

The Changebar package *

Michael Fine
Distributed Systems Architecture

Johannes Braams
`johannes.braams at texniek.nl`

Printed January 1, 2024

Contents

1	Introduction	1	5	The implementation	6
			5.1	Declarations And Initial- izations	6
2	The user interface	2	5.2	Option Processing	8
2.1	The package options	2	5.3	User Level Commands And Parameters	13
2.1.1	Specifying the printer driver	2	5.4	Macros for beginning and ending bars	31
2.1.2	Specifying the bar position	3	5.5	Macros for Making It Work Across Page Breaks	35
2.1.3	Color	3	5.6	Macros For Managing The Stacks of Bar points	40
2.1.4	Tracing	3	5.7	Macros For Checking That The .aux File Is Stable	41
2.2	Macros defined by the package	3	5.8	Macros For Making It Work With Nested Floats/Footnotes	43
2.3	Changebar parameters . .	4			
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

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

- Changebars may be nested within each other. Each level of nesting can be given a different thickness bar.
- Changebars may be nested in other environments including floats and footnotes.
- Changebars are applied to all the material within the “barred” environment, including floating bodies regardless of where the floats float to. An exception to this is margin floats.
- Changebars may cross page boundaries.
- Changebars can appear on the *outside* of the columns of `twocolumn` text.
- The colour of the changebars can be changed. This has sofar been tested with the `dvips`, `pdftex`, `vtex` 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`
- `DVItolPS`
- `DVIps`
- `emTeX`
- `TeXtures`
- `VTEX`
- `PDFTeX`
- `XeTeX`
- `luaTeX`

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.

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

2.1.2 Specifying the bar position

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

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

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

2.1.3 Color

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

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

2.1.4 Tracing

The package also implements tracing for its own debugging. The package options `tracoon` 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 L^AT_EX environment is defined. The advantage of using the environment whenever possible is that L^AT_EX will do all the work of checking the correct nesting of different environments.

`\cbdelete` The macro `\cbdelete` puts a square bar in the margin to indicate that some text was removed from the document. The macro has an optional argument to specify the width of the bar. When no argument is specified the current value of the parameter `\deletebarwidth` will be used.

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

2.3 Changebar parameters

`\changebarwidth` The width of the changebars is controlled with the \LaTeX length parameter `\changebarwidth`. Its value can be changed with the `\setlength` command. Changing the value of `\changebarwidth` affects all subsequent changebars subject to the scoping rules of `\setlength`.
`\deletebarwidth` The width of the deletebars is controlled with the \LaTeX length parameter `\deletebarwidth`. Its value can be changed with the `\setlength` command. Changing the value of `\deletebarwidth` affects all subsequent deletebars subject to the scoping rules of `\setlength`.
`\changebarsep` The separation between the text and the changebars is determined by the value of the \LaTeX length parameter `\changebarsep`.
`changebargrey` (*env.*) When one of the supported dvi to PostScript translators is used the 'blackness' of the bars can be controlled. The \LaTeX counter `changebargrey` is used for this purpose. Its value can be changed with a command like:

```
\setcounter{changebargrey}{85}
```

The value of the counter is a percentage, where the value 0 yields black bars, the value 100 yields white bars.

`outerbars` (*env.*) The changebars will be printed in the 'inside' margin of your document. This means they appear on the left side of the page. When `twoside` is in effect the bars will be printed on the right side of even pages.

3 Deficiencies and bugs

- The macros blindly use special points `\cb@minpoint` through `\cb@maxpoint`. If this conflicts with another set of macros, the results will be unpredictable. (What is really needed is a `\newspecialpoint`, analogous to `\newcount` etc. — it's not provided because the use of the points is rather rare.)
- There is a limit of $(\text{\code{\cb@maxpoint}} - \text{\code{\cb@minpoint}} + 1) / 4$ 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 $\text{\textsf{PDF}\TeX}$ or $\text{\textsf{Xe}\TeX}$.
- Internal macro names are all of the form `\cb@xxxx`. No checking for conflicts with other macros is done.
- This implementation does not work with the `multicolumn` package.
- The algorithms may fail if a floating insertion is split over multiple pages. In \LaTeX floats are not split but footnotes may be. The simplest fix to this is to prevent footnotes from being split but this may make \TeX very unhappy.

- The `\cbend` normally gets “attached” to the token after it rather than the one before it. This may lead to a longer bar than intended. For example, consider the sequence ‘word1 `\cbend` word2’. If there is a line break between ‘word1’ and ‘word2’ the bar will incorrectly be extended an extra line. This particular case can be fixed with the incantation ‘word1`\cbend{}` word2’.
- The colour support has only been tested with the `dvips` and `pdftex` drivers.

4 The basic algorithm

The changebars are implemented using the `\specials` of various `dvi` interpreting programs like `DVItOLN03` or `DVIps`. In essence, the start of a changebar defines two `\special` points in the margins at the current vertical position on the page. The end of a changebar defines another set of two points and then joins (using the “connect” `\special`) either the two points to the left or the two points to the right of the text, depending on the setting of `innerbars`, `outerbars`, `leftbars`, `rightbars` and/or `twoside`.

This works fine as long as the two points being connected lie on the same page. However, if they don’t, the bar must be artificially terminated at the page break and restarted at the top of the next page. The only way to do this (that I can think of) is to modify the output routine so that it checks if any bar is in progress when it ships out a page and, if so, adds the necessary artificial end and begin.

The obvious way to indicate to the output routine that a bar is in progress is to set a flag when the bar is begun and to unset this flag when the bar is ended. This works most of the time but, because of the asynchronous behavior of the output routine, errors occur if the bar begins or ends near a page break. To illustrate, consider the following scenario.

```

blah blah blah           % page n
blah blah blah
\cbstart                 % this does its thing and set the flag
more blah
                           <----- pagebreak occurs here
more blah
\cbend                   % does its thing and unsets flag
blah blah

```

Since `TEX` processes ahead of the page break before invoking the output routine, it is possible that the `\cbend` is processed, and the flag unset, before the output routine is called. If this happens, special action is required to generate an artificial end and begin to be added to page n and $n + 1$ respectively, as it is not possible to use a flag to signal the output routine that a bar crosses a page break.

The method used by these macros is to create a stack of the beginning and end points of each bar in the document together with the page number corresponding to each point. Then, as a page is completed, a modified output routine checks the stack to determine if any bars begun on or before the current page are terminated on subsequent pages, and handles those bars appropriately. To build the stack, information about each changebar is written to the `.aux` file as bars are processed. This information is re-read when the document is next processed. Thus, to ensure that changebars are correct, the document must be processed twice. Luckily, this

is generally required for L^AT_EX anyway. With PDFL^AT_EX 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 L^AT_EX 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 DVI to L^AT_EX 3 specials. The L^AT_EX 3 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 `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@cmta` Sometimes we need temporarily store a value. For this purpose two count registers `\cb@cmtb` and a dimension register are allocated.

```
\cb@dima 11 \newcount\cb@cmta
12 \newcount\cb@cmtb
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 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
- DVItolPS
- DVIPs
- emT_EX
- 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{DVItolPS}{\global\chardef\cb@driver@setup=1\relax}
53 \DeclareOption{dvitolps}{\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 L^AT_EX run is using PDF_TE_X and that PDF output is selected. If it is, we initialize the option and open an additional output file. If not, we ignore the option and issue a warning.

```

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

```

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

```

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

```

Give a warning if we cannot support the `pdftex` option.

```

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

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

```

For the xetex option we have to check that the current L^AT_EX run is using XeT_EX. 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 L^AT_EX run is using luaT_EX. 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 L^AT_EX 2_ε 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\color\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\color\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 \TeX is used the option `dvips` is not the right one, so in that case we have `vtex` as the default driver. When \PDF\TeX is producing PDF output, the `pdftex` option is selected.

```

150 \ifx\TeXversion\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   \if@cb@trace
191     \GenericWarning
192       {(changebar)\@spaces\@spaces}%
193       {Package changebar: #1\@gobble}%
194   \fi
195 }

```

5.3 User Level Commands And Parameters

`\driver` The user can select the specials that should be used by calling the command `\driver{<drivername>}`. Possible choices are:

- DVIToLN03
- DVIToPS
- DVIPs
- emT_EX
- T_EXtures
- V_TE_X
- 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\DVItO PS \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 $\langle number \rangle$ s indicate the two points to be connected; the $\langle length \rangle$ 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 `DVItO LN03` (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 `TEXtures` version 1.7.7, but will very likely also work with later releases of `TEXtures`.

The `\cbdelete` command seemed to create degenerate lines (i.e., lines of 0 length). PostScript will not render such lines unless the linecap is set to 1, (semi-circular ends) in which case a filled circle is shown for such lines.

```

287 \or
288 \def\cb@defpoint##1##2{%
289   \special{postscript 0 0 transform}% leave [x,y] on the stack
290   \special{rawpostscript
291     \expandafter\cb@removedim\the##2\space
292     /CBarX\the##1\space exch def
293     itransform exch pop
294     /CBarY\the##1\space exch def}%
295   \if@cb@trace\cb@trace@defpoint##1##2\fi}
296 \def\cb@connect##1##2##3{%
297   \special{rawpostscript
298     gsave 1 setlinecap \cb@ps@color\space
299     \expandafter\cb@removedim\the##3\space
300     setlinewidth
301     CBarX\the##1\space\space CBarY\the##1\space\space moveto
302     CBarX\the##2\space\space CBarY\the##2\space\space lineto
303     stroke grestore}%
304   \if@cb@trace\cb@trace@connect##1##2##3\fi}
305 \let\cb@resetpoints\relax

```

The following definitions were kindly provided by Michael Vulis.

```

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

```



```

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

```

325 \or
326 \immediate\closeout\cb@writexy
327 \immediate\openin\cb@readxy=\jobname.cb2\relax

```

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

```

328 \def\cb@pdfpoints{}
329 \def\cb@pdfpagenr{0}

```

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

```

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

```

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

```

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

```

```

346     \fi
347 }

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

407 \def\cb@pdf@scale{0.996264009963}

```

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

```

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

```

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

```

416     \begingroup
417     \cb@dima=\cb@pdfz
418     \advance\cb@dima by-\cb@pdfx
419     \advance\cb@dima by1in%
420     \cb@dima=\cb@pdf@scale\cb@dima\relax

```

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

```

421         \ifx\cb@current@color\undefined
422         \def\cb@temp{\cb@cvtpt\cb@changebargrey}%
423         \pdfliteral{q \cb@temp\space g \cb@temp\space G}%
424         \else
425         \pdfliteral{q \cb@current@color}%
426         \fi
427         \edef\cb@temp{\expandafter\cb@removedim\the\cb@dima\space}%
428         \cb@dima=\cb@pdfx
429         \advance\cb@dima-\cb@pdfy\relax
430         \cb@dima=\cb@pdfscale\cb@dima\relax
431         ##3=\cb@pdfscale##3\relax
432         \pdfliteral direct{\expandafter\cb@removedim\the##3 w
433         \cb@temp 0 m
434         \cb@temp \expandafter\cb@removedim\the\cb@dima\space 1 S Q}%
435     \endgroup

```

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

```

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

```

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

```

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

```

466 }

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

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

The code for 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.

```
470 \immediate\closeout\cb@writexy
471 \immediate\openin\cb@readxy=\jobname.cb2\relax
```

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

```
472 \def\cb@pdfpoints{}
473 \def\cb@pdfpagenr{0}
```

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

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

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

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

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

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

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

$\langle pointnr \rangle . \langle pagecount \rangle p \langle x \rangle , \langle y \rangle , \langle z \rangle pt$. When the point is found it is removed from `\cb@pdfpoints`. #9 contains the cumulative head of the list to construct the new list with the entry removed. #3–#8 are for pattern matching.

```

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

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

```

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

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

```

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

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

```

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

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

```

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

`\cb@pdf@scale` In order to get things in the right spot we need a little scaling factor. We define it here.

```

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

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

```

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

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

```

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

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

```

567         \ifx\cb@current@color\undefined
568         \def\cb@temp{\cb@cvtpt\cb@changebargrey}%
569         \pdfliteral{q \cb@temp\space g \cb@temp\space G}%
570     \else
571         \pdfliteral{q \expandafter\sec@nd@ftw@\cb@current@color\space RG
572                     \expandafter\sec@nd@ftw@\cb@current@color\space rg}%
573     \fi
574     \edef\cb@temp{\expandafter\cb@removedim\the\cb@dima\space}%
575     \cb@dima=\cb@pdftopy
576     \advance\cb@dima-\cb@pdfy\relax
577     \cb@dima=\cb@pdf@scale\cb@dima\relax
578     ##3=\cb@pdf@scale##3\relax
579     \pdfliteral{\expandafter\cb@removedim\the##3 w
580               \cb@temp 0 m
581               \cb@temp \expandafter\cb@removedim\the\cb@dima\space 1 S Q}%
582 \endgroup

```

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

```

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

```

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

```

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

```

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

```

608 \def\cb@maxpoint{9999999}
609 \let\cb@resetpoints\relax

```

The code for lua_TE_X, like for pdf_TE_X and Xe_TE_X, 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.


```

610 \or
611 \immediate\closeout\cb@writexy
612 \immediate\openin\cb@readxy=\jobname.cb2\relax

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

\cb@findpdfpoint The \cb@findpdfpoint macro finds the coordinates of point #1 on pagecount
#2. First we expand the arguments to get the real values.
615 \def\cb@findpdfpoint##1##2{%
616     \edef\cb@temp
617     {\noexpand\cb@@findpdfpoint{\the##1}{\the##2}}%
618     \cb@temp
619 }

\pdfliteral For luaTeX we also mimick PDFTeX's command \pdfliteral.
620 \def\pdfliteral##1{\pdfextension literal {##1}}

\cb@@findpdfpoint The \cb@@findpdfpoint macro finds the coordinates of point #1 on pagecount
#2. If the information is not yet in memory is it read from the .cb2 file. The
coordinates of the current point in the text will be delivered in \cb@pdfx and
\cb@pdfy, and \cb@pdfz will get the x coordinate of the changebar. If the point
is unknown, \cb@pdfx will be set to \relax.
621 \def\cb@@findpdfpoint##1##2{%
622     \ifnum##2<\cb@pdfpagenr\relax\else
623         \cb@pdfreadxy{##2}%
624     \fi
625     \let\cb@pdfx\relax
626     \ifx\cb@pdfpoints\@empty\else
627         \ifnum##2<0\relax
628         \else
629             \edef\cb@temp{%
630                 \noexpand\cb@pdffind{##1}{##2}\cb@pdfpoints\relax{}}%
631             \cb@temp
632         \fi
633     \fi
634 }

\cb@pdffind The \cb@pdffind recursively searches through \cb@pdfpoints to find point #1
on pagecount #2. \cb@pdfpoints contains entries of the form
<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.
635 \def\cb@pdffind##1##2##3.##4p##5;##6;##7pt##8\relax##9{%
636     \def\cb@next{%
637         \cb@pdffind{##1}{##2}##8\relax{##9##3.##4p##5;##6;##7pt}}%
638     \ifnum ##1=##3
639         \ifnum ##2=##4
640             \def\cb@pdfx{##5sp}%

```

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

```

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

```

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

```

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

```

`\cb@pdf@scale` In order to get things in the right spot we need a little scaling factor. We define it here.

```

694 \def\cb@pdf@scale{0.996264009963}

```

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

```

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

```

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

```

703     \begingroup
704     \cb@dima=\cb@pdfz
705     \advance\cb@dima by-\cb@pdfx
706     \advance\cb@dima by1in%
707     \cb@dima=\cb@pdf@scale\cb@dima\relax

```

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

```

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

```

```

717         \cb@dima=\cb@pdf@scale\cb@dima\relax
718         ##3=\cb@pdf@scale##3\relax
719         \pdfliteral{\expandafter\cb@removedim\the##3 w
720         \cb@temp 0 m
721         \cb@temp \expandafter\cb@removedim\the\cb@dima\space 1 S Q}%
722     \endgroup

```

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

```

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

```

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

```

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

```

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

```

748 \def\cb@maxpoint{9999999}
749 \let\cb@resetpoints\relax

```

When code for other drivers should be added it can be inserted here. When someone makes a mistake and somehow selects an unknown driver a warning is issued and the macros are defined to be no-ops.

```

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

```

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

```

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

```

\cbstart The macro `\cbstart` starts a new changebar. It has an (optional) argument that will be used to determine the width of the bar. The default width is `\changebarwidth`.

```

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

\cbend The macro `\cbend` (surprisingly) ends a changebar. The macros `\cbstart` and `\cbend` can be used when the use of a proper \LaTeX environment is not possible.

```

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

\cbdelete The macro `\cbdelete` inserts a ‘deletebar’ in the margin. It too has an optional argument to determine the width of the bar. The default width (and length) of it are stored in `\deletebarwidth`.

```

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

\cb@delete Deletebars are implemented as a special ‘change bar’. The bar is started and immediately ended. It is as long as it is wide.

```

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

\changebar The macros `\changebar` and `\endchangebar` have the same function as `\cbstart` and `\cbend` but they can be used as a \LaTeX environment to enforce correct nesting. They can *not* be used in the `tabular` and `tabbing` environments.

```

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

\nochangebars To disable changebars altogether without having to remove them from the document the macro `\nochangebars` is provided. It makes no-ops of three internal macros.

```

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

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

```

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

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

```

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

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

```

777 \newlength{\changebarsep}
778 \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).

```
779 \newcounter{changebargrey}
780 \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 pdf^AT_EX we need to pass that information on to that package.

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

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

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

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

`\@declaredcbcolor`

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

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

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

5.4 Macros for beginning and ending bars

\cb@start This macro starts a change bar. It assigns a new value to the current point and advances the counter for the next point to be assigned. It pushes this info onto **\cb@currentstack** and then sets the point by calling **\cb@setBeginPoints** with the point number. Finally, it writes the **.aux** file.

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

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

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

```
812   \cb@checkpage\z@
```

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

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

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

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

Restore **\cb@pagecount**.

```
823   \cb@pagecount\cb@cmta
824   \cb@advancePoint\ignorespaces}
```

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

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

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

```

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

```

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

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

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

```

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

```

`\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).

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

First store the identifiers in temporary registers.

```

849 \cb@cmta\cb@topleft\relax
850 \advance\cb@cmta by #1\relax
851 \cb@cntb\cb@page\relax
852 \cb@dima\cb@curbarwd\relax

```

Then pop the history stack.

```
853 \cb@pop\cb@historystack
```

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

```

854 \ifnum\cb@topleft=\cb@nil
855 \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.

```

856 \cb@FindPageNum
857 \ifnum\cb@topleft>\cb@maxpoint\else

```

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

```

858 \cb@cntb\cb@page
859 \fi

```

Now we restore the history stack to its original state.

```

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

```


Finally return the correct values

```

864 \advance\cb@cmta by -#1\relax
865 \cb@topleft\cb@cmta\relax
866 \cb@page\cb@cntb\relax
867 \cb@curbarwd\cb@dima\relax
868 }

```

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

```

869 \def\cb@FindPageNum{%
870 \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.

```

871 \else
872 \cb@push\cb@tempstack
873 \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.

```

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

```

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

```

876 \cb@topleft\cb@maxpoint\advance\cb@topleft\@ne
877 \else

```

Recursively call ourselves.

```

878 \expandafter\expandafter\expandafter\cb@FindPageNum
879 \fi
880 \fi
881 }%

```

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

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

```

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

```

`\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.

```

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

In twocolumn pages always use outerbars

```

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

0=innerbars

```

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

1=outerbars

```

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

2=leftbars

```

929     \cb@leftbar
930   \or
```

3=rightbars

```

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

`\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.

```

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

```

5.5 Macros for Making It Work Across Page Breaks

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

```

946 \newif\if@cb@pagejump

```

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

```

947 \def\cb@pagejumplist{-1}

```

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

```

948 \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.

```

949 \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.

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

```

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

```

`\@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 `\@cclv`. Then the stack `\cb@endstack` is built and processed by `\cb@processActive`. Finally the original `\@makecol` (saved as `\cb@makecol`) is executed.

```

1015 \let\ltx@makecol\@makecol
1016 \def\cb@makecol{%
1017   \if@twocolumn
1018     \cb@trace{Twocolumn: \if@firstcolumn Left \else Right \fi column}%
1019   \fi
1020   \cb@trace@stack{before makecol, page \the\c@page,
1021     \string\cb@pagecount=\the\cb@pagecount}%
1022   \let\cb@writeAux\@gobble

```

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

```

1023   \cb@checkpagecount
1024   \setbox\@cclv \vbox{%
1025     \cb@resetpoints
1026     \cb@startSpanBars
1027     \unvbox\@cclv
1028     \boxmaxdepth\maxdepth}%
1029   \global\advance\cb@pagecount by\@ne
1030   \cb@buildstack\cb@processActive

```

```
1031 \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.

```
1032 \if@twocolumn
1033   \global\setbox\@outputbox \vbox to\@colht {%
1034     \if@firstcolumn\write\@auxout{\string\cb@firstcolumntrue}%
1035     \else\write\@auxout{\string\cb@firstcolumnfalse}%
1036     \fi
1037     \dimen@ \dp\@outputbox
1038     \unvbox \@outputbox
1039     \vskip -\dimen@
1040   }%
1041 \fi
1042 \cb@trace@stack{after makecol, page \the\c@page,
1043               \string\cb@pagecount=\the\cb@pagecount}%
1044 }
1045 \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.

```
1046 \let\ltx@vtryfc\@vtryfc
1047 \def\cb@vtryfc#1{%
1048   \cb@trace{In vtryfc, page \the\c@page,
1049             \string\cb@pagecount=\the\cb@pagecount}%
1050   \let\cb@writeAux\@gobble
```

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

```
1051 \cb@checkpagecount
1052 \ltx@vtryfc{#1}%
1053 \if@twocolumn
1054   \global\setbox\@outputbox \vbox to\@colht{%
1055     \if@firstcolumn\write\@auxout{\string\cb@firstcolumntrue}%
1056     \else\write\@auxout{\string\cb@firstcolumnfalse}%
1057     \fi
1058     \unvbox\@outputbox
1059     \boxmaxdepth\maxdepth
1060   }%
1061 \fi
1062 \global\advance\cb@pagecount by \@ne
1063 }
1064 \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).

```

1065 \def\cb@processActive{%
1066   \cb@pop\cb@endstack
1067   \ifnum\cb@topleft=\cb@nil
1068   \else
1069     \setbox\@cclv\vbox{%
1070       \unvbox\@cclv
1071       \boxmaxdepth\maxdepth
1072       \advance\cb@pagecount by -1\relax
1073       \cb@setEndPoints}%
1074     \cb@push\cb@historystack
1075     \cb@push\cb@beginstack
1076     \expandafter\cb@processActive
1077   \fi}

```

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

```

1078 \def\cb@startSpanBars{%
1079   \cb@pop\cb@beginstack
1080   \ifnum\cb@topleft=\cb@nil
1081   \else
1082     \cb@setBeginPoints
1083     \cb@trace@stack{after StartSpanBars, page \the\c@page}%
1084     \expandafter\cb@startSpanBars
1085   \fi
1086 }

```

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

```

1087 \def\cb@buildstack{%
1088   \cb@initstack\cb@endstack
1089   \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.

```

1090 \def\cb@pushNextActive{%
1091   \cb@pop\cb@historystack
1092   \ifnum\cb@topleft=\cb@nil
1093   \else
1094     \ifnum\cb@page>\cb@pagecount
1095       \cb@push\cb@historystack
1096     \else
1097       \ifodd\cb@topleft
1098         \cb@push\cb@endstack
1099       \else

```

```

1100      \cb@pop\cb@endstack
1101      \fi
1102      \expandafter\expandafter\expandafter\cb@pushNextActive
1103      \fi
1104  \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 `xxxnyyyppzzz` 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.
1105 \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 1106 \cb@initstack\cb@historystack

We also need to allocate a read and a write stream for the history file.
1107 \newwrite\cb@write
1108 \newread\cb@read

And we open the history file for writing (which is done when the .aux file is read
in).
1109 \immediate\openout\cb@write=\jobname.cb\relax

\cb@endstack Allocate two stacks for the bars that span the current page break.
\cb@beginstack 1110 \cb@initstack\cb@endstack
1111 \cb@initstack\cb@beginstack

\cb@tempstack Allocate a stack for temporary storage
1112 \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.

```
1113 \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`.

```
1114 \def\cb@thehistorystack{\cb@historystack}
1115 \def\cb@pop#1{%
1116   \ifx #1\@empty
1117     \def\cb@temp{#1}%
1118     \ifx\cb@temp\cb@thehistorystack
1119       \ifeof\cb@read
1120       \else
1121         {\endlinechar=-1\read\cb@read to\cb@temp
1122          \xdef\cb@historystack{\cb@historystack\cb@temp}%
1123         }%
1124       \fi
1125     \fi
1126   \fi
1127   \ifx#1\@empty
1128     \global\cb@topleft\cb@nil
1129     \global\cb@page\z@\relax
1130   \else
1131     \expandafter\cb@carcdr#1e#1%
1132   \fi
1133   \cb@trace@pop{#1}}
```

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

```
1134 \def\cb@carcdr#1n#2p#3l#4e#5{%
1135   \global\cb@topleft#1\relax
1136   \global\cb@page#2\relax
1137   \global\cb@curbarwd#3\relax
1138   \xdef#5{#4}}
```

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

```
1139 \def\cb@push#1{%
1140   \xdef#1{\the\cb@topleft n\the\cb@page p\the\cb@curbarwd l#1}%
1141   \cb@trace@push{#1}}
1142
```

`\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.

```
1143 \def\cb@barpoint#1#2#3{\cb@canta=#2
1144   \if@cb@firstcolumn\advance\cb@canta by\m@ne\fi
1145   \immediate\write\cb@write{#1n\the\cb@canta p#3l}}
```

5.7 Macros For Checking That The .aux File Is Stable

`\AtBeginDocument` While reading the `.aux` file, \LaTeX has created the history stack in a separate file. We need to close that file and open it for reading. Also the ‘initialisation’ of the

`\special` commands has to take place. While we are modifying the macro we also include the computation of the possible positions of the changebars

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

```

1146 \AtBeginDocument{%
1147   \cb@setup@specials
Add a sentinel to \cb@pagejumpst.
1148   \cb@pagejump{999999999,}%
Compute the left and right positions of the changebars.
1149   \cb@positions
1150   \cb@trace{%
1151     Odd left : \the\cb@odd@left\space
1152     Odd right : \the\cb@odd@right\MessageBreak
1153     Even left: \the\cb@even@left\space
1154     Even right: \the\cb@even@right
1155   }%
1156   \immediate\closeout\cb@write
1157   \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.

```

1158 \AtEndDocument{%
1159   \clearpage
1160   \cb@initstack\cb@historystack
1161   \immediate\closein\cb@read
1162   \immediate\openin\cb@read=\jobname.cb\relax
Let \cb@pdfxy=\cb@checkPdfxy for checking. Make \cb@pagejump dummy.
1163   \ifx\cb@readxy\@undefined
1164   \else
1165     \immediate\closein\cb@readxy
1166     \immediate\openin\cb@readxy=\jobname.cb2\relax
1167     \def\cb@pdfpoints{}%
1168     \def\cb@pdfpagenr{0}%
1169   \fi
1170   \@cb@firstcolumnfalse
1171   \cb@checkrerun
1172   \let\cb@pdfxy\cb@checkPdfxy
1173   \let\cb@pagejump\@gobble
1174   \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.

```

1175 \def\cb@checkHistory#1#2#3{%
1176   \cb@pop\cb@historystack
1177   \ifnum #1=\cb@topleft\relax
1178     \cb@cmta=#2
1179     \ifcb@firstcolumn\advance\cb@cmta by\m@ne\fi
1180     \ifnum \cb@cmta=\cb@page\relax

```

Both page and point numbers are equal; do nothing,

```

1181     \else
but generate a warning when page numbers don't match, or
1182     \cb@error
1183     \fi
1184     \else
when point numbers don't match.
1185     \cb@error
1186     \fi}

```

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

```

1187 \def\cb@checkPdfxy#1#2#3#4#5{}

```

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

```

1188 \def\cb@rerun{%
1189   \global\let\cb@checkrerun\cb@error}
1190 \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`.

```

1191 \def\cb@error{%
1192   \PackageWarning{Changebar}%
1193     {Changebar info has changed.\MessageBreak
1194     Rerun to get the bars right}
1195   \gdef\cb@checkHistory##1##2##3{}%
1196   \let\cb@barpoint\cb@checkHistory
1197   \gdef\cb@checkPdfxy##1##2##3##4##5{}%
1198   \let\cb@pdfxy\cb@checkPdfxy}

```

5.8 Macros For Making It Work With Nested Floats/Footnotes

\end@float This is a replacement for the \LaTeX -macro of the same name. All it does is check to see if changebars are active and, if so, it puts changebars around the box containing the float. Then it calls the original \LaTeX `\end@float`.

```

1199 \let\ltx@end@float\end@float
1200 \def\cb@end@float{%
1201   \cb@trace@stack{end float on page \the\c@page}%
1202   \cb@pop\cb@currentstack
1203   \ifnum\cb@topleft=\cb@nil
1204   \else
1205     \cb@push\cb@currentstack
1206     \global\cb@curbarwd=\cb@curbarwd
1207     \@endfloatbox
1208     \global\setbox\@currbox
1209       \color@vbox
1210       \normalcolor
1211       \vbox\bgroup\cb@start[\cb@curbarwd]\unvbox\@currbox\cb@end
1212   \fi
1213   \ltx@end@float}
1214 \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`.

```
1215 \let\ltx@@xympar\@xympar
1216 \def\@xympar{%
1217   \let\end@float\ltx@end@float
1218   \ltx@@xympar
1219   \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`.

```
1220 \let\flt@float@end\float@end
Then we redefine it to insert the changebarcode.
1221 \def\float@end{%
1222   \cb@trace@stack{end float on page \the\c@page}%
1223   \cb@pop\cb@currentstack
1224   \ifnum\cb@topleft=\cb@nil
1225   \else
1226     \cb@push\cb@currentstack
1227     \global\cb@curbarwd\cb@curbarwd
1228     \@endfloatbox
1229     \global\setbox\@currbox
1230       \color@vbox
1231       \normalcolor
1232       \vbox\bgroup\cb@start[\cb@curbarwd]\unvbox\@currbox\cb@end
1233   \fi
1234   \let\end@float\ltx@end@float
1235   \flt@float@end
1236 }
```

`\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.

```
1237 \let\ltx@end@dblfloat\end@dblfloat
1238 \def\cb@end@dblfloat{%
1239   \if@twocolumn
1240     \cb@trace@stack{end dblfloat on page \the\c@page}%
1241     \cb@pop\cb@currentstack
1242     \ifnum\cb@topleft=\cb@nil
```

```

1243 \else
1244 \cb@push\cb@currentstack
1245 \global\cb@curbarwd=\cb@curbarwd
1246 \@endfloatbox
1247 \global\setbox\@currbox
1248 \color@vbox
1249 \normalcolor
1250 \vbox\bgroup\cb@start[\cb@curbarwd]\unvbox\@currbox\cb@end
1251 \fi
1252 \@endfloatbox
1253 \ifnum\@floatpenalty <\z@
1254 \@largefloatcheck
1255 \@cons\@dbldeferlist\@currbox
1256 \fi
1257 \ifnum \@floatpenalty ==-\@Mii \@Esphack\fi
1258 \else
1259 \end@float
1260 \fi}
1261 \let\end@dblfloat\cb@end@dblfloat

```

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

```

1262 \let\flt@float@dblend\float@dblend
1263 \def\float@dblend{%
1264 \cb@trace@stack{end dbl float on page \the\c@page}%
1265 \cb@pop\cb@currentstack
1266 \ifnum\cb@topleft=\cb@nil
1267 \else
1268 \cb@push\cb@currentstack
1269 \global\cb@curbarwd=\cb@curbarwd
1270 \@endfloatbox
1271 \global\setbox\@currbox
1272 \color@vbox
1273 \normalcolor
1274 \vbox\bgroup\cb@start[\cb@curbarwd]\unvbox\@currbox\cb@end
1275 \fi
1276 \let\end@dblfloat\ltx@end@dblfloat
1277 \flt@float@dblend
1278 }

```

`\@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.

```

1279 \let\ltx@footnotetext\@footnotetext
1280 \long\def\cb@footnotetext#1{%
1281 \cb@trace@stack{end footnote on page \the\c@page}%
1282 \cb@pop\cb@currentstack
1283 \ifnum\cb@topleft=\cb@nil
1284 \ltx@footnotetext{#1}%
1285 \else
1286 \cb@push\cb@currentstack
1287 \edef\cb@temp{\the\cb@curbarwd}%
1288 \ltx@footnotetext{\cb@start[\cb@temp]#1\cb@end}%

```

```

1289 \fi}
1290 \let\@footnotetext\cb@footnotetext

```

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

```

1291 \let\ltx@mpfootnotetext\@mpfootnotetext
1292 \long\def\cb@mpfootnotetext#1{%
1293   \cb@pop\cb@currentstack
1294   \ifnum\cb@toleft=\cb@nil
1295     \ltx@mpfootnotetext{#1}%
1296   \else
1297     \cb@push\cb@currentstack
1298     \edef\cb@temp{\the\cb@curbarwd}%
1299     \ltx@mpfootnotetext{\cb@start[\cb@temp]#1\cb@end}%
1300 \fi}
1301 \let\@mpfootnotetext\cb@mpfootnotetext
1302 </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> ... 801, 802	<code>\@endfloatbox</code>	A
<code>\@auxout</code> 394, 540, 681,	... 1207, 1228,	<code>\AtBeginDocument</code> . 1146
941, 954, 1034,	1246, 1252, 1270	<code>\AtEndDocument</code> ... 1158
1035, 1055, 1056	<code>\@floatpenalty</code>	B
<code>\@cb@firstcolumn</code> .. 20	... 1253, 1257	<code>\begin</code> 449, 450, 455
<code>\@cb@firstcolumnfalse</code>	<code>\@footnotetext</code> ... 1279	<code>\boxmaxdepth</code>
.. 1035, 1056, 1170	<code>\@ifnextchar</code> .. 757,	.. 1028, 1059, 1071
<code>\@cb@firstcolumntrue</code>	761, 766, 784, 800	C
... 1034, 1055	<code>\@ifpackagewith</code> ... 781	<code>\c@changebargrey</code> ..
<code>\@cb@pagejump</code> 946	<code>\@largefloatcheck</code> 1254	... 422, 568, 709
<code>\@cb@pagejumpfalse</code> .	<code>\@leftcolumn</code>	<code>\c@lor@to@ps</code> .. 138, 141
... 986, 998	... 953, 956, 957	<code>\c@page</code> 831,
<code>\@cb@pagejumptrue</code> . 984	<code>\@makecol</code> 1015	975, 1020, 1042,
<code>\@cb@trace</code> 19	<code>\@mpfootnotetext</code> . 1291	1048, 1083,
<code>\@cb@tracefalse</code> ... 127	<code>\@outputbox</code>	1201, 1222,
<code>\@cb@tracetrue</code> ... 126	... 1033, 1037,	1240, 1264, 1281
<code>\@cbcolor</code> 801, 803	1038, 1054, 1058	<code>\cb@findpdfpoint</code> .
<code>\@cclv</code> 1024,	<code>\@preamblecmds</code> 221	... 332, 335,
1027, 1069, 1070	<code>\@undeclaredcbcolor</code>	447, 476, 480,
<code>\@colht</code> . 953, 1033, 1054	... 785, 786	594, 617, 621, 734
<code>\@currbox</code> 1208, 1211,	<code>\@undefined</code> .. 65, 66,	<code>\cb@show@stack</code> 129, 168
1229, 1232,	69, 86, 89, 106,	<code>\cb@advancePoint</code> ..
1247, 1250,	109, 150, 151,	... 824, 825
1255, 1271, 1274	152, 180, 182,	<code>\cb@barpoint</code> .. 942,
<code>\@dbldeferlist</code> ... 1255	421, 567, 708, 1163	1143, 1174, 1196
<code>\@declaredcbcolor</code> .	<code>\@vtryfc</code> 1015	<code>\cb@barsplace</code> 18, 122,
... 785, 793	<code>\@xympar</code> 1215	123, 124, 125, 916

\cb@beginstack 173,	567, 571, 572,	\cb@even@right . 22,
1075, 1079, 1110	708, 712, 789, 796	38, 41, 42, 43,
\cb@botleft 7,	\cb@currentstack . .	46, 888, 903, 1154
897, 902, 905, 910 171, 811,	\cb@FindPageNum 856, 869
\cb@botright . 7, 898,	832, 1113, 1202,	\cb@findpdfpoint . .
903, 906, 908, 912	1205, 1223, 330, 409,
\cb@buildstack	1226, 1241,	413, 438, 441,
. 1030, 1087	1244, 1265,	474, 555, 559,
\cb@carcdr . 1131, 1134	1268, 1282,	585, 588, 615,
\cb@checkHistory . .	1286, 1293, 1297	696, 700, 725, 728
. 1174,	\cb@cvtptct 402, 422,	\cb@footnotetext . .
1175, 1195, 1196	548, 568, 689, 709 1280, 1290
\cb@checkpage	\cb@defpoint 234, 887,	\cb@historystack . .
. 812, 837, 848	888, 890, 891, 174, 853,
\cb@checkpagecount .	902, 903, 905, 906	861, 873, 1074,
. . 964, 1023, 1051	\cb@delete	1091, 1095,
\cb@checkPdfxy 446,	. 762, 763, 764, 771	1106, 1114,
593, 733, 1172,	\cb@dima . . . 11, 417,	1122, 1160, 1176
1187, 1197, 1198	418, 419, 420,	\cb@initstack . 1088,
\cb@checkrerun	427, 428, 429,	1105, 1106,
. . 1171, 1189, 1190	430, 434, 563,	1110, 1111,
\cb@cmta . . . 11, 813,	564, 565, 566,	1112, 1113, 1160
823, 838, 845,	574, 575, 576,	\cb@leftbar 909, 914,
849, 850, 864,	577, 581, 704,	920, 924, 926, 929
865, 870, 1143,	705, 706, 707,	\cb@luatexcheck . . .
1144, 1145,	714, 715, 716,	. 103, 104, 105, 218
1178, 1179, 1180	717, 721, 852, 867	\cb@luatexerror 106, 118
\cb@cmtb 11, 436, 437,	\cb@driver@setup . .	\cb@makecol . 1016, 1045
438, 439, 440, 50, 51,	\cb@maxpoint
441, 583, 584,	52, 53, 54, 55,	1, 243, 368, 467,
585, 586, 587,	56, 57, 58, 59,	514, 608, 655,
588, 723, 724,	60, 61, 68, 88,	748, 827, 857, 876
725, 726, 727,	108, 185, 209,	\cb@minpoint
728, 851, 858,	210, 211, 212, 3, 6, 243, 828
866, 884, 885,	213, 214, 215, 235	\cb@mpfootnotetext .
886, 899, 900,	\cb@end 760, 764, 768, 1292, 1301
901, 917, 923,	772, 830, 1211,	\cb@next . . 349, 356,
951, 952, 954,	1232, 1250,	361, 363, 366,
973, 974, 975, 976	1274, 1288, 1299	379, 384, 495,
\cb@color@pkg	\cb@end@dblfloat . .	502, 507, 509,
. 139, 142, 782 1238, 1261	512, 525, 530,
\cb@colwarn 802, 803, 804	\cb@end@float	636, 643, 648,
\cb@connect 234, 910, 912	. . 1200, 1214, 1219	650, 653, 666, 671
\cb@curbarwd 14, 810,	\cb@endstack	\cb@nextpagejump . .
852, 867, 910,	172, 1066, 1087,	. 948, 962, 979, 981
912, 939, 1137,	1098, 1100, 1110	\cb@nextpoint . . . 5,
1140, 1206,	\cb@error 458,	809, 826, 827, 828
1211, 1227,	461, 464, 599,	\cb@nil . 4, 833, 854,
1232, 1245,	602, 605, 739,	860, 874, 1067,
1250, 1269,	742, 745, 1182,	1080, 1092,
1274, 1287, 1298	1185, 1189, 1191	1128, 1203,
\cb@current@color .	\cb@even@left 22, 28,	1224, 1242,
. 138,	37, 38, 39, 40,	1266, 1283, 1294
141, 421, 425,	45, 887, 902, 1153	

<code>\cb@odd@left</code>	357, 374, 375,	1157, 1161, 1162
. . . <u>22</u> , 27, 28,	<u>472</u> , 485, 489,	<code>\cb@readxy</code> 71, 91, 111,
29, 30, 34, 35,	503, 520, 521,	327, 367, 370,
45, 890, 905, 1151	<u>613</u> , 626, 630,	471, 513, 516,
<code>\cb@odd@right</code> . . <u>22</u> ,	644, 661, 662, 1167	612, 654, 657,
30, 31, 32, 33,	<code>\cb@pdfreadxy</code>	1163, 1165, 1166
46, 891, 906, 1152 <u>337</u> , <u>365</u> ,	<code>\cb@removedim</code> <u>49</u> , 247,
<code>\cb@page</code> . . . <u>15</u> , 814,	482, <u>511</u> , 623, <u>652</u>	254, 263, 271,
839, 851, 858,	<code>\cb@pdftexcheck</code> . . .	291, 299, 309,
866, 1094, 1129,	. . . 62, 63, 64, 216	317, 427, 432,
1136, 1140, 1180	<code>\cb@pdftexerror</code> . . .	434, 574, 579,
<code>\cb@pagecount</code> <u>15</u> , 228, 65, 66, 77, 78	581, 714, 719, 721
233, 391, 409,	<code>\cb@pdftopy</code> 412, 428,	<code>\cb@rerun</code> 410,
413, 438, 441,	558, 575, 699, 715	414, 556, 560,
537, 555, 559,	<code>\cb@pdfx</code> . . 339, 353,	697, 701, 1006, <u>1188</u>
585, 588, 678,	410, 414, 418,	<code>\cb@resetpoints</code> . . .
696, 700, 725,	451, 484, 499, <u>234</u> , 1025
728, 813, 814,	556, 560, 564,	<code>\cb@rightbar</code> 911, 914,
823, 838, 839,	595, 625, 640,	918, 920, 926, 931
845, 884, 899,	697, 701, 705, 735	<code>\cb@setBeginPoints</code> .
914, 942, 951,	<code>\cb@pdfxy</code> <u>21</u> ,	816, 820, <u>882</u> , 1082
967, 970, 973,	74, 94, 114, 395,	<code>\cb@setEndPoints</code> . .
979, 981, 982,	541, 682, 1172, 1198	841, 843, <u>895</u> , 1073
983, 997, 1002,	<code>\cb@pdfy</code> . . 354, 412,	<code>\cb@setup@specials</code> .
1005, 1010,	429, 452, 500, <u>224</u> , 1147
1021, 1029,	558, 576, 596,	<code>\cb@start</code> . 758, 759,
1043, 1049,	641, 699, 716, 736	764, 766, 767,
1062, 1072, 1094	<code>\cb@pdfz</code> . . 355, 417,	770, <u>808</u> , 1211,
<code>\cb@pagejump</code>	456, 501, 563,	1232, 1250,
<u>949</u> , 954, 1148, 1173	597, 642, 704, 737	1274, 1288, 1299
<code>\cb@pagejumplist</code> . . <u>947</u>	<code>\cb@pop</code> . . . 832, 853,	<code>\cb@startSpanBars</code> .
<code>\cb@pagejumplst</code> . . .	862, 873, 1066, <u>1026</u> , <u>1078</u>
. 947, 949, 963, 980	1079, 1091,	<code>\cb@temp</code> . . 331, 333,
<code>\cb@pdf@scale</code>	1100, <u>1114</u> ,	343, 344, 370,
. . . . <u>407</u> , 420,	1176, 1202,	371, 372, 374,
430, 431, <u>553</u> ,	1223, 1241,	394, 397, 422,
566, 577, 578,	1265, 1282, 1293	423, 427, 433,
<u>694</u> , 707, 717, 718	<code>\cb@poppagejump</code> <u>961</u> , 980	434, 475, 477,
<code>\cb@pdffind</code> 343, <u>348</u> ,	<code>\cb@positions</code> . <u>22</u> , 1149	488, 490, 516,
489, <u>494</u> , 630, <u>635</u>	<code>\cb@processActive</code> .	517, 518, 520,
<code>\cb@pdfpagenr</code> <u>1030</u> , <u>1065</u>	540, 543, 568,
. 328, 336, 368,	<code>\cb@ps@color</code>	569, 574, 580,
376, <u>472</u> , 481,	. 136, 138, 141,	581, 616, 618,
514, 522, <u>613</u> ,	253, 270, 298, 316	629, 631, 657,
622, 655, 663, 1168	<code>\cb@push</code> . . 811, 861,	658, 659, 661,
<code>\cb@pdfparsexy</code>	872, 1074, 1075,	681, 684, 709,
. . . . 372, <u>386</u> ,	1095, 1098,	710, 714, 720,
518, <u>532</u> , 659, <u>673</u>	<u>1139</u> , 1205,	721, 941, 943,
<code>\cb@pdfpg</code> . 373, 376,	1226, 1244,	954, 955, 1117,
378, 387, 519,	1268, 1286, 1297	1118, 1121,
522, 524, 533,	<code>\cb@pushNextActive</code> .	1122, 1287,
660, 663, 665, 674 <u>1089</u> , <u>1090</u>	1288, 1298, 1299
<code>\cb@pdfpoints</code>	<code>\cb@read</code> <u>1106</u> ,	<code>\cb@tempstack</code>
. <u>328</u> , 340, 343,	1119, 1121,	. . . 862, 872, <u>1112</u>

<code>\cb@thehistorystack</code>	115, 326, 470, 611	<code>\evensidemargin</code> ... 37
..... 1114, 1118	<code>\cb@xetexcheck</code>	<code>\ExecuteOptions</code> 153,
<code>\cb@topleft</code> 7,	... 83, 84, 85, 217	156, 158, 162, 165
131, 133, 809,	<code>\cb@xetexerror</code> .. 86, 99	
833, 849, 854,	<code>\cbcolor</code> 4, 783, 800, 806	F
857, 860, 865,	<code>\cbdelete</code> 3, 761	<code>\float@dblend</code> 1262
870, 874, 876,	<code>\cbend</code> 3, 760	<code>\float@end</code> 1220
883, 887, 890,	<code>\cbstart</code> 3, 757	<code>\flt@float@dblend</code> .
893, 896, 897,	<code>\changebar</code> 765 1262, 1277
898, 910, 1067,	<code>changebar (env.)</code> 3	<code>\flt@float@end</code>
1080, 1092,	<code>changebargrey</code> 4 1220, 1235
1097, 1128,	<code>\changebargrey</code> 779	
1135, 1140,	<code>\changebarsep</code> ... 4,	I
1177, 1203,	32, 34, 39, 42, 777	<code>\if@cb@firstcolumn</code> .
1224, 1242,	<code>\changebarwidth</code> 20, 1144, 1179
1266, 1283, 1294 4, 33, 35,	<code>\if@cb@pagejump</code> ...
<code>\cb@topright</code> . 7, 883,	40, 43, 759, 767, 773	... 946, 989, 1000
888, 891, 896, 912	<code>\color</code> 788, 795	<code>\if@cb@trace</code>
<code>\cb@trace</code>	<code>\color@vbox</code> ... 1209,	. 19, 190, 295, 304
. 130, 132, 169,	1230, 1248, 1272	<code>\if@compatibility</code> . 196
189, 225, 230,	<code>\current@color</code> 789, 796	<code>\if@filesw</code>
375, 521, 662,	<code>\CurrentOption</code> 145	. 389, 535, 676, 936
875, 982, 1001,		<code>\if@firstcolumn</code> ...
1018, 1048, 1150	D	. 966, 978, 996,
<code>\cb@trace@connect</code> .	<code>\DeclareOption</code>	1018, 1034, 1055
..... 229,	. 50, 51, 52, 53,	<code>\if@twocolumn</code>
241, 258, 277,	54, 55, 56, 57, 913, 965,
285, 304, 323,	58, 59, 60, 61,	977, 995, 1017,
444, 591, 731, 753	62, 63, 83, 84,	1032, 1053, 1239
<code>\cb@trace@defpoint</code> .	103, 104, 122,	<code>\if@twoside</code>
..... 224,	123, 124, 125,	. 36, 920, 926, 972
238, 250, 267,	126, 127, 128,	<code>\ifodd</code> 437, 440,
282, 295, 313,	135, 137, 140, 143	584, 587, 724,
400, 546, 687, 752	<code>\DeclareRobustCommand</code>	727, 886, 901,
<code>\cb@trace@pop</code> 783	914, 917, 923,
... 132, 177, 1133	<code>\deletebarwidth</code> ...	967, 970, 976, 1097
<code>\cb@trace@push</code> 4, 763, 775	<code>\ignorespacesafterend</code>
... 130, 177, 1141	<code>\directlua</code> 106 847
<code>\cb@trace@stack</code> 129,	<code>\driver</code> 196	
176, 831, 1020,	<code>\DVIPS</code> 202, 212	L
1042, 1083,	<code>\DVItO_{PS}</code> 201, 211	<code>\lastxpos</code> 683
1201, 1222,		<code>\lastypos</code> 683
1240, 1264, 1281	E	<code>\level</code> 939, 942
<code>\cb@vtryfc</code> . 1047, 1064	<code>\egroup</code> 219	<code>\LN</code> 200, 210
<code>\cb@write</code>	<code>\emTeX</code> 203, 213	<code>\ltx@@xympar</code> 1215, 1218
.. 1106, 1145, 1156	<code>\end</code> 448, 453, 454	<code>\ltx@end@dblfloat</code> .
<code>\cb@writeAux</code> .. 893,	<code>\end@dblfloat</code> 1237, 1276 1237, 1276
908, 935, 1022, 1050	<code>\end@float</code>	<code>\ltx@end@float</code> 1199,
<code>\cb@writepagejump</code> .	.. 1199, 1234, 1259	1213, 1217, 1234
... 950, 990, 1004	<code>\endchangebar</code> 765	<code>\ltx@footnotetext</code> .
<code>\cb@writexy</code>	<code>\endlinechar</code> 1279, 1284, 1288
. 69, 70, 72, 75,	370, 516, 657, 1121	<code>\ltx@makecol</code> 1015, 1031
89, 90, 92, 95,	environments:	<code>\ltx@mpfootnotetext</code>
109, 110, 112,	changebar 3	.. 1291, 1295, 1299

<code>\ltx@vtryfc</code> .	1046, 1052	<code>\pdflastxpos</code> ..	396, 542	284, 289, 290,
<code>\luaTeX</code>	208, 218	<code>\pdflastypos</code> ..	396, 542	297, 308, 315, 479
		<code>\pdfliteral</code> ...	423,	
			425, 432, 479,	
M			569, 571, 579,	T
<code>\marginparsep</code>	778		620, 710, 712, 719	<code>\tempa</code> 198, 210, 211,
				212, 213, 214,
N		<code>\pdfoutput</code> ..	66, 67,	215, 216, 217, 218
<code>\newif</code>	19, 20, 946		152, 155, 182, 184	<code>\Textures</code>
<code>\nochangebars</code> ...	4, 769	<code>\pdfsavepos</code>		204, 214
			65, 180, 393, 539	<code>\thechangebargrey</code> .
O		<code>\pdfTeX</code>	206, 216	136
<code>\oddsidemargin</code>	29			<code>\thepage</code>
<code>outerbars</code>	4			228, 233
		S		
P		<code>\savepos</code>	680	V
<code>\PackageError</code>		<code>\sec@nd@ftw@</code> 97, 571, 572		<code>\VTeX</code>
..	78, 99, 118, 143	<code>\special</code> ..	237, 240,	205, 215
<code>\pdfextension</code>	620		243, 246, 252,	<code>\VTeXversion</code>
			262, 269, 281,	150
				X
				<code>\xeTeX</code>
				207, 217