

The Changebar package *

Michael Fine
Distributed Systems Architecture

Johannes Braams
texniek at texniek.nl

Printed July 19, 2026

Contents

1 Introduction	1	5 The implementation	6
2 The user interface	2	5.1 Declarations And Initializations	6
2.1 The package options . . .	2	5.2 Option Processing	8
2.1.1 Specifying the printer driver . . .	2	5.3 User Level Commands And Parameters	13
2.1.2 Specifying the bar position	3	5.4 Macros for beginning and ending bars	33
2.1.3 Color	3	5.5 Macros for Making It Work Across Page Breaks	38
2.1.4 Tracing	3	5.6 Macros For Managing The Stacks of Bar points .	43
2.2 Macros defined by the package	3	5.7 Macros For Checking That The .aux File Is Stable	45
2.3 Changebar parameters . .	3	5.8 Macros For Making It Work With Nested Floats/Footnotes	47
3 Deficiencies and bugs	4		
4 The basic algorithm	5		

Abstract

This package implements a way to indicate modifications in a L^AT_EX-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 pdf_{tex}, Xe_T_EX and lua_T_EX 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

- Changebars may be nested within each other. Each level of nesting can be given a different thickness bar.

*This file has version number v3.7g, last revised 2026/07/19.

- 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` and `xetex` 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 options¹ 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`

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.

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

2.1.2 Specifying the bar position

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

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

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

2.1.3 Color

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

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

2.1.4 Tracing

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

2.2 Macros defined by the package

- `\cbstart` All material between the macros `\cbstart` and `\cbend` is barred. The nesting of multiple changebars is allowed. The macro `\cbstart` has an optional parameter that specifies the width of the bar. The syntax is `\cbstart[⟨dimension⟩]`. If no width is specified, the current value of the parameter `\changebarwidth` is used. Note that `\cbstart` and `\cbend` can be used anywhere but must be correctly nested with floats and footnotes. That is, one cannot have one end of the bar inside a floating insertion and the other outside, but that would be a meaningless thing to do anyhow.
- `changebar` (*env.*) Apart from the macros `\cbstart` and `\cbend` a proper $\text{\LaTeX}$ environment is defined. The advantage of using the environment whenever possible is that $\text{\LaTeX}$ 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 $\text{\LaTeX}$ 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 L^AT_EX 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 L^AT_EX 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 L^AT_EX 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 PDF_TE_X or Xe_TE_X.
- 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 L^AT_EX floats are not split but footnotes may be. The simplest fix to this is to prevent footnotes from being split but this may make T_EX 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@cna
12 \newcount\cb@cnb
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 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 T_EX-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
- VTeX
- PDFTeX
- XeTeX
- 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{emt看}{\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 $\text{\LaTeX}$ run is using `PDFTeX` and that PDF output is selected. If it is, we initialize the option and open an additional output file. If not, we ignore the option and issue a warning.

```

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 $\text{\LaTeX}$ 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 $\text{\LaTeX}$ 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 $\text{\LaTeX}_{2\epsilon}$ 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 dviton03, 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 $\text{\TeX}$ is used the option `dvips` is not the right one, so in that case we have `vtex` as the default driver. When $\text{\PDF\TeX}$ 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 pdfL^AT_EX, while also creating pdf as output, the driver to be used is the pdf driver. Therefore we add an extra check, possibly overriding a `dvips` option that might still have been in the document.

```
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   \ifcb@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
- emT_EX
- T_EXtures
- V_T_EX
- PDFT_EX
- XeT_EX

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>}`. 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 DVItO_{PS} 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 DVItO_{PS}, but the coordinate space has a different dimension. Also this code has been made resolution independent, whereas the code for DVItO_{PS} 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 T_EXtures version 1.7.7, but will very likely also work with later releases of T_EXtures.

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   \ifcb@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   \ifcb@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
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 PDF_T_EX is 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.

```
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 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. 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`.

Scanning `\cb@pdfpoints` gets expensive when the list grows: the recursive `\cb@pdffind` re-absorbs the whole remainder of the list as a macro argument at every element it steps over, and a search that fails removes nothing. On a run during which the page breaks move (the situation described at `\cb@checkpage`, where nearly every lookup asks for a pagecount that disagrees with the recorded one and therefore fails) the list only ever grows and every later search scans all of it; with a few thousand bars such a run can take hours. `\cb@pdfxy` already stores every record in a control sequence `\cb@⟨point⟩.⟨pagecount⟩` while the auxiliary file is replayed, and as long as point numbers are unique that store is an exact index of the list, so we probe it directly. Two details preserve the original behaviour: a record that has been delivered once is removed from the list by `\cb@pdffind`, so a marker `\cb@xyused@⟨point⟩.⟨pagecount⟩` makes any second lookup fail just as it would have; and the check performed at the end of the document must see every record again, so once `\if@cb@enddocument` is set the markers are ignored and no new ones are recorded. When point numbers are not unique the original code path in `\cb@listfindpdfpoint` is taken instead: later records overwrite earlier ones in the control-sequence store, which makes it unreliable exactly then.

```
335 \def\cb@@findpdfpoint##1##2{%
336     \if@cb@pointsunique
337     \let\cb@pdfx\relax
338     \ifnum##2<0\relax\else
339         \ifcsname cb@##1.##2\endcsname
340             \if@cb@enddocument
341                 \edef\cb@temp{\csname cb@##1.##2\endcsname}%
342                 \expandafter\cb@parsexyrecord\cb@temp\cb@xynil
343             \else
344                 \ifcsname cb@xyused@##1.##2\endcsname
```

```

345         \else
346         \edef\cb@temp{\csname cb@##1.##2\endcsname}%
347         \expandafter\cb@parsexyrecord\cb@temp\cb@xynil
348         \global\expandafter\let
349         \csname cb@xyused@##1.##2\endcsname\@empty
350     \fi
351 \fi
352 \fi
353 \fi
354 \else
355     \cb@listfindpdfpoint{##1}{##2}%
356 \fi
357 }

```

`\cb@listfindpdfpoint` If the information is not yet in memory it is read from the .cb2 file.

```

358 \def\cb@listfindpdfpoint##1##2{%
359     \ifnum##2<\cb@pdfpagenr\relax\else
360     \cb@pdfreadxy{##2}%
361     \fi
362     \let\cb@pdfx\relax
363     \ifx\cb@pdfpoints\@empty\else
364     \ifnum##2<0\relax
365     \else
366     \edef\cb@temp{\noexpand\cb@pdffind{##1}{##2}\cb@pdfpoints\relax{}}%
367     \cb@temp
368     \fi
369 \fi
370 }

```

`\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.

```

371 \def\cb@pdffind##1##2##3.##4p##5/##6/##7pt##8\relax##9{%
372     \def\cb@next{%
373         \cb@pdffind{##1}{##2}##8\relax{##9##3.##4p##5/##6/##7pt}}%
374     \ifnum ##1=##3
375     \ifnum ##2=##4
376     \def\cb@pdfx{##5sp}%
377     \def\cb@pdfy{##6sp}%
378     \def\cb@pdfz{##7pt}%
379     \let\cb@next\relax
380     \gdef\cb@pdfpoints{##9##8}%
381     \fi
382 \fi
383 \ifx\relax##8\relax
384     \let\cb@next\relax
385 \fi
386 \cb@next
387 }%

```

`\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.

```

388 \def\cb@pdfreadxy##1{%
389   \let\cb@next\relax
390   \ifeof\cb@readxy
391     \global\let\cb@pdfpagenr\cb@maxpoint
392   \else
393     {\endlinechar=-1\read\cb@readxy to\cb@temp
394     \ifx\cb@temp\@empty\else
395       \expandafter\cb@pdfparsexy\cb@temp
396       \ifnum\cb@pdfpg<0\else
397         \xdef\cb@pdfpoints{\cb@pdfpoints\cb@temp}%
398         \cb@trace{PDFpoints=\cb@pdfpoints}%
399         \global\let\cb@pdfpagenr\cb@pdfpg
400       \fi
401       \ifnum\cb@pdfpg>##1\else
402         \global\def\cb@next{\cb@pdfreadxy{##1}}%
403       \fi
404     \fi
405   }%
406 \fi
407 \cb@next
408 }%
```

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

```

409 \def\cb@pdfparsexy##1.##2p##3/##4/##5pt{%
410   \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.

```

411 \def\cb@defpoint##1##2{%
412   \if@filesw
413     \begingroup
414       \edef\point{{\the##1}{\the\cb@pagecount}}%
415       \let\the=\z@
416       \pdfsavepos
417       \edef\cb@temp{\write\@auxout
418         {\string\cb@pdfxy\point
419         {\the\pdflastxpos}{\the\pdflastypos}{\the##2}}}%
420       \cb@temp
421     \endgroup
422   \fi
423   \cb@trace@defpoint##1##2%
424 }%
```

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

```

425 \def\cb@cvtpct##1{%
```

```

426     \ifnum##1<0 0\else
427     \ifnum##1>99 1\else
428     \ifnum##1<10 0.0\the##1\else
429     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.

```

430   \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.

```

431   \def\cb@connect##1##2##3{%
432     \cb@findpdfpoint{##1}\cb@pagecount
433     \ifx\cb@pdfx\relax\cb@rerun
434     \else
435       \let\cb@pdftopy\cb@pdfy
436       \cb@findpdfpoint{##2}\cb@pagecount
437       \ifx\cb@pdfx\relax\cb@rerun
438       \else

```

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

```

439       \begingroup
440       \cb@dima=\cb@pdfz
441       \advance\cb@dima by-\cb@pdfx
442       \advance\cb@dima by1in%
443       \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.

```

444       \ifx\cb@current@color\undefined
445       \def\cb@temp{\cb@cvtpt\c@changebargrey}%
446       \pdfliteral{q \cb@temp\space g \cb@temp\space G}%
447       \else
448       \pdfliteral{q \cb@current@color}%
449       \fi
450       \edef\cb@temp{\expandafter\cb@removedim\the\cb@dima\space}%
451       \cb@dima=\cb@pdftopy
452       \advance\cb@dima-\cb@pdfy\relax
453       \cb@dima=\cb@pdf@scale\cb@dima\relax
454       ##3=\cb@pdf@scale##3\relax
455       \pdfliteral direct{\expandafter\cb@removedim\the##3 w
456       \cb@temp 0 m
457       \cb@temp \expandafter\cb@removedim\the\cb@dima\space 1 S Q}%
458       \endgroup

```

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

```

459       \cb@cntb=##1\relax
460       \ifodd\cb@cntb\advance\cb@cntb 1\else\advance\cb@cntb -1\fi
461       \cb@findpdfpoint\cb@cntb\cb@pagecount
462       \cb@cntb=##2\relax

```

```

463         \ifodd\cb@cntb\advance\cb@cntb 1\else\advance\cb@cntb -1\fi
464         \cb@findpdfpoint\cb@cntb\cb@pagecount
465     \fi
466 \fi
467 \cb@trace@connect##1##2##3%
468 }%

```

`\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 L^AT_EX.

```

469 \gdef\cb@checkPdfxy##1##2##3##4##5{%
470     \cb@@findpdfpoint{##1}{##2}%
471     \ifdim##3sp=\cb@pdfx\relax
472     \ifdim##4sp=\cb@pdfy\relax
473     \ifdim##5=\cb@pdfz\relax
474     \else
475     \cb@error
476     \fi
477     \else
478     \cb@error
479     \fi
480     \else
481     \cb@error
482     \fi
483 }

```

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

```

484 \def\cb@maxpoint{9999999}
485 \let\cb@resetpoints\relax
486 \or

```

The code for Xe_TE_X is, like for PDF_TE_X, 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.

```

487 \immediate\closeout\cb@writexy
488 \immediate\openin\cb@readxy=\jobname.cb2\relax

```

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

```

489 \def\cb@pdfpoints{}
490 \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.

```

491 \def\cb@findpdfpoint##1##2{%
492     \edef\cb@temp
493     {\noexpand\cb@@findpdfpoint{\the##1}{\the##2}}%
494     \cb@temp
495 }

```

`\pdfliteral` For Xe_TE_X we mimic PDF_TE_X's command `\pdfliteral`.

```

496 \def\pdfliteral##1{\special{pdf:literal ##1}}

```

`\cb@@findpdfpoint` The `\cb@@findpdfpoint` macro finds the coordinates of point #1 on pagecount #2. 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`.

Scanning `\cb@pdfpoints` gets expensive when the list grows: the recursive `\cb@pdffind` re-absorbs the whole remainder of the list as a macro argument at every element it steps over, and a search that fails removes nothing. On a run during which the page breaks move (the situation described at `\cb@checkpage`, where nearly every lookup asks for a pagecount that disagrees with the recorded one and therefore fails) the list only ever grows and every later search scans all of it; with a few thousand bars such a run can take hours. `\cb@pdfxy` already stores every record in a control sequence `\cb@<point>.<pagecount>` while the auxiliary file is replayed, and as long as point numbers are unique that store is an exact index of the list, so we probe it directly. Two details preserve the original behaviour: a record that has been delivered once is removed from the list by `\cb@pdffind`, so a marker `\cb@xyused@<point>.<pagecount>` makes any second lookup fail just as it would have; and the check performed at the end of the document must see every record again, so once `\if@cb@enddocument` is set the markers are ignored and no new ones are recorded. When point numbers are not unique the original code path in `\cb@listfindpdfpoint` is taken instead: later records overwrite earlier ones in the control-sequence store, which makes it unreliable exactly then.

```

497 \def\cb@@findpdfpoint##1##2{%
498   \if@cb@pointsunique
499     \let\cb@pdfx\relax
500     \ifnum##2<0\relax\else
501       \ifcsname cb@##1.##2\endcsname
502         \if@cb@enddocument
503           \edef\cb@temp{\csname cb@##1.##2\endcsname}%
504           \expandafter\cb@parsexyrecord\cb@temp\cb@xynil
505         \else
506           \ifcsname cb@xyused@##1.##2\endcsname
507             \else
508               \edef\cb@temp{\csname cb@##1.##2\endcsname}%
509               \expandafter\cb@parsexyrecord\cb@temp\cb@xynil
510               \global\expandafter\let
511               \csname cb@xyused@##1.##2\endcsname\@empty
512             \fi
513           \fi
514         \fi
515       \fi
516     \else
517       \cb@listfindpdfpoint{##1}{##2}%
518     \fi
519   }

```

`\cb@listfindpdfpoint` If the information is not yet in memory it is read from the .cb2 file.

```

520 \def\cb@listfindpdfpoint##1##2{%
521   \ifnum##2<\cb@pdfpagenr\relax\else
522     \cb@pdfreadxy{##2}%
523   \fi
524   \let\cb@pdfx\relax

```

```

525 \ifx\cb@pdfpoints\@empty\else
526 \ifnum##2<0\relax
527 \else
528 \edef\cb@temp{%
529 \noexpand\cb@pdfind{##1}{##2}\cb@pdfpoints\relax{}}%
530 \cb@temp
531 \fi
532 \fi
533 }

```

`\cb@pdfind` The `\cb@pdfind` recursively searches through `\cb@pdfpoints` to find point #1 on pagecount #2. `\cb@pdfpoints` contains entries of the form `\pointnr\pagecountp<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.

```

534 \def\cb@pdfind##1##2##3.##4p##5/##6/##7pt##8\relax##9{%
535 \def\cb@next{%
536 \cb@pdfind{##1}{##2}##8\relax{##9##3.##4p##5/##6/##7pt}}%
537 \ifnum ##1=##3
538 \ifnum ##2=##4
539 \def\cb@pdfx{##5sp}%
540 \def\cb@pdfy{##6sp}%
541 \def\cb@pdfz{##7pt}%
542 \let\cb@next\relax
543 \gdef\cb@pdfpoints{##9##8}%
544 \fi
545 \fi
546 \ifx\relax##8\relax
547 \let\cb@next\relax
548 \fi
549 \cb@next
550 }%

```

`\cb@pdfready` The `\cb@pdfready` 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.

```

551 \def\cb@pdfready##1{%
552 \let\cb@next\relax
553 \ifeof\cb@ready
554 \global\let\cb@pdfpagenr\cb@maxpoint
555 \else
556 {\endlinechar=-1\read\cb@ready to\cb@temp
557 \ifx\cb@temp\@empty\else
558 \expandafter\cb@pdfparsexy\cb@temp
559 \ifnum\cb@pdfpg<0\else
560 \xdef\cb@pdfpoints{\cb@pdfpoints\cb@temp}%
561 \cb@trace{PDFpoints=\cb@pdfpoints}%
562 \global\let\cb@pdfpagenr\cb@pdfpg
563 \fi
564 \ifnum\cb@pdfpg>##1\else
565 \global\def\cb@next{\cb@pdfready{##1}}%
566 \fi

```

```

567     \fi
568   }%
569   \fi
570   \cb@next
571 }%

```

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

```

572 \def\cb@pdfparsexy##1.##2p##3/##4/##5pt{%
573   \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.

```

574 \def\cb@defpoint##1##2{%
575   \if@filesw
576     \begingroup
577       \edef\point{{\the##1}{\the\cb@pagecount}}%
578       \let\the=\z@
579       \pdfsavepos
580       \edef\cb@temp{\write\@auxout
581         {\string\cb@pdfxy\point
582           {\the\pdflastxpos}{\the\pdflastypos}{\the##2}}}%
583       \cb@temp
584     \endgroup
585   \fi
586   \cb@trace@defpoint##1##2%
587 }%

```

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

```

588 \def\cb@cvtpct##1{%
589   \ifnum##1<0 0\else
590   \ifnum##1>99 1\else
591   \ifnum##1<10 0.0\the##1\else
592   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.

```

593 \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.

```

594 \def\cb@connect##1##2##3{%
595   \cb@findpdfpoint{##1}\cb@pagecount
596   \ifx\cb@pdfx\relax\cb@rerun
597   \else
598     \let\cb@pdftopy\cb@pdfy
599     \cb@findpdfpoint{##2}\cb@pagecount

```

```

600     \ifx\cb@pdfx\relax\cb@rerun
601     \else

```

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

```

602     \begingroup
603     \cb@dima=\cb@pdfz
604     \advance\cb@dima by-\cb@pdfx
605     \advance\cb@dima by1in%
606     \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.

```

607     \ifx\cb@current@color\undefined
608     \def\cb@temp{\cb@cvt@pct\c@change@bar@grey}%
609     \pdfliteral{q \cb@temp\space g \cb@temp\space G}%
610     \else
611     \pdfliteral{q \expandafter\sec@nd@ftw@\cb@current@color\space RG
612     \expandafter\sec@nd@ftw@\cb@current@color\space rg}%
613     \fi
614     \edef\cb@temp{\expandafter\cb@removedim\the\cb@dima\space}%
615     \cb@dima=\cb@pdf@topy
616     \advance\cb@dima-\cb@pdfy\relax
617     \cb@dima=\cb@pdf@scale\cb@dima\relax
618     ##3=\cb@pdf@scale##3\relax
619     \pdfliteral{\expandafter\cb@removedim\the##3 w
620     \cb@temp 0 m
621     \cb@temp \expandafter\cb@removedim\the\cb@dima\space 1 S Q}%
622     \endgroup

```

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

```

623     \cb@cntb=##1\relax
624     \ifodd\cb@cntb\advance\cb@cntb 1\else\advance\cb@cntb -1\fi
625     \cb@findpdfpoint\cb@cntb\cb@pagecount
626     \cb@cntb=##2\relax
627     \ifodd\cb@cntb\advance\cb@cntb 1\else\advance\cb@cntb -1\fi
628     \cb@findpdfpoint\cb@cntb\cb@pagecount
629     \fi
630     \fi
631     \cb@trace@connect##1##2##3%
632     }%

```

`\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 L^AT_EX.

```

633     \gdef\cb@checkPdfxy##1##2##3##4##5{%
634     \cb@findpdfpoint{##1}{##2}%
635     \ifdim##3sp=\cb@pdfx\relax
636     \ifdim##4sp=\cb@pdfy\relax
637     \ifdim##5sp=\cb@pdfz\relax
638     \else
639     \cb@error
640     \fi
641     \else
642     \cb@error

```

```

643     \fi
644   \else
645     \cb@error
646   \fi
647 }

```

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

```

648 \def\cb@maxpoint{9999999}
649 \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.

```

650 \or
651 \immediate\closeout\cb@writexy
652 \immediate\openin\cb@readxy=\jobname.cb2\relax

```

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

```

653 \def\cb@pdfpoints{}
654 \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.

```

655 \def\cb@findpdfpoint##1##2{%
656   \edef\cb@temp
657     {\noexpand\cb@@findpdfpoint{\the##1}{\the##2}}%
658   \cb@temp
659 }

```

`\pdfliteral` For luaTeX we also mimick PDFTeX's command `\pdfliteral`.

```

660 \def\pdfliteral##1{\pdfextension literal {##1}}

```

`\cb@@findpdfpoint` The `\cb@@findpdfpoint` macro finds the coordinates of point #1 on pagecount #2. 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`.

Scanning `\cb@pdfpoints` gets expensive when the list grows: the recursive `\cb@pdffind` re-absorbs the whole remainder of the list as a macro argument at every element it steps over, and a search that fails removes nothing. On a run during which the page breaks move (the situation described at `\cb@checkpage`, where nearly every lookup asks for a pagecount that disagrees with the recorded one and therefore fails) the list only ever grows and every later search scans all of it; with a few thousand bars such a run can take hours. `\cb@pdfxy` already stores every record in a control sequence `\cb@⟨point⟩.⟨pagecount⟩` while the auxiliary file is replayed, and as long as point numbers are unique that store is an exact index of the list, so we probe it directly. Two details preserve the original behaviour: a record that has been delivered once is removed from the list by `\cb@pdffind`, so a marker `\cb@xyused@⟨point⟩.⟨pagecount⟩` makes any second lookup fail just as it would have; and the check performed at the end of the document must see every

record again, so once `\if@cb@enddocument` is set the markers are ignored and no new ones are recorded. When point numbers are not unique the original code path in `\cb@listfindpdfpoint` is taken instead: later records overwrite earlier ones in the control-sequence store, which makes it unreliable exactly then.

```

661 \def\cb@@findpdfpoint##1##2{%
662   \if@cb@pointsunique
663     \let\cb@pdfx\relax
664     \ifnum##2<0\relax\else
665       \ifcsname cb@##1.##2\endcsname
666         \if@cb@enddocument
667           \edef\cb@temp{\csname cb@##1.##2\endcsname}%
668           \expandafter\cb@parsexyrecord\cb@temp\cb@xynil
669         \else
670           \ifcsname cb@xyused@##1.##2\endcsname
671             \else
672               \edef\cb@temp{\csname cb@##1.##2\endcsname}%
673               \expandafter\cb@parsexyrecord\cb@temp\cb@xynil
674               \global\expandafter\let
675               \csname cb@xyused@##1.##2\endcsname\@empty
676             \fi
677           \fi
678         \fi
679       \else
680         \cb@listfindpdfpoint{##1}{##2}%
681       \fi
682     \fi
683   }

```

`\cb@listfindpdfpoint` If the information is not yet in memory it is read from the .cb2 file.

```

684 \def\cb@listfindpdfpoint##1##2{%
685   \ifnum##2<\cb@pdfpagenr\relax\else
686     \cb@pdfreadxy{##2}%
687   \fi
688   \let\cb@pdfx\relax
689   \ifx\cb@pdfpoints\@empty\else
690     \ifnum##2<0\relax
691     \else
692       \edef\cb@temp{%
693         \noexpand\cb@pdfind{##1}{##2}\cb@pdfpoints\relax{}}%
694       \cb@temp
695     \fi
696   \fi
697 }

```

`\cb@pdfind` The `\cb@pdfind` 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.

```

698 \def\cb@pdfind##1##2##3.##4p##5/##6/##7pt##8\relax##9{%

```

```

699 \def\cb@next{%
700   \cb@pdfind{##1}{##2}##8\relax{##9##3.##4p##5/##6/##7pt}}%
701 \ifnum ##1=##3
702   \ifnum ##2=##4
703     \def\cb@pdfx{##5sp}%
704     \def\cb@pdfy{##6sp}%
705     \def\cb@pdfz{##7pt}%
706     \let\cb@next\relax
707     \gdef\cb@pdfpoints{##9##8}%
708   \fi
709 \fi
710 \ifx\relax##8\relax
711   \let\cb@next\relax
712 \fi
713 \cb@next
714 }%

```

`\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.

```

715 \def\cb@pdfreadxy##1{%
716   \let\cb@next\relax
717   \ifeof\cb@readxy
718     \global\let\cb@pdfpagenr\cb@maxpoint
719   \else
720     {\endlinechar=-1\read\cb@readxy to\cb@temp
721     \ifx\cb@temp\@empty\else
722       \expandafter\cb@pdfparsexy\cb@temp
723       \ifnum\cb@pdfpg<0\else
724         \xdef\cb@pdfpoints{\cb@pdfpoints\cb@temp}%
725         \cb@trace{PDFpoints=\cb@pdfpoints}%
726         \global\let\cb@pdfpagenr\cb@pdfpg
727       \fi
728       \ifnum\cb@pdfpg>##1\else
729         \global\def\cb@next{\cb@pdfreadxy{##1}}%
730       \fi
731     \fi
732   }%
733 \fi
734 \cb@next
735 }%

```

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

```

736 \def\cb@pdfparsexy##1.##2p##3/##4/##5pt{%
737   \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.

```

738 \def\cb@defpoint##1##2{%

```

```

739 \if@filesw
740 \begingroup
741 \edef\point{{\the##1}{\the\cb@pagecount}}%
742 \let\the=\z@
743 \savepos
744 \edef\cb@temp{\write\@auxout
745 {\string\cb@pdfxy\point
746 {\the\lastxpos}{\the\lastypos}{\the##2}}}%
747 \cb@temp
748 \endgroup
749 \fi
750 \cb@trace@defpoint##1##2%
751 }%

```

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

```

752 \def\cb@cvtpct##1{%
753 \ifnum##1<0 0\else
754 \ifnum##1>99 1\else
755 \ifnum##1<10 0.0\the##1\else
756 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.

```

757 \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.

```

758 \def\cb@connect##1##2##3{%
759 \cb@findpdfpoint{##1}\cb@pagecount
760 \ifx\cb@pdfx\relax\cb@rerun
761 \else
762 \let\cb@pdftopy\cb@pdfy
763 \cb@findpdfpoint{##2}\cb@pagecount
764 \ifx\cb@pdfx\relax\cb@rerun
765 \else

```

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

```

766 \begingroup
767 \cb@dima=\cb@pdfz
768 \advance\cb@dima by-\cb@pdfx
769 \advance\cb@dima by1in%
770 \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.

```

771 \ifx\cb@current@color\undefined
772 \def\cb@temp{\cb@cvtpct\cb@changebargrey}%
773 \pdfliteral{q \cb@temp\space g \cb@temp\space G}%
774 \else

```

```

775         \pdfliteral{q \cb@current@color}%
776     \fi
777     \edef\cb@temp{\expandafter\cb@removedim\the\cb@dima\space}%
778     \cb@dima=\cb@pdftopy
779     \advance\cb@dima-\cb@pdfy\relax
780     \cb@dima=\cb@pdf@scale\cb@dima\relax
781     ##3=\cb@pdf@scale##3\relax
782     \pdfliteral{\expandafter\cb@removedim\the##3 w
783         \cb@temp 0 m
784         \cb@temp \expandafter\cb@removedim\the\cb@dima\space 1 S Q}%
785 \endgroup

```

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

```

786     \cb@cntb=##1\relax
787     \ifodd\cb@cntb\advance\cb@cntb 1\else\advance\cb@cntb -1\fi
788     \cb@findpdfpoint\cb@cntb\cb@pagecount
789     \cb@cntb=##2\relax
790     \ifodd\cb@cntb\advance\cb@cntb 1\else\advance\cb@cntb -1\fi
791     \cb@findpdfpoint\cb@cntb\cb@pagecount
792 \fi
793 \fi
794 \cb@trace@connect##1##2##3%
795 }%

```

`\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 $\LaTeX$.

```

796 \gdef\cb@checkPdfxy##1##2##3##4##5{%
797     \cb@findpdfpoint{##1}{##2}%
798     \ifdim##3sp=\cb@pdfx\relax
799     \ifdim##4sp=\cb@pdfy\relax
800     \ifdim##5=\cb@pdfz\relax
801     \else
802     \cb@error
803     \fi
804 \else
805     \cb@error
806     \fi
807 \else
808     \cb@error
809     \fi
810 }

```

For $\text{lua}\TeX$ we don't need a limit on the number of bar points.

```

811 \def\cb@maxpoint{9999999}
812 \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.

```

813 \else
814 \PackageWarning{Changebar}{changebars not supported in unknown setup}
815 \def\cb@defpoint##1##2{\cb@trace@defpoint##1##2}
816 \def\cb@connect##1##2##3{\cb@trace@connect##1##2##3}
817 \let\cb@resetpoints\relax
818 \fi

```

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

```
819 \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`.

```
820 \newcommand*{\cbstart}{\@ifnextchar[%]
821   {\cb@start}%
822   {\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 L^AT_EX environment is not possible.

```
823 \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`.

```
824 \newcommand*{\cbdelete}{\@ifnextchar[%]
825   {\cb@delete}%
826   {\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.

```
827 \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` and `\cbend` but they can be used as a L^AT_EX environment to enforce correct nesting. They can *not* be used in the `tabular` and `tabbing` environments.

```
828 \newenvironment{changebar}%
829   {\@ifnextchar[\cb@start}%
830   {\cb@start[\changebarwidth]}}%
831   {\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.

```
832 \newcommand*{\nochangebars}{%
833   \def\cb@start[##1]{\ignorespaces}%
834   \def\cb@delete[##1]{}%
835   \def\cb@end{\ignorespacesafterend}%
836 }
```

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

```
837 \newlength{\changebarwidth}
838 \setlength{\changebarwidth}{2pt}
```

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

```
839 \newlength{\deletebarwidth}
840 \setlength{\deletebarwidth}{4pt}
```

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

```
841 \newlength{\changebarsep}
842 \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).

```
843 \newcounter{changebargrey}
844 \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.

```
845 \ifpackagewith{changebar}{\csname cb@color@pkg\endcsname}{%
846     \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.

```
847 \DeclareRobustCommand\cbcolor{%
848     \@ifnextchar[%]
849         \@undeclaredcbcolor\@declaredcbcolor}
```

`\@undeclaredcbcolor` Call the driver-dependent command `\color@{model}` to define `\cb@current@color`.

```
850 \def\@undeclaredcbcolor[#1]#2{%
851     \begingroup
852     \color[#1]{#2}%
853     \global\let\cb@current@color\current@color
854     \endgroup
855     \ignorespaces
856 }
```

`\@declaredcbcolor`

```
857 \def\@declaredcbcolor#1{%
858     \begingroup
859     \color{#1}%
860     \global\let\cb@current@color\current@color
861     \endgroup
862     \ignorespaces}%
863 }
```

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

```
864 \def\cbcolor{\@ifnextchar[%]
865     \@@cbcolor\@cbcolor}%
```

```

866 \def\cbcolor[#1]#2{\cbcolwarn\def\cbcolor[##1]##2{}}%
867 \def\cbcolor#1{\cbcolwarn\def\cbcolor##1{}}%
868 \def\cbcolwarn{\PackageWarning{Changebar}%
869   {You didn't specify the option 'color';\MessageBreak
870    your command \string\cbcolor\space will be ignored}}%
871 }

```

5.4 Macros for beginning and ending bars

\cbstart 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 **\cbcurrentstack** and then sets the point by calling **\cbsetBeginPoints** with the point number. Finally, it writes the .aux file.

```

872 \def\cbstart[#1]{%
873   \cb@topleft=\cb@nextpoint

```

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

```

874   \cb@curbarwd#1\relax
875   \cb@push\cb@currentstack

```

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

```

876   \cb@checkpage\z@

```

Temporarily assign the page number to **\cb@pagecount** as that register is used by **\cbsetBeginPoints**. Note that its value is offset by one from the page counter.

```

877   \cb@cmta\cb@pagecount
878   \cb@pagecount\cb@page\advance\cb@pagecount\m@ne
879   \ifvmode
880     \cb@setBeginPoints
881   \else
882     \vbox to \z@{%

```

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

```

883       \vskip -\ht\strutbox
884       \cb@setBeginPoints
885       \vskip \ht\strutbox}%
886   \fi

```

Restore **\cb@pagecount**.

```

887   \cb@pagecount\cb@cmta
888   \cb@advancePoint\ignorespaces}

```

\cbadvancePoint The macro **\cbadvancePoint** advances the count register **\cb@nextpoint**. When the maximum number is reached, the numbering is reset. When the numbering is reset point numbers are no longer unique and the fast path in **\cb@checkpage** must be disabled, see the description of **\ifcb@pointsunique**.

```

889 \def\cbadvancePoint{%
890   \global\advance\cb@nextpoint by 4\relax
891   \ifnum\cb@nextpoint>\cb@maxpoint
892     \global\cb@nextpoint=\cb@minpoint\relax
893   \global\cb@pointsuniquefalse

```

894 \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.

```
895 \def\cb@end{%
896   \cb@trace@stack{end of bar on page \the\c@page}%
897   \cb@pop\cb@currentstack
898   \ifnum\cb@topleft=\cb@nil
899     \PackageWarning{Changebar}%
900     {Badly nested changebars; Expect erroneous results}%
901   \else
```

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

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

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

```
903   \cb@cmta\cb@pagecount
904   \cb@pagecount\cb@page\advance\cb@pagecount\m@ne
905   \ifvmode
906     \cb@setEndPoints
907   \else
908     \vadjust{\cb@setEndPoints}%
909   \fi
910   \cb@pagecount\cb@cmta
911 \fi
912 \ignorespacesafterend}
```

`@cb@pointsunique` The fast path in `\cb@checkpage` below relies on every point number occurring at most once in the history file (`changebar.cb`). Point numbers are reused once `\cb@nextpoint` passes `\cb@maxpoint`, which is small for some of the older drivers. This switch is cleared as soon as reuse is detected – either while the history entries are replayed from the .aux file (`\cb@barpoint` sees the same point number twice) or when `\cb@advancePoint` wraps around during the current run – after which all searches use the original code again.

```
913 \newif\if@cb@pointsunique
914 \@cb@pointsuniquefalse
```

`@cb@enddocument` Set at the start of the `\AtEndDocument` processing. The fast lookup in `c\b@@findpdfpoint` below consumes each coordinate record the first time it is delivered, mirroring the removal of a found record from `\cb@pdfpoints`; the verification pass at the end of the document must see every record again, so while this switch is true the consumed-markers are ignored and no new ones are recorded.

```
915 \newif\if@cb@enddocument
```

`\cb@parsexyrecord` Deliver a coordinate record stored by `\cb@pdfxy` (of the form $\langle x \rangle, \langle y \rangle, \langle z \rangle$, where $\langle x \rangle$ and $\langle y \rangle$ are scaled-point numbers and $\langle z \rangle$ carries its unit) in `\cb@pdfx`,

`\cb@pdfy` and `\cb@pdfz`, in exactly the form the list search in `\cb@pdffind` delivers them.

```
916 \def\cb@parsexyrecord#1,#2,#3\cb@xynil{%
917   \def\cb@pdfx{#1sp}\def\cb@pdfy{#2sp}\def\cb@pdfz{#3}}
```

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

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

Searching the history stack is expensive: `\cb@FindPageNum` pops the whole stack onto `\cb@tempstack` and pushes it back afterwards, and every push or pop rebuilds the complete stack macro, so a search that *fails* costs time quadratic in the number of entries still on the stack. Failed searches are rare while the `.aux` data is stable, but on a run during which the page breaks move (in particular the first runs of a document compiled in a fresh directory, where the table of contents and cross references are still missing) `\cb@pushNextActive` may consume entries from the history stack long before the corresponding `\cbstart` or `\cbend` is processed, after which each of these searches fails after scanning the complete stack. With a few hundred change bars such a run takes minutes instead of seconds.

Fortunately, as long as point numbers are unique (see above), the search never needs to run at all. A search *fails* unless the point was read from the history file (`\cb@bp@⟨point⟩` was defined when the `.aux` entries were replayed) and has not yet been consumed by `\cb@pushNextActive` (which defines `\cb@used@⟨point⟩`); a failed search restores all registers and the stack, so skipping it changes nothing. A search that *succeeds* also restores the stack (`\cb@FindPageNum` pushes the scanned entries, including the matched one, back) and every register, except that `\cb@page` receives the page number recorded for the point – and since `\cb@barpoint` now stores that page number in `\cb@bp@⟨point⟩`, the whole successful search can be replaced by one assignment. `\cb@@checkpage` keeps the original search for the non-unique fallback. (`\ifcsname` is available in all engines supported by current L^AT_EX, which requires the e-_T_EX extensions.)

```
918 \def\cb@checkpage#1{%
919   \ifcb@pointsunique
920     \cb@canta\cb@topleft\relax
921     \advance\cb@canta by #1\relax
922     \ifcsname cb@bp@\the\cb@canta\endcsname
923       \ifcsname cb@used@\the\cb@canta\endcsname
924       \else
925         \cb@page\csname cb@bp@\the\cb@canta\endcsname\relax
926       \fi
927     \fi
928   \else
929     \cb@@checkpage{#1}%
930   \fi}
```

`\cb@@checkpage`

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

First store the identifiers in temporary registers.

```
932 \cb@cmta\cb@topleft\relax
933 \advance\cb@cmta by #1\relax
934 \cb@cntb\cb@page\relax
935 \cb@dima\cb@curbarwd\relax
```

Then pop the history stack.

```
936 \cb@pop\cb@historystack
```

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

```
937 \ifnum\cb@topleft=\cb@nil
938 \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.

```
939 \cb@FindPageNum
940 \ifnum\cb@topleft>\cb@maxpoint\else
```

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

```
941 \cb@cntb\cb@page
942 \fi
```

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

```
943 \@whilenum\cb@topleft>\cb@nil\do{%
944 \cb@push\cb@historystack
945 \cb@pop\cb@tempstack}%
946 \fi
```

Finally return the correct values

```
947 \advance\cb@cmta by -#1\relax
948 \cb@topleft\cb@cmta\relax
949 \cb@page\cb@cntb\relax
950 \cb@curbarwd\cb@dima\relax
951 }
```

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

```
952 \def\cb@FindPageNum{%
953 \ifnum\cb@topleft=\cb@cmta
```

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.

```
954 \else
955 \cb@push\cb@tempstack
956 \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.

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

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

```
959 \cb@topleft\cb@maxpoint\advance\cb@topleft\@ne
960 \else
```

Recursively call ourselves.

```

961     \expandafter\expandafter\expandafter\cb@FindPageNum
962     \fi
963     \fi
964 }%
```

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

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

```

965 \def\cb@setBeginPoints{%
966   \cb@topright=\cb@topleft\advance\cb@topright by\@ne
967   \cb@cntb=\cb@pagecount
968   \divide\cb@cntb by\tw@
969   \ifodd\cb@cntb
970     \cb@defpoint\cb@topleft\cb@even@left
971     \cb@defpoint\cb@topright\cb@even@right
972   \else
973     \cb@defpoint\cb@topleft\cb@odd@left
974     \cb@defpoint\cb@topright\cb@odd@right
975   \fi
976   \cb@writeAux\cb@topleft
977 }
```

\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.

```

978 \def\cb@setEndPoints{%
979   \cb@topright=\cb@topleft\advance\cb@topright by\@ne
980   \cb@botleft=\cb@topleft\advance\cb@botleft by\tw@
981   \cb@botright=\cb@topleft\advance\cb@botright by\thr@@
982   \cb@cntb=\cb@pagecount
983   \divide\cb@cntb by\tw@
984   \ifodd\cb@cntb
985     \cb@defpoint\cb@botleft\cb@even@left
986     \cb@defpoint\cb@botright\cb@even@right
987   \else
988     \cb@defpoint\cb@botleft\cb@odd@left
989     \cb@defpoint\cb@botright\cb@odd@right
990   \fi
991   \cb@writeAux\cb@botright
992   \edef\cb@leftbar{%
993     \noexpand\cb@connect{\cb@topleft}{\cb@botleft}{\cb@curbarwd}}%
994   \edef\cb@rightbar{%
995     \noexpand\cb@connect{\cb@topright}{\cb@botright}{\cb@curbarwd}}%
```

In twocolumn pages always use outerbars

```

996   \if@twocolumn
997     \ifodd\cb@pagecount\cb@rightbar\else\cb@leftbar\fi
```

```

998 \else
999 \ifcase\cb@barsplace
0=innerbars
1000 \ifodd\cb@cntb
1001 \cb@rightbar
1002 \else
1003 \if@twoside\cb@leftbar\else\cb@rightbar\fi
1004 \fi
1005 \or
1=outerbars
1006 \ifodd\cb@cntb
1007 \cb@leftbar
1008 \else
1009 \if@twoside\cb@rightbar\else\cb@leftbar\fi
1010 \fi
1011 \or
2=leftbars
1012 \cb@leftbar
1013 \or
3=rightbars
1014 \cb@rightbar
1015 \fi
1016 \fi
1017 }%

```

\cb@writeAux The macro `\cb@writeAux` writes information about a changebar point to the auxiliary file. The number of the point, the pagenummer 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.

```

1018 \def\cb@writeAux#1{%
1019 \if@filesw
1020 \begingroup
1021 \edef\point{\the#1}%
1022 \edef\level{\the\cb@curbarwd}%
1023 \let\the=\z@
1024 \edef\cb@temp{\write\@auxout
1025 {\string\cb@barpoint{\point}{\the\cb@pagecount}{\level}}}%
1026 \cb@temp
1027 \endgroup
1028 \fi}

```

5.5 Macros for Making It Work Across Page Breaks

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

```
1029 \newif\if@cb@pagejump
```

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

```
1030 \def\cb@pagejumplist{-1}
```

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

```
1031 \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.

```
1032 \def\cb@pagejump#1{\xdef\cb@pagejumplist{\cb@pagejumplist,#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.

```
1033 \def\cb@writepagejump#1{
1034   \cb@cntb=\cb@pagecount
1035   \advance\cb@cntb by#1\relax
1036   \global\setbox\@leftcolumn\vbox to\@colht{%
1037     \edef\cb@temp{\write\@auxout{\string\cb@pagejump{the\cb@cntb}}}%
1038     \cb@temp
1039     \dimen@ \dp\@leftcolumn
1040     \unvbox \@leftcolumn
1041     \vskip -\dimen@
1042   }%
1043 }
```

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

```
1044 \def\cb@poppagejump#1,#2\relax{%
1045   \gdef\cb@nextpagejump{#1}%
1046   \gdef\cb@pagejumplist{#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.

```
1047 \def\cb@checkpagecount{%
1048   \if@twocolumn
1049     \if@firstcolumn
1050       \ifodd\cb@pagecount\global\advance\cb@pagecount by\@ne\fi
1051     \fi
1052   \else
1053     \ifodd\cb@pagecount\global\advance\cb@pagecount by\@ne\fi
1054   \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.

```

1055 \if@twoside
1056   \cb@cntb=\cb@pagecount
1057   \divide\cb@cntb by\tw@
1058   \advance\cb@cntb by-\c@page
1059   \ifodd\cb@cntb

```

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

```

1060   \if@twocolumn
1061     \if@firstcolumn
1062       \@whilenum\cb@pagecount>\cb@nextpagejump\do{%
1063         \expandafter\cb@poppagejump\cb@pagejump\st\relax}%
1064       \ifnum\cb@pagecount=\cb@nextpagejump
1065         \cb@trace{Page jump: \string\cb@pagecount=\the\cb@pagecount}
1066         \global\advance\cb@pagecount by\tw@
1067         \global\@cb@pagejumptrue
1068       \else
1069         \global\@cb@pagejumpfalse
1070       \fi
1071     \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.

```

1072       \if@cb@pagejump
1073       \cb@writepagejump{-3}%
1074     \fi
1075   \fi
1076 \fi
1077 \else

```

Here `\cb@pagecount` is incorrect.

```

1078   \if@twocolumn
1079     \if@firstcolumn
1080       \global\advance\cb@pagecount by\tw@
1081       \global\@cb@pagejumpfalse
1082     \else
1083       \if@cb@pagejump
1084         \cb@trace{Page jump annulled, %
1085           \string\cb@pagecount=\the\cb@pagecount}
1086       \else
1087         \cb@writepagejump{-1}%
1088         \global\advance\cb@pagecount by\tw@
1089         \cb@rerun
1090       \fi
1091     \fi
1092   \else
1093     \global\advance\cb@pagecount by\tw@
1094   \fi
1095 \fi
1096 \fi
1097 }

```

`\@makecol` These internal L^AT_EX macros are modified in order to end the changebars spanning the current page break (if any) and restart them on the next page. The modifications are needed to reset the special points for this page and add begin bars to top of box255. The bars carried over from the previous page, and hence to be restarted on this page, have been saved on the stack `\cb@beginstack`. This stack is used to define new starting points for the change bars, which are added to the top of box `\@cc1v`. Then the stack `\cb@endstack` is built and processed by `\cb@processActive`. Finally the original `\@makecol` (saved as `\cb@makecol`) is executed.

```

1098 \let\ltx@makecol\@makecol
1099 \def\cb@makecol{%
1100   \if@twocolumn
1101     \cb@trace{Twocolumn: \if@firstcolumn Left \else Right \fi column}%
1102   \fi
1103   \cb@trace@stack{before makecol, page \the\c@page,
1104                  \string\cb@pagecount=\the\cb@pagecount}%
1105   \let\cb@writeAux\@gobble

```

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

```

1106   \cb@checkpagecount
1107   \setbox\@cc1v \vbox{%
1108     \cb@resetpoints
1109     \cb@startSpanBars
1110     \unvbox\@cc1v
1111     \boxmaxdepth\maxdepth}%
1112   \global\advance\cb@pagecount by\@ne
1113   \cb@buildstack\cb@processActive
1114   \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`.

```

1115   \if@twocolumn
1116     \global\setbox\@outputbox \vbox to\@colht {%
1117       \if@firstcolumn\write\@auxout{\string\cb@firstcolumntrue}%
1118       \else\write\@auxout{\string\cb@firstcolumnfalse}%
1119       \fi
1120       \dimen@ \dp\@outputbox
1121       \unvbox \@outputbox
1122       \vskip -\dimen@
1123     }%
1124   \fi
1125   \cb@trace@stack{after makecol, page \the\c@page,
1126                  \string\cb@pagecount=\the\cb@pagecount}%
1127 }
1128 \let\@makecol\cb@makecol

```

When L^AT_EX 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.

```

1129 \let\ltx@vtryfc\@vtryfc

```

```

1130 \def\cb@vtryfc#1{%
1131   \cb@trace{In vtryfc, page \the\c@page,
1132             \string\cb@pagecount=\the\cb@pagecount}%
1133   \let\cb@writeAux\@gobble
1134   \cb@checkpagecount
1135   \ltx@vtryfc{#1}%
1136   \if@twocolumn
1137     \global\setbox\@outputbox \vbox to\@colht{%
1138       \if@firstcolumn\write\@auxout{\string\cb@firstcolumntrue}%
1139       \else\write\@auxout{\string\cb@firstcolumnfalse}%
1140       \fi
1141       \unvbox\@outputbox
1142       \boxmaxdepth\maxdepth
1143     }%
1144   \fi
1145   \global\advance\cb@pagecount by \@ne
1146 }
1147 \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).

```

```

1148 \def\cb@processActive{%
1149   \cb@pop\cb@endstack
1150   \ifnum\cb@topleft=\cb@nil
1151   \else
1152     \setbox\@cclv\vbox{%
1153       \unvbox\@cclv
1154       \boxmaxdepth\maxdepth
1155       \advance\cb@pagecount by -1\relax
1156       \cb@setEndPoints}%
1157     \cb@push\cb@historystack
1158     \cb@push\cb@beginstack
1159     \expandafter\cb@processActive
1160   \fi}

```

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

```

1161 \def\cb@startSpanBars{%
1162   \cb@pop\cb@beginstack
1163   \ifnum\cb@topleft=\cb@nil

```

```

1164 \else
1165   \cb@setBeginPoints
1166   \cb@trace@stack{after StartSpanBars, page \the\c@page}%
1167   \expandafter\cb@startSpanBars
1168 \fi
1169 }

```

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

```

1170 \def\cb@buildstack{%
1171   \cb@initstack\cb@endstack
1172   \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. Each point that is consumed here is recorded by defining `\cb@used@{point}`, so that `\cb@checkpage` knows a search for it can no longer succeed.

```

1173 \def\cb@pushNextActive{%
1174   \cb@pop\cb@historystack
1175   \ifnum\cb@toleft=\cb@nil
1176   \else
1177     \ifnum\cb@page>\cb@pagecount
1178       \cb@push\cb@historystack
1179     \else
1180       \expandafter\gdef\csname cb@used@\the\cb@toleft\endcsname{}%
1181       \ifodd\cb@toleft
1182         \cb@push\cb@endstack
1183       \else
1184         \cb@pop\cb@endstack
1185       \fi
1186       \expandafter\expandafter\expandafter\cb@pushNextActive
1187     \fi
1188 \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 `xxxnyyyppzzzl` 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.
1189 \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 1190 \cb@initstack\cb@historystack
We also need to allocate a read and a write stream for the history file.
1191 \newwrite\cb@write
1192 \newread\cb@read
And we open the history file for writing (which is done when the .aux file is read
in).
1193 \immediate\openout\cb@write=\jobname.cb\relax

\cb@endstack Allocate two stacks for the bars that span the current page break.
\cb@beginstack 1194 \cb@initstack\cb@endstack
1195 \cb@initstack\cb@beginstack

\cb@tempstack Allocate a stack for temporary storage
1196 \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.
1197 \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.
1198 \def\cb@thehistorystack{\cb@historystack}
1199 \def\cb@pop#1{%
1200   \ifx #1\@empty
1201     \def\cb@temp{#1}%
1202     \ifx\cb@temp\cb@thehistorystack
1203       \ifeof\cb@read
1204       \else
1205         {\endlinechar=-1\read\cb@read to\cb@temp
1206         \xdef\cb@historystack{\cb@historystack\cb@temp}%
1207       }%
1208     \fi
```

```

1209     \fi
1210 \fi
1211 \ifx#1\@empty
1212     \global\cb@topleft\cb@nil
1213     \global\cb@page\z@\relax
1214 \else
1215     \expandafter\cb@carcdr#1e#1%
1216 \fi
1217 \cb@trace@pop{#1}}

```

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

```

1218 \def\cb@carcdr#1n#2p#3l#4e#5{%
1219     \global\cb@topleft#1\relax
1220     \global\cb@page#2\relax
1221     \global\cb@curbarwd#3\relax
1222     \xdef#5{#4}}

```

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

```

1223 \def\cb@push#1{%
1224     \xdef#1{\the\cb@topleft n\the\cb@page p\the\cb@curbarwd l#1}%
1225     \cb@trace@push{#1}}
1226

```

\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. Each point that goes into the history file is recorded by defining **\cb@bp@⟨point⟩**, whose replacement text is the (column-adjusted) pagecount recorded for the point – exactly what a successful history search would deliver, so **\cb@checkpage** can skip the search altogether. When a point number shows up for the second time the previous run reused point numbers and the fast paths are disabled.

```

1227 \def\cb@barpoint#1#2#3{%
1228     \cb@cmta=#2
1229     \ifcb@firstcolumn\advance\cb@cmta by\m@ne\fi
1230     \ifcsname cb@bp@#1\endcsname
1231         \global\@cb@pointsuniquefalse
1232     \else
1233         \expandafter\xdef\csname cb@bp@#1\endcsname{\the\cb@cmta}%
1234     \fi
1235     \immediate\write\cb@write{#1n\the\cb@cmta p#3l}}

```

5.7 Macros For Checking That The .aux File Is Stable

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

For these actions we need to add to the L^AT_EX begin-document hook.

```

1236 \AtBeginDocument{%
1237     \cb@setup@specials

```

Add a sentinel to \cb@pagejump1st.

```
1238 \cb@pagejump{999999999,}%
```

Compute the left and right positions of the changebars.

```
1239 \cb@positions
1240 \cb@trace{%
1241   Odd left : \the\cb@odd@left\space
1242   Odd right : \the\cb@odd@right\MessageBreak
1243   Even left: \the\cb@even@left\space
1244   Even right: \the\cb@even@right
1245 }%
1246 \immediate\closeout\cb@write
1247 \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 L^AT_EX 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.

```
1248 \AtEndDocument{%
1249   \global\@cb@enddocumenttrue

1250   \clearpage
1251   \cb@initstack\cb@historystack
1252   \immediate\closein\cb@read
1253   \immediate\openin\cb@read=\jobname.cb\relax
```

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

```
1254 \ifx\cb@readyx\@undefined
1255 \else
1256   \immediate\closein\cb@readyx
1257   \immediate\openin\cb@readyx=\jobname.cb2\relax
1258   \def\cb@pdfpoints{}%
1259   \def\cb@pdfpagenr{0}%
1260 \fi
1261 \@cb@firstcolumnfalse
1262 \cb@checkrerun
1263 \let\cb@pdfxy\cb@checkPdfxy
1264 \let\cb@pagejump\@gobble
1265 \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.

```
1266 \def\cb@checkHistory#1#2#3{%
1267   \cb@pop\cb@historystack
1268   \ifnum #1=\cb@topleft\relax
1269     \cb@cna=#2
1270     \ifcb@firstcolumn\advance\cb@cna by\m@ne\fi
1271     \ifnum \cb@cna=\cb@page\relax
```

Both page and point numbers are equal; do nothing,

```
1272   \else
```

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

```
1273     \cb@error
1274     \fi
1275     \else
```

when point numbers don't match.

```
1276     \cb@error
1277     \fi}
```

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

```
1278 \def\cb@checkPdfxy#1#2#3#4#5{}
```

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

```
1279 \def\cb@rerun{%
1280   \global\let\cb@checkrerun\cb@error}
1281 \let\cb@checkrerun\relax
```

`\cb@error` When a mismatch between the changebar information in the auxiliary file and the history stack is detected a warning is issued; further checking is disabled. For `pdfTeX` and `XeTeX` we also disable `\cb@checkPdfxy`.

```
1282 \def\cb@error{%
1283   \PackageWarning{Changebar}%
1284     {Changebar info has changed.\MessageBreak
1285     Rerun to get the bars right}
1286   \gdef\cb@checkHistory##1##2##3{}%
1287   \let\cb@barpoint\cb@checkHistory
1288   \gdef\cb@checkPdfxy##1##2##3##4##5{}%
1289   \let\cb@pdfxy\cb@checkPdfxy}
```

5.8 Macros For Making It Work With Nested Floats/Footnotes

`\end@float` This is a replacement for the $\text{\LaTeX}$ -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 $\text{\LaTeX}$ `\end@float`.

```
1290 \let\ltx@end@float\end@float
1291 \def\cb@end@float{%
1292   \cb@trace@stack{end float on page \the\c@page}%
1293   \cb@pop\cb@currentstack
1294   \ifnum\cb@toleft=\cb@nil
1295     \else
1296       \cb@push\cb@currentstack
1297       \global\cb@curbarwd=\cb@curbarwd
1298       \@endfloatbox
1299       \global\setbox\@currbox
1300         \color@vbox
1301         \normalcolor
1302         \vbox\bgroup\cb@start[\cb@curbarwd]\unvbox\@currbox\cb@end
1303     \fi
1304   \ltx@end@float}
1305 \let\end@float\cb@end@float
```

This only works if this new version of `\end@float` is really used. With L^AT_EX 2.09 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 L^AT_EX 2_ε 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`.

```
1306 \let\ltx@xympar\@xympar
1307 \def\@xympar{%
1308   \let\end@float\ltx@end@float
1309   \ltx@xympar
1310   \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`.

```
1311 \let\flt@float@end\float@end
Then we redefine it to insert the changebarcode.
1312 \def\float@end{%
1313   \cb@trace@stack{end float on page \the\c@page}%
1314   \cb@pop\cb@currentstack
1315   \ifnum\cb@topleft=\cb@nil
1316   \else
1317     \cb@push\cb@currentstack
1318     \global\cb@curbarwd\cb@curbarwd
1319     \@endfloatbox
1320     \global\setbox\@currbox
1321       \color@vbox
1322       \normalcolor
1323       \vbox\bgroup\cb@start[\cb@curbarwd]\unvbox\@currbox\cb@end
1324   \fi
1325   \let\end@float\ltx@end@float
1326   \flt@float@end
1327 }
```

`\end@dblfloat` This is a replacement for the L^AT_EX-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 L^AT_EX macro had to be rewritten.

```
1328 \let\ltx@end@dblfloat\end@dblfloat
1329 \def\cb@end@dblfloat{%
1330   \if@twocolumn
1331     \cb@trace@stack{end dblfloat on page \the\c@page}%
1332     \cb@pop\cb@currentstack
1333     \ifnum\cb@topleft=\cb@nil
```

```

1334 \else
1335 \cb@push\cb@currentstack
1336 \global\cb@curbarwd=\cb@curbarwd
1337 \@endfloatbox
1338 \global\setbox\@currbox
1339 \color@vbox
1340 \normalcolor
1341 \vbox\bgroup\cb@start[\cb@curbarwd]\unvbox\@currbox\cb@end
1342 \fi
1343 \@endfloatbox
1344 \ifnum\@floatpenalty <\z@
1345 \@largefloatcheck
1346 \@cons\@dbldeferlist\@currbox
1347 \fi
1348 \ifnum \@floatpenalty =-\@Mii \@Esphack\fi
1349 \else
1350 \end@float
1351 \fi}
1352 \let\end@dblfloat\cb@end@dblfloat

```

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

```

1353 \let\flt@float@dblend\float@dblend
1354 \def\float@dblend{%
1355 \cb@trace@stack{end dbl float on page \the\c@page}%
1356 \cb@pop\cb@currentstack
1357 \ifnum\cb@toleft=\cb@nil
1358 \else
1359 \cb@push\cb@currentstack
1360 \global\cb@curbarwd=\cb@curbarwd
1361 \@endfloatbox
1362 \global\setbox\@currbox
1363 \color@vbox
1364 \normalcolor
1365 \vbox\bgroup\cb@start[\cb@curbarwd]\unvbox\@currbox\cb@end
1366 \fi
1367 \let\end@dblfloat\ltx@end@dblfloat
1368 \flt@float@dblend
1369 }

```

`\@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.

```

1370 \let\ltx@footnotetext\@footnotetext
1371 \long\def\cb@footnotetext#1{%
1372 \cb@trace@stack{end footnote on page \the\c@page}%
1373 \cb@pop\cb@currentstack
1374 \ifnum\cb@toleft=\cb@nil
1375 \ltx@footnotetext{#1}%
1376 \else
1377 \cb@push\cb@currentstack
1378 \edef\cb@temp{\the\cb@curbarwd}%
1379 \ltx@footnotetext{\cb@start[\cb@temp]#1\cb@end}%

```

```

1380 \fi}
1381 \let\@footnotetext\cb@footnotetext

```

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

```

1382 \let\ltx@mpfootnotetext\@mpfootnotetext
1383 \long\def\cb@mpfootnotetext#1{%
1384   \cb@pop\cb@currentstack
1385   \ifnum\cb@toleft=\cb@nil
1386     \ltx@mpfootnotetext{#1}%
1387   \else
1388     \cb@push\cb@currentstack
1389     \edef\cb@temp{\the\cb@curbarwd}%
1390     \ltx@mpfootnotetext{\cb@start[\cb@temp]#1\cb@end}%
1391   \fi}
1392 \let\@mpfootnotetext\cb@mpfootnotetext
1393 </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.

Symbols		
<code>\@cbcolor</code> ... 863, 864	<code>\@cclv</code> 1105,	<code>\@undeclaredcbcolor</code> 847, <u>848</u>
<code>\@auxout</code> 417,	1108, 1150, 1151	<code>\@undefined</code> .. 65, 66,
579, 742, 1022,	<code>\@colht</code> 1034, 1114, 1135	69, 86, 89, 106,
1035, 1115,	<code>\@currbox</code> 1297, 1300,	109, 150, 151,
1116, 1136, 1137	1318, 1321,	152, 180, 182,
<code>\@cb@enddocument</code> .. <u>913</u>	1336, 1339,	444, 606, 769, 1252
<code>\@cb@enddocumenttrue</code>	1344, 1360, 1363	<code>\@vtryfc</code> <u>1096</u>
..... 1247	<code>\@dbldeferlist</code> ... 1344	<code>\@xympar</code> <u>1304</u>
<code>\@cb@firstcolumn</code> .. <u>20</u>	<code>\@declaredcbcolor</code> .	
<code>\@cb@firstcolumnfalse</code>	 847, <u>855</u>	
.. 1116, 1137, 1259	<code>\@endfloatbox</code>	A
<code>\@cb@firstcolumntrue</code>	... 1296, 1317,	<code>\AtBeginDocument</code> . <u>1234</u>
..... 1115, 1136	1335, 1341, 1359	<code>\AtEndDocument</code> ... <u>1246</u>
<code>\@cb@pagejump</code> <u>1027</u>	<code>\@floatpenalty</code>	
<code>\@cb@pagejumpfalse</code> .	 1342, 1346	B
..... 1067, 1079	<code>\@footnotetext</code> ... <u>1368</u>	<code>\boxmaxdepth</code>
<code>\@cb@pagejumptrue</code> 1065	<code>\@ifnextchar</code> .. 818,	.. 1109, 1140, 1152
<code>\@cb@pointsunique</code> . <u>911</u>	822, 827, 846, 862	
<code>\@cb@pointsuniquefalse</code>	<code>\@ifpackagewith</code> ... 843	C
..... 891, 1229	<code>\@largefloatcheck</code> 1343	<code>\c@changebargrey</code> ..
<code>\@cb@pointsuniquetrue</code>	<code>\@leftcolumn</code>	 445, 607, 770
..... 912	.. 1034, 1037, 1038	<code>\c@lor@to@ps</code> .. 138, 141
<code>\@cb@trace</code> <u>19</u>	<code>\@makecol</code> <u>1096</u>	<code>\c@page</code> .. 894, 1056,
<code>\@cb@tracefalse</code> ... 127	<code>\@mpfootnotetext</code> . <u>1380</u>	1101, 1123,
<code>\@cb@tracetrue</code> 126	<code>\@outputbox</code>	1129, 1164,
<code>\@cbcolor</code> 863, 865	... 1114, 1118,	1290, 1311,
	1119, 1135, 1139	1329, 1353, 1370
	<code>\@preamblecmds</code> 221	<code>\cb@@checkpage</code> 927, <u>929</u>

<code>\cb@findpdfpoint</code> .	<code>\cb@color@pkg</code>	<code>\cb@end@dblfloat</code> . .
. 332 , 335 ,	 139 , 142 , 844	 1327 , 1350
470 , 493 , 497 ,	<code>\cb@colwarn</code> 864 , 865 , 866	<code>\cb@end@float</code>
633 , 656 , 660 , 795	<code>\cb@connect</code> 234 , 991 , 993	 1289 , 1303 , 1308
<code>\cb@show@stack</code> 129 , 168	<code>\cb@curbarwd</code> 14 , 872 ,	<code>\cb@endstack</code>
<code>\cb@advancePoint</code> . .	933 , 948 , 991 ,	 172 , 1147 , 1168 ,
. 886 , 887	993 , 1020 , 1219 ,	 1180 , 1182 , 1192
<code>\cb@barpoint</code> . . 1023 ,	1222 , 1295 ,	<code>\cb@error</code> 475 ,
1225 , 1263 , 1285	1300 , 1316 ,	 478 , 481 , 638 ,
<code>\cb@barsplace</code> 18 , 122 ,	1321 , 1334 ,	 641 , 644 , 800 ,
123 , 124 , 125 , 997	1339 , 1358 ,	 803 , 806 , 1271 ,
<code>\cb@beginstack</code> 173 ,	1363 , 1376 , 1387	 1274 , 1278 , 1280
1156 , 1160 , 1192	<code>\cb@current@color</code> .	<code>\cb@even@left</code> 22 , 28 ,
<code>\cb@botleft</code> 7 ,	 138 ,	 37 , 38 , 39 , 40 ,
978 , 983 , 986 , 991	 141 , 444 , 448 ,	 45 , 968 , 983 , 1241
<code>\cb@botright</code> . 7 , 979 ,	 606 , 610 , 611 ,	<code>\cb@even@right</code> . 22 ,
984 , 987 , 989 , 993	 769 , 773 , 851 , 858	 38 , 41 , 42 , 43 ,
<code>\cb@buildstack</code>	<code>\cb@currentstack</code> . .	 46 , 969 , 984 , 1242
. 1111 , 1168	 171 , 873 ,	<code>\cb@FindPageNum</code> 937 , 950
<code>\cb@carcdr</code> . 1213 , 1216	 895 , 1195 , 1291 ,	<code>\cb@findpdfpoint</code> . .
<code>\cb@checkHistory</code> . .	 1294 , 1312 ,	 330 , 432 ,
. 1263 ,	 1315 , 1330 ,	 436 , 461 , 464 ,
. 1264 , 1284 , 1285	 1333 , 1354 ,	 491 , 594 , 598 ,
<code>\cb@checkpage</code>	 1357 , 1371 ,	 624 , 627 , 654 ,
. 874 , 900 , 916	 1375 , 1382 , 1386	 757 , 761 , 786 , 789
<code>\cb@checkpagecount</code> .	<code>\cb@cvtptct</code> 425 , 445 ,	<code>\cb@footnotetext</code> . .
. 1045 , 1104 , 1132	 587 , 607 , 750 , 770	 1369 , 1379
<code>\cb@checkPdfxy</code> 469 ,	<code>\cb@defpoint</code> 234 , 968 ,	<code>\cb@historystack</code> . .
632 , 794 , 1261 ,	 969 , 971 , 972 ,	 174 , 934 ,
1276 , 1286 , 1287	 983 , 984 , 986 , 987	 942 , 954 , 1155 ,
<code>\cb@checkrerun</code>	<code>\cb@delete</code>	 1172 , 1176 ,
. 1260 , 1278 , 1279	 823 , 824 , 825 , 832	 1188 , 1196 ,
<code>\cb@cnta</code> 11 , 875 ,	<code>\cb@dima</code> 11 , 440 ,	 1204 , 1249 , 1265
. 885 , 901 , 908 ,	 441 , 442 , 443 ,	<code>\cb@initstack</code> . 1169 ,
. 918 , 919 , 920 ,	 450 , 451 , 452 ,	 1187 , 1188 ,
. 921 , 923 , 930 ,	 453 , 457 , 602 ,	 1192 , 1193 ,
. 931 , 945 , 946 ,	 603 , 604 , 605 ,	 1194 , 1195 , 1249
. 951 , 1226 , 1227 ,	 613 , 614 , 615 ,	<code>\cb@leftbar</code>
. 1231 , 1233 ,	 616 , 620 , 765 ,	 990 , 995 , 1001 ,
. 1267 , 1268 , 1269	 766 , 767 , 768 ,	 1005 , 1007 , 1010
<code>\cb@cntb</code> 11 , 459 , 460 ,	 775 , 776 , 777 ,	<code>\cb@listfindpdfpoint</code>
. 461 , 462 , 463 ,	 778 , 782 , 933 , 948	 355 , 358 ,
. 464 , 622 , 623 ,	<code>\cb@driver@setup</code> . .	 517 , 520 , 680 , 683
. 624 , 625 , 626 ,	 50 , 51 ,	<code>\cb@luatexcheck</code> . . .
. 627 , 784 , 785 ,	 52 , 53 , 54 , 55 ,	 103 , 104 , 105 , 218
. 786 , 787 , 788 ,	 56 , 57 , 58 , 59 ,	<code>\cb@luatexerror</code> 106 , 118
. 789 , 932 , 939 ,	 60 , 61 , 68 , 88 ,	<code>\cb@makecol</code> . 1097 , 1126
. 947 , 965 , 966 ,	 108 , 185 , 209 ,	<code>\cb@maxpoint</code>
. 967 , 980 , 981 ,	 210 , 211 , 212 ,	 1 , 243 , 391 , 484 ,
. 982 , 998 , 1004 ,	 213 , 214 , 215 , 235	 553 , 647 , 716 ,
. 1032 , 1033 ,	<code>\cb@end</code> 821 , 825 , 829 ,	 809 , 889 , 938 , 957
. 1035 , 1054 ,	 833 , 893 , 1300 ,	<code>\cb@minpoint</code>
. 1055 , 1056 , 1057	 1321 , 1339 ,	 3 , 6 , 243 , 890
	 1363 , 1377 , 1388	

<code>\cb@mpfootnotetext</code> .	1030, 1044, 1061	<code>\cb@pdfz</code> 378, 440, 473,
..... 1381, 1390		540, 602, 636,
<code>\cb@next</code> .. 372, 379,	. 342, 347, 504,	703, 765, 798, 915
384, 386, 389,	509, 667, 672, <u>914</u>	<code>\cb@pop</code> ... 895, 934,
402, 407, 534,	<code>\cb@pdf@scale</code>	943, 954, 1147,
541, 546, 548,	 <u>430</u> , 443,	1160, 1172,
551, 564, 569,	453, 454, <u>592</u> ,	1182, <u>1196</u> ,
697, 704, 709,	605, 616, 617,	1265, 1291,
711, 714, 727, 732	<u>755</u> , 768, 778, 779	1312, 1330,
<code>\cb@nextpagejump</code> ..	<code>\cb@pdf@find</code> 366, <u>371</u> ,	1354, 1371, 1382
..... <u>1029</u> ,	528, <u>533</u> , 691, <u>696</u>	<code>\cb@poppagejump</code> ...
1043, 1060, 1062	<code>\cb@pdf@pagenr</code>	 <u>1042</u> , 1061
<code>\cb@nextpoint</code> ... <u>5</u> ,	. <u>328</u> , 359, 391,	<code>\cb@positions</code> . <u>22</u> , 1237
871, 888, 889, 890	399, <u>489</u> , 521,	<code>\cb@processActive</code> .
<code>\cb@nil</code> . <u>4</u> , 896, 935,	553, 561, <u>652</u> ,	 <u>1111</u> , <u>1146</u>
941, 955, 1148,	684, 716, 724, 1257	<code>\cb@ps@color</code>
1161, 1173,	<code>\cb@pdf@parsexy</code>	. 136, 138, 141,
1210, 1292,	 395, <u>409</u> ,	253, 270, 298, 316
1313, 1331,	557, <u>571</u> , 720, <u>734</u>	<code>\cb@push</code> .. 873, 942,
1355, 1372, 1383	<code>\cb@pdfpg</code> . 396, 399,	953, 1155, 1156,
<code>\cb@odd@left</code>	401, 410, 558,	1176, 1180,
... <u>22</u> , 27, 28,	561, 563, 572,	<u>1221</u> , 1294,
29, 30, 34, 35,	721, 724, 726, 735	1315, 1333,
45, 971, 986, 1239	<code>\cb@pdf@points</code>	1357, 1375, 1386
<code>\cb@odd@right</code> .. <u>22</u> ,	. <u>328</u> , 363, 366,	<code>\cb@pushNextActive</code> .
30, 31, 32, 33,	380, 397, 398,	 <u>1170</u> , <u>1171</u>
46, 972, 987, 1240	<u>489</u> , 525, 528,	<code>\cb@read</code>
<code>\cb@page</code> <u>15</u> , 876, 902,	542, 559, 560,	<u>1188</u> ,
923, 932, 939,	652, 688, 691,	1201, 1203,
947, 1175, 1211,	705, 722, 723, 1256	1245, 1250, 1251
1218, 1222, 1269	<code>\cb@pdf@ready</code>	<code>\cb@readyxy</code> 71, 91, 111,
<code>\cb@pagecount</code> .. <u>15</u> ,	 360, <u>388</u> ,	327, 390, 393,
228, 233, 414,	522, <u>550</u> , 685, <u>713</u>	488, 552, 555,
432, 436, 461,	<code>\cb@pdf@texcheck</code> ...	651, 715, 718,
464, 576, 594,	... 62, 63, 64, 216	1252, 1254, 1255
598, 624, 627,	<code>\cb@pdf@texerror</code> ...	<code>\cb@removedim</code> <u>49</u> , 247,
739, 757, 761,	 65, 66, 77, 78	254, 263, 271,
786, 789, 875,	<code>\cb@pdf@copy</code> 435, 451,	291, 299, 309,
876, 885, 901,	597, 614, 760, 776	317, 450, 455,
902, 908, 965,	<code>\cb@pdfx</code> 337, 362, 376,	457, 613, 618,
980, 995, 1023,	433, 437, 441,	620, 775, 780, 782
1032, 1048,	471, 499, 524,	<code>\cb@rerun</code> 433,
1051, 1054,	538, 595, 599,	437, 595, 599,
1060, 1062,	603, 634, 662,	758, 762, 1087, <u>1277</u>
1063, 1064,	687, 701, 758,	<code>\cb@resetpoints</code> ...
1078, 1083,	762, 766, 796, 915	 <u>234</u> , 1106
1086, 1091,	<code>\cb@pdfxy</code>	<code>\cb@rightbar</code>
1102, 1110,	 <u>21</u> ,	. 992, 995, 999,
1124, 1130,	74, 94, 114, 418,	1001, 1007, 1012
1143, 1153, 1175	580, 743, 1261, 1287	<code>\cb@setBeginPoints</code> .
<code>\cb@pagejump</code> .. <u>1030</u> ,	<code>\cb@pdfy</code> 377, 435, 452,	878, 882, <u>963</u> , 1163
1035, 1236, 1262	472, 539, 597,	<code>\cb@setEndPoints</code> ..
<code>\cb@pagejumplist</code> . <u>1028</u>	615, 635, 702,	904, 906, <u>976</u> , 1154
<code>\cb@pagejumplst</code> 1028,	760, 777, 797, 915	<code>\cb@setup@specials</code> .
		 <u>224</u> , 1235

<code>\cb@start</code> . 819, 820,	. 130, 132, 169,	40, 43, 820, 828, 835
825, 827, 828,	189 , 225, 230,	<code>\color</code> 850, 857
831, 870 , 1300,	398, 560, 723,	<code>\color@vbox</code> . . . 1298,
1321, 1339,	956, 1063, 1082,	1319, 1337, 1361
1363, 1377, 1388	1099, 1129, 1238	<code>\current@color</code> 851, 858
<code>\cb@startSpanBars</code> .	<code>\cb@trace@connect</code> .	<code>\CurrentOption</code> 145
. 1107 , 1159	 229 ,	
<code>\cb@temp</code> 331,	241, 258, 277,	D
333, 341, 342,	285, 304, 323,	<code>\DeclareOption</code>
346, 347, 366,	467, 630, 792, 814	. 50, 51, 52, 53,
367, 393, 394,	<code>\cb@trace@defpoint</code> .	54, 55, 56, 57,
395, 397, 417,	 224 ,	58, 59, 60, 61,
420, 445, 446,	238, 250, 267,	62, 63, 83, 84,
450, 456, 457,	282, 295, 313,	103, 104, 122,
492, 494, 503,	423, 585, 748, 813	123, 124, 125,
504, 508, 509,	<code>\cb@trace@pop</code>	126, 127, 128,
528, 529, 555,	. . . 132 , 177 , 1215	135, 137, 140, 143
556, 557, 559,	<code>\cb@trace@push</code>	<code>\DeclareRobustCommand</code>
579, 582, 607,	. . . 130 , 177 , 1223	 845
608, 613, 619,	<code>\cb@trace@stack</code> 129 ,	<code>\deletebarwidth</code> . . .
620, 655, 657,	176, 894, 1101,	 4 , 824 , 837
666, 667, 671,	1123, 1164,	<code>\directlua</code> 106
672, 691, 692,	1290, 1311,	<code>\driver</code> 196
718, 719, 720,	1329, 1353, 1370	<code>\DVIPS</code> 202, 212
722, 742, 745,	<code>\cb@vtryfc</code> . 1128 , 1145	<code>\DVItOtoPS</code> 201, 211
770, 771, 775,	<code>\cb@write</code>	
781, 782, 1022,	. . . 1188 , 1233, 1244	E
1024, 1035,	<code>\cb@writeAux</code> 974 , 989 ,	<code>\egroup</code> 219
1036, 1199,	1016 , 1103, 1131	<code>\emTeX</code> 203, 213
1200, 1203,	<code>\cb@writepagejump</code> .	<code>\end@dblfloat</code> 1326 , 1365
1204, 1376,	. . . 1031 , 1071, 1085	<code>\end@float</code>
1377, 1387, 1388	<code>\cb@writexy</code>	. . . 1288 , 1323, 1348
<code>\cb@tempstack</code>	. 69, 70, 72, 75,	<code>\endchangebar</code> 826
. . . 943 , 953 , 1194	89, 90, 92, 95,	<code>\endlinechar</code>
<code>\cb@thehistorystack</code>	109, 110, 112,	393, 555, 718, 1203
. 1196 , 1200	115, 326, 487, 650	environments:
<code>\cb@toleft</code> 7 ,	<code>\cb@xetexcheck</code>	changebar 3
131, 133, 871,	. . . 83, 84, 85, 217	<code>\evensidemargin</code> . . . 37
896, 918, 930,	<code>\cb@xetexerror</code> . . 86, 99	<code>\ExecuteOptions</code> 153 ,
935, 938, 941,	<code>\cb@xynil</code>	156, 158, 162, 165
946, 951, 955,	. 342, 347, 504,	
957, 964, 968,	509, 667, 672, 914	F
971, 974, 977,	<code>\cbcolor</code> 3 , 845 , 862, 868	<code>\float@dblend</code> 1351
978, 979, 991,	<code>\cbdelete</code> 3 , 822	<code>\float@end</code> 1309
1148, 1161,	<code>\cbend</code> 3 , 821	<code>\flt@float@dblend</code> .
1173, 1178,	<code>\cbstart</code> 3 , 818	 1351 , 1366
1179, 1210,	<code>\changebar</code> 826	<code>\flt@float@end</code>
1217, 1222,	changebar (env.) 3	 1309 , 1324
1266, 1292,	changebargrey 4	
1313, 1331,	<code>\changebargrey</code> 841	I
1355, 1372, 1383	<code>\changebarsep</code> . . . 4 ,	<code>\if@cb@enddocument</code> .
<code>\cb@topright</code> . 7 , 964,	32, 34, 39, 42, 839	. 340, 502, 665, 913
969, 972, 977, 993	<code>\changebarwidth</code> . . .	<code>\if@cb@firstcolumn</code> .
<code>\cb@trace</code>	 3 , 33 , 35 ,	. . . 20, 1227, 1268

<code>\if@cb@pagejump</code> ...	<code>\level</code> 1020, 1023	<code>\pdfliteral</code> ... 446,
.. 1027, 1070, 1081	<code>\LN</code> 200, 210	448, 455, 496,
<code>\if@cb@pointsunique</code>	<code>\ltx@xympar</code> 1304, 1307	608, 610, 618,
..... 336,	<code>\ltx@end@dblfloat</code> .	659, 771, 773, 780
498, 661, 911, 917	 1326, 1365	<code>\pdfoutput</code> .. 66, 67,
<code>\if@cb@trace</code>	<code>\ltx@end@float</code> 1288,	152, 155, 182, 184
. 19, 190, 295, 304	1302, 1306, 1323	<code>\pdfsavepos</code>
<code>\if@compatibility</code> . 196	<code>\ltx@footnotetext</code> .	. 65, 180, 416, 578
<code>\if@files</code>	.. 1368, 1373, 1377	<code>\pdfTeX</code> 206, 216
412, 574, 737, 1017	<code>\ltx@makecol</code> 1096, 1112	
<code>\if@firstcolumn</code> 1047,	<code>\ltx@mpfootnotetext</code>	S
1059, 1077,	.. 1380, 1384, 1388	<code>\savepos</code> 741
1099, 1115, 1136	<code>\ltx@vtryfc</code> . 1127, 1133	<code>\sec@end@ftw@</code> 97, 610, 611
<code>\if@twocolumn</code> . 994,	<code>\luaTeX</code> 208, 218	<code>\special</code> .. 237, 240,
1046, 1058,		243, 246, 252,
1076, 1098,	M	262, 269, 281,
1113, 1134, 1328	<code>\marginparsep</code> 840	284, 289, 290,
<code>\if@twoside</code>		297, 308, 315, 496
36, 1001, 1007, 1053	N	
<code>\ifcsname</code> . 339, 344,	<code>\newif</code> 19,	T
501, 506, 664,	20, 911, 913, 1027	<code>\tempa</code> 198, 210, 211,
669, 920, 921, 1228	<code>\nochangebars</code> ... 3, 830	212, 213, 214,
<code>\ifodd</code> 460, 463, 623,		215, 216, 217, 218
626, 785, 788,	O	<code>\Textures</code> 204, 214
967, 982, 995,	<code>\oddsidemargin</code> 29	<code>\thechangebargrey</code> . 136
998, 1004, 1048,	<code>outerbars</code> 4	<code>\thepage</code> 228, 233
1051, 1057, 1179		
<code>\ignorespacesafterend</code>	P	V
..... 833, 910	<code>\PackageError</code>	<code>\VTeX</code> 205, 215
	.. 78, 99, 118, 143	<code>\VTeXversion</code> 150
L	<code>\pdfextension</code> 659	
<code>\lastxpos</code> 744	<code>\pdflastxpos</code> .. 419, 581	X
<code>\lastypos</code> 744	<code>\pdflastypos</code> .. 419, 581	<code>\xeTeX</code> 207, 217