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 six different drivers are supported, plus pdftex and Xe\TeX-support. More can easily be added.

*This file has version number v3.6d, last revised 2022-05-06.
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 \texttt{twocolumn} text.

• The colour of the changebars can be changed. This has so far been tested with the dvips, \texttt{pdftex}, \texttt{vtx} and \texttt{dextra} drivers, but it may also work with other PostScript based drivers. It will \textit{not} work for the \texttt{DVItoLN03} and \texttt{emTeX} drivers. For colored changebars to work, make sure that you specify the option \texttt{color} or \texttt{xcolor}.

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

2.1.1 Specifying the printer driver

One set of package options\footnote{For older documents the command \texttt{\driver} is available in the preamble of the document. It takes the options as defined for \texttt{I\TeX 2e} as argument.} specify the driver that will be used to print the document can be indicated. The driver may be one of:

• \texttt{DVItoLN03}

• \texttt{DVItoPS}

• \texttt{DVIps}

• \texttt{emTeX}

• \texttt{TeXtures}
• VT\TeX
• PDF\TeX
• Xe\TeX

The drivers are represented in the normal typewriter method of typing these names, or by the same entirely in lower case. Since version 3.4d the driver can be specified in a configuration file, not surprisingly called \texttt{changebar.cfg}. If it contains the command \texttt{\ExecuteOption{textures}} the \texttt{textures} option will be used for all documents that are processed while the configuration file is in \TeX’s search path.

### 2.1.2 Specifying the bar position

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 \texttt{innerbars} package option (the default), or on the outer edge of the page by use of the \texttt{outerbars} package option.

Another set of options gives the user the possibility of specifying that the bars should \textit{always} come out on the left side of the text (\texttt{leftbars}) or on the right side of the text (\texttt{rightbars}).

\textit{Note} that these options only work for \texttt{onecolumn} documents and will be ignored for a \texttt{twocolumn} document.

### 2.1.3 Color

For people who want their changebars to be colourfull the options \texttt{color} and \texttt{xcolor} are available. They define the user command \texttt{\cbcolor} and load either the \texttt{color} or the \texttt{xcolor} package.

If a configuration file specifies the \texttt{color} option and you want to override it for a certain document you can use the \texttt{grey} option.

### 2.1.4 Tracing

The package also implements tracing for its own debugging. The package options \texttt{traceon} and \texttt{traceoff} control tracing. An additional option \texttt{tracestacks} is available for the die hard who wants to know what goes on in the internal stacks maintained by this package.

### 2.2 Macros defined by the package

\texttt{\cbstart \cbend}

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{\changewbarwidth} is used. Note that \texttt{\cbstart} and \texttt{\cbend} can be used anywhere but must be correctly 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 \texttt{\textbackslash cbstart} and \texttt{\textbackslash 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 \texttt{\textbackslash 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 \texttt{\textbackslash deletebarwidth} will be used.

\nochangebars  The macro \texttt{\textbackslash nochangebars} disables the changebar commands.

\cbcolor  This macro is defined when the \texttt{color} option is selected. It’s syntax is the same as the \texttt{color} command from the \texttt{color} package.

2.3 Changebar parameters

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

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

\changebarsep  The separation between the text and the changebars is determined by the value of the \LaTeX{} length parameter \texttt{\changebarsep}.

\changebargrey  When one of the supported dvi to PostScript translators is used the ‘blackness’ of the bars can be controlled. The \LaTeX{} counter \texttt{\changebargrey} is used for this purpose. Its value can be changed with a command like:

\begin{verbatim}
\setcounter{changebargrey}{85}
\end{verbatim}

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 \texttt{twoside} is in effect the bars will be printed on the right side of even pages.

3 Deficiencies and bugs

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

- There is a limit of \((\texttt{\cb@maxpoint} - \texttt{\cb@minpoint} + 1)/4\) bars per page (four special points per bar). Using more than this number yields unpredictable
results (but that could be called a feature for a page with so many bars). This limitation could be increased if desired. There is no such limit with PDFTeX or Xe\TeX.

- 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 \texttt{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 \texttt{\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 \texttt{\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 \texttt{\cbend{}} word2’.

- The colour support has only been tested with the \texttt{dvips} and \texttt{pdftex} drivers.

4 The basic algorithm

The changebars are implemented using the \texttt{\specials} of various dvi interpreting programs like \texttt{DVItoLN03} or \texttt{DVIPS}. In essence, the start of a changebar defines two \texttt{\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” \texttt{\special}) either the two points to the left or the two points to the right of the text, depending on the setting of \texttt{innerbars}, \texttt{outerbars}, \texttt{leftbars}, \texttt{rightbars} and/or \texttt{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.

\begin{verbatim}
blah blah blah % page n
blah blah blah
\cbstart % this does its thing and set the flag
more blah
\textless---------- pagebreak occurs here
more blah
\cbend % does its thing and unsets flag
\end{verbatim}
Since \TeX processes ahead of the page break before invoking the output routine, it is possible that the \texttt{\textbackslash 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 \texttt{.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. With PDF\LaTeX generally three (or even more) runs are necessary.

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

\texttt{\textbackslash cb@maxpoint} The original version of \texttt{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.

\texttt{\textbackslash cb@minpoint} When resetting the point number we need to know what to reset it to, this is minimum number is stored in \texttt{\textbackslash cb@minpoint}. \textbf{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.

\texttt{\textbackslash cb@nil} Sometimes a void value for a point has to be returned by one of the macros. For this purpose \texttt{\textbackslash cb@nil} is used.
\texttt{\cb@nextpoint} The number of the next special point is stored in the count register \texttt{\cb@nextpoint}\texttt{ } and initially equal to \texttt{\cb@minpoint}.

\begin{verbatim}
5 \newcount\cb@nextpoint
6 \cb@nextpoint=\cb@minpoint
\end{verbatim}

\texttt{\cb@topleft} These four counters are used to identify the four special points that specify a changebar. The point defined by \texttt{\cb@topleft} is the one used to identify the changebar; the values of the other points are derived from it.

\texttt{\cb@topleft} \texttt{\cb@topright} \texttt{\cb@botleft} \texttt{\cb@botright}

\texttt{\cb@cunta} \texttt{\cb@cuntb} \texttt{\cb@dima}

\texttt{\cb@curbarwd} Sometimes we need temporarily store a value. For this purpose two count registers and a dimension register are allocated.

\texttt{\cb@curbarwd}

\texttt{\cb@page} \texttt{\cb@pagecount} The dimension register \texttt{\cb@curbarwd} is used to store the width of the current bar.

\texttt{\cb@page} \texttt{\cb@pagecount}

\texttt{\cb@barsplace} A switch is provided to control where the changebars will be printed. The value depends on the options given:

\begin{itemize}
  \item 0 for innerbars (default),
  \item 1 for outerbars,
  \item 2 gives leftbars,
  \item 3 gives rightbars.
\end{itemize}

\texttt{\cb@barsplace}\{0\}

\texttt{\@cb@trace} A switch to enable tracing of the actions of this package.

\texttt{\@cb@trace}

\texttt{\@cb@firstcolumn} A switch to find out if a point is in the left column of a twocolumn page.

\texttt{\@cb@firstcolumn}
The macro \cb@pdfxy populates the pdf x,y coordinates file. In \pdftex and \xetex mode it writes one line to \.cb2 file which is equivalent to one bar point. The default implementation is a noop. If the \pdftex or \xetex option is given it is redefined.

\def\cb@pdfxy#1#2#3#4#5{}

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.

\newdimen\cb@odd@left
\newdimen\cb@odd@right
\newdimen\cb@even@left
\newdimen\cb@even@right

Since the changebars are drawn with the \texttt{PostScript} command \texttt{lineto} and not as \LaTeX-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.

\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@even@left by \textwidth
\global\advance\cb@even@left by \changebarsep
\global\advance\cb@even@left by 0.5\changebarwidth
\global\cb@odd@right\cb@odd@left
\global\cb@even@right\cb@even@left
\global\cb@odd@right by \textwidth
\global\cb@even@right by \changebarsep
\global\cb@even@right by 0.5\changebarwidth
\global\cb@odd@left=\cb@odd@left
\global\cb@even@left=\cb@even@left
\if@twoside
\global\advance\cb@even@left by \evensidemargin
\global\cb@even@left by \changebarsep
\global\cb@even@left by -0.5\changebarwidth
\global\cb@odd@right by \textwidth
\global\cb@odd@right by \changebarsep
\global\cb@odd@right by 0.5\changebarwidth
\else
\fi
}else

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

\def\cb@positions{%
\global\cb@odd@left=\hoffset
\global\cb@even@left=\cb@odd@left
\global\advance\cb@odd@left by \oddsidemargin
\global\cb@even@left by \textwidth
\global\cb@odd@right\cb@odd@left
\global\cb@even@right\cb@even@left
\global\cb@odd@right by \textwidth
\global\cb@even@right by \changebarsep
\global\cb@odd@right by 0.5\changebarwidth
\global\cb@even@left=\cb@even@left
\global\let\cb@even@left\cb@odd@left
\global\let\cb@even@right\cb@odd@right
\else
\fi

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
}
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\langle dimen \rangle. 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\langle dimen \rangle. 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.

\begin{verbatim}
\catcode'p=12 \catcode't=12 \gdef\cb@removedim#1pt{#1}
\end{verbatim}

### 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
- Textures
- VT\TeX
- PDFT\TeX
- Xe\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 lower-case equivalents.

\begin{verbatim}
\DeclareOption{DVItoLN03}{\global\chardef\cb@driver@setup=0\relax}
\DeclareOption{dvitoln03}{\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}
\DeclareOption{VTeX}{\global\chardef\cb@driver@setup=5\relax}
\DeclareOption{vtx}{\global\chardef\cb@driver@setup=5\relax}
\end{verbatim}
For the `pdftex` option we have to check that the current \LaTeX{} run is using PDFTeX and that PDF output is selected. If it is, we initialize the option and open an additional output file. If not, we ignore the option and issue a warning.

If we cannot support the `pdftex` option.

For the `xetex` option we have to check that the current \LaTeX{} run is using XeTeX. If it is, we initialize the option and open an additional output file. If not, we ignore the option and issue a warning.

Redefine the \texttt{\cb@pdfxy} macro to write point coordinates to the `.cb2` file.
Give a warning if we cannot support the `xetex` option.

```latex
\def\cb@xetexerror{\PackageError{\PackageError}
  \text{changebar}\%
  \text{XeTeX option cannot be used}\%
  \text{You are not using XeLaTeX})}
```

The new features of \LaTeX{} make it possible to implement the \texttt{outerbars} option.

```latex
\DeclareOption{outerbars}{\def\cb@barsplace\texttt{1}}
\DeclareOption{innerbars}{\def\cb@barsplace\texttt{0}}
```

It is also possible to specify that the change bars should \textit{always} be printed on either the left or the right side of the text. For this we have the options \texttt{leftbars} and \texttt{rightbars}. Specifying \textit{either} of these options will overrule a possible \texttt{twoside} option at the document level.

```latex
\DeclareOption{leftbars}{\def\cb@barsplace\texttt{2}}
\DeclareOption{rightbars}{\def\cb@barsplace\texttt{3}}
```

A set of options to control tracing.

```latex
\DeclareOption{traceon}{\@cb@tracetrue}
\DeclareOption{traceoff}{\@cb@tracefalse}
\DeclareOption{tracestacks}{\let\cb@trace@stack\cb@@show@stack
\def\cb@trace@push#1{\cb@trace{Pushed point \texttt{\textbackslash the@cb@topleft} space on \texttt{\noexpand\#1}: #1}}
\def\cb@trace@pop#1{\cb@trace{Popped point \texttt{\textbackslash the@cb@topleft} space from \texttt{\noexpand\#1}: #1}}
}
```

Three options are introduced for colour support. The first one, \texttt{grey}, is activated by default.

```latex
\DeclareOption{grey}{\%\def\cb@ps@color{\thechangebargrey space 100 div setgray}}
```

The second option activates support for the \texttt{color} package.

```latex
\DeclareOption{color}{\%\def\cb@ps@color{\expandafter\c@lor@to@ps\cb@current@color \@@}}
```

The third option adds support for the \texttt{xcolor} package.

```latex
\DeclareOption{xcolor}{\%\def\cb@ps@color{\expandafter\c@lor@to@ps\cb@current@color \@@}}
```

Signal an error if an unknown option was specified.

```latex
\DeclareOption*{\OptionNotUsed}{\PackageError{\PackageError}
\text{Unrecognised option '{\CurrentOption}\MessageBreak
\text{known options are dvitoln03, dvitops, dvips,}\MessageBreak
\text{extex, textures, pdftex, vtex and xetex,}\MessageBreak
\text{grey, color, xcolor,}\MessageBreak
\text{outerbars, innerbars, leftbars and rightbars}}}
The default is to have grey change bars on the left side of the text on odd pages. When \texttt{VTeX} is used the option \texttt{dvips} is not the right one, so in that case we have \texttt{vtx} as the default driver. When \texttt{PDFTeX} is producing PDF output, the \texttt{pdftex} option is selected.

```latex
\textbf{\cb@@show@stack} When the stack tracing facility is turned on this command is executed. It needs to be defined \textit{before} we call \texttt{ProcessOptions}. This command shows the contents of the stack with currently ‘open’ bars, the stack with pending ends and the history stack. It does \textit{not} show the temporary stack.

\begin{verbatim}
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
  }}
\end{verbatim}
```

The default is to \textit{not} trace the stacks. This is achieved by \texttt{letting} \texttt{cb@trace@stack} to \texttt{@gobble}.

```
\textbf{\cb@trace@push} When stack tracing is turned on, these macros are used to display the push and pop operations that go on. They are defined when the package option \texttt{tracestacks} is selected.

\textbf{\cb@trace@pop} The default is to \textit{not} trace the stacks.
```

\begin{verbatim}
\let\cb@trace@push@gobble
\let\cb@trace@pop@gobble
\end{verbatim}
Now make all the selected options active, but...

We have to make sure that when the document is being processed by pdfLTEX, while also creating pdf as output, the driver to be used is the pdf driver. Therefore we add an extra check, possibly overriding a dvips option that might still have been in the document.

```
\ifx\pdfsavepos\@undefined
  \else
    \ifx\pdfoutput\@undefined
      \else
        \ifnum\pdfoutput>0
          \global\chardef\cb@driver@setup=6\relax
        \fi
    \fi
  \fi
\fi
```

\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
- TPiTEx
- VTeX
- PDFTPeX
- XeTPeX

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.
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.

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

\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.

\cb@trace@connect When tracing is on, write information about the points being connected to the log file.
The macro \texttt{\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: \texttt{\cb@defpoint{(number)\{length\}}.}

The macro \texttt{\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.

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

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

The first extension to the changebar package was for the \texttt{\special} syntax of the program DVItoPS by James Clark.
The program DVIps by Thomas Rokicki is also supported. The PostScript code is
nearly the same as for DVItoPS, but the coordinate space has a different dimen-
sion. 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 following 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 validated 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.
\special{postscript 0 0 transform}% leave [x,y] on the stack
\special{rawpostscript
\expandafter\cb@removedim\the##2\space
/CBarX\the##1\space exch def
itransform exch pop
/CBarY\the##1\space exch def}%
\if@cb@trace\cb@trace@defpoint##1##2\fi}
def\cb@connect##1##2##3{%
\special{rawpostscript
gsave 1 setlinewidth \cb@ps@color
\expandafter\cb@removedim\the##3\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}%
\if@cb@trace\cb@trace@connect##1##2##3\fi
\let\cb@resetpoints\relax

The following definitions were kindly provided by Michael Vulis.
or
def\cb@defpoint##1##2{%
\special{ps:
\expandafter\cb@removedim\the##2\space
Resolution space mul 72.27 space div space
/CBarX\the##1\space exch def currentpoint exch pop
/CBarY\the##1\space exch def}%
\cb@trace@defpoint##1##2}
def\cb@connect##1##2##3{%
\special{ps:
gsave \cb@ps@color
\expandafter\cb@removedim\the##3\space
Resolution space mul 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
stroke grestore}%
\cb@trace@connect##1##2##3}
\let\cb@resetpoints\relax

The code for PDFTEX is more elaborate as the calculations have to be done in \TeX. \cb@defpoint will write information about the coordinates of the point to the .aux file, from where it will be picked up in the next run. Then we will construct the PDF code necessary to draw the changebars.
or
\immediate\closeout\cb@writexy
\immediate\openin\cb@readxy=\jobname.cb2
\cb@pdfpoints
\cb@pdfpagenr

The \cb@pdfpoints macro contains the list of coordinates of points that have been read in memory from the .cb2 file. The \cb@pdfpagenr macro contains the next pagecount to be read in.
\def\cb@pdfpoints{}
\def\cb@pdfpagenr{0}

The `\cb@findpdfpoint` macro finds the coordinates of point #1 on pagecount #2. First we expand the arguments to get the real values.

\def\cb@findpdfpoint##1##2{\edef\cb@temp{\noexpand\cb@@findpdfpoint{\the##1}{\the##2}}}\cb@temp

\cb@@findpdfpoint

The `\cb@@findpdfpoint` macro finds the coordinates of point #1 on pagecount #2. If the information is not yet in memory is it read from the .cb2 file. The coordinates of the current point in the text will be delivered in `\cb@pdfx` and `\cb@pdfy`, and `\cb@pdfz` will get the x coordinate of the changebar. If the point is unknown, `\cb@pdfx` will be set to \relax.

\def\cb@@findpdfpoint##1##2{\ifnum##2<\cb@pdfpagenr\relax\else\cb@pdfreadxy{##2}\fi\let\cb@pdfx\relax\ifx\cb@pdfpoints\@empty\else\ifnum##2<0\relax\else\edef\cb@temp{\noexpand\cb@pdffind{##1}{##2}\cb@pdfpoints{\relax{}}}\cb@temp\fi\fi\fi}

\cb@pdffind

The `\cb@pdffind` recursively searches through `\cb@pdfpoints` to find point #1 on pagecount #2. \cb@pdfpoints contains entries of the form ⟨pointnr⟩⟨pagecount⟩p⟨x⟩⟨y⟩⟨z⟩pt. When the point is found it is removed from \cb@pdfpoints. #9 contains the cumulative head of the list to construct the new list with the entry removed. #3–#8 are for pattern matching.

\def\cb@pdffind##1##2##3.##4p##5,##6,##7pt##8\relax##9{\def\cb@next{\cb@pdffind{##1}{##2}##8\relax{##9##3.##4p##5,##6,##7pt}}\ifnum##1=##3\ifnum##2=##4\def\cb@pdfx{##5sp}\def\cb@pdfy{##6sp}\def\cb@pdfz{##7pt}\let\cb@next\relax\gdef\cb@pdfpoints{##9##8}\fi\fi\ifx\relax##8\relax\let\cb@next\relax\fi}

\cb@pdffind
The \cb@pdfreadxy macro reads lines from the .cb file in \cb@pdfpoints until the pagecount is greater than \#1 or the end of the file is reached. This ensures that all entries belonging to the current column are in memory.

\def\cb@pdfreadxy##1{%
  \let\cb@next\relax
  \ifeof\cb@readxy
    \global\let\cb@pdfpagenr\cb@maxpoint
  \else
    \{\endlinechar=-1\read\cb@readxy to\cb@temp
    \ifex\cb@temp\@empty\else
      \expandafter\cb@pdfparsexy\cb@temp
      \ifnum\cb@pdfpg<0\else
        \xdef\cb@pdfpoints{\cb@pdfpoints\cb@temp}\
        \cb@trace{PDFpoints=\cb@pdfpoints}\% 
      \fi
    \fi
  \fi
  \fi
  \cb@next
}%

The \cb@pdfparsexy macro extracts the pagecount from an entry read in from the .cb file.

\def\cb@pdfparsexy##1.##2p##3,##4,##5pt{% 
  \def\cb@pdfpg{##2}%
}

As PDF is not a programming language it does not have any variables to remember the coordinates of the current point. Therefore we write the information to the .aux file and read it in in the next run. We write the x,y coordinates of the current point in the text and the x coordinate of the change bar. We also need the value of \cb@pagecount here, not during the write.

\def\cb@defpoint##1##2{% 
  \if@filesw
    \begingroup
      \edef\point{{\the##1}{\the\cb@pagecount}}\
      \let\the=\z@
      \pdfsavepos
      \edef\cb@temp{\write\@auxout
        \string\cb@pdfxy\point
        {\the\pdflastxpos}{\the\pdflastypos}{\the##2} }\% 
  \cb@temp
  \endgroup
%
The macro \texttt{\cb@cvtpct} converts a percentage between 0 and 100 to a decimal fraction.

\begin{verbatim}
def\cb@cvtpct##1{\ifnum##1<0 0\else\ifnum##1>99 1\else\ifnum##1<10 0.0\the##1\else 0.\the##1\fi\fi\fi}
\end{verbatim}

The \texttt{\cb@connect} finds the coordinates of the begin and end points, converts them to PDF units and draws the bar with \texttt{pdfliteral}. It also sets the color or gray level, if necessary. When any of the points is unknown the bar is skipped and a rerun is signalled.

\begin{verbatim}
def\cb@connect##1##2##3{\cb@findpdfpoint{##1}\cb@pagecount\if\cb@pdfx\relax\cb@rerun\else\let\cb@pdftopy\cb@pdfy\cb@findpdfpoint{##2}\cb@pagecount\if\cb@pdfx\relax\cb@rerun\else\begin{group}
\cb@dima=\cb@pdfz\advance\cb@dima by-\cb@pdfx\advance\cb@dima by1in\cb@dima=0.996264009963\cb@dima\relax\expandafter\cb@removedim\the\cb@dima w \cb@temp 0 m \cb@temp \expandafter\cb@removedim\the\cb@dima l S Q\end{group}\pdfliteral direct{\expandafter\cb@removedim\the\cb@dima \cb@current@color\@undefined\def\cb@temp{\cb@cvtpct\cb@changebargrey} p {\cb@temp g \cb@temp G} q\pdfliteral{q \cb@temp\space g \cb@temp\space G}}\else\pdfliteral{q \cb@current@color}\fi\edef\cb@temp{\expandafter\cb@removedim\the##3 w \cb@temp 0 m \cb@temp \expandafter\cb@removedim\the\cb@dima l S Q}\fi\fi}
\end{verbatim}

We do everything in a group, so that we can freely use all kinds of registers.

First we let PDF save the graphics state. Then we generate the color selection code followed by the code to draw the changebar. Finally the graphics state is restored. We cannot use the color commands from the color package here, as the generated PDF code may be moved to the next line.

\begin{verbatim}
\if\cb@current@color\@undefined\def\cb@temp{\cb@cvtpct\cb@changebargrey}\pdfliteral{q \cb@temp\space g \cb@temp\space G}}\else\pdfliteral{q \cb@current@color}\fi\edef\cb@temp{\expandafter\cb@removedim\the##3 w \cb@temp 0 m \cb@temp \expandafter\cb@removedim\the\cb@dima l S Q}\fi\fi
\end{verbatim}

20
We look up the two unused points to get them removed from \texttt{\cb@pdfpoints}.

\begin{Verbatim}
\cb@cntb=##1\relax
\ifodd\cb@cntb\advance\cb@cntb 1\else\advance\cb@cntb -1\fi
\cb@findpdfpoint\cb@cntb\cb@pagecount
\cb@cntb=##2\relax
\ifodd\cb@cntb\advance\cb@cntb 1\else\advance\cb@cntb -1\fi
\cb@findpdfpoint\cb@cntb\cb@pagecount
\fi
\fi
\cb@trace@connect##1##2##3%
\end{Verbatim}

\texttt{\cb@checkPdfxy} The macro \texttt{\cb@checkPdfxy} checks if the coordinates of a point have changed during the current run. If so, we need to rerun \TeX.

\begin{Verbatim}
\gdef\cb@checkPdfxy##1##2##3##4##5{%
\cb@@findpdfpoint{##1}{##2}%
% \end{macrocode}
%\begin{changebar}
% \begin{macrocode}
  \ifdim##3sp=\cb@pdfx\relax
  \ifdim##4sp=\cb@pdfy\relax
  % \end{macrocode}
% \end{changebar}
% \begin{macrocode}
  \ifdim##5=\cb@pdfz\relax
  \else
  \cb@error
  \fi
  \else
  \cb@error
  \fi
  \else
  \cb@error
  \fi
  \fi
}\
\end{Verbatim}

For \TeX we don’t need a limit on the number of bar points.

\begin{Verbatim}
\def\cb@maxpoint{9999999}
\let\cb@resetpoints\relax
\or
\cb@defpoint
\end{Verbatim}

The code for \Xe\TeX is, like for \TeX, more elaborate as the calculations have to be done in \TeX. \texttt{\cb@defpoint} will write information about the coordinates of the point to the \jobname\.aux file, from where it will be picked up in the next run. Then we will construct the PDF code necessary to draw the changebars.

\begin{Verbatim}
\immediate\closeout\cb@writexy
\immediate\openin\cb@readxy=\jobname.cb2\relax
\end{Verbatim}
\texttt{\cb@pdfpoints} The \texttt{\cb@pdfpoints} macro contains the list of coordinates of points that have been read in memory from the \texttt{.cb2} file. The \texttt{\cb@pdfpagenr} macro contains the next pagecount to be read in.

449 \texttt{\def\cb@pdfpoints{}}
450 \texttt{\def\cb@pdfpagenr{0}}

\texttt{\cb@findpdfpoint} The \texttt{\cb@findpdfpoint} macro finds the coordinates of point \#1 on pagecount \#2. First we expand the arguments to get the real values.

451 \texttt{\def\cb@findpdfpoint##1##2{}}
452 \texttt{\edef\cb@temp{\noexpand\cb@@findpdfpoint{\the##1}{\the##2}}}%
453 \texttt{\cb@temp}
454 \texttt{\cb@temp}
455 }

\texttt{\pdfliteral} For Xe\TeX\ we mimick PDFTeX's command \texttt{\pdfliteral}.

456 \texttt{\def\pdfliteral##1{\special{pdf:literal ##1}}}

\texttt{\cb@@findpdfpoint} The \texttt{\cb@@findpdfpoint} macro finds the coordinates of point \#1 on pagecount \#2. If the information is not yet in memory is it read from the \texttt{.cb2} file. The coordinates of the current point in the text will be delivered in \texttt{\cb@pdfx} and \texttt{\cb@pdfy}, and \texttt{\cb@pdfz} will get the x coordinate of the changebar. If the point is unknown, \texttt{\cb@pdfx} will be set to \texttt{\relax}.

457 \texttt{\def\cb@@findpdfpoint##1##2{}}
458 \ifnum##2<\cb@pdfpagenr\relax
459 \texttt{\cb@pdfreadxy{##2}}%
460 \fi
461 \texttt{\let\cb@pdfx\relax}
462 \texttt{\ifx\cb@pdfpoints\empty\else}
463 \texttt{\ifnum##2<\relax}
464 \else
465 \texttt{\edef\cb@temp{\noexpand\cb@pdffind{##1}{##2}\cb@pdfpoints\relax{}{}}}%
466 \texttt{\cb@temp}
467 \texttt{\fi}
468 \fi
469 }

\texttt{\cb@pdffind} The \texttt{\cb@pdffind} recursively searches through \texttt{\cb@pdfpoints} to find point \#1 on pagecount \#2. \texttt{\cb@pdfpoints} contains entries of the form \texttt{⟨pointer⟩,⟨pagecount⟩p⟨x⟩,⟨y⟩,⟨z⟩pt}. When the point is found it is removed from \texttt{\cb@pdfpoints}. \#9 contains the cumulative head of the list to construct the new list with the entry removed. \#3–\#8 are for pattern matching.

470 \texttt{\def\cb@pdffind##1##2#3.#4#5,#6,#7pt##8\relax##9{}}
471 \texttt{\def\cb@next{\cb@pdffind{##1}{##2}{##9}\relax{##3}{##4#5}{##6,#7pt}}}%
472 \texttt{\ifnum##1=##3}
473 \texttt{\ifnum##2=##4}
474 \texttt{\cb@pdfx{##5sp}}%
475 \texttt{\cb@pdfy{##6sp}}%
476 \texttt{\cb@pdfz{##7pt}}%

22
The \texttt{\cb@pdfreadxy} macro reads lines from the .\texttt{cb2} file in \texttt{\cb@pdfpoints} until the pagecount is greater than \#1 or the end of the file is reached. This ensures that all entries belonging to the current column are in memory.

As PDF is not a programming language it does not have any variables to remember the coordinates of the current point. Therefore we write the information to the .\texttt{aux} file and read it in in the next run. We write the x,y coordinates of the current point in the text and the x coordinate of the change bar. We also need the value of \texttt{\cb@pagecount} here, not during the write.
\let\the=z@
\pdfsavepos
\edef\cb@temp{\write\@auxout
{\string\cb@pdfxy\point
{\the\pdflastxpos}{\the\pdflastypos}{\the##2}}%}
\cb@temp
\endgroup
\fi
\cb@trace@defpoint##1##2%
}

\cb@cvtpct 

The macro \cb@cvtpct converts a percentage between 0 and 100 to a decimal fraction.

\def\cb@cvtpct##1{\ifnum##1<0 0\else\ifnum##1>99 1\else\ifnum##1<10 0.0\the##1\else0.\the##1\fi\fi\fi}

\cb@pdf@scale

In order to get things in the right spot we need a little scaling factor. We define it here.

\def\cb@pdf@scale{0.996264009963}

\cb@connect

The \cb@connect finds the coordinates of the begin and end points, converts them to PDF units and draws the bar with \pdfliteral. It also sets the color or gray level, if necessary. When any of the points is unknown the bar is skipped and a rerun is signalled.

\def\cb@connect##1##2##3{\cb@findpdfpoint{##1}\cb@pagecount\ifx\cb@pdfx\relax\cb@rerun\else\let\cb@pdftopy\cb@pdfy\cb@findpdfpoint{##2}\cb@pagecount\ifx\cb@pdfx\relax\cb@rerun\else

We do everything in a group, so that we can freely use all kinds of registers.

\begin{group}
\cb@dima=\cb@pdfz
\advance\cb@dima by-\cb@pdfx
\advance\cb@dima bylin
\cb@dima=\cb@pdf@scale\cb@dima\relax

First we let PDF save the graphics state. Then we generate the color selection code followed by the code to draw the changebar. Finally the graphics state is restored. We cannot use the color commands from the color package here, as the generated PDF code may be moved to the next line.

\ifx\cb@current@color\@undefined
\def\cb@temp{\cb@cvtpct\c@changebargrey}%
We look up the two unused points to get them removed from `\cb@pdfpoints`.

The macro `\cb@checkPdfxy` checks if the coordinates of a point have changed during the current run. If so, we need to rerun \LaTeXX.

```latex
\cb@checkPdfxy
```
For Xe\TeX{} we don’t need a limit on the number of bar points.
\begin{verbatim}
\def\cb@maxpoint{9999999}
\let\cb@resetpoints\relax
\end{verbatim}

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.
\begin{verbatim}
\else
\PackageWarning{Changebar}{changebars not supported in unknown setup}
\def\cb@defpoint##1##2{\cb@trace@defpoint##1##2}
\def\cb@connect##1##2##3{\cb@trace@connect##1##2##3}
\let\cb@resetpoints\relax
\fi
\end{verbatim}

The last thing to do is to forget about \cb@setup@specials.
\begin{verbatim}
\global\let\cb@setup@specials\relax
\cbstart
\end{verbatim}

\texttt{\cbstart} The macro \texttt{\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
\texttt{\changebarwidth}.
\begin{verbatim}
\newcommand*{\cbstart}{\@ifnextchar \[%\]{\cb@start}%%
{\cb@start[\changebarwidth]}}
\end{verbatim}

\texttt{\cbend} The macro \texttt{\cbend} (surprisingly) ends a changebar. The macros \texttt{\cbstart} and
\texttt{\cbend} can be used when the use of a proper \LaTeX{} environment is not possible.
\begin{verbatim}
\newcommand*{\cbend}{\cb@end}
\end{verbatim}

\texttt{\cbdelete} The macro \texttt{\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 \texttt{\deletebarwidth}.
\begin{verbatim}
\newcommand*{\cbdelete}{\@ifnextchar \[%\]{\cb@delete}%%
{\cb@delete[\deletebarwidth]}}
\end{verbatim}

\texttt{\cb@delete} Deletebars are implemented as a special ‘change bar’. The bar is started and
immediately ended. It is as long as it is wide.
\begin{verbatim}
\def\cb@delete[#1]{\vbox to \z@{\vss\cb@start[#1]\vskip #1\cb@end}}
\end{verbatim}

\texttt{\changebar} The macros \texttt{\changebar} and \texttt{\endchangebar} have the same function as \texttt{\cbstart}
and \texttt{\cbend} but they can be used as a \LaTeX{} environment to enforce correct
nesting. They can not be used in the \texttt{tabular} and \texttt{tabbing} environments.
\begin{verbatim}
\newenvironment{changebar}{\@ifnextchar \[%\]{\cb@start}%%
{\cb@start[\changebarwidth]}}%
\end{verbatim}
\nochangebars 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.

\newcommand*{\nochangebars}{%
  \def\cb@start[#1]{}%
  \def\cb@delete[#1]{}%
  \let\cb@end\relax%
%
}\changebarwidth The default width of the changebars is stored in the dimension register \changebarwidth.

\newlength{\changebarwidth}
\setlength{\changebarwidth}{2pt}

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

\newlength{\deletebarwidth}
\setlength{\deletebarwidth}{4pt}

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

\newlength{\changebarsep}
\setlength{\changebarsep}{0.5\marginparsep}

\changebargrey 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).

\newcounter{changebargrey}
\setcounter{changebargrey}{65}

\cbcolor \cbcolor{\declared-colour} switches the colour of the changebars to \declared-colour, which must previously have been defined using \definecolor. This colour will stay in effect until the end of the current \TeX group.

\cbcolor{\model}{\colour-specification} is similar to the above, but uses a colour not declared by \definecolor. The allowed \model’s vary depending on the driver. The syntax of the \colour-specification argument depends on the model.

\DeclareRobustCommand{\cbcolor}{%}
\cbcolor{\declared-cbcolor}{%}
\cbcolor{\undeclared-cbcolor}{\declared-cbcolor}
\@undeclaredcbcolor Call the driver-dependent command \color@{}(model) to define \cb@current@color.

\def\@undeclaredcbcolor[#1]{%  
\begingroup  
\color[#1]{#2}  
\global\let\cb@current@color\current@color  
\endgroup  
\ignorespaces  
}  

5.4 Macros for beginning and ending bars

\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  
\cb@currentstack and then sets the point by calling \cb@setBeginPoints with  
the point number. Finally, it writes the .aux file.

\def\cb@start[#1]{%  
\cb@topleft=\cb@nextpoint  
\cb@curbarwd=#1\relax  
\cb@push\cb@currentstack  
\cb@checkpage  
\cb@push\cb@currentstack  
\cb@checkpage\z@  
Temporarily assign the page number to \cb@pagecount as that register is used by  
\cb@setBeginPoints. Note that it’s value is offset by one from the page counter.

\cb@cnta\cb@pagecount

28
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}

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 beginning point number. It writes the .aux file and joins the points. When in horizontal mode we put the call to \texttt{cb@setEndPoints} inside a \texttt{vadjust}. This ensures that things with a large depth, e.g. a parbox or formula will be completely covered. By default these have their baseline centered, and thus otherwise the changebar would stop there.

\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}{Badly nested changebars; Expect erroneous results}\else Call \texttt{cb@checkpage} to find the page this point finally ends up on.\fi\cb@checkpage\thr@@\cb@cnta\cb@pagecount\cb@page\advance\cb@pagecount\m@ne\ifvmode\cb@setEndPoints\else\vadjust{\cb@setEndPoints}\fi}
\end{verbatim}

Again, we need to temporarily overwrite \texttt{cb@pagecount}.

\begin{verbatim}
\cb@cnta\cb@pagecount\cb@pagecount\cb@page\advance\cb@pagecount\m@ne\ifvmode\cb@setEndPoints\else\vadjust{\cb@setEndPoints}\fi
\end{verbatim}
The macro \texttt{\textbackslash 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 \texttt{\textbackslash cb@topleft} and \texttt{\textbackslash cb@page}. The resulting page will be stored in \texttt{\textbackslash cb@page}. The parameter indicates whether we are searching for a begin point (0) or end point (3).

\begin{verbatim}
\def\cb@checkpage#1{\relax
First store the identifiers in temporary registers.
\cb@cnta\cb@topleft\relax
\advance\cb@cnta by #1\relax
\cb@cntb\cb@page\relax
\cb@dima\cb@curbarwd\relax
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 \texttt{\textbackslash cb@topleft} is found. 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.
\cb@FindPageNum
\ifnum\cb@topleft>\cb@maxpoint\else
Now that we've found it overwrite \texttt{\textbackslash cb@cntb} with the \texttt{\textbackslash cb@page} from the stack.
\cb@cntb\cb@page
\fi
Now we restore the history stack to it's original state.
\@whilenum\cb@topleft>\cb@nil\do{%}
\cb@push\cb@historystack
\cb@pop\cb@tempstack}
\fi
Finally return the correct values
\advance\cb@cnta by -#1\relax
\cb@topleft\cb@cnta\relax
\cb@page\cb@cntb\relax
\cb@curbarwd\cb@dima\relax
)}
\end{verbatim}

\texttt{\textbackslash cb@FindPageNum} recursively searches through the history stack until an entry is found that is equal to \texttt{\textbackslash cb@cnta}.
We have found it, exit the macro, otherwise push the current entry on the temporary stack and pop a new one from the history stack.

\else
\cb@push\cb@tempstack
\cb@pop\cb@historystack

When the user adds changebars 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
\else
Recursively call ourselves.
\expandafter\expandafter\expandafter\cb@FindPageNum
\fi
\fi
}\cb@setBeginPoints

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 \c@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@setBeginPoints{%
\cb@topright=\cb@topleft\advance\cb@topright by\@ne
\cb@cntb=\cb@pagecount
\divide\cb@cntb by\tw@
\ifodd\cb@cntb
\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
}\cb@setEndPoints

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.
\def\cb@setEndPoints{%
\cb@topright=\cb@topleft\advance\cb@topright by\@ne
\cb@botleft=\cb@topleft\advance\cb@botleft by\tw@
\cb@botright=\cb@topleft\advance\cb@botright by\thr@@
\cb@cntb=\cb@pagecount
\divide\cb@cntb by\tw@
\ifodd\cb@cntb
\cb@defpoint\cb@botleft\cb@even@left
\cb@defpoint\cb@botright\cb@even@right
\else
\cb@defpoint\cb@botleft\cb@odd@left
\cb@defpoint\cb@botright\cb@odd@right
\fi
\cb@writeAux\cb@botright
\edef\cb@leftbar{\noexpand\cb@connect\cb@topleft\cb@botleft\cb@curbarwd}%
\edef\cb@rightbar{\noexpand\cb@connect\cb@topright\cb@botright\cb@curbarwd}%
}

In twocolumn pages always use outerbars
\if@twocolumn
\ifodd\cb@pagecount\cb@rightbar\else\cb@leftbar\fi
\else
\ifcase\cb@barsplace
0=innerbars
\ifodd\cb@cntb
\cb@rightbar
\else
\if@twoside\cb@leftbar\else\cb@rightbar\fi
\fi
\or
1=outerbars
\ifodd\cb@cntb
\cb@leftbar
\else
\if@twoside\cb@rightbar\else\cb@leftbar\fi
\fi
\or
2=leftbars
\cb@leftbar
\or
3=rightbars
\cb@rightbar
\fi
\fi
}
\cb@WriteAux  The macro \cb@WriteAux writes information about a changebar point to the auxiliary file. The number of the point, the pagernumber and the width of the bar are written out as arguments to \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 \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.

\begin{verbatim}
775 \def\cb@WriteAux#1{%
776 \if@filesw
777 \begingroup
778 \edef\point{\the#1}%
779 \edef\level{\the\cb@curbarwd}%
780 \let\the=\z@%
781 \edef\cb@temp{\write\@auxout
782 {\string\cb@barpoint{\point}{\the\cb@pagecount}{\level}}}%
783 \cb@temp
784 \endgroup
785 \fi}
\end{verbatim}

5.5 Macros for Making It Work Across Page Breaks

\cb@pagejump  A switch to indicate that we have made a page correction.

\begin{verbatim}
786 \newif\if\cb@pagejump
\end{verbatim}

\cb@pagejumplist  The list of pagecounts to be corrected.

\begin{verbatim}
787 \def\cb@pagejumplist{-1}
\end{verbatim}

\cb@nextpagejump  The next pagecount from the list.

\begin{verbatim}
788 \def\cb@nextpagejump{-1}
\end{verbatim}

\cb@pagejump  This macro is written to the .aux file when a pagecount in a lefthand column should be corrected. The argument is the incorrect pagecount.

\begin{verbatim}
789 \def\cb@pagejump#1{\xdef\cb@pagejumplist{\cb@pagejumplist,#1}}
\end{verbatim}

\cb@writepagejump  This macro writes a \cb@pagejump entry to the .aux file. It does it by putting the \write command in the \@leftcolumn so that it will be properly positioned relative to the bar points.

\begin{verbatim}
790 \def\cb@writepagejump#1{\{
791 \cb@cntb=\cb@pagecount
792 \advance\cb@cntb by#1\relax
793 \global\setbox\@leftcolumn\vbox to\@colht{%
794 \edef\cb@temp{\write\@auxout{\string\cb@pagejump{\the\cb@cntb}}}%
795 \cb@temp
796 \vskip \dp\@leftcolumn
797 \unvbox \@leftcolumn
798 \vskip \dimen0
799 \}}%
800 }
\end{verbatim}
\cb@poppagejump Pop an entry from pagejumplst. The entry is put in \cb@nextpagejump.
\begin{verbatim}
def\cb@poppagejump#1,#2\relax{\
gdef\cb@nextpagejump{#1}\
gdef\cb@pagejumplst{#2}}
\end{verbatim}

\cb@checkpagecount This macro checks that \cb@pagecount is correct at the beginning of a column or page. First we ensure that \cb@pagecount has the proper parity: odd in the righthand column of a twocolumn page, even in the lefthand column of a twocolumn page and in onecolumn pages.
\begin{verbatim}
def\cb@checkpagecount{%\if@twocolumn\if@firstcolumn\ifodd\cb@pagecount\global\advance\cb@pagecount by\@ne\fi\else\ifodd\cb@pagecount\global\advance\cb@pagecount by\@ne\fi\fi\else\ifodd\cb@pagecount\global\advance\cb@pagecount by\@ne\fi\fi\end{verbatim}

Also, in twosided documents, \cb@pagecount/2 must be odd on even pages and even on odd pages. If necessary, increase \cb@pagecount by 2. For onesided documents, we don’t do this as it doesn’t matter (but it would be harmless). In the righthand column in twoside documents we must check if \cb@pagecount/2 has the proper parity (see below). If it is incorrect, the page number has changed after the lefthand column, so \cb@pagecount is incorrect there. Therefore we write a command in the .aux file so that in the next run the lefthand column will correct its \cb@pagecount. We also need to signal a rerun. If the correction was made in the lefthand column, the flag \cb@pagejump is set, and we have to be careful in the righthand column. If in the righthand column the flag is set and \cb@pagecount is correct, the correction in the lefthand column worked, but we still have to write into the .aux file for the next run. If on the other hand \cb@pagecount is incorrect while the flag is set, apparently the correction in the lefthand column should not have been done (probably because the document has changed), so we do nothing.
\begin{verbatim}
@if@twoside\cb@cntb=\cb@pagecount\divide\cb@cntb by\tw@\advance\cb@cntb by-\c@page\ifodd\cb@cntb\cb@trace{Page jump: %\cb@pagecount\the\cb@pagecount}\global\advance\cb@pagecount by\tw@\global\@cb@pagejumptrue\else\fi\fi\end{verbatim}

Here \cb@pagecount seems correct. Check if there is a page jump.
\begin{verbatim}
@if@twocolumn\if@firstcolumn\@whilenum\cb@pagecount>\cb@nextpagejump\do{%\expandafter\cb@poppagejump\cb@pagejumplst\relax}%\ifnum\cb@pagecount=\cb@nextpagejump\cb@trace{Page jump: string\cb@pagecount=\the\cb@pagecount}\global\advance\cb@pagecount by\tw@\global\@cb@pagejumptrue\else\fi\end{verbatim}
In the righthand column check the flag (see above). If set, write a pagejump, but compensate for the increase done in the lefthand column.

Here \cb@pagecount is incorrect.

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 box \texttt{\@cclv}. The bars carried over from the previous page, and hence to be restarted on this page, have been saved on the stack \texttt{\cb@beginstack}. 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{\cb@endstack} is built and processed by \texttt{\cb@processActive}. Finally the original \texttt{\@makecol} (saved as \texttt{\cb@makecol}) is executed.

\let\ltx@makecol\@makecol
\def\cb@makecol{%
  \if@twocolumn
    \if@firstcolumn
      \global\advance\cb@pagecount by\tw@
      \global\cb@pagejumpfalse
    \else
      \cf@trace{Page jump annulled, \string\cb@pagecount=\the\cb@pagecount}
      \else
      \fi
    \fi
  \else
    \fi
  \fi
}%
\let\ltx@makecol\@makecol
\def\cb@makecol{%
  \if@twocolumn
    \if@firstcolumn
      \global\advance\cb@pagecount by\tw@
      \global\cb@pagejumpfalse
    \else
      \cf@trace{Page jump annulled, \string\cb@pagecount=\the\cb@pagecount}
      \else
      \fi
    \fi
  \else
    \fi
  \fi
}
First make sure that \cb@pagecount is correct. Then add the necessary bar points at beginning and end.
\cb@checkpagecount
\setbox\@cclv \vbox{\cb@resetpoints
\cb@startSpanBars
\unvbox\@cclv
\boxmaxdepth\maxdepth}\
\global\advance\cb@pagecount by\@ne
\cb@buildstack\cb@processActive
\ltx@makecol
In twocolumn pages write information to the aux file to indicate which column we are in. This write must precede the whole column, including floats. Therefore we insert it in the front of \@outputbox.
\if@twocolumn
\global\setbox\@outputbox \vbox to\@colht {\if@firstcolumn\write\@auxout{\string\@cb@firstcolumntrue}%
\else\write\@auxout{\string\@cb@firstcolumnfalse}%
\fi
\unvbox\@outputbox
\vskip-\dimen@}
\fi
\cb@trace@stack{after makecol, page \the\c@page,}
\string\cb@pagecount=\the\cb@pagecount}%
\let\@makecol\cb@makecol
When LATEX makes a page with only floats it doesn’t use \@makecol; instead it calls \@vtryfc, so we have to modify this macro as well. In twocolumn mode we must write either \@cb@firstcolumntrue or \@cb@firstcolumnfalse to the .aux file.
\def\cb@vtryfc#1{%
\cb@trace{In vtryfc, page \the\c@page,}
\string\cb@pagecount=\the\cb@pagecount}%
\let\cb@writeAux\@gobble
First make sure that \cb@pagecount is correct. Then generate a \@cb@firstcolumntrue or \@cb@firstcolumnfalse in twocolumn mode.
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 \cb@startSaves
push active point back onto history stack (need to reprocess on next page).

\cb@startSpanBars This macro defines new points for each bar that was pushed on the \cb@beginstack. Afterwards \cb@beginstack is empty.

\cb@processActive{\cb@pop\cb@endstack\ifnum\cb@topleft=\cb@nil\else\setbox\cclv\vbox{\unvbox\cclv\boxmaxdepth\maxdepth\advance\cb@pagecount by -1\relax\cb@setEndPoints}\cb@push\cb@historystack\cb@push\cb@beginstack\expandafter\cb@processActive\fi}
The macro \texttt{\cb@buildstack} initializes the stack with open bars and starts populating it.

\begin{verbatim}
\def\cb@buildstack{%
  \cb@initstack\cb@endstack
  \cb@pushNextActive}
\end{verbatim}

This macro pops the top element off the history stack (\texttt{\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 (\texttt{\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.

\begin{verbatim}
\def\cb@pushNextActive{%
  \cb@pop\cb@historystack
  \ifnum\cb@topleft=\cb@nil
    \else
      \ifnum\cb@page>\cb@pagecount
        \cb@push\cb@historystack
      \else
        \ifodd\cb@topleft
          \cb@push\cb@endstack
        \else
          \cb@pop\cb@endstack
        \fi
      \expandafter\expandafter\expandafter\cb@pushNextActive
      \fi
    \fi
  \fi}
\end{verbatim}

### 5.6 Macros For Managing The Stacks of Bar points

The macros make use of four stacks corresponding to \texttt{\special defpoints}. Each stack takes the form \texttt{<element> ... <element>}

Each element is of the form xxxyyyyzzz 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 of 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 \cb@historystack.

\cb@initstack \ A macro to (globally) initialize a stack.
945 \def\cb@initstack#1{\xdef#1{}}

\cb@historystack \ We need to initialise a stack to store the entries read from the external history file.
946 \cb@initstack\cb@historystack

\cb@write \ We also need to allocate a read and a write stream for the history file.
947 \newwrite\cb@write
948 \newread\cb@read

\cb@read \ And we open the history file for writing (which is done when the .aux file is read in).
949 \immediate\openout\cb@write=\jobname.cb\relax

\cb@endstack \ Allocate two stacks for the bars that span the current page break.
950 \cb@initstack\cb@endstack

\cb@beginstack \ Allocate a stack for temporary storage
951 \cb@initstack\cb@beginstack

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

\cb@currentstack \ This macro pops the top element off the named stack and puts the point value into \cb@topleft, the page value into \cb@page and the bar width into \cb@curbarwd. If the stack is empty, it returns a void value (\cb@nil) in \cb@topleft and sets \cb@page=0.
954 \def\cb@thehistorystack{\cb@historystack}
955 \def\cb@pop#1{%
956   \ifx #1\@empty
957     \def\cb@temp{#1}%
958     \ifx\cb@temp\cb@thehistorystack
959       \ifeof\cb@read
960         \else
961         \{\endlinechar=-1\read\cb@read to\cb@temp
962         \xdef\cb@historystack{\cb@historystack\cb@temp}%
963         \}%
964       \fi
965     \fi
966   \fi
967   \ifx\@empty
968     \global\cb@topleft\cb@nil

This macro is used to ‘decode’ a stack entry.

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

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

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.

\begin{verbatim}
\AtBeginDocument{\cb@setup@specials}
\cb@pagejump{999999999,}
\cb@positions
\cb@trace{%
  Odd left : \the\cb@odd@left\space
  Odd right : \the\cb@odd@right\MessageBreak
  Even left : \the\cb@even@left\space
  Even right: \the\cb@even@right
  }%
\immediate\closeout\cb@write
\immediate\openin\cb@read=\jobname.cb\relax}
\end{verbatim}
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 \TeX 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.

\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 \TeX 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.

\cb@checkHistory 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@checkPdfxy Dummy definition for `\cb@checkPdfxy`. This will be overwritten by the `pdftex` and `xetex` options.
The macro \texttt{\cb@rerun} is called when we detect that we need to rerun \LaTeX{}.

\begin{verbatim}
\def\cb@rerun{%
  \global\let\cb@checkrerun\cb@error}
\let\cb@checkrerun\relax
\cb@error
\end{verbatim}

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. For pdf\TeX{} and Xe\TeX{} we also disable \texttt{\cb@checkPdfxy}.

\begin{verbatim}
\def\cb@error{%
  \PackageWarning{Changebar}{Changebar info has changed.\MessageBreak}
  \gdef\cb@checkHistory##1##2##3{}%
  \let\cb@barpoint\cb@checkHistory
  \gdef\cb@checkPdfxy##1##2##3##4##5{}%
  \let\cb@pdfxy\cb@checkPdfxy}
\end{verbatim}

5.8 Macros For Making It Work With Nested Floats/Footnotes

\texttt{\end@float} 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{} \texttt{\end@float}.

\begin{verbatim}
\let\ltx@end@float\end@float
\def\cb@end@float{%
  \cb@trace@stack{end float on page \the\c@page}%
  \cb@pop\cb@currentstack
  \ifnum\cb@topleft=\cb@nil
    \else
      \cb@push\cb@currentstack
  \fi
  \global\cb@curbarwd=\cb@curbarwd
  \@endfloatbox
  \global\setbox\@currbox
  \color@vbox
  \normalcolor
  \vbox\bgroup\cb@start[\cb@curbarwd]\unvbox\@currbox\cb@end

\texttt{\end@float}
\end{verbatim}

This only works if this new version of \texttt{\end@float} is really used. With \LaTeX{}2.09 the document styles used to contain:

\begin{verbatim}
  \let\endfigure\end@float
\end{verbatim}

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

42
\@xympar  There is one snag with this redefinition in that the macro \end@float is also used by the command \marginpar. This may lead to problems with stack underflow. Therefore we need to redefine an internal macro from the marginal paragraph mechanism as well. The solution is to make sure this macro uses the original definition of \end@float.

\begin{verbatim}
1055 \let\ltx@@xympar\@xympar
1056 \def\@xympar{% 
1057 \let@end@float\ltx@end@float 
1058 \ltx@@xympar 
1059 \let@end@float\cb@end@float}
\end{verbatim}

\@float@end  When the float package is being used we need to take care of its changes to the float mechanism. It defines its own macros (\float@end and \float@dblend which need to be modified for changebars to work.

First we'll save the original as \flt@float@end.

\begin{verbatim}
1060 \let\flt@float@end\float@end
1061 \def\float@end{% 
1062 \cb@trace@stack{end float on page \the\c@page}\
1063 \cb@pop\cb@currentstack 
1064 \ifnum\cb@topleft=\cb@nil 
1065 \else 
1066 \cb@push\cb@currentstack 
1067 \global\cb@curbarwd=\cb@curbarwd 
1068 \@endfloatbox 
1069 \global\setbox\@currbox 
1070 \color@vbox 
1071 \normalcolor 
1072 \vbox\bgroup\cb@start[\cb@curbarwd]\unvbox\@currbox\cb@end 
1073 \fi 
1074 \let@end@float\ltx@end@float 
1075 \flt@float@end 
1076 }
\end{verbatim}

\@float@dblfloat  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. In this case the \LaTeX macro had to be rewritten.

\begin{verbatim}
1077 \let\ltx@end@dblfloat\end@dblfloat
1078 \def\cb@end@dblfloat{% 
1079 \if@twocolumn 
1080 \cb@trace@stack{end dblfloat on page \the\c@page}\
1081 \cb@pop\cb@currentstack 
1082 \ifnum\cb@topleft=\cb@nil 
1083 \else 
1084 \cb@push\cb@currentstack 
1085 \global\cb@curbarwd=\cb@curbarwd 
1086 \@endfloatbox 
1087 \global\setbox\@currbox 
\end{verbatim}

43
Something similar needs to be done for the case where the float package is being used...

\let\flt@float@dblend\float@dblend
\def\float@dblend{%
\cb@trace@stack{end dbl float on page \the\c@page}%
\cb@pop\cb@currentstack
\ifnum\cb@topleft=\cb@nil
\ltx@footnotetext{#1}%
\else
\cb@push\cb@currentstack
\global\cb@curbarwd=\cb@curbarwd
\let\ltx@footnotetext\@footnotetext
\global\setbox\@currbox
\color@vbox
\normalcolor
\vboxgroup\cb@start[\cb@curbarwd]\unvbox\@currbox\cb@end
\fi
\let\end@dblfloat\ltx@end@dblfloat
\flt@float@dblend
}

\let\ltx@footnotetext\@footnotetext
\long\def\cb@footnotetext#1{%
\cb@trace@stack{end footnote on page \the\c@page}%
\cb@pop\cb@currentstack
\ifnum\cb@topleft=\cb@nil
\ltx@footnotetext{#1}%
\else
\cb@push\cb@currentstack
\edef\cb@temp{\the\cb@curbarwd}%
\ltx@footnotetext{\cb@start[\cb@temp]#1\cb@end}%
\fi
}
\@mpfootnotetext  Replacement for the \LaTeX\ macro of the same name. Same thing as \@footnotetext.

\let\ltx@mpfootnotetext\@mpfootnotetext
\long\def\cb@mpfootnotetext#1{\%
\cb@pop\cb@currentstack
\ifnum\cb@topleft=\cb@nil
\ltx@mpfootnotetext{#1}\
\else
\cb@push\cb@currentstack
\edef\cb@temp{\the\cb@curbarwd}\
\ltx@mpfootnotetext{\cb@start[\cb@temp]#1\cb@end}\
\fi}
\let\@mpfootnotetext\cb@mpfootnotetext
\endinput