The eqparbox package*

Scott Pakin
scott+eqp@pakin.org

January 2, 2010

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 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 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 data in Table 1 are put into equal-sized `\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

*This document corresponds to eqparbox v3.1, dated 2010/01/01.

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 2$_\varepsilon$.

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-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*⟩}

These macros are almost identical to `\parbox`, `\makebox`, `\framebox`, and `\savebox`, respectively. 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`, and `\savebox` in any LaTeX 2ε 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 `\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 a box produced by one of the pre-

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.

ceding 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%
\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 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}%
```

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

```
    \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}
```

Finally, 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 ⟨*pos*⟩ argument, which causes

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.	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.

Figure 3: Line-by-line transcription of text with full justification

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` 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 3 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}
```

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., list environments) are also not valid within the ⟨*text*⟩ argument of an eqparbox macro. As a corollary, eqparbox's macros 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.

`\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
```

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{%
      \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}%
      }%
    }%
    \eqp@taglist
  \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.

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

`\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
```

```
  }%
}
```

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

```
\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`.

```
\newcommand{\eqp@settowidth}[2]{%
  \settowidth{#1}{{%
    \eqp@storefont
    \begin{tabular}{@{}>{\eqp@restorefont}l<{\eqp@storefont}@{}}%
      #2%
    \end{tabular}%
  }}%
}
```

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

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

`\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$_\varepsilon$ 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 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*⟩.

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

`\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}%
}
```

Per-tag memory usage The eqparbox package defines three macros for each unique tag: `\eqp@this@`⟨*tag*⟩, `\eqp@next@`⟨*tag*⟩, and `\eqp@seen@`⟨*tag*⟩. Consequently, each unique tag subtracts three strings from TeX's string pool and three multiletter control sequences from that pool. In addition, each unique tag ⟨*tag*⟩ subtracts $|\texttt{\textbackslash eqp@this@}|+|\langle tag\rangle|+|\texttt{\textbackslash eqp@next@}|+|\langle tag\rangle|+|\texttt{\textbackslash eqp@seen@}|+|\langle tag\rangle| = 27 + 3 \times |\langle tag\rangle|$ string characters from TeX'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 $(27 + 3 \times 5) + (27 + 3 \times 7) = 90$ TeX string characters.

6 Future Work

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

- Evalaute `\eqparbox`'s ⟨*text*⟩ argument exactly once in case it contains side effects. Currently, ⟨*text*⟩ is evaluated twice: once to determine its natural size and once to put it in a box 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 LaTeX 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 ⟨*text*⟩ within a box, then check `\badness` to see if the box was too small for ⟨*text*⟩. If not (i.e., the box was not overfull), then the initial box can be reused directly without needing to re-typeset ⟨*text*⟩. With this approach, `\eqparbox` will work anywhere `\parbox` works. Unfortunately, I don't know how to get at the `\badness` of the `\hbox`es that comprise the `\vbox` utilized by `\parbox`. If anyone has a suggestion, I'm all ears.

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.