The \texttt{xpatch} package
Extending \texttt{etoolbox} patching commands\textsuperscript{*}

Enrico Gregorio\textsuperscript{†}

Released 2012/10/02

1 Introduction

The well known \texttt{etoolbox} package provides a bunch of functions for patching existing commands; in particular \texttt{\patchcmd}, \texttt{\pretocmd} and \texttt{\apptocmd} that do a wonderful job, but suffer from a limitation: if some package has defined

\begin{verbatim}
\newcommand{\xyz}[1][x]{-#1!}
\end{verbatim}

where \texttt{\xyz} has an optional argument, then \texttt{\patchcmd} and siblings cannot be used to modify the workings of \texttt{\xyz}. The same happens when a command has been defined with \texttt{\DeclareRobustCommand}.

The reason for this is \TeX\technical or, better, \LaTeX\technical. When \LaTeX\ performs the above definition, the expansion of \texttt{\xyz} will be

\begin{verbatim}
@protected@testopt \xyz \xyz {x}
\end{verbatim}

where \texttt{\@protected@testopt} is a macro that essentially checks whether we are in a “protected” context, so that expansion should not be performed all the way (in moving arguments or write operations), or not; in the former case it issues a protected version of \texttt{\xyz}, while in the latter case it expands the macro \texttt{\\xyz} that is a single command (yes, with a backslash in its name) which contains the real definition; a way to access this definition is to issue the command

\begin{verbatim}
\expandafter\show\csname\string\xyz\endcsname
\end{verbatim}

which will print in the log file the message

> \texttt{\xyz}=	exttt{\long macro: [\#1]->-\#1!}.

As usual, after -> we see the definition. In order to use \texttt{\patchcmd} to change the exclamation mark into a hyphen one must do

\begin{verbatim}
\expandafter\patchcmd\csname\string\xyz\endcsname{!}{-}{-}{-}
\end{verbatim}

(see the documentation of \texttt{etoolbox} for details about the arguments).

A similar situation happens if \texttt{\xyz} has been defined by

\textsuperscript{*}This file describes version 0.3, last revised 2012/10/02.
\textsuperscript{†}E-mail: Enrico DOT Gregorio AT univr DOT it
\DeclareRobustCommand{\xyz}{something}

A \show\xyz would show the cryptic

> \xyz=macro:
->\protect \xyz .

and only a close look reveals the clever trick used by the \LaTeX{} team: the \protect is not applied to \xyz, but to the macro \xyz_ which has a space at the end of its name! And this macro is the one that contains the real definition. Indeed,

\expandafter\show\csname xyz\space\endcsname
produces the message

> \xyz =\long macro:
->something.

In this case, in order to apply \patchcmd we must say

\expandafter\patchcmd\csname xyz\space\endcsname{s}{S}{\{}{\}}</p>

If the macro with \DeclareRobustCommand is defined to have an optional argument, say

\DeclareRobustCommand{\xyz}[1][x]{-#1!}

one has to combine the two tricks:

\expandafter\patchcmd\csname\string\xyz\space\endcsname{!}{-}{\}{}{\}}</p>

It’s hard and error prone to remember all these tricks, so this package comes to the rescue.

**Caveat**

This package is still in a preliminary version, but no relevant changes to the interface should be introduced in later versions. A different and more powerful implementation is under testing, see the package *regexpatch*.

## 2 Commands

The commands introduced by this package are

- \xpatchcmd
- \xpretocmd
- \xapptocmd

which have the same syntax as the similar commands provided by *etoolbox* and apply to all kind of commands defined by

- the \LaTeX{} kernel macros \newcommand, \renewcommand, \providecommand, but also \newenvironment and \renewenvironment;
- the \LaTeX{} kernel macro for defining robust commands \DeclareRobustCommand;
• the etoolbox macros `\newrobustcmd, \renewrobustcmd, \providerobustcmd`.

Notice that patching the definition of the environment `foo` requires patching `\foo` or `\endfoo`.

These commands will act as the original ones if the macro to patch is not robust or with optional arguments.

Moreover the package defines

• `\xpatchbibmacro`
• `\xpretobibmacro`
• `\xapptobibmacro`

that can be used to patch commands defined with biblatex’s `\newbibmacro`. Say that we have

`\newbibmacro{foo.bar}[2]{#1 and #2}`

Then, to change `and` into `und`, we can now say

`\xpatchbibmacro{foo.bar}{and}{und}{}{}`

Patching these macros requires resorting to the very cryptic

```
\expandafter\patchcmd\csname abx@macro@\detokenize{foo.bar}\endcsname{and}{und}{}{0}
```

that would become an astonishing

```
\expandafter\patchcmd\csname\expandafter\string\csname abx@macro@\detokenize{foo.bar}\endcsname\endcsname{and}{und}{}{0}
```

if the original definition had been with an optional argument, say

`\newbibmacro{foo.bar}[2][x]{#1 and #2}`

For biblatex users there are also

• `\xpatchbibdriver`
• `\xpretobibdriver`
• `\xapptobibdriver`

for patching commands defined with `\DeclareBibliographyDriver`. One could use, for patching the driver `foo`,

```
\makeatletter
\patchcmd\blx@bbx@foo\{X}{Y}{<success>}{<failure>}
\pretocmd\blx@bbx@foo\{P}
\apptocmd\blx@bbx@foo\{A}
\makeatother
```

but having a lighter interface can be handy. Since our macros use `\pretocmd` and `\apptocmd` for consistency, remember to always use the `{success}` and `{failure}` arguments also with `\xpretobibdriver` and `\xapptobibdriver`.

Under the same philosophy, one can use the macros
• \xpatchfieldformat,  
  \xpretofieldformat,  
  \xapptofieldformat,  

• \xpatchnameformat,  
  \xpretonameformat,  
  \xapptonameformat,  

• \xpatchlistformat,  
  \xpretonameformat,  
  \xapptonameformat,  

• \xpatchindexfieldformat,  
  \xpretoindexfieldformat,  
  \xapptoindexfieldformat,  

• \xpatchindexnameformat,  
  \xpretoindexnameformat,  
  \xapptoindexnameformat,  

• \xpatchindexlistformat,  
  \xpretoindexlistformat,  
  \xapptoindexlistformat,  

for the \biblatex internal macro defined respectively with  
\DeclareFieldFormat, \DeclareNameFormat, \DeclareListFormat,  
\DeclareIndexFieldFormat, \DeclareIndexNameFormat, \DeclareIndexListFormat.

All the eighteen \x\ldots format commands take a first optional argument, with default value *, see later on.

Finally, the package defines the commands

• \xshowcmd  
• \xshowbibmacro  
• \xshowbibdriver  
• \xshowfieldformat  
• \xshownameformat  
• \xshowlistformat  
• \xshowindexfieldformat  
• \xshowindexnameformat  
• \xshowindexlistformat

that are the analog of \show to see the “real” definition of a macro, be it defined with optional arguments or as a robust command; the \xshowbib\ldots and \xshow\ldots format ones are for the corresponding \biblatex macros. The last six have an optional first argument (default value *).
3 Using the original commands

The original \texttt{\patchcmd} has still its use: suppose you want to modify the default for the optional argument passed to a macro: if the original definition is

\texttt{\newcommand{\xyz}{[1][x]{-#1!}}}

then one can say

\texttt{\patchcmd{\xyz}{\{x\}}{\{y\}}{}}

because of the way \texttt{\xyz} is defined, as shown before.

4 Syntax

\texttt{\patchcmd{⟨command⟩}{⟨search⟩}{⟨replace⟩}{⟨success⟩}{⟨failure⟩}}
\texttt{\preto{⟨command⟩}{⟨search⟩}{⟨replace⟩}{⟨success⟩}{⟨failure⟩}}
\texttt{\postto{⟨command⟩}{⟨search⟩}{⟨replace⟩}{⟨success⟩}{⟨failure⟩}}
\texttt{\patchbibmacro{⟨name⟩}{⟨search⟩}{⟨replace⟩}{⟨success⟩}{⟨failure⟩}}
\texttt{\prebibmacro{⟨name⟩}{⟨search⟩}{⟨replace⟩}{⟨success⟩}{⟨failure⟩}}
\texttt{\postbibmacro{⟨name⟩}{⟨search⟩}{⟨replace⟩}{⟨success⟩}{⟨failure⟩}}
\texttt{\patchbibdriver{⟨name⟩}{⟨search⟩}{⟨replace⟩}{⟨success⟩}{⟨failure⟩}}
\texttt{\prebibdriver{⟨name⟩}{⟨search⟩}{⟨replace⟩}{⟨success⟩}{⟨failure⟩}}
\texttt{\postbibdriver{⟨name⟩}{⟨search⟩}{⟨replace⟩}{⟨success⟩}{⟨failure⟩}}
\texttt{\patchfieldformat{⟨entrytype⟩}{⟨name⟩}{⟨search⟩}{⟨replace⟩}{⟨success⟩}{⟨failure⟩}}
\texttt{\prefieldformat{⟨entrytype⟩}{⟨name⟩}{⟨search⟩}{⟨replace⟩}{⟨success⟩}{⟨failure⟩}}
\texttt{\postfieldformat{⟨entrytype⟩}{⟨name⟩}{⟨search⟩}{⟨replace⟩}{⟨success⟩}{⟨failure⟩}}
\texttt{\patchnameformat{⟨entrytype⟩}{⟨name⟩}{⟨search⟩}{⟨replace⟩}{⟨success⟩}{⟨failure⟩}}
\texttt{\prenameformat{⟨entrytype⟩}{⟨name⟩}{⟨search⟩}{⟨replace⟩}{⟨success⟩}{⟨failure⟩}}
\texttt{\postnameformat{⟨entrytype⟩}{⟨name⟩}{⟨search⟩}{⟨replace⟩}{⟨success⟩}{⟨failure⟩}}
\texttt{\patchlistformat{⟨entrytype⟩}{⟨name⟩}{⟨search⟩}{⟨replace⟩}{⟨success⟩}{⟨failure⟩}}
\texttt{\prelistformat{⟨entrytype⟩}{⟨name⟩}{⟨search⟩}{⟨replace⟩}{⟨success⟩}{⟨failure⟩}}
\texttt{\postlistformat{⟨entrytype⟩}{⟨name⟩}{⟨search⟩}{⟨replace⟩}{⟨success⟩}{⟨failure⟩}}
\texttt{\patchindexfieldformat{⟨entrytype⟩}{⟨name⟩}{⟨search⟩}{⟨replace⟩}{⟨success⟩}{⟨failure⟩}}
\texttt{\preindexfieldformat{⟨entrytype⟩}{⟨name⟩}{⟨search⟩}{⟨replace⟩}{⟨success⟩}{⟨failure⟩}}
\texttt{\postindexfieldformat{⟨entrytype⟩}{⟨name⟩}{⟨search⟩}{⟨replace⟩}{⟨success⟩}{⟨failure⟩}}
\texttt{\patchindexnameformat{⟨entrytype⟩}{⟨name⟩}{⟨search⟩}{⟨replace⟩}{⟨success⟩}{⟨failure⟩}}
\texttt{\preindexnameformat{⟨entrytype⟩}{⟨name⟩}{⟨search⟩}{⟨replace⟩}{⟨success⟩}{⟨failure⟩}}
\texttt{\postindexnameformat{⟨entrytype⟩}{⟨name⟩}{⟨search⟩}{⟨replace⟩}{⟨success⟩}{⟨failure⟩}}
\texttt{\patchindexlistformat{⟨entrytype⟩}{⟨name⟩}{⟨search⟩}{⟨replace⟩}{⟨success⟩}{⟨failure⟩}}
\texttt{\preindexlistformat{⟨entrytype⟩}{⟨name⟩}{⟨search⟩}{⟨replace⟩}{⟨success⟩}{⟨failure⟩}}
\texttt{\postindexlistformat{⟨entrytype⟩}{⟨name⟩}{⟨search⟩}{⟨replace⟩}{⟨success⟩}{⟨failure⟩}}
\texttt{\showcmd{⟨command⟩}}
\texttt{\showbibname{⟨name⟩}}
\texttt{\showbibdriver{⟨name⟩}}
\texttt{\showfieldformat{⟨entrytype⟩}{⟨name⟩}}
\shownameformat{⟨entrytype⟩}{⟨name⟩}
\showlistformat{⟨entrytype⟩}{⟨name⟩}
\showindexfieldformat{⟨entrytype⟩}{⟨name⟩}
\showindexnameformat{⟨entrytype⟩}{⟨name⟩}
\showindexlistformat{⟨entrytype⟩}{⟨name⟩}

Here ⟨command⟩ is the command’s name (with the backslash), while ⟨name⟩ is the string that appears as the argument to \newbibmacro, \DeclareBibliographyDriver, \DeclareFieldFormat, \DeclareNameFormat, \DeclareListFormat, \DeclareIndexFieldFormat, \DeclareIndexNameFormat or \DeclareIndexListFormat respectively; ⟨search⟩, ⟨replace⟩, ⟨prepend⟩ and ⟨append⟩ are the list of tokens that are to be used for the specific tasks; ⟨success⟩ and ⟨failure⟩ are token lists to be executed if the patching succeeds or fails respectively. I find it useful to use \ddt as ⟨failure⟩, so that \TeX will stop for the undefined control sequence when the patching fails.

All the \x...format macros have an optional argument that by default is *.

It’s important to remember that patching commands that have @-commands in their name or replacement text must always be performed between \makeatletter and \makeatother.

5 Limitations and warnings

Macros defined in devious ways might trick \xpatchcmd and siblings, although many precautions have been taken in order this not to happen. Always check with care.

Remember that one must never use the old trick
\let\ORIxyx\xyz
\renewcommand{\xyz}[1][x]{+\ORIxyz[#1]}

if \xyz had been defined with an optional argument. For such things it’s better to use \xpatchcmd and friends or employ the letltxmacro package by H. Oberdiek, that provides \LetLtxMacro for purposes like this one.

Although this package has been written with the experimental \textit{Bi\TeX} macros, the commands can’t be used to patch commands defined with the xparse interface, in general.

If a command appears to have one optional argument at the user level, this doesn’t mean it has been defined with \newcommand directly. One should always check the definitions with \show and \xshowcmd before trying a patch: of course one has to know what a command does, in order to patch it. And, when first testing the patch, it’s best to set \tracingpatches.

6 History

Version 0.1 First public release.

Version 0.2 Added \x...bibdriver macros; fixed a bug for control symbols defined with \newcommand and an optional argument.

Version 0.3 Added \x...format macros (by kind request of the biblatex maintainers).
7 The implementation of \texttt{xpatch}

\begin{verbatim}
\ProvidesExplPackage{\ExplFileName}{\ExplFileDate}{\ExplFileVersion}{\ExplFileDescription}
A check to make sure that expl3 is not too old
\@ifpackagelater { expl3 } { 2011/10/09 }
\PackageError { xpatch } { Support-package-l3kernel-too-old. }
{ Please-install-an-up-to-date-version-of-l3kernel-
  using-your-TeX-package-manager-or-from-CTAN.\\
  Loading-xpatch-will-abort! }
\tex_endinput:D
\end{verbatim}

The \texttt{xparse} and \texttt{etoolbox} packages are required.
\begin{verbatim}
\RequirePackage{xparse,etoolbox}
\end{verbatim}

7.1 Utilities, variables and constants

Generate a variant of \texttt{\tl_if_in:NnT} to get the expanded second argument.
\begin{verbatim}
\cs_generate_variant:Nn \tl_if_in:NnT { Nx }
\end{verbatim}

A boolean for the testing of robust commands.
\begin{verbatim}
\bool_new:N \l__xpatch_protect_bool
\end{verbatim}

The constant \texttt{c_backslash_str} is defined in \texttt{l3str} that’s not loaded at the moment, so we save a bit of memory not loading it.
\begin{verbatim}
\cs_if_exist:NF \c_backslash_str
{ \tl_const:Nx \c_backslash_str { \cs_to_str:N \textbackslash } }
\end{verbatim}

A “bizarre” token list that’s quite improbable to find in the replacement text of a macro.
\begin{verbatim}
\tl_const:Nx \c__xpatch_bizarre_tl
{ \tl_to_str:n { **)-(**/**\]-[** } }
\end{verbatim}

Internal token lists for storing the various parts of the command to be patched.
\begin{verbatim}
\tl_new:N \l__xpatch_name_tl
\tl_new:N \l__xpatch_repl_tl
\end{verbatim}

7.2 The main functions

The main function takes as first argument one of \texttt{\patchcmd}, \texttt{\pretocmd} or \texttt{\apptocmd}; the second argument is the command we want to patch.

Some technical remarks. Suppose we have the following definitions:

\begin{verbatim}
\DeclareRobustCommand{\xa}[1]{xa (DeclareRobustCommand-noopt)}
\DeclareRobustCommand{\xb}[1][x]{xb (DeclareRobustCommand-opt)}
\newcommand{\xc}[1][]{xc (newcommand-opt)}
\newrobustcmd{\xad}[1][]{xad (newrobustcmd-opt)}
\DeclareRobustCommand{\yd}[1][1]{1 (DeclareRobustCommand-noopt)}
\DeclareRobustCommand{\ze}[2][2]{2 (DeclareRobustCommand-opt)}
\newcommand{\zd}[3][]3 (newcommand-opt))
\newrobustcmd{\zd}[4][]4 (newrobustcmd-opt)}
\end{verbatim}
Then the first level expansions are, respectively,

\[ +\protect/\text{\textbackslash xaa}_{\text{\textbackslash L}} + \]
\[ +\protect/\text{\textbackslash xab}_{\text{\textbackslash L}} + \]
\[ +\text{@protected@testopt}/\text{\textbackslash xac}_{\text{\textbackslash L}}{\texttt{}} + \]
\[ +\text{@testopt}/\text{\textbackslash xad}_{\text{\textbackslash L}}{\texttt{}} + \]
\[ +\text{x@protect}/\text{\textbackslash xae}_{\text{\textbackslash L}}{\texttt{}} + \]
\[ +\text{x@protect}/\text{\textbackslash xaf}_{\text{\textbackslash L}}{\texttt{}} + \]
\[ +\text{@protected@testopt}/\text{\textbackslash xag}_{\text{\textbackslash L}}{\texttt{}} + \]
\[ +\text{@testopt}/\text{\textbackslash xah}_{\text{\textbackslash L}}{\texttt{}} + \]
where the + is used to delimit the expansions and show the spaces. Remember that \texttt{\textbackslash show} always adds a space after a control word, but not after a control symbol such as \texttt{\1}. However, in lines 5 and 6, \texttt{\1} is not a control symbol any more. So we have to take care of \texttt{\protect, \texttt{x@protect, \texttt{@protected@testopt} and \texttt{@testopt}}. But it’s not simply sufficient to check for the presence of such a token at the start of the replacement text, or we’ll be confused by macros such as \texttt{\textbackslash linebreak}, whose replacement text starts with \texttt{\textbackslash @testopt}. So we’ll check also for the presence of the subsequent tokens, that depend on the macro’s name. We add a perhaps useless “random” string at the beginning, as we’d like to ensure that the matches are exactly at the start of the replacement text.

\begin{verbatim}
\cs_new:Npn \xpatch_main:NN #1 #2
\{
\bool_set_false:N \l__xpatch_protect_bool
First of all we store the command-to-patch name.
\tl_set:Nx \l__xpatch_name_tl { \cs_to_str:N #2 }
We store the replacement text of the command-to-patch, but adding the bizarre token list in front of it which consists of all category 12 characters, just to be sure that the matches are at the beginning.\footnote{This part will be reimplemented as soon as \texttt{l3regex} stabilizes.}
\tl_set:Nx \l__xpatch_repl_tl { \c__xpatch_bizarre_tl \token_get_replacement_spec:N #2 }
We look whether the token list contains the bizarre list followed by \texttt{\protect} and the same name (with two spaces) which happens if \texttt{#2} is a control sequence defined by \texttt{\DeclareRobustCommand}, so we add a space to the command name.
\tl_if_in:NxT \l__xpatch_repl_tl
\{\c__xpatch_bizarre_tl \token_to_str:N \protect \c_space_tl \c_backslash_str \l__xpatch_name_tl \c_space_tl \c_space_tl \}
\}
\{ \bool_set_true:N \l__xpatch_protect_bool
\tl_put_right:Nn \l__xpatch_name_tl { \c_space_tl }
\}
We look whether the token list contains the bizarre list followed by \texttt{\texttt{x@protect}} which happens if \texttt{#2} is a control symbol defined by \texttt{\DeclareRobustCommand}, so we add a space to the command name.
\tl_if_in:NxT \l__xpatch_repl_tl
\{ \c__xpatch_bizarre_tl \token_to_str:N \protect \c_space_tl \c_backslash_str \l__xpatch_name_tl \c_space_tl \c_space_tl \}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
\}
In both the preceding cases we have to do another check, so we set a boolean to true.

We look whether the token list contains the bizarre list followed by \@protected@testopt which happens if \#2 is a control word with an optional argument (from \newcommand).

\ifin:NX \__xpatch_repl_tl
{ \c__xpatch_bizarre_tl \token_to_str:N \x@protect \c_space_tl \c_backslash_str \l__xpatch_name_tl \c_backslash_str \}
{ \bool_set_true:N \l__xpatch_protect_bool \tl_put_right:Nn \l__xpatch_name_tl { \c_space_tl }
}

We look whether the token list contains the bizarre list followed by \@protected@testopt which happens if \#2 is a control symbol with an optional argument (from \newcommand).

\ifin:NX \__xpatch_repl_tl
{ \c__xpatch_bizarre_tl \token_to_str:N \@protected@testopt \c_space_tl \c_backslash_str \l__xpatch_name_tl \c_backslash_str \}
{ \tl_put_left:Nn \l__xpatch_name_tl { \c_backslash_str }
}

We look whether the token list contains the bizarre list followed by \@testopt which happens if \#2 is a command with an optional argument (from \newrobustcmd).

\ifin:NX \__xpatch_repl_tl
{ \c__xpatch_bizarre_tl \token_to_str:N \@testopt \c_space_tl \c_backslash_str \c_backslash_str \l__xpatch_name_tl \c_backslash_str \}
{ \tl_put_left:Nn \l__xpatch_name_tl { \c_backslash_str }
}

In both the preceding cases, we add a backslash in front of the command’s name.

If the command-to-patch was defined by \DeclareRobustCommand we have to do another test, namely checking whether it has an optional argument and, in this case, adding a backslash in front of the name. We replicate the test for \@protected@testopt.

\bool_if:NT \l__xpatch_protect_bool
{ \}
\_tl_set:Nx \l__xpatch_repl_tl
{ \c__xpatch_bizarre_tl
\exp_after:wN \token_get_replacement_spec:N
\cs:w \l__xpatch_name_tl \cs_end: }
\tl_if_in:NxT \l__xpatch_repl_tl
{
\c__xpatch_bizarre_tl
\token_to_str:N \@protected@testopt \c_space_tl
\c_backslash_str \l__xpatch_name_tl
\c_space_tl \c_backslash_str \c_backslash_str
}
\tl_put_left:Nn \l__xpatch_name_tl { \c_backslash_str }
}\tl_if_in:NxT \l__xpatch_repl_tl
{
\c__xpatch_bizarre_tl
\token_to_str:N \@protected@testopt \c_space_tl
\c_backslash_str \l__xpatch_name_tl
\c_space_tl \c_backslash_str \c_backslash_str
}
\tl_put_left:Nn \l__xpatch_name_tl { \c_backslash_str }
\tl_if_in:NxT \l__xpatch_repl_tl
{\tl_set:Nx \l__xpatch_repl_tl
{ \c__xpatch_bizarre_tl
\exp_after:wN \token_get_replacement_spec:N
\cs:w \l__xpatch_name_tl \cs_end: }
\tl_if_in:NxT \l__xpatch_repl_tl
{\tl_set:Nx \l__xpatch_repl_tl
{ \c__xpatch_bizarre_tl
\token_to_str:N \@protected@testopt \c_space_tl
\c_backslash_str \l__xpatch_name_tl
\c_space_tl \c_backslash_str \c_backslash_str
}
\tl_put_left:Nn \l__xpatch_name_tl { \c_backslash_str }
}\tl_if_in:NxT \l__xpatch_repl_tl
{\tl_set:Nx \l__xpatch_repl_tl
{ \c__xpatch_bizarre_tl
\token_to_str:N \@protected@testopt \c_space_tl
\c_backslash_str \l__xpatch_name_tl
\c_space_tl \c_backslash_str \c_backslash_str
}
\tl_put_left:Nn \l__xpatch_name_tl { \c_backslash_str }
That's the last operation!
Finally, we pass the real command-to-patch name to the patching macro.
\exp_after:wN \l__xpatch_name_tl \cs_end:
\NewDocumentCommand{\xpatchcmd}{}{\xpatch_main:NN \patchcmd }
\NewDocumentCommand{\xpretocmd}{}{\xpatch_main:NN \pretocmd }
\NewDocumentCommand{\xapptocmd}{}{\xpatch_main:NN \apptocmd }
\NewDocumentCommand{\xshowcmd}{}{\xpatch_main:NN \show }
We generate a variant of \xpatch_main:NN to accept a macro's name as its second argument.
\cs_generate_variant:Nn \xpatch_main:NN { Nc }
Now we can define the patching macros for \newbibmacro defined commands. In case one uses a wrong name, it will remain in the hash space, but it shouldn’t be a problem: \tracingpatches must be used when testing, and it will warn about an undefined macro or one equivalent to \relax.
\NewDocumentCommand{\xpatchbibmacro} { m }{\xpatch_main:Nc \patchcmd { abx@macro@ \tl_to_str:n {#1} } }
\NewDocumentCommand{\xpretobibmacro} { m }{\xpatch_main:Nc \pretocmd { abx@macro@ \tl_to_str:n {#1} } }
\NewDocumentCommand{\xapptobibmacro} { m }{\xpatch_main:Nc \apptocmd { abx@macro@ \tl_to_str:n {#1} } }
\NewDocumentCommand{\xshowbibmacro} { m }{\xpatch_main:Nc \show { abx@macro@ \tl_to_str:n {#1} } }
The macros for patching commands defined with \DeclareFieldFormat; all that holds for the preceding commands is valid also for the following groups of similar commands.
\NewDocumentCommand{\xpatchfieldformat} { O{*} m }{\xpatch_main:Nc \patchcmd { abx@ffd@ \tl_to_str:n {#1} \tl_to_str:n {#2} } }
\NewDocumentCommand{\xpretffieldformat} { O{*} m }{\xpatch_main:Nc \pretocmd { abx@ffd@ \tl_to_str:n {#1} \tl_to_str:n {#2} } }
\NewDocumentCommand{\xapptffieldformat} { O{*} m }{\xpatch_main:Nc \apptocmd { abx@ffd@ \tl_to_str:n {#1} \tl_to_str:n {#2} } }
\NewDocumentCommand{\xshowffieldformat} { O{*} m }{\xpatch_main:Nc \show { abx@ffd@ \tl_to_str:n {#1} \tl_to_str:n {#2} } }
\NewDocumentCommand{\xshowfieldformat}{O{*}m}{\xpatch_main:Nc \show{abx@ffd@\tl_to_str:n {#1}@\tl_to_str:n {#2}}}\DeclareNameFormat:\NewDocumentCommand{\xpatchnameformat}{O{*}m}{\xpatch_main:Nc \patchcmd{abx@nfd@\tl_to_str:n {#1}@\tl_to_str:n {#2}}}\NewDocumentCommand{\xpretonameformat}{O{*}m}{\xpatch_main:Nc \pretocmd{abx@nfd@\tl_to_str:n {#1}@\tl_to_str:n {#2}}}\NewDocumentCommand{\xapptonameformat}{O{*}m}{\xpatch_main:Nc \apptocmd{abx@nfd@\tl_to_str:n {#1}@\tl_to_str:n {#2}}}\NewDocumentCommand{\xshownameformat}{O{*}m}{\xpatch_main:Nc \show{abx@ffd@\tl_to_str:n {#1}@\tl_to_str:n {#2}}}\DeclareListFormat:\NewDocumentCommand{\xpatchlistformat}{O{*}m}{\xpatch_main:Nc \patchcmd{abx@lfd@\tl_to_str:n {#1}@\tl_to_str:n {#2}}}\NewDocumentCommand{\xpretolistformat}{O{*}m}{\xpatch_main:Nc \pretocmd{abx@lfd@\tl_to_str:n {#1}@\tl_to_str:n {#2}}}\NewDocumentCommand{\xapptolistformat}{O{*}m}{\xpatch_main:Nc \apptocmd{abx@lfd@\tl_to_str:n {#1}@\tl_to_str:n {#2}}}\NewDocumentCommand{\xshowlistformat}{O{*}m}{\xpatch_main:Nc \show{abx@lfd@\tl_to_str:n {#1}@\tl_to_str:n {#2}}}\DeclareIndexFieldFormat:\NewDocumentCommand{\xpatchindexfieldformat}{O{*}m}{\xpatch_main:Nc \patchcmd{abx@fid@\tl_to_str:n {#1}@\tl_to_str:n {#2}}}\NewDocumentCommand{\xpretoindexfieldformat}{O{*}m}{\xpatch_main:Nc \pretocmd{abx@fid@\tl_to_str:n {#1}@\tl_to_str:n {#2}}}\NewDocumentCommand{\xapptoindexfieldformat}{O{*}m}{\xpatch_main:Nc \apptocmd{abx@fid@\tl_to_str:n {#1}@\tl_to_str:n {#2}}}\NewDocumentCommand{\xshowindexfieldformat}{O{*}m}{\xpatch_main:Nc \show{abx@fid@\tl_to_str:n {#1}@\tl_to_str:n {#2}}}\DeclareIndexNameFormat:\NewDocumentCommand{\xpatchindexnameformat}{O{*}m}{\xpatch_main:Nc \patchcmd{abx@nid@\tl_to_str:n {#1}@\tl_to_str:n {#2}}}\NewDocumentCommand{\xpretoindexnameformat}{O{*}m}{\xpatch_main:Nc \pretocmd{abx@nid@\tl_to_str:n {#1}@\tl_to_str:n {#2}}}\NewDocumentCommand{\xapptoindexnameformat}{O{*}m}{\xpatch_main:Nc \apptocmd{abx@nid@\tl_to_str:n {#1}@\tl_to_str:n {#2}}}\NewDocumentCommand{\xshowindexnameformat}{O{*}m}{\xpatch_main:Nc \show{abx@nid@\tl_to_str:n {#1}@\tl_to_str:n {#2}}}\DeclareIndexListFormat:\NewDocumentCommand{\xpatchindexlistformat}{O{*}m}{\xpatch_main:Nc \patchcmd{abx@lid@\tl_to_str:n {#1}@\tl_to_str:n {#2}}}\NewDocumentCommand{\xpretoindexlistformat}{O{*}m}{\xpatch_main:Nc \pretocmd{abx@lid@\tl_to_str:n {#1}@\tl_to_str:n {#2}}}\NewDocumentCommand{\xapptoindexlistformat}{O{*}m}{\xpatch_main:Nc \apptocmd{abx@lid@\tl_to_str:n {#1}@\tl_to_str:n {#2}}}\NewDocumentCommand{\xshowindexlistformat}{O{*}m}{\xpatch_main:Nc \show{abx@lid@\tl_to_str:n {#1}@\tl_to_str:n {#2}}}Finally, the patching macros for \textit{biblatex} drivers that don’t need the overhead of \texttt{\xpatch_main:NN}.
The italic numbers denote the pages where the corresponding entry is described, numbers underlined point to the definition, all others indicate the places where it is used.

### Symbols

- \@ifpackagelater .......................... 3
- \@protected@testopt .......................... 52, 62, 87
- \@testopt .................................. 72
- \if  ........................................ 9, 18
- \apptocmd .................................. 99, 107, 115, 123, 131, 139, 147, 155, 163
- \bool_if:NT .................................. 78
- \bool_new:N .................................. 16
- \bool_set_false:N ............................ 25
- \bool_set_true:N ............................... 36, 46
- \c__xpatch_bizarre_tl ...................... 19, 28, 31, 41, 51, 61, 71, 81, 86
- \c_backslash_str ........................... 17, 18, 33, 43, 53, 54, 57, 63, 64, 67, 73, 76, 88, 89, 92
- \c_space_tl .................................. 32, 33, 37, 42, 47, 52, 54, 62, 72, 87, 89
- \cs:w ......................................... 83, 95
- \cs_end: ...................................... 83, 95
- \cs_generate_variant:Nn ..................... 15, 101
- \cs_if_exist:NF ............................... 17
- \cs_new:Npn .................................. 23
- \cs_to_str:N .................................. 18, 26
- \exp_after:wN ................................ 82, 95
- \exp_args:Nc .................................. 159, 161, 163, 165
- \ExplFileDate ................................ 2
- \ExplFileDescription ....................... 2
- \ExplFileDescription ....................... 2
- \ExplFileVersion ......................... 2
- \tl_const:Nx .................................. 18, 19
- \tl_if_in:Nx .................................. 15
- \tl_if_in:Nxt ................................ 29, 39, 49, 59, 69, 84
- \tl_new:N ..................................... 21, 22
- \tl_put_left:Nn ................................ 57, 67, 76, 92
- \l__xpatch_name_tl ......................... 21, 26, 33, 37, 43, 47, 53, 57, 63, 67, 73, 76, 83, 88, 92, 95
- \l__xpatch_protect_bool ................... 16, 25, 36, 46, 78
- \l__xpatch_repl_tl ......................... 
  ........................................... 22, 27, 29, 39, 49, 59, 69, 80, 84
- \PackageError ................................ 6
- \patchcmd .................................... 97, 103, 111, 127, 135, 143, 151, 159
- \pretocmd .................................... 98, 105, 113, 121, 129, 137, 145, 153, 161
- \protect ...................................... 32
- \ProvidesExplPackage ....................... 1
- \RequirePackage .............................. 14
- \show ......................................... 100, 109, 117, 125, 133, 141, 149, 157, 165
- \tex_endinput:D .............................. 12
- \tl_const:N .................................. 18, 19
- \tl_if_in:N .................................. 15
- \tl_if_in:Nxt ................................ 29, 39, 49, 59, 69, 84
- \tl_new:N ..................................... 21, 22
- \tl_put_left:Nn ................................ 57, 67, 76, 92
Change History

v0.2
General: Added \x...bibdriver macros; fixed a bug for control symbols defined with \newcommand and an optional argument. 1

v0.3
General: Added a bunch of biblatex related commands 1