The Changebar package *

Michael Fine
Distributed Systems Architecture

Johannes Braams
`johannes.braams at texniek.nl`

Printed January 9, 2024

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, XeTeX and luaTeX support. More can easily be added.

1 Introduction

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

Features

---

*This file has version number v3.7d, last revised 2024/01/09.

- 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_ε 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 PDFTeX or 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<sub>E</sub>X

- PDFT<sub>E</sub>X

- XeT<sub>E</sub>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_EX 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}
```

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}
```

Give a warning if we cannot support the xetex option.

```
99  \def\cb@xetexerror{\PackageError
100       {changebar}%
101       {XeTeX option cannot be used}%
102       {You are not using XeLaTeX}}
```

```
103 \DeclareOption{luaTeX}{\cb@luatexcheck}
104 \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..

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

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

```
114     \gdef\cb@pdfxy##1##2##3##4##5{%
115       \immediate\write\cb@writexy{##1.##2p##3;##4;##5}%
116       \expandafter\gdef\csname cb@##1.##2\endcsname{##3,##4,##5}}
117   \fi}
```

Give a warning if we cannot support the luatex option.

```
118 \def\cb@luatexerror{\PackageError
119       {changebar}%
120       {luaTeX option cannot be used}%
121       {You are not using luaLaTeX}}
```

The new features of LATEX 2$_\varepsilon$ make it possible to implement the outerbars option.

```
122 \DeclareOption{outerbars}{\def\cb@barsplace{1}}
123 \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.

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

A set of options to control tracing.

```
126 \DeclareOption{traceon}{\@cb@tracetrue}
127 \DeclareOption{traceoff}{\@cb@tracefalse}
128 \DeclareOption{tracestacks}{%
129   \let\cb@trace@stack\cb@@show@stack
130   \def\cb@trace@push#1{\cb@trace{%
131     Pushed point \the\cb@topleft\space on \noexpand#1: #1}}%
132   \def\cb@trace@pop#1{\cb@trace{%
133     Popped point \the\cb@topleft\space from \noexpand#1: #1}}%
134 }
```

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

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

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

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

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

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

Signal an error if an unknown option was specified.

```
143 \DeclareOption*{\OptionNotUsed\PackageError
144     {changebar}%
145     {Unrecognised option '\CurrentOption'\MessageBreak
146       known options are dvitoln03, dvitops, dvips,\MessageBreak
147       emtex, textures, pdftex, vtex and xetex,
148       grey, color, xcolor,\MessageBreak
149       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.

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

```
157      \else
158        \ExecuteOptions{innerbars,traceoff,dvips,grey}
159      \fi
160    \fi
161  \else
162    \ExecuteOptions{innerbars,traceoff,xetex,grey}
163  \fi
164 \else
165    \ExecuteOptions{innerbars,traceoff,vtex,grey}
166 \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.

```
167 \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.

```
168 \def\cb@@show@stack#1{%
169   \cb@trace{%
170     stack status at #1:\MessageBreak
171     current stack: \cb@currentstack\MessageBreak
172     \@spaces end stack: \cb@endstack\MessageBreak
173     \space\space begin stack: \cb@beginstack\MessageBreak
174     history stack: \cb@historystack
175     }}
```

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

```
176 \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.

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

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

```
179 \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.

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

```
187   \fi
188 \fi
```

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

```
189 \newcommand{\cb@trace}[1]{%
190   \if@cb@trace
191     \GenericWarning
192       {(changebar)\@spaces\@spaces}%
193       {Package changebar: #1\@gobble}%
194   \fi
195   }
```

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.

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

207       \def\xeTeX{XETEX}

208       \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`.

```
209        \global\chardef\cb@driver@setup=0\relax
210        \ifx\tempa\LN         \global\chardef\cb@driver@setup=0\fi
211        \ifx\tempa\DVItoPS    \global\chardef\cb@driver@setup=1\fi
212        \ifx\tempa\DVIPS      \global\chardef\cb@driver@setup=2\fi
213        \ifx\tempa\emTeX      \global\chardef\cb@driver@setup=3\fi
214        \ifx\tempa\Textures \global\chardef\cb@driver@setup=4\fi
215        \ifx\tempa\VTeX       \global\chardef\cb@driver@setup=5\fi
216        \ifx\tempa\pdfTeX     \cb@pdftexcheck\fi
217        \ifx\tempa\xeTeX      \cb@xetexcheck\fi

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

219      \egroup}
```

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

```
220    {\def\do{\noexpand\do\noexpand}
221      \xdef\@preamblecmds{\@preamblecmds \do\driver}
222      }
223 \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.

```
224 \def\cb@trace@defpoint#1#2{%
225   \cb@trace{%
226     defining point \the#1 at position \the#2
227     \MessageBreak
228     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.

```
229 \def\cb@trace@connect#1#2#3{%
230   \cb@trace{%
231     connecting points \the#1 and \the#2; barwidth: \the#3
232     \MessageBreak
233     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⟩}`

14

The two ⟨*number*⟩s indicate the two points to be connected; the ⟨*length*⟩ is the width of the bar.

```
234 \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.

```
235 \ifcase\cb@driver@setup
236   \def\cb@defpoint##1##2{%
237     \special{ln03:defpoint \the##1(\the##2,)}%
238     \cb@trace@defpoint##1##2}
239   \def\cb@connect##1##2##3{%
240     \special{ln03:connect \the##1\space\space \the##2\space \the##3}%
241     \cb@trace@connect##1##2##3}
242   \def\cb@resetpoints{%
243     \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.

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

251   \def\cb@connect##1##2##3{%
252     \special{dvitops: inline
253               gsave \cb@ps@color\space
254               \expandafter\cb@removedim\the##3\space 6.5536 mul\space
255               CBarX\the##1\space\space CBarY\the##1\space\space moveto
256               CBarX\the##2\space\space CBarY\the##2\space\space lineto
257               stroke grestore}%
258     \cb@trace@connect##1##2##3}
259   \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.

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

```
268    \def\cb@connect##1##2##3{%
269      \special{ps:
270                    gsave \cb@ps@color\space
271                    \expandafter\cb@removedim\the##3\space
272                    Resolution\space mul\space 72.27\space div\space
273                    setlinewidth
274                    CBarX\the##1\space\space CBarY\the##1\space\space moveto
275                    CBarX\the##2\space\space CBarY\the##2\space\space lineto
276                    stroke grestore}%
277      \cb@trace@connect##1##2##3}
278    \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.

```
279 \or
280    \def\cb@defpoint##1##2{%
281      \special{em:point \the##1,\the##2}%
282      \cb@trace@defpoint##1##2}
283    \def\cb@connect##1##2##3{%
284      \special{em:line \the##1,\the##2,\the##3}%
285      \cb@trace@connect##1##2##3}
286    \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.

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

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

The following definitions were kindly provided by Michael Vulis.

```
306 \or
307    \def\cb@defpoint##1##2{%
308      \special{pS:
309                    \expandafter\cb@removedim\the##2\space
310                    Resolution\space mul\space 72.27\space div\space
```

16

```
311              /CBarX\the##1\space exch def currentpoint exch pop
312              /CBarY\the##1\space exch def}%
313       \cb@trace@defpoint##1##2}
314   \def\cb@connect##1##2##3{%
315     \special{pS:
316              gsave \cb@ps@color\space
317              \expandafter\cb@removedim\the##3\space
318              Resolution\space mul\space 72.27\space div\space
319              setlinewidth
320              CBarX\the##1\space\space CBarY\the##1\space\space moveto
321              CBarX\the##2\space\space CBarY\the##2\space\space lineto
322              stroke grestore}%
323       \cb@trace@connect##1##2##3}
324   \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.

```
325 \or
326   \immediate\closeout\cb@writexy
327   \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.

```
328   \def\cb@pdfpoints{}
329   \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.

```
330   \def\cb@findpdfpoint##1##2{%
331     \edef\cb@temp
332       {\noexpand\cb@@findpdfpoint{\the##1}{\the##2}}%
333     \cb@temp
334   }
```

\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.

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

17

```
346      \fi
347   }
```

\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.

```
348    \def\cb@pdffind##1##2##3.##4p##5;##6;##7pt##8\relax##9{%
349      \def\cb@next{%
350        \cb@pdffind{##1}{##2}##8\relax{##9##3.##4p##5;##6;##7pt}}%
351      \ifnum ##1=##3
352        \ifnum ##2=##4
353          \def\cb@pdfx{##5sp}%
354          \def\cb@pdfy{##6sp}%
355          \def\cb@pdfz{##7pt}%
356          \let\cb@next\relax
357          \gdef\cb@pdfpoints{##9##8}%
358        \fi
359      \fi
360      \ifx\relax##8\relax
361        \let\cb@next\relax
362      \fi
363      \cb@next
364   }%
```

\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.

```
365    \def\cb@pdfreadxy##1{%
366      \let\cb@next\relax
367      \ifeof\cb@readxy
368        \global\let\cb@pdfpagenr\cb@maxpoint
369      \else
370        {\endlinechar=-1\read\cb@readxy to\cb@temp
371          \ifx\cb@temp\@empty\else
372            \expandafter\cb@pdfparsexy\cb@temp
373            \ifnum\cb@pdfpg<0\else
374              \xdef\cb@pdfpoints{\cb@pdfpoints\cb@temp}%
375              \cb@trace{PDFpoints=\cb@pdfpoints}%
376              \global\let\cb@pdfpagenr\cb@pdfpg
377            \fi
378            \ifnum\cb@pdfpg>##1\else
379              \global\def\cb@next{\cb@pdfreadxy{##1}}%
380            \fi
381          \fi
382        }%
383      \fi
384      \cb@next
385   }%
```

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

```
386    \def\cb@pdfparsexy##1.##2p##3;##4;##5pt{%
387      \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.

```
388    \def\cb@defpoint##1##2{%
389      \if@filesw
390        \begingroup
391          \edef\point{{\the##1}{\the\cb@pagecount}}%
392          \let\the=\z@
393          \pdfsavepos
394          \edef\cb@temp{\write\@auxout
395            {\string\cb@pdfxy\point
396              {\the\pdflastxpos}{\the\pdflastypos}{\the##2}}}%
397          \cb@temp
398        \endgroup
399      \fi
400      \cb@trace@defpoint##1##2%
401    }%
```

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

```
402    \def\cb@cvtpct##1{%
403      \ifnum##1<0 0\else
404      \ifnum##1>99 1\else
405      \ifnum##1<10 0.0\the##1\else
406      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.

```
407 \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.

```
408    \def\cb@connect##1##2##3{%
409      \cb@findpdfpoint{##1}\cb@pagecount
410      \ifx\cb@pdfx\relax\cb@rerun
411      \else
412        \let\cb@pdftopy\cb@pdfy
413        \cb@findpdfpoint{##2}\cb@pagecount
414        \ifx\cb@pdfx\relax\cb@rerun
415        \else
```

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

```
416          \begingroup
417            \cb@dima=\cb@pdfz
418            \advance\cb@dima by-\cb@pdfx
419            \advance\cb@dima by1in%
420            \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.

```
421                \ifx\cb@current@color\@undefined
422                  \def\cb@temp{\cb@cvtpct\c@changebargrey}%
423                  \pdfliteral{q \cb@temp\space g \cb@temp\space G}%
424                \else
425                  \pdfliteral{q \cb@current@color}%
426                \fi
427                \edef\cb@temp{\expandafter\cb@removedim\the\cb@dima\space}%
428                \cb@dima=\cb@pdftopy
429                \advance\cb@dima-\cb@pdfy\relax
430                \cb@dima=\cb@pdf@scale\cb@dima\relax
431                ##3=\cb@pdf@scale##3\relax
432                \pdfliteral direct{\expandafter\cb@removedim\the##3 w
433                  \cb@temp 0 m
434                  \cb@temp \expandafter\cb@removedim\the\cb@dima\space l S Q}%
435              \endgroup
```

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

```
436                \cb@cntb=##1\relax
437                \ifodd\cb@cntb\advance\cb@cntb 1\else\advance\cb@cntb -1\fi
438                \cb@findpdfpoint\cb@cntb\cb@pagecount
439                \cb@cntb=##2\relax
440                \ifodd\cb@cntb\advance\cb@cntb 1\else\advance\cb@cntb -1\fi
441                \cb@findpdfpoint\cb@cntb\cb@pagecount
442              \fi
443            \fi
444            \cb@trace@connect##1##2##3%
445          }%
```

\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.

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

```
466    }
```

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

```
467    \def\cb@maxpoint{9999999}
468    \let\cb@resetpoints\relax
469 \or
```

The code for XeT_EX is, like for PDFT_EX, more elaborate as the calculations have to be done in T_EX. \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.

```
470    \immediate\closeout\cb@writexy
471    \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.

```
472    \def\cb@pdfpoints{}
473    \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.

```
474    \def\cb@findpdfpoint##1##2{%
475        \edef\cb@temp
476            {\noexpand\cb@@findpdfpoint{\the##1}{\the##2}}%
477        \cb@temp
478    }
```

\pdfliteral For XeT_EX we mimick PDFT_EX's command \pdfliteral.

```
479    \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.

```
480    \def\cb@@findpdfpoint##1##2{%
481      \ifnum##2<\cb@pdfpagenr\relax\else
482        \cb@pdfreadxy{##2}%
483      \fi
484      \let\cb@pdfx\relax
485      \ifx\cb@pdfpoints\@empty\else
486        \ifnum##2<0\relax
487        \else
488          \edef\cb@temp{%
489            \noexpand\cb@pdffind{##1}{##2}\cb@pdfpoints\relax{}}%
490          \cb@temp
491        \fi
492      \fi
493    }
```

\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.

```
494    \def\cb@pdffind##1##2##3.##4p##5;##6;##7pt##8\relax##9{%
495      \def\cb@next{%
496        \cb@pdffind{##1}{##2}##8\relax{##9##3.##4p##5;##6;##7pt}}%
497      \ifnum ##1=##3
498        \ifnum ##2=##4
499          \def\cb@pdfx{##5sp}%
500          \def\cb@pdfy{##6sp}%
501          \def\cb@pdfz{##7pt}%
502          \let\cb@next\relax
503          \gdef\cb@pdfpoints{##9##8}%
504        \fi
505      \fi
506      \ifx\relax##8\relax
507        \let\cb@next\relax
508      \fi
509      \cb@next
510    }%
```

\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.

```
511    \def\cb@pdfreadxy##1{%
512      \let\cb@next\relax
513      \ifeof\cb@readxy
514        \global\let\cb@pdfpagenr\cb@maxpoint
515      \else
516        {\endlinechar=-1\read\cb@readxy to\cb@temp
517          \ifx\cb@temp\@empty\else
518            \expandafter\cb@pdfparsexy\cb@temp
519            \ifnum\cb@pdfpg<0\else
520              \xdef\cb@pdfpoints{\cb@pdfpoints\cb@temp}%
521              \cb@trace{PDFpoints=\cb@pdfpoints}%
522              \global\let\cb@pdfpagenr\cb@pdfpg
523            \fi
524            \ifnum\cb@pdfpg>##1\else
525              \global\def\cb@next{\cb@pdfreadxy{##1}}%
526            \fi
527          \fi
528        }%
529      \fi
530      \cb@next
531    }%
```

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

```
532    \def\cb@pdfparsexy##1.##2p##3;##4;##5pt{%
533      \def\cb@pdfpg{##2}}%
```

As PDF is not a programming language it does not have any variables to re-
member 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.

```
534    \def\cb@defpoint##1##2{%
535      \if@filesw
536        \begingroup
537          \edef\point{{\the##1}{\the\cb@pagecount}}%
538          \let\the=\z@
539          \pdfsavepos
540          \edef\cb@temp{\write\@auxout
541            {\string\cb@pdfxy\point
542              {\the\pdflastxpos}{\the\pdflastypos}{\the##2}}}%
543          \cb@temp
544        \endgroup
545      \fi
546      \cb@trace@defpoint##1##2%
547    }%
```

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

```
548    \def\cb@cvtpct##1{%
549      \ifnum##1<0 0\else
550      \ifnum##1>99 1\else
551      \ifnum##1<10 0.0\the##1\else
552      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.

```
553 \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.

```
554    \def\cb@connect##1##2##3{%
555      \cb@findpdfpoint{##1}\cb@pagecount
556      \ifx\cb@pdfx\relax\cb@rerun
557      \else
558        \let\cb@pdftopy\cb@pdfy
559        \cb@findpdfpoint{##2}\cb@pagecount
560        \ifx\cb@pdfx\relax\cb@rerun
561        \else
```

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

```
562          \begingroup
563            \cb@dima=\cb@pdfz
564            \advance\cb@dima by-\cb@pdfx
565            \advance\cb@dima by1in%
566            \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.

23

```
567            \ifx\cb@current@color\@undefined
568              \def\cb@temp{\cb@cvtpct\c@changebargrey}%
569              \pdfliteral{q \cb@temp\space g \cb@temp\space G}%
570            \else
571              \pdfliteral{q \expandafter\sec@nd@ftw@\cb@current@color\space RG
572                          \expandafter\sec@nd@ftw@\cb@current@color\space rg}%
573            \fi
574            \edef\cb@temp{\expandafter\cb@removedim\the\cb@dima\space}%
575            \cb@dima=\cb@pdftopy
576            \advance\cb@dima-\cb@pdfy\relax
577            \cb@dima=\cb@pdf@scale\cb@dima\relax
578            ##3=\cb@pdf@scale##3\relax
579            \pdfliteral{\expandafter\cb@removedim\the##3 w
580              \cb@temp 0 m
581              \cb@temp \expandafter\cb@removedim\the\cb@dima\space l S Q}%
582          \endgroup
```

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

```
583          \cb@cntb=##1\relax
584          \ifodd\cb@cntb\advance\cb@cntb 1\else\advance\cb@cntb -1\fi
585          \cb@findpdfpoint\cb@cntb\cb@pagecount
586          \cb@cntb=##2\relax
587          \ifodd\cb@cntb\advance\cb@cntb 1\else\advance\cb@cntb -1\fi
588          \cb@findpdfpoint\cb@cntb\cb@pagecount
589        \fi
590      \fi
591      \cb@trace@connect##1##2##3%
592    }%
```

`\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.

```
593    \gdef\cb@checkPdfxy##1##2##3##4##5{%
594    \cb@@findpdfpoint{##1}{##2}%
595    \ifdim##3sp=\cb@pdfx\relax
596      \ifdim##4sp=\cb@pdfy\relax
597        \ifdim##5sp=\cb@pdfz\relax
598        \else
599        \cb@error
600        \fi
601      \else
602        \cb@error
603      \fi
604    \else
605      \cb@error
606    \fi
607    }
```

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

```
608    \def\cb@maxpoint{9999999}
609    \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.

```
610 \or
611     \immediate\closeout\cb@writexy
612     \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.

```
613     \def\cb@pdfpoints{}
614     \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.

```
615     \def\cb@findpdfpoint##1##2{%
616         \edef\cb@temp
617             {\noexpand\cb@@findpdfpoint{\the##1}{\the##2}}%
618         \cb@temp
619     }
```

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

```
620     \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.

```
621     \def\cb@@findpdfpoint##1##2{%
622         \ifnum##2<\cb@pdfpagenr\relax\else
623             \cb@pdfreadxy{##2}%
624         \fi
625         \let\cb@pdfx\relax
626         \ifx\cb@pdfpoints\@empty\else
627             \ifnum##2<0\relax
628             \else
629                 \edef\cb@temp{%
630                     \noexpand\cb@pdffind{##1}{##2}\cb@pdfpoints\relax{}}%
631                 \cb@temp
632             \fi
633         \fi
634     }
```

\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.

```
635     \def\cb@pdffind##1##2##3.##4p##5;##6;##7pt##8\relax##9{%
636         \def\cb@next{%
637             \cb@pdffind{##1}{##2}##8\relax{##9##3.##4p##5;##6;##7pt}}%
638         \ifnum ##1=##3
639             \ifnum ##2=##4
640                 \def\cb@pdfx{##5sp}%
```

```
641          \def\cb@pdfy{##6sp}%
642          \def\cb@pdfz{##7pt}%
643          \let\cb@next\relax
644          \gdef\cb@pdfpoints{##9##8}%
645        \fi
646      \fi
647      \ifx\relax##8\relax
648        \let\cb@next\relax
649      \fi
650      \cb@next
651    }%
```

\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.

```
652    \def\cb@pdfreadxy##1{%
653      \let\cb@next\relax
654      \ifeof\cb@readxy
655        \global\let\cb@pdfpagenr\cb@maxpoint
656      \else
657        {\endlinechar=-1\read\cb@readxy to\cb@temp
658          \ifx\cb@temp\@empty\else
659            \expandafter\cb@pdfparsexy\cb@temp
660            \ifnum\cb@pdfpg<0\else
661              \xdef\cb@pdfpoints{\cb@pdfpoints\cb@temp}%
662              \cb@trace{PDFpoints=\cb@pdfpoints}%
663              \global\let\cb@pdfpagenr\cb@pdfpg
664            \fi
665            \ifnum\cb@pdfpg>##1\else
666              \global\def\cb@next{\cb@pdfreadxy{##1}}%
667            \fi
668          \fi
669        }%
670      \fi
671      \cb@next
672    }%
```

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

```
673    \def\cb@pdfparsexy##1.##2p##3;##4;##5pt{%
674      \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.

```
675    \def\cb@defpoint##1##2{%
676      \if@filesw
677        \begingroup
678          \edef\point{{\the##1}{\the\cb@pagecount}}%
679          \let\the=\z@
680          \savepos
```

26

```
681        \edef\cb@temp{\write\@auxout
682          {\string\cb@pdfxy\point
683            {\the\lastxpos}{\the\lastypos}{\the##2}}}%
684        \cb@temp
685      \endgroup
686    \fi
687    \cb@trace@defpoint##1##2%
688  }%
```

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

```
689    \def\cb@cvtpct##1{%
690    \ifnum##1<0 0\else
691    \ifnum##1>99 1\else
692    \ifnum##1<10 0.0\the##1\else
693    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.

```
694 \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.

```
695    \def\cb@connect##1##2##3{%
696    \cb@findpdfpoint{##1}\cb@pagecount
697    \ifx\cb@pdfx\relax\cb@rerun
698    \else
699      \let\cb@pdftopy\cb@pdfy
700      \cb@findpdfpoint{##2}\cb@pagecount
701      \ifx\cb@pdfx\relax\cb@rerun
702      \else
```

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

```
703        \begingroup
704          \cb@dima=\cb@pdfz
705          \advance\cb@dima by-\cb@pdfx
706          \advance\cb@dima by1in%
707          \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.

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

27

```
717        \cb@dima=\cb@pdf@scale\cb@dima\relax
718        ##3=\cb@pdf@scale##3\relax
719        \pdfliteral{\expandafter\cb@removedim\the##3 w
720          \cb@temp 0 m
721          \cb@temp \expandafter\cb@removedim\the\cb@dima\space l S Q}%
722      \endgroup
```

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

```
723        \cb@cntb=##1\relax
724        \ifodd\cb@cntb\advance\cb@cntb 1\else\advance\cb@cntb -1\fi
725        \cb@findpdfpoint\cb@cntb\cb@pagecount
726        \cb@cntb=##2\relax
727        \ifodd\cb@cntb\advance\cb@cntb 1\else\advance\cb@cntb -1\fi
728        \cb@findpdfpoint\cb@cntb\cb@pagecount
729      \fi
730    \fi
731    \cb@trace@connect##1##2##3%
732  }%
```

`\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.

```
733  \gdef\cb@checkPdfxy##1##2##3##4##5{%
734    \cb@@findpdfpoint{##1}{##2}%
735    \ifdim##3sp=\cb@pdfx\relax
736      \ifdim##4sp=\cb@pdfy\relax
737        \ifdim##5sp=\cb@pdfz\relax
738        \else
739        \cb@error
740        \fi
741      \else
742        \cb@error
743      \fi
744    \else
745      \cb@error
746    \fi
747  }
```

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

```
748    \def\cb@maxpoint{9999999}
749    \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.

```
750 \else
751   \PackageWarning{Changebar}{changebars not supported in unknown setup}
752   \def\cb@defpoint##1##2{\cb@trace@defpoint##1##2}
753   \def\cb@connect##1##2##3{\cb@trace@connect##1##2##3}
754   \let\cb@resetpoints\relax
755 \fi
```

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

```
756 \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.

```
757 \newcommand*{\cbstart}{\@ifnextchar[%]
758                          {\cb@start}%
759                          {\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.

```
760 \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.

```
761 \newcommand*{\cbdelete}{\@ifnextchar[%]
762   {\cb@delete}%
763   {\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.

```
764 \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.

```
765 \newenvironment{changebar}%
766                {\@ifnextchar[{\cb@start}%]
767                            {\cb@start[\changebarwidth]}}%
768                {\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.

```
769 \newcommand*{\nochangebars}{%
770   \def\cb@start[##1]{\ignorespaces}%
771   \def\cb@delete[##1]{}%
772   \def\cb@end{\ignorespacesafterend}
773   }
```

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

```
774 \newlength{\changebarwidth}
775 \setlength{\changebarwidth}{2pt}
```

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

```
776 \newlength{\deletebarwidth}
777 \setlength{\deletebarwidth}{4pt}
```

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

```
778 \newlength{\changebarsep}
779 \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).

```
780 \newcounter{changebargrey}
781 \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.

```
782 \@ifpackagewith{changebar}{\csname cb@color@pkg\endcsname}{%
783     \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.

```
784     \DeclareRobustCommand\cbcolor{%
785       \@ifnextchar[%
786         \@undeclaredcbcolor\@declaredcbcolor}
```

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

```
787     \def\@undeclaredcbcolor[#1]#2{%
788       \begingroup
789         \color[#1]{#2}%
790         \global\let\cb@current@color\current@color
791       \endgroup
792       \ignorespaces
793       }
```

\@declaredcbcolor

```
794     \def\@declaredcbcolor#1{%
795       \begingroup
796         \color{#1}%
797         \global\let\cb@current@color\current@color
798       \endgroup
799       \ignorespaces}%
800     }{%
```

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

```
801     \def\cbcolor{\@ifnextchar[%
802       \@@cbcolor\@cbcolor}%
803     \def\@@cbcolor[#1]#2{\cb@colwarn\def\@@cbcolor[##1]##2{}}%
804     \def\@cbcolor#1{\cb@colwarn\def\@cbcolor##1{}}%
805     \def\cb@colwarn{\PackageWarning{Changebar}%
806       {You didn't specify the option 'color';\MessageBreak
807         your command \string\cbcolor\space will be ignored}}%
808     }
```

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.

```
809 \def\cb@start[#1]{%
810    \cb@topleft=\cb@nextpoint
```

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

```
811    \cb@curbarwd#1\relax
812    \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.

```
813    \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.

```
814    \cb@cnta\cb@pagecount
815    \cb@pagecount\cb@page\advance\cb@pagecount\m@ne
816    \ifvmode
817      \cb@setBeginPoints
818    \else
819      \vbox to \z@{%
```

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

```
820        \vskip -\ht\strutbox
821        \cb@setBeginPoints
822        \vskip \ht\strutbox}%
823    \fi
```

Restore \cb@pagecount.

```
824    \cb@pagecount\cb@cnta
825    \cb@advancePoint\ignorespaces}
```

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

```
826 \def\cb@advancePoint{%
827    \global\advance\cb@nextpoint by 4\relax
828    \ifnum\cb@nextpoint>\cb@maxpoint
829      \global\cb@nextpoint=\cb@minpoint\relax
830    \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.

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

```
832    \cb@trace@stack{end of bar on page \the\c@page}%
833    \cb@pop\cb@currentstack
834    \ifnum\cb@topleft=\cb@nil
835      \PackageWarning{Changebar}%
836        {Badly nested changebars; Expect erroneous results}%
837    \else
```

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

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

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

```
839      \cb@cnta\cb@pagecount
840      \cb@pagecount\cb@page\advance\cb@pagecount\m@ne
841      \ifvmode
842        \cb@setEndPoints
843      \else
844        \vadjust{\cb@setEndPoints}%
845      \fi
846      \cb@pagecount\cb@cnta
847    \fi
848    \ignorespacesafterend}
```

\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).

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

First store the identifiers in temporary registers.

```
850    \cb@cnta\cb@topleft\relax
851    \advance\cb@cnta by #1\relax
852    \cb@cntb\cb@page\relax
853    \cb@dima\cb@curbarwd\relax
```

Then pop the history stack.

```
854    \cb@pop\cb@historystack
```

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

```
855    \ifnum\cb@topleft=\cb@nil
856    \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.

```
857      \cb@FindPageNum
858      \ifnum\cb@topleft>\cb@maxpoint\else
```

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

```
859        \cb@cntb\cb@page
860      \fi
```

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

```
861      \@whilenum\cb@topleft>\cb@nil\do{%
862        \cb@push\cb@historystack
863        \cb@pop\cb@tempstack}%
864    \fi
```

Finally return the correct values

```
865    \advance\cb@cnta by -#1\relax
866    \cb@topleft\cb@cnta\relax
867    \cb@page\cb@cntb\relax
868    \cb@curbarwd\cb@dima\relax
869    }
```

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

```
870 \def\cb@FindPageNum{%
871    \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.

```
872    \else
873      \cb@push\cb@tempstack
874      \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.

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

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

```
877        \cb@topleft\cb@maxpoint\advance\cb@topleft\@ne
878      \else
```

Recursively call ourselves.

```
879        \expandafter\expandafter\expandafter\cb@FindPageNum
880      \fi
881    \fi
882    }%
```

\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.

```
883 \def\cb@setBeginPoints{%
884    \cb@topright=\cb@topleft\advance\cb@topright by\@ne
885    \cb@cntb=\cb@pagecount
886    \divide\cb@cntb by\tw@
887    \ifodd\cb@cntb
888      \cb@defpoint\cb@topleft\cb@even@left
889      \cb@defpoint\cb@topright\cb@even@right
890    \else
891      \cb@defpoint\cb@topleft\cb@odd@left
892      \cb@defpoint\cb@topright\cb@odd@right
893    \fi
894    \cb@writeAux\cb@topleft
895    }
```

\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.

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

In twocolumn pages always use outerbars

```
914   \if@twocolumn
915       \ifodd\cb@pagecount\cb@rightbar\else\cb@leftbar\fi
916   \else
917     \ifcase\cb@barsplace
```

0=innerbars

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

1=outerbars

```
924       \ifodd\cb@cntb
925         \cb@leftbar
926       \else
927         \if@twoside\cb@rightbar\else\cb@leftbar\fi
928       \fi
929     \or
```

2=leftbars

```
930       \cb@leftbar
931     \or
```

3=rightbars

```
932       \cb@rightbar
933     \fi
934   \fi
935   }%
```

\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.

```
936 \def\cb@writeAux#1{%
937   \if@filesw
938     \begingroup
939       \edef\point{\the#1}%
940       \edef\level{\the\cb@curbarwd}%
941       \let\the=\z@
942       \edef\cb@temp{\write\@auxout
943         {\string\cb@barpoint{\point}{\the\cb@pagecount}{\level}}}%
944       \cb@temp
945     \endgroup
946   \fi}
```

5.5   Macros for Making It Work Across Page Breaks

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

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

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

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

\cb@nextpagejump The next pagecount from the list.

```
949 \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.

```
950 \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.

```
951 \def\cb@writepagejump#1{
952   \cb@cntb=\cb@pagecount
953   \advance\cb@cntb by#1\relax
954   \global\setbox\@leftcolumn\vbox to\@colht{%
955     \edef\cb@temp{\write\@auxout{\string\cb@pagejump{\the\cb@cntb}}}%
956     \cb@temp
957     \dimen@ \dp\@leftcolumn
958     \unvbox \@leftcolumn
959     \vskip -\dimen@
960   }%
961 }
```

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

```
962 \def\cb@poppagejump#1,#2\relax{%
963   \gdef\cb@nextpagejump{#1}%
964   \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.

```
965 \def\cb@checkpagecount{%
966   \if@twocolumn
967     \if@firstcolumn
968       \ifodd\cb@pagecount\global\advance\cb@pagecount by\@ne\fi
969     \fi
970   \else
971     \ifodd\cb@pagecount\global\advance\cb@pagecount by\@ne\fi
972   \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.

```
973   \if@twoside
974     \cb@cntb=\cb@pagecount
975     \divide\cb@cntb by\tw@
976     \advance\cb@cntb by-\c@page
977     \ifodd\cb@cntb
```

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

```
978       \if@twocolumn
979         \if@firstcolumn
980           \@whilenum\cb@pagecount>\cb@nextpagejump\do{%
981             \expandafter\cb@poppagejump\cb@pagejumplst\relax}%
982           \ifnum\cb@pagecount=\cb@nextpagejump
983             \cb@trace{Page jump: \string\cb@pagecount=\the\cb@pagecount}
984             \global\advance\cb@pagecount by\tw@
985             \global\@cb@pagejumptrue
986           \else
987             \global\@cb@pagejumpfalse
988           \fi
989         \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.

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

36

```
993        \fi
994      \fi
995    \else
```

Here `\cb@pagecount` is incorrect.

```
996      \if@twocolumn
997        \if@firstcolumn
998          \global\advance\cb@pagecount by\tw@
999          \global\@cb@pagejumpfalse
1000        \else
1001          \if@cb@pagejump
1002            \cb@trace{Page jump annulled, %
1003                       \string\cb@pagecount=\the\cb@pagecount}
1004          \else
1005            \cb@writepagejump{-1}%
1006            \global\advance\cb@pagecount by\tw@
1007            \cb@rerun
1008          \fi
1009        \fi
1010      \else
1011        \global\advance\cb@pagecount by\tw@
1012      \fi
1013    \fi
1014  \fi
1015 }
```

`\@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.

```
1016 \let\ltx@makecol\@makecol
1017 \def\cb@makecol{%
1018   \if@twocolumn
1019     \cb@trace{Twocolumn: \if@firstcolumn Left \else Right \fi column}%
1020   \fi
1021   \cb@trace@stack{before makecol, page \the\c@page,
1022                   \string\cb@pagecount=\the\cb@pagecount}%
1023   \let\cb@writeAux\@gobble
```

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

```
1024   \cb@checkpagecount
1025   \setbox\@cclv \vbox{%
1026     \cb@resetpoints
1027     \cb@startSpanBars
1028     \unvbox\@cclv
1029     \boxmaxdepth\maxdepth}%
1030   \global\advance\cb@pagecount by\@ne
1031   \cb@buildstack\cb@processActive
```

```
1032    \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.

```
1033    \if@twocolumn
1034      \global\setbox\@outputbox \vbox to\@colht {%
1035        \if@firstcolumn\write\@auxout{\string\@cb@firstcolumntrue}%
1036        \else\write\@auxout{\string\@cb@firstcolumnfalse}%
1037        \fi
1038        \dimen@ \dp\@outputbox
1039        \unvbox \@outputbox
1040        \vskip -\dimen@
1041        }%
1042    \fi
1043    \cb@trace@stack{after makecol, page \the\c@page,
1044                    \string\cb@pagecount=\the\cb@pagecount}%
1045    }
1046 \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.

```
1047 \let\ltx@vtryfc\@vtryfc
1048 \def\cb@vtryfc#1{%
1049    \cb@trace{In vtryfc, page \the\c@page,
1050                    \string\cb@pagecount=\the\cb@pagecount}%
1051    \let\cb@writeAux\@gobble
```

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

```
1052    \cb@checkpagecount
1053    \ltx@vtryfc{#1}%
1054    \if@twocolumn
1055      \global\setbox\@outputbox \vbox to\@colht{%
1056        \if@firstcolumn\write\@auxout{\string\@cb@firstcolumntrue}%
1057        \else\write\@auxout{\string\@cb@firstcolumnfalse}%
1058        \fi
1059        \unvbox\@outputbox
1060        \boxmaxdepth\maxdepth
1061      }%
1062    \fi
1063    \global\advance\cb@pagecount by \@ne
1064 }
1065 \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).
```

```
1066 \def\cb@processActive{%
1067   \cb@pop\cb@endstack
1068   \ifnum\cb@topleft=\cb@nil
1069   \else
1070     \setbox\@cclv\vbox{%
1071       \unvbox\@cclv
1072       \boxmaxdepth\maxdepth
1073       \advance\cb@pagecount by -1\relax
1074       \cb@setEndPoints}%
1075     \cb@push\cb@historystack

1076     \cb@push\cb@beginstack
1077     \expandafter\cb@processActive
1078   \fi}
```

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

```
1079 \def\cb@startSpanBars{%
1080   \cb@pop\cb@beginstack
1081   \ifnum\cb@topleft=\cb@nil
1082   \else
1083     \cb@setBeginPoints
1084     \cb@trace@stack{after StartSpanBars, page \the\c@page}%
1085     \expandafter\cb@startSpanBars
1086   \fi
1087   }
```

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

```
1088 \def\cb@buildstack{%
1089   \cb@initstack\cb@endstack
1090   \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.

```
1091 \def\cb@pushNextActive{%
1092   \cb@pop\cb@historystack
1093   \ifnum\cb@topleft=\cb@nil
1094   \else
1095     \ifnum\cb@page>\cb@pagecount
1096       \cb@push\cb@historystack
1097     \else
1098       \ifodd\cb@topleft
1099         \cb@push\cb@endstack
1100       \else
```

39

```
1101          \cb@pop\cb@endstack
1102        \fi
1103        \expandafter\expandafter\expandafter\cb@pushNextActive
1104      \fi
1105    \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.

```
1106 \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
```
1107 \cb@initstack\cb@historystack
```

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

```
1108 \newwrite\cb@write
1109 \newread\cb@read
```

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

```
1110 \immediate\openout\cb@write=\jobname.cb\relax
```

\cb@endstack   Allocate two stacks for the bars that span the current page break.
\cb@beginstack
```
1111 \cb@initstack\cb@endstack
1112 \cb@initstack\cb@beginstack
```

\cb@tempstack   Allocate a stack for temporary storage

```
1113 \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.

```
1114 \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.

```
1115 \def\cb@thehistorystack{\cb@historystack}
1116 \def\cb@pop#1{%
1117   \ifx #1\@empty
1118     \def\cb@temp{#1}%
1119     \ifx\cb@temp\cb@thehistorystack
1120       \ifeof\cb@read
1121       \else
1122         {\endlinechar=-1\read\cb@read to\cb@temp
1123          \xdef\cb@historystack{\cb@historystack\cb@temp}%
1124         }%
1125       \fi
1126     \fi
1127   \fi
1128   \ifx#1\@empty
1129     \global\cb@topleft\cb@nil
1130     \global\cb@page\z@\relax
1131   \else
1132     \expandafter\cb@carcdr#1e#1%
1133   \fi
1134   \cb@trace@pop{#1}}
```

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

```
1135 \def\cb@carcdr#1n#2p#3l#4e#5{%
1136   \global\cb@topleft#1\relax
1137   \global\cb@page#2\relax
1138   \global\cb@curbarwd#3\relax
1139   \xdef#5{#4}}
```

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

```
1140 \def\cb@push#1{%
1141   \xdef#1{\the\cb@topleft n\the\cb@page p\the\cb@curbarwd l#1}%
1142   \cb@trace@push{#1}}
1143
```

\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.

```
1144 \def\cb@barpoint#1#2#3{\cb@cnta=#2
1145   \if@cb@firstcolumn\advance\cb@cnta by\m@ne\fi
1146   \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.

```
1147 \AtBeginDocument{%
1148    \cb@setup@specials
```

Add a sentinel to `\cb@pagejumplst`.

```
1149    \cb@pagejump{999999999,}%
```

Compute the left and right positions of the changebars.

```
1150    \cb@positions
1151    \cb@trace{%
1152      Odd left  : \the\cb@odd@left\space
1153      Odd right : \the\cb@odd@right\MessageBreak
1154      Even left: \the\cb@even@left\space
1155      Even right: \the\cb@even@right
1156      }%
1157    \immediate\closeout\cb@write
1158    \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.

```
1159 \AtEndDocument{%
1160    \clearpage
1161    \cb@initstack\cb@historystack
1162    \immediate\closein\cb@read
1163    \immediate\openin\cb@read=\jobname.cb\relax
```

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

```
1164    \ifx\cb@readxy\@undefined
1165    \else
1166      \immediate\closein\cb@readxy
1167      \immediate\openin\cb@readxy=\jobname.cb2\relax
1168      \def\cb@pdfpoints{}%
1169      \def\cb@pdfpagenr{0}%
1170    \fi
1171    \@cb@firstcolumnfalse
1172    \cb@checkrerun
1173    \let\cb@pdfxy\cb@checkPdfxy
1174    \let\cb@pagejump\@gobble
1175    \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.

```
1176 \def\cb@checkHistory#1#2#3{%
1177    \cb@pop\cb@historystack
1178    \ifnum #1=\cb@topleft\relax
1179      \cb@cnta=#2
1180      \if@cb@firstcolumn\advance\cb@cnta by\m@ne\fi
1181      \ifnum \cb@cnta=\cb@page\relax
```

42

Both page and point numbers are equal; do nothing,

```
1182    \else
```

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

```
1183      \cb@error
1184    \fi
1185  \else
```

when point numbers don't match.

```
1186    \cb@error
1187  \fi}
```

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

```
1188 \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.

```
1189 \def\cb@rerun{%
1190   \global\let\cb@checkrerun\cb@error}
1191 \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`.

```
1192 \def\cb@error{%
1193   \PackageWarning{Changebar}%
1194     {Changebar info has changed.\MessageBreak
1195       Rerun to get the bars right}
1196   \gdef\cb@checkHistory##1##2##3{}%
1197   \let\cb@barpoint\cb@checkHistory
1198   \gdef\cb@checkPdfxy##1##2##3##4##5{}%
1199   \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`.

```
1200 \let\ltx@end@float\end@float
```

```
1201 \def\cb@end@float{%
1202   \cb@trace@stack{end float on page \the\c@page}%
1203   \cb@pop\cb@currentstack
1204   \ifnum\cb@topleft=\cb@nil
1205   \else
1206     \cb@push\cb@currentstack
1207     \global\cb@curbarwd=\cb@curbarwd
1208     \@endfloatbox
1209     \global\setbox\@currbox
1210       \color@vbox
1211       \normalcolor
1212       \vbox\bgroup\cb@start[\cb@curbarwd]\unvbox\@currbox\cb@end
1213   \fi
1214   \ltx@end@float}
1215 \let\end@float\cb@end@float
```

This only works if this new version of \end@float is really used. With LaTeX2.09 the documentstyles used to contain:

    \let\endfigure\end@float

In that case this binding has to be repeated after the redefinition of \end@float. However, the LaTeX 2ε 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.

```
1216 \let\ltx@@xympar\@xympar
1217 \def\@xympar{%
1218   \let\end@float\ltx@end@float
1219   \ltx@@xympar
1220   \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.

```
1221 \let\flt@float@end\float@end
```

Then we redefine it to insert the changebarcode.

```
1222 \def\float@end{%
1223   \cb@trace@stack{end float on page \the\c@page}%
1224   \cb@pop\cb@currentstack
1225   \ifnum\cb@topleft=\cb@nil
1226   \else
1227     \cb@push\cb@currentstack
1228     \global\cb@curbarwd\cb@curbarwd
1229     \@endfloatbox
1230     \global\setbox\@currbox
1231       \color@vbox
1232       \normalcolor
1233       \vbox\bgroup\cb@start[\cb@curbarwd]\unvbox\@currbox\cb@end
1234   \fi
1235   \let\end@float\ltx@end@float
1236   \flt@float@end
1237   }
```

\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.

```
1238 \let\ltx@end@dblfloat\end@dblfloat
1239 \def\cb@end@dblfloat{%
1240   \if@twocolumn
1241     \cb@trace@stack{end dblfloat on page \the\c@page}%
1242     \cb@pop\cb@currentstack
1243     \ifnum\cb@topleft=\cb@nil
```

```
1244    \else
1245      \cb@push\cb@currentstack
1246      \global\cb@curbarwd=\cb@curbarwd
1247      \@endfloatbox
1248      \global\setbox\@currbox
1249        \color@vbox
1250        \normalcolor
1251        \vbox\bgroup\cb@start[\cb@curbarwd]\unvbox\@currbox\cb@end
1252    \fi
1253    \@endfloatbox
1254    \ifnum\@floatpenalty <\z@
1255      \@largefloatcheck
1256      \@cons\@dbldeferlist\@currbox
1257    \fi
1258    \ifnum \@floatpenalty =-\@Mii \@Esphack\fi
1259  \else
1260    \end@float
1261  \fi}
1262 \let\end@dblfloat\cb@end@dblfloat
```

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

```
1263 \let\flt@float@dblend\float@dblend
1264 \def\float@dblend{%
1265   \cb@trace@stack{end dbl float on page \the\c@page}%
1266   \cb@pop\cb@currentstack
1267   \ifnum\cb@topleft=\cb@nil
1268   \else
1269     \cb@push\cb@currentstack
1270     \global\cb@curbarwd=\cb@curbarwd
1271     \@endfloatbox
1272     \global\setbox\@currbox
1273       \color@vbox
1274       \normalcolor
1275       \vbox\bgroup\cb@start[\cb@curbarwd]\unvbox\@currbox\cb@end
1276   \fi
1277   \let\end@dblfloat\ltx@end@dblfloat
1278   \flt@float@dblend
1279   }
```

\@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.

```
1280 \let\ltx@footnotetext\@footnotetext

1281 \long\def\cb@footnotetext#1{%
1282   \cb@trace@stack{end footnote on page \the\c@page}%
1283   \cb@pop\cb@currentstack
1284   \ifnum\cb@topleft=\cb@nil
1285     \ltx@footnotetext{#1}%
1286   \else
1287     \cb@push\cb@currentstack
1288     \edef\cb@temp{\the\cb@curbarwd}%
1289     \ltx@footnotetext{\cb@start[\cb@temp]#1\cb@end}%
```

```
1290    \fi}
1291 \let\@footnotetext\cb@footnotetext
```

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

```
1292 \let\ltx@mpfootnotetext\@mpfootnotetext

1293 \long\def\cb@mpfootnotetext#1{%
1294    \cb@pop\cb@currentstack
1295    \ifnum\cb@topleft=\cb@nil
1296      \ltx@mpfootnotetext{#1}%
1297    \else
1298      \cb@push\cb@currentstack
1299      \edef\cb@temp{\the\cb@curbarwd}%
1300      \ltx@mpfootnotetext{\cb@start[\cb@temp]#1\cb@end}%
1301    \fi}
1302 \let\@mpfootnotetext\cb@mpfootnotetext
1303 ⟨/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