The eqparbox package*

Scott Pakin
scott+eqp@pakin.org

September 3, 2017

Abstract

The eqparbox package makes it easy to define a group of boxes (such as those produced by `\parbox` or `\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 `flushright` environment and in `\parbox`es as wide as the widest text ("Rosencrantz & Guildenstern Are Dead"), they will appear as desired. Similarly, if the company names in Figure 2 are both put in a `\parbox` as wide as "Thingamabobs, Ltd.," the job titles in a `\parbox` as wide as "Senior Widget Designer," and the dates in a `\parbox` as wide as "1/95–present," then they can be spaced evenly by separating them with `\hfill`s.

The problem is in choosing the width for each set of `\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:

*This document corresponds to eqparbox v4.1, dated 2017/09/03.

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 2ε.

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`es 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 `l`, `c`, or `r` column in a `tabular`; `\eqparbox`es in the same group are analogous to cells in the same column. Unlike the cells in a `tabular` column, however, a group of `\eqparbox`es can be spread throughout the document.

2 Usage

```
\eqparbox [⟨pos⟩] [⟨height⟩] [⟨inner-pos⟩] {⟨tag⟩} {⟨text⟩}
\eqmakebox [⟨tag⟩] [⟨pos⟩] {⟨text⟩}
\eqframebox [⟨tag⟩] [⟨pos⟩] {⟨text⟩}
\eqsavebox {⟨cmd⟩} [⟨tag⟩] [⟨pos⟩] {⟨text⟩}
\begin{eqminipage} [⟨pos⟩] [⟨height⟩] [⟨inner-pos⟩] {⟨tag⟩}
  ⟨text⟩
\end{eqminipage}
```

These are almost identical to, respectively, the `\parbox`, `\makebox`, `\framebox`, and `\savebox` macros and the `minipage` environment. The key difference is that the ⟨*width*⟩ argument is replaced by a ⟨*tag*⟩ argument. (For a description of the remaining arguments, look up `\parbox`, `\makebox`, `\framebox`, `\savebox`, and `minipage` in any LaTeX 2$_\varepsilon$ book or in the `usrguide.pdf` file that comes with all TeX distributions.) ⟨*tag*⟩ 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 `latex` will need to be run a second time for the various box widths to stabilize.

```
\eqboxwidth {⟨tag⟩}
```

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 `\eqparbox` and using `\settowidth` to measure it, the `eqparbox` package defines a convenience routine called `\eqboxwidth` that achieves the same result.

`\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 `l` column, while the final cell behaves as if it were typeset in a `p` column. In actuality, the column is an `l` column; an `\eqparbox` for the first four cells ensures the column stretches appropriately while a `\parbox` of width `\eqboxwidth{`⟨*tag*⟩`}` 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 `\eqparbox` cells and `\parbox` cells.

```
\eqsetminwidth {⟨tag⟩} {⟨width⟩}
\eqsetmaxwidth {⟨tag⟩} {⟨width⟩}
```

These macros override the width calculation for boxes associated with tag ⟨*tag*⟩, ensuring that they are no narrower than a given minimum (`\eqsetminwidth`) and no wider than a given maximum (`\eqsetmaxwidth`).

Table 1: A `tabular` that stretches to fit some cells while forcing others to wrap

Wide
Wider
Wider than that
This is a fairly wide cell
While this cell's text wraps, the previous cells (whose text doesn't wrap) determine the width of the column.

`\eqsetminwidthto {`⟨*tag*⟩`} {`⟨*text*⟩`}`
`\eqsetmaxwidthto {`⟨*tag*⟩`} {`⟨*text*⟩`}`

These macros are analogous to `\eqsetminwidth` and `\eqsetmaxwidth` but automatically compute the natural width of the given text and use that as the minimum (`\eqsetminwidthto`) or maximum (`\eqsetmaxwidthto`) width for boxes using tag ⟨*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 an `\eqparbox`-based helper macro, `\showquote`:

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

⋮

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

```
\showquote{Italian proverb}{}
```

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 `tabular`s

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:

```
2017-02-22 & \eqparbox{topiclist}{%
               \begin{enumerate}
                 \item Hardware upgrades
```

Table 2: Lists within a `tabular`

Meeting date	Topics discussed
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
```

3.3 Hanging indentation

Consider the paragraphs depicted in Figure 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 `list` environment that typesets its labels with `\eqparbox` this is fairly straightforward:

```
\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, ...
```

```
  \item[The final exciting thing]  I am fantastic.  Blah,
     blah, blah, ...
\end{list}
```

Stuff about me	I am great. Blah, blah.
More stuff	I am wonderful. Blah, blah. Did I mention that blah, blah, blah, blah, blah, blah, blah, blah, blah, blah, blah, blah, blah, blah, blah?
The final exciting thing	I am fantastic. Blah, blah.

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 4. 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 ⟨*pos*⟩ argument, which causes the ⟨*text*⟩ 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`

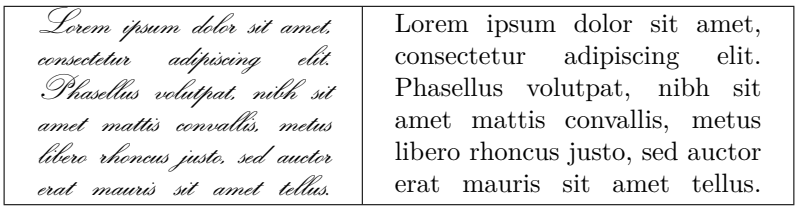

Figure 4: Line-by-line transcription of text with full justification

macro to define a new `tabular` column type, "`S`", that stretches whitespace as needed to fit the widest line in the column:

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

That code works by storing the current cell's contents within a box called `\tstretchbox` then passing `\tstretchbox`'s contents to `\eqmakebox`. (The `tabular` environment does not enable a cell's contents to be passed directly to a macro, hence the `lrbox` trickery.) Note that the "`S`" column type takes an argument, which is the tag to pass to `\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 calligra package.

```
\begin{tabular}{|l|l|}
  \hline
  \calligra
  \begin{tabular}{S{handwritten}}
    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}
  &
  \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} \\
  \hline
\end{tabular}
```

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 `\eqsetmaxwidth` macro (and the array package's `\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:

```
\newcolumntype{M}[1]{%
  >{\begin{lrbox}{\csname#1box\endcsname}}%
  l%
  <{\end{lrbox}%
    \eqparbox[t]{#1}{\unhcopy\csname#1box\endcsname\strut}}%
}
```

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

```
\eqsetmaxwidth{maybebig}{0.5\linewidth}
\newsavebox{\maybebigbox}
\begin{tabular}{|M{maybebig}|l|}
  \hline
  Very short & Good \\
  A little bit longer & Okay \\
\end{tabular}
```

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:

```
\eqsetmaxwidth{maybebig}{0.5\linewidth}
\newsavebox{\maybebigbox}
\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}
```

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

Very short	Good
A little bit longer	Okay

(a) Output when the text is narrow

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

(b) Output when the text is wide

Figure 5: Combining the features of `l` and `p` columns

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 `\eqmakebox[`⟨*tag*⟩`][r]{`⟨*text*⟩`}` and center that:

```
\begin{tabular}{@{}lccc@{}} \hline
  & \multicolumn{3}{c}{Sales (in millions)} \\ \cline{2-4}
  \multicolumn{1}{c}{\raisebox{1ex}[2ex]{Product}} &
  October & November & December \\ \hline

  Widgets      & \eqmakebox[oct][r]{           55.2} &
                 \eqmakebox[nov][r]{\bfseries  89.2} &
                 \eqmakebox[dec][r]{           57.9} \\
  Doohickeys   & \eqmakebox[oct][r]{\bfseries  65.0} &
                 \eqmakebox[nov][r]{\tiny       N/A} &
                 \eqmakebox[dec][r]{            9.3} \\
  Thingamabobs & \eqmakebox[oct][r]{           10.4} &
                 \eqmakebox[nov][r]{            8.0} &
                 \eqmakebox[dec][r]{\bfseries 109.7} \\ \hline
\end{tabular}
```

Table 3: Sample sales data

Product	Sales (in millions)		
	October	November	December
Widgets	55.2	**89.2**	57.9
Doohickeys	**65.0**	N/A	9.3
Thingamabobs	10.4	8.0	**109.7**

4 Limitations

Unfortunately, eqparbox's macros have a number of limitations not exhibited by the corresponding LaTeX 2_ε commands. First, eqparbox's macros internally typeset the given text within a `tabular` environment—specifically, using "`@{}l@{}`" as the template—in order to determine the text's natural width. Consequently, commands not valid within such a `tabular` (e.g., `verbatim` environments) are also not valid within the ⟨*text*⟩ argument of an eqparbox macro. As a corollary, the macros defined by the eqparbox package can appear only where a `tabular` is also acceptable.

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

5 Implementation

The one-sentence summary of the implementation is, "As 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 `.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

`\eqp@tempdima` `\eqp@tempdimb`

Define a couple temporary ⟨*dimen*⟩s for use in a variety of locations.

```
\newlength{\eqp@tempdima}
\newlength{\eqp@tempdimb}
```

`\eqp@taglist`

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

```
\def\eqp@taglist{}
```

`\ifeqp@must@rerun` `\eqp@must@reruntrue` `\eqp@must@rerunfalse`

If an `eqparbox` is wider than the maximum-width `eqparbox` with the same tag, we need to store the new maximum width and request that the user re-run `latex`. We use `\ifeqp@must@rerun` and `\eqp@must@reruntrue` to assist with this.

```
\newif\ifeqp@must@rerun
```

The `\eqp@settowidth` macro requires the array package's ability to inject code into every cell.

```
\RequirePackage{array}
```

`\eqp@tabular@box` `\eqp@list@box`

The `\eqp@settowidth` macro requires a box, `\eqp@tabular@box`, in which to store the entire input text. `\eqp@settowidth` also requires a box, `\eqp@list@box`, in which to store nested `list` environments.

```
\newsavebox{\eqp@tabular@box}
\newsavebox{\eqp@list@box}
```

`\eqp@list@indent` The `\eqp@settowidth` macro stores the accumulated `list` indentation in `\eqp@list@indent`.

```
\newlength{\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.

```
\RequirePackage{environ}
```

5.2 Width calculation

`\eqp@storefont` `\eqp@restorefont` 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
  }%
}
```

`\eqp@settowidth` 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` 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,`

description}=\endlist. 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
```

list 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{} \\
  \sbox\@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}%
}%
```

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}%
```

```
      \advance\eqp@list@indent by \leftmargin
      \advance\eqp@list@indent by \rightmargin
      \advance\eqp@list@indent by \itemindent
      \global\setbox\eqp@list@box=\hbox\bgroup
      \begin{tabular}{@{}l@{}}%
    }{%
      \item[]%
      \end{tabular}%
      \egroup
      \global\advance\@listdepth\m@ne
    }%
    \eqp@endings
```

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

```
    \global\let\eqp@par=\par
    \eqp@storefont
    \begin{tabular}{@{}>{\eqp@restorefont}l<{\eqp@storefont}@{}}%
      \global\@setpar{\\}%
      #2%
      \\ \box\eqp@list@box
    \end{tabular}%
    \global\@restorepar
  }%
\endgroup
```

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

```
  \settowidth{#1}{\box\eqp@tabular@box}%
}
```

`\eqp@compute@width` The following function does all the real work for the eqparbox package. It takes two parameters—⟨*tag*⟩ and ⟨*text*⟩—and ensures that all boxes with the same tag will be as wide as the widest box with that tag. It ends by passing ⟨*tag*⟩ and ⟨*text*⟩ to the `\eqp@produce@box` command, which was defined by the calling macro to produce a box using one of the existing LaTeX 2_ε commands.

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

```
\long\def\eqp@compute@width#1#2{%
  \eqp@settowidth{\eqp@tempdimb}{#2}%
```

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 `\eqparbox` with tag ⟨*tag*⟩ are left-justified), but it's better to be safe than sorry.

```
  \global\eqp@must@reruntrue
  \expandafter\xdef\csname eqp@this@#1\endcsname{\the\eqp@tempdimb}%
  \expandafter\xdef\csname eqp@next@#1\endcsname{\the\eqp@tempdimb}%
\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\xdef\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\xdef\csname eqp@next@#1\endcsname{\the\eqp@tempdimb}%
  \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 `\eqp@produce@box`. 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@produce@box{\eqp@tempdima}{#2}%
}
```

`\eqp@set@min@width` Given a tag and a textual length, ensure that `\eqp@this@`⟨*tag*⟩ represents a width of at least ⟨*length*⟩.

```
\def\eqp@set@min@width#1#2{%
  \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 assign a value to ⟨*tag*⟩ and request that the user re-run `latex`.

```
    \global\eqp@must@reruntrue
    \expandafter\xdef\csname eqp@this@#1\endcsname{#2}%
    \expandafter\xdef\csname eqp@next@#1\endcsname{#2}%
  \else
```

If we get here, then we *have* previously seen tag ⟨*tag*⟩. We ensure its width is at least `#2`.

```
    \eqp@tempdima=\csname eqp@this@#1\endcsname\relax
    \eqp@tempdimb=#2\relax
    \ifdim\eqp@tempdima<\eqp@tempdimb
      \expandafter\xdef\csname eqp@this@#1\endcsname{\the\eqp@tempdimb}%
    \fi
    \eqp@tempdima=\csname eqp@next@#1\endcsname\relax
    \ifdim\eqp@tempdima<\eqp@tempdimb
      \expandafter\xdef\csname eqp@next@#1\endcsname{\the\eqp@tempdimb}%
    \fi
  \fi
  \@ifundefined{eqp@seen@#1}{%
    \expandafter\gdef\csname eqp@seen@#1\endcsname{}%
    \@cons\eqp@taglist{{#1}}%
  }{}%
}
```

5.3 Author macros

`\eqparbox` We want `\eqparbox` to take the same arguments as `\parbox`, with the same default values for the optional arguments. The only difference in argument processing is that `\eqparbox` has a ⟨*tag*⟩ argument where `\parbox` has ⟨*width*⟩.

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

`\eqparbox` takes an optional ⟨*pos*⟩ argument that defaults to `c`. It passes the value of this argument to `\eqparbox@i`.

```
\DeclareRobustCommand{\eqparbox}{%
  \@ifnextchar[%]
    {\eqparbox@i}%
    {\eqparbox@iii[c][\relax][s]}%
}
```

`\eqparbox@i` `\eqparbox@i` takes a ⟨*pos*⟩ argument followed by an optional ⟨*height*⟩ argument that defaults to `\relax`. It passes both ⟨*pos*⟩ and ⟨*height*⟩ to `\eqparbox@ii`.

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

`\eqparbox@ii` `\eqparbox@ii` takes ⟨*pos*⟩ and ⟨*height*⟩ arguments followed by an optional ⟨*inner-pos*⟩ argument that defaults to ⟨*pos*⟩. It passes ⟨*pos*⟩, ⟨*height*⟩, and ⟨*inner-pos*⟩ to `\eqparbox@iii`.

```
\def\eqparbox@ii[#1][#2]{%
  \@ifnextchar[%]
    {\eqparbox@iii[#1][#2]}%
    {\eqparbox@iii[#1][#2][#1]}%
}
```

`\eqparbox@iii` `\eqparbox@iii` takes ⟨*pos*⟩, ⟨*height*⟩ and ⟨*inner-pos*⟩ arguments. It defines an `\eqp@produce@box` macro that takes a ⟨*width*⟩ argument and a ⟨*text*⟩ argument and passes all of ⟨*pos*⟩, ⟨*height*⟩, ⟨*inner-pos*⟩, ⟨*width*⟩, and ⟨*text*⟩ to LaTeX's `\parbox` macro. `\eqparbox@iii` ends by calling `\eqp@compute@width`, which will eventually invoke `\eqp@produce@box`.

```
\def\eqparbox@iii[#1][#2][#3]{%
  \long\gdef\eqp@produce@box##1##2{%
    \parbox[#1][#2][#3]{##1}{##2}%
  }%
  \eqp@compute@width
}
```

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

```
\DeclareRobustCommand{\eqminipage}{%
  \@ifnextchar[%]
    {\eqminipage@i}%
    {\eqminipage@iii[c][\relax][s]}%
}
\let\endeqpminipage=\relax
```

`\eqminipage@i` `\eqminipage@i` takes a ⟨*pos*⟩ argument followed by an optional ⟨*height*⟩ argument that defaults to `\relax`. It passes both ⟨*pos*⟩ and ⟨*height*⟩ to `\eqminipage@ii`.

```
\long\def\eqminipage@i[#1]{%
  \@ifnextchar[%]
    {\eqminipage@ii[#1]}%
    {\eqminipage@iii[#1][\relax][s]}%
}
```

`\eqminipage@ii` `\eqminipage@ii` takes ⟨*pos*⟩ and ⟨*height*⟩ arguments followed by an optional ⟨*inner-pos*⟩ argument that defaults to ⟨*pos*⟩. It passes ⟨*pos*⟩, ⟨*height*⟩, and ⟨*inner-pos*⟩ to `\eqminipage@iii`.

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

`\eqminipage@iii` This is where `eqminipage` differs from `\eqparbox`. Like `\eqparbox@iii`, `\eqminipage@iii` takes ⟨*pos*⟩, ⟨*height*⟩ and ⟨*inner-pos*⟩ 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 provided 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][#3]{####1}%
        ####2%
      \end{minipage}%
    }%
    \eqp@compute@width{#4}{##1}%
  }%
  \Collect@Body\eqminipage@iv
}
```

`\eqmakebox` `\eqmakebox` provides an automatic-width analogue to LaTeX'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 ⟨*tag*⟩ argument where `\makebox` has ⟨*width*⟩. Note that if ⟨*width*⟩ is not specified, `\eqmakebox` simply invokes `\makebox`.

```
\DeclareRobustCommand{\eqmakebox}{%
  \@ifnextchar[%]
    {\eqlrbox@i\makebox}%
    {\makebox}%
}
```

`\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`.

```
\DeclareRobustCommand{\eqframebox}{%
  \@ifnextchar[%]
    {\eqlrbox@i\framebox}%
    {\framebox}%
}
```

`\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`.

```
\DeclareRobustCommand{\eqsavebox}[1]{%
  \@ifnextchar[%]
    {\eqlrbox@i{\savebox{#1}}}%
    {\savebox{#1}}%
}
```

`\eqlrbox@i` `\eqlrbox@i` takes a `{`⟨*command*⟩`}` argument (one of `\makebox`, `\framebox`, or `\savebox{`⟨*cmd*⟩`}`) and 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`.

```
\def\eqlrbox@i#1[#2]{%
  \@ifnextchar[%]
    {\eqlrbox@ii{#1}[#2]}%
    {\eqlrbox@ii{#1}[#2][c]}%
}
```

`\eqlrbox@ii` `\eqp@produce@box` `\eqlrbox@i` 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`.

```
\def\eqlrbox@ii#1[#2][#3]{%
  \long\gdef\eqp@produce@box##1##2{%
    #1[##1][#3]{##2}%
  }%
  \eqp@compute@width{#2}%
}
```

`\eqboxwidth` For the times that the user wants to make something other than a box to match an `\eqparbox`'s width, we provide `\eqboxwidth`. `\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.

```
\newcommand*{\eqboxwidth}[1]{%
  \@ifundefined{eqp@this@#1}{0pt}{\csname eqp@this@#1\endcsname}%
}
```

`\eqsetminwidth` The `\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.

```
\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}%
}
```

`\eqsetmaxwidth` The `\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.

```
\newcommand{\eqsetmaxwidth}[2]{%
  \@tempdima=#2\relax
  \expandafter\xdef\csname eqp@maxwd@#1\endcsname{\the\@tempdima}%
}
```

`\eqsetminwidthto` The `\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.

```
\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}%
}
```

`\eqsetmaxwidthto` The `\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.

```
\newcommand{\eqsetmaxwidthto}[2]{%
  \eqp@settowidth{\@tempdima}{#2}%
  \expandafter\xdef\csname eqp@maxwd@#1\endcsname{\the\@tempdima}%
}
```

5.4 End-of-document processing

At the `\end{document}`, for each tag ⟨*tag*⟩ we see if `\eqp@next@`⟨*tag*⟩, which was initialized to `0.0pt`, is different from `\eqp@this@`⟨*tag*⟩, 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 `\eqp@this@`⟨*tag*⟩ to `\eqp@next@`⟨*tag*⟩ and the next run's `\eqp@next@`⟨*tag*⟩ to `0pt`.

```
\AtEndDocument{%
  \begingroup
```

`\@elt` The `\eqp@taglist` list is of the form "`\@elt {`⟨tag_1⟩`} \@elt {`⟨tag_2⟩`} …". We therefore locally define `\@elt` to take the name of a tag and perform all of the checking described above and then merely execute `\eqp@taglist`.

```
\def\@elt#1{%
```

Complain if the tag's minimum width is greater than its maximum width.

```
  \@ifundefined{eqp@minwd@#1}{}{%
    \@ifundefined{eqp@maxwd@#1}{}{%
      \ifdim\csname eqp@minwd@#1\endcsname>\csname eqp@maxwd@#1\endcsname
        \PackageWarning{eqparbox}{For tag `#1',
          minimum width (\csname eqp@minwd@#1\endcsname) >
          maximum width (\csname eqp@maxwd@#1\endcsname)}%
      \fi
    }%
  }%
```

Make the `.aux` file define `\eqp@this@`⟨tag⟩ to the current value of `\eqp@next@`⟨tag⟩ and `\eqp@next@`⟨tag⟩ to `0pt`.

```
  \eqp@tempdima\csname eqp@this@#1\endcsname\relax
  \eqp@tempdimb\csname eqp@next@#1\endcsname\relax
  \ifdim\eqp@tempdima=\eqp@tempdimb
  \else
    \@latex@warning@no@line{Rerun to correct the width of eqparbox `#1'}%
  \fi
  \immediate\write\@auxout{%
    \string\expandafter\string\gdef\string\csname\space
    eqp@this@#1\string\endcsname{%
      \csname eqp@next@#1\endcsname
    }%
    ^^J%
    \string\expandafter\string\gdef\string\csname\space
     eqp@next@#1\string\endcsname{0pt}%
  }%
```

Also make the `.aux` file define `\eqp@minwd@`⟨tag⟩ and `\eqp@maxwd@`⟨tag⟩ to their current value, if any.

```
  \@ifundefined{eqp@minwd@#1}{}{%
    \immediate\write\@auxout{%
      \string\expandafter\string\gdef\string\csname\space
      eqp@minwd@#1\string\endcsname{%
        \csname eqp@minwd@#1\endcsname
      }%
    }%
  }%
  \@ifundefined{eqp@maxwd@#1}{}{%
    \immediate\write\@auxout{%
      \string\expandafter\string\gdef\string\csname\space
      eqp@maxwd@#1\string\endcsname{%
        \csname eqp@maxwd@#1\endcsname
      }%
```

```
      }%
    }%
  }%
  \eqp@taglist
\endgroup
```

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.

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

Change History

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.