Making cutouts in paragraphs*

Peter Wilson† and Alan Hoenig ‡ Maintained by David Carlisle‡

2021/10/13

Abstract

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

Contents

1 Introduction 1

2 The cutwin package 1
  2.1 General ......................................................... 2
  2.2 Rectangular cutouts ......................................... 2
  2.3 Shaped cutouts .............................................. 3

3 The package code 5
  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.
†Herries Press
‡https://github.com/latex-package-repositories/cutwin/issues
2.1 General

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

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

\begin{verbatim}
\newcommand{\cutfuzz}{%
  \vbadness=50000
  \hbadness=50000
  \sloppy}
\end{verbatim}

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 Text the bottom (the number of lines specified for the in cutout should not exceed the 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.

\texttt{\textbackslash cutout} The \texttt{\textbackslash 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{verbatim}
\begin{cutout}{⟨numtop⟩}{⟨leftwidth⟩}{⟨rightwidth⟩}{⟨numcut⟩}
\end{verbatim}

where \texttt{⟨numtop⟩} is the number of full lines above the window and \texttt{⟨numcut⟩} is the number of lines to be cut (giving the height of the window). The meaning of the lengths \texttt{⟨leftwidth⟩} and \texttt{⟨rightwidth⟩} depend on the location of the cutout:

- for a centered cutout \texttt{⟨leftwidth⟩} and \texttt{⟨rightwidth⟩} are the lengths of the text lines at the left and right sides of the window;
- for an open left cutout \texttt{⟨leftwidth⟩} is ignored and \texttt{⟨rightwidth⟩} is the length of the lines to the right of the cutout; and
- for an open right cutout \texttt{⟨rightwidth⟩} is ignored and \texttt{⟨leftwidth⟩} is the length of the lines at the left of the cutout.

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

The default definition of \texttt{windowpagestuff} is:
\begin{verbatim}
\newcommand*{\windowpagestuff}{}
\end{verbatim}
and you can change it as you wish. For instance, I used the following to put some text centrally within the above cutout.
\begin{verbatim}
\renewcommand*{\windowpagestuff}{%
\centering\bfseries
Text \hfill in \hfill Window \par}
\end{verbatim}
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 paragraph. 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 dollar sign as a simple logo.

The \texttt{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{verbatim}
\begin{shapedcutout}{⟨numtop⟩}{⟨numcut⟩}{⟨shapespec⟩}
\end{verbatim}
\end{verbatim}
where \texttt{⟨numtop⟩} is the number of full lines above the window, \texttt{⟨numcut⟩} is the number of lines to be cut (giving the height of the window) and \texttt{⟨shapespec⟩} is the specification of the length of the lines bordering the cutout.

More precisely \texttt{⟨shapespec⟩} 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 \texttt{⟨numcut⟩} pairs.

- For example you might do something along the lines of:
\begin{verbatim}
\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}
\end{verbatim}
\end{verbatim}
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 \texttt{⟨numcut⟩} entries. For instance, given the above definition of \texttt{\mycut
then a call out for an open window would be like:
\begin{shapedcutout}{2}{6}\mycut
\end{shapedcutout}

In a shaped cutout the macro \texttt{\textbackslash picinwindow} is placed at the center of the gap in the first line of the cutout. The default \texttt{\textbackslash picinwindow} is a zero-sized picture whose contents is \texttt{\textbackslash putstuffinpic}.
\texttt{\textbackslash picinwindow} and \texttt{\textbackslash putstuffinpic} are initially defined as

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

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

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

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 LaTeX2ε.

```
\NeedsTeXFormat{LaTeX2e}
\ProvidesPackage{cutwin}[2021/10/13 v0.2 cutout windows]
```

#### 3.2 General

We need lots of variables. First some counts.

```
\newcount\c@twwinlines % window lines
\newcount\c@twtoplines % top lines
\newcount\c@twcnt % a count
```

And some lengths.

```
\newdimen\c@twlftside % left width
\newdimen\c@twrtside % right width
\newdimen\c@twtopht % height of top text
\newdimen\c@twvsilg % Vertical Shift or InterLine Glue
```

And some tokens.

```
\newtoks\c@twtoka % build of parshape spec
\newtoks\c@twtokb % build of parshape spec
```

And some boxes.

```
\newbox\c@twrawtext % text as input
\newbox\c@twindow % text for window sides
\newbox\c@twfinaltext % final assembled cutout paragraph
\newbox\c@twslicea % slice A of window text
\newbox\c@twsliceb % slice B of window text
```

And some lengths.

```
\newdimen\c@twdima % formally \@tempdimb
\newdimen\c@twdimb % formally \@tempdimc
```

User commands for positioning a cutout; left, right, or center. The default is \opencutcenter. \c@twl@c is the internal representation.

```
\newcommand*{\opencutleft}{\def\c@twl@c{-1}}
\newcommand*{\opencutright}{\def\c@twl@c{1}}
\newcommand*{\opencutcenter}{\def\c@twl@c{0}}
```

```
\opencutcenter
```
\cutfuzz \newcommand{\cutfuzz}{\vbadness=50000 \hbadness=50000 \hfuzz=1pt \sloppy}

\c@twcalcilg \newcommand*{\c@twcalcilg}{% \c@twvsilg=\baselineskip \setbox0=\hbox{(% \advance\c@twvsilg-\ht0 \advance\c@twvsilg-\dp0)

3.3 Rectangular cutouts
\pageinwindow \newcommand*{\pageinwindow}{% \c@twdimb=\c@twwinlines\baselineskip \c@twdima=\hsize \ifnum\c@twl@c=\m@ne \advance\c@twdima -\c@twrtside \else \ifnum\c@twl@c=\@ne \advance\c@twdima -\c@twlftside \else \fi \fi \fi \fi \fi \begin{picture}(0,0) \put(0,0){% \raisebox{4pt}{% %\fbox{\begin{minipage}[t][\c@twdimb][c]{\c@twdima}\pageinwindowstuff %}% end fbox \end{picture}} \newcommand*{\windowpagestuff}{}

\begin{cutout}{⟨numtop⟩}{⟨leftwidth⟩}{⟨rightwidth⟩}{⟨numcut⟩}

where ⟨numtop⟩ is the number of full lines above the window, ⟨leftwidth⟩ and ⟨rightwidth⟩ are the widths of the text at the sides of the window, and ⟨numcut⟩ 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). \texttt{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.

\begin{verbatim}
\newenvironment{cutout}{% \cutfuzz \c@twtoplines=#1\relax \c@twwinlines=#4\relax \c@twlftside=#2\relax \c@twrtside=#3\relax \c@twtoka={}% Generate the \parshape specification. \c@twmakeparspec Reset the arguments and calculate a vertical shift. \c@twtoplines=#1\relax \c@twwinlines=#4\relax \c@twcalcshift \vskip-\c@twvsilg Open the \c@twrawtext box, call the \parshape and start collecting the text to be windowed. \setbox\c@twrawtext=\vbox\bgroup \parshape=\c@twcnt \the\c@twtoka}{% Now the code for the actions at \texttt{end\{cutout\}}, which starts by ending the \c@twrawtext box, resetting \parshape and calculating the interline glue. {\egroup% end \box\c@twrawtext \parshape=0 \% reset parshape; \c@twcalcilg \% find ILG using current font If there are lines above the window, split them off from \c@twrawtext into \c@twfinaltext. \ifnum\c@twtoplines>\z@ \setbox\c@twfinaltext=\vsplit\c@twrawtext to\c@twtoplines\baselineskip \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 \textit{numcut}. \ifnum\c@twfinaltext=\baselineskip \c@twtopht=\c@twwinlines\baselineskip 80 \fi Split the window lines from what is left in the \c@twrawtext box into box \c@twholdwin which will then contain the narrowed text for the window side(s). \setbox\c@twholdwin=\vsplit\c@twrawtext to\c@twtopht
\end{verbatim}
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.

\ifnum\c@twl@c=\z@ \c@twcompctr{\c@twholdwin}{\c@twwindow}\fi

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

\setbox\c@twfinaltext=
\vbox{\ifnum\c@twtoplines>\z@\unvbox\c@twfinaltext\vskip\c@twvsilg\fi
\unvbox\c@twwindow\vskip-\c@twvsilg\unvbox\c@twrawtext}\

We’re done, hand off the paragraph.
\box\c@twfinaltext

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

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

\newcommand*{\c@twcompctr}[2]{%\def\c@twfirst{1}%
\loop\advance\c@twwinlines\m@ne
Get a pair of lines and remove skips.
\setbox\c@twslicea=\vsplit#1 to\baselineskip
\setbox\c@twsliceb=\vsplit#1 to\baselineskip
\c@twprune{\c@twslicea}{\c@tfltside}%
\c@twprune{\c@twsliceb}{\c@trtside}%
\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.
\setbox#2=\vbox{\unvbox#2\hbox to\hsize{\box\c@twslicea\pageinwindow\hfil\box\c@twsliceb}}\%\else
For further lines just put the texts at the left and right.
\setbox#2=\vbox{\unvbox#2\hbox to\hsize{\box\c@twslicea\hfil\box\c@twsliceb}}\%
\fi
\def\c@twfirst{2}%
\ifnum\c@twwinlines>\z@\repeat

\c@twcomplftrt \c@twwindow
\c@twcomplftrt\{\langle linepairbox \rangle\}\{\langle composedbox \rangle\} composes an open (left or right) sided rectangular window.

\newcommand*{\c@twcomplftrt}[2]{%\def\c@twfirst{1}%
\loop\advance\c@twwinlines\m@ne
\setbox#2=\vbox{\unvbox#2\hbox to\hsize{\box\c@twslicea\pageinwindow\hfil\box\c@twsliceb}}\%\else
For further lines just put the texts at the left and right.
\setbox#2=\vbox{\unvbox#2\hbox to\hsize{\box\c@twslicea\hfil\box\c@twsliceb}}\%
\fi
\def\c@twfirst{2}%
\ifnum\c@twwinlines>\z@\repeat
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.

\begin{verbatim}
\setbox\c@twslicea=\vsplit#1 to\baselineskip
\ifnum\c@twl@c=\m@ne% open left, text at right
 \c@twprune{%c@twslicea}{\c@twlftside}%
 \ifnum\c@twfirst=\@ne
 \setbox#2=\vbox{\unvbox#2\hbox
to\hsize{\pageinwindow\hfil\box\c@twslicea}}%
 \else
 \setbox#2=\vbox{\unvbox#2\hbox
to\hsize{\mbox{}\hfil\box\c@twslicea}}%
 \fi
 \def\c@twfirst{2}%
 \else
 \ifnum\c@twl@c=\@ne% open right, text at left
 \c@twprune{%c@twslicea}{\c@twrtside}%
 \ifnum\c@twfirst=\@ne\relax
 \setbox#2=\vbox{\unvbox#2\hbox
to\hsize{\box\c@twslicea\pageinwindow}}%
 \else
 \setbox#2=\vbox{\unvbox#2\hbox
to\hsize{\box\c@twslicea}}%
 \fi
 \def\c@twfirst{2}%
 \fi
 \fi
\end{verbatim}

\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⟩.

\begin{verbatim}
\newcommand*{\c@twprune}{2}{%}
\unvbox#1 \setbox#1=\lastbox % \box#1 is now an \hbox
\setbox#1=\hbox to#2{\strut\unhbox#1\unskip}}%
\end{verbatim}

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

\begin{verbatim}
\c@twmakeparspec \c@twmakeparspec{\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.
\c@twcnt=\c@twinlines
\ifnum\c@twl@c=\z@\multiply \c@twcnt by \tw@
\fi
\advance\c@twcnt by \ctwtoplines \advance\c@twcnt by \@one
If there are top lines generate a Opt \hsize for each
\ifnum\c@twtoplines=\z@
\loop\c@twtop=\expandafter{\the\c@twtop Opt \hsize}%
\end{verbatim}
Now do the cutout portion of the spec.

For open left calculate the width of the open cutout as $c@twlftside$.

Loop over the windowed lines.

That finishes the cutout portion. For the remaining lines in the paragraph just generate a single Opt $c@twlftside$.

\ctwcalcshift 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.
3.4 Shaped cutouts

\begin{shapedcutout}{⟨numtop⟩}{⟨numcut⟩}{⟨shapespec⟩}

where ⟨numtop⟩ is the number of full lines above the window, ⟨numcut⟩ is the number of lines to be cut (giving the height of the window) and ⟨shapespec⟩ 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 (⟨shapespec⟩), the user’s parspec.

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

\begin{verbatim}
\newenvironment{shapedcutout}{%}
\begin{shapedcutout}{⟨numtop⟩}{⟨numcut⟩}{⟨shapespec⟩}
\c@twb holds arg 3 (⟨shapespec⟩), the user’s parspec.

\end{verbatim}
\c@twmaketopoddspec Make up the easy part of the odd parshape specification; total number \c@twcnt and the toplines spec (~\c@twtoka).}
\newcommand*\c@twmaketopoddspec{\c@twcnt=\c@twwinlines\ifnum\c@twl@c=\z@ \multiply\c@twcnt by \tw@ \fi\advance\c@twcnt by \c@twtoplines + 2(window lines) + 1%% \c@twcnt is total of toplines + 2(window lines) + 1
\c@twtoka={}\ifnum\c@twtoplines>\z@ \loop\c@twtoka=\expandafter{\the\c@twtoka 0pt \hsize}\advance\c@twtoplines -1\relax\ifnum\c@twtoplines>\z@ \repeat
\fi}
\c@twaddtospec Adds a ‘zero-indented line’ to a parshape spec being assembled in \c@twtokb.
\newcommand*\c@twaddtospec\[1\]{\c@twtokb=\c@twxpf\{\the\c@twtokb 0pt #1 }}
\c@twbuildoddspec\c@txpf \c@twbuildoddspec\langle commalist \rangle builds up the parshape spec for the odd cutout lines from the comma-separated list of lengths in \langle commalist \rangle.
\c@txpf is a shorthand for \expandafter to try and make the code shorter to read.
\c@twspec \c@twspec is used as a temporary variable when iterating over a comma-separated list.
\let\c@twxpf\expandafter
\newcommand*\c@twbuildoddspec\[1\]{\c@twtokb=\c@twxpf\langle the \c@twtokb Opt \ hsize \rangle
\expandafter\c@twxpf\c@twxpf\c@twxpf\c@twxpf\c@twxpf\c@twxpf\c@twxpf\langle \c@twspec \rangle}}
3.4 Shaped cutouts

\c@twcompoddctr  Compose the lines of an odd shaped center cutout.

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.

\newcommand*{\c@twcompoddctr}{\c@twb}[2]{%  
  \def\c@twrounds{1}\%  
  \def\c@twfirst{1}\%  
  \@for\c@twlspec:=\c@twb\do{%  
    \ifnum\c@twrounds=1  
    \setbox\c@twslicea=\vsplit#1 to\baselineskip % first of pair  
    \c@twprune{\c@twslicea}{\c@twlspec}\%  
    \def\c@twrounds{2}\%  
    \else  
    \setbox\c@twsliceb=\vsplit#1 to\baselineskip % second of pair  
    \c@twprune{\c@twsliceb}{\c@twlspec}\%  
    \ifnum\c@twfirst=1  
    \setbox#2=\vbox{\unvbox#2\hbox to\hsize{\box\c@twslicea\hfil\picinwindow\hfil\box\c@twsliceb}}\%  
    \def\c@twfirst{2}\%  
    \else  
    \setbox#2=\vbox{\unvbox#2\hbox to\hsize{\box\c@twslicea}}\%  
    \fi  
    \def\c@twrounds{1}\%  
    \fi}%}

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

\newcommand*{\c@twcompoddlftr}{\c@twb}[2]{%  
  \def\c@twfirst{1}\%  
  \@for\c@twlspec:=\c@twb\do{%  
    \setbox\c@twslicea=\vsplit#1 to\baselineskip % get a line  
    \c@twprune{\c@twslicea}{\c@twlspec}\%  
    \ifnum\c@twl@c=-1% open left, text at right  
    \ifnum\c@twfirst=1  
    \setbox#2=\vbox{\unvbox#2\hbox to\hsize{\mbox{}\hfil\picinwindow\hfil\box\c@twslicea}}\%  
    \def\c@twfirst{2}\%  
    \else  
    \setbox#2=\vbox{\unvbox#2\hbox to\hsize{\box\c@twslicea}}\%  
    \fi  
    \else  
    \ifnum\c@twl@c=1% open right, text at left  
    \ifnum\c@twfirst=1  
    \setbox#2=\vbox{\unvbox#2\hbox to\hsize{\box\c@twslicea\hfil\picinwindow\hfil}}\%  
    \def\c@twfirst{2}\%  
    \else  
    \setbox#2=\vbox{\unvbox#2\hbox to\hsize{\mbox{}}\hfil\box\c@twslicea}}\%  
    \fi  
    \else  
    \ifnum\c@twl@c=0% open neither  
    \setbox#2=\vbox{\unvbox#2\hbox to\hsize{\box\c@twslicea}}\%  
    \fi  
    \fi}
The end of this package.

References


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.

Symbols

\@for \ldots 232, 237, 257
\@tempdimb \ldots 19
\@tempdimc \ldots 20
\@twcalcilg \ldots 29, 73, 195
\@twcalcshift \ldots 68, 107, 190
\@twcount \ldots 142, 217, 219, 220
\@twcomppctr \ldots 83, 92
\@twcomplfrt \ldots 85, 108
\@twcomppdfctr \ldots 205, 234
\@twcomppdffrt \ldots 207, 255
\@twmakeparspec \ldots 65, 139
\@twmaketoddspec \ldots 186, 214
\@twprune \ldots 97, 98, 113, 124
\@twfinaltext \ldots 13, 97, 98, 113, 124
\@twfirst \ldots 197, 209, 210, 213
\@twtext \ldots 71, 75, 81, 85, 91
\@twfinal \ldots 191, 197, 203, 212
\@twfinal \ldots 125, 132, 236, 245, 248, 256
\@twfirst \ldots 31, 97, 98, 113, 124
\@twl@c \ldots 71, 75, 81, 85, 91
\@twlg \ldots 31, 97, 98, 113, 124
\@twl@c \ldots 261, 264, 271, 274
\@twl@c \ldots 7, 37, 43, 63, 98
\@twl@c \ldots 83, 85, 203, 205, 207
\@twl@c \ldots 101, 104, 111
\@twl@c \ldots 78, 82, 112, 123
\@twl@c \ldots 113, 116, 119
\@twl@c \ldots 141, 150, 155
\@twl@c \ldots 124, 127, 130
\@twl@c \ldots 158, 173, 200
\@twl@c \ldots 239, 240, 247
\@twl@c \ldots 204, 216, 260, 270
\@twl@c \ldots 251, 258, 259
\@twl@c \ldots 142, 144, 192
\@twl@c \ldots 62, 97, 124, 151
\@twl@c \ldots 96, 98, 101, 104
\@twl@c \ldots 152, 156, 159, 161
\@twl@c \ldots 243, 244, 247, 251
\@twl@c \ldots 232, 233, 237
\@twl@c \ldots 11.
Index

64, 70, 146, 154, 156, 159, 161, 166, 192, 221, 223
\c@twtokb .......... 11, 192, 228, 231
\c@twtoph ...... \c@twtopht . \c@twtoplines .......
\c@twvslig ..... \c@twvwindow ... \c@twvwinlines ...... \c@twxfpf .... 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
\R \put ............... 47
\putstuffinpic 178, 180
\S \shapedcutout (environment) ... 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, 137, 210–212, 246, 250, 262, 266, 272, 276
\Vsplit ...... 75, 81, 95, 96, 111, 197, 203, 239, 243, 258
\W \windowpagestuff . 2, 33