The Changebar package [*]

Michael Fine
Distributed Systems Architecture

Johannes Braams
Kersengaarde 33
2723 BP Zoetermeer
The Netherlands
`texniek at braams.cistron.nl`

Printed September 19, 2005

Contents

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 support. 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

---

[*] This file has version number v3.5c, last revised 2005/09/18.

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.
- The colour of the changebars can be changed. This has sofar been tested with the dvips, pdftex and vtex drivers, but it may also work with other PostScript based drivers. It will *not* work for the `DVItoLN03` and emTEX drivers. For colored changebars to work, make sure that you specify the option color or 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[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
- TEXtures
- VTEX
- PDFTEX

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 `changebar.cfg`. If it contains the command `\ExecuteOption{textures}` the textures option will be used for all documents that are processed while the configuration file is in TEX's search path.

---

[1] For older documents the command `\driver` is available in the preamble of the document. It takes the options as defined for LATEX 2ε as argument.

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

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

Note that these options only work for *onecolumn* documents and will be ignored for a twocolumn document.

2.1.3 Color

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

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

2.1.4 Tracing

The package also implements tracing for its own debugging. The package options traceon and traceoff control tracing. An additional option 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

`\cbstart` `\cbend` All material between the macros `\cbstart` and `\cbend` is barred. The nesting of multiple changebars is allowed. The macro `\cbstart` has an optional parameter that specifies the width of the bar. The syntax is `\cbstart[`⟨*dimension*⟩`]`. If no width is specified, the current value of the parameter `\changebarwidth` is used. Note that `\cbstart` and `\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 `\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.

`\cbcolor` This macro is defined when the color option is selected. It's syntax is the same as the `\color` command from the `color` package.

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 `\deletebarwidth` 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`.

`changebargrey`

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 `\cb@minpoint` through `\cb@maxpoint`. 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 (`\cb@maxpoint`−`\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.

- 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'.

- The colour support has only been tested with the dvips and pdftex drivers.

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 n and $n + 1$ 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. With PDFLaTeX 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

`\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.

```
⟨*package⟩
\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.

```
\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.

```
\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`.

```
\newcount\cb@nextpoint
\cb@nextpoint=\cb@minpoint
```

`\cb@topleft`
`\cb@topright`
`\cb@botleft`
`\cb@botright`

These four 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.

```
\newcount\cb@topleft
\newcount\cb@topright
\newcount\cb@botleft
\newcount\cb@botright
```

`\cb@cnta`
`\cb@cntb`
`\cb@dima`

Sometimes we need temporarily store a value. For this purpose two count registers and a dimension register are allocated.

```
\newcount\cb@cnta
\newcount\cb@cntb
\newdimen\cb@dima
```

`\cb@curbarwd`

The dimension register `\cb@curbarwd` is used to store the width of the current bar.

```
\newdimen\cb@curbarwd
```

`\cb@page`
`\cb@pagecount`

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

```
\newcount\cb@page
\newcount\cb@pagecount
\cb@pagecount=0
```

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

- 0 for innerbars (default),
- 1 for outerbars,
- 2 gices leftbars,
- 3 gives rightbars.

```
\def\cb@barsplace{0}
```

`@cb@trace` A switch to enable tracing of the actions of this package

```
\newif\if@cb@trace
```

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

```
\newif\if@cb@firstcolumn
```

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

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

`\cb@positions` This macro calculates the (horizontal) positions of the changebars.

`\cb@odd@left` `\cb@odd@right` `\cb@even@left` `\cb@even@right` 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 PostScript command `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 `\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\cb@odd@right\cb@odd@left
  \global\advance\cb@odd@right by \textwidth
  \global\advance\cb@odd@right by \changebarsep
  \global\advance\cb@odd@right by 0.5\changebarwidth
  \global\advance\cb@odd@left by -\changebarsep
  \global\advance\cb@odd@left by -0.5\changebarwidth
```

On even sided pages we need to use `\evensidemargin` in the calculations when 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
  \global\advance\cb@even@right by \textwidth
  \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}}
```

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
- emTeX
- Textures
- VTeX
- PDFTeX

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.

```
\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{vtex}{\global\chardef\cb@driver@setup=5\relax}
```

```
\DeclareOption{PDFTeX}{\cb@pdftexcheck}
\DeclareOption{pdftex}{\cb@pdftexcheck}
```

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.

```
\def\cb@pdftexcheck{%
  \ifx\pdfsavepos\@undefined\cb@pdftexerror
  \else\ifx\pdfoutput\@undefined\cb@pdftexerror
  \else\ifnum\pdfoutput>0
    \global\chardef\cb@driver@setup=6\relax
    \ifx\cb@writexy\@undefined
      \newwrite\cb@writexy
      \newread\cb@readxy
      \immediate\openout\cb@writexy=\jobname.cb2\relax
    \fi
```

Redefine the `\cb@pdfxy` macro to write point coordinates to the `.cb2` file.

```
    \gdef\cb@pdfxy##1##2##3##4##5{%
      \immediate\write\cb@writexy{##1.##2p##3,##4,##5}%
      \expandafter\gdef\csname cb@##1.##2\endcsname{##3,##4,##5}}
  \else\cb@pdftexerror\fi\fi\fi}
```

Give a warning if we cannot support the pdftex option.

```
\def\cb@pdftexerror{\PackageError
      {changebar}%
      {PDFTeX option cannot be used}%
      {You are using a LaTeX run which does not generate PDF\MessageBreak
        or you are using a very old version of PDFTeX}}
```

The new features of LaTeX 2_ε make it possible to implement the outerbars option.

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

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.

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

A set of options to control tracing.

```
\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 \the\cb@topleft\space on \noexpand#1: #1}}%
  \def\cb@trace@pop#1{\cb@trace{%
      Popped point \the\cb@topleft\space from \noexpand#1: #1}}%
  }
```

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

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

The second option activates support for the `color` package.

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

The third option adds support for the `xcolor` package.

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

Signal an error if an unknown option was specified.

```
\DeclareOption*{\OptionNotUsed\PackageError
      {changebar}%
      {Unrecognised option ‘\CurrentOption’\MessageBreak
        known options are dvitoln03, dvitops, dvips,\MessageBreak
        emtex, textures, pdftex and vtex,
        grey, color, xcolor,\MessageBreak
        outerbars, innerbars, leftbars and rightbars}}
```

The default is to have grey change bars on the left side of the text on odd pages. When VTeX is used the option dvips is not the right one, so in that case we have vtex as the default driver. When PDFTeX is producing PDF output, the pdftex option is selected.

```
\ifx\VTeXversion\@undefined
  \ifx\pdfoutput\@undefined
    \ExecuteOptions{innerbars,traceoff,dvips,grey}
  \else
    \ifnum\pdfoutput>0
      \ExecuteOptions{innerbars,traceoff,pdftex,grey}
    \else
      \ExecuteOptions{innerbars,traceoff,dvips,grey}
    \fi
  \fi
\else
  \ExecuteOptions{innerbars,traceoff,vtex,grey}
\fi
```

A local configuration file may be used to define a site wide default for the driver, by calling `\ExecuteOptions` with the appropriate option. This will override the default specified above.

```
\InputIfFileExists{changebar.cfg}{}{}
```

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

```
\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`ting `\cb@trace@stack` to `\@gobble`.

```
\let\cb@trace@stack\@gobble
```

`\cb@trace@push` `\cb@trace@pop` 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 tracestacks is selected.

The default is to *not* trace the stacks.

```
\let\cb@trace@push\@gobble
\let\cb@trace@pop\@gobble
```

Now make all the selected options active, but...

```
\ProcessOptions\relax
```

We have to make sure that when the document is being processed by pdfLaTeX, 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 doucment.

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

`\cb@trace` A macro that formats the tracing messages.

```
\newcommand{\cb@trace}[1]{%
  \if@cb@trace
    \GenericWarning
      {(changebar)\@spaces\@spaces}%
      {Package 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
- VTEX
- PDFTEX

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.

```
\if@compatibility
  \def\driver#1{%
    \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}%
      \def\VTeX{VTEX}%
      \def\pdfTeX{PDFTEX}
```

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`.

```
      \global\chardef\cb@driver@setup=0\relax
      \ifx\tempa\LN       \global\chardef\cb@driver@setup=0\fi
      \ifx\tempa\DVItoPS  \global\chardef\cb@driver@setup=1\fi
      \ifx\tempa\DVIPS    \global\chardef\cb@driver@setup=2\fi
      \ifx\tempa\emTeX    \global\chardef\cb@driver@setup=3\fi
      \ifx\tempa\Textures \global\chardef\cb@driver@setup=4\fi
      \ifx\tempa\VTeX     \global\chardef\cb@driver@setup=5\fi
      \ifx\tempa\pdfTeX   \cb@pdftexcheck\fi
    \egroup}
```

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

```
  {\def\do{\noexpand\do\noexpand}
    \xdef\@preamblecmds{\@preamblecmds \do\driver}
    }
\fi
```

`\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.

```
\def\cb@trace@defpoint#1#2{%
  \cb@trace{%
    defining point \the#1 at position \the#2
    \MessageBreak
    cb@pagecount: \the\cb@pagecount; page \thepage}}
```

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

```
\def\cb@trace@connect#1#2#3{%
  \cb@trace{%
    connecting points \the#1 and \the#2; barwidth: \the#3
    \MessageBreak
    cb@pagecount: \the\cb@pagecount; page \thepage}}
```

`\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.

```
\def\cb@setup@specials{%
```

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 `\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}}
```

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 6.5536 mul\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{dvitops: inline
              gsave \cb@ps@color\space
              \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}%
    \cb@trace@connect##1##2##3}
  \let\cb@resetpoints\relax
```

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 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 `pt`s into pixels which of course is done by dividing by 72.27 (`pt`s per inch) and multiplying by `Resolution` giving the resolution of the PostScript device in use as a PostScript variable.

```
\or
  \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}%
    \cb@trace@defpoint##1##2}
  \def\cb@connect##1##2##3{%
    \special{ps:
              gsave \cb@ps@color\space
              \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
              stroke grestore}%
    \cb@trace@connect##1##2##3}
  \let\cb@resetpoints\relax
```

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.

```
\or
  \def\cb@defpoint##1##2{%
    \special{em:point \the##1,\the##2}%
    \cb@trace@defpoint##1##2}
  \def\cb@connect##1##2##3{%
    \special{em:line \the##1,\the##2,\the##3}%
    \cb@trace@connect##1##2##3}
  \let\cb@resetpoints\relax
```

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.

```
\or
  \def\cb@defpoint##1##2{%
    \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 setlinecap \cb@ps@color\space
                \expandafter\cb@removedim\the##3\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}%
    \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\space 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\space
                \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
                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\relax
```

`\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}
```

`\cb@findpdfpoint` 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
}
```

`\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@next
}%
```

`\cb@pdfreadxy`

The `\cb@pdfreadxy` macro reads lines from the `.cb2` 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
      \ifx\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}%
          \global\let\cb@pdfpagenr\cb@pdfpg
        \fi
        \ifnum\cb@pdfpg>##1\else
          \global\def\cb@next{\cb@pdfreadxy{##1}}%
        \fi
      \fi
    }%
  \fi
  \cb@next
}%
```

`\cb@pdfparsexy`

The `\cb@pdfparsexy` macro extracts the pagecount from an entry read in from the `.cb2` 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
  \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\else
  0.\the##1\fi\fi\fi}
```

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.

```
       \begingroup
         \cb@dima=\cb@pdfz
         \advance\cb@dima by-\cb@pdfx
         \advance\cb@dima by1in
         \cb@dima=0.996264009963\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}%
           \pdfliteral{q \cb@temp\space g \cb@temp\space G}%
         \else
           \pdfliteral{q \cb@current@color}%
         \fi
         \edef\cb@temp{\expandafter\cb@removedim\the\cb@dima\space}%
         \cb@dima=\cb@pdftopy
         \advance\cb@dima-\cb@pdfy\relax
         \cb@dima=0.996264009963\cb@dima\relax
         ##3=0.996264009963##3\relax
         \pdfliteral direct{\expandafter\cb@removedim\the##3 w
           \cb@temp 0 m
           \cb@temp \expandafter\cb@removedim\the\cb@dima\space l S Q}%
      \endgroup
```

We look up the two unused points to get them removed from `\cb@pdfpoints`.

```
      \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%
  }%
```

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

```
  \gdef\cb@checkPdfxy##1##2##3##4##5{%
    \cb@@findpdfpoint{##1}{##2}%
    \ifnum##3=\cb@pdfx\relax
      \ifnum##4=\cb@pdfy\relax
        \ifdim##5=\cb@pdfz\relax
        \else
        \cb@error
        \fi
      \else
        \cb@error
      \fi
    \else
      \cb@error
    \fi
  }
```

For PDFTeX we don't need a limit on the number of bar points.

```
  \def\cb@maxpoint{9999999}
  \let\cb@resetpoints\relax
```

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.

```
\or
\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
```

The last thing to do is to forget about `\cb@setup@specials`.

```
\global\let\cb@setup@specials\relax}
```

`\cbstart` 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`.

```
\newcommand*{\cbstart}{\@ifnextchar [%]
                        {\cb@start}%
                        {\cb@start[\changebarwidth]}}
```

`\cbend` 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.

```
\newcommand*{\cbend}{\cb@end}
```

`\cbdelete` 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`.

```
\newcommand*{\cbdelete}{\@ifnextchar [%]
  {\cb@delete}%
  {\cb@delete[\deletebarwidth]}}
```

`\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.

```
\def\cb@delete[#1]{\vbox to \z@{\vss\cb@start[#1]\vskip #1\cb@end}}
```

`\changebar` `\endchangebar` 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.

```
\newenvironment{changebar}%
               {\@ifnextchar [{\cb@start}%
                              {\cb@start[\changebarwidth]}}%
               {\cb@end}
```

`\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 dimen register `\changebarsep`.

```
\newlength{\changebarsep}
\setlength{\changebarsep}{30pt}
```

`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}
```

When one of the options color or xcolor was selected we need to load the appropriate package. When we’re run by pdfLaTeX we need to pass that information on to that package.

```
\@ifpackagewith{changebar}{\csname cb@color@pkg\endcsname}{%
    \RequirePackage{\cb@color@pkg}%
```

Then we need to define the command `\cbcolor` which is a slightly modified copy of the command `\color` from the `color` package.

`\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{%
  \@ifnextchar[\@undeclaredcbcolor\@declaredcbcolor}
```

`\@undeclaredcbcolor` Call the driver-dependent command `\color@`⟨*model*⟩ to define `\cb@current@color`.

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

`\@declaredcbcolor`

```
\def\@declaredcbcolor#1{%
  \begingroup
    \color{#1}%
    \global\let\cb@current@color\current@color
  \endgroup
  \ignorespaces}%
}{%
```

When the color option wasn't specified the usage of the `\cbcolor` command results in a warning message.

```
\def\cbcolor{\@ifnextchar[%]
  \@@cbcolor\@cbcolor}%
\def\@@cbcolor[#1]#2{\cb@colwarn\def\@@cbcolor[##1]##2{}}%
\def\@cbcolor#1{\cb@colwarn\def\@cbcolor##1{}}%
\def\cb@colwarn{\PackageWarning{Changebar}%
  {You didn't specify the option `color';\MessageBreak
    your command \string\cbcolor\space will be ignored}}%
}
```

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
```

Store the width of the¡ current bar in `\cb@curbarwd`.

```
  \cb@curbarwd#1\relax
  \cb@push\cb@currentstack
```

Now find out on which page the start of this bar finaly ends up; due to the asynchronous nature of the output routine it might be a different page. The macro `\cb@checkpage` finds the page number on the history stack.

```
  \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
  \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
```

Restore `\cb@pagecount`.

```
  \cb@pagecount\cb@cnta
  \cb@advancePoint}
```

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

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

`\cb@end` This macro ends a changebar. It pops the current point and nesting level off `\cb@currentstack` and sets the end point by calling `\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 `\cb@setEndPoints` inside a `\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.

```
\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 `\cb@checkpage` to find the page this point finally ends up on.

```
    \cb@checkpage\thr@@
```

Again, we need to temporarily overwrite `\cb@pagecount`.

```
    \cb@cnta\cb@pagecount
    \cb@pagecount\cb@page\advance\cb@pagecount\m@ne
    \ifvmode
      \cb@setEndPoints
    \else
      \vadjust{\cb@setEndPoints}%
    \fi

    \cb@pagecount\cb@cnta
  \fi
  \ignorespaces}
```

`\cb@checkpage` The macro `\cb@checkpage` checks the history stack in order to find out on which page a set of points finaly ends up.

We expect the identification of the points in `\cb@topleft` and `\cb@page`. The resulting page will be stored in `\cb@page`. The parameter indicates whether we are searching for a begin point (0) or end point (3).

```
\def\cb@checkpage#1{%
```

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 `\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 `\cb@cntb` with the `\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
  }
```

`\cb@FindPageNum`

`\cb@FindPageNum` recursively searches through the history stack until an entry is found that is equal to `\cb@cnta`.

```
\def\cb@FindPageNum{%
  \ifnum\cb@topleft=\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 wether 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 pagenumber 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.

```
\def\cb@writeAux#1{%
  \if@filesw
    \begingroup
      \edef\point{\the#1}%
      \edef\level{\the\cb@curbarwd}%
      \let\the=\z@
      \edef\cb@temp{\write\@auxout
        {\string\cb@barpoint{\point}{\the\cb@pagecount}{\level}}}%
      \cb@temp
    \endgroup
  \fi}
```

5.5 Macros for Making It Work Across Page Breaks

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

```
\newif\if@cb@pagejump
```

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

```
\def\cb@pagejumplst{-1}
```

`\cb@nextpagejump` The next pagecount from the list.

```
\def\cb@nextpagejump{-1}
```

`\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.

```
\def\cb@pagejump#1{\xdef\cb@pagejumplst{\cb@pagejumplst,#1}}
```

`\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.

```
\def\cb@writepagejump#1{
  \cb@cntb=\cb@pagecount
  \advance\cb@cntb by#1\relax
  \global\setbox\@leftcolumn\vbox to\@colht{%
    \edef\cb@temp{\write\@auxout{\string\cb@pagejump{\the\cb@cntb}}}%
    \cb@temp
    \dimen@ \dp\@leftcolumn
    \unvbox \@leftcolumn
    \vskip -\dimen@
  }%
}
```

`\cb@poppagejump` Pop an entry from `pagejumplst`. The entry is put in `\cb@nextpagejump`.

```
\def\cb@poppagejump#1,#2\relax{%
  \gdef\cb@nextpagejump{#1}%
  \gdef\cb@pagejumplst{#2}}
```

`\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.

```
\def\cb@checkpagecount{%
  \if@twocolumn
    \if@firstcolumn
      \ifodd\cb@pagecount\global\advance\cb@pagecount by\@ne\fi
    \fi
  \else
    \ifodd\cb@pagecount\global\advance\cb@pagecount by\@ne\fi
  \fi
```

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.

```
  \if@twoside
    \cb@cntb=\cb@pagecount
    \divide\cb@cntb by\tw@
    \advance\cb@cntb by-\c@page
    \ifodd\cb@cntb
```

Here `\cb@pagecount` seems correct. Check if there is a page jump.

```
      \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
            \global\@cb@pagejumpfalse
          \fi
        \else
```

In the righthand column check the flag (see above). If set, write a pagejump, but compensate for the increase done in the lefthand column.

```
          \if@cb@pagejump
            \cb@writepagejump{-3}%
          \fi
```

```
        \fi
      \fi
    \else
```

Here `\cb@pagecount` is incorrect.

```
      \if@twocolumn
        \if@firstcolumn
          \global\advance\cb@pagecount by\tw@
          \global\@cb@pagejumpfalse
        \else
          \if@cb@pagejump
            \cb@trace{Page jump annulled, %
                      \string\cb@pagecount=\the\cb@pagecount}
          \else
            \cb@writepagejump{-1}%
            \global\advance\cb@pagecount by\tw@
            \cb@rerun
          \fi
        \fi
      \else
        \global\advance\cb@pagecount by\tw@
      \fi
    \fi
  \fi
}
```

`\@makecol`
`\@vtryfc`

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 `\cb@beginstack`. This stack is used to define new starting points for the change bars, which are added to thetop of box `\@cclv`. Then the stack `\cb@endstack` is built and processed by `\cb@processActive`. Finally the original `\@makecol` (saved as `\cb@makecol`) is executed.

```
\let\ltx@makecol\@makecol
\def\cb@makecol{%
  \if@twocolumn
    \cb@trace{Twocolumn: \if@firstcolumn Left \else Right \fi column}%
  \fi
  \cb@trace@stack{before makecol, page \the\c@page,
                  \string\cb@pagecount=\the\cb@pagecount}%
  \let\cb@writeAux\@gobble
```

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
       \dimen@ \dp\@outputbox
       \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.

```
\let\ltx@vtryfc\@vtryfc
\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.

```
  \cb@checkpagecount
  \ltx@vtryfc{#1}%
  \if@twocolumn
    \global\setbox\@outputbox \vbox to\@colht{%
      \if@firstcolumn\write\@auxout{\string\@cb@firstcolumntrue}%
      \else\write\@auxout{\string\@cb@firstcolumnfalse}%
      \fi
      \unvbox\@outputbox
      \boxmaxdepth\maxdepth
    }%
  \fi
  \global\advance\cb@pagecount by \@ne
}
\let\@vtryfc\cb@vtryfc
```

`\cb@processActive` 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).
```

```
\def\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}
```

`\cb@startSpanBars`

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

```
\def\cb@startSpanBars{%
  \cb@pop\cb@beginstack
  \ifnum\cb@topleft=\cb@nil
  \else
    \cb@setBeginPoints
    \cb@trace@stack{after StartSpanBars, page \the\c@page}%
    \expandafter\cb@startSpanBars
  \fi
  }
```

`\cb@buildstack`
`\cb@endstack`

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

```
\def\cb@buildstack{%
  \cb@initstack\cb@endstack
  \cb@pushNextActive}
```

`\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.

```
\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}
```

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 `<element> ... <element>`

Each element is of the form `xxxnyyypzzzl` 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 `\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 `\cb@endstack` and `\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 `\cb@currentstack` contains all the current bars. A `\cb@start` pushes an element onto this stack. A `\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.

```
\def\cb@initstack#1{\xdef#1{}}
```

`\cb@historystack`
`\cb@write`
`\cb@read`

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

```
\cb@initstack\cb@historystack
```

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

```
\newwrite\cb@write
\newread\cb@read
```

And we open the history file for writing (which is done when the `.aux` file is read in).

```
\immediate\openout\cb@write=\jobname.cb\relax
```

`\cb@endstack`
`\cb@beginstack`

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

```
\cb@initstack\cb@endstack
\cb@initstack\cb@beginstack
```

`\cb@tempstack`

Allocate a stack for temporary storage

```
\cb@initstack\cb@tempstack
```

`\cb@currentstack` And we allocate an extra stack that is needed to implement nesting without having to rely on TeX's grouping mechanism.

```
\cb@initstack\cb@currentstack
```

`\cb@pop` 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`.

```
\def\cb@thehistorystack{\cb@historystack}
\def\cb@pop#1{%
  \ifx #1\@empty
    \def\cb@temp{#1}%
    \ifx\cb@temp\cb@thehistorystack
      \ifeof\cb@read
      \else
        {\endlinechar=-1\read\cb@read to\cb@temp
         \xdef\cb@historystack{\cb@historystack\cb@temp}%
        }%
      \fi
    \fi
  \fi
  \ifx#1\@empty
    \global\cb@topleft\cb@nil
    \global\cb@page\z@\relax
  \else
    \expandafter\cb@carcdr#1e#1%
  \fi
  \cb@trace@pop{#1}}
```

`\cb@carcdr` This macro is used to 'decode' a stack entry.

```
\def\cb@carcdr#1n#2p#3l#4e#5{%
  \global\cb@topleft#1\relax
  \global\cb@page#2\relax
  \global\cb@curbarwd#3\relax
  \xdef#5{#4}}
```

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

```
\def\cb@push#1{%
  \xdef#1{\the\cb@topleft n\the\cb@page p\the\cb@curbarwd l#1}%
  \cb@trace@push{#1}}

```

`\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.

```
\def\cb@barpoint#1#2#3{\cb@cnta=#2
  \if@cb@firstcolumn\advance\cb@cnta by\m@ne\fi
  \immediate\write\cb@write{#1n\the\cb@cnta p#3l}}
```

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.

```
\AtBeginDocument{%
  \cb@setup@specials
```

Add a sentinel to `\cb@pagejumplst`.

```
  \cb@pagejump{999999999,}%
```

Compute the left and right positions of the changebars.

```
  \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}
```

`\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.

```
\AtEndDocument{%
  \clearpage
  \cb@initstack\cb@historystack
  \immediate\closein\cb@read
  \immediate\openin\cb@read=\jobname.cb%
```

Let `\cb@pdfxy=\cb@checkPdfxy` for checking. Make `\cb@pagejump` dummy.

```
  \ifx\cb@readxy\@undefined
  \else
    \immediate\closein\cb@readxy
    \immediate\openin\cb@readxy=\jobname.cb2%
    \def\cb@pdfpoints{}%
    \def\cb@pdfpagenr{0}%
  \fi
  \@cb@firstcolumnfalse
  \cb@checkrerun
  \let\cb@pdfxy\cb@checkPdfxy
  \let\cb@pagejump\@gobble
  \let\cb@barpoint\cb@checkHistory}
```

`\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.

```
\def\cb@checkHistory#1#2#3{%
  \cb@pop\cb@historystack
  \ifnum #1=\cb@topleft\relax
    \cb@cnta=#2
    \if@cb@firstcolumn\advance\cb@cnta by\m@ne\fi
    \ifnum \cb@cnta=\cb@page\relax
```

Both page and point numbers are equal; do nothing,

```
    \else
```

but generate a warning when page numbers don't match, or

```
      \cb@error
    \fi
  \else
```

when point numbers don't match.

```
    \cb@error
  \fi}
```

Dummy definition for `\cb@checkPdfxy`. This will be overwritten by the `pdftex` option.

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

`\cb@rerun` The macro `\cb@rerun` is called when we detect that we need to rerun LaTeX.

```
\def\cb@rerun{%
  \global\let\cb@checkrerun\cb@error}
\let\cb@checkrerun\relax
```

`\cb@error` 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 pdftex we also disable `\cb@checkPdfxy`.

```
\def\cb@error{%
  \PackageWarning{Changebar}%
    {Changebar info has changed.\MessageBreak
     Rerun to get the bars right}
  \gdef\cb@checkHistory##1##2##3{}%
  \let\cb@barpoint\cb@checkHistory

  \gdef\cb@checkPdfxy##1##2##3##4##5{}%
  \let\cb@pdfxy\cb@checkPdfxy}
```

5.8 Macros For Making It Work With Nested Floats/Footnotes

`\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 `\end@float`.

```
\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
    \global\cb@curbarwd=\cb@curbarwd
    \@endfloatbox
    \global\setbox\@currbox
      \color@vbox
      \normalcolor
      \vbox\bgroup\cb@start[\cb@curbarwd]\unvbox\@currbox\cb@end
  \fi
  \ltx@end@float}
\let\end@float\cb@end@float
```

This only works if this new version of `\end@float` is really used. With LaTeX2.09 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 2$_\varepsilon$ class files use `\newenvironment` to define the figure and table environments. In that case there is no need to rebind `\endfigure`.

`\float@end` When the `float` package is being used we need to take care of its changes to the float mechanism. It defines it's 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`.

```
\let\flt@float@end\float@end
```

Then we redefine it to insert the changebarcode.

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

`\end@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.

```
\let\ltx@end@dblfloat\end@dblfloat
\def\cb@end@dblfloat{%
  \if@twocolumn
    \cb@trace@stack{end dblfloat on page \the\c@page}%
    \cb@pop\cb@currentstack
    \ifnum\cb@topleft=\cb@nil
    \else
```

```
      \cb@push\cb@currentstack
      \global\cb@curbarwd=\cb@curbarwd
      \@endfloatbox
      \global\setbox\@currbox
        \color@vbox
        \normalcolor
        \vbox\bgroup\cb@start[\cb@curbarwd]\unvbox\@currbox\cb@end
    \fi
    \@endfloatbox
    \ifnum\@floatpenalty <\z@
      \@largefloatcheck
      \@cons\@dbldeferlist\@currbox
    \fi
    \ifnum \@floatpenalty =-\@Mii \@Esphack\fi
  \else
    \end@float
  \fi}
\let\end@dblfloat\cb@end@dblfloat
```

`\float@dblend` 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
  \else
    \cb@push\cb@currentstack
    \global\cb@curbarwd=\cb@curbarwd
    \@endfloatbox
    \global\setbox\@currbox
      \color@vbox
      \normalcolor
      \vbox\bgroup\cb@start[\cb@curbarwd]\unvbox\@currbox\cb@end
  \fi
  \let\end@dblfloat\ltx@end@dblfloat
  \flt@float@dblend
  }
```

`\@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.

```
\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}
```

```
949 \let\@footnotetext\cb@footnotetext
```

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

```
950 \let\ltx@mpfootnotetext\@mpfootnotetext
951 \long\def\cb@mpfootnotetext#1{%
952   \cb@pop\cb@currentstack
953   \ifnum\cb@topleft=\cb@nil
954     \ltx@mpfootnotetext{#1}%
955   \else
956     \cb@push\cb@currentstack
957     \edef\cb@temp{\the\cb@curbarwd}%
958     \ltx@mpfootnotetext{\cb@start[\cb@temp]#1\cb@end}%
959   \fi}
960 \let\@mpfootnotetext\cb@mpfootnotetext
961 ⟨/package⟩
```

Index

Numbers in *italics* indicate the page where the macro is described, the underlined numbers indicate the number of the line of code where the macro is defined, all other numbers indicate where a macro is used.