The Changebar package *

Michael Fine
Distributed Systems Architecture

Johannes Braams
johannes.braams at texniek.nl

Printed December 24, 2023

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 and XeTeXsupport. More can easily be added.

1 Introduction

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

   Features

---

*This file has version number v3.7a, last revised 2023/12/24.

- 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, vtex andxetex 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

- XeTEX

- luaTEX

---

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

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.

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` All material between the macros `\cbstart` and `\cbend` is barred. The nesting of
`\cbend` 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 (*env.*) 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 (env.)   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 (env.)   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.

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 PDFTeXor XeTeX.

- 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.

```
1 ⟨*package⟩
2 \def\cb@maxpoint{80}
```

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

```
3 \def\cb@minpoint{1}
```

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

```
4 \def\cb@nil{0}
```

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

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

\cb@topleft These four counters are used to identify the four special points that specify a
\cb@topright changebar. The point defined by \cb@topleft is the one used to identify the
\cb@botleft changebar; the values of the other points are derived from it.
\cb@botright
```
7 \newcount\cb@topleft
8 \newcount\cb@topright
9 \newcount\cb@botleft
10 \newcount\cb@botright
```

\cb@cnta Sometimes we need temporarily store a value. For this purpose two count registers
\cb@cntb and a dimension register are allocated.
\cb@dima
```
11 \newcount\cb@cnta
12 \newcount\cb@cntb
13 \newdimen\cb@dima
```

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

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

\cb@page The macros need to keep track of the number of pages/columns output so far. To
\cb@pagecount 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).

```
15 \newcount\cb@page
16 \newcount\cb@pagecount
17 \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 gives leftbars,

   3 gives rightbars.

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

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

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

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

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

\cb@pdfxy 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.

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

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

\cb@odd@left Because the margins can differ for even and odd pages and because changebars
\cb@odd@right are sometimes on different sides of the paper we need four dimensions to store the
\cb@even@left result.
\cb@even@right
```
22 \newdimen\cb@odd@left
23 \newdimen\cb@odd@right
24 \newdimen\cb@even@left
25 \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.

```
26 \def\cb@positions{%
```

```
27    \global\cb@odd@left=\hoffset
28    \global\cb@even@left\cb@odd@left
29    \global\advance\cb@odd@left by \oddsidemargin
30    \global\cb@odd@right\cb@odd@left
31    \global\advance\cb@odd@right by \textwidth
32    \global\advance\cb@odd@right by \changebarsep
33    \global\advance\cb@odd@right by 0.5\changebarwidth
34    \global\advance\cb@odd@left by -\changebarsep
35    \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.

```
36    \if@twoside
37      \global\advance\cb@even@left by \evensidemargin
38      \global\cb@even@right\cb@even@left
39      \global\advance\cb@even@left by -\changebarsep
40      \global\advance\cb@even@left by -0.5\changebarwidth
41      \global\advance\cb@even@right by \textwidth
42      \global\advance\cb@even@right by \changebarsep
43      \global\advance\cb@even@right by 0.5\changebarwidth
44    \else
```

Otherwise just copy the result for odd pages.

```
45      \global\let\cb@even@left\cb@odd@left
46      \global\let\cb@even@right\cb@odd@right
47    \fi
48    }
```

\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.

```
49  {\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

- VT$_E$X

- PDFT$_E$X

- XeT$_E$X

- luaTeX

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.

```
50 \DeclareOption{DVItoLN03}{\global\chardef\cb@driver@setup=0\relax}
51 \DeclareOption{dvitoln03}{\global\chardef\cb@driver@setup=0\relax}
52 \DeclareOption{DVItoPS}{\global\chardef\cb@driver@setup=1\relax}
53 \DeclareOption{dvitops}{\global\chardef\cb@driver@setup=1\relax}
54 \DeclareOption{DVIps}{\global\chardef\cb@driver@setup=2\relax}
55 \DeclareOption{dvips}{\global\chardef\cb@driver@setup=2\relax}
56 \DeclareOption{emTeX}{\global\chardef\cb@driver@setup=3\relax}
57 \DeclareOption{emtex}{\global\chardef\cb@driver@setup=3\relax}
58 \DeclareOption{textures}{\global\chardef\cb@driver@setup=4\relax}
59 \DeclareOption{Textures}{\global\chardef\cb@driver@setup=4\relax}
60 \DeclareOption{VTeX}{\global\chardef\cb@driver@setup=5\relax}
61 \DeclareOption{vtex}{\global\chardef\cb@driver@setup=5\relax}

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

For the pdftex option we have to check that the current LaTeX run is using PDFT$_E$X 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.

```
64 \def\cb@pdftexcheck{%
65   \ifx\pdfsavepos\@undefined\cb@pdftexerror
66   \else\ifx\pdfoutput\@undefined\cb@pdftexerror
67   \else\ifnum\pdfoutput>0
68     \global\chardef\cb@driver@setup=6\relax
69     \ifx\cb@writexy\@undefined
70       \newwrite\cb@writexy
71       \newread\cb@readxy
72       \immediate\openout\cb@writexy=\jobname.cb2\relax
73     \fi
```

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

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

Give a warning if we cannot support the pdftex option.

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

83 \DeclareOption{XeTeX}{\cb@xetexcheck}
84 \DeclareOption{xetex}{\cb@xetexcheck}
```

9

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..

```
85 \def\cb@xetexcheck{%
86   \expandafter\ifx\csname XeTeXrevision\endcsname\@undefined \cb@xetexerror
87   \else
88     \global\chardef\cb@driver@setup=7\relax
89     \ifx\cb@writexy\@undefined
90       \newwrite\cb@writexy
91       \newread\cb@readxy
92       \immediate\openout\cb@writexy=\jobname.cb2\relax
93     \fi
```

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

```
94     \gdef\cb@pdfxy##1##2##3##4##5{%
95       \immediate\write\cb@writexy{##1.##2p##3,##4,##5}%
96       \expandafter\gdef\csname cb@##1.##2\endcsname{##3,##4,##5}}
97     \gdef\sec@nd@ftw@##1 ##2{##2}
98   \fi}
```

```
99  \DeclareOption{luaTeX}{\cb@luatexcheck}
100 \DeclareOption{luatex}{\cb@luatexcheck}
```

For the luatex option we have to check that the current LaTeX run is using luaTeX. If it is, we initialize the option and open an additional output file. If not, we ignore the option and issue a warning..

```
101 \def\cb@luatexcheck{%
102   \ifx\directlua\@undefined \cb@luatexerror
103   \else
104     \global\chardef\cb@driver@setup=8\relax
105     \ifx\cb@writexy\@undefined
106       \newwrite\cb@writexy
107       \newread\cb@readxy
108       \immediate\openout\cb@writexy=\jobname.cb2\relax
109     \fi
```

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

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

Give a warning if we cannot support the luatex option.

```
114 \def\cb@luatexerror{\PackageError
115       {changebar}%
116       {luaTeX option cannot be used}%
117       {You are not using luaLaTeX}}
```

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

```
118 \DeclareOption{outerbars}{\def\cb@barsplace{1}}
119 \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.

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

A set of options to control tracing.

```
122 \DeclareOption{traceon}{\@cb@tracetrue}
123 \DeclareOption{traceoff}{\@cb@tracefalse}
124 \DeclareOption{tracestacks}{%
125   \let\cb@trace@stack\cb@@show@stack
126   \def\cb@trace@push#1{\cb@trace{%
127     Pushed point \the\cb@topleft\space on \noexpand#1: #1}}%
128   \def\cb@trace@pop#1{\cb@trace{%
129     Popped point \the\cb@topleft\space from \noexpand#1: #1}}%
130   }
```

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

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

The second option activates support for the color package.

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

The third option adds support for the xcolor package.

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

Signal an error if an unknown option was specified.

```
139 \DeclareOption*{\OptionNotUsed\PackageError
140     {changebar}%
141     {Unrecognised option '\CurrentOption'\MessageBreak
142       known options are dvitoln03, dvitops, dvips,\MessageBreak
143       emtex, textures, pdftex, vtex and xetex,
144       grey, color, xcolor,\MessageBreak
145       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.

```
146 \ifx\VTeXversion\@undefined
147   \expandafter\ifx\csname XeTeXrevision\endcsname\@undefined
148     \ifx\pdfoutput\@undefined
149       \ExecuteOptions{innerbars,traceoff,dvips,grey}
150     \else
151       \ifnum\pdfoutput>0
152         \ExecuteOptions{innerbars,traceoff,pdftex,grey}
153       \else
154         \ExecuteOptions{innerbars,traceoff,dvips,grey}
155       \fi
156     \fi
157   \else
```

```
158      \ExecuteOptions{innerbars,traceoff,xetex,grey}
159   \fi
160 \else
161   \ExecuteOptions{innerbars,traceoff,vtex,grey}
162 \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.

```
163 \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.

```
164 \def\cb@@show@stack#1{%
165   \cb@trace{%
166     stack status at #1:\MessageBreak
167     current stack: \cb@currentstack\MessageBreak
168     \@spaces end stack: \cb@endstack\MessageBreak
169     \space\space begin stack: \cb@beginstack\MessageBreak
170     history stack: \cb@historystack
171     }}
```

The default is to *not* trace the stacks. This is achieved by \letting \cb@trace@stack to \@gobble.

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

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

The default is to *not* trace the stacks.

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

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

```
175 \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.

```
176 \ifx\pdfsavepos\@undefined
177 \else
178   \ifx\pdfoutput\@undefined
179   \else
180     \ifnum\pdfoutput>0
181       \global\chardef\cb@driver@setup=6\relax
182     \fi
183   \fi
184 \fi
```

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

```
185 \newcommand{\cb@trace}[1]{%
```

```
186  \if@cb@trace
187    \GenericWarning
188      {(changebar)\@spaces\@spaces}%
189      {Package changebar: #1\@gobble}%
190  \fi
191  }
```

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

- XeTeX

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.

```
192 \if@compatibility
193   \def\driver#1{%
194     \bgroup\edef\next{\def\noexpand\tempa{#1}}%
195       \uppercase\expandafter{\next}%
196     \def\LN{DVITOLN03}%
197     \def\DVItoPS{DVITOPS}%
198     \def\DVIPS{DVIPS}%
199     \def\emTeX{EMTEX}%
200     \def\Textures{TEXTURES}%
201     \def\VTeX{VTEX}%
202     \def\pdfTeX{PDFTEX}%

203       \def\xeTeX{XETEX}

204       \def\luaTeX{LUATEX}
```

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.

```
205       \global\chardef\cb@driver@setup=0\relax
206       \ifx\tempa\LN        \global\chardef\cb@driver@setup=0\fi
207       \ifx\tempa\DVItoPS   \global\chardef\cb@driver@setup=1\fi
208       \ifx\tempa\DVIPS     \global\chardef\cb@driver@setup=2\fi
209       \ifx\tempa\emTeX     \global\chardef\cb@driver@setup=3\fi
```

```
210     \ifx\tempa\Textures  \global\chardef\cb@driver@setup=4\fi
211     \ifx\tempa\VTeX      \global\chardef\cb@driver@setup=5\fi
212     \ifx\tempa\pdfTeX    \cb@pdftexcheck\fi
213     \ifx\tempa\xeTeX     \cb@xetexcheck\fi

214     \ifx\tempa\luaTeX    \cb@luatexcheck\fi

215   \egroup}
```

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

```
216   {\def\do{\noexpand\do\noexpand}
217    \xdef\@preamblecmds{\@preamblecmds \do\driver}
218    }
219 \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.

```
220 \def\cb@trace@defpoint#1#2{%
221   \cb@trace{%
222     defining point \the#1 at position \the#2
223     \MessageBreak
224     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.

```
225 \def\cb@trace@connect#1#2#3{%
226   \cb@trace{%
227     connecting points \the#1 and \the#2; barwidth: \the#3
228     \MessageBreak
229     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.

```
230 \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.

```
231 \ifcase\cb@driver@setup
```

14

```
232    \def\cb@defpoint##1##2{%
233      \special{ln03:defpoint \the##1(\the##2,)}%
234      \cb@trace@defpoint##1##2}
235    \def\cb@connect##1##2##3{%
236      \special{ln03:connect \the##1\space\space \the##2\space \the##3}%
237      \cb@trace@connect##1##2##3}
238    \def\cb@resetpoints{%
239      \special{ln03:resetpoints \cb@minpoint \space\cb@maxpoint}}
```

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

```
240  \or
241    \def\cb@defpoint##1##2{%
242      \special{dvitops: inline
243                \expandafter\cb@removedim\the##2\space 6.5536 mul\space
244                /CBarX\the##1\space exch def currentpoint exch pop
245                /CBarY\the##1\space exch def}%
246      \cb@trace@defpoint##1##2}

247    \def\cb@connect##1##2##3{%
248      \special{dvitops: inline
249                gsave \cb@ps@color\space
250                \expandafter\cb@removedim\the##3\space 6.5536 mul\space
251                CBarX\the##1\space\space CBarY\the##1\space\space moveto
252                CBarX\the##2\space\space CBarY\the##2\space\space lineto
253                stroke grestore}%
254      \cb@trace@connect##1##2##3}
255    \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 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.

```
256  \or
257    \def\cb@defpoint##1##2{%
258      \special{ps:
259                \expandafter\cb@removedim\the##2\space
260                Resolution\space mul\space 72.27\space div\space
261                /CBarX\the##1\space exch def currentpoint exch pop
262                /CBarY\the##1\space exch def}%
263      \cb@trace@defpoint##1##2}

264    \def\cb@connect##1##2##3{%
265      \special{ps:
266                gsave \cb@ps@color\space
267                \expandafter\cb@removedim\the##3\space
268                Resolution\space mul\space 72.27\space div\space
269                setlinewidth
270                CBarX\the##1\space\space CBarY\the##1\space\space moveto
271                CBarX\the##2\space\space CBarY\the##2\space\space lineto
272                stroke grestore}%
```

```
273        \cb@trace@connect##1##2##3}
274    \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.

```
275 \or
276    \def\cb@defpoint##1##2{%
277       \special{em:point \the##1,\the##2}%
278       \cb@trace@defpoint##1##2}
279    \def\cb@connect##1##2##3{%
280       \special{em:line \the##1,\the##2,\the##3}%
281       \cb@trace@connect##1##2##3}
282    \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.

```
283 \or
284    \def\cb@defpoint##1##2{%
285       \special{postscript 0 0 transform}% leave [x,y] on the stack
286       \special{rawpostscript
287                \expandafter\cb@removedim\the##2\space
288                /CBarX\the##1\space exch def
289                itransform exch pop
290                /CBarY\the##1\space exch def}%
291       \if@cb@trace\cb@trace@defpoint##1##2\fi}

292    \def\cb@connect##1##2##3{%
293       \special{rawpostscript
294                gsave 1 setlinecap \cb@ps@color\space
295                \expandafter\cb@removedim\the##3\space
296                setlinewidth
297                CBarX\the##1\space\space CBarY\the##1\space\space moveto
298                CBarX\the##2\space\space CBarY\the##2\space\space lineto
299                stroke grestore}%
300       \if@cb@trace\cb@trace@connect##1##2##3\fi}
301    \let\cb@resetpoints\relax
```

The following definitions were kindly provided by Michael Vulis.

```
302 \or
303    \def\cb@defpoint##1##2{%
304       \special{pS:
305                \expandafter\cb@removedim\the##2\space
306                Resolution\space mul\space 72.27\space div\space
307                /CBarX\the##1\space exch def currentpoint exch pop
308                /CBarY\the##1\space exch def}%
309       \cb@trace@defpoint##1##2}

310    \def\cb@connect##1##2##3{%
311       \special{pS:
312                gsave \cb@ps@color\space
313                \expandafter\cb@removedim\the##3\space
314                Resolution\space mul\space 72.27\space div\space
315                setlinewidth
```

```
316                     CBarX\the##1\space\space CBarY\the##1\space\space moveto
317                     CBarX\the##2\space\space CBarY\the##2\space\space lineto
318                     stroke grestore}%
319       \cb@trace@connect##1##2##3}
320     \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.

```
321 \or
322     \immediate\closeout\cb@writexy
323     \immediate\openin\cb@readxy=\jobname.cb2\relax
```

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

```
324     \def\cb@pdfpoints{}
325     \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.

```
326     \def\cb@findpdfpoint##1##2{%
327         \edef\cb@temp
328             {\noexpand\cb@@findpdfpoint{\the##1}{\the##2}}%
329         \cb@temp
330     }
```

\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.

```
331     \def\cb@@findpdfpoint##1##2{%
332         \ifnum##2<\cb@pdfpagenr\relax\else
333             \cb@pdfreadxy{##2}%
334         \fi
335         \let\cb@pdfx\relax
336         \ifx\cb@pdfpoints\@empty\else
337             \ifnum##2<0\relax
338             \else
339                 \edef\cb@temp{\noexpand\cb@pdffind{##1}{##2}\cb@pdfpoints\relax{}}%
340                 \cb@temp
341             \fi
342         \fi
343     }
```

\cb@pdffind   The \cb@pdffind recursively searches through \cb@pdfpoints to find point #1
               on pagecount #2. \cb@pdfpoints contains entries of the form
               $\langle pointnr\rangle.\langle pagecount\rangle$p$\langle x\rangle,\langle y\rangle,\langle z\rangle$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.

```
344    \def\cb@pdffind##1##2##3.##4p##5,##6,##7pt##8\relax##9{%
345      \def\cb@next{%
346        \cb@pdffind{##1}{##2}##8\relax{##9##3.##4p##5,##6,##7pt}}%
347      \ifnum ##1=##3
348        \ifnum ##2=##4
349          \def\cb@pdfx{##5sp}%
350          \def\cb@pdfy{##6sp}%
351          \def\cb@pdfz{##7pt}%
352          \let\cb@next\relax
353          \gdef\cb@pdfpoints{##9##8}%
354        \fi
355      \fi
356      \ifx\relax##8\relax
357        \let\cb@next\relax
358      \fi
359      \cb@next
360    }%
```

\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.

```
361    \def\cb@pdfreadxy##1{%
362      \let\cb@next\relax
363      \ifeof\cb@readxy
364        \global\let\cb@pdfpagenr\cb@maxpoint
365      \else
366        {\endlinechar=-1\read\cb@readxy to\cb@temp
367          \ifx\cb@temp\@empty\else
368            \expandafter\cb@pdfparsexy\cb@temp
369            \ifnum\cb@pdfpg<0\else
370              \xdef\cb@pdfpoints{\cb@pdfpoints\cb@temp}%
371              \cb@trace{PDFpoints=\cb@pdfpoints}%
372              \global\let\cb@pdfpagenr\cb@pdfpg
373            \fi
374            \ifnum\cb@pdfpg>##1\else
375              \global\def\cb@next{\cb@pdfreadxy{##1}}%
376            \fi
377          \fi
378        }%
379      \fi
380      \cb@next
381    }%
```

\cb@pdfparsexy   The \cb@pdfparsexy macro extracts the pagecount from an entry read in from
the .cb2 file.

```
382    \def\cb@pdfparsexy##1.##2p##3,##4,##5pt{%
383      \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.

18

```
384    \def\cb@defpoint##1##2{%
385      \if@filesw
386        \begingroup
387          \edef\point{{\the##1}{\the\cb@pagecount}}%
388          \let\the=\z@
389          \pdfsavepos
390          \edef\cb@temp{\write\@auxout
391            {\string\cb@pdfxy\point
392              {\the\pdflastxpos}{\the\pdflastypos}{\the##2}}}%
393          \cb@temp
394        \endgroup
395      \fi
396      \cb@trace@defpoint##1##2%
397    }%
```

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

```
398    \def\cb@cvtpct##1{%
399      \ifnum##1<0 0\else
400      \ifnum##1>99 1\else
401      \ifnum##1<10 0.0\the##1\else
402      0.\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.

```
403    \def\cb@pdf@scale{0.996264009963}
```

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.

```
404    \def\cb@connect##1##2##3{%
405      \cb@findpdfpoint{##1}\cb@pagecount
406      \ifx\cb@pdfx\relax\cb@rerun
407      \else
408        \let\cb@pdftopy\cb@pdfy
409        \cb@findpdfpoint{##2}\cb@pagecount
410        \ifx\cb@pdfx\relax\cb@rerun
411        \else
```

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

```
412          \begingroup
413            \cb@dima=\cb@pdfz
414            \advance\cb@dima by-\cb@pdfx
415            \advance\cb@dima by1in%
416            \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.

```
417            \ifx\cb@current@color\@undefined
418              \def\cb@temp{\cb@cvtpct\c@changebargrey}%
419              \pdfliteral{q \cb@temp\space g \cb@temp\space G}%
```

```
420           \else
421             \pdfliteral{q \cb@current@color}%
422           \fi
423           \edef\cb@temp{\expandafter\cb@removedim\the\cb@dima\space}%
424           \cb@dima=\cb@pdftopy
425           \advance\cb@dima-\cb@pdfy\relax
426           \cb@dima=\cb@pdf@scale\cb@dima\relax
427           ##3=\cb@pdf@scale##3\relax
428           \pdfliteral direct{\expandafter\cb@removedim\the##3 w
429             \cb@temp 0 m
430             \cb@temp \expandafter\cb@removedim\the\cb@dima\space l S Q}%
431         \endgroup
```

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

```
432         \cb@cntb=##1\relax
433         \ifodd\cb@cntb\advance\cb@cntb 1\else\advance\cb@cntb -1\fi
434         \cb@findpdfpoint\cb@cntb\cb@pagecount
435         \cb@cntb=##2\relax
436         \ifodd\cb@cntb\advance\cb@cntb 1\else\advance\cb@cntb -1\fi
437         \cb@findpdfpoint\cb@cntb\cb@pagecount
438       \fi
439     \fi
440     \cb@trace@connect##1##2##3%
441   }%
```

`\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.

```
442   \gdef\cb@checkPdfxy##1##2##3##4##5{%
443     \cb@@findpdfpoint{##1}{##2}%
444 %     \end{macrocdode}
445 %\begin{changebar}
446 %   \begin{macrocode
447     \ifdim##3sp=\cb@pdfx\relax
448       \ifdim##4sp=\cb@pdfy\relax
449 %     \end{macrocdode}
450 %\end{changebar}
451 %   \begin{macrocode
452         \ifdim##5=\cb@pdfz\relax
453         \else
454           \cb@error
455         \fi
456       \else
457         \cb@error
458       \fi
459     \else
460       \cb@error
461     \fi
462   }
```

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

```
463   \def\cb@maxpoint{9999999}
464   \let\cb@resetpoints\relax
465 \or
```

The code for XeTeX is, like for PDFTeX, 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.

```
466    \immediate\closeout\cb@writexy
467    \immediate\openin\cb@readxy=\jobname.cb2\relax
```

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

```
468    \def\cb@pdfpoints{}
469    \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.

```
470    \def\cb@findpdfpoint##1##2{%
471        \edef\cb@temp
472            {\noexpand\cb@@findpdfpoint{\the##1}{\the##2}}%
473        \cb@temp
474    }
```

\pdfliteral For XeTeX we mimick PDFTeX's command \pdfliteral.

```
475    \def\pdfliteral##1{\special{pdf:literal ##1}}
```

\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.

```
476    \def\cb@@findpdfpoint##1##2{%
477        \ifnum##2<\cb@pdfpagenr\relax\else
478            \cb@pdfreadxy{##2}%
479        \fi
480        \let\cb@pdfx\relax
481        \ifx\cb@pdfpoints\@empty\else
482            \ifnum##2<0\relax
483            \else
484                \edef\cb@temp{%
485                    \noexpand\cb@pdffind{##1}{##2}\cb@pdfpoints\relax{}}%
486                \cb@temp
487            \fi
488        \fi
489    }
```

\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.

```
490    \def\cb@pdffind##1##2##3.##4p##5,##6,##7pt##8\relax##9{%
491        \def\cb@next{%
492            \cb@pdffind{##1}{##2}##8\relax{##9##3.##4p##5,##6,##7pt}}%
```

21

```
493     \ifnum ##1=##3
494       \ifnum ##2=##4
495         \def\cb@pdfx{##5sp}%
496         \def\cb@pdfy{##6sp}%
497         \def\cb@pdfz{##7pt}%
498         \let\cb@next\relax
499         \gdef\cb@pdfpoints{##9##8}%
500       \fi
501     \fi
502     \ifx\relax##8\relax
503       \let\cb@next\relax
504     \fi
505     \cb@next
506   }%
```

\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.

```
507   \def\cb@pdfreadxy##1{%
508     \let\cb@next\relax
509     \ifeof\cb@readxy
510       \global\let\cb@pdfpagenr\cb@maxpoint
511     \else
512       {\endlinechar=-1\read\cb@readxy to\cb@temp
513         \ifx\cb@temp\@empty\else
514           \expandafter\cb@pdfparsexy\cb@temp
515           \ifnum\cb@pdfpg<0\else
516             \xdef\cb@pdfpoints{\cb@pdfpoints\cb@temp}%
517             \cb@trace{PDFpoints=\cb@pdfpoints}%
518             \global\let\cb@pdfpagenr\cb@pdfpg
519           \fi
520           \ifnum\cb@pdfpg>##1\else
521             \global\def\cb@next{\cb@pdfreadxy{##1}}%
522           \fi
523         \fi
524       }%
525     \fi
526     \cb@next
527   }%
```

\cb@pdfparsexy The \cb@pdfparsexy macro extracts the pagecount from an entry read in from the .cb2 file.

```
528   \def\cb@pdfparsexy##1.##2p##3,##4,##5pt{%
529     \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.

```
530   \def\cb@defpoint##1##2{%
531     \if@filesw
532       \begingroup
```

```
533         \edef\point{{\the##1}{\the\cb@pagecount}}%
534         \let\the=\z@
535         \pdfsavepos
536         \edef\cb@temp{\write\@auxout
537           {\string\cb@pdfxy\point
538             {\the\pdflastxpos}{\the\pdflastypos}{\the##2}}}%
539         \cb@temp
540       \endgroup
541     \fi
542     \cb@trace@defpoint##1##2%
543   }%
```

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

```
544   \def\cb@cvtpct##1{%
545     \ifnum##1<0 0\else
546     \ifnum##1>99 1\else
547     \ifnum##1<10 0.0\the##1\else
548     0.\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.

```
549 \def\cb@pdf@scale{0.996264009963}
```

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.

```
550   \def\cb@connect##1##2##3{%
551     \cb@findpdfpoint{##1}\cb@pagecount
552     \ifx\cb@pdfx\relax\cb@rerun
553     \else
554       \let\cb@pdftopy\cb@pdfy
555       \cb@findpdfpoint{##2}\cb@pagecount
556       \ifx\cb@pdfx\relax\cb@rerun
557       \else
```

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

```
558         \begingroup
559           \cb@dima=\cb@pdfz
560           \advance\cb@dima by-\cb@pdfx
561           \advance\cb@dima by1in%
562           \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.

```
563           \ifx\cb@current@color\@undefined
564             \def\cb@temp{\cb@cvtpct\c@changebargrey}%
565             \pdfliteral{q \cb@temp\space g \cb@temp\space G}%
566           \else
567             \pdfliteral{q \expandafter\sec@nd@ftw@\cb@current@color\space RG
568                           \expandafter\sec@nd@ftw@\cb@current@color\space rg}%
```

```
569            \fi
570            \edef\cb@temp{\expandafter\cb@removedim\the\cb@dima\space}%
571            \cb@dima=\cb@pdftopy
572            \advance\cb@dima-\cb@pdfy\relax
573            \cb@dima=\cb@pdf@scale\cb@dima\relax
574            ##3=\cb@pdf@scale##3\relax
575            \pdfliteral{\expandafter\cb@removedim\the##3 w
576              \cb@temp 0 m
577              \cb@temp \expandafter\cb@removedim\the\cb@dima\space l S Q}%
578          \endgroup
```

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

```
579          \cb@cntb=##1\relax
580          \ifodd\cb@cntb\advance\cb@cntb 1\else\advance\cb@cntb -1\fi
581          \cb@findpdfpoint\cb@cntb\cb@pagecount
582          \cb@cntb=##2\relax
583          \ifodd\cb@cntb\advance\cb@cntb 1\else\advance\cb@cntb -1\fi
584          \cb@findpdfpoint\cb@cntb\cb@pagecount
585        \fi
586      \fi
587      \cb@trace@connect##1##2##3%
588    }%
```

`\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.

```
589    \gdef\cb@checkPdfxy##1##2##3##4##5{%
590      \cb@@findpdfpoint{##1}{##2}%
591      \ifdim##3sp=\cb@pdfx\relax
592        \ifdim##4sp=\cb@pdfy\relax
593          \ifdim##5sp=\cb@pdfz\relax
594          \else
595          \cb@error
596          \fi
597        \else
598          \cb@error
599        \fi
600      \else
601        \cb@error
602      \fi
603    }
```

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

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

The code for luaTeX, like for pdfTeX and XeTeX, 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.

```
606 \or
607    \immediate\closeout\cb@writexy
608    \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.

```
609  \def\cb@pdfpoints{}
610  \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.

```
611  \def\cb@findpdfpoint##1##2{%
612      \edef\cb@temp
613          {\noexpand\cb@@findpdfpoint{\the##1}{\the##2}}%
614      \cb@temp
615  }
```

`\pdfliteral` For luaTEX we also mimick PDFTEX's command `\pdfliteral`.

```
616  \def\pdfliteral##1{\pdfextension literal {##1}}
```

`\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`.

```
617  \def\cb@@findpdfpoint##1##2{%
618      \ifnum##2<\cb@pdfpagenr\relax\else
619          \cb@pdfreadxy{##2}%
620      \fi
621      \let\cb@pdfx\relax
622      \ifx\cb@pdfpoints\@empty\else
623          \ifnum##2<0\relax
624          \else
625              \edef\cb@temp{%
626                  \noexpand\cb@pdffind{##1}{##2}\cb@pdfpoints\relax{}}%
627              \cb@temp
628          \fi
629      \fi
630  }
```

`\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.

```
631  \def\cb@pdffind##1##2##3.##4p##5,##6,##7pt##8\relax##9{%
632      \def\cb@next{%
633          \cb@pdffind{##1}{##2}##8\relax{##9##3.##4p##5,##6,##7pt}}%
634      \ifnum ##1=##3
635          \ifnum ##2=##4
636              \def\cb@pdfx{##5sp}%
637              \def\cb@pdfy{##6sp}%
638              \def\cb@pdfz{##7pt}%
```

```
639          \let\cb@next\relax
640          \gdef\cb@pdfpoints{##9##8}%
641        \fi
642      \fi
643      \ifx\relax##8\relax
644        \let\cb@next\relax
645      \fi
646      \cb@next
647    }%
```

\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.

```
648    \def\cb@pdfreadxy##1{%
649      \let\cb@next\relax
650      \ifeof\cb@readxy
651        \global\let\cb@pdfpagenr\cb@maxpoint
652      \else
653        {\endlinechar=-1\read\cb@readxy to\cb@temp
654          \ifx\cb@temp\@empty\else
655            \expandafter\cb@pdfparsexy\cb@temp
656            \ifnum\cb@pdfpg<0\else
657              \xdef\cb@pdfpoints{\cb@pdfpoints\cb@temp}%
658              \cb@trace{PDFpoints=\cb@pdfpoints}%
659              \global\let\cb@pdfpagenr\cb@pdfpg
660            \fi
661            \ifnum\cb@pdfpg>##1\else
662              \global\def\cb@next{\cb@pdfreadxy{##1}}%
663            \fi
664          \fi
665        }%
666      \fi
667      \cb@next
668    }%
```

\cb@pdfparsexy The \cb@pdfparsexy macro extracts the pagecount from an entry read in from
the .cb2 file.

```
669    \def\cb@pdfparsexy##1.##2p##3,##4,##5pt{%
670      \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.

```
671    \def\cb@defpoint##1##2{%
672      \if@filesw
673        \begingroup
674          \edef\point{{\the##1}{\the\cb@pagecount}}%
675          \let\the=\z@
676          \savepos
677          \edef\cb@temp{\write\@auxout
```

```
678            {\string\cb@pdfxy\point
679              {\the\lastxpos}{\the\lastypos}{\the##2}}}%
680          \cb@temp
681        \endgroup
682      \fi
683      \cb@trace@defpoint##1##2%
684    }%
```

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

```
685    \def\cb@cvtpct##1{%
686      \ifnum##1<0 0\else
687      \ifnum##1>99 1\else
688      \ifnum##1<10 0.0\the##1\else
689      0.\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.

```
690 \def\cb@pdf@scale{0.996264009963}
```

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.

```
691    \def\cb@connect##1##2##3{%
692      \cb@findpdfpoint{##1}\cb@pagecount
693      \ifx\cb@pdfx\relax\cb@rerun
694      \else
695        \let\cb@pdftopy\cb@pdfy
696        \cb@findpdfpoint{##2}\cb@pagecount
697        \ifx\cb@pdfx\relax\cb@rerun
698        \else
```

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

```
699          \begingroup
700            \cb@dima=\cb@pdfz
701            \advance\cb@dima by-\cb@pdfx
702            \advance\cb@dima by1in%
703            \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.

```
704            \ifx\cb@current@color\@undefined
705              \def\cb@temp{\cb@cvtpct\c@changebargrey}%
706              \pdfliteral{q \cb@temp\space g \cb@temp\space G}%
707            \else
708              \pdfliteral{q \cb@current@color}%
709            \fi
710            \edef\cb@temp{\expandafter\cb@removedim\the\cb@dima\space}%
711            \cb@dima=\cb@pdftopy
712            \advance\cb@dima-\cb@pdfy\relax
```

27

```
713            \cb@dima=\cb@pdf@scale\cb@dima\relax
714            ##3=\cb@pdf@scale##3\relax
715            \pdfliteral{\expandafter\cb@removedim\the##3 w
716              \cb@temp 0 m
717              \cb@temp \expandafter\cb@removedim\the\cb@dima\space l S Q}%
718          \endgroup
```

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

```
719          \cb@cntb=##1\relax
720          \ifodd\cb@cntb\advance\cb@cntb 1\else\advance\cb@cntb -1\fi
721          \cb@findpdfpoint\cb@cntb\cb@pagecount
722          \cb@cntb=##2\relax
723          \ifodd\cb@cntb\advance\cb@cntb 1\else\advance\cb@cntb -1\fi
724          \cb@findpdfpoint\cb@cntb\cb@pagecount
725        \fi
726      \fi
727      \cb@trace@connect##1##2##3%
728    }%
```

`\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.

```
729    \gdef\cb@checkPdfxy##1##2##3##4##5{%
730      \cb@@findpdfpoint{##1}{##2}%
731      \ifdim##3sp=\cb@pdfx\relax
732        \ifdim##4sp=\cb@pdfy\relax
733          \ifdim##5=\cb@pdfz\relax
734          \else
735            \cb@error
736          \fi
737        \else
738          \cb@error
739        \fi
740      \else
741        \cb@error
742      \fi
743    }
```

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

```
744    \def\cb@maxpoint{9999999}
745    \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.

```
746 \else
747   \PackageWarning{Changebar}{changebars not supported in unknown setup}
748   \def\cb@defpoint##1##2{\cb@trace@defpoint##1##2}
749   \def\cb@connect##1##2##3{\cb@trace@connect##1##2##3}
750   \let\cb@resetpoints\relax
751 \fi
```

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

```
752 \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`.

```
753 \newcommand*{\cbstart}{\@ifnextchar [%
754                    {\cb@start}%
755                    {\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.

```
756 \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`.

```
757 \newcommand*{\cbdelete}{\@ifnextchar [%
758   {\cb@delete}%
759   {\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.

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

\changebar The macros `\changebar` and `\endchangebar` have the same function as `\cbstart`
\endchangebar 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.

```
761 \newenvironment{changebar}%
762                    {\@ifnextchar [{\cb@start}%
763                                    {\cb@start[\changebarwidth]}}%
764                    {\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.

```
765 \newcommand*{\nochangebars}{%
766   \def\cb@start[##1]{}%
767   \def\cb@delete[##1]{}%
768   \let\cb@end\relax}
```

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

```
769 \newlength{\changebarwidth}
770 \setlength{\changebarwidth}{2pt}
```

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

```
771 \newlength{\deletebarwidth}
772 \setlength{\deletebarwidth}{4pt}
```

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

```
773 \newlength{\changebarsep}
774 \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).

```
775 \newcounter{changebargrey}
776 \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.

```
777 \@ifpackagewith{changebar}{\csname cb@color@pkg\endcsname}{%
778     \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.

```
779     \DeclareRobustCommand\cbcolor{%
780         \@ifnextchar[\@undeclaredcbcolor\@declaredcbcolor}
```

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

```
781     \def\@undeclaredcbcolor[#1]#2{%
782         \begingroup
783             \color[#1]{#2}%
784             \global\let\cb@current@color\current@color
785         \endgroup
786         \ignorespaces
787         }
```

\@declaredcbcolor

```
788     \def\@declaredcbcolor#1{%
789         \begingroup
790             \color{#1}%
791             \global\let\cb@current@color\current@color
792         \endgroup
793         \ignorespaces}%
794     }{%
```

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

```
795     \def\cbcolor{\@ifnextchar[%]
796         \@@cbcolor\@cbcolor}%
797     \def\@@cbcolor[#1]#2{\cb@colwarn\def\@@cbcolor[##1]##2{}}%
798     \def\@cbcolor#1{\cb@colwarn\def\@cbcolor##1{}}%
799     \def\cb@colwarn{\PackageWarning{Changebar}%
800         {You didn't specify the option 'color';\MessageBreak
801             your command \string\cbcolor\space will be ignored}}%
802     }
```

30

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.

```
803 \def\cb@start[#1]{%
804    \cb@topleft=\cb@nextpoint
```

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

```
805    \cb@curbarwd#1\relax
806    \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.

```
807    \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.

```
808    \cb@cnta\cb@pagecount
809    \cb@pagecount\cb@page\advance\cb@pagecount\m@ne
810    \ifvmode
811       \cb@setBeginPoints
812    \else
813       \vbox to \z@{%
```

When we are in horizontal mode we jump up a line to set the starting point of the changebar.

```
814          \vskip -\ht\strutbox
815          \cb@setBeginPoints
816          \vskip \ht\strutbox}%
817    \fi
```

Restore \cb@pagecount.

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

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

```
820 \def\cb@advancePoint{%
821    \global\advance\cb@nextpoint by 4\relax
822    \ifnum\cb@nextpoint>\cb@maxpoint
823       \global\cb@nextpoint=\cb@minpoint\relax
824    \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.

```
825 \def\cb@end{%
```

```
826    \cb@trace@stack{end of bar on page \the\c@page}%
827    \cb@pop\cb@currentstack
828    \ifnum\cb@topleft=\cb@nil
829      \PackageWarning{Changebar}%
830        {Badly nested changebars; Expect erroneous results}%
831    \else
```

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

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

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

```
833      \cb@cnta\cb@pagecount
834      \cb@pagecount\cb@page\advance\cb@pagecount\m@ne
835      \ifvmode
836        \cb@setEndPoints
837      \else
838        \vadjust{\cb@setEndPoints}%
839      \fi
840      \cb@pagecount\cb@cnta
841    \fi
842    \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).

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

First store the identifiers in temporary registers.

```
844    \cb@cnta\cb@topleft\relax
845    \advance\cb@cnta by #1\relax
846    \cb@cntb\cb@page\relax
847    \cb@dima\cb@curbarwd\relax
```

Then pop the history stack.

```
848    \cb@pop\cb@historystack
```

If it was empty there is nothing to check and we're done.

```
849    \ifnum\cb@topleft=\cb@nil
850    \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.

```
851      \cb@FindPageNum
852      \ifnum\cb@topleft>\cb@maxpoint\else
```

Now that we've found it overwrite `\cb@cntb` with the `\cb@page` from the stack.

```
853        \cb@cntb\cb@page
854      \fi
```

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

```
855      \@whilenum\cb@topleft>\cb@nil\do{%
856        \cb@push\cb@historystack
857        \cb@pop\cb@tempstack}%
858    \fi
```

Finally return the correct values

```
859    \advance\cb@cnta by -#1\relax
860    \cb@topleft\cb@cnta\relax
861    \cb@page\cb@cntb\relax
862    \cb@curbarwd\cb@dima\relax
863    }
```

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

```
864 \def\cb@FindPageNum{%
865    \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.

```
866    \else
867      \cb@push\cb@tempstack
868      \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.

```
869      \ifnum\cb@topleft=\cb@nil
870        \cb@trace{Ran out of history stack, new changebar?}%
```

In this case we give \cb@topleft an 'impossible value' to remember this special situation.

```
871        \cb@topleft\cb@maxpoint\advance\cb@topleft\@ne
872      \else
```

Recursively call ourselves.

```
873        \expandafter\expandafter\expandafter\cb@FindPageNum
874      \fi
875    \fi
876 }%
```

\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.

```
877 \def\cb@setBeginPoints{%
878    \cb@topright=\cb@topleft\advance\cb@topright by\@ne
879    \cb@cntb=\cb@pagecount
880    \divide\cb@cntb by\tw@
881    \ifodd\cb@cntb
882      \cb@defpoint\cb@topleft\cb@even@left
883      \cb@defpoint\cb@topright\cb@even@right
884    \else
885      \cb@defpoint\cb@topleft\cb@odd@left
886      \cb@defpoint\cb@topright\cb@odd@right
887    \fi
888    \cb@writeAux\cb@topleft
889    }
```

\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.

```
890 \def\cb@setEndPoints{%
891   \cb@topright=\cb@topleft\advance\cb@topright by\@ne
892   \cb@botleft=\cb@topleft\advance\cb@botleft by\tw@
893   \cb@botright=\cb@topleft\advance\cb@botright by\thr@@
894    \cb@cntb=\cb@pagecount
895   \divide\cb@cntb by\tw@
896   \ifodd\cb@cntb
897     \cb@defpoint\cb@botleft\cb@even@left
898     \cb@defpoint\cb@botright\cb@even@right
899   \else
900     \cb@defpoint\cb@botleft\cb@odd@left
901     \cb@defpoint\cb@botright\cb@odd@right
902   \fi
903   \cb@writeAux\cb@botright
904   \edef\cb@leftbar{%
905     \noexpand\cb@connect{\cb@topleft}{\cb@botleft}{\cb@curbarwd}}%
906   \edef\cb@rightbar{%
907     \noexpand\cb@connect{\cb@topright}{\cb@botright}{\cb@curbarwd}}%
```

In twocolumn pages always use outerbars

```
908   \if@twocolumn
909       \ifodd\cb@pagecount\cb@rightbar\else\cb@leftbar\fi
```

```
910   \else
911     \ifcase\cb@barsplace
```

0=innerbars

```
912       \ifodd\cb@cntb
913         \cb@rightbar
914       \else
915         \if@twoside\cb@leftbar\else\cb@rightbar\fi
916       \fi
917     \or
```

1=outerbars

```
918       \ifodd\cb@cntb
919         \cb@leftbar
920       \else
921         \if@twoside\cb@rightbar\else\cb@leftbar\fi
922       \fi
923     \or
```

2=leftbars

```
924       \cb@leftbar
925     \or
```

3=rightbars

```
926       \cb@rightbar
927     \fi
928   \fi
929   }%
```

\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.

```
930 \def\cb@writeAux#1{%
931   \if@filesw
932     \begingroup
933       \edef\point{\the#1}%
934       \edef\level{\the\cb@curbarwd}%
935       \let\the=\z@
936       \edef\cb@temp{\write\@auxout
937         {\string\cb@barpoint{\point}{\the\cb@pagecount}{\level}}}%
938       \cb@temp
939     \endgroup
940   \fi}
```

5.5   Macros for Making It Work Across Page Breaks

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

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

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

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

\cb@nextpagejump The next pagecount from the list.

```
943 \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.

```
944 \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.

```
945 \def\cb@writepagejump#1{
946   \cb@cntb=\cb@pagecount
947   \advance\cb@cntb by#1\relax
948   \global\setbox\@leftcolumn\vbox to\@colht{%
949     \edef\cb@temp{\write\@auxout{\string\cb@pagejump{\the\cb@cntb}}}%
950     \cb@temp
951     \dimen@ \dp\@leftcolumn
952     \unvbox \@leftcolumn
953     \vskip -\dimen@
954   }%
955 }
```

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

```
956 \def\cb@poppagejump#1,#2\relax{%
957   \gdef\cb@nextpagejump{#1}%
958   \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.

```
959 \def\cb@checkpagecount{%
960   \if@twocolumn
961     \if@firstcolumn
962       \ifodd\cb@pagecount\global\advance\cb@pagecount by\@ne\fi
963     \fi
964   \else
965     \ifodd\cb@pagecount\global\advance\cb@pagecount by\@ne\fi
966   \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.

```
967   \if@twoside
968     \cb@cntb=\cb@pagecount
969     \divide\cb@cntb by\tw@
970     \advance\cb@cntb by-\c@page
971     \ifodd\cb@cntb
```

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

```
972       \if@twocolumn
973         \if@firstcolumn
974           \@whilenum\cb@pagecount>\cb@nextpagejump\do{%
975             \expandafter\cb@poppagejump\cb@pagejumplst\relax}%
976           \ifnum\cb@pagecount=\cb@nextpagejump
977             \cb@trace{Page jump: \string\cb@pagecount=\the\cb@pagecount}
978             \global\advance\cb@pagecount by\tw@
979             \global\@cb@pagejumptrue
980           \else
981             \global\@cb@pagejumpfalse
982           \fi
983         \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.

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

```
987        \fi
988      \fi
989    \else
```

Here `\cb@pagecount` is incorrect.

```
990      \if@twocolumn
991        \if@firstcolumn
992          \global\advance\cb@pagecount by\tw@
993          \global\@cb@pagejumpfalse
994        \else
995          \if@cb@pagejump
996            \cb@trace{Page jump annulled, %
997                      \string\cb@pagecount=\the\cb@pagecount}
998          \else
999            \cb@writepagejump{-1}%
1000           \global\advance\cb@pagecount by\tw@
1001           \cb@rerun
1002         \fi
1003       \fi
1004     \else
1005       \global\advance\cb@pagecount by\tw@
1006     \fi
1007   \fi
1008 \fi
1009 }
```

`\@makecol` These internal LATEX macros are modified in order to end the changebars span-
`\@vtryfc` ning 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.

```
1010 \let\ltx@makecol\@makecol
1011 \def\cb@makecol{%
1012   \if@twocolumn
1013     \cb@trace{Twocolumn: \if@firstcolumn Left \else Right \fi column}%
1014   \fi
1015   \cb@trace@stack{before makecol, page \the\c@page,
1016                   \string\cb@pagecount=\the\cb@pagecount}%
1017   \let\cb@writeAux\@gobble
```

First make sure that `\cb@pagecount` is correct. Then add the necessary bar points
at beginning and end.

```
1018   \cb@checkpagecount
1019   \setbox\@cclv \vbox{%
1020     \cb@resetpoints
1021     \cb@startSpanBars
1022     \unvbox\@cclv
1023     \boxmaxdepth\maxdepth}%
1024   \global\advance\cb@pagecount by\@ne
1025   \cb@buildstack\cb@processActive
```

```
1026    \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`.

```
1027      \if@twocolumn
1028        \global\setbox\@outputbox \vbox to\@colht {%
1029          \if@firstcolumn\write\@auxout{\string\@cb@firstcolumntrue}%
1030          \else\write\@auxout{\string\@cb@firstcolumnfalse}%
1031          \fi
1032          \dimen@ \dp\@outputbox
1033          \unvbox \@outputbox
1034          \vskip -\dimen@
1035          }%
1036      \fi
1037      \cb@trace@stack{after makecol, page \the\c@page,
1038                     \string\cb@pagecount=\the\cb@pagecount}%
1039    }
1040 \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.

```
1041 \let\ltx@vtryfc\@vtryfc
1042 \def\cb@vtryfc#1{%
1043    \cb@trace{In vtryfc, page \the\c@page,
1044                     \string\cb@pagecount=\the\cb@pagecount}%
1045    \let\cb@writeAux\@gobble
```

First make sure that `\cb@pagecount` is correct. Then generate a `\@cb@firstcolumntrue` or `\@cb@firstcolumnfalse` in twocolumn mode.

```
1046    \cb@checkpagecount
1047    \ltx@vtryfc{#1}%
1048    \if@twocolumn
1049      \global\setbox\@outputbox \vbox to\@colht{%
1050        \if@firstcolumn\write\@auxout{\string\@cb@firstcolumntrue}%
1051        \else\write\@auxout{\string\@cb@firstcolumnfalse}%
1052        \fi
1053        \unvbox\@outputbox
1054        \boxmaxdepth\maxdepth
1055      }%
1056    \fi
1057    \global\advance\cb@pagecount by \@ne
1058 }
1059 \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).
```

```
1060 \def\cb@processActive{%
1061   \cb@pop\cb@endstack
1062   \ifnum\cb@topleft=\cb@nil
1063   \else
1064     \setbox\@cclv\vbox{%
1065       \unvbox\@cclv
1066       \boxmaxdepth\maxdepth
1067       \advance\cb@pagecount by -1\relax
1068       \cb@setEndPoints}%
1069     \cb@push\cb@historystack

1070     \cb@push\cb@beginstack
1071     \expandafter\cb@processActive
1072   \fi}
```

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

```
1073 \def\cb@startSpanBars{%
1074   \cb@pop\cb@beginstack
1075   \ifnum\cb@topleft=\cb@nil
1076   \else
1077     \cb@setBeginPoints
1078     \cb@trace@stack{after StartSpanBars, page \the\c@page}%
1079     \expandafter\cb@startSpanBars
1080   \fi
1081   }
```

\cb@buildstack   The macro \cb@buildstack initializes the stack with open bars and starts popu-
\cb@endstack   lating it.

```
1082 \def\cb@buildstack{%
1083   \cb@initstack\cb@endstack
1084   \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.

```
1085 \def\cb@pushNextActive{%
1086   \cb@pop\cb@historystack
1087   \ifnum\cb@topleft=\cb@nil
1088   \else
1089     \ifnum\cb@page>\cb@pagecount
1090       \cb@push\cb@historystack
1091     \else
1092       \ifodd\cb@topleft
1093         \cb@push\cb@endstack
1094       \else
```

39

```
1095        \cb@pop\cb@endstack
1096      \fi
1097      \expandafter\expandafter\expandafter\cb@pushNextActive
1098    \fi
1099  \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.

```
1100 \def\cb@initstack#1{\xdef#1{}}
```

\cb@historystack We need to initialise a stack to store the entries read from the external history
\cb@write file.
\cb@read
```
1101 \cb@initstack\cb@historystack
```

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

```
1102 \newwrite\cb@write
1103 \newread\cb@read
```

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

```
1104 \immediate\openout\cb@write=\jobname.cb\relax
```

\cb@endstack Allocate two stacks for the bars that span the current page break.
\cb@beginstack
```
1105 \cb@initstack\cb@endstack
1106 \cb@initstack\cb@beginstack
```

\cb@tempstack Allocate a stack for temporary storage

```
1107 \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.

```
1108 \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.

```
1109 \def\cb@thehistorystack{\cb@historystack}
1110 \def\cb@pop#1{%
1111   \ifx #1\@empty
1112     \def\cb@temp{#1}%
1113     \ifx\cb@temp\cb@thehistorystack
1114       \ifeof\cb@read
1115       \else
1116         {\endlinechar=-1\read\cb@read to\cb@temp
1117          \xdef\cb@historystack{\cb@historystack\cb@temp}%
1118         }%
1119       \fi
1120     \fi
1121   \fi
1122   \ifx#1\@empty
1123     \global\cb@topleft\cb@nil
1124     \global\cb@page\z@\relax
1125   \else
1126     \expandafter\cb@carcdr#1e#1%
1127   \fi
1128   \cb@trace@pop{#1}}
```

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

```
1129 \def\cb@carcdr#1n#2p#3l#4e#5{%
1130   \global\cb@topleft#1\relax
1131   \global\cb@page#2\relax
1132   \global\cb@curbarwd#3\relax
1133   \xdef#5{#4}}
```

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

```
1134 \def\cb@push#1{%
1135   \xdef#1{\the\cb@topleft n\the\cb@page p\the\cb@curbarwd l#1}%
1136   \cb@trace@push{#1}}
1137
```

\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.

```
1138 \def\cb@barpoint#1#2#3{\cb@cnta=#2
1139   \if@cb@firstcolumn\advance\cb@cnta by\m@ne\fi
1140   \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

41

`\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.

```
1141 \AtBeginDocument{%
1142   \cb@setup@specials
```

Add a sentinel to `\cb@pagejumplst`.

```
1143   \cb@pagejump{999999999,}%
```

Compute the left and right positions of the changebars.

```
1144   \cb@positions
1145   \cb@trace{%
1146     Odd left  : \the\cb@odd@left\space
1147     Odd right : \the\cb@odd@right\MessageBreak
1148     Even left: \the\cb@even@left\space
1149     Even right: \the\cb@even@right
1150     }%
1151   \immediate\closeout\cb@write
1152   \immediate\openin\cb@read=\jobname.cb\relax}
```

`\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.

```
1153 \AtEndDocument{%
1154   \clearpage
1155   \cb@initstack\cb@historystack
1156   \immediate\closein\cb@read
1157   \immediate\openin\cb@read=\jobname.cb\relax
```

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

```
1158   \ifx\cb@readxy\@undefined
1159   \else
1160     \immediate\closein\cb@readxy
1161     \immediate\openin\cb@readxy=\jobname.cb2\relax
1162     \def\cb@pdfpoints{}%
1163     \def\cb@pdfpagenr{0}%
1164   \fi
1165   \@cb@firstcolumnfalse
1166   \cb@checkrerun
1167   \let\cb@pdfxy\cb@checkPdfxy
1168   \let\cb@pagejump\@gobble
1169   \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.

```
1170 \def\cb@checkHistory#1#2#3{%
1171   \cb@pop\cb@historystack
1172   \ifnum #1=\cb@topleft\relax
1173     \cb@cnta=#2
1174     \if@cb@firstcolumn\advance\cb@cnta by\m@ne\fi
1175     \ifnum \cb@cnta=\cb@page\relax
```

Both page and point numbers are equal; do nothing,

```
1176     \else
```

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

```
1177       \cb@error
1178     \fi
1179   \else
```

when point numbers don't match.

```
1180     \cb@error
1181   \fi}
```

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

```
1182 \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.

```
1183 \def\cb@rerun{%
1184   \global\let\cb@checkrerun\cb@error}
1185 \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 pdfTeXand XeTeXwe also disable `\cb@checkPdfxy`.

```
1186 \def\cb@error{%
1187   \PackageWarning{Changebar}%
1188     {Changebar info has changed.\MessageBreak
1189      Rerun to get the bars right}
1190   \gdef\cb@checkHistory##1##2##3{}%
1191   \let\cb@barpoint\cb@checkHistory
1192   \gdef\cb@checkPdfxy##1##2##3##4##5{}%
1193   \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`.

```
1194 \let\ltx@end@float\end@float
```

```
1195 \def\cb@end@float{%
1196   \cb@trace@stack{end float on page \the\c@page}%
1197   \cb@pop\cb@currentstack
1198   \ifnum\cb@topleft=\cb@nil
1199   \else
1200     \cb@push\cb@currentstack
1201     \global\cb@curbarwd=\cb@curbarwd
1202     \@endfloatbox
1203     \global\setbox\@currbox
1204       \color@vbox
1205       \normalcolor
1206       \vbox\bgroup\cb@start[\cb@curbarwd]\unvbox\@currbox\cb@end
1207   \fi
1208   \ltx@end@float}
1209 \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 LaTeX2ε class files use \newenvironment to define the figure and table environments. In that case there is no need to rebind \endfigure.

\@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 the this macro uses the original definition of \end@float.

```
1210 \let\ltx@@xympar\@xympar
1211 \def\@xympar{%
1212   \let\end@float\ltx@end@float
1213   \ltx@@xympar
1214   \let\end@float\cb@end@float}
```

\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.

```
1215 \let\flt@float@end\float@end
```

Then we redefine it to insert the changebarcode.

```
1216 \def\float@end{%
1217   \cb@trace@stack{end float on page \the\c@page}%
1218   \cb@pop\cb@currentstack
1219   \ifnum\cb@topleft=\cb@nil
1220   \else
1221     \cb@push\cb@currentstack
1222     \global\cb@curbarwd\cb@curbarwd
1223     \@endfloatbox
1224     \global\setbox\@currbox
1225       \color@vbox
1226       \normalcolor
1227       \vbox\bgroup\cb@start[\cb@curbarwd]\unvbox\@currbox\cb@end
1228   \fi
1229   \let\end@float\ltx@end@float
1230   \flt@float@end
1231   }
```

\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.

```
1232 \let\ltx@end@dblfloat\end@dblfloat
1233 \def\cb@end@dblfloat{%
1234   \if@twocolumn
1235     \cb@trace@stack{end dblfloat on page \the\c@page}%
1236     \cb@pop\cb@currentstack
1237     \ifnum\cb@topleft=\cb@nil
```

```
1238     \else
1239       \cb@push\cb@currentstack
1240       \global\cb@curbarwd=\cb@curbarwd
1241       \@endfloatbox
1242       \global\setbox\@currbox
1243         \color@vbox
1244         \normalcolor
1245         \vbox\bgroup\cb@start[\cb@curbarwd]\unvbox\@currbox\cb@end
1246     \fi
1247     \@endfloatbox
1248     \ifnum\@floatpenalty <\z@
1249       \@largefloatcheck
1250       \@cons\@dbldeferlist\@currbox
1251     \fi
1252     \ifnum \@floatpenalty =-\@Mii \@Esphack\fi
1253   \else
1254     \end@float
1255   \fi}
1256 \let\end@dblfloat\cb@end@dblfloat
```

\float@dblend Something similar needs to be done for the case where the `float` package is being used...

```
1257 \let\flt@float@dblend\float@dblend
1258 \def\float@dblend{%
1259   \cb@trace@stack{end dbl float on page \the\c@page}%
1260   \cb@pop\cb@currentstack
1261   \ifnum\cb@topleft=\cb@nil
1262   \else
1263     \cb@push\cb@currentstack
1264     \global\cb@curbarwd=\cb@curbarwd
1265     \@endfloatbox
1266     \global\setbox\@currbox
1267       \color@vbox
1268       \normalcolor
1269       \vbox\bgroup\cb@start[\cb@curbarwd]\unvbox\@currbox\cb@end
1270   \fi
1271   \let\end@dblfloat\ltx@end@dblfloat
1272   \flt@float@dblend
1273   }
```

\@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.

```
1274 \let\ltx@footnotetext\@footnotetext
```

```
1275 \long\def\cb@footnotetext#1{%
1276   \cb@trace@stack{end footnote on page \the\c@page}%
1277   \cb@pop\cb@currentstack
1278   \ifnum\cb@topleft=\cb@nil
1279     \ltx@footnotetext{#1}%
1280   \else
1281     \cb@push\cb@currentstack
1282     \edef\cb@temp{\the\cb@curbarwd}%
1283     \ltx@footnotetext{\cb@start[\cb@temp]#1\cb@end}%
```

```
1284    \fi}
1285 \let\@footnotetext\cb@footnotetext
```

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

```
1286 \let\ltx@mpfootnotetext\@mpfootnotetext
```

```
1287 \long\def\cb@mpfootnotetext#1{%
1288    \cb@pop\cb@currentstack
1289    \ifnum\cb@topleft=\cb@nil
1290      \ltx@mpfootnotetext{#1}%
1291    \else
1292      \cb@push\cb@currentstack
1293      \edef\cb@temp{\the\cb@curbarwd}%
1294      \ltx@mpfootnotetext{\cb@start[\cb@temp]#1\cb@end}%
1295    \fi}
1296 \let\@mpfootnotetext\cb@mpfootnotetext
1297 ⟨/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.

47

48