The pbox package[*]

Simon Law
sfllaw@engmail.uwaterloo.ca

June 5, 2005

1 Introduction

Most skilled LaTeX users are familiar with the various box commands. These commands include `\makebox`, `\framebox`, `\savebox`, and `\parbox`. These boxes takes a parameter that specifies the width of box to create. To simplify matters, there are the `\mbox`, `\fbox`, and `\sbox` commands that fit the box created to the size of its contents. Conspicuously absent, however, is a `\pbox` command.

2 A variable-width `\parbox`

At first glance, it seems quite inappropriate to create a `\pbox` command. After all, the size of a multi-line box will most likely be limited by the `\textwidth` or `\columnwidth` of the text it encloses. When a line of text is too long, it will be wrapped by TeX's own line-breaking algorithms. However, there are certain circumstances where one would want a variable-width `\parbox`.

For example, you may want to align the top and bottom lines of multi-line boxes. The simplest way to do this is with `parbox` commands with an arbitrary width.

```
Hello
World!     Bonjour
           monde!
```

```
\parbox[b]{2cm}{Hello\\World!}%
\parbox[t]{2cm}{Bonjour\\monde!}
```

`\pbox` However, this is not convenient. It may take several attempts to guess the correct width; and if there was ever a need to change the contents of the boxes, then the hard-coded widths must be changed as well. It would ideal to have a box that would collapse to the minimal required width.

```
Hello
World!Bonjour
      monde!
```

```
\pbox[b]{0.5\textwidth}{Hello\\World!}%
\pbox[t]{0.5\textwidth}{Bonjour\\monde!}
```

Notice how the exclaimation mark and the capital B have no extra space between each other, implying that `\pbox` creates minimal-width boxes.

[*]Version v1.0, last revised 2003/04/06

3 Determining minimum widths

This is all well and good, but how does one measure the width of one of these boxes? Well, a rather painful way would be to use `\settowidth` in conjunction with a `\parbox`. But it is far easier to do it with the new width commands.

`\settominwidth` The `\settominwidth` command works very similarly to the standard `\settowidth` command.

> `\settowidth[`*max_width*`]{`*cmd*`}{`*text*`}` sets the value of the a length command *cmd* equal to the width of the multi-line *text*. The optional argument *max_width* allows you to specify the maximum width that will be returned; it defaults to `\columnwidth`.

`\widthofpbox` To provide completeness for the calc package, the `\widthofpbox` command was implemented to complement the `\widthof` command.

> `\widthofpbox{`*text*`}` returns the width of the multi-line *text*.

Here is an example:

I
need
support

```
I\\
need\\
support\\
\rule{\widthofpbox{I\\need\\support}}{0.4pt}
```

4 Limitations

Unfortunately, there are some limitations in this package. One of the intrinsic limitations is that you cannot do anything in a `\pbox` that you could not do in a `\parbox`. This seems quite reasonable, so it should not be a hardship.

Since `\pbox` is implemented using the **tabular** environment, there are some things that cannot, and should not be used. You should note that errant `&` characters within a `\pbox` do not generate meaningful error messages. As well, it is unfortunate that `\linebreak` and `\newline` do not work as expected.

Since it is a box, you cannot use the **verbatim** environment within. I recommend that you use the fancyvrb package which contains the **BVerbatim** and **LVerbatim** environments for typesetting boxed verbatim text.

Alas, I have also discovered that certain uses of `\widthof` and `\widthofpbox` do not work within the docstrip environment.

5 Implementation

I use the standard calc package for general math. As well, I wish to support a `\widthofpbox` command, so I will demand that the `\widthof` command exists as well.

```
\RequirePackage{calc}
```

In order to perform `\lengthtest`s and `\equality` tests, I need to include the standard ifthen package. This also provides me with simple conditionals.

```
\RequirePackage{ifthen}
```

`\settominwidth` The minimum length is determined by the clever use of the **tabular** environment. It knows how to calculate the minimum requisite width for a column, and the way determines the end of a column is with its end of row command `\\` . This command is conveniently similar to the command typically used to break lines.

As you can see, `#1` defaults to the width of a column. This will either be `\textwidth` or the width defined by the **twocolumn** option, or even the multicol package.

```
\newcommand{\settominwidth}[3][\columnwidth]{%
```

Here, I set the length command `#2`. Notice the argument to the **tabular** environment. I use `@{}` to eliminate any horizontal padding, and use the `l` alignment to grab the width of the text in `#3`.

```
  \settowidth{#2}{\begin{tabular}{@{}l@{}}#3\end{tabular}}%
```

Finally, I wish to make sure that the length I have set in `#2` is not larger than the maximum stored in `#1`.

```
  \ifthenelse{\lengthtest{#1<#2}}{\setlength{#2}{#1}}}
```

`\widthofpbox` In order to find the width of a `\pbox`, I use the same **tabular** trick from `\settominwidth`. I use the `\widthof` command in order to preserve its semantics instead of trying to emulate them using my `\settominwidth` command.

I do *not* check against a maximum length here. Restricting this command to a maximum length would mean that I throw away length information if the text is too long.

```
\newcommand{\widthofpbox}[1]{%
  \widthof{\begin{tabular}{@{}l@{}}#1\end{tabular}}}
```

`\pbox` `\pb@xi` `\pb@xii` It is not possible to implement `\pbox` in a simple way. The command definition commands in LaTeX don't afford you more than one optional parameter; however, `\parbox` has three.

In order to faithfully simulate the three optional arguments, I must trick LaTeX in to catching three optional arguments [1]. Therefore `\pbox`, `\pb@xi`, and `\pb@xii` are used to capture the optional arguments in the `\pb@xargi`, `\pb@xargii`, and `\pb@xargiii` commands. These are then passsed to `\pb@xiii` for actual processing.

```
\DeclareRobustCommand*{\pbox}[1][]{%
  \def\pb@xargi{#1}%
  \pb@xi}
\DeclareRobustCommand*{\pb@xi}[1][]{%
  \def\pb@xargii{#1}%
  \pb@xii}
\DeclareRobustCommand*{\pb@xii}[1][]{%
  \def\pb@xargiii{#1}%
  \pb@xiii}
```

`\pb@xiii` In order to create the final paragraph box, I parse out the two manditory arguments. I then use the provided maximal length `#1` to determine the actual width of the `\parbox`.

```
\newlength{\pb@xlen}
\DeclareRobustCommand{\pb@xiii}[2]{%
  \settominwidth[#1]{\pb@xlen}{#2}%
```

Since the default optional arguments are all empty, I should be able to just pass them to `\parbox`. However, `\parbox` interprets empty optional values differently than just non-existant optional values. So, I must make complicated decisions; if an optional argument is empty, then I will just skip it..

```
\ifthenelse{\equal{\pb@xargi}{}}
  {\parbox{\pb@xlen}{#2}}
  {\ifthenelse{\equal{\pb@xargii}{}}
    {\ifthenelse{\equal{\pb@xargiii}{}}
      {\parbox[\pb@xargi]{\pb@xlen}{#2}}
      {\parbox[\pb@xargi][][\pb@xargiii]{\pb@xlen}{#2}}}
    {\ifthenelse{\equal{\pb@xargiii}{}}
      {\parbox[\pb@xargi][\pb@xargii]{\pb@xlen}{#2}}
      {\parbox[\pb@xargi][\pb@xargii][\pb@xargiii]{\pb@xlen}{#2}}}}%
```

Finally, I must clean up the optional arguments and remove their special meaning. As well, I will terminate the `\parbox` I have created with an empty `\makebox` in order to prevent the `\def\pb@x...\relax` commands from interfering with other commands that expect `\pbox` to solely consist of a box.

```
\def\pb@xargi\relax
\def\pb@xargii\relax
\def\pb@xargiii\relax
\makebox[0pt]{}}
```

References

[1] Robin Fairbanks. "A command with two optional arguments." *TEX Frequently Asked Questions.* `http://www.tex.ac.uk/cgi-bin/texfaq2html?label=twooptarg` (current 6 April 2003.)