The Changebar package *

Michael Fine
Distributed Systems Architecture

Johannes Braams
texniek at braams.xs4all.nl

Printed March 11, 2018

Contents

1 Introduction 1

2 The user interface 2

2.1 The package options 2

2.1.1 Specifying the printer driver 2

2.1.2 Specifying the bar position 3

2.1.3 Color 3

2.1.4 Tracing 3

2.2 Macros defined by the package 3

2.3 Changebar parameters 3

3 Deficiencies and bugs 4

4 The basic algorithm 5

5 The implementation 6

5.1 Declarations And Initializations 6

5.2 Option Processing 8

5.3 User Level Commands And Parameters 12

5.4 Macros for beginning and ending bars 26

5.5 Macros for Making It Work Across Page Breaks 31

5.6 Macros For Managing The Stacks of Bar points 35

5.7 Macros For Checking That The .aux File Is Stable 37

5.8 Macros For Making It Work With Nested Floats/Footnotes 39

Abstract

This package implements a way to indicate modifications in a \LaTeX-document by putting bars in the margin. It realizes this by making use of the \texttt{special} commands supported by ‘dvi drivers’. Currently six different drivers are supported, plus \texttt{pdftex} and \texttt{Xe\LaTeX} 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 v3.6c, last revised 2018/03/09.
• Changebars may be nested within each other. Each level of nesting can be given a different thickness bar.

• Changebars may be nested in other environments including floats and footnotes.

• Changebars are applied to all the material within the “barred” environment, including floating bodies regardless of where the floats float to. An exception to this is margin floats.

• Changebars may cross page boundaries.

• Changebars can appear on the outside of the columns of two column 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\footnote{For older documents the command \texttt{\textbackslash driver} is available in the preamble of the document. It takes the options as defined for \texttt{\textbackslash bf TeX} as argument.} specify the driver that will be used to print the document can be indicated. The driver may be one of:

• DVItoLN03
• DVItoPS
• DVIps
• emTeX
• TeXtures
• VTeX
• PDFTeX
• XeTeX

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 \texttt{\textbackslash 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 colourful the options `color` and `xcolor` are available. They define the user command `\cbcolor` and load either the `color` or the `xcolor` package.

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

2.1.4 Tracing

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

2.2 Macros defined by the package

`\cbstart` All material between the macros `\cbstart` and `\cbend` is barred. The nesting of multiple changebars is allowed. The macro `\cbstart` has an optional parameter that specifies the width of the bar. The syntax is `\cbstart[⟨dimension⟩]`. If no width is specified, the current value of the parameter `\changebarwidth` is used.

Note that `\cbstart` and `\cbend` can be used anywhere but must be correctly nested with floats and footnotes. That is, one cannot have one end of the bar inside a floating insertion and the other outside, but that would be a meaningless thing to do anyhow.

`\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

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

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

3 Deficiencies and bugs

- The macros blindly use special points \cb@minpoint through \cb@maxpoint. If this conflicts with another set of macros, the results will be unpredictable. (What is really needed is a \newspecialpoint, analogous to \newcount etc. — it’s not provided because the use of the points is rather rare.)

- There is a limit of (\cb@maxpoint – \cb@minpoint +1)/4 bars per page (four special points per bar). Using more than this number yields unpredictable results (but that could be called a feature for a page with so many bars). This limitation could be increased if desired. There is no such limit with PDFTeX or XeTeX.

- Internal macro names are all of the form \cb@xxxx. No checking for conflicts with other macros is done.

- This implementation does not work with the multicolumn package.

- The algorithms may fail if a floating insertion is split over multiple pages. In \LaTeX floats are not split but footnotes may be. The simplest fix to this is to prevent footnotes from being split but this may make TeX very unhappy.

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

- The colour support has only been tested with the \dvips and \pdftex drivers.
4 The basic algorithm

The changebars are implemented using the `\specials` of various dvi interpreting programs like DVItoLN03 or DVIPs. In essence, the start of a changebar defines two `\special` points in the margins at the current vertical position on the page. The end of a changebar defines another set of two points and then joins (using the “connect” `\special`) either the two points to the left or the two points to the right of the text, depending on the setting of `innerbars`, `outerbars`, `leftbars`, `rightbars` and/or `twoside`.

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

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

```
blah blah blah % page n
blah blah blah
\cbstart % this does its thing and set the flag
more blah
\cbend % does its thing and unsets flag
more 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 \예TeX\ anyway. With PDF\예TeX\ 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 \예TeX\ macros check for this condition and add the extra bar.
5 The implementation

5.1 Declarations And Initializations

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}

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}

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}

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

These four counters are used to identify the four special points that specify a 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

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

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

14 \newdimen\cb@curbarwd

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 pagernumber 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
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.

\def\cb@barsplace{0} \cb@trace

A switch to enable tracing of the actions of this package.

\newif\if@cb@trace

\cb@firstcolumn

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

\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.

\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.

\newdimen\cb@odd@left
\newdimen\cb@odd@right
\newdimen\cb@even@left
\newdimen\cb@even@right

Since the changebars are drawn with the PostScript command lineto and not as \TeX-like rules the reference points lie on the center of the changebar, therefore the calculation has to add or subtract half of the width of the bar to keep \changebarsep whitespace between the bar and the body text.

First the position for odd pages is calculated.

\def\cb@positions{%
\global\cb@odd@left=\hoffset
\global\cb@even@left=\cb@odd@left
\global\advance\cb@odd@left by \oddsidemargin
\global\cb@odd@right=\cb@odd@left
\global\advance\cb@odd@right by \textwidth
\global\advance\cb@odd@right by \changebarsep
\global\advance\cb@odd@left by -\changebarsep
\global\advance\cb@odd@right by 0.5\changebarwidth
\global\advance\cb@odd@left by -0.5\changebarwidth

\global\advance\cb@even@right by \changebarsep
\global\advance\cb@even@left by -\changebarsep
\global\advance\cb@even@right by 0.5\changebarwidth
\global\advance\cb@even@left by -0.5\changebarwidth

\global\advance\cb@odd@right by \changebarsep
\global\advance\cb@odd@left by -\changebarsep
\global\advance\cb@odd@right by 0.5\changebarwidth
\global\advance\cb@odd@left by -0.5\changebarwidth
\global\advance\cb@odd@right by \changebarsep
\global\advance\cb@odd@left by -\changebarsep
\global\advance\cb@odd@right by 0.5\changebarwidth
\global\advance\cb@odd@left by -0.5\changebarwidth
On even sided pages we need to use \texttt{\evensidemargin} in the calculations when twoside is in effect.

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

\texttt{\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 \texttt{\the\langle\texttt{dimen}\rangle}. This can be done by defining a command that has a delimited argument like:

```latex
\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 \texttt{\the\langle\texttt{dimen}\rangle}. Thus the characters that delimit the argument of \texttt{\cb@removedim} also have to have category code 12. To keep the changes local the macro \texttt{\cb@removedim} is defined in a group.

```latex
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 \texttt{\usepackage} call. Possible choices are:

- DVItoLN03
- DVItoPS
- DVIPS
- em\TeX
- Textures
- VT\TeX
- PDF\TeX
- Xe\TeX
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.
\begin{verbatim}
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}
\end{verbatim}

For the \texttt{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.
\begin{verbatim}
64 \def\cb@pdftexcheck{%
65 \ifxsame{\pdfoutput}{-1} \cb@pdftexerror
66 \else
67 \ifnum\pdfoutput=0 \cb@pdftexerror
68 \else
69 \global\chardef\cb@driver@setup=6\relax
70 \ifx\cb@writexy\@undefined
71 \newwrite\cb@writexy
72 \newread\cb@readxy
73 \immediate\openout\cb@writexy=\jobname.cb2\relax
74 \fi
75 \gdef\cb@pdfxy##1##2##3##4##5{%
76 \immediate\write\cb@writexy{##1.##2p##3,##4,##5}%
77 \expandafter\gdef\csname cb@##1.##2\endcsname{##3,##4,##5}}
78 \else \cb@pdftexerror\fi\fi
\end{verbatim}

Give a warning if we cannot support the \texttt{pdftex} option.
\begin{verbatim}
79 \def\cb@pdftexerror{\PackageError
80 \{changebar\}%
81 \{PDF\TeX{} option cannot be used\}%
82 \{You are using a \LaTeX{} run which does not generate PDF\MessageBreak
83 or you are using a very old version of PDFTeX\}}
\end{verbatim}

For the \texttt{xetex} option we have to check that the current \TeX{} run is using Xe\TeX{}. If it is, we initialize the option and open an additional output file. If not, we ignore the option and issue a warning.
\begin{verbatim}
84 \def\cb@xetexcheck{%
85 \expandafter\ifx\csname XeTeXrevision\endcsname\@undefined \cb@xetexerror
86 \else
87 \global\chardef\cb@driver@setup=7\relax
88 \fi
\end{verbatim}
Redefine the \cb@pdfxy macro to write point coordinates to the .cb2 file.

\gdef\cb@pdfxy##1##2##3##4##5{\immediate\write\cb@writexy{##1.##2p##3,##4,##5}\expandafter\gdef\csname cb@##1.##2\endcsname{##3,##4,##5}}

Give a warning if we cannot support the xetex option.

\gdef\cb@xetexerror\PackageError{changebar}{\textit{XeTeX option cannot be used}}{You are not using XeLaTeX}

The new features of \LaTeXe make it possible to implement the \textit{outerbars} option.

\DeclareOption{outerbars}{\def\cb@barsplace{1}}
\DeclareOption{innerbars}{\def\cb@barsplace{0}}

It is also possible to specify that the change bars should \textit{always} be printed on either the left or the right side of the text. For this we have the options \textit{leftbars} and \textit{rightbars}. Specifying \textit{either} of these options will overrule a possible \textit{twoside} option at the document level.

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

A set of options to control tracing.

\DeclareOption{traceon}{\@cb@tracetrue}
\DeclareOption{traceoff}{\@cb@tracefalse}
\DeclareOption{tracestacks}{\let\cb@trace@stack\cb@@show@stack
\def\cb@trace@push#1{(\cb@trace{\textbf{Pushed point }\the\cb@topleft\space on \noexpand#1: \noexpand#1}}
\def\cb@trace@pop#1{(\cb@trace{\textbf{Popped point }\the\cb@topleft\space from \noexpand#1: \noexpand#1}}
}

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

\DeclareOption{grey}{\def\cb@ps@color{\thechangebargrey\space 100 div setgray}}

The second option activates support for the \textit{color} package.

\DeclareOption{color}{\def\cb@ps@color{\expandafter\color@to@ps\cb@current@color@\@0}}
\def\cb@color@pkg{color}

The third option adds support for the \textit{xcolor} package.

\DeclareOption{xcolor}{\def\cb@ps@color{\expandafter\color@to@ps\cb@current@color@\@0}}
\def\cb@color@pkg{xcolor}
Signal an error if an unknown option was specified.
\DeclareOption*{\OptionNotUsed\PackageError
\{\Unrecognised option \CurrentOption\MessageBreak
known options are dvit0n03, dvitops, dvips,\MessageBreak
textures, pdftex, vtex and xetex,\MessageBreak
grey, color, xcolor,\MessageBreak
outerbars, innerbars, leftbars and rightbars\}\}}

The default is to have grey change bars on the left side of the text on odd pages. When VT\TeX{} is used the option dvips is not the right one, so in that case we have vtex as the default driver. When PDFTeX is producing PDF output, the pdftex option is selected.
\Ifx\VTeXversion\@undefined
\Ifx\csname XeTeXrevision\endcsname\@undefined
\Ifx\pdfoutput\@undefined
\ExecuteOptions{innerbars,traceoff,dvips,grey}
\else
\ifnum\pdfoutput>0
\ExecuteOptions{innerbars,traceoff,pdftex,grey}
\else
\ExecuteOptions{innerbars,traceoff,dvips,grey}
\fi
\else
\ExecuteOptions{innerbars,traceoff,xetex,grey}
\else
\ExecuteOptions{innerbars,traceoff,vtex,grey}
\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.
\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.
\def\cb@@show@stack#1{%
\cb@trace{%
stack status at #1:\MessageBreak
current stack: \cb@currentstack\MessageBreak
\@spaces end stack: \cb@endstack\MessageBreak
\space\space begin stack: \cb@beginstack\MessageBreak
history stack: \cb@historystack
}
}

The default is to not trace the stacks. This is achieved by \let\cb@trace@stack to \@gobble.
\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 \texttt{tracestacks} is selected.

The default is to \textit{not} trace the stacks.

\begin{verbatim}
158 \let\cb@trace@push\@gobble
159 \let\cb@trace@pop\@gobble
\end{verbatim}

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

\begin{verbatim}
160 \ProcessOptions\relax
\end{verbatim}

We have to make sure that when the document is being processed by \texttt{pdf\LaTeX}, while also creating pdf as output, the driver to be used is the pdf driver. Therefore we add an extra check, possibly overriding a \texttt{dvips} option that might still have been in the document.

\begin{verbatim}
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
\end{verbatim}

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

\begin{verbatim}
170 \newcommand{\cb@trace}{\%}
171 \if\cb@trace
172 \GenericWarning
173 \{\texttt{changebar}\@spaces\@spaces\%
174 \{\texttt{Package changebar: \#1}\@gobble\%
175 \fi
176 }
\end{verbatim}

\subsection*{5.3 User Level Commands And Parameters}

\driver The user can select the specials that should be used by calling the command \texttt{\driver{\textit{drivername}}}.

Possible choices are:

\begin{itemize}
  \item DVItoLN03
  \item DVItoPS
  \item DVIps
  \item em\LaTeX
  \item \TeX\types
  \item V\TeX
  \item PDFTeX
  \item Xe\TeX
\end{itemize}
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.

\if@compatibility
  \def\driver#1{%
    \bgroup\edef\next{\def\noexpand\tempa{#1}}%
    \uppercase\expandafter{\next}%
    \def\LN{DVITOLN03}%
    \def\DVItoPS{DVITOPS}%
    \def\DVIPS{DVIPS}%
    \def\emTeX{EMTEX}%
    \def\Textures{TEXTURES}%
    \def\VTeX{VTEX}%
    \def\pdfTeX{PDFTEX}%
    \def\xeTeX{xetex}%
    \cb@driver@setup=0\relax
    \ifx\tempa\LN \cb@driver@setup=0\fi
    \ifx\tempa\DVItoPS \cb@driver@setup=1\fi
    \ifx\tempa\DVIPS \cb@driver@setup=2\fi
    \ifx\tempa\emTeX \cb@driver@setup=3\fi
    \ifx\tempa\Textures \cb@driver@setup=4\fi
    \ifx\tempa\VTeX \cb@driver@setup=5\fi
    \ifx\tempa\pdfTeX \cb@pdftexcheck\fi
    \ifx\tempa\xeTeX \cb@xetexcheck\fi
  \egroup}

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

\global\chardef\cb@driver@setup=0\relax
\ifx\tempa\LN \global\chardef\cb@driver@setup=0\fi
\ifx\tempa\DVItoPS \global\chardef\cb@driver@setup=1\fi
\ifx\tempa\DVIPS \global\chardef\cb@driver@setup=2\fi
\ifx\tempa\emTeX \global\chardef\cb@driver@setup=3\fi
\ifx\tempa\Textures \global\chardef\cb@driver@setup=4\fi
\ifx\tempa\VTeX \global\chardef\cb@driver@setup=5\fi
\ifx\tempa\pdfTeX \cb@pdftexcheck\fi
\ifx\tempa\xeTeX \cb@xetexcheck\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.

\cb@trace@connect When tracing is on, write information about the points being connected to the log file.
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)}.

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.

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.

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.

The first extension to the changebar package was for the \special syntax of the program DVItoPS by James Clark.
The program DVIps by Thomas Rokicki is also supported. The PostScript code is nearly the same as for DVItoPS, but the coordinate space has a different dimension. Also this code has been made resolution independent, whereas the code for DVItoPS might still be resolution dependent.

So far all the positions have been calculated in pt units. DVIps uses pixels internally, so we have to convert pts into pixels which of course is done by dividing by 72.27 (pts per inch) and multiplying by \texttt{Resolution} giving the resolution of the PostScript device in use as a PostScript variable.

\begin{verbatim}
\def\cb@defpoint##1##2{% 
  \special{ps:
    \expandafter\cb@removedim\the##2\space
    Resolution\space mul\space 72.27\space div\space
    \space exch def \space currentpoint \space exch pop
    \space exch def}
  \cb@trace@defpoint##1##2}
\end{verbatim}

The following addition is for the drivers written by Eberhard Mattes. The \texttt{\special} syntax used here is supported since version 1.5 of his driver programs.

\begin{verbatim}
\def\cb@defpoint##1##2{% 
  \special{em:point \the##1,\the##2}%
  \cb@trace@defpoint##1##2}
\end{verbatim}

The following definitions are validated with \TeXtures version 1.7.7, but will very likely also work with later releases of \TeXtures.

The \texttt{\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, (semicircular ends) in which case a filled circle is shown for such lines.

\begin{verbatim}
\def\cb@defpoint##1##2{% 
  \special{postscript 0 0 transform}% leave \[x,y\] on the stack
  \special{rawpostscript
    \expandafter\cb@removedim\the##2\space
    \space exch def \space currentpoint \space exch pop
    \space exch def}
  \cb@trace@defpoint##1##2\fi}
\end{verbatim}
The following definitions were kindly provided by Michael Vulis.

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

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.

The `\cb@findpdfpoint` macro finds the coordinates of point #1 on pagecount #2. First we expand the arguments to get the real values.
The \cb@findpdfpoint macro finds the coordinates of point \#1 on pagecount \#2. If the information is not yet in memory it is 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.

\begin{verbatim}
def\cb@findpdfpoint##1##2{%  \ifnum##2<\cb@pdfpagenr\relax\else\cb@pdfreadxy{##2}\fi\let\cb@pdfx\relax\ifx\cb@pdfpoints\@empty\else\ifnum##2<0\relax\else\edef\cb@temp{{\noexpand\cb@pdffind{##1}{##2}\cb@pdfpoints\relax{}}}%\cb@temp\fi\fi\fi}
\end{verbatim}

\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 . \langle p(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.

\begin{verbatim}
def\cb@pdffind##1##2##3.##4p##5,##6,##7pt##8\relax##9{%  \def\cb@next{{\cb@pdffind{##1}{##2}##8\relax{##9##3.##4p##5,##6,##7pt}}}%  \ifnum##1=##3\ifnum##2=##4\def\cb@pdfx{##5sp}%\def\cb@pdfy{##6sp}%\def\cb@pdfz{##7pt}%\let\cb@next\relax\gdef\cb@pdfpoints{##9##8}\fi\fi\ifx\relax##8\relax\let\cb@next\relax\fi\cb@next}%
\end{verbatim}

\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.

\begin{verbatim}
def\cb@pdfreadxy#1{%  \let\cb@next\relax\ifeof\cb@readxy\global\let\cb@pdfpagenr\cb@maxpoint\else\ifnum\cb@pdfpg<0\let\cb@next\relax\gdef\cb@pdfpoints{##9##8}\fi\fi%\endlinechar=-1\read\cb@readxy to\cb@temp\ifx\cb@temp\@empty\else\expandafter\cb@pdfparsexy\cb@temp\fi\fi\let\cb@next\relax%\else\endlinechar=-1\read\cb@readxy to\cb@temp\ifx\cb@temp\@empty\else\expandafter\cb@pdfparsexy\cb@temp\fi\fi\fi%\else\endlinechar=-1\read\cb@readxy to\cb@temp\ifx\cb@temp\@empty\else\expandafter\cb@pdfparsexy\cb@temp\fi\fi\fi\fi}\
\end{verbatim}
\texttt{\textbackslash cb@pdfparsexy} The \texttt{\textbackslash cb@pdfparsexy} macro extracts the pagecount from an entry read in from the .cb2 file.

\begin{verbatim}
\def\cb@pdfparsexy##1.##2p##3,##4,##5pt{\def\cb@pdfpg{##2}}
\end{verbatim}

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 \texttt{\textbackslash cb@pagecount} here, not during the write.

\begin{verbatim}
\def\cb@defpoint##1##2{\if@filesw\begingroup\edef\point{{\the##1}{\the\cb@pagecount}}\let\the=\z@\pdfsavepos\edef\cb@temp{\write\@auxout\string\cb@pdfxy\point{\the\pdflastxpos}{\the\pdflastypos}{\the##2}}}\cb@temp\endgroup\fi\cb@trace@defpoint##1##2%}
\end{verbatim}

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

\begin{verbatim}
\def\cb@cvtpct##1{\ifnum##1<0 0\else\ifnum##1>99 1\else\ifnum##1<10 0.0\the##1\else 0.\the##1\fi\fi\fi}
\end{verbatim}

The \texttt{\textbackslash cb@connect} finds the coordinates of the begin and end points, converts them to PDF units and draws the bar with \texttt{\textbackslash 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.

\begin{verbatim}
\def\cb@connect##1##2##3{\cb@findpdfpoint{##1}\cb@pagecount\ifx\cb@pdfx\relax\cb@rerun\fi\edef\point{{\the\cb@pagecount}{{\the\cb@pagecount}}\let\the=\z@\pdfsavepos\edef\cb@temp{\write\@auxout\string\cb@pdfxy\point{\the\pdflastxpos}{\the\pdflastypos}{\the##2}}}\cb@temp\endgroup\cb@trace@defpoint##1##2\}
\end{verbatim}

18
We do everything in a group, so that we can freely use all kinds of registers.

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.

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

\cb@checkPdfxy The macro \cb@checkPdfxy checks if the coordinates of a point have changed during the current run. If so, we need to rerun \LaTeX.
For PDFTEX we don’t need a limit on the number of bar points.
\%\begin{macrocode}
\ifdim##5=\cb@pdfz\relax
\else
\cb@error
\fi
\else
\cb@error
\fi
\else
\cb@error
\fi
\end{changebar}
\% \begin{macrocode}
def\cb@maxpoint{9999999}
def\cb@resetpoints{\relax}
or
The code for XeTEX is, like for PDFTEX, more elaborate as the calculations
have to be done in T\TeX. \cb@defpoint will write information about the coordi-
nates 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.
\immediate\closeout\cb@writexy
\immediate\openin\cb@readxy=\jobname.cb2\relax
\cb@pdfpoints \cb@pdfpagenr
The \cb@pdfpoints macro contains the list of coordinates of points that have
been read in memory from the .cb2 file. The \cb@pdfpagenr macro contains the
next pagecount to be read in.
def\cb@pdfpoints{}
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.
def\cb@findpdfpoint##1##2{%
def\cb@temp
\cb@temp
}
\pdfliteral
For XeTEX we mimick PDFTEX’s command \pdfliteral.
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.
def\cb@@findpdfpoint##1##2{%
\ifnum##2<\cb@pdfpagenr\relax\else
\cb@pdfreadxy{##2}%
\fi
\let\cb@pdfx\relax
20
The \texttt{cb@pdffind} recursively searches through \texttt{cb@pdfpoints} to find point \texttt{#1} on pagecount \texttt{#2}. \texttt{cb@pdfpoints} contains entries of the form \texttt{\langle pointnr \rangle\langle pagecount \rangle\langle x \rangle, \langle y \rangle, \langle z \rangle pt}. When the point is found it is removed from \texttt{cb@pdfpoints}. \texttt{#9} contains the cumulative head of the list to construct the new list with the entry removed. \texttt{#3-#8} are for pattern matching.

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

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 \texttt{\cb@pagecount} here, not during the write.

\texttt{\cb@connect} finds the coordinates of the begin and end points, converts them to PDF units and draws the bar with \texttt{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.

The \texttt{\cb@connect} finds the coordinates of the begin and end points, converts them to PDF units and draws the bar with \texttt{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.
We do everything in a group, so that we can freely use all kinds of registers.

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.

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

The macro \cb@checkPdfxy checks if the coordinates of a point have changed during the current run. If so, we need to rerun \LaTeXX.
For Xe\TeX{} we don’t need a limit on the number of bar points.
\def\cb@maxpoint{9999999}
\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.
\else
\PackageWarning{Changebar}{changebars not supported in unknown setup}
\def\cb@defpoint##1##2{\cb@trace@defpoint##1##2}
\def\cb@connect##1##2##3{\cb@trace@connect##1##2##3}
\let\cb@resetpoints\relax
\fi

The last thing to do is to forget about $\cb@setup@specials$.
\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
$\changebarwidth$.
\newcommand*{\cbstart}{\@ifnextchar [\%}{\cb@start}\
{\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.
\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 $\deletebarwidth$.
\newcommand*{\cbdelete}{\@ifnextchar [\%}{\cb@delete}\
{\cb@delete[\deletebarwidth]}}
\cbdelete Deletebars are implemented as a special ‘change bar’. The bar is started and
immediately ended. It is as long as it is wide.
\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 \verb|tabular| and \verb|tabbing| environments.
\newenvironment{changebar}{\cbstart}{\cb@end}
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.

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

\changebarwidth The default width of the changebars is stored in the dimension register \changebarwidth.
\newlength{\changebarwidth} \setlength{\changebarwidth}{2pt}

\deletebarwidth The default width of the deletebars is stored in the dimension register \deletebarwidth.
\newlength{\deletebarwidth} \setlength{\deletebarwidth}{4pt}

\changebarsep The default separation between all bars and the text is stored in the dimension register \changebarsep.
\newlength{\changebarsep} \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).
\newcounter{changebargrey} \setcounter{changebargrey}{65}

When one of the options color or xcolor was selected we need to load the appropriate package. When we’re run by pdfLaTeX we need to pass that information on to that package.
\@ifpackagewith{changebar}{\csname cb@color@pkg\endcsname}{% \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.
\DeclareRobustCommand{\cbcolor}{% \undeclaredcbcolor \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 LaTeX 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.
\cbcolor{\&undeclaredcbcolor}}
\undeclaredcbcolor Call the driver-dependent command \color\{model\} to define \cb@current@color.

\@undeclaredcbcolor
\def\@undeclaredcbcolor[#1]#2{%}
\begingroup
\color[#1]{#2}%
\global\let\cb@current@color\current@color
\endgroup
\ignorespaces
}

\@declaredcbcolor
\def\@declaredcbcolor#1{%}
\begingroup
\color{#1}%
\global\let\cb@current@color\current@color
\endgroup
\ignorespaces}%

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

\def\cbcolor{\@ifnextchar\[%\]
\@@cbcolor\@cbcolor}%
\def\@@cbcolor[#1]#2{\cb@colwarn\def\@@cbcolor[#1]#2{}}%
\def\@cbcolor#1{\cb@colwarn\def\@cbcolor#1{}}%
\def\cb@colwarn{\PackageWarning{Changebar}%
{You didn’t specify the option `color';\MessageBreak your command \string\cbcolor space will be ignored}}%
}

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.

\def\cb@start[#1]{%}
\cb@topleft=\cb@nextpoint
Store the width of the current bar in \cb@curbarwd.
\cb@curbarwd=\relax
\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.

\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.

\cb@cnta=\cb@pagecount
\cb@pagecount=\cb@page\advance\cb@pagecount\m@ne
\ifvmode
\cb@setBeginPoints
\else
\vbox to \z@{%
When we are in horizontal mode we jump up a line to set the starting point of the changebar.

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

This macro ends a changebar. It pops the current point and nesting level off \texttt{cb@currentstack} and sets the end point by calling \texttt{cb@setEndPoints} with the parameter corresponding to the \texttt{beginning} point number. It writes the \texttt{.aux} file and joins the points. When in horizontal mode we put the call to \texttt{cb@setEndPoints} inside a \texttt{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.

The macro \texttt{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 \texttt{cb@topleft} and \texttt{cb@page}. The resulting page will be stored in \texttt{cb@page}. The parameter indicates whether we are searching for a begin point (0) or end point (3).


First store the identifiers in temporary registers.

Then pop the history stack.

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

Now keep popping the stack until \texttt{\textbackslash cb\textbackslash 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.

Now that we’ve found it overwrite \texttt{\textbackslash cb\textbackslash cntb} with the \texttt{\textbackslash cb\textbackslash page} from the stack.

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

Finally return the correct values

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

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.

\texttt{\textbackslash ifnum\textbackslash cb\textbackslash topleft=\textbackslash cb\textbackslash nil}
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{\noexpand\cb@connect{\cb@topleft}{\cb@botleft}{\cb@curbarwd}}%
In twocolumn pages always use outerbars

\if@twocolumn
  \ifodd\cb@pagecount\cb@rightbar\else\cb@leftbar\fi
\else
  \ifcase\cb@barsplace
    0=innerbars
      \ifodd\cb@cntb
        \cb@rightbar
      \else
        \if@twoside\cb@leftbar\else\cb@rightbar\fi
      \fi
    \or
    1=outerbars
      \ifodd\cb@cntb
        \cb@rightbar
      \else
        \if@twoside\cb@leftbar\else\cb@rightbar\fi
      \fi
    \or
    2=leftbars
      \cb@leftbar
    \or
    3=rightbars
      \cb@rightbar
  \fi
\fi
\fi

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

\def\cb@writeAux#1{%
  \if@files\%
    \begingroup
      \edef\point{\the#1}\%
      \edef\level{\the\cb@curbarwd}\%
      \let\the=\z@
      \edef\cb@temp{\write\@auxout}{\string\cb@barpoint{\point}{\the\cb@pagecount}{\the\level}}%
      \cb@temp
    \endgroup
  \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 \dimen0 \dp@leftcolumn
797 \unvbox@leftcolumn
798 \vskip-\dimen0
799 }%
800 }
\cb@poppagejump Pop an entry from pagejumplist. The entry is put in \cb@nextpagejump.
801 \def\cb@poppagejump#1,#2\relax{%
802 \gdef\cb@nextpagejump{#1}%
803 \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.
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 two-side 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 \texttt{\cb@pagecount} is incorrect there. Therefore we write a command in the \texttt{.aux} file so that in the next run the lefthand column will correct its \texttt{\cb@pagecount}. We also need to signal a rerun. If the correction was made in the lefthand column, the flag \texttt{@cb@pagejump} is set, and we have to be careful in the righthand column. If in the righthand column the flag is set and \texttt{\cb@pagecount} is correct, the correction in the lefthand column worked, but we still have to write into the \texttt{.aux} file for the next run. If on the other hand \texttt{\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.

\begin{verbatim}
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 \texttt{\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@pagejumpst\relax%
821 \ifnum\cb@pagecount=\cb@nextpagejump
822 \cb@trace{Page jump: \string\cb@pagecount=\the\cb@pagecount}
823 \global\cb@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 \if@cb@pagejump
830 \cb@writepagejump{-3}%
831 \fi
832 \fi
833 \fi
834 \else
Here \texttt{\cb@pagecount} is incorrect.
835 \if@twocolumn
836 \if@firstcolumn
837 \global\cb@advance\cb@pagecount by\tw@
838 \global@cb@pagejumpfalse
839 \else
840 \if@cb@pagejump
841 \cb@trace{Page jump annulled, %
842 \string\cb@pagecount=\the\cb@pagecount}
843 \else
844 \cb@writepagejump{-1}%
845 \global\cb@advance\cb@pagecount by\tw@
846 \cb@rerun
847 \fi
848 \fi
\end{verbatim}
These internal \LaTeX{} 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 box \texttt{box255}. The bars carried over from the previous page, and hence to be restarted on this page, have been saved on the stack \texttt{cb@beginstack}. This stack is used to define new starting points for the change bars, which are added to the top of box \texttt{@cclv}. Then the stack \texttt{cb@endstack} is built and processed by \texttt{cb@processActive}. Finally the original \texttt{@makecol} (saved as \texttt{cb@makecol}) is executed.

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

In two-column 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 \texttt{@outputbox}.
When \TeX{} makes a page with only floats it doesn’t use \texttt{@makecol}; instead it calls \texttt{@vtryfc}, so we have to modify this macro as well. In twocolumn mode we must write either \texttt{@cb@firstcolumntrue} or \texttt{@cb@firstcolumnfalse} to the \texttt{.aux} file.

\begin{verbatim}
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 \texttt{\cb@pagecount} is correct. Then generate a \texttt{@cb@firstcolumntrue} or \texttt{@cb@firstcolumnfalse} in twocolumn mode.

\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 }
\end{verbatim}

\texttt{\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:

\begin{verbatim}
905 \def\cb@processActive{%
906 \cb@pop\cb@endstack
907 \ifnum\cb@topleft=\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@beginstack
915 \cb@push\cb@historystack
916 \expandafter\cb@processActive
917 \fi}
\end{verbatim}

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

\def\cb@startSpanBars{%
  \cb@pop\cb@beginstack
  \ifnum\cb@topleft=\cb@nil
    \else
      \cb@setBeginPoints
      \cb@trace@stack{after StartSpanBars, page \the\c@page}\
      \expandafter\cb@startSpanBars
    \fi
  \fi}

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

\def\cb@buildstack{%
  \cb@initstack\cb@endstack
  \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.

\def\cb@pushNextActive{%
  \cb@pop\cb@historystack
  \ifnum\cb@topleft=\cb@nil
    \else
      \ifnum\cb@page>\cb@pagecount
        \cb@push\cb@historystack
      \else
        \ifodd\cb@topleft
          \cb@push\cb@endstack
        \else
          \cb@pop\cb@endstack
        \fi
      \fi
      \expandafter\expandafter\expandafter\cb@pushNextActive
    \fi
  \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 \texttt{<element> ... <element>}

Each element is of the form \texttt{xxxxyyyyzzzl} where \texttt{xxx} is the number of the special point, \texttt{yyy} is the page on which this point is set, and \texttt{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 \texttt{\cb@currentstack} contains all the current bars. A \texttt{\cb@start} pushes an element onto this stack. A \texttt{\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 \texttt{\cb@historystack}.

\begin{Verbatim}
\cb@initstack \texttt{\cb@initstack} A macro to (globally) initialize a stack.
\end{Verbatim}

\begin{Verbatim}
\cb@historystack \texttt{\cb@historystack} We need to initialise a stack to store the entries read from the external history file.
\end{Verbatim}

\begin{Verbatim}
\cb@initstack \cb@read \cb@write \texttt{\cb@read} \texttt{\cb@write} We also need to allocate a read and a write stream for the history file.
\end{Verbatim}

\begin{Verbatim}
\cb@endstack \texttt{\cb@endstack} Allocate two stacks for the bars that span the current page break.
\end{Verbatim}

\begin{Verbatim}
\cb@beginstack \texttt{\cb@beginstack} Allocate a stack for temporary storage
\end{Verbatim}

\begin{Verbatim}
\cb@currentstack \texttt{\cb@currentstack} And we allocate an extra stack that is needed to implement nesting without having to rely on \TeX{}’s grouping mechanism.
\end{Verbatim}

\begin{Verbatim}
\cb@pop \texttt{\cb@pop} This macro pops the top element off the named stack and puts the point value into \texttt{\cb@topleft}, the page value into \texttt{\cb@page} and the bar width into \texttt{\cb@curbarwd}. If the stack is empty, it returns a void value (\texttt{\cb@nil}) in \texttt{\cb@topleft} and sets \texttt{\cb@page}=0.
\end{Verbatim}

\begin{Verbatim}
\cb@thehistorystack \texttt{\cb@thehistorystack} \def\cb@thehistorystack\{\cb@historystack}\}
\end{Verbatim}

\begin{Verbatim}
\def\cb@pop#1{\% \texttt{\cb@pop#1{}} \texttt{\ifx \empty} \def\cb@temp#1{\% \texttt{\ifx \cb@temp\cb@thehistorystack} \\texttt{\else}} \\texttt{\endlinechar=-1\read\cb@read to\cb@temp} \\texttt{\xdef\cb@historystack\{\cb@historystack\cb@temp\}} \% \texttt{\fi} \%}
\end{Verbatim}
\fi
\fi
\ifx#1\@empty
\global\cb@topleft\cb@nil
\global\cb@page\z@\relax
\else
\expandafter\cb@carcdr#1\expandafter\cb@trace@pop{#1}}

\cb@carcdr \ This macro is used to 'decode' a stack entry.
\def\cb@carcdr#1n#2p#3l#4e#5{%
\global\cb@topleft#1\relax
\global\cb@page#2\relax
\global\cb@curbarwd#3\relax
\xdef#5{#4}}%
\cb@push \ The macro \cb@push Pushes \cb@topleft, \cb@page and \cb@curbarwd onto the
top of the named stack.
\def\cb@push#1{%
\xdef#1{\the\cb@topleft n\the\cb@page p\the\cb@curbarwd l#1}%
\cb@trace@push{#1}}%
\cb@barpoint \ The macro \cb@barpoint populates the history file. It writes one line to .cb file
which is equivalent to one \langle element \rangle described above.
\def\cb@barpoint#1#2#3{\cb@cnta=#2
\if@cb@firstcolumn\advance\cb@cnta by\m@ne\fi
\immediate\write\cb@write{#1n\the\cb@cnta p#3l}}%

5.7 Macros For Checking That The .aux File Is Stable
\AtBeginDocument \ While reading the .aux file, \LaTeX{} has created the history stack in a separate file.
We need to close that file and open it for reading. Also the 'initialisation' of the \special commands has to take place. While we are modifying the macro we also
include the computation of the possible positions of the changebars.
For these actions we need to add to the \LaTeX{} begin-document hook.
\AtBeginDocument{%
\cb@setup@specials
Add a sentinel to \cb@pagejumplst.
\cb@pagejump{(999999999,)}%
Compute the left and right positions of the changebars.
\cb@positions
\cb@trace{%
Odd left : \the\cb@odd@left\space
Odd right : \the\cb@odd@right\MessageBreak
Even left : \the\cb@even@left\space
Even right : \the\cb@even@right
}
\immediate\closeout\cb@write
\immediate\openin\cb@read={\jobname.cb}
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 \TeX itself implies that there are.) Then closes the `.cb` file and reopen it for checking. Initialize history stack (to be read from file). Let `\cb@barpoint=\cb@checkHistory` for checking.

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

Both page and point numbers are the same as the arguments #1 and #2 respectively. Prints a warning message if different.

The macro `\cb@rerun` is called when we detect that we need to rerun \TeX. Dummy definition for `\cb@checkPdfxy`. This will be overwritten by the `pdftex` and `xetex` options.
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 \xelatex we also disable \cb@checkPdfxy.

5.8 Macros For Making It Work With Nested Floats/Footnotes

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.

This only works if this new version of \end@float is really used. With \LaTeX2e
the documentstyles used to contain:

In that case this binding has to be repeated after the redefinition of \end@float.
However, the \LaTeX2e class files use \newenvironment to define the \texttt{figure}
and \texttt{table} environments. In that case there is no need to rebind \end@float.

There is one snag with this redefinition in that the macro \end@float is also used
by the command \texttt{\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.
When the `float` package is being used we need to take care of its changes to the float mechanism. It defines its own macros (\texttt{\textbackslash float@end} and \texttt{\textbackslash float@dblend}) which need to be modified for changebars to work.

First we'll save the original as \texttt{\textbackslash flt@float@end}.

\begin{verbatim}
def \float@end{\cb@trace@stack{end float on page \the\c@page}\cb@pop\cb@currentstack
  \ifnum\cb@topleft=\cb@nil
    \else
      \cb@push\cb@currentstack
      \global\cb@curbarwd=\cb@curbarwd
      \@endfloatbox
      \global\setbox\@currbox
      \color@vbox
      \normalcolor
      \vbox{\begin{cb}[\cb@curbarwd]\unvbox\@currbox\cb@end}
  \fi
  \let\end@float\ltx@end@float
  \flt@float@end}
end@dblfloat
\end{verbatim}

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

\begin{verbatim}
def \cb@end@dblfloat{%\cb@trace@stack{end dblfloat on page \the\c@page}\cb@pop\cb@currentstack
  \ifnum\cb@topleft=\cb@nil
    \else
      \cb@push\cb@currentstack
      \global\cb@curbarwd=\cb@curbarwd
      \@endfloatbox
      \global\setbox\@currbox
      \color@vbox
      \normalcolor
      \vbox{\begin{cb}[\cb@curbarwd]\unvbox\@currbox\cb@end}
  \fi
  \@endfloatbox
  \ifnum\@floatpenalty <\z@
    \@largefloatcheck
    \@cons\@dbldeferlist\@currbox
  \fi
  \ifnum\@floatpenalty =-\@Mii \@Esphack\fi
  \else
  \end@float
  \fi}
\let\end@dblfloat\cb@end@dblfloat
\end{verbatim}

This is a replacement for the \LaTeX-macro of the same name. All it does is check to see if changebars are active and, if so, it puts changebars around the box containing the float. In this case the \LaTeX macro had to be rewritten.
\float@dblend \Something similar needs to be done for the case where the float package is being used...

\let\flt@float@dblend\float@dblend
\def\float@dblend{%
  \cb@trace@stack{end dbl float on page \the\c@page}%
  \cb@pop\cb@currentstack
  \ifnum\cb@topleft=\cb@nil
    \else
      \cb@push\cb@currentstack
      \global\cb@curbarwd=\cb@curbarwd
      \endfloatbox
      \global\setbox\@currbox
      \color@vbox
      \normalcolor
      \vbox\bgroup\cb@start[\cb@curbarwd]\unvbox\@currbox\cb@end
  \fi
  \let\end@dblfloat\ltx@end@dblfloat
  \flt@float@dblend
}[
1114]

\@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.

\let\ltx@footnotetext\@footnotetext
\long\def\cb@footnotetext#1{%
  \cb@trace@stack{end footnote on page \the\c@page}%
  \cb@pop\cb@currentstack
  \ifnum\cb@topleft=\cb@nil
    \ltx@footnotetext{#1}%
  \else
    \cb@push\cb@currentstack
    \edef\cb@temp{\the\cb@curbarwd}%
    \ltx@footnotetext{\cb@start[\cb@temp]#1\cb@end}%
  \fi
}[
1130]

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

\let\ltx@mpfootnotetext\@mpfootnotetext
\long\def\cb@mpfootnotetext#1{%
  \cb@pop\cb@currentstack
  \ifnum\cb@topleft=\cb@nil
    \ltx@mpfootnotetext{#1}%
  \else
    \cb@push\cb@currentstack
    \edef\cb@temp{\the\cb@curbarwd}%
    \ltx@mpfootnotetext{\cb@start[\cb@temp]#1\cb@end}%
  \fi
}[
1141]

41
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.