The Changebar package *

Michael Fine
Distributed Systems Architecture
Johannes Braams
Kersengaarde 33
2723 BP Zoetermeer
The Netherlands
texniek@braams.cistron.nl

Printed October 13, 2002

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. More can easily be added.

1 Introduction

Important note Just as with cross references and labels, you usually need to pro-
cess 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 v3.4d, last revised 2001/09/04.
• 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 only been tested with the dvips driver, but it may also work with other PostScript based drivers. It will not work for the DVItoLN03 and emTeX drivers.

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

One set of package options\footnote{For older documents the command \texttt{\driver} is available in the preamble of the document. It takes the options as defined for \LaTeX{} 2e 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
• \texttt{emTeX}
• \texttt{VTeX}

The drivers are represented in the normal typewriter method of typing these names, or by the same entirely in lower case.

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 \texttt{innerbars} package option (the default), or on the outer edge of the page by use of the \texttt{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 (\texttt{leftbars}) or on the right side of the text (\texttt{rightbars}).

For people who want their changebars to be colourfull the option \texttt{color} is available. It defines the user command \texttt{\cbcolor} and loads the \texttt{color} package.

The package also implements tracing for its own debugging. The package options \texttt{traceon} and \texttt{traceoff} control tracing. An additional option \texttt{tracestacks} is available for the die hard who wants to know what goes on in the internal stacks this package maintains.
2.2 Macros defined by the package

\cbstart \cbend 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[\(\text{dimension}\)]. If no width is specified, the current value of the parameter \changebarwidth is used. Note that \cbstart and \cbend can be used anywhere but must be correctly nested with floats and footnotes. That is, one cannot have one end of the bar inside a floating insertion and the other outside, but that would be a meaningless thing to do anyhow.

changebar Apart from the macros \cbstart and \cbend a proper \LaTeX{} environment is defined. The advantage of using the environment whenever possible is that \LaTeX{} will do all the work of checking the correct nesting of different environments.

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

\nochangepbar The macro \nochangepbar disables the changebar commands.
\cbcolor This macro is defined when the \texttt{color} option is selected. It’s syntax is the same as the \texttt{color} command from the \texttt{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 \texttt{setlength} command. Changing the value of \changebarwidth affects all subsequent changebars subject to the scoping rules of \texttt{setlength}.

\deletebarwidth The width of the deletebars is controlled with the \LaTeX{} length parameter \deletebarwidth. Its value can be changed with the \texttt{setlength} command. Changing the value of \changebarwidth affects all subsequent deletebars subject to the scoping rules of \texttt{setlength}.

\changebarsep The separation between the text and the changebars is determined by the value of \changebarsep.
\changebargrey When one of the supported \textsc{dvi} to \textsc{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:

\begin{verbatim}
\setcounter{changebargrey}{85}
\end{verbatim}

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 \texttt{twoside} is in effect the bars will be printed on the right side of even pages. This behaviour can be changed by including the command \texttt{outerbarstrue} in your document.

3 Deficiencies and bugs

- The macros blindly use special points \texttt{\cb@minpoint} through \texttt{\cb@maxpoint}. If this conflicts with another set of macros, the results will be unpredictable.
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.

\begin{verbatim}
blab blab blab       % page n
blab blab blab
\cbstart            % this does its thing and set the flag
more blab

<--------------- pagebreak occurs here
more blab
\cbend              % does its thing and unsets flag
\end{verbatim}
Since \TeX processes ahead of the page break before invoking the output routine, it is possible that the \texttt{\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 \texttt{.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.

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

\texttt{\cb@maxpoint} The original version of \texttt{changebar.sty} only supported the \texttt{DVitoLN03} specials. The \texttt{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 \texttt{(\*package)}
2 \texttt{\def\cb@maxpoint{80}}

\texttt{\cb@minpoint} When resetting the point number we need to know what to reset it to, this is minimum number is stored in \texttt{\cb@minpoint}. \textbf{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 \texttt{\def\cb@minpoint{1}}

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

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

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

5 \texttt{\newcounter{\cb@nextpoint}}
6 \texttt{\cb@nextpoint=\cb@minpoint}
These four counters are used to identify the four special points that specify a changebar. The point defined by \texttt{cb@topleft} is the one used to identify the changebar; the values of the other points are derived from it.

\newcount{cb@topleft}
\newcount{cb@topright}
\newcount{cb@botleft}
\newcount{cb@botright}

\texttt{cb@curbarwd} The dimension register \texttt{cb@curbarwd} is used to store the width of the current bar.
\newdimen{cb@curbarwd}

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

\texttt{outerbars} A switch is provided to control where the changebars will be printed.
\newif{outerbars}
\texttt{@cb@trace} A switch to enable tracing of the actions of this package
\newif{cb@trace}

\texttt{cb@positions} This macro calculates the (horizontal) positions of the changebars.

\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 \texttt{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 \texttt{changebarsep} whitespace between the bar and the body text.

First the position for odd pages is calculated. I

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

\begin{verbatim}
\if@twoside
  \global\advance\cb@even@left by \evensidemargin
  \global\cb@even@right\cb@even@left
  \global\advance\cb@even@left by \textwidth
  \global\advance\cb@even@right by \changebarwidth
\else
  \global\let\cb@even@left\cb@odd@left
  \global\let\cb@even@right\cb@odd@right
\fi
\end{verbatim}

\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\dimen}. This can be done by defining a command that has a delimited argument like:

\begin{verbatim}
\def\cb@removedim#1pt{#1}
\end{verbatim}

We encounter one problem though, the category code of the letters ‘pt’ is 12 when produced as the output from \texttt{\the\dimen}. 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.

\begin{verbatim}
{\catcode`\p=12\catcode`\t=12 \gdef\cb@removedim#1pt{#1}}
\end{verbatim}

\section{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
- V\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 lowercase equivalents.

\begin{verbatim}
\DeclareOption{DVItoLN03}{\global\chardef\cb@driver@setup=0\relax}
\DeclareOption{dvitoln03}{\global\chardef\cb@driver@setup=0\relax}
\DeclareOption{DVItoPS}{\global\chardef\cb@driver@setup=1\relax}
\end{verbatim}
The new features of \LaTeX{} make it possible to implement the outerbars option. 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 override a possible twoside option at the document level.
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{%
    Pushed point \the\cb@topleft\space on \noexpand#1: #1}}%
  \def\cb@trace@pop#1{\cb@trace{%
    Popped point \the\cb@topleft\space from \noexpand#1: #1}}%
}%

Two 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}}
\DeclareOption{color}{%
  \def\cb@ps@color{\expandafter\c@lor@to@ps\cb@current@color\@@}}

Signal an error if an unknown option was specified.

\DeclareOption*{\OptionNotUsed\PackageError
  \CurrentOption'}%
  \known options are dvitoln03, dvitops, dvips,
  emtex, textures and vtex\MessageBreak
  grey, color,
  outerbars, innerbars, leftbars and rightbars.}}

The default is to have grey change bars on the left side of the text on odd pages. When \TeX is used the option dvips is not the right one, so in that case we have vtex as the default driver.

\ifx\VTeXversion\undefined
  \ExecuteOptions{innerbars,traceoff,dvips,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 \@gobble.

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

These macros are used to display the push and pop operations that go on when stack tracing is turned on. They are defined when the package option tracestacks is selected.

The default is to not trace the stacks.

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

\ProcessOptions\relax

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

\newcommand{\cb@trace}[1]{
\if@cb@trace
\GenericWarning{(changebar)\@spaces\@spaces}\
\PackageWarning{changebar: #1\@gobble}\fi
}

5.3 User Level Commands And Parameters

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

- DVItoln03
- DVItops
- DVlps
- emTeX
- TExtures
- VTeX

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.
The choice has to be communicated to the macro \texttt{\cb@setup@specials} that will be called from within \texttt{document}. For this purpose the control sequence \texttt{\cb@driver@setup} is used. It receives a numeric value using \texttt{\chardef}.

\begin{verbatim}
150 \global\chardef\cb@driver@setup=0\relax
151 \ifx\tempa\LN \global\chardef\cb@driver@setup=0\fi
152 \ifx\tempa\DVItoPS \global\chardef\cb@driver@setup=1\fi
153 \ifx\tempa\DVIPS \global\chardef\cb@driver@setup=2\fi
154 \ifx\tempa\emTeX \global\chardef\cb@driver@setup=3\fi
155 \ifx\tempa\Textures \global\chardef\cb@driver@setup=4\fi
156 \ifx\tempa\VTeX \global\chardef\cb@driver@setup=5\fi
\end{verbatim}

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

\begin{verbatim}
158 {\def\do{\noexpand\do\noexpand}
159 \xdef\@preamblecmds{\@preamblecmds \do\driver}
160 }
161 \fi
\end{verbatim}

\texttt{\cb@setup@specials} defines macros containing the driver specific \texttt{\special} macros. It will be called from within the \texttt{\begin{document}} command.

\texttt{\cb@trace@defpoint} When tracing is on, write information about the point being defined to the log file.

\begin{verbatim}
162 \def\cb@trace@defpoint#1#2{%
163 \cb@trace{%
164 defining point \the#1 at position \the#2
165 }\MessageBreak
166 \cb@pagecount: \the\cb@pagecount; page \thepage}
\end{verbatim}

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

\begin{verbatim}
167 \def\cb@trace@connect#1#2#3{%
168 \cb@trace{%
169 connecting points \the#1 and \the#2; barwidth: \the#3
170 }\MessageBreak
171 \cb@pagecount: \the\cb@pagecount; page \thepage}
\end{verbatim}

\texttt{\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: \texttt{\cb@defpoint\{number\}\{length\}}.

\texttt{\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.

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

\begin{verbatim}
172 \def\cb@setup@specials{%
\end{verbatim}
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 `DVIToLN3` (actually one of its predecessors, `ln03dvi`). Therefore this syntax is defined first.

\begin{verbatim}
\ifcase\cb@driver@setup
  \def\cb@defpoint##1##2{%
    \special{ln03:defpoint \the##1(\the##2,)\%}
  \cb@trace@defpoint##1##2}
\def\cb@connect##1##2##3{%
  \special{ln03:connect \the##1\space space \the##2\space space \the##3\%}
\cb@trace@connect##1##2##3}
\def\cb@resetpoints{%
  \special{ln03:resetpoints \cb@minpoint \space \cb@maxpoint}}
\else
  \def\cb@defpoint##1##2{%
    \special{dvitops: inline\expandafter\cb@removedim\the##2\space 6.5536 mul\space/CBarX\the##1\space exch def currentpoint exch pop
      /CBarY\the##1\space exch def,}
  \cb@trace@defpoint##1##2}
\def\cb@connect##1##2##3{%
  \special{dvitops: inline\gsave \cb@ps@color\space
    /\expandafter\cb@removedim\the##3\space 12 mul\space/CBarX\the##1\space/CBarY\the##1\space moveto
    /CBarX\the##2\space/CBarY\the##2\space lineto
    \cb@trace@connect##1##2##3}
  \let\cb@resetpoints\relax
\end{verbatim}

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

\begin{verbatim}
\or
  \def\cb@defpoint##1##2{%
    \special{dvitops: inline\expandafter\cb@removedim\the##2\space
      \expandafter\cb@removedim\the##2\space 6.5536 mul\space/CBarX\the##1\space exch def currentpoint exch pop
      /CBarY\the##1\space exch def,}
  \cb@trace@defpoint##1##2}
\def\cb@connect##1##2##3{%
  \special{dvitops: inline\gsave \cb@ps@color\space
    \expandafter\cb@removedim\the##3\space 12 mul\space/CBarX\the##1\space/CBarY\the##1\space moveto
    /CBarX\the##2\space/CBarY\the##2\space lineto
    \stroke\gsrestore\cb@trace@connect##1##2##3}
  \let\cb@resetpoints\relax
\end{verbatim}

The program `DVITps` 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. `DVITps` 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.

\begin{verbatim}
\or
  \def\cb@defpoint##1##2{%
    \special{ps:\expandafter\cb@removedim\the##2\space
      \expandafter\cb@removedim\the##2\space 6.5536 mul\space Resolution\space mul\space 72.27\space div\space
      \expandafter\cb@removedim\the##3\space 12 mul\space/CBarX\the##1\space exch def currentpoint exch pop
      /CBarY\the##1\space exch def,}
  \cb@trace@defpoint##1##2}
\def\cb@connect##1##2##3{%
  \special{ps:\gsave \cb@ps@color\space
    \expandafter\cb@removedim\the##3\space 12 mul\space/CBarX\the##1\space/CBarY\the##1\space moveto
    /CBarX\the##2\space/CBarY\the##2\space lineto
    \stroke\gsrestore\cb@trace@connect##1##2##3}
\end{verbatim}

12
The latest addition is for the drivers written by Eberhard Mattes. The \special syntax used here is supported since version 1.5 of his driver programs.

The following definitions are validated with \TeXtures version 1.7.7, but will very likely also work with later releases of \TeXtures. The \cbdelete command seemed to create degenerate lines (i.e., lines of 0 length). PostScript will not render such lines unless the linecap is set to 1, (semicircular ends) in which case a filled circle is shown for such lines.

The following definitions were kindly provided by Michael Vulis.
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.

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

\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.

\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.

\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.

\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.

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

\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.
\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 dimen register \changebarsep.
\newlength{\changebarsep}
\setlength{\changebarsep}{30pt}

\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 the option color was selected we need to load the color package.
\@ifpackagewith{changebar}{color}{{%
\RequirePackage{color}%
Then we need to define the command \cbcolor which is a slightly modified copy of the command \color from the color package.
\cbcolor{\textit{declared-colour}} switches the colour of the changebars to \textit{declared-colour}, which must previously have been defined using \definecolor. This colour will stay in effect until the end of the current \TeX group.
\cbcolor{\textit{model}}{\textit{colour-specification}} is similar to the above, but uses a colour not declared by \definecolor. The allowed \textit{model}'s vary depending on the driver. The syntax of the \textit{colour-specification} argument depends on the model.
\DeclareRobustCommand\cbcolor{%
\@ifnextchar[\@undeclaredcbcolor\@declaredcbcolor}{%
\@undeclaredcbcolor[#1]{%\@ifundefined{color@#1}{{
\colorerror{model `#1'}}}{\csname color@#1\endcsname\cb@current@color{#1}}}%
\@declaredcbcolor{%\@ifundefined{\string\color @#1}{{
\colorerror{`#1'}}}{\expandafter\let\expandafter\cb@current@color\csname\string\color @#1\endcsname}}}%

\@undeclaredcbcolor Call the driver-dependent command \texttt{\color@(model)} to define \texttt{\cb@current@color}.
\@ifnum\@declaredcbcolor=0{
\def\@declaredcbcolor{1}{%
\@ifundefined{\string\color @#1}{{
\colorerror{`#1'}}}{\expandafter\let\expandafter\cb@current@color\csname\string\color @#1\endcsname}}}%
\@ifnum\@undeclaredcbcolor=0{%
\def\@undeclaredcbcolor{1}{%
\@ifundefined{\string\color @#1}{{
\colorerror{`#1'}}}{\expandafter\let\expandafter\cb@current@color\csname\string\color @#1\endcsname}}}%

\@declaredcbcolor
When the `color` option wasn’t specified the usage of the `\cbcolor` command results in an error message.

```latex
\def\cbcolor{%\def\@cbcolor\cb@colwarn\def\cbcolor[#1]{}}% \def\cb@colwarn{\PackageError{Changebar}{You didn't specify the option `color'; your command \string\cbcolor\space will be ignored}{Either add the option `color' or remove the offending commands}}%}
```

### 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.

```latex
\def\cb@start[#1]{% \cb@topleft=\cb@nextpoint \cb@curbarwd#1\relax \cb@push\cb@currentstack \cb@checkpage@ne \cb@setBeginPoints \ifvmode \vbox to \z@{\cb@setBeginPoints} \fi \cb@pagecount\@tempcnta \cb@advancePoint}
```

**\cb@advancePoint**

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

```latex
\def\cb@advancePoint{%}
```
\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 \textit{beginning} point number. It writes the .aux file and joins the points.

\begin{verbatim}
\def\cb@end{% 
\cb@trace@stack{end of bar on page \the\c@page}% 
\cb@pop\cb@currentstack 
\ifnum\cb@topleft=\cb@nil 
\PackageWarning{Changebar}{Badly nested changebars; Expect erroneous results}% 
\else 
Call \cb@checkpage to find the page this point finally ends up on. 
\cb@checkpage\tw@
 Again, we need to temporarily overwrite \cb@pagecount. 
\@tempcnta\cb@topleft\@tempcntb\cb@page 
Then pop the history stack. 
\cb@pop\cb@historystack 
If it was empty there is nothing to check and we’re done. 
\ifnum\cb@topleft=\cb@nil 
\else 
Now keep popping the stack until \cb@topleft is no longer less than the value of \@tempcnta. 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.
\@whilenum\cb@topleft<\@tempcnta\do{% 
\cb@push\cb@tempstack 
\cb@pop\cb@historystack 
When the user adds changebars to his document we might run out of the history stack before we find a match. This would send \TeX into an endless loop if it wasn’t detected and handled. 
\ifnum\cb@topleft=\cb@nil 
\cb@trace{ran out of history stack, new changebar?}% 
\end{verbatim}

\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.

\begin{verbatim}
\def\cb@checkpage#1{% 
First store the identifiers in temporary registers. 
\@tempcnta\cb@topleft\@tempcntb\cb@page 
Then pop the history stack. 
\cb@pop\cb@historystack 
If it was empty there is nothing to check and we’re done. 
\ifnum\cb@topleft=\cb@nil 
\else 
Now keep popping the stack until \cb@topleft is no longer less than the value of \@tempcnta. 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.
\@whilenum\cb@topleft<\@tempcnta\do{% 
\cb@push\cb@tempstack 
\cb@pop\cb@historystack 
When the user adds changebars to his document we might run out of the history stack before we find a match. This would send \TeX into an endless loop if it wasn’t detected and handled. 
\ifnum\cb@topleft=\cb@nil 
\cb@trace{ran out of history stack, new changebar?}% 
\end{verbatim}
In this case we give \cb@topleft an ‘impossible value’ to remember this special situation.
\begin{verbatim}
\cb@topleft\cb@maxpoint\advance\cb@topleft\@ne
\fi
\}
\end{verbatim}
If we are looking for the start point of a bar we may have found it now, for the end point we need to pop one more value. If \cb@topleft has become larger than \cb@maxpoint we haven’t found what we’re looking for and we’ve run out of the stack.
\begin{verbatim}
\ifnum\cb@topleft>\cb@maxpoint\else
  \ifodd\cb@pagecount
    \cb@defpoint\cb@topleft\cb@even@left
    \cb@defpoint\cb@topright\cb@even@right
  \else
    \cb@defpoint\cb@topleft\cb@odd@left
    \cb@defpoint\cb@topright\cb@odd@right
  \fi
\fi
\@tempcntb\cb@page
\fi
\end{verbatim}
Now we restore the history stack to it’s original state.
\begin{verbatim}
\@whilenum\cb@topleft>\cb@nil\do{\%
  \cb@push\cb@historystack
  \cb@pop\cb@tempstack}%
\fi
\end{verbatim}
Finally return the correct values.
\begin{verbatim}
\cb@topleft\@tempcnta\cb@page\@tempcntb
\}
\end{verbatim}
\begin{description}
\item[\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.
\begin{verbatim}
\def\cb@setBeginPoints{%
  \cb@topright=\cb@topleft\advance\cb@topright by\@ne
  \ifodd\cb@pagecount
    \cb@defpoint\cb@topleft\cb@even@left
    \cb@defpoint\cb@topright\cb@even@right
  \else
    \cb@defpoint\cb@topleft\cb@odd@left
    \cb@defpoint\cb@topright\cb@odd@right
  \fi
  \cb@writeAux\cb@topleft
\}
\end{verbatim}

\item[\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.
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.
5.5 Macros for Making It Work Across Page Breaks

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 255. 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.

When \LaTeX{} 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.

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 box 255
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).

462 \def\cb@processActive{%
463 \cb@pop\cb@endstack
464 \ifnum\cb@topleft=\cb@nil
465 \else
466 \setbox\@cclv\vbox{%
467 \unvbox\@cclv
468 \boxmaxdepth\maxdepth
469 \advance\cb@pagecount by -1\relax
470 \cb@setEndPoints}%
471 \cb@push\cb@historystack
472 \cb@push\cb@beginstack
473 \expandafter\cb@processActive
474 \fi}

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

475 \def\cb@startSpanBars{%
476 \cb@pop\cb@beginstack
477 \ifnum\cb@topleft=\cb@nil
478 \else
479 \cb@setBeginPoints
480 \cb@trace@stack{after StartSpanBars, page \the\c@page}%
481 \expandafter\cb@startSpanBars
482 \fi
483 }

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

484 \def\cb@buildstack{%
485 \cb@initstack\cb@endstack
486 \cb@pushNextActive}

\cb@pushNextActive This macro pops the top element off the history stack (\cb@historystack). If the
top left point is on a future page, it is pushed back onto the history stack and
processing stops. If the point on the current or a previous page and it has an odd
number, the point is pushed on the stack with end points \cb@endstack; if the
point has an even number, it is popped off the stack with end points since the bar
to which it belongs has terminated on the current page.

487 \def\cb@pushNextActive{%
488 \cb@pop\cb@historystack
489 \ifnum\cb@topleft=\cb@nil
490 \else
491 \ifnum\cb@page>\cb@pagecount
492 \cb@push\cb@historystack
493 \else
494 \ifodd\cb@topleft
495 \cb@push\cb@endstack
496 \else
5.6 Macros For Managing The Stacks of Bar points

The macros make use of four stacks corresponding to \special defpoints. Each stack takes the form <element> ... <element>

Each element is of the form xxxnyyypzzzl where xxx is the number of the special point, yyy is the page on which this point is set, and zzz is the dimension used when connecting this point.

The stack \cb@historystack is built from the log information and initially lists all the points. As pages are processed, points are popped off the stack and discarded.

The stack \cb@endstack and \cb@beginstack are two temporary stacks used by the output routine and contain the stack with definitions for 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.

\cb@historystack \cb@write \cb@read

We need to initialise a stack to store the entries read from the external history file.

\cb@initstack\cb@historystack

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

\newwrite\cb@write\newread\cb@read

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

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

\cb@endstack \cb@beginstack

Allocate two stacks for the bars that span the current page break.

\cb@initstack\cb@endstack \cb@initstack\cb@beginstack

\cb@tempstack

Allocate a stack for temporary storage

\cb@initstack\cb@tempstack
And we allocate an extra stack that is needed to implement nesting without having to rely on \TeX’s grouping mechanism.

\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.

\def\cb@pop#1{\ifx #1\cb@historystack\ifeof\cb@read{\endlinechar=-1\read\cb@read to\@temp\xdef\cb@historystack{\cb@historystack\@temp}%%%%%%%%}\fi\fi\ifx#1\@empty\global\cb@topleft\cb@nil\global\cb@page\z@elax\else\expandafter\cb@carcdr#1e#1\fi\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 (element) described above.

\def\cb@barpoint#1#2#3{\immediate\write\cb@write{#1n#2p#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
}
We need to issue a \clearpage to flush rest of document. (Note that I believe there is contention in this area: are there in fact situations in which the end-document hooks need to be called before the final \clearpage? — the documentation of \LaTeX\ itself implies that there are.) Then closes the .cb file and reopen it for checking. Initialize history stack (to be read from file). Let \cb@barpoint=\cb@checkHistory for checking.

\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.

\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.

5.8 Macros For Making It Work With Nested Floats/Footnotes

\end@float This is a replacement for the \LaTeX-macro of the same name. All it does is check to see if changebars are active and, if so, it puts changebars around the box containing the float. Then it calls the original \LaTeX\ \end@float.
This only works if this new version of \end@float is really used. With \LaTeX{}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 \LaTeX{}2g class files use \newenvironment to define the figure and table environments. In that case there is no need to rebind \endfigure.

\@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\cb@footnote=\@footnotetext
\long\def\@footnotetext#1{%
  \cb@trace@stack{end footnote 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{\group\cb@start[\cb@curbarwd]\unvbox\@currbox\cb@end}
  \fi
  \cb@endfloat}

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

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

Numbers in *italics* indicate the page where the macro is described, the underlined numbers indicate the number of the line of code where the macro is defined, all other numbers indicate where a macro is used.

### Symbols

- \@cclv .................................. 441, 444, 457, 466, 467
- \@currbox .......................... 573, 576
- \@declaredcboxcolor .............. 292, 298
- \@empty .................................. 520
- \@endfloatbox .......................... 572
- \@footnotetext .......................... 579
- \@gobble .......................... 128
- 130, 136, 440, 453
- \@ifnextchar .......................... 270, 274, 292, 305
- \@ifpackagewith .................................. 289
- \@ifundefined .................................. 294, 299
- \@makecol .................................. 437
- \@mpfootnotetext ....................... 590
- \@one .......................... 319, 361, 378, 389, 446, 459
- \@outputbox .......................... 454
- \@preamblecmds .......................... 159
- \@spaces .................................. 124, 135
- \@temp .................................. 515, 516
- \@tempcnta .......................... 320, 330, 345, 348, 352, 356, 375
- \@tempcntb .......................... 352, 369, 375
- \@undeclaredcboxcolor .............. 292, 293
- \@vtryfc .................................. 437
- \@whilenum .......................... 396, 371
- \@AtBeginDocument .................. 537

### B

- \begin .................................. 268
- \begingroup .................................. 428
- \bgroup .................................. 142, 576
- \boxmaxdepth .................................. 445, 458, 468
- \@lor@error .................................. 295, 300
- \@lor@tops .................................. 107
- \@page .................................. 338, 439, 449, 480, 566, 581
- \@catcode .................................. 44
- \@cb@show@stack .......................... 98, 120
- \@cb@AdvancePoint ....................... 331, 332
- \@cb@barpoint .......................... 433, 526, 547, 563
- \@cb@beginstack .......................... 125, 472, 476, 507
- \@cb@botleft .................................. 390, 393, 397, 400, 406
- \@cb@botright .......................... 270, 274, 276, 280
- \@cb@buildstack .......................... 447, 460, 484
- \@cb@carcdr .......................... 524, 527
- \@cb@checkHistory .......................... 547, 548, 562, 563
- \@cb@checkpage .......................... 319, 344, 351
- \@cb@colwarn .......................... 307-309
- \@cb@connect .................................. 406, 408
- \@cb@curbarwd .................................. 317, 406, 408
- \@cb@defpoint .................................. 172, 380, 381, 383

### A

- \@advance .................................. 24, 26
- 30, 32, 34-38, 63-65, 67-69
- \@whilenum .......................... 391, 446, 459, 469
- \@AtBeginDocument .................. 537

### C

- \@cb@current@color .......................... 107, 296, 301
- \@cb@currentstack  ....................... 123, 318, 339
- \@cb@defpoint .................................. 172
- \@cb@defpoint .................................. 380, 381, 383

### D

- \@cb@defpoint .................................. 380, 381, 383, 384, 393, 394
- \@cb@delete .......................... 270, 271, 272, 279
- \@cb@driver@setup .......................... 45-56, 150-156, 173
- \@cb@end .................................. 272, 276, 280
- \@cb@endfloat .......................... 564, 578
- \@cb@endstack .......................... 124, 463
- \@cb@even@left .................................. 17
- \@cb@error .................................. 23, 32-35, 40
- \@cb@even@right .................................. 17
- \@cb@footnote .................................. 33, 36-38, 41
- \@cb@footnote .................................. 74, 79, 85-88
- \@cb@footnote .................................. 90, 93, 381, 394, 401
- \@cb@footnote .................................. 537, 584, 588
- \@cb@historystack .......................... 126, 354, 355, 367, 372, 471
- \@cb@makecol .................................. 488, 492, 503
- \@cb@makecol .................................. 512, 516, 544, 549
- \@cb@makecol .................................. 485, 502
- \@cb@makecol .................................. 503, 507-510, 544
- \@cb@leftbar .......................... 405, 410, 414, 416
- \@cb@makecol .................................. 422
- \@cb@maxpoint .......................... 181, 334, 361, 364
- \@cb@minpoint .......................... 181, 6, 181, 335
- \@cb@mpfootnote .......................... 590, 594, 598
- \@cb@nextpoint .......................... 531, 336, 333-335
- \@cb@nil .................................. 340
- \@cb@nil .................................. 354, 359, 371
- \@cb@nil .................................. 464, 477, 489
- \@cb@nil .................................. 521, 568, 583, 593
- \@cb@odd@left .......................... 17, 22-25, 29