The eqparbox package*

Scott Pakin
pakin@uiuc.edu

2001/04/19

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-widthed boxes. if the data in Table 1 are put into equal-sized

*This file has version number v1.00, last revised 2001/04/19.

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**

- 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 1: Excerpt from a sample résumé

`\parbox`es, each containing a `\raggedleft` for right-justification, the `\parbox`es 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 `\hfill`s.

The problem is in choosing the width for each set of `\parbox`es. 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 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-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 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`; `\eqparbox`es 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 LaTeX 2ε book or in the `usrguide.tex` file that comes with LaTeX 2ε.) ⟨*tag*⟩ can be any valid identifier. All `\eqparbox`es 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 `\eqparbox`es 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 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%
```

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.

```
\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}
```

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 2: Paragraphs with hanging indentation

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 utilizes code similar to the following:

```
\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}}{%
    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, ...

  \item[More stuff]  I am wonderful.  Blah, blah, blah, ...

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

4 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` `\eqp@tempdimb` `\eqp@tempdimc`

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

```
2 \newlength{\eqp@tempdima}
3 \newlength{\eqp@tempdimb}
4 \newlength{\eqp@tempdimc}
```

`\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.

```
5 \newif\ifeqp@must@rerun
6
7 \AtEndDocument{%
8   \ifeqp@must@rerun
9     \@latex@warning@no@line{Rerun to correct eqparbox widths}
10  \fi
11 }
```

`\eqp@settowidth`

This macro is just like `\settowidth`, but it puts its argument in a `tabular`, which means that it can contain `\\`. Is there a better way to find the natural width of something like "`This is split \\ across lines.`"?

```
12 \def\eqp@settowidth#1#2{%
```

```
  \settowidth{#1}{\begin{tabular}{@{}l@{}}#2\end{tabular}}%
}
```

\eqparbox
\eqparbox@i
\eqparbox@ii

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 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.

```
\DeclareRobustCommand\eqparbox{%
  \@ifnextchar[%
    {\eqparbox@i}%
    {\eqparbox@iii[c][\relax][s]}%
}
\def\eqparbox@i[#1]{%
  \@ifnextchar[%
    {\eqparbox@ii[#1]}%
    {\eqparbox@iii[#1][\relax][s]}%
}
\def\eqparbox@ii[#1][#2]{%
  \@ifnextchar[%
    {\eqparbox@iii[#1][#2]}%
    {\eqparbox@iii[#1][#2][#1]}%
}
```

\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 three global variables for each tag: \eqp@⟨*tag*⟩, \eqp@first⟨*tag*⟩, and \eqp@next⟨*tag*⟩. \eqp@⟨*tag*⟩ is the maximum width seen so far for tag ⟨*tag*⟩. It is initialized to \eqp@first@⟨*tag*⟩, if defined, otherwise to the width of ⟨*text*⟩. \eqp@next@⟨*tag*⟩ works the same way, but is always initialized to 0.0pt. At the end of a run, eqparbox prepares the

next run (via the `.aux` file) to initialize `\eqp@first@`⟨*tag*⟩ to the final value of `\eqp@next@`⟨*tag*⟩.

`\eqp@next@`⟨*tag*⟩ is needed to detect whether the widest text with tag ⟨*tag*⟩ has been removed/shrunk. `\eqp@first@`⟨*tag*⟩ is needed so `\eqp@`⟨*tag*⟩ can be initialized to it, while `\eqp@next@`⟨*tag*⟩ is initialized to `0.0pt`.

```
\def\eqparbox@iii[#1][#2][#3]#4#5{%
  \expandafter%
  \ifx\csname eqp@#4\endcsname\relax
```

If we get here, then this is the first use of ⟨*tag*⟩ in this document. In the following `\ifx` statement, we initialize `\eqp@`⟨*tag*⟩ to the value of `\eqp@first@`⟨*tag*⟩, if defined, otherwise to the width of ⟨*text*⟩.

```
    \expandafter\global\expandafter\newlength\csname eqp@#4\endcsname
    \expandafter\global\expandafter\newlength\csname eqp@next@#4\endcsname
    \expandafter%
    \ifx\csname eqp@first@#4\endcsname\relax
```

If we didn't encounter tag ⟨*tag*⟩ on our previous run, then 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
      \global\eqp@settowidth{\csname eqp@#4\endcsname}{#5}%
    \else
      \global\csname eqp@#4\endcsname=\csname eqp@first@#4\endcsname\relax
    \fi
```

At the `\end{document}`, we see if `\eqp@next@`⟨*tag*⟩, which was initialized to `0.0pt`, is smaller than `\eqp@`⟨*tag*⟩, which was initialized to the maximum box width from the previous run. If so, we initialize the next run's `\eqp@first@`⟨*tag*⟩ to `\eqp@next@`⟨*tag*⟩ and tell the user to re-run `latex`, because the widest box with tag ⟨*tag*⟩ must have been removed or shrunk. Otherwise, we initialize the next run's `\eqp@first@`⟨*tag*⟩ to `\eqp@`⟨*tag*⟩.

```
    \AtEndDocument{%
      \expandafter\let\expandafter\eqp@tempdima\csname eqp@next@#4\endcsname
      \expandafter\let\expandafter\eqp@tempdimb\csname eqp@#4\endcsname
      \ifnum\eqp@tempdima<\eqp@tempdimb
        \@latex@warning@no@line{Rerun to correct width of eqparbox `#4'}
        \immediate\write\@auxout{%
          \string\global\string\newdimen%
            \expandafter\string\csname eqp@first@#4\endcsname^^J%
          \string\global\expandafter\string\csname eqp@first@#4\endcsname=%
            \expandafter\the\eqp@tempdima\string\relax
        }
      \else
        \immediate\write\@auxout{%
          \string\global\string\newdimen%
            \expandafter\string\csname eqp@first@#4\endcsname^^J%
          \string\global\expandafter\string\csname eqp@first@#4\endcsname=%
            \expandafter\the\eqp@tempdimb\string\relax
        }
```

```
      \fi
    }%
  \fi
```

Each invocation, we check to see if ⟨*text*⟩ is wider than the previous maximum for tag ⟨*tag*⟩. If so, we set `\eqp@must@reruntrue`, so the user will later be notified to re-run `latex`. The next run will start with the maximum width of `\eqp@`⟨*tag*⟩.

```
  \expandafter\let\expandafter\eqp@tempdima\csname eqp@#4\endcsname
  \expandafter\let\expandafter\eqp@tempdimb\csname eqp@next@#4\endcsname
  \eqp@settowidth{\eqp@tempdimc}{#5}%
  \ifnum\eqp@tempdima<\eqp@tempdimc
    \global\eqp@tempdima=\eqp@tempdimc\relax
    \eqp@must@reruntrue
  \fi
```

Increase `\eqp@next@`⟨*tag*⟩ to the width of ⟨*text*⟩, if necessary.

```
  \ifnum\eqp@tempdimb<\eqp@tempdimc
    \global\eqp@tempdimb=\eqp@tempdimc\relax
  \fi
```

Finally, we can call `\parbox`. We pass it ⟨*pos*⟩, ⟨*height*⟩, ⟨*inner-pos*⟩, and ⟨*text*⟩ directly, and we pass it `\eqp@`⟨*tag*⟩ for its ⟨*width*⟩ argument.

```
  \parbox[#1][#2][#3]{\eqp@tempdima}{#5}%
}
```

`\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@`⟨*tag*⟩ is defined, it's returned. Otherwise, if if `\eqp@first@`⟨*tag*⟩ is defined, it's returned. Otherwise, `0.0pt` is returned.

Because we use `\def` to define `\eqboxwidth` and we return only ⟨*dimen*⟩s, it's legal to precede `\eqboxwidth` with `\the` or anything else that expects to be followed by a ⟨*dimen*⟩.

```
\def\eqboxwidth#1{%
  \expandafter%
  \ifx\csname eqp@#1\endcsname\relax
    \expandafter%
    \ifx\csname eqp@first@#1\endcsname\relax
      \z@
    \else
      \csname eqp@first@#1\endcsname
    \fi
  \else
    \csname eqp@#1\endcsname
  \fi
}
```

⟨/package⟩

Index

Numbers written in italic refer to the page where the corresponding entry is described, the ones underlined to the code line of the definition, the rest to the code lines where the entry is used.