Abstract

A new environment, \texttt{tabularx}, is defined, which takes the same arguments as \texttt{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 \texttt{X} in the preamble argument.

This package requires the \texttt{array} package.

1 Introduction

This package implements a version of the \texttt{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 \texttt{comp.text.tex}.

\begin{verbatim}
\begin{tabularx}{⟨width⟩}{[⟨pos⟩]{⟨preamble⟩}}
\end{tabularx}
\end{verbatim}

The arguments of \texttt{tabularx} are essentially the same as those of the standard \texttt{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 \texttt{tabularx} environment should be denoted with the letter \texttt{X} in the preamble argument. The \texttt{X} column specification will be converted to \texttt{p{⟨some value⟩}} once the correct column width has been calculated.

2 Examples

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

<table>
<thead>
<tr>
<th>Multicolumn entry!</th>
<th>THREE</th>
<th>FOUR</th>
</tr>
</thead>
<tbody>
<tr>
<td>one</td>
<td>The width of this column depends on the width of the table.\textsuperscript{1}</td>
<td>three</td>
</tr>
</tbody>
</table>

\textsuperscript{1}You can now use \texttt{\footnote} inside \texttt{tabularx}!
If we change the first line to \begin{tabularx}{300pt}{|c|X|c|X|} we get:

<table>
<thead>
<tr>
<th>Multicolumn entry!</th>
<th>THREE</th>
<th>FOUR</th>
</tr>
</thead>
<tbody>
<tr>
<td>one</td>
<td>The width of this column depends on the width of the table.</td>
<td>three</td>
</tr>
</tbody>
</table>

3 Differences between \texttt{tabularx} and \texttt{tabular*}

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

- \texttt{tabularx} modifies the widths of the columns, whereas \texttt{tabular*} modifies the widths of the inter-column spaces.

- \texttt{tabular} and \texttt{tabular*} environments may be nested with no restriction, however if one \texttt{tabularx} environment occurs inside another, then the inner one must be enclosed by \{ \}.

- The body of the \texttt{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.\footnote{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.}

- \texttt{tabular*} uses a primitive capability of \TeX{} to modify the inter column space of an alignment. \texttt{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 \texttt{tabularx}

4.1 Terminal output

\texttt{\tracingtabularx} If this declaration is made, say in the document preamble, then all following \texttt{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 \texttt{\tracingtabularx} declaration, either of the options \texttt{infoshow} or \texttt{debugshow} may be given, either in the \texttt{\usepackage} command that loads \texttt{tabularx}, or as a global option in the \texttt{\documentclass} command.

4.2 The environment used to typeset the \texttt{X} columns

By default the \texttt{X} specification is turned into \texttt{p\{some value\}}. Such narrow columns often require a special format, this may be achieved using the > syntax of \texttt{array.sty}. So for example you may give a specification of \texttt{\{small\}X}. Another format which is useful in narrow columns is ragged right, however \Euler\TeX{}'s \texttt{\raggedright} macro redefines \texttt{\} in a way which conflicts with its use in a tabular or array environments. For this reason this package introduces the command \texttt{\arraybackslash}, this may be used after a \texttt{\raggedright}, \texttt{\raggedleft}
or \centering declaration. Thus a tabularx preamble may specify
\{\raggedright\arraybackslash\}X.

\newcolumntype These preamble specifications may of course be saved using the command
\newcolumntype defined in array.sty. Thus we may say
\newcolumntype\{Y\}\{\small\raggedright\arraybackslash\}X\}
and then use Y in the tabularx preamble argument.

\tabularxcolumn The X columns are set using the p column which corresponds to \parbox[t].
You may want them set using, say, the m column, which corresponds to \parbox[c]. It is not possible to change the column type using the > syntax, so another system is provided. \tabularxcolumn should be defined to be a macro
with one argument, which expands to the tabular preamble specification that you
want to correspond to X. The argument will be replaced by the calculated width
of a column.

The default is \newcommand{\tabularxcolumn\[1\]}{p\{#1\}}. So we may
change this with a command such as:
\renewcommand{\tabularxcolumn\[1\]}{\{\small\}m\{#1\}}

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\hsize\linewidth=\hsize\}X
\{\hsize=1.5\hsize\linewidth=\hsize\}X
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.

5 Support for tagged PDF

With version 2.12a the package is made tagging aware, which means that it will
automatically produce tagged tables (necessary, for example, for accessibility) if
tagging is requested via DocumentMetadata.

More granular control, e.g., explicitly deciding which cells are header cells,
etc., is currently under development, but syntax for this will not appear in this
package. Instead it will become available across all tabular-generating packages
and then automatically apply here as well.

Enabling \LaTeX{} to automatically produce tagged PDF is a long-term project
and this is a tiny step in this puzzle. For more information on the project and
already available functionality, see https://latex-project.org/publications/

6 The Macros

1 \langle\texttt{\textbackslash package}\rangle

We only need two changes for tagging support, but they require that a recent
\LaTeX{} kernel is used (which should be no problem if \texttt{tools} is distributed in parallel
with the kernel.

2 \texttt{\langle\textasciitilde\textbackslash NeedsTeXFormat\{LaTeX2e\}\{2024/06/01\}\rangle}

3 \texttt{\textbackslash DeclareOption\{infoshow\}\{\textbackslash AtEndOfPackage\textbackslash tracingtabularx\}}

4 \texttt{\textbackslash DeclareOption\{debugshow\}\{\textbackslash AtEndOfPackage\textbackslash tracingtabularx\}}

5 \texttt{\textbackslash ProcessOptions}

This requires \texttt{array.sty}.

6 \texttt{\textbackslash RequirePackage\{array\}\{1994/02/03\}}

First some registers etc. that we need.

7 \texttt{\newdimen\TX@col@width}

8 \texttt{\newdimen\TX@old@table}

9 \texttt{\newdimen\TX@old@col}

10 \texttt{\newdimen\TX@target}

11 \texttt{\newdimen\TX@delta}

12 \texttt{\newcount\TX@cols}

13 \texttt{\newif\ifTX@}

Now a trick to get the body of an environment into a token register, without
doing any expansion. This does not do any real checking of nested environments,
so if you should need to nest one \texttt{tabularx} inside another, the inner one must be
surrounded by \{ \}. \texttt{\textbackslash tabularx}

Prior to v1.06, this macro took two arguments, which were saved in separate
registers before the table body was saved by \texttt{\textbackslash TX@get\textbackslash body}. Unfortunately this
disables the [t] optional argument. Now just save the width specification sepa-
ately, then clear the token register \texttt{\textbackslash tok\textbackslash s@}. Finally call \texttt{\textbackslash TX@get\textbackslash body}
to begin saving the body of the table. The \{\texttt{\textbackslash ifnum}\texttt{0='}1\texttt{\textbackslash fi}\} was added at v1.03, to allow
\texttt{tabularx} to appear inside a \texttt{\textbackslash halign}.\footnote{This adds an extra level of grouping, which is not really needed. Instead, I could use \texttt{here},
and \texttt{\textbackslash ifnum}\texttt{0='}1\texttt{\textbackslash fi} below, however the code here would then have to be moved after the first line,
because of the footnote to page 386 of the \LaTeX{}book.}

This mechanism of grabbing an environment body does have the disadvantage
(shared with the AMS alignment environments) that you can not make extension
environments by code such as

\begin{verbatim}
\newenvironment{foo}{\begin{tabularx}{XX}}{\end{tabularx}}
\end{verbatim}
as the code is looking for a literal string \end{tabularx} to stop scanning. Since version 2.02, one may avoid this problem by using \tabularx and \endtabularx directly in the definition:
\newenvironment{foo}{\begin{tabularx}{XX}}{\end{tabularx}}
The scanner now looks for the end of the current environment (foo in this example). There are some restrictions on this usage, the principal one being that \endtabularx must not be inside any { } pairs, so that the code before \endtabularx may be extracted and added to the table body (prior to version 2.09 \endtabularx had to be the first token of the ‘end code’ of the environment).

14 \def\tabularx#1{%
Allow \tabularx \endtabularx (but not \begin{tabularx} \end{tabularx}) to be used in \newenvironment definitions.
\edef\TX@{\@currenvir}%
\relax
\def\TX@target{#1}%
\setlength\TX@target{#1}%
\typeout{Target width: #1 = \the\TX@target}%
\toks@{}
\TX@get@body}
\endtabularx

This does not do very much...
\end{tabularx}
\TX@get@body Place all tokens as far as the first \end into a token register. Then call \TX@find@end to see if we are at \end{tabularx}.
\long\def\TX@get@body#1\end{tabularx}{%
\tx@get@body{\the\toks@#1}}
\TX@find@end If we are at \end{tabularx}, call \TX@get@body, otherwise add \end{...} to the register, and call \TX@get@body again.
\long\def\TX@find@end#1{%
\def\@tempa{#1}%
\ifx\@tempa\TX@
\TX@endtabularx
\else
\toks@{
\expandafter{\the\toks@#1}}
\expandafter\TX@get@body
\fi}
\TX@endtabularx Split up the end code, and extract the part that lives in the table body.
\long\def\TX@endtabularxa{%
#1\endtabularx\end{tabularx}\TX@endtabularxa{%
\expandafter{\the\toks@#1}}
\relax
\toks@{
\expandafter{\the\toks@#1}}
\fi}
\TX@endtabularxb Split up the end code, and extract the part that lives outside the table body.
\long\def\TX@endtabularxb{%
#1\endtabularx\end{tabularx}\TX@endtabularxb{%
\expandafter{\the\toks@#1}}
\fi}
Now that all the parts of the table specification are stored in registers, we can begin the work of setting the table.

The algorithm for finding the correct column widths is as follows. Firstly set the table with each $X$ column the width of the final table. Assuming that there is at least one $X$ column, this will produce a table that is too wide. Divide the excess width by the number of $X$ columns, and reduce the column width by this amount. Reset the table. If the table is not now the correct width, a \texttt{\textbackslash multicolumn} entry must be ‘hiding’ one of the $X$ columns, and so there is one less $X$ column affecting the width of the table. So we reduce by 1 the number of $X$ columns and repeat the process.

Although I have tried to make \texttt{\textbackslash tabularx} look like an environment, it is in fact a command, all the work is done by this macro.

Define the $X$ column, with an internal version of the \texttt{\textbackslash newcolumntype} command. The \texttt{\textbackslash expandafter} commands enable \texttt{\textbackslash NC\textbackslash newcol} to get the expansion of \texttt{\textbackslash tabularxcolumn\{\textbackslash TX@col@width\}} as its argument. This will be the definition of an $X$ column, as discussed in section 4.

Initialise the column width, and the number of $X$ columns. The number of $X$ columns is set to one, which means that the initial count will be one too high, but this value is decremented before it is used in the main loop.

Since v1.02, switch the definition of \texttt{\textbackslash verb}.

Since v1.05, save the values of all \LaTeX counters, the list \texttt{\textbackslash cl@ckpt} contains the names of all the \LaTeX counters that have been defined so far. We expand \texttt{\textbackslash setcounter} at this point, as it results in fewer tokens being stored in \texttt{\textbackslash TX@ckpt}, but the actual resetting of the counters occurs when \texttt{\textbackslash TX@ckpt} is expanded after each trial run. Actually since v1.07, use something equivalent to the expansion of the original definition of \texttt{\textbackslash setcounter}, so that \texttt{\textbackslash tabularx} works in conjunction with \texttt{calc.sty}.
Type out some headings (unless this is disabled).

While we do trial typesetting we suspend any tagging if that is active:

First attempt. Modify the X definition to count X columns.

Repeatedly decrease column width until table is the correct width, or stops shrinking, or the columns become too narrow. If there are no multicolumn entries, this will only take one attempt.

And here we restart it again:

One last time, with warnings back on (see appendix D) use `tabular*` to put it in a box of the right size, in case the algorithm failed to find the correct size.

Since v1.04, locally make `\footnotetext` save its argument in a token register. Since v1.06, `\toks@` contains the preamble specification, and possible optional argument, as well as the table body.

Now the alignment is finished, and the \ has restored the original meaning of `\footnotetext` expand the register `\footnotext` which will execute a series of \footnotetext commands. We need to be careful about clearing the register as we may be inside a nested tabularX.

Now finish off the `\end{tabularx}` here as the `\end{tabularx}` in the user's file is never expanded. Now use `\TX@` rather than `tabularx`.

We also need to finish off the group started by `{\ifnum0='}\fi` in the macro \tabularx.

\footnotetext
\end{tabularx}\TX@false\@tempdimb\maxdimen\divide\@tempdimb\TX@cols\ifdim\TX@col@width>\@tempdimb\TX@typeout@{Don’t exceed \maxdimen}\wd\@tempboxa\maxdimen\fi\ifdim\TX@old@table=\wd\@tempboxa

If we have reduced the column width, but the table width has not changed, we stop the loop, and output the table (which will cause an over-full alignment) with the previous value of \TX@col@width.

\TX@col@width\TX@old@col\TX@typeout@{Reached minimum width, backing up.}\else

Otherwise calculate the amount by which the current table is too wide.

\dimen@=\wd\@tempboxa\advance\dimen@ -\TX@target\ifdim\dimen@<\TX@delta\TX@typeout@{Reached target.}\else

Reduce the number of effective X columns by one. (Checking that we do not get 0, as this would produce an error later.) Then divide excess width by the number of effective columns, and calculate the new column width. Temporarily store this value (times $-1$) in \dimen@.

\ifnum\TX@cols>1\m@ne\advance\TX@cols\m@ne\divide\dimen@\TX@cols\advance\dimen@ -\TX@col@width\ifdim\dimen@<\z@\TX@typeout@{Reached target.}\else

If the new width would be too narrow, abort the loop. At the moment too narrow means less than 0 pt!

Prior to v2.03, if the loop was aborted here, the X columns were left with the width of the previous run, but this may make the table far too wide as initial guesses are always too big. Now force to \TX@error@width which defaults to be 1em. If you want to get the old behaviour stick \renewcommand{\TX@error@width{\TX@col@width} in a package file loaded after tabularx.

\PackageWarning{tabularx}{X Columns too narrow (table too wide)\MessageBreak}\TX@col@width\TX@error@width\relax\else

8
Otherwise save the old settings, and set the new column width. Set the flag to true so that the table will be set, and the loop will be executed again.

\begin{verbatim}
107 \TX@old@col\TX@col@width
108 \TX@old@table\wd\@tempboxa
109 \TX@col@width-\dimen0
110 \TX@true
111 \fi
112 \fi
113 \fi}
\end{verbatim}

\TX@error@width If the calculated width is negative, use this instead.

114 \def\TX@error@width{1em}

\TX@delta Accept a table that is within \hfuzz of the correct width.

115 \TX@delta\hfuzz

Initialise the \textit{X} column. The definition can be empty here, as it is set for each \texttt{tabularx} environment.

116 \newcolumntype{X}{}

\texttt{tabularxcolumn} The default definition of \texttt{X} is \texttt{p\{#1\}}.

117 \def\tabularxcolumn#1{p\{#1\}}

\TX@newcol A little macro just used to cut down the number of \texttt{expandafter} commands needed.

118 \def\TX@newcol{\newcol@{X}\[0\]}

\TX@trial Make a test run.

119 \def\TX@trial#1{%
120 \setbox\@tempboxa\hbox{%

Any extra commands. This is used on the first run to count the number of \texttt{X} columns.

121 #1\relax
122 \let\@footnotetext\TX@trial@ftn
123 \let\TX@vwarn\@empty

Do not nest \texttt{tabularx} environments during trial runs. This would waste time, and the global setting of \texttt{TX@cols} would break the algorithm.

124 \expandafter\let\expandafter\tabularx\csname tabular*\endcsname
125 \expandafter\let\expandafter\endtabularx\csname endtabular*\endcsname

Added at v1.05: disable \texttt{writes} during a trial run. This trick is from the \TeXbook.\footnote{Actually the \TeXbook trick does not work correctly, so changed for v2.05.}

126 \def\write{\begingroup
127 \def\let{\afterassignment\endgroup\toks0}%
128 \afterassignment\let\count0}%
\end{verbatim}
Turn off warnings (see appendix D). Also prevent them being turned back on by setting the parameter names to be registers.

Make the table, and finish the hbox. Since v1.06, \toks@ contains the preamble specification, and possible optional argument, as well as the table body.

Since v1.05 reset all \TeX counters, by executing \TXckpt.

Print some statistics. Added \TXalign in v1.05, to line up the columns.

Macro added at v1.05, to improve the printing of the tracing info.

\arraybackslash hack.
\tracingtabular Print statistics on column and table widths.
\TXtypeout The default is to be quiet.
\TXftn A token register for saving footnote texts.
\TXftntext Inside the alignment just save up the footnote text in a token register.
On trial runs, gobble footnote texts.

This last section was added at Version 1.02. Previous versions documented the fact that \verb did not work inside \texttt{tabularx}, but that did not stop people using it! This usually put \TeX into an irrecoverable error position, with error messages that did not mention the cause of the error. The ‘poor man’s \verb’ (and \verb*) defined here is based on page 382 of the \TeXbook. As explained there, doing verbatim this way means that spaces are not treated correctly, and so \verb* may well be useless, however I consider this section of code to be error-recovery, rather than a real implementation of verbatim.

The mechanism is quite general, and any macro which wants to allow a form of \verb to be used within its argument may \let\verb=\TXverb. (Making sure to restore the real definition later!)

\verb and \verb* are subject to the following restrictions:

1. Spaces in the argument are not read verbatim, but may be skipped according to \TeX’s usual rules.
2. Spaces will be added to the output after control words, even if they were not present in the input.
3. Unless the argument is a single space, any trailing space, whether in the original argument, or added as in (2), will be omitted.
4. The argument must not end with \`, so \verb\` is not allowed, however, because of (3), \verb\` produces \.
5. The argument must be balanced with respect to \{ and \}. So \verb\{ is not allowed.
6. A comment character like \% will not appear verbatim. It will act as usual, commenting out the rest of the input line!
7. The combinations ?' and !' will appear as ¿ and ¡ if the \cmtt font is being used.

\TXverb The internal definition of \verb. Spaces will be replaced by \textasciitilde, so for the star-form, \verb\` is \textasciitilde, which we obtain as \texttt{\textasciitilde}. Use {\ifnum0='}\fi rather than {\hgroup} to allow & to appear in the argument.

\TXverb Get the ‘almost verbatim’ text using \meaning. The ‘!’ is added to the front of the user supplied text, to ensure that the whole argument does not consist of a single \{ group. \TeX would strip the outer braces from such a group. The ‘!’ will be removed later.

Originally I followed Knuth, and had \def\@tempa{##1}, however, this did not allow # to appear in the argument. So in v1.04, I changed this to use a token
register, and \edef. This allows \# to appear but makes each one appear twice, so later we loop through, replacing \## by \#.

\def\TX@vb#1{\def\@tempa##1#1{\toks@{##1}\edef\@tempa{\the\toks@}}\expandafter\TX@v\meaning\@tempa\ \ \ifnum0='\fi}}\@tempa!}

\TX@v Strip the initial segment of the \meaning, including the ‘!’ added earlier.

\def\TX@v#1!{\afterassignment\TX@vfirst\let\@tempa= }

As explained above we are going to replace \## pairs by \#. To do this we need non-special \# tokens. Make * into a parameter token so that we can define macros with arguments. The normal meanings will be restored by the \endgroup later.

\begingroup
\catcode’*\catcode’\#=12
\TX@vfirst As a special case, prevent the first character from being dropped. This makes \verb*\ | produce \textbackslash. Then call \TX@v@. This is slightly tricky since v1.04, as I have to ensure that an actual \# rather than a command \let to \# is passed on if the first character is \#.

\gdef\TX@vfirst{% 
  \if\@tempa#% 
    \def\@tempb{\TX@v@#} 
  \else 
    \let\@tempb\TX@v@
    \if\@tempa\space~\else\@tempa\fi
  \fi 
  \@tempb}

\TX@v@ Loop through the \meaning, replacing all spaces by \textasciitilde. If the last character is a space it is dropped, so that \verb*\LaTeX\ produces \LaTeX\ not \LaTeX\. The rewritten tokens are then further processed to replace \## pairs.

\gdef\TX@v@*1 *2{\TX@v@hash*1##\relax\if*2\\else~\expandafter\TX@v@fi*2}

\TX@v@hash The inner loop, replacing \## by \#.

\gdef\TX@v@hash*1##*2{*1\ifx*2\relax\fi*2\\else~\expandafter\TX@v@fi*2}

As promised, we now restore the normal meanings of \# and *.

\endgroup

\TX@vwarn Warn the user the first time this \verb is used.

\gdef\TX@vwarn{% \@warning{\noexpand\verb may be unreliable inside tabularx}% \global\let\TX@vwarn@empty}

\TX@vwarn ⟨/package⟩