After `\newcommand{\gb}{Great Britain\xspace}`

`\gb` is a very nice place to live.

Great Britain is a very nice place to live.

`\gb`, a small island off the coast of France.

Great Britain, a small island off the coast of France.

`\xspace` saves the user from having to type `\_` or `{}` after most occurrences of a macro name in text. However if either of these constructions follows `\xspace`, a space is not added by `\xspace`. This means that it is safe to add `\xspace` to the end of an existing macro without making too many changes in your document.

Sometimes `\xspace` may make the wrong decision, and add a space when it is not required. In these cases follow the macro with `{}`, as this has the effect of suppressing the space.

Note that this package must be loaded *after* any language (or other) packages that make punctuation characters ‘active’.

## 2 The Macros

1 `\package`

`\xspace` `\xspace` just looks ahead, and then calls `\@xspace`.

2 `\DeclareRobustCommand\xspace{\futurelet\@let@token\@xspace}`

`\@xspace` If the next token is one of a specified list of characters, do nothing, otherwise add a space. If you often use a different punctuation character, add the appropriate line (do not forget the `\fi` at the end!)

---

\*This file has version number v1.06, last revised 1997/10/13.