The \texttt{eqparbox} package\footnote{This document corresponds to \texttt{eqparbox} v4.1, dated 2017/09/03.}

Scott Pakin

\texttt{scott+eap@pakin.org}

September 3, 2017

Abstract

The \texttt{eqparbox} package makes it easy to define a group of boxes (such as those produced by \texttt{parbox} or \texttt{makebox}) whose members all have the same width, the natural width of the widest member. A document can contain any number of groups, and each group can contain any number of members. This simple, equal-width mechanism can be used for a variety of alignment purposes, as is evidenced by the examples in this document.

1 Motivation

Let’s start with a little test. How would you typeset Figure 1 in which the names of the quotations’ authors are left-justified relative to each other but as a group abut the right margin? And second, how would you typeset the résumé excerpt shown in Figure 2 while meeting the following requirements:

1. The header columns must be left-justified relative to each other.
2. The header columns should be evenly spaced across the page.
3. Page breaks should be allowed within the résumé.

The two questions can be answered the same way: by putting various blocks of text into equal-widthed boxes. If the author names in Figure 1 are placed within a \texttt{flushright} environment and in \texttt{parbox}es as wide as the widest text (“Rosen-crantz & Guildenstern Are Dead”), they will appear as desired. Similarly, if the company names in Figure 2 are both put in a \texttt{parbox} as wide as “Thingamabobs, Ltd.,” the job titles in a \texttt{parbox} as wide as “Senior Widget Designer,” and the dates in a \texttt{parbox} as wide as “1/95–present,” then they can be spaced evenly by separating them with \texttt{\hfill}s.

The problem is in choosing the width for each set of \texttt{parbox}es. Considering for now just Figure 2 the user must typeset the résumé once to see which entry in each column is the widest and then assign lengths appropriately:
The only medicine for suffering, crime, and all other woes of mankind, is wisdom. Teach a man to read and write, and you have put into his hands the great keys of the wisdom box. But it is quite another thing to open the box.

— Thomas Huxley

I would like a simple life
yet all night I am laying
poems away in a long box.

It is my immortality box,
my lay-away plan,
my coffin.

— Anne Sexton
The Ambition Bird

We have four boxes with which to defend our freedom: the soap box, the ballot box, the jury box, and the cartridge box.

— Larry McDonald

I saw the Count lying within the box upon the earth, some of which the rude falling from the cart had scattered over him. He was deathly pale, just like a waxen image, and the red eyes glared with the horrible vindictive look which I knew so well.

— Bram Stoker
Dracula

Life in a box is better than no life at all, I expect. You’d have a chance, at least. You could lie there thinking, “Well, at least I’m not dead.”

— Tom Stoppard
Rosencrantz & Guildenstern Are Dead

Alla fin del gioco tanto va nel sacco il re quanto la pedina.
(After the game, the king and pawn go into the same box.)

— Italian proverb

Figure 1: Quotations with left-aligned attributions
Widgets, Inc.  Senior Widget Designer  1/95–present

- Supervised the development of the new orange and blue widget lines.
- Improved the design of various widgets, making them less sticky and far less likely to explode.
- Made widget management ten times more cost-effective.

Thingamabobs, Ltd.  Lead Engineer  9/92–12/94

- Found a way to make thingamabobs run on solar power.
- Drafted a blueprint for a new doohickey-compatibility module for all cool-mint thingamabobs.
- Upgraded superthingamabob specification document from Microsoft Word to \LaTeX{}.

Figure 2: Excerpt from a sample résumé

\newlength{\placewidth}
\settowidth{\placewidth}{Thingamabobs, Ltd.} % Employment 2
\newlength{\jobtitlewidth}
\settowidth{\jobtitlewidth}{Senior Widget Designer} % Employment 1
\newlength{\dateswidth}
\settowidth{\dateswidth}{1/95--present} % Employment 1

Every time a piece of information changes, it must be changed in two places: in the résumé itself and in the \settowidth command. When employment information is added or deleted, the \settowidth commands must be modified to reflect the new maximum-widthed entry in each column. If only there were a simpler way to keep a set of \parbox{\textwidth}s as wide as the widest entry in the set ...

That simpler way is the eqparbox package. eqparbox exports an \eqparbox macro that works just like \parbox, except that instead of specifying the width of the box, one specifies the group that the box belongs to. All boxes in the same group will be typeset as wide as the widest member of the group. In that sense, an \eqparbox behaves like a cell in an 1, c, or r column in a tabular; \eqparboxes in the same group are analogous to cells in the same column. Unlike the cells in a tabular column, however, a group of \eqparboxes can be spread throughout the document.
2 Usage

These are almost identical to, respectively, the \parbox, makebox, framebox, and savebox macros and the minipage environment. The key difference is that the \texttt{\langle width\rangle} argument is replaced by a \texttt{\langle tag\rangle} argument. (For a description of the remaining arguments, look up \parbox, makebox, framebox, savebox, and minipage in any \LaTeX\ 2ε book or in the usrguide.pdf file that comes with all \TeX\ distributions.) \texttt{\langle tag\rangle} can be any valid identifier. All boxes produced using the same tag are typeset in a box wide enough to hold the widest of them. Discounting \TeX's limitations, any number of tags can be used in the same document, and any number of boxes can share a tag. The only catch is that \texttt{latex} will need to be run a second time for the various box widths to stabilize.

\texttt{\eqboxwidth \{\langle tag\rangle\}}

It is sometimes useful to take the width of a box produced by one of the preceding commands. While the width can be determined by creating an \texttt{\parbox} and using \texttt{\settowidth} to measure it, the \texttt{eqparbox} package defines a convenience routine called \texttt{\eqboxwidth} that achieves the same result.

\texttt{\eqboxwidth} makes it easy to typeset something like Table 1. Table 1's only column expands to fit the widest cell in the column, excluding the final cell. The final cell's text word-wraps within whatever space is allocated to it. In a sense, the first four cells behave as if they were typeset in an \texttt{l} column, while the final cell behaves as if it were typeset in a \texttt{p} column. In actuality, the column is an \texttt{l} column; an \texttt{eqparbox} for the first four cells ensures the column stretches appropriately while a \texttt{parbox} of width \texttt{\eqboxwidth\{\langle tag\rangle\}} in the final cell ensures that the final cell word-wraps.

Section 3.5 presents a more general version of this approach that doesn’t require cells to be divided explicitly into \texttt{eqparbox} cells and \texttt{parbox} cells.

\texttt{\eqsetminwidth \{\langle tag\rangle\} \{\langle width\rangle\}}

\texttt{\eqsetmaxwidth \{\langle tag\rangle\} \{\langle width\rangle\}}

These macros override the width calculation for boxes associated with tag \texttt{\langle tag\rangle}, ensuring that they are no narrower than a given minimum (\texttt{\eqsetminwidth}) and no wider than a given maximum (\texttt{\eqsetmaxwidth}).
Table 1: A tabular that stretches to fit some cells while forcing others to wrap

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Wide</td>
<td></td>
</tr>
<tr>
<td>Wider</td>
<td></td>
</tr>
<tr>
<td>Wider than that</td>
<td></td>
</tr>
<tr>
<td>This is a fairly wide cell</td>
<td></td>
</tr>
<tr>
<td>While this cell’s text wraps, the previous cells (whose text doesn’t wrap) determine the width of the column.</td>
<td></td>
</tr>
</tbody>
</table>

These macros are analogous to \texttt{\textbackslash eqsetminwidth} and \texttt{\textbackslash eqsetmaxwidth} but automatically compute the natural width of the given text and use that as the minimum (\texttt{\textbackslash eqsetminwidthto}) or maximum (\texttt{\textbackslash eqsetmaxwidthto}) width for boxes using tag \texttt{\textbackslash tag}.

3 Examples

This section presents some sample uses of the macros described in Section 2.

3.1 Figures and tables from previous sections

Figure 1 was typeset using a \texttt{\textbackslash eqparbox}-based helper macro, \texttt{\textbackslash showquote}:

\begin{verbatim}
\usepackage{ifmtarg}
\makeatletter
\newcommand{\showquote}[2]{%
  \begin{flushright}
  ---~\texttt{\textbackslash eqparbox}{quotebox}{\sffamily#1}\
  \@ifnotmtarg{#2}{\mbox{}\phantom{---}~\texttt{\textbackslash eqparbox}{quotebox}{\sffamily\itshape#2}\
  }%
  \end{flushright}%
  \par%
}%
\makeatother

\textit{(After the game, the king and pawn go into the same box.)}
\end{verbatim}
Figure 2’s headings were typeset with the following code:

\noindent \eqparbox{place}{\textbf{Widgets, Inc.}} \hfill \eqparbox{title}{\textbf{Senior Widget Designer}} \hfill \eqparbox{dates}{\textbf{1/95--present}}

\noindent \eqparbox{place}{\textbf{Thingamabobs, Ltd.}} \hfill \eqparbox{title}{\textbf{Lead Engineer}} \hfill \eqparbox{dates}{\textbf{9/92--12/94}}

Finally, Table 1 was typeset using the following code:

\begin{tabular}{|@{}l@{}}
\hline
\eqparbox[b]{wtab}{Wide} \hline
\eqparbox[b]{wtab}{Wider} \hline
\eqparbox[b]{wtab}{Wider than that} \hline
\eqparbox[b]{wtab}{This is a fairly wide cell} \hline
\parbox[b]{\eqboxwidth{wtab}}{\strut While this cell’s text wraps, the previous cells (whose text doesn’t wrap) determine the width of the column.} \hline
\end{tabular}

3.2 Lists within tabulars

List environments (itemize, enumerate, etc.) cannot appear directly within a tabular cell. Instead, they must be wrapped within a `parbox`. The problem is that the `parbox` width must be specified; it can’t be determined automatically. Fortunately, as of version 4.0 of `eqparbox`, the `\eqparbox` macro can contain list environments, and these are automatically sized to their widest item, just like any other `\eqparbox` contents. Table 2 presents an example of `enumerate` lists appearing within `tabular` cells. The code for this is straightforward, thanks to `eqparbox`:

\begin{enumerate}
\item Hardware upgrades
\end{enumerate}
Table 2: Lists within a \textit{tabular}

<table>
<thead>
<tr>
<th>Meeting date</th>
<th>Topics discussed</th>
</tr>
</thead>
</table>
| 2017-02-22   | 1. Hardware upgrades  
               2. Barbara’s retirement  
               3. Revised 27B/6 paperwork |
| 2017-03-01   | 1. Printer low on toner  
               2. Message from the V.P. |
| 2017-03-08   | 1. Product to ship next week  
               2. Floors to be recarpeted  
               3. Too many meetings |

\item Barbara’s retirement
\item Revised 27B/6 paperwork
\end{enumerate}
\hline

\subsection{Hanging indentation}

Consider the paragraphs depicted in Figure\textsuperscript{3}. We’d like the paragraph labels set on the left, as shown, but we’d also like to allow both intra- and inter-paragraph page breaks. Of course, if the labels are made wider or narrower, we’d like the paragraph widths to adjust automatically. By using a custom \texttt{list} environment that typesets its labels with \texttt{eqparbox} this is fairly straightforward:

\begin{verbatim}
\begin{list}{}{% 
\renewcommand{\makelabel}[1]{\eqparbox{b}{listlab}{#1}}% \setlength{\labelwidth}{\eqboxwidth{listlab}}% \setlength{\labelsep}{2em}% \setlength{\parsep}{2ex plus 2pt minus 1pt}% \setlength{\itemsep}{0pt}% \setlength{\leftmargin}{\labelwidth+\labelsep}% \setlength{\rightmargin}{0pt}% 
\item[Stuff about me] I am great. Blah, blah, blah, ...
\item[More stuff] I am wonderful. Blah, blah, blah, ...
\end{list}
\end{verbatim}

7
Item[The final exciting thing] I am fantastic. Blah, blah, blah, ...
end{list}


Figure 3: Paragraphs with hanging indentation

3.4 Justified, parallel text

Consider line-by-line transcription of a piece of text as illustrated by the mockup in Figure 3. The idea is to juxtapose a scanned piece of handwritten text with its typeset version (or, similarly, to typeset a piece of text in one language alongside a line-by-line translation into another language). The challenge is in ensuring that (1) the same words appear on corresponding lines of text and that (2) the typeset text is fully justified. While the parallel package can typeset fully justified paragraphs aligned in parallel columns, it does not support the alignment of individual lines. tabular and minipage environments provide control of line breaks but do not support full justification of the text when explicit line breaks are used.

One solution is to use eqparbox’s \eqmakebox macro. Like \makebox, \eqmakebox supports the “s” (stretch) value for the \langle pos \rangle argument, which causes the \langle text \rangle argument to stretch to the width of the box. However, while \makebox requires the width to be specified explicitly, \eqmakebox automatically sizes all boxes that use the same tag (in this case, each line of the input paragraph) to the widest text’s natural width. Here’s how to use the array package’s \newcolumntype
macro to define a new \texttt{tabular} column type, “S”, that stretches whitespace as needed to fit the widest line in the column:

\begin{verbatim}
\newsavebox{\tstretchbox}
\newcolumntype{S}[@1]{%$
\begin{lrbox}{\tstretchbox}$l\%
<s{\end{lrbox}$\eqmakebox[#1][s]{\unhcopy\tstretchbox}}
\end{verbatim}

That code works by storing the current cell’s contents within a box called \texttt{\tstretchbox} then passing \texttt{\tstretchbox}'s contents to \texttt{\eqmakebox}. (The \texttt{tabular} environment does not enable a cell’s contents to be passed directly to a macro, hence the \texttt{lrbox} trickery.) Note that the “S” column type takes an argument, which is the tag to pass to \texttt{\eqmakebox}. Using the preceding definition we can typeset Figure 4 as follows. To simulate scanned handwriting in the left column we use the Calligra handwriting font provided by the \texttt{calligra} package.

\begin{verbatim}
\begin{tabular}{|l|l|}
\hline
\texttt{\calligra\begin{tabular}{S{written}}}
Lorem ipsum dolor sit amet, \
consectetur adipiscing elit. \
Phasellus volutpat, nibh sit \
amet mattis convallis, metus \
libero rhoncus justo, sed auctor \erat mauris sit amet tellus. \\
\end{tabular}
\texttt{\begin{tabular}{S{written}}}
\begin{tabular}{S{typeset}}
Lorem ipsum dolor sit amet, \
consectetur adipiscing elit. \
Phasellus volutpat, nibh sit \
amet mattis convallis, metus \
libero rhoncus justo, sed auctor \erat mauris sit amet tellus. \\
\end{tabular}
\end{verbatim}

Figure 4: Line-by-line transcription of text with full justification
3.5 Combining $l$ and $p$ column properties in a tabular

In a tabular environment, $l$ columns, which automatically fit the column to its contents, are good for short pieces of text. Long pieces of text are best set within a $p$ column, which wraps text within a specified width. But which column type should you use to typeset text whose width is unknown (e.g., if the text is generated programmatically)? With the help of eqparbox’s \texttt{eqsetmaxwidth} macro (and the array package’s \texttt{newcolumntype} macro), it is possible to define a column type that behaves like $l$ for short pieces of text and like $p$ for long pieces of text:

\begin{verbatim}
\newcolumntype{M}{[1]{%  
  >\begin{lrbox}{\csname#1box\endcsname}%  
  \begin{lrbox}{\csname#1box\endcsname}%  
  \eqparbox[1]{#1}{\unhcopy\csname#1box\endcsname\strut}%
\end{lrbox}%
\end{tabular}
\end{verbatim}

This can then be used as follows to produce the output shown in Figure 5(a):

\begin{verbatim}
\begin{tabular}{|M{maybebig}|l|}
\hline
  Very short & Good \\
  A little bit longer & Okay \\
\end{tabular}
\end{verbatim}

Because the text in the first column is narrower than half the line width, the column behaves like an $l$ column. Now observe what happens if we add a long piece of text to the column:

\begin{verbatim}
\begin{tabular}{|M{maybebig}|l|}
\hline
  Very short & Good \\
  A little bit longer & Okay \\
  Almost certainly excessively long, even given the point we’re trying to make about box widths & Bad \\
\end{tabular}
\end{verbatim}

As Figure 5(b) shows, the first column now behaves like a $p$ column, specifically $p{0.5\textwidth}$. 

\end{tabular} \\hline\end{tabular}
3.6 Centering a column of right-justified data

The data in each of the Sales columns in Table 3 are centered relative to their column header. However, they are also right-justified relative to each other. To achieve this effect we simply need to put the data in each column in a right-justified box using \texttt{\textbackslash eqmakebox[\langle tag\rangle][r]{\langle text\rangle}} and center that:

\begin{tabular}{@{}lccc@{}}
\hline
\multicolumn{1}{c}{Product} & October & November & December \\
hline
Widgets & 55.2 & 89.2 & 57.9 \\
Dooohickeys & 65.0 & N/A & 9.3 \\
Thingamabobs & 10.4 & 8.0 & 109.7 \\
\hline
\end{tabular}

Table 3: Sample sales data
4 Limitations

Unfortunately, \texttt{eqparbox}'s macros have a number of limitations not exhibited by the corresponding \LaTeXe\ commands. First, \texttt{eqparbox}'s macros internally typeset the given text within a \texttt{tabular} environment—specifically, using "@{}l@{}" as the template—in order to determine the text’s natural width. Consequently, commands not valid within such a \texttt{tabular} (e.g., \texttt{verbatim} environments) are also not valid within the \texttt{⟨text⟩} argument of an \texttt{eqparbox} macro. As a corollary, the macros defined by the \texttt{eqparbox} package can appear only where a \texttt{tabular} is also acceptable.

A second limitation is that \texttt{eqparbox}'s macros typeset their \texttt{⟨text⟩} argument \textit{twice}: once within a \texttt{tabular} to determine the natural width and again within a box wide enough to hold all text associated with tag \texttt{⟨tag⟩}. This approach may cause unexpected results if \texttt{⟨text⟩} is non-idempotent (i.e., has side effects). For example, if \texttt{⟨text⟩} increments a counter, the counter will be incremented twice per invocation of \texttt{\eqparbox}.

5 Implementation

The one-sentence summary of the implementation is, “As \texttt{eqparbox} goes along, it keeps track of the maximum width of each box type, and when it’s finished, it writes those widths to the .\texttt{aux} file for use on subsequent runs.” If you’re satisfied with that summary, then read no further. Otherwise, get ready to tackle the following annotated code listing.

5.1 Preliminaries

\begin{verbatim}
\eqp@tempdima \newlength{\eqp@tempdima}
\eqp@tempdimb \newlength{\eqp@tempdimb}
\eqp@taglist \def{\eqp@taglist{}}
\ifeqp@must@rerun \eqp@must@reruntrue \eqp@must@rerunfalse
\if\eqp@must@rerun\else\eqp@must@rerunfalse\fi
\eqp@tabular@box \eqp@list@box\end{verbatim}

Define a couple temporary \texttt{⟨dimen⟩}s for use in a variety of locations.
1 \newlength{\eqp@tempdima}
2 \newlength{\eqp@tempdimb}
3 \def{\eqp@taglist{}}
4 \if\eqp@must@rerun\else\eqp@must@rerunfalse\fi

Define a list of all of the tags we encountered in the author’s document.

If an \texttt{eqparbox} is wider than the maximum-width \texttt{eqparbox} with the same tag, we need to store the new maximum width and request that the user re-run \texttt{latex}. We use \texttt{\if\eqp@must@rerun\else\eqp@must@rerunfalse\fi} to assist with this.

The \texttt{\eqp@settowidth} macro requires the \texttt{array} package’s ability to inject code into every cell.

\begin{verbatim}
\eqp@tablular@box \eqp@list@box
\end{verbatim}

The \texttt{\eqp@settowidth} macro requires a box, \texttt{\eqp@tablular@box}, in which to store the entire input text. \texttt{\eqp@list@box} also requires a box, \texttt{\eqp@list@box}, in which to store nested \texttt{list} environments.
The `\eqp@list@indent` macro stores the accumulated list indentation in `\eqp@list@indent`.

The `\eqminipage` environment requires the `environ` package’s `\Collect@Body`, which passes the body of an environment to a macro as a single argument.

5.2 Width calculation

To find the natural width of a piece of text, we put it in a table and take the width of that. The problem is that font changes are not preserved across line breaks (table cells). We therefore define an `\eqp@storefont` macro which itself defines an `\eqp@restorefont` macro that restores the current font and font size to its current state.

```
\newcommand*{\eqp@storefont}{%
  \xdef{\eqp@restorefont}{%}
  \noexpand{\usefont{\f@encoding}{\f@family}{\f@series}{\f@shape}}%
  \noexpand{\fontsize{\f@size}{\f@baselineskip}}%
  \noexpand{\selectfont}
}%
```

This macro is just like `\settowidth`, but it puts its argument in a `tabular`, which means that it can contain `\\`. We use the `array` package’s “>” and “<” template parameters to inject an `\eqp@restorefont` at the start of every cell and an `\eqp@storefont` at the end of every cell. Doing so preserves fonts and font sizes across `\\` boundaries, just like `\parbox`.

One catch is that lists cannot be included directly within a `tabular`. True, they can be placed within a `\parbox` that itself is within a `tabular` cell, but the whole point is that we’re trying to calculate how wide that `\parbox` should be. The trick we use here, therefore, is to redefine the `list` environment as a single-column `tabular` plus space for `\labelwidth` and `\labelsep`—we ignore all other list-formatting parameters—and `\item` as `\\`. There will be an extra row at the beginning, but all we care about here is computing a width, not a height, so that’s acceptable.

```
\newcommand{\eqp@settowidth}[2]{% 
  \begingroup
  \global\setbox\eqp@tabular@box=\hbox{\eqp@endings
  \eqp@storefont \newsavebox{\eqp@list@box}

\eqp@list@indent
```

Unfortunately, we can’t simply redefine the `list` environment, which underlies `itemize`, `enumerate`, and `description` lists, because their definitions in the standard classes do not include a proper `\begin{list}...\end{list}`. Instead, those parent environments call `\list` directly and `\let\end{itemize,enumerate,`
Our workaround is to reissue those \let bindings after redefining \list and \endlist ourselves.

```
\def\eqp@endings{}%
\ifx\enditemize\endlist
  \g@addto@macro\eqp@endings{\let\enditemize=\endlist}\
\fi
\ifx\endenumerate\endlist
  \g@addto@macro\eqp@endings{\let\endenumerate=\endlist}\
\fi
\ifx\enddescription\endlist
  \g@addto@macro\eqp@endings{\let\enddescription=\endlist}\
\fi
```

As described above, we locally redefine the \list environment as a single-column \tabular and the \item macro as \. We begin by copying a block of code from `ltlists.dtx` that sets the default formatting parameters for a list of the current depth. This is important because \trivlist environments (e.g., \center and \flushleft) reset some of the parameters, which would otherwise screw up our width calculation.

```
\renewenvironment{list}{}[2]{% 
  \ifnum \@listdepth >5\relax 
    \@toodeep 
  \else 
    \global\advance\@listdepth\@ne 
  \fi 
  \rightmargin\z@ 
  \listparindent\z@ 
  \itemindent\z@ 
  \csname @list\romannumeral\the\@listdepth\endcsname ##2\relax
\item
```

We locally redefine \item to start a new row of the \tabular, then flush any nested lists from the previous \item at the current nesting level, and finally adjust the current indentation based on the item’s label.

```
\renewcommand*{\item}{}[1][]{% 
  \mbox{}\ 
  \box\eqp@list@box\mbox{} \ 
  \box\@tempboxa{\makelabel{####1}}% 
  \ifdim\wd\@tempboxa\labelwidth
    \advance\eqp@list@indent by -\labelwidth
    \advance\eqp@list@indent by \wd\@tempboxa
  \fi 
  \hspace*{\eqp@list@indent}%
}\item
```

To measure the width of a \list we introduce a single-column \tabular that includes \eqp@list@indent’s worth of padding (= \leftmargin + \rightmargin + \itemindent) to mimic the width of the original \list environment.

```
\hspace*{-\eqp@list@indent}"
```
Finally, we place the given text—list or not—within a \texttt{tabular} so the preceding \texttt{\settowidth} can measure its width. Because the text may contain paragraph breaks we redefine \texttt{\par} as \texttt{\\} to turn them into line breaks and restore \texttt{\par}'s original definition when the \texttt{tabular} ends.

\begin{verbatim}
  \global\let\eqp@par=\par
  \eqp@storefont
  \begin{tabular}{@{}>{\eqp@restorefont}l<{\eqp@storefont}@{}}
    \global\@setpar{\}\% \\
    \box\eqp@list@box
  \end{tabular}
  \global\@restorepar
\end{verbatim}

Now that we've constructed a \texttt{tabular} with lines of the input text as cells we can use \LaTeX{}'s \texttt{\settowidth} macro to take its width.

\begin{verbatim}
\settowidth{#1}{\box\eqp@tabular@box}
\end{verbatim}

\texttt{\eqp@compute@width} The following function does all the real work for the \texttt{eqparbox} package. It takes two parameters—\texttt{\langle\text{tag}\rangle} and \texttt{\langle\text{text}\rangle}—and ensures that all boxes with the same tag will be as wide as the widest box with that tag. It ends by passing \texttt{\langle\text{tag}\rangle} and \texttt{\langle\text{text}\rangle} to the \texttt{\eqp@produce@box} command, which was defined by the calling macro to produce a box using one of the existing \LaTeX{} commands.

To keep track of box widths, \texttt{\eqp@compute@width} makes use of two global variables for each tag: \texttt{\eqp@this\langle\text{tag}\rangle} and \texttt{\eqp@next\langle\text{tag}\rangle}. \texttt{\eqp@this\langle\text{tag}\rangle} is the maximum width ever seen for tag \texttt{\langle\text{tag}\rangle}, including in previous \texttt{latex} runs. \texttt{\eqp@next\langle\text{tag}\rangle} works the same way but is always initialized to 0.0pt. It represents the maximum width to assume in \texttt{\textit{subsequent latex}} runs. It is needed to detect whether the widest text with tag \texttt{\langle\text{tag}\rangle} has been removed/shrunk. At the end of a run, \texttt{eqparbox} prepares the next run (via the .aux file) to initialize \texttt{\eqp@this\langle\text{tag}\rangle} to the final value of \texttt{\eqp@next\langle\text{tag}\rangle}.

We first clamp the box width, currently in \eqp@tempdimb, to the range [\eqp@minwd⟨tag⟩, \eqp@maxwd⟨tag⟩]. As these bounds are not necessarily defined we first have to check for their existence.

\@ifundefined{eqp@minwd#1}{}{% 
\ifdim\eqp@tempdimb<\csname eqp@minwd#1\endcsname 
\eqp@tempdimb=\csname eqp@minwd#1\endcsname 
\fi 
}\@ifundefined{eqp@maxwd#1}{}{% 
\ifdim\eqp@tempdimb>\csname eqp@maxwd#1\endcsname 
\eqp@tempdimb=\csname eqp@maxwd#1\endcsname 
\fi 
}\expandafter\ifx\csname eqp@this#1\endcsname\relax 
If we get here, then we’ve never encountered tag ⟨tag⟩, even in a previous latex run. We request that the user re-run latex. This is not always necessary (e.g., when all uses of the \eapparbox with tag ⟨tag⟩ are left-justified), but it’s better to be safe than sorry.

\global\eqp@must@reruntrue 
\expandafter\gdef\csname eqp@seen#1\endcsname{}% 
\@cons\eqp@taglist{{#1}}% 
\else 
If we get here, then we have previously seen tag ⟨tag⟩. We just have to keep track of the maximum text width associated with it.

\eqp@tempdima=\csname eqp@this#1\endcsname\relax 
\ifdim\eqp@tempdima<\eqp@tempdimb 
\expandafter\gdef\csname eqp@this#1\endcsname{\the\eqp@tempdimb}% 
\global\eqp@must@reruntrue 
\fi 
\eqp@tempdima=\csname eqp@next#1\endcsname\relax 
\ifdim\eqp@tempdima<\eqp@tempdimb 
\expandafter\gdef\csname eqp@next#1\endcsname{\the\eqp@tempdimb}% 
\fi 
\fi 
\fi 

The first time we encounter tag ⟨tag⟩ in the current document we ensure LATEX will notify the user if he needs to re-run latex on account of that tag.

\@ifundefined{eqp@seen#1}{}{% 
\expandafter\gdef\csname eqp@seen#1\endcsname{}% 
\@cons\eqp@taglist{{#1}}% 
} 

Finally, we can call \eapproducebox. We pass it \eqp@this⟨tag⟩ for its ⟨width⟩ argument and #2 for its ⟨text⟩ argument.

\eqp@tempdima=\csname eqp@this#1\endcsname\relax 
\eqp@producebox{\eqp@tempdimb}{#2}
\texttt{eqp@set@min@width}  Given a tag and a textual length, ensure that \texttt{eqp@this@}(tag) represents a width of at least \texttt{⟨length⟩}.

\begin{verbatim}
111  \def\eqp@set@min@width#1#2{% 112   \expandafter\ifx\csname eqp@this@#1\endcsname\relax
113     If we get here, then we've never encountered tag \texttt{⟨tag⟩}, even in a previous \texttt{latex} run. We assign a value to \texttt{⟨tag⟩} and request that the user re-run \texttt{latex}.
114     \global\eqp@must@reruntrue
115   \else
116     If we get here, then we have previously seen tag \texttt{⟨tag⟩}. We ensure its width is at least \texttt{#2}.
117       \eqp@tempdima=\csname eqp@this@#1\endcsname\relax
118       \eqp@tempdimb=#2\relax
119       \ifdim\eqp@tempdima<\eqp@tempdimb
120         \expandafter\xdef\csname eqp@this@#1\endcsname{\the\eqp@tempdimb}\
121       \fi
122       \eqp@tempdima=\csname eqp@next@#1\endcsname\relax
123       \ifdim\eqp@tempdima<\eqp@tempdimb
124         \expandafter\xdef\csname eqp@next@#1\endcsname{\the\eqp@tempdimb}\
125       \fi
126   \fi
127 \@ifundefined{eqp@seen@#1}{% 128     \expandafter\gdef\csname eqp@seen@#1\endcsname{}%
129     \@cons\eqp@taglist{{#1}}%
130   }{}%}
\end{verbatim}

5.3 Author macros
\texttt{eqparbox}  We want \texttt{eqparbox} to take the same arguments as \texttt{parbox}, with the same default values for the optional arguments. The only difference in argument processing is that \texttt{eqparbox} has a \texttt{(tag)} argument where \texttt{parbox} has \texttt{(width)}.

Because \texttt{eqparbox} has more than one optional argument, we can’t use a single function defined by \texttt{DeclareRobustCommand}. Instead, we have to split \texttt{eqparbox} into \texttt{eqparbox}, \texttt{eqparbox@i}, \texttt{eqparbox@ii}, and \texttt{eqparbox@iii} macros, which correspond to \texttt{parbox}, \texttt{@iparbox}, \texttt{@iiiparbox}, and \texttt{@iiiiiparbox} in \texttt{ltboxes.dtx}.

\texttt{eqparbox} takes an optional \texttt{(pos)} argument that defaults to \texttt{c}. It passes the value of this argument to \texttt{eqparbox@i}.

\begin{verbatim}
132  \DeclareRobustCommand{\eqparbox}{%
133    \@ifnextchar[% 134      {\eqparbox@i}%
135      {\eqparbox@ii[\texttt{c}][\texttt{relax}][\texttt{s}]}%
136  }
\end{verbatim}
\eqparbox
\eqparbox takes a \textit{pos} argument followed by an optional \textit{height} argument that defaults to \texttt{relax}. It passes both \textit{pos} and \textit{height} to \eqparboxii.

\begin{verbatim}
def\eqparbox@i[#1]{%
    \@ifnextchar[%
        {\eqparbox@ii[#1]}%
        {\eqparbox@iii[#1][\relax][s]}%
    }%
\end{verbatim}

\eqparboxii
\eqparboxii takes \textit{pos} and \textit{height} arguments followed by an optional \textit{inner-pos} argument that defaults to \textit{pos}. It passes \textit{pos}, \textit{height}, and \textit{inner-pos} to \eqparboxiii.

\begin{verbatim}
def\eqparbox@ii[#1][#2]{%
    \@ifnextchar[%
        {\eqparbox@iii[#1][#2]}%
        {\eqparbox@iii[#1][#2][#1]}%
    }%
\end{verbatim}

\eqparboxiii
\eqparboxiii takes \textit{pos}, \textit{height} and \textit{inner-pos} arguments. It defines an \texttt{eqp@produce@box} macro that takes a \textit{width} argument and a \textit{text} argument and passes all of \textit{pos}, \textit{height}, \textit{inner-pos}, \textit{width}, and \textit{text} to \TeX\ 's \texttt{parbox} macro. \eqparboxiii ends by calling \texttt{eqp@compute@width}, which will eventually invoke \texttt{eqp@produce@box}.

\begin{verbatim}
\long\def\eqparboxiii[#1][#2][#3][#4][#5]{%
    \parbox[#1][#2][#3][#4][#5]{#1}{#2}%
}%
\end{verbatim}

eqminipage
The \texttt{eqminipage} environment is implemented almost exactly like the \texttt{eqparbox} macro above. Just like \texttt{eqparbox}, \texttt{eqminipage} takes an optional \textit{pos} argument that defaults to c. It passes the value of this argument to \eqminipage@i.

\begin{verbatim}
\DeclareRobustCommand{\eqminipage}{%
    \@ifnextchar[%
        {\eqminipage@i}%
        {\eqminipage@iii[c][\relax][s]}%
    }%
\end{verbatim}

\eqminipage@i
\eqminipage@i takes a \textit{pos} argument followed by an optional \textit{height} argument that defaults to \texttt{relax}. It passes both \textit{pos} and \textit{height} to \eqminipage@ii.

\begin{verbatim}
\long\def\eqminipage@i[#1]{%
    \@ifnextchar[%
        {\eqminipage@ii[#1]}%
        {\eqminipage@iii[#1][\relax][s]}%
    }%
\end{verbatim}
\eqminipage@ii \eqminipage@ii takes \langle pos \rangle and \langle height \rangle arguments followed by an optional\langle inner-pos \rangle argument that defaults to \langle pos \rangle. It passes \langle pos \rangle, \langle height \rangle, and \langle inner-pos \rangle to \eqminipage@iii.

\def\eqminipage@ii[#1][#2]{%  
\@ifnextchar[\null{}\eqminipage@iii[#1][#2][#1]%}

\eqminipage@iii This is where eqminipage differs from \eqparbox. Like \eqparbox@iii, \eqminipage@iii takes \langle pos \rangle, \langle height \rangle and \langle inner-pos \rangle arguments. However, while \eqparbox@iii expects to be followed by a tag and text, \eqminipage@iii consumes the tag itself. \eqminipage@iii then uses environ’s \Collect@Body macro to collect everything up to the \end{eqminipage} into a single argument, which it passes to \eqminipage@iv.

\def\eqminipage@iii[#1][#2][#3]#4{%  
\eqminipage@iv\eqp@produce@box  
This code is a bit confusing due to the definition of a macro within a macro 
within a macro. \eqminipage@iv, which is invoked by \collect@body, is passed 
the body of the eqminipage environment as an argument. In then defines an 
\eqp@produce@box macro with the parameter list that \eqp@compute@width 
expects: a width (####1) and text (####2). \eqp@produce@box typesets a 
minipage with that width and text and the formatting parameters pro-
vided to \eqminipage@iii (#1, #2, and #3). Finally, \eqminipage@iv invokes 
\eqp@compute@width with the tag passed to \eqminipage@iii as #4 and the 
text passed to \eqminipage@iv as ##1.

\long\def\eqminipage@iv##1{%  
\long\gdef\eqp@produce@box####1####2{%  
\begin{minipage}####1####2\end{minipage}%%}

\eqp@compute@width{#4}{##1}%
\Collect@Body\eqminipage@iv

\eqmakebox \eqmakebox provides an automatic-width analogue to L\TeX’s \makebox. It takes 
the same arguments as \makebox with the same default values for the optional 
arguments. The only difference in argument processing is that \eqmakebox has a\langle tag \rangle argument where \makebox has \langle width \rangle. Note that if \langle width \rangle is not specified, 
\eqmakebox simply invokes \makebox.

\DeclareRobustCommand{\eqmakebox}{%  
\@ifnextchar{\null{}\eqlrbx@i\eqmakebox}%
\eqmakebox%}}
\eqframebox  \eqframebox provides an automatic-width analogue to LATEX's \framebox. It takes the same arguments as \framebox with the same default values for the optional arguments. The only difference in argument processing is that \eqframebox has a ⟨tag⟩ argument where \framebox has ⟨width⟩. Note that if ⟨width⟩ is not specified, \eqframebox simply invokes \framebox.

\begin{verbatim}
\DeclareRobustCommand{\eqframebox}{
\@ifnextchar[\]{\eqlrbox@i\framebox}{\framebox}
}
\end{verbatim}

\eqsavebox  \eqsavebox provides an automatic-width analogue to LATEX's \savebox. It takes the same arguments as \savebox with the same default values for the optional arguments. The only difference in argument processing is that \eqsavebox has a ⟨tag⟩ argument where \savebox has ⟨width⟩. Note that if ⟨width⟩ is not specified, \eqsavebox simply invokes \savebox.

\begin{verbatim}
\DeclareRobustCommand{\eqsavebox}{\@ifnextchar[\]{\eqlrbox@i{\savebox{#1}}}{\savebox{#1}}
\end{verbatim}

\eqlrbox@i  \eqlrbox@i takes a ⟨command⟩ argument (one of \makebox, \framebox, or \savebox{⟨cmd⟩}), a ⟨tag⟩ argument and checks if those arguments are followed by a ⟨pos⟩ argument. If not, then ⟨pos⟩ defaults to “c”. All of ⟨command⟩, ⟨tag⟩, and ⟨pos⟩ are passed to \eqlrbox@ii.

\begin{verbatim}
\def\eqlrbox@i#1[#2]{\@ifnextchar[\]{\eqlrbox@ii{#1}[#2][c]}{\eqlrbox@ii{#1}[#2][c]}}
\end{verbatim}

\eqlrbox@ii  \eqlrbox@ii takes a ⟨command⟩ argument (one of \makebox, \framebox, or \savebox{⟨cmd⟩}), a ⟨tag⟩ argument, and a ⟨pos⟩ argument. It defines \eqp@produce@box to take a ⟨width⟩ argument and a ⟨text⟩ argument and invoke ⟨command⟩[⟨width⟩][⟨pos⟩]{⟨text⟩}. \eqlrbox@ii ends by calling \eqp@compute@width, which will eventually invoke \eqp@produce@box.

\begin{verbatim}
\def\eqlrbox@ii#1[#2][#3]{\@ifnextchar[\]{\eqp@produce@box#1[#2][#3][c]}{\eqp@produce@box#1[#2][#3][c]}}
\end{verbatim}

\eqp@produce@box  For the times that the user wants to make something other than a box to match an \eqparbox's width, we provide \eqp@produce@box. \eqboxwidth returns the width of a box corresponding to a given tag. More precisely, if \eqp@this@⟨tag⟩ is defined, it's returned. Otherwise, 0pt is returned.
\texttt{eqsetminwidth} \quad The \texttt{eqsetminwidth} macro accepts a tag and a length and records that the user wants the associated box to be no narrower than the given length.

\texttt{eqsetmaxwidth} \quad The \texttt{eqsetmaxwidth} macro accepts a tag and a length and records that the user wants the associated box to be no wider than the given length.

\texttt{eqsetminwidthto} \quad The \texttt{eqsetminwidthto} macro accepts a tag and a piece of text and records that the user wants the associated box to be no narrower than the text, typeset at its natural width.

\texttt{eqsetmaxwidthto} \quad The \texttt{eqsetmaxwidthto} macro accepts a tag and a piece of text and records that the user wants the associated box to be no wider than the text, typeset at its natural width.

\section{End-of-document processing}

At the \texttt{\end{document}}, for each tag \texttt{\langle tag\rangle} we see if \texttt{\eqp@next@\langle tag\rangle}, which was initialized to 0.0pt, is different from \texttt{\eqp@this@\langle tag\rangle}, which was initialized to the maximum box width from the previous run. If so, we issue an informational message. In any case, we initialize the next run’s \texttt{\eqp@this@\langle tag\rangle} to \texttt{\eqp@next@\langle tag\rangle} and the next run’s \texttt{\eqp@next@\langle tag\rangle} to 0pt.

\AtEndDocument{\begingroup

\newcommand*{\eqboxwidth}[1]{{}\@ifundefined{eqp@this@#1}{\csname eqp@this@#1\endcsname}{\csname eqp@this@#1\endcsname}}
\newcommand{\eqsetminwidth}[2]{%\@tempdima=#2\relax\expandafter\xdef\csname eqp@minwd@#1\endcsname{\the\@tempdima}\eqp@set@min@width{#1}{\csname eqp@minwd@#1\endcsname}}
\newcommand{\eqsetmaxwidth}[2]{%\@tempdima=#2\relax\expandafter\xdef\csname eqp@maxwd@#1\endcsname{\the\@tempdima}}
\newcommand{\eqsetminwidthto}[2]{%\eqp@settowidth{\@tempdima}{#2}\expandafter\xdef\csname eqp@minwd@#1\endcsname{\the\@tempdima}\eqp@set@min@width{#1}{\csname eqp@minwd@#1\endcsname}}
\newcommand{\eqsetmaxwidthto}[2]{%\eqp@settowidth{\@tempdima}{#2}\expandafter\xdef\csname eqp@maxwd@#1\endcsname{\the\@tempdima}}

\AtEndDocument{\begingroup

\section{End-of-document processing}

At the \texttt{\end{document}}, for each tag \texttt{\langle tag\rangle} we see if \texttt{\eqp@next@\langle tag\rangle}, which was initialized to 0.0pt, is different from \texttt{\eqp@this@\langle tag\rangle}, which was initialized to the maximum box width from the previous run. If so, we issue an informational message. In any case, we initialize the next run’s \texttt{\eqp@this@\langle tag\rangle} to \texttt{\eqp@next@\langle tag\rangle} and the next run’s \texttt{\eqp@next@\langle tag\rangle} to 0pt.

\AtEndDocument{\begingroup

\newcommand*{\eqboxwidth}[1]{{}\@ifundefined{eqp@this@#1}{\csname eqp@this@#1\endcsname}{\csname eqp@this@#1\endcsname}}
\newcommand{\eqsetminwidth}[2]{%\@tempdima=#2\relax\expandafter\xdef\csname eqp@minwd@#1\endcsname{\the\@tempdima}\eqp@set@min@width{#1}{\csname eqp@minwd@#1\endcsname}}
\newcommand{\eqsetmaxwidth}[2]{%\@tempdima=#2\relax\expandafter\xdef\csname eqp@maxwd@#1\endcsname{\the\@tempdima}}
\newcommand{\eqsetminwidthto}[2]{%\eqp@settowidth{\@tempdima}{#2}\expandafter\xdef\csname eqp@minwd@#1\endcsname{\the\@tempdima}\eqp@set@min@width{#1}{\csname eqp@minwd@#1\endcsname}}
\newcommand{\eqsetmaxwidthto}[2]{%\eqp@settowidth{\@tempdima}{#2}\expandafter\xdef\csname eqp@maxwd@#1\endcsname{\the\@tempdima}}
The \texttt{eqp@taglist} list is of the form \texttt{`\@elt \{\textit{tag}_1\} \@elt \{\textit{tag}_2\} \ldots'}. We therefore locally define \texttt{\@elt} to take the name of a tag and perform all of the checking described above and then merely execute \texttt{eqp@taglist}.

\begin{verbatim}
def\@elt#1{\%
\ifundefined{eqp@minwd@#1}{\{}\%
\ifundefined{eqp@maxwd@#1}{\{}\%
\PackageWarning{eqparbox}{For tag `#1', minimum width (\csname eqp@minwd@#1\endcsname) > maximum width (\csname eqp@maxwd@#1\endcsname)}\%
\else\%
\immediate\write\@auxout{\string\expandafter\string\gdef\string\csname\string eqp@this@#1\string\endcsname{\csname eqp@next@#1\endcsname}}\%
\fi\%
\fi\%
\immediate\write\@auxout{\string\expandafter\string\gdef\string\csname\string eqp@minwd@#1\string\endcsname{\csname eqp@minwd@#1\endcsname}}\%
\fi\%
\immediate\write\@auxout{\string\expandafter\string\gdef\string\csname\string eqp@maxwd@#1\string\endcsname{\csname eqp@maxwd@#1\endcsname}}\%
\ifdim\csname eqp@this@#1\endcsname=\csname eqp@next@#1\endcsname\else\%
\@latex@warning@no@line{Rerun to correct the width of eqparbox `#1'}\%
\fi\%
\immediate\write\@auxout{\string\expandafter\string\gdef\string\csname\string eqp@this@#1\string\endcsname{\csname eqp@next@#1\endcsname}}\%
\}
\immediate\write\@auxout{\string\expandafter\string\gdef\string\csname\string eqp@next@#1\string\endcsname{0pt}}\%
\end{verbatim}

\texttt{\@ifundefined{eqp@minwd@#1}{\{}\%}
\texttt{\@ifundefined{eqp@maxwd@#1}{\{}\%}
\texttt{\PackageWarning{eqparbox}{For tag `#1', minimum width (\csname eqp@minwd@#1\endcsname) > maximum width (\csname eqp@maxwd@#1\endcsname)}\%}
\texttt{\else\%}
\texttt{\immediate\write\@auxout{\string\expandafter\string\gdef\string\csname\string eqp@this@#1\string\endcsname{\csname eqp@next@#1\endcsname}}\%}
\texttt{\immediate\write\@auxout{\string\expandafter\string\gdef\string\csname\string eqp@minwd@#1\string\endcsname{\csname eqp@minwd@#1\endcsname}}\%}
\texttt{\immediate\write\@auxout{\string\expandafter\string\gdef\string\csname\string eqp@maxwd@#1\string\endcsname{\csname eqp@maxwd@#1\endcsname}}\%}
\texttt{\ifdim\csname eqp@this@#1\endcsname=\csname eqp@next@#1\endcsname\else\%
\@latex@warning@no@line{Rerun to correct the width of eqparbox `#1'}\%
\fi\%
\immediate\write\@auxout{\string\expandafter\string\gdef\string\csname\string eqp@this@#1\string\endcsname{\csname eqp@next@#1\endcsname}}\%
\}
\immediate\write\@auxout{\string\expandafter\string\gdef\string\csname\string eqp@minwd@#1\string\endcsname{\csname eqp@minwd@#1\endcsname}}\%
\texttt{\@ifundefined{eqp@maxwd@#1}{\{}\%}
\texttt{\immediate\write\@auxout{\string\expandafter\string\gdef\string\csname\string eqp@maxwd@#1\string\endcsname{\csname eqp@maxwd@#1\endcsname}}\%}
\end{verbatim}

Also make the .aux file define \texttt{eqp@minwd\{\textit{tag}\}} and \texttt{eqp@maxwd\{\textit{tag}\}} to their current value, if any.
We output a generic “rerun latex” message if we encountered a tag that was not present on the previous run. (This is always the case on the first run or the first run after deleting the corresponding .aux file.

\ifep@must@rerun
  \@latex@warning@no@line{Rerun to correct eqparbox widths}
\fi

Change History

v1.0
  General: Initial version ............ 1

v2.0
  \@elt: Modified to allow numbers
  in tag names (suggested by
  Martin Vaeth) ............... 22
  \eqp@compute@width: Removed
  extraneous \globals
  (suggested by David Kastrup) 15
  \eqp@settowidth: Modified to
  store and restore the font
  across \ boundaries
  (suggested by Mike Shell) .... 13
  General: Rewrote to use only two
  \texttt{dimen}s total and the rest
  macros (problem reported by
  Gilles Pérez-Lambert and
  Plamen Tanovski; solution
  suggested by David Kastrup
  and Donald Arseneau) ........ 1

v2.1
  \eqboxwidth: Rewrote so as to be
  compatible with the \texttt{calc}
  package’s \texttt{setlength}
  command (problem initially
  reported by Gary L. Gray and
  narrowed down by Martin
  Vaeth) ....................... 20

v3.0
  \eqmakebox: Included Rob
  Verhoeven’s \texttt{eqmakebox} macro 19

v3.1
  \eqframebox: Introduced this
  macro ....................... 20
  \eqmakebox: Modified the
  argument processing to match
  \texttt{makebox}’s ............. 19
  \eqp@compute@width: Restructured the package to
  make all user-callable functions
  eventually call
  \eqp@compute@width, which
  does the bulk of the work .... 15
  \eqsavebox: Introduced this macro 20

v4.0
  \@elt: Modified to honor minimum
  and maximum text widths, as
  set by \texttt{eqset\{max.min\}width}
  and \texttt{eqset\{max.min\}widthto} . 22
  \eqp@settowidth: Added support
  for list environments ........ 13
  Added support for
  multi-paragraph input ...... 13
  \eqsetmaxwidth: Introduced this
  macro ....................... 21
  \eqsetmaxwidthto: Introduced
  this macro .................. 21
  \eqsetminwidth: Introduced this
  macro ....................... 21
  \eqsetminwidthto: Introduced
  this macro .................. 21
eqminipage: Introduced this environment ....................... 18 \eqsetminwidth: Defined
\eqp@this@{tag} and \eqp@next@{tag} appropriately 21
v4.1 \eqp@set@min@width: Introduced this helper macro for \eqsetminwidth
\eqsetminwidthto ...................... 17 \eqsetminwidthto: Define \eqp@this@{tag} and \eqp@next@{tag} appropriately 21

Index

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

Symbols
\@elt ...................... 229 \eqlrbbox@ii 197, 198, 200
\latex@warning@no@line .............................. 243, 274 \eqmakebox .................. 180
\listdepth 31, 34, 39, 61 \eqminipage .................. 153 \eqminipage (environment) .................. 153
\restorepar .......................... 71 \eqminipage@ii 155, 159
\setpar .............................. 67 \eqminipage@iii 156, 161, 164
\tooodeep ...................... 32 162, 166, 167, 169
\eqminipage@iv .................. 170 \eqminipage@iv ........................ 204
\eqp@compute@width 76, 151, 176, 204
\eqp@endings ..................... 20 \eqp@list@box ................................ 6
\eqp@list@indent ................. 8, 43, 55, 69
\eqp@must@rerunfalse ........... 4 \eqp@list@indent .................. 51–54
\eqp@must@reruntrue ............. 4, 90, 97, 113
\eqp@par ............................. 64 \eq@produce@box .................. 109, 147, 170, 200
\eq@produce@box .................. 109, 147, 170, 200
\eqp@restorefont .................. 10, 66 \eqp@restorefont .................. 111, 212, 221
\eq@set@min@width ................... 153 \eqp@set@min@width ................... 153
\eq@set@min@widthto ................ 17 \eq@set@min@widthto ................ 17
\eqp@set@min@widthto ................ 17 \eqp@set@min@widthto ................ 17
\@elt ........................ 229 \eq@tempdimb ............... 1
\latex@warning@no@line .............................. 243, 274 \eqparbox .................. 132
\listdepth 31, 34, 39, 61 \eqparbox@i .................. 134, 137
\restorepar .......................... 71 \eqparbox@ii ........................ 139, 142
\setpar .............................. 67 \eqparbox@iii ........................ 135,
\tooodeep ...................... 32 140, 144, 145, 147
\eqparbox .................. 187, 188
\f@baselineskip .................. 13 \eqsavebox .................. 190
\f@encoding ............................ 12 \eqsetmaxwidth .................. 214
\f@family ............................. 12 \eqsetmaxwidthto .................. 223
\f@series ............................. 12 \eqsetminwidth .................. 200
\f@shape ............................. 12 \eqsetminwidthto .................. 218
\f@size ............................. 13 \f@baselineskip .................. 13
\f@encoding ............................ 12 \f@family ............................. 12
\f@series ............................. 12
\f@shape ............................. 12
\f@size ............................. 13
\fontsize ............................. 13
\framebox .................. 187, 188
<table>
<thead>
<tr>
<th>G</th>
<th>\listparindent ...... 37</th>
</tr>
</thead>
<tbody>
<tr>
<td>\g@addto@macro 22, 25, 28</td>
<td>\RequirePackage ... 5, 9</td>
</tr>
<tr>
<td>\hspace ............ 49, 51</td>
<td>\rightmargin ... 36, 53</td>
</tr>
<tr>
<td>M</td>
<td>\makebox ...... 182, 183</td>
</tr>
<tr>
<td>\makelabel ........... 44</td>
<td>\savebox ...... 192, 193</td>
</tr>
<tr>
<td>I</td>
<td>\item .................. 41</td>
</tr>
<tr>
<td>\ifeqpmustrerun 4, 273</td>
<td>\selectfont ........... 14</td>
</tr>
<tr>
<td>\itemindent ........... 38, 54</td>
<td>\newsavebox ...... 6, 7</td>
</tr>
<tr>
<td>L</td>
<td>\PackageWarning ... 233</td>
</tr>
<tr>
<td>\labelwidth ....... 45, 46</td>
<td>\usefont ............. 12</td>
</tr>
<tr>
<td>\leftmargin ........... 52</td>
<td>\par .................. 64</td>
</tr>
<tr>
<td>list (environment) ... 30</td>
<td>\parbox ........... 149</td>
</tr>
<tr>
<td>R</td>
<td>\parallel (package) ... 8</td>
</tr>
<tr>
<td>S</td>
<td>\settowidth ......... 74</td>
</tr>
<tr>
<td>P</td>
<td>\PackageWarning ... 233</td>
</tr>
<tr>
<td>\labelwidth ....... 45, 46</td>
<td>\usefont ............. 12</td>
</tr>
<tr>
<td>\par .................. 64</td>
<td>\parbox ........... 149</td>
</tr>
<tr>
<td>W</td>
<td>\write ...... 245, 255, 263</td>
</tr>
</tbody>
</table>