

The `eqparbox` package*

Scott Pakin
scott+eqp@pakin.org

2004/08/02

Abstract

The `eqparbox` package makes it easy to define a group of `\parboxes` 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 Table 1, in which the numbers are right-justified relative to each other but centered as a group within each column. And second, how would you typeset the résumé excerpt shown in Figure 1 while meeting the following requirements:

1. The header columns must be left-justified relative to each other.
2. The headers 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-width boxes. If the data in Table 1 are put into equal-sized `\parboxes`, each containing a `\raggedleft` for right-justification, the `\parboxes` can then be centered to achieve the desired result. Similarly, if the company names in Figure 1 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 `\hfills`.

The problem is in choosing the width for each set of `\parboxes`. For Table 1, this isn't too difficult, because digits are the same width as each other in most fonts. Each `\parbox`, therefore, need be only as wide as the largest sequence of

*This file has version number v2.1, last revised 2004/08/02.

Table 1: Sample sales data

Product	Sales (in millions)		
	October	November	December
Widgets	55.2	89.2	57.9
Doohickeys	65.0	64.1	9.3
Thingamabobs	10.4	8.0	109.7

Widgets, Inc.	Senior Widget Designer	1/95–present
<ul style="list-style-type: none"> • 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
<ul style="list-style-type: none"> • 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 $\text{\LaTeX} 2_{\epsilon}$. 		

Figure 1: Excerpt from a sample résumé

digits expected. Figure 1 is more of a bother. The user must typeset the résumé once to see which entry in each column is the widest and then assign lengths appropriately:

```

\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-width entry in each column. If only there were a simpler way to keep a set of `\parboxes` as wide as the widest entry in the set...

That simpler way is the `eqparbox` package. `eqparbox` exports a macro, `\eqparbox`, which 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`; `\eqparboxes` in the same group are analogous to cells in the same column.

2 Usage

`\eqparbox` The primary macro in the `eqparbox` package is `\eqparbox`. Usage is almost identical to that of `\parbox`:

```
\eqparbox [<pos>] [<height>] [<inner-pos>] {<tag>} {<text>}
```

The only difference is that, where `\parbox` has its *<width>* argument, `\eqparbox` has *<tag>*. (For a description of the remaining arguments, look up `\parbox` in any $\text{\LaTeX} 2_{\epsilon}$ book or in the `usrguide.tex` file that comes with $\text{\LaTeX} 2_{\epsilon}$.) *<tag>* can be any valid identifier. All `\eqparboxes` with the same tag will be 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 `\eqparboxes` 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` It is sometimes useful to take the width of an `\eqparbox` to use in other \LaTeX 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 2. Table 2's only column expands to fit the widest cell in the column, excluding the final cell. The

Table 2: 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.

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.

3 Examples

Figure 1'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}}

:

```

Table 1 was entered as follows:

```

\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      & \eqparbox{oct}{\raggedleft      55.2 } &
              \eqparbox{nov}{\raggedleft\textbf{ 89.2}} &
              \eqparbox{dec}{\raggedleft      57.9 } \\
Doohickeys   & \eqparbox{oct}{\raggedleft\textbf{ 65.0}} &
              \eqparbox{nov}{\raggedleft      64.1 } &
              \eqparbox{dec}{\raggedleft      9.3 } \\
Thingamabobs & \eqparbox{oct}{\raggedleft      10.4 } &
              \eqparbox{nov}{\raggedleft      8.0 } &
              \eqparbox{dec}{\raggedleft\textbf{109.7}} \\ \hline
\end{tabular}

```

Note that the above can be simplified by defining a macro that combines `\eqparbox` and `\raggedleft`. Furthermore, because the numeric data being typeset are all approximately the same width, a single tag could reasonably replace `oct`, `nov`, and `dec`. As it stands, the code serves more as an illustration than as an optimal way to typeset Table 1.

Finally, Table 2 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}

```

As an additional example, consider the paragraphs depicted in Figure 2. 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. (Can any word processor do that, incidentally?) By using a custom `list` environment which 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, ...

```


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.

```

1 \*package>

\eqp@tempdima Define a couple temporary <dimen>s for use in a variety of locations.
\eqp@tempdimb 2 \newlength{\eqp@tempdima} \newlength{\eqp@tempdimb}

\eqp@taglist Define a list of all of the tags we encountered in the author’s document.
3 \def\eqp@taglist{}

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

```

At the `\end{document}`, for each tag $\langle tag \rangle$ 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`.

```

5 \AtEndDocument{%
6 \begingroup

```

`\@elt` The `\eqp@taglist` list is of the form “`\@elt {<tag1>} \@elt {<tag2>} ...`”. 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`.

```

7 \def\@elt#1{%
8 \eqp@tempdima\csname eqp@this@#1\endcsname\relax
9 \eqp@tempdimb\csname eqp@next@#1\endcsname\relax
10 \ifdim\eqp@tempdima=\eqp@tempdimb
11 \else
12 \@latex@warning@no@line{Rerun to correct the width of eqparbox ‘#1’}%
13 \fi
14 \immediate\write\@auxout{%
15 \string\expandafter\string\gdef\string\csname\space
16 eqp@this@#1\string\endcsname{%
17 \csname eqp@next@#1\endcsname
18 }%
19 ^^J%
20 \string\expandafter\string\gdef\string\csname\space
21 eqp@next@#1\string\endcsname{0pt}%
22 }%
23 }%

```

```

24 \eqp@taglist
25 \endgroup

```

We output the 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.

```

26 \ifeqp@must@rerun
27 \@latex@warning@no@line{Rerun to correct eqparbox widths}
28 \fi
29 }

```

`\eqp@storefont` To find the natural width of a piece of text, we put it in a table and take the
`\eqp@restorefont` 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.

```

30 \newcommand*{\eqp@storefont}{%
31 \xdef\eqp@restorefont{%
32 \noexpand\usefont{\f@encoding}{\f@family}{\f@series}{\f@shape}%
33 \noexpand\fontsize{\f@size}{\f@baselineskip}%
34 \noexpand\selectfont
35 }%
36 }

```

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

```

37 \RequirePackage{array}

```

`\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`.

```

38 \newcommand{\eqp@settowidth}[2]{%
39 \settowidth{#1}{{%
40 \eqp@storefont
41 \begin{tabular}{@{}>{\eqp@restorefont}l<{\eqp@storefont}@{}}%
42 #2%
43 \end{tabular}}%
44 }}%
45 }

```

`\eqparbox` We want `\eqparbox` to take the same arguments as `\parbox`, with the same default
`\eqparbox@i` values for the optional arguments. The only difference in argument processing is
`\eqparbox@ii` 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 the following four macros:

`\eqparbox` Takes zero or more optional arguments. First optional argument defaults to `c`. Calls `\eqparbox@i`.

`\eqparbox@i` Takes one or more optional arguments. Second optional argument defaults to `\relax`. Calls `\eqparbox@ii`.

`\eqparbox@ii` Takes two or more optional arguments. Third optional argument defaults to `s` if either of the first two arguments is absent or to the first argument if both are present. Calls `\eqparbox@iii`.

`\eqparbox@iii` Takes three optional arguments and two mandatory arguments. Does all the work for `\eqparbox`.

Note the direct correspondence between these macros and `ltboxes.dtx`'s `\parbox`, `\@iparbox`, `\@iiparbox`, and `\@iiiparbox` macros.

```

46 \DeclareRobustCommand\eqparbox{%
47   \@ifnextchar[%
48     {\eqparbox@i}%
49     {\eqparbox@iii[c][\relax][s]}%
50 }
51 \def\eqparbox@i[#1]{%
52   \@ifnextchar[%
53     {\eqparbox@ii[#1]}%
54     {\eqparbox@iii[#1][\relax][s]}%
55 }
56 \def\eqparbox@ii[#1][#2]{%
57   \@ifnextchar[%
58     {\eqparbox@iii[#1][#2]}%
59     {\eqparbox@iii[#1][#2][#1]}%
60 }

```

`\eqparbox@iii` The following function does all the real work for `\eqparbox`. It takes five parameters—`<pos>`, `<height>`, `<inner-pos>`, `<tag>`, and `<text>`—and ensures that all boxes with the same tag will be as wide as the widest box with that tag.

To keep track of box widths, `\eqparbox` 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 dest 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>`.

```

61 \def\eqparbox@iii[#1][#2][#3]#4#5{%
62   \eqp@settowidth{\eqp@tempdimb}{#5}%
63   \expandafter
64   \ifx\csname eqp@this@#4\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 $\langle tag \rangle$ are left-justified), but it’s better to be safe than sorry.

```

65   \global\eqp@must@reruntrue
66   \expandafter\xdef\csname eqp@this@#4\endcsname{\the\eqp@tempdimb}%
67   \expandafter\xdef\csname eqp@next@#4\endcsname{\the\eqp@tempdimb}%
68   \else

```

If we get here, then we *have* previously seen tag $\langle tag \rangle$. We just have to keep track of the maximum text width associated with it.

```

69   \eqp@tempdima=\csname eqp@this@#4\endcsname\relax
70   \ifdim\eqp@tempdima<\eqp@tempdimb
71     \expandafter\xdef\csname eqp@this@#4\endcsname{\the\eqp@tempdimb}%
72     \global\eqp@must@reruntrue
73   \fi

74   \eqp@tempdima=\csname eqp@next@#4\endcsname\relax
75   \ifdim\eqp@tempdima<\eqp@tempdimb
76     \expandafter\xdef\csname eqp@next@#4\endcsname{\the\eqp@tempdimb}%
77   \fi
78   \fi

```

The first time we encounter tag $\langle tag \rangle$ in the current document we ensure L^AT_EX will notify the user if he needs to re-run `latex` on account of that tag.

```

79   \ifundefined{eqp@seen@#4}{%
80     \expandafter\gdef\csname eqp@seen@#4\endcsname{}%
81     \@cons\eqp@taglist{#4}%
82   }{}%

```

Finally, we can call `\parbox`. We pass it $\langle pos \rangle$, $\langle height \rangle$, $\langle inner-pos \rangle$, and $\langle text \rangle$ directly, and we pass it `\eqp@this@ $\langle tag \rangle$` for its $\langle width \rangle$ argument.

```

83   \eqp@tempdima=\csname eqp@this@#4\endcsname\relax
84   \parbox[#1][#2][#3]{\eqp@tempdima}{#5}%
85 }

```

`\eqboxwidth` For the times that the user wants to make something other than a box 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@ $\langle tag \rangle$` is defined, it’s returned. Otherwise, `0pt` is returned.

```

86 \newcommand*{\eqboxwidth}[1]{%
87   \ifundefined{eqp@this@#1}{0pt}{\csname eqp@this@#1\endcsname}%
88 }

```

Per-tag memory usage The `eqparbox` package defines three macros for each unique tag: `\eqp@this@ $\langle tag \rangle$` , `\eqp@next@ $\langle tag \rangle$` , and `\eqp@seen@ $\langle tag \rangle$` . Consequently, each unique tag subtracts three strings from T_EX’s string pool and three multiletter control sequences from that pool. In addition, each unique tag $\langle tag \rangle$ subtracts $|\code{\eqp@this@}| + |\langle tag \rangle| + |\code{\eqp@next@}| + |\langle tag \rangle| + |\code{\eqp@seen@}| + |\langle tag \rangle| = 27 + 3 \times |\langle tag \rangle|$ string characters from T_EX’s pool of string characters. For example, a document that invokes `\eqparbox` with tags “hello” and “goodbye”

will utilize $3 \times 2 = 6$ strings, $3 \times 2 = 6$ multiletter control sequences, and $(27 + 3 \times 5) + (27 + 3 \times 7) = 90$ T_EX string characters.

89 </package>

6 Future Work

The following are some of the features people have requested I implement in `eqparbox`:

- Evaluate `\eqparbox`'s $\langle text \rangle$ argument exactly once in case it contains side effects. Currently, $\langle text \rangle$ is evaluated twice: once to determine its natural size and once to put it in a parbox of the width associated with the given tag. (Feature requested by Bernd Schandl.)
- Get `eqparbox` to work properly within an algorithmic environment. (Feature requested by Mike Shell.)
- Support pre-1999 L^AT_EX 2_ε. (Feature requested by Mike Shell.)

One idea I've been toying with that may resolve the first two items in that list is to typeset $\langle text \rangle$ within a parbox, then check `\badness` to see if the box was too small for $\langle text \rangle$. If not (i.e., the box was not overfull), then the initial parbox can be reused directly without needing to re-typeset $\langle text \rangle$. With this approach, `\eqparbox` will work anywhere `\parbox` works. Unfortunately, I don't know how to get at the `\badness` of the `\hboxes` that comprise the `\vbox` utilized by `\parbox`. If anyone has a suggestion, I'm all ears.

Change History

v1.0		store and restore the font across
General: Initial version	1	<code>\</code> boundaries (suggested by
v2.0		Mike Shell)
<code>\@elt</code> : Modified to allow numbers		<code>\eqparbox@iii</code> : Removed extrane-
in tag names (suggested by		ous <code>\globals</code> (suggested by
Martin Vaeth)	7	David Kastrup)
General: Rewrote to use only two		
$\langle dimen \rangle$ s total and the rest		v2.1
macros (problem reported by		<code>\eqboxwidth</code> : Rewrote so as to be
Gilles Pérez-Lambert and Pla-		compatible with the <code>calc</code> pack-
men Tanovski; solution sug-		age's <code>\setlength</code> command
gested by David Kastrup and		(problem initially reported by
Donald Arseneau)	1	Gary L. Gray and narrowed
<code>\eqp@settowidth</code> : Modified to		down by Martin Vaeth)
		10

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	<code>\eqp@tempdima</code>	<code>\fontsize</code> 33
<code>\@cons</code> 81	. . <u>2</u> , 8, 10, 69,	
<code>\@elt</code> <u>7</u>	70, 74, 75, 83, 84	I
<code>\@latex@warning@no@line</code>	<code>\eqp@tempdimb</code>	<code>\ifeqp@must@rerun</code> <u>4</u> , 26
. 12, 27	<u>2</u> , 9, 10, 62, 66,	
	67, 70, 71, 75, 76	P
A	<code>\eqparbox</code> <u>46</u>	<code>\parbox</code> 84
<code>\AtEndDocument</code> 5	<code>\eqparbox@i</code> <u>46</u>	R
	<code>\eqparbox@ii</code> <u>46</u>	<code>\RequirePackage</code> . . . 37
	<code>\eqparbox@iii</code>	S
	. 49, 54, 58, 59, <u>61</u>	<code>\selectfont</code> 34
E	F	<code>\settowidth</code> 39
<code>\eqboxwidth</code> <u>86</u>	<code>\f@baselineskip</code> . . . 33	
<code>\eqp@must@rerunfalse</code> <u>4</u>	<code>\f@encoding</code> 32	U
<code>\eqp@must@reruntrue</code>	<code>\f@family</code> 32	<code>\usefont</code> 32
. <u>4</u> , 65, 72	<code>\f@series</code> 32	
<code>\eqp@restorefont</code> <u>30</u> , 41	<code>\f@shape</code> 32	W
<code>\eqp@settowidth</code> . <u>38</u> , 62	<code>\f@size</code> 33	<code>\write</code> 14
<code>\eqp@storefont</code> <u>30</u> , 40, 41		
<code>\eqp@taglist</code> . . <u>3</u> , 24, 81		