

The `pbox` package*

Simon Law
sfillaw@sfillaw.ca

November 24, 2011

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	¹	<code>\parbox[b]{2cm}{Hello\\World!}%</code>
	monde!	²	<code>\parbox[t]{2cm}{Bonjour\\monde!}</code>

`\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		¹	<code>\pbox[b]{0.5\textwidth}{Hello\\World!}%</code>
monde!		²	<code>\pbox[t]{0.5\textwidth}{Bonjour\\monde!}</code>

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

*Version v1.1, last revised 2011/11/24

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

1	<code>I\\</code>
2	<code>need\\</code>
3	<code>support\\</code>
4	<code>\rule{\widthofpbox{I\\need\\support}}{0.4pt}</code>

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.

```
1 \RequirePackage{calc}
```

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

```
2 \RequirePackage{ifthen}
```

`\setminwidth` 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.

```
3 \newcommand{\setminwidth}[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`.

```
4 \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`.

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

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

```
6 \newcommand{\widthofpbox}[1]{%
```

```
7 \widthof{\begin{tabular}{@{}l@{}}#1\end{tabular}}}
```

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

In order to faithfully simulate the three optional arguments, I must trick L^AT_EX 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 passed to `\pb@xiii` for actual processing.

```
8 \DeclareRobustCommand*{\pbox}[1][]{%
```

```
9 \def\pb@xargi{#1}%
```

```

10 \pb@xi}
11 \DeclareRobustCommand*\pb@xi}[1] [] {%
12   \def\pb@xargii{#1}%
13   \pb@xii}
14 \DeclareRobustCommand*\pb@xii}[1] [] {%
15   \def\pb@xargiii{#1}%
16   \pb@xiii}

```

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

```

17 \newlength{\pb@xlen}
18 \DeclareRobustCommand*\pb@xiii}[2] {%
19   \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-existent optional values. So, I must make complicated decisions; if an optional argument is empty, then I will just skip it..

```

20   \ifthenelse{\equal{\pb@xargi}{}}{
21     {\parbox{\pb@xlen}{#2}}
22     {\ifthenelse{\equal{\pb@xargii}{}}{
23       {\ifthenelse{\equal{\pb@xargiii}{}}{
24         {\parbox[\pb@xargi]{\pb@xlen}{#2}}
25         {\parbox[\pb@xargi] [] [\pb@xargiii]{\pb@xlen}{#2}}}
26       {\ifthenelse{\equal{\pb@xargiii}{}}{
27         {\parbox[\pb@xargi] [\pb@xargii]{\pb@xlen}{#2}}
28         {\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.

```

29 \def\pb@xargi\relax
30 \def\pb@xargii\relax
31 \def\pb@xargiii\relax
32 \makebox[0pt]{}

```

References

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

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.

P			
<code>\pb@xargi</code>	9, 20, 24, 25, 27–29	<code>\pbox</code>	1, <u>8</u>
<code>\pb@xargii</code>	12, 22, 27, 28, 30	<code>\protect</code>	5
<code>\pb@xargiii</code>	15, 23, 25, 26, 28, 31	S	
<code>\pb@xi</code>	<u>8</u>	<code>\settominwidth</code>	2, <u>3</u> , 19
<code>\pb@xii</code>	<u>8</u>	W	
<code>\pb@xiii</code>	16, <u>17</u>	<code>\widthof</code>	7
<code>\pb@xlen</code>	17, 19, 21, 24, 25, 27, 28	<code>\widthofpbox</code>	2, <u>6</u>

Change History

v1.0		<code>\settominwidth</code> : Always return a
General: Initial release.	1	robust value.
v1.1		
General: Switch to GPLv3.	1	