Abstract

This package implements a way to indicate modifications in a \LaTeX-document by putting bars in the margin. It realizes this by making use of the `special' commands supported by 'dvi drivers'. Currently four different drivers are supported. More can easily be added.

1 Introduction

Important note Just as with cross references and labels, you usually need to process the document twice (and sometimes three times) to ensure that the changebars come out correctly. However, a warning will be given if another pass is required.

Features

- Changebars may be nested within each other. Each level of nesting can be given a different thickness bar.
• Changebars may be nested in other environments including floats and footnotes.

• Changebars are applied to all the material within the “barred” environment, including floating bodies regardless of where the floats float to. An exception to this is margin floats.

• Changebars may cross page boundaries.

• Changebars can appear on the outside of the columns of twocolumn text.

2 The user interface

This package has options to specify some details of its operation, and also defines several macros.

2.1 The package options

One set of package options\(^1\) specify the driver that will be used to print the document can be indicated. The driver may be one of:

- DVItoLN03
- DVItoPS
- DVIps
- emTeX

The drivers are represented in the normal typewriter method of typing these names, or by the same entirely in lower case.

The position of the bars may either be on the inner edge of the page (the left column on a recto or single-sided page, the right column of a verso page) by use of the innerbars package option (the default), or on the outer edge of the page by use of the the outerbars package option.

Another set of options give the user the possibility to specify that the bars should always come out on the left side of the text (leftbars) or on the right side of the text (rightbars).

The package also implements tracing for its own debugging. The package options traceon and traceoff control tracing. An additional option tracestack is available for the die hard who want to know what goes on in the internal stacks this package maintains.

2.2 Macros defined by the package

\begin{tabular}{l}
\texttt{\cbstart} \\
\texttt{\cbend}
\end{tabular}

All material between the macros \texttt{\cbstart} and \texttt{\cbend} is barred. The nesting of multiple changebars is allowed. The macro \texttt{\cbstart} has an optional parameter that specifies the width of the bar. The syntax is \texttt{\cbstart[(dimension)]}. If no width is specified, the current value of the parameter \texttt{\changebarwidth} is used. Note that \texttt{\cbstart} and \texttt{\cbend} can be used anywhere but must be correctly

\(^1\)For older documents the command \texttt{\driver} is available in the preamble of the document. It takes the options as defined for \TeX\ 2ε as argument.
nested with floats and footnotes. That is, one cannot have one end of the bar inside a floating insertion and the other outside, but that would be a meaningless thing to do anyhow.

\changebar Apart from the macros \cbstart and \cbend a proper \LaTeX \ environment is defined. The advantage of using the environment whenever possible is that \LaTeX \ will do all the work of checking the correct nesting of different environments.

\cbdelete The macro \cbdelete puts a square bar in the margin to indicate that some text was removed from the document. The macro has an optional argument to specify the width of the bar. When no argument is specified the current value of the parameter \deletebarwidth will be used.

\nochangebars The macro \nochangebars disables the changebar commands.

### 2.3 Changebar parameters

\changebarwidth The width of the changebars is controlled with the \LaTeX \ length parameter \changebarwidth. Its value can be changed with the \setlength \ command. Changing the value of \changebarwidth affects all subsequent changebars subject to the scoping rules of \setlength.

\deletebarwidth The width of the deletebars is controlled with the \LaTeX \ length parameter \deletebarwidth. Its value can be changed with the \setlength \ command. Changing the value of \changebarwidth affects all subsequent deletebars subject to the scoping rules of \setlength.

\changebarsep The separation between the text and the changebars is determined by the value of the \LaTeX \ length parameter \changebarsep. When one of the supported dvi to PostScript translators is used the ‘blackness’ of the bars can be controlled. The \LaTeX \ counter \changebargrey is used for this purpose. Its value can be changed with a command like:

\setcounter{changebargrey}{85}

The value of the counter is a percentage, where the value 0 yields black bars, the value 100 yields white bars.

\outerbars The changebars will be printed in the ‘inside’ margin of your document. This means they appear on the left side of the page. When \twoside is in effect the bars will be printed on the right side of even pages. This behaviour can be changed by including the command \outerbarstrue in your document.

### 3 Deficiencies and bugs

- The macros blindly use special points \cbminpoint through \cbmaxpoint. If this conflicts with another set of macros, the results will be unpredictable. (What is really needed is a \newspecialpoint, analogous to \newcount etc. — it’s not provided because the use of the points is rather rare.)

- There is a limit of \((\cbmaxpoint - \cbminpoint + 1)/4\) bars per page (four special points per bar). Using more than this number yeilds unpredictable results (but that could be called a feature for a page with so many bars). This limitation could be increased if desired.
• Internal macro names are all of the form `\cb@xxxx`. No checking for conflicts with other macros is done.

• This implementation does not work with the `multicolumn` package.

• The algorithms may fail if a floating insertion is split over multiple pages. In \LaTeX{} floats are not split but footnotes may be. The simplest fix to this is to prevent footnotes from being split but this may make \TeX{} very unhappy.

• The `\cbend` normally gets “attached” to the token after it rather than the one before it. This may lead to a longer bar than intended. For example, consider the sequence ‘word1 `\cbend` word2’. If there is a line break between ‘word1’ and ‘word2’ the bar will incorrectly be extended an extra line. This particular case can be fixed with the incantation ‘word1 `\cbend`{} word2’.

4 The basic algorithm

The changebars are implemented using the `\specials` of various dvi interpreting programs like `DVItoLN03` or `DVIps`. In essence, the start of a changebar defines two `\special` points in the margins at the current vertical position on the page. The end of a changebar defines another set of two points and then joins (using the “connect” `\special`) either the two points to the left or the two points to the right of the text, depending on the setting of `innerbars`, `outerbars`, `leftbars`, `rightbars` and/or `twoside`.

This works fine as long as the two points being connected lie on the same page. However, if they don’t, the bar must be artificially terminated at the page break and restarted at the top of the next page. The only way to do this (that I can think of) is to modify the output routine so that it checks if any bar is in progress when it ships out a page and, if so, adds the necessary artificial end and begin.

The obvious way to indicate to the output routine that a bar is in progress is to set a flag when the bar is begun and to unset this flag when the bar is ended. This works most of the time but, because of the asynchronous behavior of the output routine, errors occur if the bar begins or ends near a page break. To illustrate, consider the following scenario.

```
blah blah blah % page n
blah blah blah
`\cbstart` % this does its thing and set the flag
more blah
<------------- pagebreak occurs here
more blah
`\cbend` % does its thing and unsets flag
blah blah
```

Since \TeX{} processes ahead of the page break before invoking the output routine, it is possible that the `\cbend` is processed, and the flag unset, before the output routine is called. If this happens, special action is required to generate an artificial end and begin to be added to page \texttt{n} and \texttt{n+1} respectively, as it is not possible to use a flag to signal the output routine that a bar crosses a page break.

The method used by these macros is to create a stack of the beginning and end points of each bar in the document together with the page number corresponding
to each point. Then, as a page is completed, a modified output routine checks the stack to determine if any bars begun on or before the current page are terminated on subsequent pages, and handles those bars appropriately. To build the stack, information about each changebar is written to the .aux file as bars are processed. This information is re-read when the document is next processed. Thus, to ensure that changebars are correct, the document must be processed twice. Luckily, this is generally required for \LaTeX anyway.

This approach is sufficiently general to allow nested bars, bars in floating insertions, and bars around floating insertions. Bars inside floats and footnotes are handled in the same way as bars in regular text. Bars that encompass floats or footnotes are handled by creating an additional bar that floats with the floating material. Modifications to the appropriate \LaTeX macros check for this condition and add the extra bar.

5 The implementation

5.1 Declarations And Initializations

\cb@maxpoint The original version of changebar.sty only supported the DVItoLN03 specials. The LN03 printer has a maximum number of points that can be defined on a page. Also for some PostScript printers the number of points that can be defined can be limited by the amount of memory used. Therefore, the consecutive numbering of points has to be reset when the maximum is reached. This maximum can be adapted to the printers needs.
1 (*package)
2 \def\cb@maxpoint{80}

\cb@minpoint When resetting the point number we need to know what to reset it to, this is minimum number is stored in \cb@minpoint. This number has to be odd because the algorithm that decides whether a bar has to be continued on the next page depends on this.
3 \def\cb@minpoint{1}

\cb@nil Sometimes a void value for a point has to be returned by one of the macros. For this purpose \cb@nil is used.
4 \def\cb@nil{0}

\cb@nextpoint The number of the next special point is stored in the count register \cb@nextpoint and initially equal to \cb@minpoint.
5 \newcount\cb@nextpoint
6 \cb@nextpoint=\cb@minpoint

\cb@topleft \cb@topright \cb@botleft \cb@botright These for counters are used to identify the four special points that specify a changebar. The point defined by \cb@topleft is the one used to identify the changebar; the values of the other points are derived from it.
7 \newcount\cb@topleft
8 \newcount\cb@topright
9 \newcount\cb@botleft
10 \newcount\cb@botright
\cb@curbarwd  The dimension register \texttt{\cb@curbarwd} is used to store the width of the current bar.

\cb@page \cb@pagecount  The macros need to keep track of the number of pages output so far. To this end the counter \texttt{\cb@pagecount} is used. When a pagename is read from the history stack, it is stored in the counter \texttt{\cb@page}. The counter \texttt{\cb@pagecount} is initially 0; it gets incremented during the call to \texttt{@makebox} (see section 5.5).

\newdimen\cb@odd@left \newdimen\cb@odd@right \newdimen\cb@even@left \newdimen\cb@even@right  This macro calculates the (horizontal) positions of the changebars.

Because the margins can differ for even and odd pages and because changebars are sometimes on different sides of the paper we need four dimensions to store the result.

\if@twoside 
\global\advance\cb@even@left by \evensidemargin
\global\cb@even@right\cb@even@left
\global\advance\cb@even@left by -\changebarsep
\global\advance\cb@even@left by -0.5\changebarwidth
\fi 

On even sided pages we need to use \texttt{\evensidemargin} in the calculations when \texttt{twoside} is in effect.

\if@twoside 
\global\advance\cb@even@left by \evensidemargin
\global\cb@even@right\cb@even@left
\global\advance\cb@even@left by -\changebarsep
\global\advance\cb@even@left by -0.5\changebarwidth
\fi 

\global\advance\cb@even@left by \textwidth
\global\advance\cb@odd@right by \changebarsep
\global\advance\cb@odd@right by 0.5\changebarwidth

Since the changebars are drawn with the \texttt{POSTSCRIPT} command \texttt{lineto} and not as TeX-like rules the reference points lie on the center of the changebar, therefore the calculation has to add or subtract half of the width of the bar to keep \texttt{\changebarsep} whitespace between the bar and the body text.

First the position for odd pages is calculated. I
\global\advance\cb@even@right by \changebarsep
\global\advance\cb@even@right by 0.5\changebarwidth
\else
Otherwise just copy the result for odd pages.
\global\let\cb@even@left\cb@odd@left
\global\let\cb@even@right\cb@odd@right
\fi}
\cb@removedim In PostScript code, length specifications are without dimensions. Therefore we need a way to remove the letters ‘pt’ from the result of the operation \the⟨dimen⟩. This can be done by defining a command that has a delimited argument like:
\def\cb@removedim#1pt{#1}
We encounter one problem though, the category code of the letters ‘pt’ is 12 when produced as the output from \the⟨dimen⟩. Thus the characters that delimit the argument of \cb@removedim also have to have category code 12. To keep the changes local the macro \cb@removedim is defined in a group.
\{\catcode'\p=12 \catcode'\t=12 \gdef\cb@removedim#1pt{#1}\}
\cb@removedim

5.2 Option Processing

The user should select the specials that should be used by specifying the driver name as an option to the \usepackage call. Possible choices are:

- DVItoLN03
- DVItoPS
- DVIPS
- em\TeX

The intent is that the driver names should be case-insensitive, but the following code doesn’t achieve this: it only permits the forms given above and their lowercase equivalents.

\DeclareOption{DVItoLN03}{\global\chardef\cb@driver@setup=0}\relax
\DeclareOption{dvitoIn03}{\global\chardef\cb@driver@setup=0}\relax
\DeclareOption{DVItoPS}{\global\chardef\cb@driver@setup=1}\relax
\DeclareOption{dvitops}{\global\chardef\cb@driver@setup=1}\relax
\DeclareOption{DVIPS}{\global\chardef\cb@driver@setup=2}\relax
\DeclareOption{dvips}{\global\chardef\cb@driver@setup=2}\relax
\DeclareOption{emTeX}{\global\chardef\cb@driver@setup=3}\relax
\DeclareOption{emtex}{\global\chardef\cb@driver@setup=3}\relax
\DeclareOption{Textures}{\global\chardef\cb@driver@setup=4}\relax
\DeclareOption{Textures}{\global\chardef\cb@driver@setup=4}\relax

The new features of \LaTeX{} make it possible to implement the outerbars option.
\DeclareOption{outerbars}{\outerbarstrue}
\DeclareOption{innerbars}{\outerbarsfalse}
It is also possible to specify that the change bars should *always* be printed on either the left or the right side of the text. For this we have the options `leftbars` and `rightbars`. Specifying *either* of these options will overrule a possible `twoside` option at the document level.

```latex
\DeclareOption{leftbars}{%
  \def\cb@positions{%
    \global\cb@odd@left=\hoffset
    \global\cb@even@left=\cb@odd@left
    \global\advance\cb@odd@left by \oddsidemargin
    \global\advance\cb@odd@left by -\changebarsep
    \global\advance\cb@odd@left by -0.5\changebarwidth
    \if@twoside
      \global\advance\cb@even@left by \evensidemargin
      \global\advance\cb@even@left by -\changebarsep
      \global\advance\cb@even@left by -0.5\changebarwidth
    \else
      \global\let\cb@even@left\cb@odd@left
    \fi
    \global\let\cb@odd@right\cb@odd@left
    \global\let\cb@even@right\cb@even@left
  }}

\DeclareOption{rightbars}{%
  \def\cb@positions{%
    \global\cb@odd@right=\hoffset
    \global\cb@even@right=\cb@odd@right
    \global\advance\cb@odd@right by \oddsidemargin
    \global\advance\cb@odd@right by \textwidth
    \global\advance\cb@odd@right by \changebarsep
    \global\advance\cb@odd@right by 0.5\changebarwidth
    \if@twoside
      \global\advance\cb@even@right by \evensidemargin
      \global\advance\cb@even@right by \textwidth
      \global\advance\cb@even@right by \changebarsep
      \global\advance\cb@even@right by 0.5\changebarwidth
    \else
      \global\let\cb@even@left\cb@odd@left
    \fi
    \global\let\cb@odd@left\cb@odd@right
    \global\let\cb@even@left\cb@even@right
  }}
```

A set of options to control tracing.

```latex
\DeclareOption{traceon}{\cb@tracertrue}
\DeclareOption{traceoff}{\cb@tracefalse}
\DeclareOption{tracestacks}{\let\cb@trace@stack\cb@@show@stack}
```

Signal an error if an unknown option was specified.

```latex
\DeclareOption*{\OptionNotUsed\PackageError
  {Unrecognised option '{\CurrentOption}'%}
  {known options are dvitoln03, dvitops, dvips and entex,}MessageBreak
  outerbars, innerbars, leftbars and rightbars.}}
```

The default is to have the change bars on the left side of the text on odd pages.

```latex
\ExecuteOptions{innerbars,traceoff,dvips}
```
When the stack tracing facility is turned on this command is executed. It needs to be defined before we call \ProcessOptions.

\def\cb@@show@stack#1{%
\cb@trace%{
stack status at #1:\MessageBreak
current stack: \cb@currentstack\MessageBreak
@spaces end stack: \cb@endstack\MessageBreak
@space\space begin stack: \cb@beginstack\MessageBreak
history stack: \cb@historystack
}
}}

The default is to not trace the stacks. This is achieved by \let\cb@trace@stack\relax.
\let\cb@trace@stack@stack\@gobble
\ProcessOptions\relax

\cb@trace A macro that formats the tracing messages.
\newcommand\cb@trace[1]{%
\if@cb@trace
\GenericWarning{(changebar)@spaces@spaces}%
\PackageWarning{changebar: #1@gobble}%
\fi
}

5.3 User Level Commands And Parameters

\driver The user can select the specials that should be used by calling the command \driver{(drivername)}. Possible choices are:

- DVIToLN03
- DVIToPS
- DVIPS
- emTeX
- TExtures

This command can only be used in the preamble of the document.

The argument should be case-insensitive, so it is turned into a string containing all uppercase characters. To keep some definitions local, everything is done within a group.

% \changes{v3.3e}{1998/02/24}{Added \cs{Textures}}
\if@compatibility
\bgroup\edef\next{\def\noexpand\tempa{#1}}%
\uppercase\expandafter{\next}%
\def\LN{DVITOLN03}%
\def\DVIToPS{DVITOPS}%
\def\DVIPS{DVIPS}%
\def\emTeX{EMTEX}%
\def\Textures{TEXTURES}%
\egroup
}
The choice has to be communicated to the macro \cb@setup@specials that will be called from within \document. For this purpose the control sequence \cb@driver@setup is used. It receives a numeric value using \chardef.

\begin{verbatim}
129 \global\chardef\cb@driver@setup=0\relax
130 \ifx\tempa\LN \global\chardef\cb@driver@setup=0\fi
131 \ifx\tempa\DVIToPS \global\chardef\cb@driver@setup=1\fi
132 \ifx\tempa\DVIPS \global\chardef\cb@driver@setup=2\fi
133 \ifx\tempa\emTeX \global\chardef\cb@driver@setup=3\fi
134 \ifx\tempa\Textures \global\chardef\cb@driver@setup=4\fi
\egroup}
\end{verbatim}

We add \driver to \@preamblecmds, which is a list of commands to be used only in the preamble of a document.

\begin{verbatim}
136 \{\def\do{\noexpand\do\noexpand}
137 \xdef\@preamblecmds{\@preamblecmds \do\driver}
138 \}
139 \fi
\end{verbatim}

\cb@setup@specials The macro \cb@setup@specials defines macros containing the driver specific \special macros. It will be called from within the \begin{document} command.

\cb@trace@defpoint When tracing is on, write information about the point being defined to the log file.

\begin{verbatim}
140 \def\cb@trace@defpoint#1#2{%
141 \cb@trace%
142 defining point \the#1 at position \the#2
143 \MessageBreak
144 \cb@pagecount: \the\cb@pagecount; page \thepage}%
\end{verbatim}

\cb@trace@connect When tracing is on, write information about the points being connected to the log file.

\begin{verbatim}
145 \def\cb@trace@connect#1#2#3{%
146 \cb@trace%
147 connecting points \the#1 and \the#2; barwidth: \the#3
148 \MessageBreak
149 \cb@pagecount: \the\cb@pagecount; page \thepage}%
\end{verbatim}

\cb@defpoint The macro \cb@defpoint is used to define one of the two points of a bar. It has two arguments, the number of the point and the distance from the left side of the paper. Its syntax is: \cb@defpoint{(number)}{(length)}.

\cb@resetpoints The macro \cb@resetpoints can be used to instruct the printer driver that it should send a corresponding instruction to the printer. This is really only used for the LN03 printer.

\cb@connect The macro \cb@connect is used to instruct the printer driver to connect two points with a bar. The syntax is \cb@connect{(number)}{(number)}{(length)} The two (number)s indicate the two points to be connected; the (length) is the width of the bar.

\begin{verbatim}
150 \def\cb@setup@specials{%
\end{verbatim}

The control sequence \cb@driver@setup expands to a number which indicates the driver that will be used. The original changebar.sty was written with only the
The special syntax of the program DVItoLN03 (actually one of its predecessors, ln03dvi). Therefore this syntax is defined first.

\ifcase\cb@driver@setup
\def\cb@defpoint##1##2{%
\special{ln03:defpoint \the##1(\the##2,)}%
\cb@trace@defpoint##1##2%
\def\cb@connect##1##2##3{%
\special{ln03:connect \the##1\space\space \the##2\space \the##3}%
\cb@trace@connect##1##2##3%
\def\cb@resetpoints{%
\special{ln03:resetpoints \cb@minpoint \space\cb@maxpoint}}%
\or
\def\cb@defpoint##1##2{%
\special{dvitops: inline}
\expandafter\cb@removedim\the##2\space 6.5536 mul\space
/CBarX\the##1\space exch def currentpoint exch pop
/CBarY\the##1\space exch def}%
\def\cb@connect##1##2##3{%
\special{dvitops: inline}
gsave \thechangebargrey\space 100 div setgray
\expandafter\cb@removedim\the##3\space 6.5536 mul\space
CBarX\the##1\space\space CBarY\the##1\space\space moveto
CBarX\the##2\space\space CBarY\the##2\space\space lineto
stroke grestore}%
\let\cb@resetpoints\relax
\else
\def\cb@defpoint##1##2{%
\special{ps:}
\expandafter\cb@removedim\the##2\space Resolution\space mul\space 72.27\space div\space
/CBarX\the##1\space exch def currentpoint exch pop
/CBarY\the##1\space exch def}%
\def\cb@connect##1##2##3{%
\special{ps:}
gsave \thechangebargrey\space 100 div setgray
\expandafter\cb@removedim\the##3\space Resolution\space mul\space 72.27\space div\space
CBarX\the##1\space\space CBarY\the##1\space\space moveto
CBarX\the##2\space\space CBarY\the##2\space\space lineto
stroke grestore}%
\fi

The first extension to the changebar option was for the special syntax of the program DVItoPS by James Clark.

\or
\def\cb@defpoint##1##2{%
\special{dvitops: inline}
\expandafter\cb@removedim\the##2\space
Resolution\space mul\space 72.27\space div\space
/CBarX\the##1\space exch def currentpoint exch pop
/CBarY\the##1\space exch def}%
\def\cb@connect##1##2##3{%
\special{dvitops: inline}
gsave \thechangebargrey\space 100 div setgray
\expandafter\cb@removedim\the##3\space
Resolution\space mul\space 72.27\space div\space
setlinewidth
CBarX\the##1\space\space CBarY\the##1\space\space moveto
CBarX\the##2\space\space CBarY\the##2\space\space lineto
}

The program DVIps by Thomas Rockicki is also supported. The PostScript code is nearly the same as for DVItoPS, but the coordinate space has a different dimension. Also this code has been made resolution independent, whereas the code for DVItoPS might still be resolution dependent.

So far all the positions have been calculated in pt units. DVIps uses pixels internally, so we have to convert pts into pixels which of course is done by dividing by 72.27 (pts per inch) and multiplying by Resolution giving the resolution of the PostScript device in use as a PostScript variable.
The latest addition is for the drivers written by Eberhard Mattes. The \special syntax used here is supported since version 1.5 of his driver programs.

The following definitions are valid tested with TExtures version 1.7.7, but will very likely also work with later releases of TExtures.

The \cbdelete command seemed to create degenerate lines (i.e., lines of 0 length). PostScript will not render such lines unless the linecap is set to 1, (semi-circular ends) in which case a filled circle is shown for such lines.

When code for other drivers should be added it can be inserted here. When someone makes a mistake and somehow selects an unknown driver a warning is issued and the macros are defined to be no-ops.

When code for other drivers should be added it can be inserted here. When someone makes a mistake and somehow selects an unknown driver a warning is issued and the macros are defined to be no-ops.

The macro \cbstart starts a new changebar. It has an (optional) argument that will be used to determine the width of the bar. The default width is \changebarwidth.
The macro \cbend (surprisingly) ends a changebar. The macros \cbstart and \cbend can be used when the use of a proper \LaTeX{} environment is not possible.

The macro \cbdelete inserts a ‘deletebar’ in the margin. It too has an optional argument to determine the width of the bar. The default width (and length) of it are stored in \deletebarwidth.

Deletebars are implemented as a special ‘change bar’. The bar is started and immediately ended. It is as long as it is wide.

The macros \changebar and \endchangebar have the same function as \cbstart and \cbend but they can be used as a \LaTeX{} environment to enforce correct nesting. They can not be used in the \tabular and \tabbing environments.

To disable changebars altogether without having to remove them from the document the macro \nochangebars is provided. It makes no-ops of three internal macros.

The default width of the changebars is stored in the dimension register \changebarwidth.

The default width of the deletebars is stored in the dimension register \deletebarwidth.

The default separation between all bars and the text is stored in the dimen register \changebarsep.

When the document is printed using one of the PostScript drivers the bars do not need to be black; with PostScript it is possible to have grey, and colored, bars. The percentage of greyness of the bar is stored in the count register \changebargrey. It can have values between 0 (meaning white) and 100 (meaning black). It has been suggested to introduce colored change bars but such an option has yet to be implemented.
5.4 Macros for beginning and ending bars

\texttt{\cb@start} This macro starts a change bar. It assigns a new value to the current point and advances the counter for the next point to be assigned. It pushes this info onto \texttt{\cb@currentstack} and then sets the point by calling \texttt{\cb@setBeginPoints} with the point number. Finally, it writes the .aux file.

\begin{verbatim}
\def\cb@start[#1]{%
  \cb@topleft=\cb@nextpoint
  \cb@curbarwd#1\relax
  \cb@push\cb@currentstack
  Now find out on which page the start of this bar finally ends up; due to the asynchronous nature of the output routine it might be a different page. The macro \texttt{\cb@checkpage} finds the page number on the history stack.
  \cb@checkpage@ne
  Temporarily assign the page number to \texttt{\cb@pagecount} as that register is used by \texttt{\cb@setBeginPoints}. Note that its value is offset by one from the page counter.
  \@tempcnta\cb@pagecount
  \cb@pagecount\cb@page\advance\cb@pagecount\m@ne
  \ifvmode
    \cb@setBeginPoints
  \else
    \vbox to \z@{\
    When we are in horizontal mode we jump up a line to set the starting point of the changebar.
    \vskip -\ht\strutbox
    \cb@setBeginPoints
    \vskip \ht\strutbox}%
  \fi
  \cb@pagecount\@tempcnta
  \cb@advancePoint}
\end{verbatim}

\texttt{\cb@advancePoint} The macro \texttt{\cb@advancePoint} advances the count register \texttt{\cb@nextpoint}. When the maximum number is reached, the numbering is reset.

\begin{verbatim}
\def\cb@advancePoint{%
  \global\advance\cb@nextpoint by 4\relax
  \ifnum\cb@nextpoint>\cb@maxpoint
    \global\cb@nextpoint=\cb@minpoint\relax
  \fi}
\end{verbatim}

\texttt{\cb@end} This macro ends a changebar. It pops the current point and nesting level off \texttt{\cb@currentstack} and sets the end point by calling \texttt{\cb@setEndPoints} with the parameter corresponding to the \textit{beginning} point number. It writes the .aux file and joins the points.

\begin{verbatim}
\def\cb@end{%
  \cb@trace@stack{end of bar on page \the\c@page}%
  \cb@pop\cb@currentstack
  \ifnum\cb@topleft=\cb@nil
    \PackageWarning{Changebar}%
  \else
    \cb@setEndPoints
  \fi
  \cb@advancePoint}
\end{verbatim}
{Badly nested changebars; Expect erroneous results}\%
\else
\cb@checkpage to find the page this point finally ends up on.
\cb@checkpage\tw@
Again, we need to temporarily overwrite \cb@pagecount.
\@tempcnta\cb@pagecount
\cb@pagecount\cb@page\advance\cb@pagecount\m@ne
\cb@setEndPoints
\cb@pagecount\@tempcnta
\fi
\ignorespaces}
\cb@checkpage

The macro \cb@checkpage checks the history stack in order to find out on which page a set of points finally ends up.
We expect the identification of the points in \cb@topleft and \cb@page. The resulting page will be stored in \cb@page.
\def\cb@checkpage#1{%
\@tempcnta\cb@topleft\@tempcntb\cb@page
Then pop the history stack.
\cb@pop\cb@historystack
If it was empty there is nothing to check and we’re done.
\ifnum\cb@topleft=\cb@nil
\else
Now keep popping the stack until \cb@topleft is no longer less than the value of \@tempcnta. The values popped from the stack are pushed on a temporary stack to be pushed back later. This could perhaps be implemented more efficiently if the stacks had a different design.
\@whilenum\cb@topleft<\@tempcnta\do{%\cb@push\cb@tempstack
\cb@pop\cb@historystack
When the users adds chang bars to his document we might run out of the history stack before we find a match. This would send \TeX into an endless loop if it wasn’t detected and handled.
\ifnum\cb@topleft=\cb@nil
\cb@trace{Ran out of history stack, new changebar?}\
In this case we give \cb@topleft an ‘impossible value’ to remember this special situation.
\cb@topleft\cb@maxpoint\advance\cb@topleft\@ne
\fi
}%
If we are looking for the start point of a bar we may have found it now, for the end point we need to pop one more value. If \cb@topleft has become larger than \cb@maxpoint we haven’t found what we’re looking for and we’ve run out of the stack.
\ifnum\cb@topleft>\cb@maxpoint\else
\ifodd#1\else

\end{document}
Now that we’ve found it overwrite \@tempcntb with the \cb@page from the stack.

\fi

Now we restore the history stack to it’s original state.

\fi

Finally return the correct values.

\def\cb@setBeginPoints{%
    \cb@topright=\cb@topleft\advance\cb@topright by\@ne
    \ifodd\cb@pagecount
        \cb@defpoint\cb@topleft\cb@even@left
        \cb@defpoint\cb@topright\cb@even@right
    \else
        \cb@defpoint\cb@topleft\cb@odd@left
        \cb@defpoint\cb@topright\cb@odd@right
    \fi
    \cb@writeAux\cb@topleft
}

The macro \cb@setBeginPoints assigns a position to the top left and top right points. It determines whether the point is on an even or an odd page and uses the right dimension to position the point. Keep in mind that the value of \cb@pagecount is one less than the value of \cb@page unless the latter has been reset by the user.

The top left point is used to write an entry on the .aux file to create the history stack on the next run.

\def\cb@setEndPoints{%
    \cb@topright=\cb@topleft\advance\cb@topright by\@ne
    \ifodd\cb@pagecount
        \cb@defpoint\cb@topleft\cb@even@left
        \cb@defpoint\cb@topright\cb@even@right
    \else
        \if@twoside
            \cb@defpoint\cb@topleft\cb@odd@left
            \cb@defpoint\cb@topright\cb@odd@right
        \else
            \cb@defpoint\cb@topleft\cb@even@left
            \cb@defpoint\cb@topright\cb@even@right
        \fi
    \fi
    \cb@defpoint\cb@botleft\cb@even@left
    \cb@defpoint\cb@botright\cb@even@right
}

The macro \cb@setEndPoints assigns positions to the bottom points for a change bar. It then instructs the driver to connect two points with a bar. The macro assumes that the width of the bar is stored in \cb@curbarwd.

The bottom right point is used to write to the .aux file to signal the end of the current bar on the history stack.
The macro \texttt{\cb@writeAux} writes information about a changebar point to the auxiliary file. The number of the point, the pagenumber and the width of the bar are written out as arguments to \texttt{\cb@barpoint}. This latter macro will be expanded when the auxiliary file is read in. The macro assumes that the width of bar is stored in \texttt{\cb@curbarwd}.

The code is only executed when auxiliary files are enabled, as there’s no sense in trying to write to an unopened file.

5.5 Macros for Making It Work Across Page Breaks

These internal \LaTeX macros are modified in order to end the changebars spanning the current page break (if any) and restart them on the next page. The modifications are needed to reset the special points for this page and add begin bars to top of box255. The bars carried over from the previous page, and hence
to be restarted on this page, have been saved on the stack \texttt{\begin{stack}}. This
stack is used to define new starting points for the change bars, which are added
to the top of box \texttt{\cclv}. Then the stack \texttt{\end{stack}} is built and processed
by \texttt{\processActive}. Finally the original \texttt{\makecol} (saved as \texttt{\makecol})
is executed.

370 \begin{verbatim}
\let\cb@makecol\makecol
\def\makecol{%
  \cb@resetpoint
  $\cb@startSpanBars$
  \unvbox\cclv
  $\boxmaxdepth$\maxdepth
  $\cb@processActive$
  \cb@makecol
  \cb@resetpoint
  $\cb@startSpanBars$
  \unvbox\cclv
  $\boxmaxdepth$\maxdepth
  \cb@processActive
  \cb@makecol
}
\end{verbatim}

When \LaTeX{} makes a page with only floats it doesn’t use \texttt{\makecol}; instead it
calls \texttt{\vtryfc}, so we have to modify this macro as well.

384 \begin{verbatim}
\let\vtryfc\vtryfc
\def\vtryfc{%
  \let\writeAux\gobble
  $\setbox\outputbox$\vbox{%
    $\cb@resetpoint$
    $\cb@startSpanBars$
    \unvbox\cclv
    $\boxmaxdepth$\maxdepth
  }
  \global$\advance\cb@pagecount$by\@ne
  \cb@processActive
  \cb@vtryfc
}
\end{verbatim}

This macro processes each element on span stack. Each element represents a bar
that crosses the page break. There could be more than one if bars are nested. It
works as follows:

pop top element of span stack
if point null (i.e., stack empty) then done
else
  do an end bar on box255
  save start for new bar at top of next page in \texttt{\startSaves}
  push active point back onto history stack (need to reprocess
  on next page).

395 \begin{verbatim}
\def\processActive{%
  \cb@pop\endstack
  \ifnum\cb@topleft=\nil
    \else
    \setbox\cclv\vbox{%
      \unvbox\cclv
    $\boxmaxdepth$\maxdepth
  $\cb@processActive$
  \end{verbatim}
This macro defines new points for each bar that was pushed on the \cb@beginstack. Afterwards \cb@beginstack is empty.

\cb@buildstack The macro \cb@buildstack initializes the stack with open bars and starts populating it.

\cb@pushNextActive This macro pops the top element off the history stack (\cb@historystack). If the top left point is on a future page, it is pushed back onto the history stack and processing stops. If the point on the current or a previous page and it has an odd number, the point is pushed on the stack with end points \cb@endstack; if the point has an even number, it is popped off the stack with end points since the bar to which it belongs has terminated on the current page.

5.6 Macros For Managing The Stacks of Bar points

The macros make use of four stacks corresponding to \special defpoints. Each stack takes the form \texttt{<element> ... <element>}
Each element is of the form xxxnyypzzzl where xxx is the number of the special point, yyy is the page on which this point is set, and zzz is the dimension used when connecting this point.

The stack \texttt{\cb@historystack} is built from the log information and initially lists all the points. As pages are processed, points are popped off the stack and discarded.

The stack \texttt{\cb@endstack} and \texttt{\cb@beginstack} are two temporary stacks used by the output routine and contain the stack with definitions for all bars crossing the current pagebreak (there may be more than one with nested bars). They are built by popping elements off the history stack.

The stack \texttt{\cb@currentstack} contains all the current bars. A \texttt{\cb@start} pushes an element onto this stack. A \texttt{\cb@end} pops the top element off the stack and uses the info to terminate the bar.

For performance and memory reasons, the history stack, which can be very long, is special cased and a file is used to store this stack rather than an internal macro. The “external” interface to this stack is identical to what is described above. However, when the history stack is popped, a line from the file is first read and appended to the macro \texttt{\cb@historystack}.

\texttt{\cb@initstack} A macro to (globally) initialize a stack.

\texttt{\cb@historystack} \texttt{\cb@write} \texttt{\cb@read}

We need to initialise a stack to store the entries read from the external history file.

We also need to allocate a read and a write stream for the history file.

And we open the history file for writing (which is done when the \texttt{.aux} file is read in).

\texttt{\immediate\openout\cb@write=\jobname.cb}\relax

\texttt{\cb@endstack} \texttt{\cb@beginstack}

Allocate two stacks for the bars that span the current page break.

\texttt{\cb@tempstack}

Allocate a stack for temporary storage

\texttt{\cb@currentstack}

And we allocate an extra stack that is needed to implement nesting without having to rely on \TeX’s grouping mechanism.

\texttt{\cb@pop}

This macro pops the top element off the named stack and puts the point value into \texttt{\cb@topleft}, the page value into \texttt{\cb@page} and the bar width into \texttt{\cb@curbarwd}. If the stack is empty, it returns a void value (\texttt{\cb@nil}) in \texttt{\cb@topleft} and sets \texttt{\cb@page=0}.

\texttt{\def\cb@pop#1{}}
\cb@carcdr This macro is used to ‘decode’ a stack entry.

\cb@push The macro \cb@push Pushes \cb@topleft, \cb@page and \cb@curbarwd onto the top of the named stack.

\cb@barpoint The macro \cb@barpoint populates the history file. It writes one line to .cb file which is equivalent to one (element) described above.

5.7 Macros For Checking That The .aux File Is Stable

\AtBeginDocument While reading the .aux file, \LaTeX{} has created the history stack in a separate file. We need to close that file and open it for reading. Also the ‘initialisation’ of the \special{} commands has to take place. While we are modifying the macro we also include the computation of the possible positions of the changebars

For these actions we need to add to the \LaTeX{} begin-document hook.

\AtEndDocument We need to issue a \clearpage{} to flush rest of document. (Note that I believe there is contention in this area: are there in fact situations in which the end-document hooks need to be called before the final \clearpage{} — the documentation of \LaTeX{} itself implies that there are.) Then closes the .cb file and reopens it for checking. Initialize history stack (to be read from file). Let \cb@barpoint=\cb@checkHistory for checking.

\AtBeginDocument{\%}
\cb@setup@specials
\cb@positions
\immediate\closeout\cb@write
\immediate\openin\cb@read=\jobname.cb}

\AtEndDocument{\%}
\clearpage
\cb@initstack\cb@historystack
\immediate\closein\cb@read
\immediate\openin\cb@read=\jobname.cb%}
\let\cb@barpoint=\cb@checkHistory}
\cb@checkHistory\hspace{10pt} Pops the top of the history stack (\jobname.cb) and checks to see if the point and page numbers are the same as the arguments #1 and #2 respectively. Prints a warning message if different.

\ cb@checkHistory#1#2#3{% 474 \cb@pop\cb@historystack 475 \ifnum #1=\cb@topleft\relax 476 \ifnum #2=\cb@page\relax 477 Both page and point numbers are equal; do nothing, 478 \else 479 \cb@error 480 \fi 481 \else 482 \cb@error 483 \fi} 484\cb@error\hspace{10pt} When a mismatch between the changebar information in the auxiliary file and the history stack is detected a warning is issued; further checking is disabled.

\cb@error\hspace{10pt} \def\cb@error{% 484 \PackageWarning{Changebar}% 485 \{Changebar info has changed.\MessageBreak 486 Rerun to get the bars right\}% 488 \gdef\cb@checkHistory##1##2##3{}% 489 \let\cb@barpoint=\cb@checkHistory} 5.8 Macros For Making It Work With Nested Floats/Footnotes

\end@float\hspace{10pt} This is a replacement for the \LaTeX-macro of the same name. All it does is check to see if changebars are active and, if so, it puts changebars around the box containing the float. Then it calls the original \LaTeX \end@float.

\end@float\hspace{10pt} \def\end@float{% 490 \cb@trace@stack{end float on page \the\c@page}% 491 \cb@pop\cb@currentstack 492 \ifnum\cb@topleft=\cb@nil 493 \cb@push\cb@currentstack 494 \global\cb@curbarwd=\cb@curbarwd 495 \egroup 496 \global\setbox\@currbox=\vbox{\cb@start[\cb@curbarwd]\unvbox\@currbox}\cb@end 497 \cb@endfloat} 498 This only works if this new version of \end@float is really used. With \LaTeXX2.09 the documentstyles used to contain:

\let\endfigure=\end@float
In that case this binding has to be repeated after the redefinition of `\end@float`. However, the \LaTeX\ 2ε class files use `\newenvironment` to define the `figure` and `table` environments. In that case there is no need to rebind `\endfigure`.

\@footnotetext

This is a replacement for the \LaTeX\ macro of the same name. It simply checks to see if changebars are active, and if so, wraps the macro argument (i.e., the footnote) in changebars.

503 \let\cb@footnote=\@footnotetext
504 \long\def\@footnotetext#1{\%  
505 \cb@trace@stack{end footnote on page \the@c@page}\%  
506 \cb@pop\cb@currentstack  
507 \ifnum\cb@topleft=\cb@nil  
508 \cb@footnote[#1]\%  
509 \else  
510 \cb@push\cb@currentstack  
511 \edef\cb@temp{\the\cb@curbarwd}\%  
512 \cb@footnote{\cb@start[\cb@temp]\#1\cb@end}\%  
513 \fi}

\@mpfootnotetext

Replacement for the \LaTeX\ macro of the same name. Same thing as `\@footnotetext`.

514 \let\cb@mpfootnote=\@mpfootnotetext
515 \long\def\@mpfootnotetext#1{\%  
516 \cb@pop\cb@currentstack  
517 \ifnum\cb@topleft=\cb@nil  
518 \cb@mpfootnote[#1]\%  
519 \else  
520 \cb@push\cb@currentstack  
521 \edef\cb@temp{\the\cb@curbarwd}\%  
522 \cb@mpfootnote{\cb@start[\cb@temp]\#1\cb@end}\%  
523 \fi}

⟨/package⟩