# The widetable package

Claudio Beccari*

Version number v.1.4; last revision 2017/09/09.

## Contents



**Abstract**

This package allows to typeset tables of specified width, provided they fit in one page. Instead of introducing an infinite stretching glue, which has an unsymmetrical effect in standard LaTeX, here the `\tabcolsep` dimension is computed so as to have the table come out with the proper width.

## 1 Legalese



*`claudio dot beccari at gmail dot com`

## 2 Introduction

It is well known that when the standard environment `tabular*` is opened with a specified width, it is necessary to introduce in the delimiter declaration `@{...}` of (possibly) the first cell of the model row a declaration such as

```
\extracolsep{\fill}
```

in addition to other possible printable delimiters, such as vertical lines, and other fixed spacing commands. The effect is that the extra stretchable glue operates only on the left of each cell *after* (to the *right* of) the cell that received the declaration; the first cell will never get larger in spite of the presence of this glue.

Another package, `tabularX`, normally distributed by the LaTeX 3 Team with every version of the TeX system distribution, allows to create expandable cells, provided they contain only text. These expandable cells are identified with the column identifier `X`; this identifier defines a paragraph-like cell, the width of which gets determined after some runs of the typesetter on the same source tabular material, so as to find out the correct width of the textual columns.

The approach here is a little bit different: the cell contents need not be textual and no cell width is determined in one or more runs of the typesetter; instead the inter column glue is determined so as to fill every cell on both sides with the proper space. The macros contained in this package are insensitive to the particular kind of cell descriptors and to the presence of multiple `\multicolumn` commands. It proved to work properly also if the `array` package extensions are used. Nevertheless if multiple `\multirow` commands in different rows "interlace" the columns thei work on, poor results are obtained, and sometimes no result at all is obtained except a warning message.

On the other hand, as well as for `tabularX`, it needs to typeset the table three times; the first two times with standard values for the inter column glue `\tabcolsep`, in order to find the exact parameters of the linear dependence of the table width from the value of that glue; then executes some computations so as to extrapolate the final correct value of `\tabcolsep`, and on the third run it eventually typesets the table with the specified width.

The time increase needed for these three table typesettings is in general rather negligible, nevertheless if a specific document contained many dozens of such tables, the compilation time might become perceivable.

It might be noticed that in order to perform the necessary computations a fractional division algorithm must be used; since 2009 any TeX installation uses the ε-TeX extensions; therefore fractional division is not any more an issue as it was in previous versions of this package.

## 3 Usage

This package issues an error message only in case the environment includes other unhidden environment; this is explained in the Implementation section. Here it is

assumed that the table is first typeset to its natural width; should it appear too small, and should it be typeset at a larger width, for example by filling the total `\linewidth` available at that specific point, then and only then the `tabular` environment is changed to `widetable`. Should the initial table be moderately larger than the `\linewidth`, then it might be shrunk to `\linewidth` with `widetable`, provided there are enough columns, and therefore delimiters, to be reduced in size. Of course it's impossible to typeset any table with any negative value of `\tabcolsep`; or better, it is possible, but the result in general is very messy.

In other words `widetable` should be used as a second resort, so as to correct some typesetting features not considered aesthetically acceptable.

The syntax for the use of the environment `widetable` is the same as that of the `tabular*` environment; the only difference is the name. Therefore one has to specify:

`\begin{widetable}{`⟨*width*⟩`}[`⟨*alignment*⟩`]{`⟨*column descriptors*⟩`}`
⟨*row of cells*⟩`\\`
⟨*row of cells*⟩`\\`
...
⟨*row of cells*⟩`\\`
⟨*row of cells*⟩`\\`
`\end{widetable}`

# 4 The method

The principle on which this little package is based is the following; suppose a certain table is typeset with an inter column glue $t_0 = 0$ and that its width turns out to be $l_0$; suppose the same tabular material is typeset again with an inter column glue $t_1 > 0$ so that the table table gets as large as $l_1 > l_0$. Then, if the table has to be as wide as $l$ the inter column glue must equal the value

$$t = \frac{l - l_0}{l_1 - l_0} \cdot t_1$$

Therefore we need to run the typesetting of the same tabular material with the two values of the inter column glue set to zero and to $t_1$, respectively, so as to find the widths $l_0$ and $l_1$. Afterwards it has to determine the correct final value $t$, and typeset once again the same tabular material for the last time.

Of course the first two runs must put their results into suitable boxes so as to avoid outputting them into the output file, while at the same time allowing to record the width of the enclosing boxes.

# 5 The long division algorithm

In previous versions we provided a fractional length division macro to perform such computations; the subtractions and multiplication could be done with regular

primitive commands of the TEX engine; but the fractional division required a special long division algorithm.

With the ε-TEX extensions to the typesetting engine (already provided in `xetex` and `latex`, therefore available in `XeLaTeX` and `LuaLaTeX`) there is no need to create any long division macro, because such computation is already provided as a primitive command.

# 6 Using the ε-TEX facilies

In facts the ε-TEX extension provides the scaling operation: given the length $L_1$ and two homogeneous quantities $X_1$ and $X_2$ (where such quantities may be either two integer numbers, or two dimensions), such scaling operation scales $L_1$ to $L_2$ by computing

$$L_2 = L_1 \cdot \frac{X_2}{X_1}$$

The intermediate results are actually done with integer arithmetics (internally length are an integer number of scaled points) but are done in double words so as to avoid underflow and overflows almost always. Some unusual situations might exist where underflows or overflows may occur, but they must be very unusual, and very unlikely to happen for the calculations of this package. In case of overflow a `\maxdimen` value is automatically obtained and computations go on as possible, may be giving rise to other errors or warnings, for example overfull lines.

The use of the ε-TEX extensions implies that this package works correctly only with modern engines and kernel formats. This is why the package starts with a statement where the kernel format is required to be quite recent.

# 7 Acknowledgements

I must deeply thank Enrico Gregorio for the revision of this package macros and for his wise suggestions about the correct programming style. If some glitch still remains in the programming style, that is just my fault.

# 8 Implementation

We identify the package and the necessary LATEX kernel

```
1 \ProvidesPackage{widetable}[2017/09/05 v.1.2 A simpler way to produce
2      tables of specified width]
```

The first thing to do is to globally define a certain number of TEX dimensions and counters; these dimension and counter registers are selected among the extra registers available with the ε-TEX extensions. In facts such extensions allow to use registers identified by numbers higher than 255, a limit of the good-old-TEX; Enrico Gregorio in 2009 suggested to avoid such kind of usage, stating that not all users upgrade so often their TEX system; OK, if they don't upgrade, they have

available the previous versions of this package; now I assume that after so many years from when the engines were extended with the $\varepsilon$-TeX facilities, I can't see why this package should resort to old approaches.

```
\dimendef\wt@Numer=2
\dimendef\wt@Denom=4
\countdef\wt@Num=2
\countdef\wt@Den=4
```

We require the `xparse` package in order to define the environment `widetable` with its extended commands.

```
\usepackage{xparse}
```

We define a local scaling macro `\WT@scale` to execute the scaling operation that is of interest to the `widetable` package. We provide some tests so as to set some values for very unusual situations. The four arguments are as such: `#1`: length to be scaled; `#2`: numerator of the scaling ratio; `#3`: denominator of the scaling ratio; `#4`: scaled length.

```
\newcommand\WT@scale[4]{\begingroup
\wt@Num #2\relax \wt@Den #3\relax
\ifnum#3=\z@
  \@tempdima\ifnum#2<0-\fi\maxdimen
\else
  \@tempdima\dimexpr#1*#2/#3\relax
\fi
\edef\x{\noexpand\endgroup\noexpand\setlength{#4}{\the\@tempdima}}%
\x}
```

At this point it will be the `widetable` environment responsibility to call `\WT@scale` with the proper arguments

Now we define the dimension register that is to contain the desired table width. We further define the start of the tabular typesetting that will be useful in a while. Actually the table preamble is being saved into a macro, so that when the ⟨*width*⟩ and the ⟨*column descriptors*⟩ are given to the opening environment statement, these saved quantities can be used again and again.

```
\newdimen\wt@width
```

A new boolean, `wt@scartare`, is defined; this boolean variable will be set true in order to detect if the table body is is not well formed, with `\begin` and `\end` statements that don't match, and the like; actually the `widetable` environment can contain other environment, even another `widetable` environment, but the external one should not be upset by the internal ones. In order to achieve this result, it is necessary that any embedded environment is hidden int a group delimited by a pair of matching braces.

```
\newif\ifwt@scartare\wt@scartarefalse

```

The environment opening as well as the environment closing are defined by means of low level commands. Due to the syntax of the opening command that requires two compulsory arguments, these are saved in the recently defined dimension register and to a macro respectively; another macro `\wt@getTable` is used to

get the body of the table; the `\end{widetable}` statement represents the ending delimiter of the table contents.

```
\DeclareDocumentCommand\widetable{m O{c} m}
{% OPENING WIDETABLE
  \def\@tempC{widetable}%
  \setlength{\wt@width}{#1}%
  \def\wt@preamble{#3}%
  \def\wt@starttabular{\tabular[#2]{#3}}%
  \wt@getTable}%
```

The closing statement will actually do the greatest part of the job. First of all if the above mentioned boolean variable is true, it skips everything and it does not set any table; but if the boolean variable is false, the table body is well formed and it can do the job as described in the previous sections. It first sets `\tabcolsep` to zero and sets the resulting table in box zero; the lower level `tabular` with the information saved into `\wt@startabular` and the body of the table contained into the token register zero.

Then it sets `\tabcolsep` to 1 cm (arbitrarily chosen) and typesets again the table into box two. The width of box zero is $l_0$ and that of box two is $l_1$; these are the lengths needed by the equation that evaluates the final typesetting inter-column spacing. The arbitrary constant of 1 cm is $t_1$, and the specified width $l$ is the dimension saved into `\wt@width`. The subtractions are operated directly on the dimension registers `\wt@width` (the numerator) and on the auxiliary register `\@tempdimenb`; the `\WT@scale` command is executed in order to get the scaling ratio and the final definitive value of `\tabcolsep` is eventually computed. The table is finally typeset without using boxes, while the contents of box zero and two are restored upon exiting the environment to any value they might have contained before entering `widetable`.

```
\def\endwidetable{% CLOSING WIDETABLE
  \ifwt@scartare
    \noindent\null
  \else
    \tabcolsep=\z@
    \setbox\z@=\hbox{\wt@starttabular\the\toks@\endtabular}%
    \tabcolsep=6pt\relax
    \setbox\tw@=\hbox{\wt@starttabular\the\toks@\endtabular}%
    \ifdim\wt@width<\wd\z@
       \@tempdimb=\dimexpr\wd\tw@-\wd\z@\relax
       \PackageWorning{widetable}{%
       The natural width ‘\the\wd\z@’ of the tabular
       material\MessageBreak
       is larger than the specified width
       ‘\the\wt@width’\MessageBreak\null\space \MessageBreak
       The table is typeset with the default column spacing}%
    \else
       \@tempdimb=\dimexpr\wt@width-\wd\z@\relax
    \fi
    \@tempdimc=\dimexpr\wd\tw@-\wd\z@\relax
```

```
      \WT@scale{\tabcolsep}{\@tempdimb}{\@tempdimc}{\tabcolsep}\relax
    \wt@starttabular\the\toks@\endtabular
  \fi
  \ignorespacesafterend
}

```

Of course other actions must be performed before executing the closing environment statement. We need a macro `wt@finetabella` that is equivalent to the ending environment statement.

```
\def\wt@finetabella{\end{widetable}}%

```

We finally can define the all important macro that gets the table body; it requires two delimited arguments: in `#1` the table body and, after the `\end` command, the closing environment name will be set in `#2`. The environment name is assigned to the macro `\@tempB`, which is checked against the correct name `widetable` saved in macro `\@tempC` by the opening command. If the names match, then the table body is assigned to the token register zero, to be used later on by the typesetting macros. But if the names don't match, then something went wrong and a package message is issued to explain what happened and how the program will manage the situation.

Specifically the names may not match if a cell contained another environment and its whole `\begin{...}...\end{...}` was not closed within a pair of matched braces. If an enclosed environment is hidden within a group, the delimited argument macro `\wt@getTable` will ignore such embedded environment, otherwise it will get a non matching name and messy things might happen. Besides warning about this fact, the body of the table, at least what has been read by the macro, will be discarded and substituted with a box containing a message; therefore a table will be typeset, but not the desired one. The remaining part of the body remains in the input stream and might cause, presumably, strange errors, such as `&` characters used outside a tabular or array environment. We must take care of this so that the typesetting procedure does not crash.

```
\def\wt@getTable#1\end#2{\def\@tempB{#2}%
  \ifx\@tempB\@tempC
    \toks@={#1}%
    \expandafter\wt@finetabella
  \else
    \PackageWarning{widetable}{%
      The table contains environment `\@tempB' %
      \MessageBreak
      not enclosed in braces. This is expressly forbidden!%
      \MessageBreak
      The table is not typeset  and is substituted%
      \MessageBreak
      with a framed box}%
      \advance\wt@width-2\fboxsep
    \noindent\fbox{\parbox{\wt@width}{The table was not typeset
     because it contains a visible \texttt{\char`\\end} in one
```

```
71      or more cells.}}\par
72     \expandafter\wt@finishTable
73   \fi
74 }
75
```

In order to avoid a complete mess, we have to iteratively gobble the rest of the input stream until a valid `\end{widetable}` is encountered; Actually the following macro will do a nice job in general, but it is not infallible if the input stream is really composed in a very bad way. In facts it calls itself again and again, always gobbling it arguments, until a valid terminating environment name matches the name `widetable`.

```
76 \def\wt@finishTable#1\end#2{%
77   \def\@tempB{#2}%
78   \ifx\@tempB\@tempC
79     \wt@scartaretrue\expandafter\wt@finetabella
80   \else
81     \expandafter\wt@finishTable
82   \fi
83 }
```

# 9 Conclusion

Tables should always have their natural width, but... The default value of`\tabcolsep` is fixed by the document class, it is not prescribed by a supreme law: therefore what does it mean "natural width". Probably the one determined by the class default value of `\tabcolsep` so all tables have the same general look.

Nevertheless sometimes a table is slightly wider than the current measure; why not shrink the table by shrinking `\tabcolsep` by the right amount in order to fit the measure? The result might be a table where only the intercolumn spaces are shrunk, not the whole table, fonts, drawings, and figures included, a result easily obtainable with a `\resizebox` command available through the graphicx.sty package. Nobody forbids to follow this technique, of course, but the `widetable` route might yield a better result.

The same is true when a natural width table is slightly shorter than the measure; enlarging it by retouching the `\tabcolsep` intercolumn space might be the right solution in order to avoid a multitude of slightly different indents or left margins.

This package might be useful also for copying some macros so as to avoid some programming in other packages; this use is certainly permitted by the LaTeX Project Public License, which sets the observance of very light obligations.

```
84
85 \endinput
```