

# Making cutouts in paragraphs\*

Peter Wilson<sup>†</sup> and Alan Hoenig      Maintained by David Carlisle<sup>‡</sup>

2021/10/13

## Abstract

The `cutwin` package helps in making a cutout window in the middle of a paragraph.

## Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>The <code>cutwin</code> package</b>	<b>1</b>
2.1	General . . . . .	2
2.2	Rectangular cutouts . . . . .	2
2.3	Shaped cutouts . . . . .	3
<b>3</b>	<b>The package code</b>	<b>5</b>
3.1	Preliminaries . . . . .	5
3.2	General . . . . .	5
3.3	Rectangular cutouts . . . . .	6
3.4	Shaped cutouts . . . . .	11

## 1 Introduction

This manual is typeset according to the conventions of the `LATEX` `DOCSTRIP` utility which enables the automatic extraction of the `LATEX` macro source files [GM04].

Section 2 describes the usage of the `cutwin` package and commented source code is in Section 3.

## 2 The `cutwin` package

The code provided by the `cutwin` package is meant to help in creating windows, or cutouts, in a text-only paragraph. It is based on code originally published by Alan Hoenig [Hoe87].

---

\*This file (`cutwin.dtx`) has version number v0.2, last revised 2021/10/13.

<sup>†</sup>Herries Press

<sup>‡</sup><https://github.com/latex-package-repositories/cutwin/issues>

## 2.1 General

`\opencutleft`    Declarations specifying where a window is to be placed. The choices are:  
`\opencutright`    `\opencutleft` open into the left margin, `\opencutright` open into the right margin,  
`\opencutcenter`    and `\opencutcenter`, which is the default, open in the ‘center’ of the text,  
                          i.e. have text on both sides of the window.

`\cutfuzz`        This is provided as a convenience to reduce the number of overfull and underfull warnings. Its default definition is:

```
\newcommand{\cutfuzz}{%
  \vbadness=50000
  \hbadness=50000
  \sloppy}
```

and it is only applied to the paragraph being windowed.

## 2.2 Rectangular cutouts

A rectangular space can be placed in a paragraph with the text flowing across the gap. The space may break open into the top or side of the paragraph or, with some care, into 

<b>Text</b>	the bottom (the number of lines specified for the
<b>in</b>	cutout should not exceed the
<b>Window</b>	amount of text available for those lines).

 Some text or a logo or graphic may be placed within the window, or it may be left empty. In this instance I have put three short bold text lines in the window opening. The window should not be too wide as it can be difficult to track the exterior text lines across the gap.

`cutout`        The `cutout` environment, the body of which must be a single paragraph, enables a rectangular window to be cut out of the paragraph with the text flowing across the cutout. Use as:

```
\begin{cutout}{\langle numtop \rangle}{\langle leftwidth \rangle}{\langle rightwidth \rangle}{\langle numcut \rangle}
```

where  $\langle numtop \rangle$  is the number of full lines above the window and  $\langle numcut \rangle$  is the number of lines to be cut (giving the height of the window). The meaning of the lengths  $\langle leftwidth \rangle$  and  $\langle rightwidth \rangle$  depend on the location of the cutout:

- for a centered cutout  $\langle leftwidth \rangle$  and  $\langle rightwidth \rangle$  are the lengths of the text lines at the left and right sides of the window;
- for an open left cutout  $\langle leftwidth \rangle$  is ignored and  $\langle rightwidth \rangle$  is the length of the lines to the right of the cutout; and
- for an open right cutout  $\langle rightwidth \rangle$  is ignored and  $\langle leftwidth \rangle$  is the length of the lines at the left of the cutout.

`\pageinwindow`    The macro `\pageinwindow` puts a zero-sized `picture` positioned at the left  
`\windowpagestuff` of the window aligned with the first line of the window (i.e., at the top left of the cutout). The `picture` consists of a `minipage` sized to fit the window. The contents of the minipage is `\windowpagestuff`. These two macros may be used to put a graphic or text into the windowed area.

The default definition of `\windowpagestuff` is:  
`\newcommand*\windowpagestuff{}\{}`  
 and you can change it as you wish. For instance, I used the following to put some text centrally within the above cutout.

```
\renewcommand*\windowpagestuff{%
  \centering\bfseries
  Text \ in \ Window \par}
```

You may well need to experiment to get everything adjusted to your satisfaction.

## 2.3 Shaped cutouts

A *shaped cutout* is one where the shape of the window is specified by the user who has to supply the length of the text lines bordering the cutout. Normally there is text on either side of  
 the window but \$ it could be open at either the left  
 or right side of the para- graph. It is possible to put a logo or some  
 text in the window. In this paragraph with a shaped cutout I have used a large \$  
 sign as a simple logo.

shapedcutout

The `shapedcutout` environment, the body of which must be a single paragraph, enables an arbitrary shaped window to be cut out of the paragraph with the text flowing across the cutout. Use as:

`\begin{shapedcutout}{\langle numtop \rangle}{\langle numcut \rangle}{\langle shap espec \rangle}` where  $\langle numtop \rangle$  is the number of full lines above the window,  $\langle numcut \rangle$  is the number of lines to be cut (giving the height of the window) and  $\langle shap espec \rangle$  is the specification of the length of the lines bordering the cutout.

More precisely  $\langle shap espec \rangle$  is a comma-separated list of the lengths of the text lines bordering the window.

- For a centered cutout one pair of entries are required for each cut line denoting the length of the left and right part of the cut line. There must be exactly  $\langle numcut \rangle$  pairs.
- For example you might do something along the lines of:

```
\newcommand*\mycut{%
  0.1\textwidth, 0.3\textwidth,
  0.2\textwidth, 0.4\textwidth,
  0.3\textwidth, 0.5\textwidth}
\begin{shapedcutout}{2}{3}{\mycut}
...
```

which is what I used to create the shaped cutout above.

- For an open cutout each entry is the text length of a line. There must be exactly  $\langle numcut \rangle$  entries. For instance, given the above definition of `\mycut`

then a call out for an open window would be like:

```
\begin{shapedcutout}{2}{6}{\mycut}
```

`\picinwindow` In a shaped cutout the macro `\picinwindow` is placed at the center of the gap in the first line of the cutout. The default `\picinwindow` is a zero-sized picture whose contents is `\putstuffinpic`.

`\picinwindow` and `\putstuffinpic` are initially defined as

```
\newcommand*\picinwindow{%
  \begin{picture}(0,0)
    \putstuffinpic
  \end{picture}}
\newcommand*\putstuffinpic{}
```

You can change `\putstuffinpic` to place what you want in the picture. For example, to put the large \$ symbol in the shaped cutout paragraph above I used:

```
\renewcommand*\putstuffinpic{%
  \put(0,-8){\makebox(0,0){\Huge\bfseries \$}}}
```

You have to adjust the placement to suit your purposes and the shape of the cutout.

## 3 The package code

To try and avoid name clashes, all the internal commands include the string `c@tw`.

### 3.1 Preliminaries

Announce the name and version of the package, which requires L<sup>A</sup>T<sub>E</sub>X 2<sub>ε</sub>.

```
1 \*pkg>
2 \NeedsTeXFormat{LaTeX2e}
3 \ProvidesPackage{cutwin}[2021/10/13 v0.2 cutout windows]
```

### 3.2 General

```
\c@twwinlines We need lots of variables. First some counts.
\c@twtoplines 4 \newcount\c@twwinlines % window lines
\c@twcnt      5 \newcount\c@twtoplines % top lines
              6 \newcount\c@twcnt      % a count

\c@twlftside  And some lengths.
\c@twrtside  7 \newdimen\c@twlftside % left width
\c@twtopht   8 \newdimen\c@twrtside % right width
\c@twvsilg   9 \newdimen\c@twtopht  % height of top text
            10 \newdimen\c@twvsilg   % Vertical Shift or InterLine Glue

\c@twtoka    And some tokens.
\c@twtokb    11 \newtoks\c@twtoka    % build of parshape spec
            12 \newtoks\c@twtokb    % build of parshape spec

\c@twrawtext  And some boxes.
\c@twholdwin 13 \newbox\c@twrawtext   % text as input
\c@twwindow  14 \newbox\c@twholdwin % text for window sides
\c@twfinaltext 15 \newbox\c@twwindow % composed window
\c@twslicea   16 \newbox\c@twfinaltext % final assembled cutout paragraph
\c@twsliceb   17 \newbox\c@twslicea % slice A of window text
            18 \newbox\c@twsliceb % slice B of window text

\c@twdima    And some lengths.
\c@twdimb    19 \newdimen\c@twdima % formally \@tempdimb
            20 \newdimen\c@twdimb % formally \@tempdimc

\opencutleft User commands for positioning a cutout; left, right, or center. The default is
\opencutright \opencutcenter. \c@twl@c is the internal representation.
\opencutcenter 21 \newcommand*\opencutleft{\def\c@twl@c{-1}}
\c@twl@c       22 \newcommand*\opencutright{\def\c@twl@c{1}}
              23 \newcommand*\opencutcenter{\def\c@twl@c{0}}

24 \opencutcenter
```

`\cutfuzz` An attempt to stop TeX moaning about over/under full h/v boxes.

```
25 \newcommand{\cutfuzz}{\vbadness=50000
26   \hbadness=50000
27   %   \hfuzz=1pt
28   \sloppy}
```

`\c@twcalcilg` Calculate the interline glue.

```
29 \newcommand*{\c@twcalcilg}{%
30   \c@twvsihg=\baselineskip
31   \setbox0=\hbox{ }%
32   \advance\c@twvsihg-\ht0 \advance\c@twvsihg-\dp0}
```

### 3.3 Rectangular cutouts

`\pageinwindow` User modifiable macros for putting (`\pageinwindow`), via a zero-sized picture, stuff (`\windowpagestuff`) in a cutout window.

```
33 \newcommand*{\pageinwindow}{%
34   \c@twdimb=\c@twinlines\baselineskip % cutout height
35   \c@twdima=\hsize
36   \ifnum\c@twl@c=\m@ne % openleft
37     \advance\c@twdima -\c@twrtside
38   \else
39     \ifnum\c@twl@c=\@ne % openright
40       \advance\c@twdima -\c@twlftside
41     \else% center
42       \advance\c@twdima - \c@twlftside
43       \advance\c@twdima - \c@twrtside
44     \fi
45   \fi
46   \begin{picture}(0,0)%
47     \put(0,0){%
48       \raisebox{4pt}{%
49 %\fbox{%
50         \begin{minipage}[t][\c@twdimb][c]{\c@twdima}%
51           \windowpagestuff
52         \end{minipage}%
53 %}% end fbox
54       }% end raisebox
55     }% end put
56   \end{picture}}
57 \newcommand*{\windowpagestuff}{}
```

`cutout` The environment for cutting a rectangular window from a paragraph.

```
\begin{cutout}{\langle numtop \rangle}{\langle leftwidth \rangle}{\langle rightwidth \rangle}{\langle numcut \rangle}
```

where  $\langle numtop \rangle$  is the number of full lines above the window,  $\langle leftwidth \rangle$  and  $\langle rightwidth \rangle$  are the widths of the text at the sides of the window, and  $\langle numcut \rangle$  is the number of lines to be cut (giving the height of the window).

The basic method is to split the paragraph into three parts (a) the top lines above the window, (b) the window lines and (c) the rest (which will be below the

window). `\parshape` is used to do the splitting. The top lines are left at their natural length, each line crossing the window is treated as a pair of short lines, and the rest are left at their natural length.

The top lines are put into one box, the windowed ones into another and then there are the remainder. When being boxed, the window lines are combined pairwise to make single lines with space in the middle. Finally, the boxes are output.

```
58 \newenvironment{cutout}[4]{%
59   \cutfuzz
60   \c@twtotoplines=#1\relax
61   \c@twwinlines=#4\relax
62   \c@twlftside=#2\relax
63   \c@twrtside=#3\relax
64   \c@twtoka={}%

```

Generate the `\parshape` specification.

```
65   \c@twmakeparspec

```

Reset the arguments and calculate a vertical shift.

```
66   \c@twtotoplines=#1\relax
67   \c@twwinlines=#4\relax
68   \c@twcalcshift \vskip-\c@twvsilg

```

Open the `\c@twwrawtext` box, call the `\parshape` and start collecting the text to be windowed.

```
69   \setbox\c@twwrawtext=\vbox\bgroup
70   \parshape=\c@twcnt \the\c@twtoka}%

```

Now the code for the actions at `\end{cutout}`, which starts by ending the `\c@twwrawtext` box, resetting `\parshape` and calculating the interline glue.

```
71   {\egroup% end \box\c@twwrawtext
72   \parshape=0 % reset parshape;
73   \c@twcalcilg % find ILG using current font

```

If there are lines above the window, split them off from `\c@twwrawtext` into `\c@twfinaltext`.

```
74   \ifnum\c@twtotoplines>\z@
75     \setbox\c@twfinaltext=\vsplit\c@twwrawtext to\c@twtotoplines\baselineskip
76   \fi

```

Calculate the ‘height’ of the lines that make up the window. If the window is in the center then this is twice the expected height (at this point each final window line is stored as a pair of lines), otherwise it is the expected height based on `\numcut`.

```
77   \c@twtotopht=\c@twwinlines\baselineskip
78   \ifnum\c@twl@cc=\z@ % center
79     \c@twtotopht=2\c@twtotopht
80   \fi

```

Split the window lines from what is left in the `\c@twwrawtext` box into box `\c@twwholdwin` which will then contain the narrowed text for the window side(s).

```
81   \setbox\c@twwholdwin=\vsplit\c@twwrawtext to\c@twtotopht

```

Now ‘compose’ the window side(s) text (`\c@twholdwin`) into the final set of windowed lines (`\c@twwindow`). The process depends on whether the cutout is at the left, right, or center.

```
82 \ifnum\c@twl@c=\z@% center
83 \c@twcompctr{\c@twholdwin}{\c@twwindow}%
84 \else% left or right
85 \c@twcomplftrt{\c@twholdwin}{\c@twwindow}%
86 \fi
```

Assemble the various boxes into the final box (`\c@twfinaltext`) to be output.

```
87 \setbox\c@twfinaltext=
88 \vbox{\ifnum\c@twtoplines>\z@\unvbox\c@twfinaltext\vskip\c@twvsilg\fi
89 \unvbox\c@twwindow%
90 \vskip-\c@twvsilg\unvbox\c@twrawtext}%

```

We’re done, hand off the paragraph.

```
91 \box\c@twfinaltext}
```

`\c@twcompctr` `\c@twcompctr{<linepairbox>}{<composedbox>}` composes a center window box `\c@twfirst` `<linepairbox>` consisting of pairs of short lines into a box `<composedbox>` where the pairs have been assembled into single lines.

`\c@twfirst` is used as a flag for indicating the first line of a cutout.

```
92 \newcommand*{\c@twcompctr}[2]{%
93 \def\c@twfirst{1}%
94 \loop\advance\c@twwinlines\m@ne
```

Get a pair of lines and remove skips.

```
95 \setbox\c@twslicea=\vsplit#1 to\baselineskip
96 \setbox\c@twsliceb=\vsplit#1 to\baselineskip
97 \c@twprune{\c@twslicea}{\c@twlftside}%
98 \c@twprune{\c@twsliceb}{\c@twrtside}%
99 \ifnum\c@twfirst=\@ne
```

The first time put the texts into a box at the left and right with `\pageinwindow` at the end of the left text.

```
100 \setbox#2=\vbox{\unvbox#2\hbox
101 to\hsize{\box\c@twslicea\pageinwindow\hfil\box\c@twsliceb}}%
102 \else
```

For further lines just put the texts at the left and right.

```
103 \setbox#2=\vbox{\unvbox#2\hbox
104 to\hsize{\box\c@twslicea\hfil\box\c@twsliceb}}%
105 \fi
106 \def\c@twfirst{2}%
107 \ifnum\c@twwinlines>\z@\repeat}
```

`\c@twcomplftrt` Compose an open (left or right) sided rectangular window.

```
108 \newcommand*{\c@twcomplftrt}[2]{%
109 \def\c@twfirst{1}%
110 \loop\advance\c@twwinlines\m@ne
```



For an open window we simply deal with one line at a time, not pairs. In other respects the code is generally similar to that for `\c@twcompctr`.

```

111 \setbox\c@twslicea=\vsplit#1 to\baselineskip
112 \ifnum\c@twl@c=\m@ne% open left, text at right
113 \c@twprune{\c@twslicea}{\c@twrtside}%
114 \ifnum\c@twfirst=\@ne
115 \setbox#2=\vbox{\unvbox#2\hbox
116 to\hsize{\pageinwindow\hfil\box\c@twslicea}}%
117 \else
118 \setbox#2=\vbox{\unvbox#2\hbox
119 to\hsize{\mbox{}}\hfil\box\c@twslicea}}%
120 \fi
121 \def\c@twfirst{2}%
122 \else
123 \ifnum\c@twl@c=\@ne% open right, text at left
124 \c@twprune{\c@twslicea}{\c@twlftside}%
125 \ifnum\c@twfirst=\@ne\relax
126 \setbox#2=\vbox{\unvbox#2\hbox
127 to\hsize{\box\c@twslicea\pageinwindow}}%
128 \else
129 \setbox#2=\vbox{\unvbox#2\hbox
130 to\hsize{\box\c@twslicea}}%
131 \fi
132 \def\c@twfirst{2}%
133 \fi
134 \fi
135 \ifnum\c@twinlines>\z@repeat}

```

`\c@twprune` `\c@twprune{<vbox>}{<width>}` chops off the `\lastskip`. It takes a `<vbox>` containing a single `\hbox`, `\unvboxes` it, cancels the `\lastskip` which can be put at the right of a short `\parshape` line, then puts it in a box width `<width>`.

```

136 \newcommand*\c@twprune}[2]{%
137 \unvbox#1 \setbox#1=\lastbox % \box#1 is now an \hbox
138 \setbox#1=\hbox to#2{\strut\unhbox#1\unskip}}

```

`\c@twmakeparspec` Calculate the required `\parshape` spec for a paragraph with a rectangular cutout window.

```

139 \newcommand*\c@twmakeparspec){%
    \c@twcnt is the total number of lines for the \parshape, i.e., the number of the
    top lines plus (twice) the number of window line plus one for the remaining lines.
140 \c@twcnt=\c@twinlines
141 \ifnum\c@twl@c=\z@
142 \multiply \c@twcnt by \tw@
143 \fi
144 \advance\c@twcnt by \c@twtoplines \advance\c@twcnt by \@ne
    If there are top lines generate a 0pt \hsize for each
145 \ifnum\c@twtoplines>\z@
146 \loop\c@twtoka=\expandafter\the\c@twtoka 0pt \hsize}%

```

```

147     \advance\c@twtoplines -1\relax
148     \ifnum\c@twtoplines>\z@\repeat
149     \fi

Now do the cutout portion of the spec.
150     \ifnum\c@twl@c=\m@ne % openleft

For open left calculate the width of the open cutout as \c@twlftside.
151     \c@twlftside=\hsize
152     \advance\c@twlftside -\c@twrtside
153     \fi

Loop over the windowed lines.
154     \loop\c@twtoka=%
155     \ifnum\c@twl@c=\m@ne % openleft

For open left generate a \c@twlftside \c@twrtside for each.
156         \expandafter{\the\c@twtoka \c@twlftside \c@twrtside}%
157     \else
158         \ifnum\c@twl@c=\@ne % openright

For open right generate a \Opt c@twlftside for each
159         \expandafter{\the\c@twtoka Opt \c@twlftside}%
160     \else %center

For centered generate Opt \c@twlftside Opt \c@twrtside for each pair.
161         \expandafter{\the\c@twtoka Opt \c@twlftside Opt \c@twrtside}%
162     \fi
163     \fi
164     \advance\c@twwinlines \m@ne
165     \ifnum\c@twwinlines>\z@\repeat

That finishes the cutout portion. For the remaining lines in the paragraph just
generate a single Opt \hsize.
166     \c@twtoka=\expandafter{\the\c@twtoka Opt \hsize}}

\c@twcalcshift Calculate the estimated vertical shift needed for the window. I determined the
values experimentally based on a 10pt font. In may be different for different fonts,
but I hope not.
167 \newcommand*{\c@twcalcshift}{% vertical shift
168     \c@twvsihg=\c@twwinlines\baselineskip
169     \ifnum\c@twtoplines<\@ne
170         \advance\c@twvsihg -0.25\baselineskip
171     \fi
172     \c@twvsihg=0.5\c@twvsihg
173     \ifnum\c@twl@c=\z@\else
174         \c@twvsihg=0.5\c@twvsihg
175     \fi}

```

### 3.4 Shaped cutouts

`\picinwindow` A zero-sized picture, with contents `\putstuffinpic`, which is placed in the center of the first gap in a shaped cutout.

```

176 \newcommand*\picinwindow}{%
177   \begin{picture}(0,0)
178   \putstuffinpic
179   \end{picture}}

```

`\putstuffinpic` Default `\putstuffinpic` is empty.

```

180 \newcommand*\putstuffinpic{}

```

`shapedcutout` A shaped cutout where the user defines the shape.

`\c@twb` `\begin{shapedcutout}{\langle numtop \rangle}{\langle numcut \rangle}{\langle shap espec \rangle}`  
 where `\langle numtop \rangle` is the number of full lines above the window, `\langle numcut \rangle` is the number of lines to be cut (giving the height of the window) and `\langle shap espec \rangle` is the user's specification of the shape of the surroundings of the cutout. This is in the form of a comma-separated list of either the pairs of widths of the left and right texts of a centered cutout or the widths of the left or right texts of an open cutout. `\c@twb` holds arg 3 (`\langle shap espec \rangle`), the user's parspec.

The code is very similar to that for the `cutout` environment.

```

181 \newenvironment{shapedcutout}[3]{%
182   \cutfuzz
183   \c@twtoplines=#1\relax
184   \c@twwinlines=#2\relax
185   \edef\c@twb{#3}%          user's parspec

```

Generate the top lines portion of the parspec followed by the cutout portion.

```

186   \c@twmaketopoddspec
187   \c@twbuildoddspec{#3}

```

Continue like the `cutout` code.

```

188   \c@twtoplines=#1\relax
189   \c@twwinlines=#2\relax
190   \c@twcalcshift \vskip-\c@twvsilg
191   \setbox\c@twrawtext=\vbox\bgroup

```

`\c@twcnt` is the total number of parshape lines; `\c@twtoka` is the spec for the top lines; `\c@twtokb` is the spec for the cutout lines; and `Opt \hsize` is the spec for the remainder of the paragraph.

```

192   \parshape=\c@twcnt \the\c@twtoka \the\c@twtokb Opt \hsize}%

```

The code for the end of the environment, where most of the work is done. It is similar to the code for the end of the `cutout` environment.

```

193 {\egroup
194   \parshape=0
195   \c@twcalcilg
196   \ifnum\c@twtoplines>\z@
197     \setbox\c@twfinaltext=\vsplit\c@twrawtext to\c@twtoplines\baselineskip
198   \fi

```

```

199 \c@twtopht=\c@twwinlines\baselineskip
200 \ifnum\c@twl@c=\z@ % center
201   \c@twtopht=2\c@twtopht
202 \fi
203 \setbox\c@twholdwin=\vsplit\c@twrawtext to\c@twtopht
204 \ifnum\c@twl@c=\z@%   center
205   \c@twcompoddctr{\c@twholdwin}{\c@twwindow}%
206 \else% open left or right
207   \c@twcompoddlftrt{\c@twholdwin}{\c@twwindow}%
208 \fi
209 \setbox\c@twfinaltext=
210   \vbox{\ifnum\c@twtoplines>\z@\unvbox\c@twfinaltext\vskip\c@twvsilg\fi
211   \unvbox\c@twwindow%
212   \vskip-\c@twvsilg\unvbox\c@twrawtext}%
213 \box\c@twfinaltext}

```

`\c@twmaketopoddspec` Make up the easy part of the odd `\parshape` specification; total number `\c@twcnt` and the topline spec (`\c@twtoka`).

```

214 \newcommand*\c@twmaketopoddspec}{%
215   \c@twcnt=\c@twwinlines
216   \ifnum\c@twl@c=\z@
217     \multiply \c@twcnt by \tw@
218   \fi
219   \advance\c@twcnt by \c@twtoplines \advance\c@twcnt by \@ne
220 %% \c@twcnt is total of topline + 2(window lines) + 1
221   \c@twtoka={}%
222   \ifnum\c@twtoplines>\z@
223     \loop\c@twtoka=\expandafter\the\c@twtoka Opt \hsize}%
224     \advance\c@twtoplines -1\relax
225     \ifnum\c@twtoplines>\z@\repeat
226   \fi}

```

`\c@twaddtospec` Adds a ‘zero-indented line’ to a `\parshape` spec being assembled in `\c@twtokb`.

```

227 \newcommand*\c@twaddtospec}[1]{%
228   \c@twtokb=\c@twxpf{\the\c@twtokb Opt #1 }}

```

`\c@twbuildoddspec` `\c@twbuildoddspec{<commalist>}` builds up the `\parshape` spec for the odd cutout lines from the comma-separated list of lengths in `<commalist>`.

`\c@twxpf` is a shorthand for `\expandafter` to try and make the code shorter to read.

`\c@twlspec` `\c@twlspec` is used as a temporary variable when iterating over a comma-separated list.

```

229 \let\c@twxpf\expandafter
230 \newcommand*\c@twbuildoddspec}[1]{%
231   \c@twtokb={}%
232   \@for\c@twlspec:=#1\do{%
233     \c@twxpf\c@twxpf\c@twxpf\c@twaddtospec\c@twxpf{\c@twlspec}}

```

`\c@twcompoddctr` Compose the lines of an odd shaped center cutout.  
`\c@twrounds` We go through the user's shape list an item at a time but we need to collect pairs of items. The `\c@twrounds` variable is for managing the pairing. `\c@twfirst` is a flag for positioning the `\picinwindow` in the first line of the cutout.

```

234 \newcommand*\c@twcompoddctr}[2]{%
235   \def\c@twrounds{1}%
236   \def\c@twfirst{1}%
237   \@for\c@twlspec:=\c@twb\do{%
238     \ifnum\c@twrounds=1
239       \setbox\c@twslicea=\vsplit#1 to\baselineskip % first of pair
240       \c@twprune{\c@twslicea}{\c@twlspec}%
241       \def\c@twrounds{2}%
242     \else
243       \setbox\c@twsliceb=\vsplit#1 to\baselineskip % second of pair
244       \c@twprune{\c@twsliceb}{\c@twlspec}%
245       \ifnum\c@twfirst=1
246         \setbox#2=\vbox{\unvbox#2\hbox
247           to\hsize{\box\c@twslicea\hfil\picinwindow\hfil\box\c@twsliceb}}%
248         \def\c@twfirst{2}%
249       \else
250         \setbox#2=\vbox{\unvbox#2\hbox
251           to\hsize{\box\c@twslicea\hfil\box\c@twsliceb}}%
252       \fi
253       \def\c@twrounds{1}%
254     \fi}}

```

`\c@twcompoddlftrt` Compose the open (left or right) lines of an odd shaped cutout.

```

255 \newcommand*\c@twcompoddlftrt}[2]{%
256   \def\c@twfirst{1}%
257   \@for\c@twlspec:=\c@twb\do{%
258     \setbox\c@twslicea=\vsplit#1 to\baselineskip % get a line
259     \c@twprune{\c@twslicea}{\c@twlspec}%
260     \ifnum\c@twl@c=\m@ne% open left, text at right
261       \ifnum\c@twfirst=1
262         \setbox#2=\vbox{\unvbox#2\hbox
263           to\hsize{\mbox{}}\hfil\picinwindow\hfil\box\c@twslicea}}%
264         \def\c@twfirst{2}%
265       \else
266         \setbox#2=\vbox{\unvbox#2\hbox
267           to\hsize{\mbox{}}\hfil\box\c@twslicea}}%
268       \fi
269     \else
270       \ifnum\c@twl@c=\@ne% open right, text at left
271         \ifnum\c@twfirst=1
272           \setbox#2=\vbox{\unvbox#2\hbox
273             to\hsize{\box\c@twslicea\hfil\picinwindow\hfil}}%
274           \def\c@twfirst{2}%
275         \else
276           \setbox#2=\vbox{\unvbox#2\hbox

```

```

277         to\hsize{\box\c@twslicea\hfil}}%
278         \fi
279     \fi
280 \fi}}

```

The end of this package.

```
281 \</pkg>
```

## References

- [GM04] Frank Mittelbach and Michel Goossens. *The LaTeX Companion*. Second edition. Addison-Wesley Publishing Company, 2004.
- [Hoe87] Alan Hoenig. TeX does windows — The conclusion, *TUGboat*, vol 8, no 2, pp 211–215, 1987.

## 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.

<b>Symbols</b>	<code>\c@twcompoddctr</code> 205, <u>234</u>	240, 244, 257, 259
<code>\@for</code> . . . . . 232, 237, 257	<code>\c@twcompoddlftrt</code> . . . . . 207, <u>255</u>	<code>\c@twmakeparspec</code> 65, <u>139</u>
<code>\@tempdimb</code> . . . . . 19	<code>\c@twdima</code> . . . . . <u>19</u>	<code>\c@twmaketopoddspec</code> . . . . . 186, <u>214</u>
<code>\@tempdimc</code> . . . . . 20	<code>\c@twdimb</code> . . . <u>19</u> , 34, 50	<code>\c@twprune</code> . . . . .
<b>B</b>	<code>\c@twfinaltext</code> . <u>13</u> , 75, 87, 88, 91, 197, 209, 210, 213	97, 98, 113, 124, <u>136</u> , 240, 244, 259
<code>\baselineskip</code> . . . . . 30, 34, 75, 77, 95, 96, 111, 168, 170, 197, 199, 239, 243, 258	<code>\c@twfirst</code> . . . . <u>92</u> , 109, 114, 121, 125, 132, 236, 245, 248, 256, 261, 264, 271, 274	<code>\c@twrawtext</code> . <u>13</u> , 69, 71, 75, 81, 90, 191, 197, 203, 212
<b>C</b>	<code>\c@twaddtospec</code> <u>227</u> , 233	<code>\c@twrounds</code> . . . . . <u>234</u>
<code>\c@twb</code> . . . . <u>181</u> , 237, 257	<code>\c@twholdwin</code> . <u>13</u> , 81, 83, 85, 203, 205, 207	<code>\c@twrtside</code> . . . . . <u>7</u> , 37, 43, 63, 98, 113, 152, 156, 161
<code>\c@twbuildoddspec</code> . . . . . 187, <u>229</u>	<code>\c@twl@c</code> . <u>21</u> , 36, 39, 78, 82, 112, 123, 141, 150, 155, 158, 173, 200, 204, 216, 260, 270	<code>\c@twslicea</code> <u>13</u> , 95, 97, 101, 104, 111, 113, 116, 119, 124, 127, 130, 239, 240, 247, 251, 258, 259, 263, 267, 273, 277
<code>\c@twcalcilg</code> <u>29</u> , 73, 195	<code>\c@twlftside</code> <u>7</u> , 40, 42, 62, 97, 124, 151, 152, 156, 159, 161	<code>\c@twsliceb</code> . . . . <u>13</u> , 96, 98, 101, 104, 243, 244, 247, 251
<code>\c@twcalcshift</code> . . . . . 68, <u>167</u> , 190	<code>\c@twlspec</code> . . . <u>229</u> , 232, 233, 237,	<code>\c@twtoka</code> . . . . . <u>11</u> ,
<code>\c@twcnt</code> . <u>4</u> , 70, 140, 142, 144, 192, 215, 217, 219, 220		
<code>\c@twcompctr</code> . . . . 83, <u>92</u>		
<code>\c@twcomplftrt</code> . 85, <u>108</u>		

- 64, 70, 146, 154,  
156, 159, 161,  
166, 192, 221, 223
- `\c@twtokb` .....  
    . 11, 192, 228, 231
- `\c@twtopht` .. 7, 77,  
    79, 81, 199, 201, 203
- `\c@twtoplines` .....  
    .... 4, 60, 66,  
    74, 75, 88, 144,  
    145, 147, 148,  
    169, 183, 188,  
    196, 197, 210,  
    219, 222, 224, 225
- `\c@twvsihg` .. 7, 30,  
    32, 68, 88, 90,  
    168, 170, 172,  
    174, 190, 210, 212
- `\c@twwindow` .. 13, 83,  
    85, 89, 205, 207, 211
- `\c@twwinlines` .....  
    .... 4, 34, 61,  
    67, 77, 94, 107,  
    110, 135, 140,  
    164, 165, 168,  
    184, 189, 199, 215
- `\c@twxpf` ..... 228, 229
- `\cutfuzz` . 2, 25, 59, 182
- cutout (environment)  
    ..... 2, 58
- D**
- `\do` ..... 232, 237, 257
- `\dp` ..... 32
- E**
- `\edef` ..... 185
- environments:  
    cutout ..... 2, 58  
    shapedcutout . 3, 181
- H**
- `\hbadness` ..... 26
- `\hfuzz` ..... 27
- `\ht` ..... 32
- L**
- `\lastbox` ..... 137
- `\loop` ..... 94,  
    110, 146, 154, 223
- M**
- `\mbox` ..... 119, 263, 267
- `\multiply` .... 142, 217
- N**
- `\newbox` ..... 13–18
- `\newcount` ..... 4–6
- `\newtoks` ..... 11, 12
- O**
- `\opencutcenter` 2, 21, 24
- `\opencutleft` ..... 2, 21
- `\opencutright` .... 2, 21
- P**
- `\pageinwindow` .....  
    2, 33, 101, 116, 127
- `\parshape` 70, 72, 192, 194
- `\picinwindow` .... 4,  
    176, 247, 263, 273
- `\ProvidesPackage` ... 3
- `\put` ..... 47
- `\putstuffinpic` 178, 180
- R**
- `\raisebox` ..... 48
- `\repeat` ..... 107,  
    135, 148, 165, 225
- S**
- shapedcutout (environ-  
    ment) .... 3, 181
- `\strut` ..... 138
- U**
- `\unhbox` ..... 138
- `\unskip` ..... 138
- `\unvbox` .....  
    88–90, 100, 103,  
    115, 118, 126,  
    129, 137, 210–  
    212, 246, 250,  
    262, 266, 272, 276
- V**
- `\vbadness` ..... 25
- `\vbox` ... 69, 88, 100,  
    103, 115, 118,  
    126, 129, 191,  
    210, 246, 250,  
    262, 266, 272, 276
- `\vsplit` ..... 75, 81,  
    95, 96, 111, 197,  
    203, 239, 243, 258
- W**
- `\windowpagestuff` . 2, 33