The **rjlpshap** class*

Robert J Lee
`latex@rjlee.homelinux.org`

July 9, 2009

1 Introduction

This package provides low-level helper macros and environments. It is intended for authors of LaTeX packages, who wish to programmatically change the shape of paragraphs. It overcomes several difficulties with TeX's `\parshape` register:

1. It is not possible to re-use a length register when issuing a `\parshape` command; this makes it difficult to calculate the shape of a paragraph in LaTeX using some algorithms that re-use one or more variables (registers).
2. The number of lines must be known *before* the lengths of each line, which can lead to difficult calculations.
3. The format of the `\parshape` command is very strict and evaluated early on by TeX, and so it doesn't easily allow for parameter-replacement, making it difficult to use any calculations at all.

It differs from the `shapepar` package, in that it provides no actual calculations for paragraph shapes. Instead, it provides a framework by which paragraph shapes can be calculated in a single run of the document by the author (or package writer).

The package provides both environments and commands by which the author can accumulate any number of lengths and then use them in a `\parshape` command injected into the document, without the need to re-run LaTeX.

---

*This document corresponds to rjlpshap v1.0, dated 2004/11/05.

2 Usage

There are two modes of operation: environment mode and command mode. Environment mode has the advantage that it does not require the number of lengths to be known in advance, while command mode provides various options for passing pre-calculated lengths.

2.1 Environment Mode

Environment mode uses a single environment, `parshapecollect`, which starts and ends before the paragraph of text. The contents of the environment are a LaTeX program, supplied by the user, that calculates the lengths of each line.

`parshapecollect` The `parshapecollect` environment goes at the start of a paragraph, and the entire environment defines a single `\parshape` command. Any even number of lengths are collected, and at the end of the environment, the `\parshape` command is automatically generated and inserted into the document.

`\parshapelenout` This macro takes one mandatory parameter, which is the next length in the `\parshape` command. Note that the parameter is expanded: the value of the length at the time `\parshapelenout` is called is the value which is used — this means the same length command can be re-used repeatedly. The value may be a length register, and the same register may be re-used after changing its contents, for example:

```
\newlength{\foo}
\begin{parshapecollect}
   \setlength{\foo}{1in}\parshapelenout{\foo}
   \setlength{\foo}{3in}\parshapelenout{\foo}
   \setlength{\foo}{0.5in}\parshapelenout{\foo}
   \setlength{\foo}{3.5in}\parshapelenout{\foo}
   \setlength{\foo}{0in}\parshapelenout{\foo}
   \setlength{\foo}{4in}\parshapelenout{\foo}
\end{parshapecollect}
\lipsum[1]
```

Produces:

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

`\parshapearrlenout` This macro is the same as `\parshapelenout`, but allows the programmer to pass the length as the value from an array (see the `arrayjob` package). This is intended for situations where pre-calculated lengths are to be used alongside calculations; to pass an array of lengths the various command modes may be easier. The parameters to the macro are the name of the array and the name of a counter containing the index.

For example, the above paragraph could have been created using:

```
\newarray{foo} \newcounter{ctr}
\begin{parshapecollect}
   \foo(1)={1in}\setcounter{ctr}{1}\parshapearrlenout{foo}{ctr}
   \foo(2)={3in}\setcounter{ctr}{2}\parshapearrlenout{foo}{ctr}
   \foo(3)={0.5in}\setcounter{ctr}{3}\parshapearrlenout{foo}{ctr}
   \foo(1)={3.5in}\setcounter{ctr}{1}\parshapearrlenout{foo}{ctr}
   \foo(2)={0in}\setcounter{ctr}{2}\parshapearrlenout{foo}{ctr}
   \foo(2)={4in}\parshapearrlenout{foo}{ctr}
\end{parshapecollect}
\lipsum[1]
```

2.2 Command Mode

Command mode enables the author to use calculated shapes by outputting the \parshape command as a single operation at the end of the calculation.

`\parshapeary` This macro takes one parameter, being the name of an array. The contents of the array are simply expanded and passed directly to `\parshape` — the first parameter is the number of lines, then for each line the left margin and line width are supplied in subsequent elements.

For example:

```
\newcommand{\ltest}{50pt}
\newlength{\test}\setlength{\test}{300pt}
\newarray{sizes}\readarray{sizes}{2&0em&\the\test&\ltest&\test}
\parshapeary{sizes}
\lipsum[1]
```

produces:

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

`\parshapearray` This macro takes two parameters. The first is also the name of an array. The only difference to `\parshapeary` is that instead of passing the width of the line, the width of the right margin is passed instead. This means the macro needs to know the width of the line.

The second parameter is the length of the line. This is usually \columnwidth, passing any other value will effectively change the right margin. (\columnwidth is often equal to \textwidth, but in multi-column mode, \columnwidth is more likely to be correct, unless you want to overstrike the text with text in another column — so \columnwidth is the recommended length to use).

In a similar vein to the previous example:

```
\newcommand{\ltest}{50pt}
\newlength{\test}\setlength{\test}{75pt}
\newlength{\cwidth}\setlength{\cwidth}{\columnwidth}
\addtolength{\cwidth}{-50pt}

\newarray{sizes}\readarray{sizes}{2&0em&\the\test&\ltest&\test}
\parshapearray{sizes}{\cwidth}
\lipsum[1]
```

which produces:

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

`\Parshapearray` This macro also takes two parameters, and is used to combine multiple shapes, where the width of the paragraph is not consistent throughout the paragraph.

The first parameter is the name of an array, in the same format as the first parameter to `\parshapeary`.

The second parameter is the name of a second array. The first element is the number of lines in this second array, and subsequent elements are the widths of lines (excluding margins) in that array — starting from the first line. As usual, if an insufficient number of lines are passed, the last line will be re-used.

The actual width of a line will be the total width excluding margins (as per the second parameter), minus the left margin (taken from the first parameter, even indexes), minus the right margin (as per the first parameter, odd indexes).

In this way, it is possible to set a shaped paragraph that allows for non-straight margins or non-rectangular paper — for example, to typeset a list item in such a way as to allow for a cut-out on the page.

For a larger example, consider that the paper has a sloped shape that extends 1em wider for every line for the first 15 lines only. The author wishes to set a text in an quadrilateral, with margins offset $\frac{1}{8}$" to the right for each line, but extending equally to follow the width of the paper.

```
\newarray{quad}
\readarray{quad}{17 & 0.000in & 2.000in & 0.125in & 1.875in &
  0.250in & 1.750in & 0.375in & 1.625in & 0.500in & 1.500in &
  0.625in & 1.375in & 0.750in & 1.250in & 0.875in & 1.125in &
  1.000in & 1.000in & 1.125in & 0.875in & 1.250in & 0.750in &
  1.375in & 0.675in & 1.500in & 0.500in & 1.625in & 0.375in &
  1.750in & 0.250in & 1.875in & 0.125in & 2.000in & 0.000in}
\newarray{papershape}
\readarray{papershape}{15& 25em & 26em & 27em & 28em & 29em & 30em & 31em
           & 32em & 33em & 34em & 35em & 36em & 37em & 38em & 39em}
\Parshapearray{quad}{papershape} \lipsum[1]
```

Produces:

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo.

Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

3 Prerequisites

This package requires the `arrayjob` package from CTAN[1]. Arrays are used as a convenient mechanism to pass values to functions, although they may not be used in environment mode.

4 Limitations and Possible Improvements

- Usage of this package can be slow, because two temporary files are used for each \parshape. Theoretically, only one should be needed and so it should be possible to improve this.
- There is no support for combining arrays, other than to change the paper size itself. Extra commands could be provided to calculate the intersection of two cutouts, or to add margins together.
- In a future version, it may be possible to avoid temporary files completely by using token registers, or \aftergroup, with some form of string processing.

5 Package Name

The somewhat obscure name “rjlpshap” was chosen following the guidelines given in *LaTeX 2ε for Class and Package Writers*[2] (section 2.7.3, “Make it Portable”). The common prefix used by these packages is ”rjl–”, which was chosen as the author’s initials as he is not writing these packages as part of any organisation.

The postfix of “–pshap” was chosen as a contraction of “parshape”, the TeX command to which this package is a front end. The name was contracted due to the 5-character length restriction. It was also considered that that the obvious package name of “parshape” would be best avoided to avoid confusion with the existing shapepar[3] package, and also because parshape is a standard TeX command —

---

[1] http://tug.ctan.org/cgi-bin/ctanPackageInformation.py?id=arrayjob

[2] `http://www.tex.ac.uk/tex-archive/macros/latex/doc/clsguide.pdf`

[3] `http://www.ctan.org/tex-archive/macros/latex/contrib/shapepar/`

the only requirement that exists is to avoid the name of a standard package, but avoiding the name of a standard macro also seems like a good idea as this leaves the name free for use as a standard package name in the future.

6 Other References

The following references were essential in producing this package:

- The TEXBook[4]
- TEXLive package contributions[5]
- TDS Guidelines[6]

7 Implementation

Arrayjob: we need to pass arrays of lengths and length-commands, so build on the established array handling functionality.

```
\RequirePackage{arrayjob}
```

Forloop: we need to iterate over the arrays that are passed in

```
\RequirePackage{forloop}
```

Internal commands use @ as a letter, not a special character, so switch catcodes

```
\makeatletter
```

7.1 Environment mode

`parshapecollect` Environment to handle writing the file, cleaning up etc.

Note the importance of the at the end of each line here: we're in horizontal mode so any space characters get expanded and added into the start of the paragraph. Any spaces here are output before \parshape itself, and so will be visible in the output page. The \ignorespaces is a last line of defence against any extra spaces inadvertently introduced by the user.

```
\newenvironment{parshapecollect}{%
  \immediate\openout\rjlpshap@writer=\jobname.parshape%
  \setcounter{rjlpshap@linecount}{0}%
  \setboolean{rjlpshap@isodd}{false}%
  \ignorespaces
}{%
  \immediate\closeout\rjlpshap@writer%
```

---

[4] ISBN: 978-0201134483

[5] http://tug.org/texlive/pkgcontrib.html

[6] http://dante.ctan.org/TDS-guidelines.html

Because the environment forms a group, we need to use aftergroup to delay the \parshape command. Otherwise the group will end first and \parshape will be discarded.

```
  \rjlpshap@doparshape%
  \aftergroup\input\aftergroup{\aftergroup\jobname\aftergroup%
    .\aftergroup%
    p\aftergroup%
    a\aftergroup%
    r\aftergroup%
    s\aftergroup%
    h\aftergroup%
    a\aftergroup%
    p\aftergroup%
    e\aftergroup%
    t\aftergroup%
    m\aftergroup%
    p\aftergroup}\ignorespaces
}
```

`\rjlpshap@doparshape`

Internal method to handle the final processing of the built up \parshape command. The lengths have been written to the first temporary file, so write the entire \parshape command to the second one, ready to be input.

```
\newcommand{\rjlpshap@doparshape}{%
  \immediate\openout\rjlpshap@finalwriter=\jobname.parshapetmp%
  \immediate\write\rjlpshap@finalwriter{\noexpand\parshape \arabic{rjlpshap@linecount}}%
  \openin\rjlpshap@reader=\jobname.parshape%
  \setboolean{rjlpshap@loop}{true}%
  \whiledo{\boolean{rjlpshap@loop}}{%
    \read\rjlpshap@reader to\rjlpshap@temp%
%   NB: \ifeof matches \loop...\repeat
    \ifeof\rjlpshap@reader%
    \setboolean{rjlpshap@loop}{false}%
    \else
    \immediate\write\rjlpshap@finalwriter{\rjlpshap@temp }%
    \fi
  }%
  \immediate\closeout\rjlpshap@finalwriter%
  \immediate\closein\rjlpshap@reader%
  \let\rjlpshap@temp=\relax%
}
```

`\parshapelenout`

Used inside a parshapecollect environment, writes length #1 to the \parshape command file. (If used anywhere else, it'll just write to the console since the file will not be open).

```
\newcommand{\parshapelenout}[1]{%
  \setlength{\rjlpshap@tempwidth}{#1}%
  \immediate\write\rjlpshap@writer{\the\rjlpshap@tempwidth}%
  \ifthenelse{\boolean{rjlpshap@isodd}}{%
    \setboolean{rjlpshap@isodd}{false}%
  }{%
    \stepcounter{rjlpshap@linecount}%
    \setboolean{rjlpshap@isodd}{true}%
  }%
  \ignorespaces
}
```

`\parshapearrlenout` Used inside a parshapecollect environment, writes a length to the \parshape command file. As \parshapelenout, except that the element is passed as the member of an array.

Length is taken from array with name #1, element at index #2

```
\newcommand{\parshapearrlenout}[2]{%
  \testarray{#1}(\arabic{#2})%
  \parshapelenout{\temp@macro}%
  \ignorespaces
}
```

7.2 Command Mode

`\parshapeary` As \parshape, but takes one argument being an array name defined in arrayjob form. The array contains the arguments, starting with the count at index 1, then one length per array element.

```
\newcommand{\parshapeary}[1]{%
  \testarray{#1}(1)%
  \edef\rjlpshap@max{\temp@macro}%
  \begin{parshapecollect}%
    \forloop{rjlpshap@ctr}{1}{\not \value{rjlpshap@ctr} > \rjlpshap@max }{%
      \setcounter{rjlpshap@ctr2}{\value{rjlpshap@ctr}}%
      \addtocounter{rjlpshap@ctr2}{\value{rjlpshap@ctr}}%
      \parshapearrlenout{#1}{rjlpshap@ctr2}%
      \stepcounter{rjlpshap@ctr2}%
      \parshapearrlenout{#1}{rjlpshap@ctr2}%
    }%
  \end{parshapecollect}%
  \typeout{parshapeary end}%
  \ignorespaces
}
```

`\parshapearray` As \parshapeary, but the second parameter is the name of a single-parameter macro taking a number and returning the width of the line on that line

```
\newcommand{\parshapearray}[2]{%
  \begin{parshapecollect}%
    \testarray{#1}(1)%
    \edef\rjlpshap@max{\temp@macro}%
    \forloop{rjlpshap@ctr}{1}{\not \value{rjlpshap@ctr} > \rjlpshap@max }{%
      \setcounter{rjlpshap@ctr2}{\value{rjlpshap@ctr}}%
      \addtocounter{rjlpshap@ctr2}{\value{rjlpshap@ctr}}%
      \edef\rjlpshap@pos{\arabic{rjlpshap@ctr2}}%
      \testarray{#1}(\rjlpshap@pos)%
      \setlength{\rjlpshap@tempwidth}{\temp@macro}%
      \setlength{\rjlpshap@templinewidth}{#2}%
      \addtolength{\rjlpshap@templinewidth}{-1\rjlpshap@tempwidth}%
      \parshapelenout{\rjlpshap@tempwidth}%
      \stepcounter{rjlpshap@ctr2}%
      \edef\rjlpshap@pos{\arabic{rjlpshap@ctr2}}%
      \testarray{#1}(\rjlpshap@pos)%
      \setlength{\rjlpshap@tempwidth}{\temp@macro}%
      \addtolength{\rjlpshap@templinewidth}{-1\rjlpshap@tempwidth}%
      \parshapelenout{\rjlpshap@templinewidth}%
    }%
  \end{parshapecollect}%
}
```

`\Parshapearray` First parameter is as \parshapea{r,rra}y, but each pair is left and right margin rather than left margin and text width, while the second tracks the page width.

i.e.

The first parameter (an array) is the shape of the paragraph relative to the page.

- first element is the number of pairs of lengths in the array
- subsequent arguments are pairs of lengths, left margin width then right margin width.

The second parameter is the overall text width. This is given in the following format:

- first element is the number lengths
- subsequent elements are one width per parameter

```
\newcommand{\Parshapearray}[2]{%
  \testarray{#1}(1)\edef\rjlpshap@maxA{\temp@macro}%
  \testarray{#2}(1)\edef\rjlpshap@maxB{\temp@macro}%
  \ifthenelse{\rjlpshap@maxA > \rjlpshap@maxB} {\edef\rjlpshap@max{\rjlpshap@maxA}}{\edef\rj
  \begin{parshapecollect}%
    \forloop{rjlpshap@ctr}{1}{\not \value{rjlpshap@ctr} > \rjlpshap@max}{%
      \ifthenelse{\value{rjlpshap@ctr} > \rjlpshap@maxA}{%
        \setcounter{rjlpshap@ctr2}{\rjlpshap@maxA}%
        \addtocounter{rjlpshap@ctr2}{\rjlpshap@maxA}%
      }{%
        \setcounter{rjlpshap@ctr2}{\value{rjlpshap@ctr}}%
        \addtocounter{rjlpshap@ctr2}{\value{rjlpshap@ctr}}%
      }%
```

left margin from #1:

```
      \parshapearrlenout{#1}{rjlpshap@ctr2}%
```

get the line width

```
      \ifthenelse{\value{rjlpshap@ctr} > \rjlpshap@maxB}{%
```

```
    \edef\rjlpshap@pos{\rjlpshap@maxB}%
    \setcounter{rjlpshap@ctr3}{\rjlpshap@maxB}%
    \typeout{A pos=\rjlpshap@pos\space of \rjlpshap@maxB}%
    \testarray{#2}(\arabic{rjlpshap@ctr3})%
    \typeout{temp=\temp@macro}%
    \setlength{\rjlpshap@templinewidth}{\temp@macro}%
   }{%
    \stepcounter{rjlpshap@ctr}%
    \edef\rjlpshap@pos{\arabic{rjlpshap@ctr}}%
    \typeout{B pos=\rjlpshap@pos}%
    \addtocounter{rjlpshap@ctr}{-1}%
    \testarray{#2}(\rjlpshap@pos)%
    \setlength{\rjlpshap@templinewidth}{\temp@macro}%
   }%
   \typeout{width=\the\rjlpshap@templinewidth}%
```

subtract the left margin from the line width

```
   \addtolength{\rjlpshap@templinewidth}{-1\rjlpshap@tempwidth}%
```

subtract the right margin from the line width

```
   \stepcounter{rjlpshap@ctr2}%
   \edef\rjlpshap@pos{\arabic{rjlpshap@ctr2}}%
   \testarray{#1}(\rjlpshap@pos)%
   \setlength{\rjlpshap@tempwidth}{\temp@macro}%
   \addtolength{\rjlpshap@templinewidth}{-1\rjlpshap@tempwidth}%
   \parshapelenout{\rjlpshap@templinewidth}%
  }%
 \end{parshapecollect}%
}
```

7.3 Internal Definitions

rjlpshap@ctr: outermost loop counter:

```
\newcounter{rjlpshap@ctr}
```

rjlpshap@ctr2: used for index calculations internally:

```
\newcounter{rjlpshap@ctr2}
```

rjlpshap@ctr3: used for index calculations internally:

```
\newcounter{rjlpshap@ctr3}
```

rjlpshap@tempwidth: used as a register to read widths from arrays, and for calculations

```
\newlength{\rjlpshap@tempwidth}
```

rjlpshap@templinewidth: used as a register to calculate the line width when subtracting margins:

```
\newlength{\rjlpshap@templinewidth}
```

boolean used for breaking out of loops

```
\newboolean{rjlpshap@loop}
```

boolean used to track if the number of lengths output is odd or even

```
\newboolean{rjlpshap@isodd}
```

Filehandles needed for reading and writing temporary files

```
\newwrite\rjlpshap@writer % holds \parshape lengths
\newwrite\rjlpshap@finalwriter  % holds entire \parshape command
\newread\rjlpshap@reader % used to read writer to transfer to finalwriter
```

rjlpshap@linecount: counts the number of lines read in the environment

```
\newcounter{rjlpshap@linecount}
```

Finally, undo the effect of \makeatletter, and make @ be a special character again.

```
\makeatother
```