

# The Changebar package \*

Michael Fine  
Distributed Systems Architecture

Johannes Braams  
texniek at braams.xs4all.nl

Printed May 7, 2022

<b>Contents</b>		<b>5 The implementation</b>	<b>6</b>
<b>1 Introduction</b>	<b>2</b>	5.1 Declarations And Initializations . . . . .	6
<b>2 The user interface</b>	<b>2</b>	5.2 Option Processing . . . . .	9
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 . . . . .	28
2.1.2 Specifying the bar position . . . . .	3	5.5 Macros for Making It Work Across Page Breaks	33
2.1.3 Color . . . . .	3	5.6 Macros For Managing The Stacks of Bar points .	38
2.1.4 Tracing . . . . .	3	5.7 Macros For Checking That The .aux File Is Stable . . . . .	40
2.2 Macros defined by the package . . . . .	3	5.8 Macros For Making It Work With Nested Floats/Footnotes . . . . .	42
2.3 Changebar parameters . . . . .	4		
<b>3 Deficiencies and bugs</b>	<b>4</b>		
<b>4 The basic algorithm</b>	<b>5</b>		

## Abstract

This package implements a way to indicate modifications in a L<sup>A</sup>T<sub>E</sub>X-document by putting bars in the margin. It realizes this by making use of the `\special` commands supported by ‘dvi drivers’. Currently six different drivers are supported, plus pdftex and XeT<sub>E</sub>Xsupport. More can easily be added.

---

\*This file has version number v3.6d, last revised 2022-05-06.

# 1 Introduction

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

## Features

- Changebars may be nested within each other. Each level of nesting can be given a different thickness bar.
- 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 so far 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 options<sup>1</sup> specify the driver that will be used to print the document can be indicated. The driver may be one of:

- `DVItoLN03`
- `DVItoPS`
- `DVIps`
- `emTeX`
- `TeXtures`

---

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

- V<sub>T</sub>E<sub>X</sub>
- P<sub>D</sub>F<sub>T</sub>E<sub>X</sub>
- X<sub>e</sub>T<sub>E</sub>X

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 T<sub>E</sub>X's search path.

### 2.1.2 Specifying the bar position

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

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

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

### 2.1.3 Color

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

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

### 2.1.4 Tracing

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

## 2.2 Macros defined by the package

`\cbstart` All material between the macros `\cbstart` and `\cbend` is barred. The nesting of  
`\cbend` multiple changebars is allowed. The macro `\cbstart` has an optional parameter that specifies the width of the bar. The syntax is `\cbstart[⟨dimension⟩]`. If no width is specified, the current value of the parameter `\changebarwidth` is used. Note that `\cbstart` and `\cbend` can be used anywhere but must be correctly nested with floats and footnotes. That is, one cannot have one end of the bar

inside a floating insertion and the other outside, but that would be a meaningless thing to do anyhow.

<code>changebar</code>	Apart from the macros <code>\cbstart</code> and <code>\cbend</code> a proper $\LaTeX$ environment is defined. The advantage of using the environment whenever possible is that $\LaTeX$ will do all the work of checking the correct nesting of different environments.
<code>\cbdelete</code>	The macro <code>\cbdelete</code> 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 <code>\deletebarwidth</code> will be used.
<code>\nochangebars</code>	The macro <code>\nochangebars</code> disables the changebar commands.
<code>\cbcolor</code>	This macro is defined when the <code>color</code> option is selected. It's syntax is the same as the <code>\color</code> command from the <code>color</code> package.

### 2.3 Changebar parameters

<code>\changebarwidth</code>	The width of the changebars is controlled with the $\LaTeX$ length parameter <code>\changebarwidth</code> . Its value can be changed with the <code>\setlength</code> command. Changing the value of <code>\changebarwidth</code> affects all subsequent changebars subject to the scoping rules of <code>\setlength</code> .
<code>\deletebarwidth</code>	The width of the deletebars is controlled with the $\LaTeX$ length parameter <code>\deletebarwidth</code> . Its value can be changed with the <code>\setlength</code> command. Changing the value of <code>\deletebarwidth</code> affects all subsequent deletebars subject to the scoping rules of <code>\setlength</code> .
<code>\changebarsep</code>	The separation between the text and the changebars is determined by the value of the $\LaTeX$ length parameter <code>\changebarsep</code> .
<code>changebargrey</code>	When one of the supported dvi to PostScript translators is used the ‘blackness’ of the bars can be controlled. The $\LaTeX$ counter <code>changebargrey</code> 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.

<code>outerbars</code>	The changebars will be printed in the ‘inside’ margin of your document. This means they appear on the left side of the page. When <code>twoside</code> is in effect the bars will be printed on the right side of even pages.
------------------------	---

## 3 Deficiencies and bugs

- The macros blindly use special points `\cb@minpoint` through `\cb@maxpoint`. If this conflicts with another set of macros, the results will be unpredictable. (What is really needed is a `\newspecialpoint`, analogous to `\newcount` etc. — it’s not provided because the use of the points is rather rare.)
- There is a limit of  $(\cb@maxpoint - \cb@minpoint + 1)/4$  bars per page (four special points per bar). Using more than this number yields unpredictable

results (but that could be called a feature for a page with so many bars). This limitation could be increased if desired. There is no such limit with PDF<sub>T</sub>E<sub>X</sub> or Xe<sub>T</sub>E<sub>X</sub>.

- 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  $\text{\LaTeX}$  floats are not split but footnotes may be. The simplest fix to this is to prevent footnotes from being split but this may make  $\text{\TeX}$  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  $\backslash\text{cbend}$  is processed, and the flag unset, before the output routine is called. If this happens, special action is required to generate an artificial end and begin to be added to page  $n$  and  $n + 1$  respectively, as it is not possible to use a flag to signal the output routine that a bar crosses a page break.

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

This approach is sufficiently general to allow nested bars, bars in floating insertions, and bars around floating insertions. Bars inside floats and footnotes are handled in the same way as bars in regular text. Bars that encompass floats or footnotes are handled by creating an additional bar that floats with the floating material. Modifications to the appropriate  $\LaTeX$  macros check for this condition and add the extra bar.

## 5 The implementation

### 5.1 Declarations And Initializations

`\cb@maxpoint` The original version of `changebar.sty` only supported the `DVItoLN03` specials. The LN03 printer has a maximum number of points that can be defined on a page. Also for some PostScript printers the number of points that can be defined can be limited by the amount of memory used. Therefore, the consecutive numbering of points has to be reset when the maximum is reached. This maximum can be adapted to the printers needs.

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

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

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

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

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

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

```

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

```

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

```

7 \newcount\cb@topleft
8 \newcount\cb@topright
9 \newcount\cb@botleft
10 \newcount\cb@botright

```

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

```

11 \newcount\cb@cnta
12 \newcount\cb@cntb
13 \newdimen\cb@dima

```

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

```

14 \newdimen\cb@curbarwd

```

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

```

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

```
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<sub>E</sub>X-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\langle dimen\rangle`. 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\langle dimen\rangle`. Thus the characters that delimit the argument of `\cb@removedim` also have to have category code 12. To keep the changes local the macro `\cb@removedim` is defined in a group.

```
49 {\catcode'\p=12\catcode'\t=12 \gdef\cb@removedim#1pt{#1}}
```

## 5.2 Option Processing

The user should select the specials that should be used by specifying the driver name as an option to the `\usepackage` call. Possible choices are:

- DVItolN03
- DVItOPS
- DVIPs
- emT<sub>E</sub>X
- Textures
- VT<sub>E</sub>X
- PDFT<sub>E</sub>X
- XeT<sub>E</sub>X

The intent is that the driver names should be case-insensitive, but the following code doesn’t achieve this: it only permits the forms given above and their lower-case equivalents.

```
50 \DeclareOption{DVItolN03}{\global\chardef\cb@driver@setup=0\relax}
51 \DeclareOption{dvitoln03}{\global\chardef\cb@driver@setup=0\relax}
52 \DeclareOption{DVItOPS}{\global\chardef\cb@driver@setup=1\relax}
53 \DeclareOption{dvitops}{\global\chardef\cb@driver@setup=1\relax}
54 \DeclareOption{DVIPs}{\global\chardef\cb@driver@setup=2\relax}
55 \DeclareOption{dvips}{\global\chardef\cb@driver@setup=2\relax}
56 \DeclareOption{emTeX}{\global\chardef\cb@driver@setup=3\relax}
57 \DeclareOption{emtex}{\global\chardef\cb@driver@setup=3\relax}
58 \DeclareOption{textures}{\global\chardef\cb@driver@setup=4\relax}
59 \DeclareOption{Textures}{\global\chardef\cb@driver@setup=4\relax}
60 \DeclareOption{VTEX}{\global\chardef\cb@driver@setup=5\relax}
61 \DeclareOption{vtex}{\global\chardef\cb@driver@setup=5\relax}
```

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

For the `pdftex` option we have to check that the current  $\LaTeX$  run is using `PDFTeX` and that PDF output is selected. If it is, we initialize the option and open an additional output file. If not, we ignore the option and issue a warning.

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

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

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

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

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

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

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

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

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

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

Give a warning if we cannot support the xetex option.

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

The new features of L<sup>A</sup>T<sub>E</sub>X 2<sub>ε</sub> make it possible to implement the outerbars option.

```
103 \DeclareOption{outerbars}{\def\cb@barsplace{1}}
104 \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.

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

A set of options to control tracing.

```
107 \DeclareOption{traceon}{\@cb@tracetrue}
108 \DeclareOption{traceoff}{\@cb@tracefalse}
109 \DeclareOption{tracestacks}{%
110   \let\cb@trace@stack\cb@@show@stack
111   \def\cb@trace@push#1{\cb@trace{%
112     Pushed point \the\cb@topleft\space on \noexpand#1: #1}}%
113   \def\cb@trace@pop#1{\cb@trace{%
114     Popped point \the\cb@topleft\space from \noexpand#1: #1}}%
115 }
```

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

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

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

```
118 \DeclareOption{color}{%
119   \def\cb@ps@color{\expandafter\color@to@ps\cb@current@color\@}%
120   \def\cb@color@pkg{color}}
```

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

```
121 \DeclareOption{xcolor}{%
122   \def\cb@ps@color{\expandafter\color@to@ps\cb@current@color\@}%
123   \def\cb@color@pkg{xcolor}}
```

Signal an error if an unknown option was specified.

```
124 \DeclareOption*{\OptionNotUsed\PackageError
125   {changebar}%
126   {Unrecognised option ‘\CurrentOption’\MessageBreak
127     known options are dviton3, dvitops, dvips,\MessageBreak
128     emtex, textures, pdftex, vtex and xetex,
129     grey, color, xcolor,\MessageBreak
130     outerbars, innerbars, leftbars and rightbars}}
```

The default is to have grey change bars on the left side of the text on odd pages. When  $\text{V}\text{T}\text{E}\text{X}$  is used the option `dvips` is not the right one, so in that case we have `vtex` as the default driver. When  $\text{P}\text{D}\text{F}\text{T}\text{E}\text{X}$  is producing PDF output, the `pdftex` option is selected.

```

131 \ifx\VTexversion\@undefined
132 \expandafter\ifx\csname XeTeXrevision\endcsname\@undefined
133   \ifx\pdfoutput\@undefined
134     \ExecuteOptions{innerbars,traceoff,dvips,grey}
135   \else
136     \ifnum\pdfoutput>0
137       \ExecuteOptions{innerbars,traceoff,pdftex,grey}
138     \else
139       \ExecuteOptions{innerbars,traceoff,dvips,grey}
140     \fi
141   \fi
142 \else
143   \ExecuteOptions{innerbars,traceoff,xetex,grey}
144 \fi
145 \else
146   \ExecuteOptions{innerbars,traceoff,vtex,grey}
147 \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.

```

148 \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.

```

149 \def\cb@show@stack#1{%
150   \cb@trace{%
151     stack status at #1:\MessageBreak
152     current stack: \cb@currentstack\MessageBreak
153     \@spaces end stack: \cb@endstack\MessageBreak
154     \space\space begin stack: \cb@beginstack\MessageBreak
155     history stack: \cb@historystack
156   }}

```

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

```

157 \let\cb@trace@stack\@gobble

```

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

The default is to *not* trace the stacks.

```

158 \let\cb@trace@push\@gobble
159 \let\cb@trace@pop\@gobble

```

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

```
160 \ProcessOptions\relax
```

We have to make sure that when the document is being processed by pdfL<sup>A</sup>T<sub>E</sub>X, 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.

```
161 \ifx\pdfsavepos\@undefined
162 \else
163   \ifx\pdfoutput\@undefined
164   \else
165     \ifnum\pdfoutput>0
166     \global\chardef\cb@driver@setup=6\relax
167     \fi
168   \fi
169 \fi
```

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

```
170 \newcommand{\cb@trace}[1]{%
171   \if@cb@trace
172     \GenericWarning
173       {(changebar)\@spaces\@spaces}%
174       {Package changebar: #1\@gobble}%
175   \fi
176 }
```

### 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
- DVItolPS
- DVIPs
- emT<sub>E</sub>X
- T<sub>E</sub>Xtures
- VT<sub>E</sub>X
- PDFT<sub>E</sub>X
- XeT<sub>E</sub>X

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.

```

177 \if@compatibility
178   \def\driver#1{%
179     \bgroup\edef\next{\def\noexpand\tempa{#1}}%
180     \uppercase\expandafter{\next}%
181     \def\LN{DVITOLN03}%
182     \def\DVItO{PS}{DVITOPS}%
183     \def\DVIPS{DVIPS}%
184     \def\emTeX{EMTEX}%
185     \def\Textures{TEXTURES}%
186     \def\VTex{VTEX}%
187     \def\pdfTeX{PDFTEX}%
188     \def\XeTeX{XETEX}

```

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`.

```

189   \global\chardef\cb@driver@setup=0\relax
190   \ifx\tempa\LN      \global\chardef\cb@driver@setup=0\fi
191   \ifx\tempa\DVItO  \global\chardef\cb@driver@setup=1\fi
192   \ifx\tempa\DVIPS  \global\chardef\cb@driver@setup=2\fi
193   \ifx\tempa\emTeX  \global\chardef\cb@driver@setup=3\fi
194   \ifx\tempa\Textures \global\chardef\cb@driver@setup=4\fi
195   \ifx\tempa\VTex   \global\chardef\cb@driver@setup=5\fi
196   \ifx\tempa\pdfTeX \cb@pdftexcheck\fi
197   \ifx\tempa\XeTeX  \cb@xetexcheck\fi
198   \egroup}

```

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

```

199   {\def\do{\noexpand\do\noexpand}
200   \xdef\@preamblecmds{\@preamblecmds \do\driver}
201   }
202 \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.

```

203 \def\cb@trace@defpoint#1#2{%
204   \cb@trace{%
205     defining point \the#1 at position \the#2
206     \MessageBreak
207     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.

```

208 \def\cb@trace@connect#1#2#3{%

```

```

209 \cb@trace{%
210     connecting points \the#1 and \the#2; barwidth: \the#3
211     \MessageBreak
212     cb@pagecount: \the\cb@pagecount; page \thepage}}

```

`\cb@defpoint` The macro `\cb@defpoint` is used to define one of the two points of a bar. It has two arguments, the number of the point and the distance from the left side of the paper. Its syntax is: `\cb@defpoint{<number>}{<length>}`.

`\cb@resetpoints` The macro `\cb@resetpoints` can be used to instruct the printer driver that it should send a corresponding instruction to the printer. This is really only used for the LN03 printer.

`\cb@connect` The macro `\cb@connect` is used to instruct the printer driver to connect two points with a bar. The syntax is `\cb@connect{<number>}{<number>}{<length>}`. The two `<number>`s indicate the two points to be connected; the `<length>` is the width of the bar.

```

213 \def\cb@setup@specials{%

```

The control sequence `\cb@driver@setup` expands to a number which indicates the driver that will be used. The original `changebar.sty` was written with only the `\special` syntax of the program `DVItoLN03` (actually one of its predecessors, `ln03dvi`). Therefore this syntax is defined first.

```

214 \ifcase\cb@driver@setup
215   \def\cb@defpoint##1##2{%
216     \special{ln03:defpoint \the##1(\the##2,)}%
217     \cb@trace@defpoint##1##2}
218   \def\cb@connect##1##2##3{%
219     \special{ln03:connect \the##1\space\space \the##2\space \the##3}%
220     \cb@trace@connect##1##2##3}
221   \def\cb@resetpoints{%
222     \special{ln03:resetpoints \cb@minpoint \space\cb@maxpoint}}

```

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

```

223 \or
224   \def\cb@defpoint##1##2{%
225     \special{dvitops: inline
226               \expandafter\cb@removedim\the##2\space 6.5536 mul\space
227               /CBarX\the##1\space exch def currentpoint exch pop
228               /CBarY\the##1\space exch def}%
229     \cb@trace@defpoint##1##2}
230   \def\cb@connect##1##2##3{%
231     \special{dvitops: inline
232               gsave \cb@ps@color\space
233               \expandafter\cb@removedim\the##3\space 6.5536 mul\space
234               CBarX\the##1\space\space CBarY\the##1\space\space moveto
235               CBarX\the##2\space\space CBarY\the##2\space\space lineto
236               stroke grestore}%

```

```

237 \cb@trace@connect###1##2###3}
238 \let\cb@resetpoints\relax

```

The program DVIPs by Thomas Rokicki is also supported. The PostScript code is nearly the same as for DVItO<sub>PS</sub>, but the coordinate space has a different dimension. Also this code has been made resolution independent, whereas the code for DVItO<sub>PS</sub> 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.

```

239 \or
240 \def\cb@defpoint###1##2{%
241 \special{ps:
242 \expandafter\cb@removedim\the##2\space
243 Resolution\space mul\space 72.27\space div\space
244 /CBarX\the##1\space exch def currentpoint exch pop
245 /CBarY\the##1\space exch def}%
246 \cb@trace@defpoint###1##2}
247 \def\cb@connect###1##2###3{%
248 \special{ps:
249 gsave \cb@ps@color\space
250 \expandafter\cb@removedim\the##3\space
251 Resolution\space mul\space 72.27\space div\space
252 setlinewidth
253 CBarX\the##1\space\space CBarY\the##1\space\space moveto
254 CBarX\the##2\space\space CBarY\the##2\space\space lineto
255 stroke grestore}%
256 \cb@trace@connect###1##2###3}
257 \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.

```

258 \or
259 \def\cb@defpoint###1##2{%
260 \special{em:point \the##1,\the##2}%
261 \cb@trace@defpoint###1##2}
262 \def\cb@connect###1##2###3{%
263 \special{em:line \the##1,\the##2,\the##3}%
264 \cb@trace@connect###1##2###3}
265 \let\cb@resetpoints\relax

```

The following definitions are validated with T<sub>E</sub>X<sub>t</sub>ures version 1.7.7, but will very likely also work with later releases of T<sub>E</sub>X<sub>t</sub>ures.

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.

```

266 \or
267 \def\cb@defpoint###1##2{%

```

```

268 \special{postscript 0 0 transform}% leave [x,y] on the stack
269 \special{rawpostscript
270     \expandafter\cb@removedim\the##2\space
271     /CBarX\the##1\space exch def
272     itransform exch pop
273     /CBarY\the##1\space exch def}%
274 \ifcb@trace\cb@trace@defpoint##1##2\fi}
275 \def\cb@connect##1##2##3{%
276 \special{rawpostscript
277     gsave 1 setlinecap \cb@ps@color\space
278     \expandafter\cb@removedim\the##3\space
279     setlinewidth
280     CBarX\the##1\space\space CBarY\the##1\space\space moveto
281     CBarX\the##2\space\space CBarY\the##2\space\space lineto
282     stroke grestore}%
283 \ifcb@trace\cb@trace@connect##1##2##3\fi}
284 \let\cb@resetpoints\relax

```

The following definitions were kindly provided by Michael Vulis.

```

285 \or
286 \def\cb@defpoint##1##2{%
287 \special{pS:
288     \expandafter\cb@removedim\the##2\space
289     Resolution\space mul\space 72.27\space div\space
290     /CBarX\the##1\space exch def currentpoint exch pop
291     /CBarY\the##1\space exch def}%
292 \cb@trace@defpoint##1##2}
293 \def\cb@connect##1##2##3{%
294 \special{pS:
295     gsave \cb@ps@color\space
296     \expandafter\cb@removedim\the##3\space
297     Resolution\space mul\space 72.27\space div\space
298     setlinewidth
299     CBarX\the##1\space\space CBarY\the##1\space\space moveto
300     CBarX\the##2\space\space CBarY\the##2\space\space lineto
301     stroke grestore}%
302 \cb@trace@connect##1##2##3}
303 \let\cb@resetpoints\relax

```

The code for PDF<sub>TEX</sub> is more elaborate as the calculations have to be done in <sub>TEX</sub>. `\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.

```

304 \or
305 \immediate\closeout\cb@writexy
306 \immediate\openin\cb@ready=\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.

```

307 \def\cb@pdfpoints{}
308 \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.
309 \def\cb@findpdfpoint##1##2{%
310     \edef\cb@temp
311         {\noexpand\cb@@findpdfpoint{\the##1}{\the##2}}%
312     \cb@temp
313 }

\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.
314 \def\cb@@findpdfpoint##1##2{%
315     \ifnum##2<\cb@pdfpagenr\relax\else
316         \cb@pdfready{##2}%
317     \fi
318     \let\cb@pdfx\relax
319     \ifx\cb@pdfpoints\@empty\else
320         \ifnum##2<0\relax
321         \else
322             \edef\cb@temp{\noexpand\cb@pdfind{##1}{##2}\cb@pdfpoints\relax{}}%
323             \cb@temp
324         \fi
325     \fi
326 }

\cb@pdfind The \cb@pdfind recursively searches through \cb@pdfpoints to find point #1 on
pagecount #2. \cb@pdfpoints contains entries of the form <pointnr>.<pagecount>p<x>,<y>,<z>pt.
When the point is found it is removed from \cb@pdfpoints. #9 contains the cu-
mulative head of the list to construct the new list with the entry removed. #3-#8
are for pattern matching.
327 \def\cb@pdfind##1##2##3.##4p##5,##6,##7pt##8\relax##9{%
328     \def\cb@next{\cb@pdfind{##1}{##2}##8\relax{##9##3.##4p##5,##6,##7pt}}%
329     \ifnum ##1=##3
330         \ifnum ##2=##4
331             \def\cb@pdfx{##5sp}%
332             \def\cb@pdfy{##6sp}%
333             \def\cb@pdfz{##7pt}%
334             \let\cb@next\relax
335             \gdef\cb@pdfpoints{##9##8}%
336         \fi
337     \fi
338     \ifx\relax##8\relax
339         \let\cb@next\relax
340     \fi

```

```

341   \cb@next
342 }%

```

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

```

343 \def\cb@pdfready##1{%
344   \let\cb@next\relax
345   \ifeof\cb@ready
346     \global\let\cb@pdfpagenr\cb@maxpoint
347   \else
348     {\endlinechar=-1\read\cb@ready to\cb@temp
349     \ifx\cb@temp@empty\else
350       \expandafter\cb@pdfparsexy\cb@temp
351       \ifnum\cb@pdfpg<0\else
352         \xdef\cb@pdfpoints{\cb@pdfpoints\cb@temp}%
353         \cb@trace{PDFpoints=\cb@pdfpoints}%
354         \global\let\cb@pdfpagenr\cb@pdfpg
355       \fi
356       \ifnum\cb@pdfpg>##1\else
357         \global\def\cb@next{\cb@pdfready{##1}}%
358       \fi
359     \fi
360   }%
361   \fi
362   \cb@next
363 }%

```

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

```

364 \def\cb@pdfparsexy##1.##2p##3,##4,##5pt{%
365   \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.

```

366 \def\cb@defpoint##1##2{%
367   \if@filesw
368     \begingroup
369     \edef\point{{\the##1}{\the\cb@pagecount}}%
370     \let\the=\z@
371     \pdfsavepos
372     \edef\cb@temp{\write\@auxout
373     {\string\cb@pdfxy\point
374     {\the\pdflastxpos}{\the\pdflastypos}{\the##2}}}%
375     \cb@temp
376   \endgroup

```

```

377   \fi
378   \cb@trace@defpoint##1##2%
379 }%

```

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

```

380 \def\cb@cvt pct##1{%
381   \ifnum##1<0 0\else
382   \ifnum##1>99 1\else
383   \ifnum##1<10 0.0\the##1\else
384   0.\the##1\fi\fi\fi}

```

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

```

385 \def\cb@connect##1##2##3{%
386   \cb@findpdfpoint{##1}\cb@pagecount
387   \ifx\cb@pdfx\relax\cb@rerun
388   \else
389   \let\cb@pdftopy\cb@pdfy
390   \cb@findpdfpoint{##2}\cb@pagecount
391   \ifx\cb@pdfx\relax\cb@rerun
392   \else

```

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

```

393   \begingroup
394   \cb@dima=\cb@pdfz
395   \advance\cb@dima by-\cb@pdfx
396   \advance\cb@dima by1in
397   \cb@dima=0.996264009963\cb@dima\relax

```

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

```

398   \ifx\cb@current@color\undefined
399     \def\cb@temp{\cb@cvt pct\c@changebargrey}%
400     \pdfliteral{q \cb@temp\space g \cb@temp\space G}%
401   \else
402     \pdfliteral{q \cb@current@color}%
403   \fi
404   \edef\cb@temp{\expandafter\cb@removedim\the\cb@dima\space}%
405   \cb@dima=\cb@pdftopy
406   \advance\cb@dima-\cb@pdfy\relax
407   \cb@dima=0.996264009963\cb@dima\relax
408   ##3=0.996264009963##3\relax
409   \pdfliteral direct{\expandafter\cb@removedim\the##3 w
410     \cb@temp 0 m
411     \cb@temp \expandafter\cb@removedim\the\cb@dima\space 1 S Q}%

```

```

412     \endgroup
    We look up the two unused points to get them removed from \cb@pdfpoints.
413     \cb@cntb=##1\relax
414     \ifodd\cb@cntb\advance\cb@cntb 1\else\advance\cb@cntb -1\fi
415     \cb@findpdfpoint\cb@cntb\cb@pagecount
416     \cb@cntb=##2\relax
417     \ifodd\cb@cntb\advance\cb@cntb 1\else\advance\cb@cntb -1\fi
418     \cb@findpdfpoint\cb@cntb\cb@pagecount
419     \fi
420 \fi
421 \cb@trace@connect##1##2##3%
422 }%

```

`\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<sup>A</sup>T<sub>E</sub>X.

```

423 \gdef\cb@checkPdfxy##1##2##3##4##5{%
424   \cb@@findpdfpoint{##1}{##2}%
425 %   \end{macrocode}
426 %\begin{changebar}
427 %   \begin{macrocode
428   \ifdim##3sp=\cb@pdfx\relax
429   \ifdim##4sp=\cb@pdfy\relax
430 %   \end{macrocode}
431 %\end{changebar}
432 %   \begin{macrocode
433     \ifdim##5=\cb@pdfz\relax
434     \else
435     \cb@error
436     \fi
437   \else
438     \cb@error
439   \fi
440 \else
441   \cb@error
442 \fi
443 }

```

For PDF<sub>T</sub>E<sub>X</sub> we don't need a limit on the number of bar points.

```

444 \def\cb@maxpoint{9999999}
445 \let\cb@resetpoints\relax
446 \or

```

The code for Xe<sub>T</sub>E<sub>X</sub> is, like for PDF<sub>T</sub>E<sub>X</sub>, more elaborate as the calculations have to be done in T<sub>E</sub>X. `\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.

```

447 \immediate\closeout\cb@writexy
448 \immediate\openin\cb@ready=\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.
449 \def\cb@pdfpoints{}
450 \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.
451 \def\cb@findpdfpoint##1##2{%
452     \edef\cb@temp
453         {\noexpand\cb@@findpdfpoint{\the##1}{\the##2}}%
454     \cb@temp
455 }

\pdfliteral For XeTeX we mimick PDFTeX's command \pdfliteral.
456 \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.
457 \def\cb@@findpdfpoint##1##2{%
458     \ifnum##2<\cb@pdfpagenr\relax\else
459         \cb@pdfready{##2}%
460     \fi
461     \let\cb@pdfx\relax
462     \ifx\cb@pdfpoints\@empty\else
463         \ifnum##2<0\relax
464         \else
465             \edef\cb@temp{\noexpand\cb@pdffind{##1}{##2}\cb@pdfpoints\relax{}}%
466             \cb@temp
467         \fi
468     \fi
469 }

\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 cu-
mulative head of the list to construct the new list with the entry removed. #3-#8
are for pattern matching.
470 \def\cb@pdffind##1##2##3.##4p##5,##6,##7pt##8\relax##9{%
471     \def\cb@next{\cb@pdffind{##1}{##2}##8\relax{##9##3.##4p##5,##6,##7pt}}%
472     \ifnum ##1=##3
473         \ifnum ##2=##4
474             \def\cb@pdfx{##5sp}%
475             \def\cb@pdfy{##6sp}%
476             \def\cb@pdfz{##7pt}%

```

```

477     \let\cb@next\relax
478     \gdef\cb@pdfpoints{##9##8}%
479     \fi
480 \fi
481 \ifx\relax##8\relax
482     \let\cb@next\relax
483 \fi
484 \cb@next
485 }%

```

`\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.

```

486 \def\cb@pdfreadxy##1{%
487     \let\cb@next\relax
488     \ifeof\cb@readxy
489         \global\let\cb@pdfpagenr\cb@maxpoint
490     \else
491         {\endlinechar=-1\read\cb@readxy to\cb@temp
492         \ifx\cb@temp\@empty\else
493             \expandafter\cb@pdfparsexy\cb@temp
494             \ifnum\cb@pdfpg<0\else
495                 \xdef\cb@pdfpoints{\cb@pdfpoints\cb@temp}%
496                 \cb@trace{PDFpoints=\cb@pdfpoints}%
497                 \global\let\cb@pdfpagenr\cb@pdfpg
498             \fi
499             \ifnum\cb@pdfpg>##1\else
500                 \global\def\cb@next{\cb@pdfreadxy{##1}}%
501             \fi
502         \fi
503     }%
504 \fi
505 \cb@next
506 }%

```

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

```

507 \def\cb@pdfparsexy##1.##2p##3,##4,##5pt{%
508     \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.

```

509 \def\cb@defpoint##1##2{%
510     \if@filesw
511         \begingroup
512         \edef\point{{\the##1}{\the\cb@pagecount}}%

```

```

513     \let\the=\z@
514     \pdfsavepos
515     \edef\cb@temp{\write\@auxout
516       {\string\cb@pdfxy\point
517         {\the\pdflastxpos}{\the\pdflastypos}{\the##2}}}%
518     \cb@temp
519   \endgroup
520 \fi
521 \cb@trace@defpoint##1##2%
522 }%

```

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

```

523 \def\cb@cvt pct##1{%
524   \ifnum##1<0 0\else
525   \ifnum##1>99 1\else
526   \ifnum##1<10 0.0\the##1\else
527   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.

```

528 \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.

```

529 \def\cb@connect##1##2##3{%
530   \cb@findpdfpoint{##1}\cb@pagecount
531   \ifx\cb@pdfx\relax\cb@rerun
532   \else
533     \let\cb@pdfx\cb@pdfy
534     \cb@findpdfpoint{##2}\cb@pagecount
535     \ifx\cb@pdfx\relax\cb@rerun
536     \else

```

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

```

537     \begingroup
538     \cb@dima=\cb@pdfz
539     \advance\cb@dima by-\cb@pdfx
540     \advance\cb@dima by1in
541     \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.

```

542     \ifx\cb@current@color\undefined
543     \def\cb@temp{\cb@cvt pct\c@changebargrey}%

```

```

544         \pdfliteral{q \cb@temp\space g \cb@temp\space G}%
545     \else
546         \pdfliteral{q \expandafter\sec@nd@ftw@\cb@current@color\space RG
547             \expandafter\sec@nd@ftw@\cb@current@color\space rg}%
548     \fi
549     \edef\cb@temp{\expandafter\cb@removedim\the\cb@dima\space}%
550     \cb@dima=\cb@pdfx
551     \advance\cb@dima-\cb@pdfy\relax
552     \cb@dima=\cb@pdf@scale\cb@dima\relax
553     ##3=\cb@pdf@scale##3\relax
554     \pdfliteral{\expandafter\cb@removedim\the##3 w
555         \cb@temp 0 m
556         \cb@temp \expandafter\cb@removedim\the\cb@dima\space 1 S Q}%
557 \endgroup

```

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

```

558     \cb@cntb=##1\relax
559     \ifodd\cb@cntb\advance\cb@cntb 1\else\advance\cb@cntb -1\fi
560     \cb@findpdfpoint\cb@cntb\cb@pagecount
561     \cb@cntb=##2\relax
562     \ifodd\cb@cntb\advance\cb@cntb 1\else\advance\cb@cntb -1\fi
563     \cb@findpdfpoint\cb@cntb\cb@pagecount
564 \fi
565 \fi
566 \cb@trace@connect##1##2##3%
567 }%

```

\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<sup>A</sup>T<sub>E</sub>X.

```

568 \gdef\cb@checkPdfxy##1##2##3##4##5{%
569     \cb@findpdfpoint{##1}{##2}%
570 % \end{macrocode}
571 %\begin{changebar}
572 % \begin{macrocode
573     \ifdim##3sp=\cb@pdfx\relax
574     \ifdim##4sp=\cb@pdfy\relax
575 % \end{macrocode}
576 %\end{changebar}
577 % \begin{macrocode
578     \ifdim##5=\cb@pdfz\relax
579     \else
580     \cb@error
581     \fi
582     \else
583     \cb@error
584     \fi
585     \else
586     \cb@error
587     \fi
588 }

```

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

```
589 \def\cb@maxpoint{9999999}
590 \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.

```
591 \else
592 \PackageWarning{Changebar}{changebars not supported in unknown setup}
593 \def\cb@defpoint##1##2{\cb@trace@defpoint##1##2}
594 \def\cb@connect##1##2##3{\cb@trace@connect##1##2##3}
595 \let\cb@resetpoints\relax
596 \fi
```

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

```
597 \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`.

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

`\cbend` The macro `\cbend` (surprisingly) ends a changebar. The macros `\cbstart` and `\cbend` can be used when the use of a proper L<sup>A</sup>T<sub>E</sub>X environment is not possible.

```
601 \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`.

```
602 \newcommand*{\cbdelete}{\@ifnextchar [%]
603 \cb@delete}%
604 \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.

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

`\changebar` and `\endchangebar` The macros `\changebar` and `\endchangebar` have the same function as `\cbstart` and `\cbend` but they can be used as a L<sup>A</sup>T<sub>E</sub>X environment to enforce correct nesting. They can *not* be used in the `tabular` and `tabbing` environments.

```
606 \newenvironment{changebar}%
607 \cb@start}%
608 \cb@start[\changebarwidth]}%
609 \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.

```

610 \newcommand*\nochangebars}{%
611   \def\cb@start[##1]{}%
612   \def\cb@delete[##1]{}%
613   \let\cb@end\relax}

```

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

```

614 \newlength{\changebarwidth}
615 \setlength{\changebarwidth}{2pt}

```

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

```

616 \newlength{\deletebarwidth}
617 \setlength{\deletebarwidth}{4pt}

```

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

```

618 \newlength{\changebarsep}
619 \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).

```

620 \newcounter{changebargrey}
621 \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<sup>L</sup>A<sub>T</sub>E<sub>X</sub> we need to pass that information on to that package.

```

622 \@ifpackagewith{changebar}{\csname cb@color@pkg\endcsname}{%
623   \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<sub>E</sub>X 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.

```

624 \DeclareRobustCommand\cbcolor{%
625   \@ifnextchar[\undeclaredcbcolor\declaredcbcolor}

```

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

```
626 \def\@undeclaredcbcolor[#1]#2{%
627   \begingroup
628   \color[#1]{#2}%
629   \global\let\cb@current@color\current@color
630 \endgroup
631 \ignorespaces
632 }
```

`\@declaredcbcolor`

```
633 \def\@declaredcbcolor#1{%
634   \begingroup
635   \color{#1}%
636   \global\let\cb@current@color\current@color
637 \endgroup
638 \ignorespaces}%
639 }%
```

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

```
640 \def\cbcolor{\@ifnextchar[%]
641   \@cbcolor\@cbcolor}%
642 \def\@cbcolor[#1]#2{\cb@colwarn\def\@cbcolor[##1]##2{}}%
643 \def\@cbcolor#1{\cb@colwarn\def\@cbcolor##1{}}%
644 \def\cb@colwarn{\PackageWarning{Changebar}%
645   {You didn't specify the option 'color';\MessageBreak
646   your command \string\cbcolor\space will be ignored}}%
647 }
```

## 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.

```
648 \def\cb@start[#1]{%
649   \cb@topleft=\cb@nextpoint
```

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

```
650 \cb@curbarwd#1\relax
651 \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.

```
652 \cb@checkpage\z@
```

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

```
653 \cb@canta\cb@pagecount
```

```

654 \cb@pagecount\cb@page\advance\cb@pagecount\m@ne
655 \ifvmode
656   \cb@setBeginPoints
657 \else
658   \vbox to \z@{%

```

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

```

659     \vskip -\ht\strutbox
660     \cb@setBeginPoints
661     \vskip \ht\strutbox}%
662 \fi

```

Restore `\cb@pagecount`.

```

663 \cb@pagecount\cb@canta
664 \cb@advancePoint}

```

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

```

665 \def\cb@advancePoint{%
666   \global\advance\cb@nextpoint by 4\relax
667   \ifnum\cb@nextpoint>\cb@maxpoint
668     \global\cb@nextpoint=\cb@minpoint\relax
669   \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.

```

670 \def\cb@end{%
671   \cb@trace@stack{end of bar on page \the\c@page}%
672   \cb@pop\cb@currentstack
673   \ifnum\cb@toleft=\cb@nil
674     \PackageWarning{Changebar}%
675     {Badly nested changebars; Expect erroneous results}%
676   \else

```

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

```

677   \cb@checkpage\thr@@

```

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

```

678   \cb@canta\cb@pagecount
679   \cb@pagecount\cb@page\advance\cb@pagecount\m@ne
680   \ifvmode
681     \cb@setEndPoints
682   \else
683     \vadjust{\cb@setEndPoints}%

```

```

684   \fi
685   \cb@pagecount\cb@cnta
686   \fi
687   \ignorespaces}

```

`\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).

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

First store the identifiers in temporary registers.

```

689 \cb@cnta\cb@topleft\relax
690 \advance\cb@cnta by #1\relax
691 \cb@cntb\cb@page\relax
692 \cb@dima\cb@curbarwd\relax

```

Then pop the history stack.

```
693 \cb@pop\cb@historystack
```

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

```

694 \ifnum\cb@topleft=\cb@nil
695 \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.

```

696   \cb@FindPageNum
697   \ifnum\cb@topleft>\cb@maxpoint\else

```

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

```

698     \cb@cntb\cb@page
699   \fi

```

Now we restore the history stack to its original state.

```

700   \@whilenum\cb@topleft>\cb@nil\do{%
701     \cb@push\cb@historystack
702     \cb@pop\cb@tempstack}%
703   \fi

```

Finally return the correct values

```

704 \advance\cb@cnta by -#1\relax
705 \cb@topleft\cb@cnta\relax
706 \cb@page\cb@cntb\relax
707 \cb@curbarwd\cb@dima\relax
708 }

```

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

```

709 \def\cb@FindPageNum{%
710   \ifnum\cb@topleft=\cb@cnta

```

We have found it, exit the macro, otherwise push the current entry on the temporary stack and pop a new one from the history stack.

```
711 \else
712   \cb@push\cb@tempstack
713   \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 T<sub>E</sub>X into an endless loop if it wasn't detected and handled.

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

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

```
716     \cb@topleft\cb@maxpoint\advance\cb@topleft\@ne
717   \else
```

Recursively call ourselves.

```
718     \expandafter\expandafter\expandafter\cb@FindPageNum
719     \fi
720 \fi
721 }%
```

`\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.

```
722 \def\cb@setBeginPoints{%
723   \cb@topright=\cb@topleft\advance\cb@topright by\@ne
724   \cb@cntb=\cb@pagecount
725   \divide\cb@cntb by\tw@
726   \ifodd\cb@cntb
727     \cb@defpoint\cb@topleft\cb@even@left
728     \cb@defpoint\cb@topright\cb@even@right
729   \else
730     \cb@defpoint\cb@topleft\cb@odd@left
731     \cb@defpoint\cb@topright\cb@odd@right
732   \fi
733   \cb@writeAux\cb@topleft
734 }
```

`\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.

```

735 \def\cb@setEndPoints{%
736   \cb@topright=\cb@topleft\advance\cb@topright by\@ne
737   \cb@botleft=\cb@topleft\advance\cb@botleft by\tw@
738   \cb@botright=\cb@topleft\advance\cb@botright by\thr@
739   \cb@cntb=\cb@pagecount
740   \divide\cb@cntb by\tw@
741   \ifodd\cb@cntb
742     \cb@defpoint\cb@botleft\cb@even@left
743     \cb@defpoint\cb@botright\cb@even@right
744   \else
745     \cb@defpoint\cb@botleft\cb@odd@left
746     \cb@defpoint\cb@botright\cb@odd@right
747   \fi
748   \cb@writeAux\cb@botright
749   \edef\cb@leftbar{%
750     \noexpand\cb@connect{\cb@topleft}{\cb@botleft}{\cb@curbarwd}}%
751   \edef\cb@rightbar{%
752     \noexpand\cb@connect{\cb@topright}{\cb@botright}{\cb@curbarwd}}%

```

In twocolumn pages always use outerbars

```

753   \if@twocolumn
754     \ifodd\cb@pagecount\cb@rightbar\else\cb@leftbar\fi
755   \else
756     \ifcase\cb@barsplace

```

0=innerbars

```

757     \ifodd\cb@cntb
758       \cb@rightbar
759     \else
760       \if@twoside\cb@leftbar\else\cb@rightbar\fi
761     \fi
762   \or

```

1=outerbars

```

763     \ifodd\cb@cntb
764       \cb@leftbar
765     \else
766       \if@twoside\cb@rightbar\else\cb@leftbar\fi
767     \fi
768   \or

```

2=leftbars

```

769     \cb@leftbar
770   \or

```

3=rightbars

```

771     \cb@rightbar
772   \fi
773 \fi
774 }%

```

`\cb@writeAux` The macro `\cb@writeAux` writes information about a changebar point to the auxiliary file. The number of the point, the pagenumber and the width of the bar are written out as arguments to `\cb@barpoint`. This latter macro will be expanded when the auxiliary file is read in. The macro assumes that the width of bar is stored in `\cb@curbarwd`.

The code is only executed when auxiliary files are enabled, as there's no sense in trying to write to an unopened file.

```

775 \def\cb@writeAux#1{%
776   \if@filesw
777     \begingroup
778       \edef\point{\the#1}%
779       \edef\level{\the\cb@curbarwd}%
780       \let\the=\z@
781       \edef\cb@temp{\write@auxout
782         {\string\cb@barpoint{\point}{\the\cb@pagecount}{\level}}}%
783       \cb@temp
784     \endgroup
785   \fi}

```

## 5.5 Macros for Making It Work Across Page Breaks

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

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

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

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

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

```
788 \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.

```
789 \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.

```

790 \def\cb@writepagejump#1{
791   \cb@cntb=\cb@pagecount
792   \advance\cb@cntb by#1\relax
793   \global\setbox\@leftcolumn\vbox to\colht{%
794     \edef\cb@temp{\write@auxout{\string\cb@pagejump{\the\cb@cntb}}}%
795     \cb@temp
796     \dimen@ \dp\@leftcolumn
797     \unvbox \@leftcolumn
798     \vskip -\dimen@
799   }%
800 }

```

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

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

`\cb@checkpagecount` This macro checks that `\cb@pagecount` is correct at the beginning of a column or page. First we ensure that `\cb@pagecount` has the proper parity: odd in the righthand column of a twocolumn page, even in the lefthand column of a twocolumn page and in onecolumn pages.

```
804 \def\cb@checkpagecount{%
805   \if@twocolumn
806     \if@firstcolumn
807       \ifodd\cb@pagecount\global\advance\cb@pagecount by\@ne\fi
808     \fi
809   \else
810     \ifodd\cb@pagecount\global\advance\cb@pagecount by\@ne\fi
811   \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.

```
812 \if@twoside
813   \cb@cntb=\cb@pagecount
814   \divide\cb@cntb by\tw@
815   \advance\cb@cntb by-\c@page
816   \ifodd\cb@cntb
```

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

```
817   \if@twocolumn
818     \if@firstcolumn
819       \@whilenum\cb@pagecount>\cb@nextpagejump\do{%
820         \expandafter\cb@poppagejump\cb@pagejumplst\relax}%
821       \ifnum\cb@pagecount=\cb@nextpagejump
822         \cb@trace{Page jump: \string\cb@pagecount=\the\cb@pagecount}
823         \global\advance\cb@pagecount by\tw@
824         \global\@cb@pagejumptrue
825       \else
```

```

826         \global\cb@pagejumpfalse
827     \fi
828     \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.
829         \ifcb@pagejump
830         \cb@writepagejump{-3}%
831     \fi
832     \fi
833 \fi
834 \else
    Here \cb@pagecount is incorrect.
835     \iftwocolumn
836     \iffirstcolumn
837         \global\advance\cb@pagecount by\tw@
838         \global\cb@pagejumpfalse
839     \else
840         \ifcb@pagejump
841         \cb@trace{Page jump annulled, %
842                 \string\cb@pagecount=\the\cb@pagecount}
843     \else
844         \cb@writepagejump{-1}%
845         \global\advance\cb@pagecount by\tw@
846         \cb@rerun
847     \fi
848     \fi
849     \else
850         \global\advance\cb@pagecount by\tw@
851     \fi
852 \fi
853 \fi
854 }

```

`\@makecol` These internal L<sup>A</sup>T<sub>E</sub>X 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.

```

855 \let\ltx@makecol\@makecol
856 \def\cb@makecol{%
857     \iftwocolumn
858     \cb@trace{Twocolumn: \iffirstcolumn Left \else Right \fi column}%
859     \fi

```

```

860 \cb@trace@stack{before makecol, page \the\c@page,
861         \string\cb@pagecount=\the\cb@pagecount}%
862 \let\cb@writeAux\@gobble

```

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

```

863 \cb@checkpagecount
864 \setbox\@cclv \vbox{%
865   \cb@resetpoints
866   \cb@startSpanBars
867   \unvbox\@cclv
868   \boxmaxdepth\maxdepth}%
869 \global\advance\cb@pagecount by\@ne
870 \cb@buildstack\cb@processActive
871 \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`.

```

872 \if@twocolumn
873   \global\setbox\@outputbox \vbox to\@colht {%
874     \if@firstcolumn\write\@auxout{\string\cb@firstcolumntrue}%
875     \else\write\@auxout{\string\cb@firstcolumnfalse}%
876     \fi
877     \dimen@ \dp\@outputbox
878     \unvbox \@outputbox
879     \vskip -\dimen@
880   }%
881 \fi
882 \cb@trace@stack{after makecol, page \the\c@page,
883         \string\cb@pagecount=\the\cb@pagecount}%
884 }
885 \let\@makecol\cb@makecol

```

When  $\LaTeX$  makes a page with only floats it doesn't use `\@makecol`; instead it calls `\@vtryfc`, so we have to modify this macro as well. In twocolumn mode we must write either `\cb@firstcolumntrue` or `\cb@firstcolumnfalse` to the `.aux` file.

```

886 \let\ltx@vtryfc\@vtryfc
887 \def\cb@vtryfc#1{%
888   \cb@trace{In vtryfc, page \the\c@page,
889         \string\cb@pagecount=\the\cb@pagecount}%
890   \let\cb@writeAux\@gobble

```

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

```

891 \cb@checkpagecount
892 \ltx@vtryfc{#1}%
893 \if@twocolumn
894   \global\setbox\@outputbox \vbox to\@colht{%
895     \if@firstcolumn\write\@auxout{\string\cb@firstcolumntrue}%

```

```

896     \else\write\@auxout{\string\cb@firstcolumnfalse}%
897     \fi
898     \unvbox\@outputbox
899     \boxmaxdepth\maxdepth
900   }%
901 \fi
902 \global\advance\cb@pagecount by \@ne
903 }
904 \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).

```

```

905 \def\cb@processActive{%
906   \cb@pop\cb@endstack
907   \ifnum\cb@toleft=\cb@nil
908   \else
909     \setbox\@cclv\vbox{%
910       \unvbox\@cclv
911       \boxmaxdepth\maxdepth
912       \advance\cb@pagecount by -1\relax
913       \cb@setEndPoints}%
914     \cb@push\cb@historystack
915     \cb@push\cb@beginstack
916     \expandafter\cb@processActive
917   \fi}

```

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

```

918 \def\cb@startSpanBars{%
919   \cb@pop\cb@beginstack
920   \ifnum\cb@toleft=\cb@nil
921   \else
922     \cb@setBeginPoints
923     \cb@trace@stack{after StartSpanBars, page \the\c@page}%
924     \expandafter\cb@startSpanBars
925   \fi
926 }

```

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

```

927 \def\cb@buildstack{%
928   \cb@initstack\cb@endstack
929   \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 is 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.

```

930 \def\cb@pushNextActive{%
931   \cb@pop\cb@historystack
932   \ifnum\cb@topleft=\cb@nil
933   \else
934     \ifnum\cb@page>\cb@pagecount
935     \cb@push\cb@historystack
936   \else
937     \ifodd\cb@topleft
938     \cb@push\cb@endstack
939   \else
940     \cb@pop\cb@endstack
941   \fi
942   \expandafter\expandafter\expandafter\cb@pushNextActive
943   \fi
944 \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 `xxxnyyppzzz` 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.
945 \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 946 \cb@initstack\cb@historystack

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

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

\cb@endstack Allocate two stacks for the bars that span the current page break.
\cb@beginstack 950 \cb@initstack\cb@endstack
951 \cb@initstack\cb@beginstack

\cb@tempstack Allocate a stack for temporary storage
952 \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.
953 \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.

954 \def\cb@thehistorystack{\cb@historystack}
955 \def\cb@pop#1{%
956   \ifx #1\@empty
957     \def\cb@temp{#1}%
958     \ifx\cb@temp\cb@thehistorystack
959       \ifeof\cb@read
960         \else
961           {\endlinechar=-1\read\cb@read to\cb@temp
962           \xdef\cb@historystack{\cb@historystack\cb@temp}%
963           }%
964         \fi
965       \fi
966     \fi
967     \ifx#1\@empty
968       \global\cb@topleft\cb@nil

```

```

969   \global\cb@page\z@\relax
970   \else
971     \expandafter\cb@carcdr#1e#1%
972   \fi
973   \cb@trace@pop{#1}}

```

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

```

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

```

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

```

979 \def\cb@push#1{%
980   \xdef#1{\the\cb@topleft n\the\cb@page p\the\cb@curbarwd l#1}%
981   \cb@trace@push{#1}}
982

```

`\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.

```

983 \def\cb@barpoint#1#2#3{\cb@canta=#2
984   \ifcb@firstcolumn\advance\cb@canta by\m@ne\fi
985   \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, L<sup>A</sup>T<sub>E</sub>X 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<sup>A</sup>T<sub>E</sub>X begin-document hook.

```

986 \AtBeginDocument{%
987   \cb@setup@specials

   Add a sentinel to \cb@pagejumplst.
988   \cb@pagejump{999999999,}%

   Compute the left and right positions of the changebars.
989   \cb@positions
990   \cb@trace{%
991     Odd left : \the\cb@odd@left\space
992     Odd right : \the\cb@odd@right\MessageBreak
993     Even left: \the\cb@even@left\space
994     Even right: \the\cb@even@right
995   }%
996   \immediate\closeout\cb@write
997   \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<sup>A</sup>T<sub>E</sub>X 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.

```

998 \AtEndDocument{%
999   \clearpage
1000  \cb@initstack\cb@historystack
1001  \immediate\closein\cb@read
1002  \immediate\openin\cb@read=\jobname.cb\relax
      Let \cb@pdfxy=\cb@checkPdfxy for checking. Make \cb@pagejump dummy.
1003  \ifx\cb@readxy\undefined
1004  \else
1005    \immediate\closein\cb@readxy
1006    \immediate\openin\cb@readxy=\jobname.cb2\relax
1007    \def\cb@pdfpoints{}%
1008    \def\cb@pdfpagenr{0}%
1009  \fi
1010  \@cb@firstcolumnfalse
1011  \cb@checkrerun
1012  \let\cb@pdfxy\cb@checkPdfxy
1013  \let\cb@pagejump\@gobble
1014  \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.

```

1015 \def\cb@checkHistory#1#2#3{%
1016  \cb@pop\cb@historystack
1017  \ifnum #1=\cb@topleft\relax
1018    \cb@cnta=#2
1019    \ifcb@firstcolumn\advance\cb@cnta by\m@ne\fi
1020    \ifnum \cb@cnta=\cb@page\relax
      Both page and point numbers are equal; do nothing,
1021    \else
      but generate a warning when page numbers don't match, or
1022      \cb@error
1023      \fi
1024    \else
      when point numbers don't match.
1025      \cb@error
1026      \fi}
      Dummy definition for \cb@checkPdfxy. This will be overwritten by the pdftex
      and xetex options.
1027 \def\cb@checkPdfxy#1#2#3#4#5{}

```

`\cb@rerun` The macro `\cb@rerun` is called when we detect that we need to rerun L<sup>A</sup>T<sub>E</sub>X.

```
1028 \def\cb@rerun{%
1029   \global\let\cb@checkrerun\cb@error}
1030 \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 pdf<sub>T</sub>E<sub>X</sub> and Xe<sub>T</sub>E<sub>X</sub> we also disable `\cb@checkPdfxy`.

```
1031 \def\cb@error{%
1032   \PackageWarning{Changebar}%
1033     {Changebar info has changed.\MessageBreak
1034     Rerun to get the bars right}
1035   \gdef\cb@checkHistory##1##2##3{%
1036     \let\cb@barpoint\cb@checkHistory
1037     \gdef\cb@checkPdfxy##1##2##3##4##5{%
1038       \let\cb@pdfxy\cb@checkPdfxy}
```

## 5.8 Macros For Making It Work With Nested Floats/Footnotes

`\end@float` This is a replacement for the L<sup>A</sup>T<sub>E</sub>X-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 L<sup>A</sup>T<sub>E</sub>X `\end@float`.

```
1039 \let\ltx@end@float\end@float
1040 \def\cb@end@float{%
1041   \cb@trace@stack{end float on page \the\c@page}%
1042   \cb@pop\cb@currentstack
1043   \ifnum\cb@topleft=\cb@nil
1044   \else
1045     \cb@push\cb@currentstack
1046     \global\cb@curbarwd=\cb@curbarwd
1047     \@endfloatbox
1048     \global\setbox\@currbox
1049       \color@vbox
1050       \normalcolor
1051       \vbox\bgroup\cb@start[\cb@curbarwd]\unvbox\@currbox\cb@end
1052   \fi
1053   \ltx@end@float}
1054 \let\end@float\cb@end@float
```

This only works if this new version of `\end@float` is really used. With L<sup>A</sup>T<sub>E</sub>X 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<sup>A</sup>T<sub>E</sub>X 2<sub>ε</sub> 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`.

```
1055 \let\ltx@xympar\@xympar
1056 \def\@xympar{%
1057   \let\end@float\ltx@end@float
1058   \ltx@xympar
1059   \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`.

```
1060 \let\flt@float@end\float@end
```

Then we redefine it to insert the changebarcode.

```
1061 \def\float@end{%
1062   \cb@trace@stack{end float on page \the\c@page}%
1063   \cb@pop\cb@currentstack
1064   \ifnum\cb@topleft=\cb@nil
1065   \else
1066     \cb@push\cb@currentstack
1067     \global\cb@curbarwd\cb@curbarwd
1068     \@endfloatbox
1069     \global\setbox\@currbox
1070     \color@vbox
1071     \normalcolor
1072     \vbox\bgroup\cb@start[\cb@curbarwd]\unvbox\@currbox\cb@end
1073   \fi
1074   \let\end@float\ltx@end@float
1075   \flt@float@end
1076 }
```

`\end@dblfloat` This is a replacement for the  $\text{\LaTeX}$ -macro of the same name. All it does is check to see if changebars are active and, if so, it puts changebars around the box containing the float. In this case the  $\text{\LaTeX}$  macro had to be rewritten.

```
1077 \let\ltx@end@dblfloat\end@dblfloat
1078 \def\cb@end@dblfloat{%
1079   \if@twocolumn
1080     \cb@trace@stack{end dblfloat on page \the\c@page}%
1081     \cb@pop\cb@currentstack
1082     \ifnum\cb@topleft=\cb@nil
1083     \else
1084       \cb@push\cb@currentstack
1085       \global\cb@curbarwd=\cb@curbarwd
1086       \@endfloatbox
1087       \global\setbox\@currbox
```

```

1088     \color@vbox
1089     \normalcolor
1090     \vbox\bgroup\cb@start[\cb@curbarwd]\unvbox\@currbox\cb@end
1091   \fi
1092   \@endfloatbox
1093   \ifnum\@floatpenalty <\z@
1094     \@largefloatcheck
1095     \@cons\@dbldeferlist\@currbox
1096   \fi
1097   \ifnum \@floatpenalty =-\@Mii \@Esphack\fi
1098 \else
1099   \end@float
1100 \fi}
1101 \let\end@dblfloat\cb@end@dblfloat

```

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

```

1102 \let\flt@float@dblend\float@dblend
1103 \def\float@dblend{%
1104   \cb@trace@stack{end dbl float on page \the\c@page}%
1105   \cb@pop\cb@currentstack
1106   \ifnum\cb@topleft=\cb@nil
1107   \else
1108     \cb@push\cb@currentstack
1109     \global\cb@curbarwd=\cb@curbarwd
1110     \@endfloatbox
1111     \global\setbox\@currbox
1112       \color@vbox
1113       \normalcolor
1114       \vbox\bgroup\cb@start[\cb@curbarwd]\unvbox\@currbox\cb@end
1115   \fi
1116   \let\end@dblfloat\ltx@end@dblfloat
1117   \flt@float@dblend
1118 }

```

`\@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.

```

1119 \let\ltx@footnotetext\@footnotetext
1120 \long\def\cb@footnotetext#1{%
1121   \cb@trace@stack{end footnote on page \the\c@page}%
1122   \cb@pop\cb@currentstack
1123   \ifnum\cb@topleft=\cb@nil
1124     \ltx@footnotetext{#1}%
1125   \else
1126     \cb@push\cb@currentstack
1127     \edef\cb@temp{\the\cb@curbarwd}%
1128     \ltx@footnotetext{\cb@start[\cb@temp]#1\cb@end}%
1129   \fi}

```

```

1130 \let\@footnotetext\cb@footnotetext

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

1131 \let\ltx@mpfootnotetext\@mpfootnotetext
1132 \long\def\cb@mpfootnotetext#1{%
1133   \cb@pop\cb@currentstack
1134   \ifnum\cb@topleft=\cb@nil
1135     \ltx@mpfootnotetext{#1}%
1136   \else
1137     \cb@push\cb@currentstack
1138     \edef\cb@temp{\the\cb@curbarwd}%
1139     \ltx@mpfootnotetext{\cb@start[\cb@temp]#1\cb@end}%
1140   \fi}
1141 \let\@mpfootnotetext\cb@mpfootnotetext
1142 </package>

```