The keycommand package

Florent Chervet <florent.chervet@free.fr>

July 22, 2009

Abstract

keycommand provides an easy way to define commands or environments with optional keys. It provides \newkeycommand and its relative \renewkeycommand, \newkeyenvironment, \renewkeyenvironment and \providekeycommand.
Moreover it is possible to define key-commands using \def, \edef, \gdef or \xdef via the \keycmd prefix.
This package requires and is based on the package kvsetkeys by Heiko Oberdiek.
It is designed to work with ε-TEX for the code uses the primives \unexpanded and \protected.

Contents

1 Introduction

1.1 User interface

With keycommand it becomes very easy to define commands with optional keys. Just say:

```
\newkeycommand\CommandWithKeys[kOne=default,...][2]{%
                definition with \commandkey{kOne} etc. #1 and #2}
```

As far as the keys are optional, it is not allowed to have another optional parameter in a key-command.

This documentation is produced with the DocStrip utility.

⟶ To get the documentation, run (thrice): `pdflatex keycommand.dtx`
⟶ To get the package, run: `etex keycommand.dtx`

keycommand enables us to define key-environments as well, and provides:

	`\newkeycommand`	`\renewkeycommand`
	`\newkeyenvironment`	`\renewkeyenvironment`
and:	`\providekeycommand`	

Moreover, if you need (or prefer) the syntax of `\def` (or `\gdef`, `\edef`, `\xdef`) you shall refer to the section The `\keycmd` prefix (in Implementation).

`\newkeycommand` {⟨*control sequence*⟩} [⟨*key-value list*⟩] [⟨*number of args*⟩] {⟨*definition*⟩}

keycommand allow LaTeX users to define commands with key in a easy way. Better is a small example than a long talking: let's define a command `\Rule` whose `width`, `thickness` and `raise` can be specified as keys.

With keycommand we just have to say:

```
\newkeycommand\Rule[raise=.4ex,width=1em,thick=.4pt][1]{%
    \rule[\commandkey{raise}]{\commandkey{width}}{\commandkey{thick}}
    #1%
    \rule[\commandkey{raise}]{\commandkey{width}}{\commandkey{thick}}}
```

which defines the keys `width`, `thick` and `raise` with their default values (if not specified): `1em`, `.4pt` and `.4ex`. Now `\Rule` can be used as follow:

```
1:  \Rule[width=2em]{hello}            ⟶ width=2em,thick=.4pt,raise=.4ex
2:  \Rule[thick=1pt,width=2em]{hello}  ⟶ width=2em,thick=1pt,raise=.4ex
3:  \Rule{hello}                       ⟶ width=1em,thick=.4pt,raise=.4ex
4:  \Rule[thick=2pt,raise=1ex]{hello}  ⟶ width=1em,thick=2pt,raise=1ex
```

et cætera.

They will produce:

1: ——hello——
2: ——hello——
3: —hello—
4: ▬hello▬

`\newkeyenvironment` {⟨*envir name*⟩} [⟨*key-values pairs*⟩] [⟨*number of args*⟩] {⟨*begin*⟩} {⟨*end*⟩}

In the same way, you may define environments with optional keys as follow:

```
\newkeyenvironment{EnvirWithKeys}[kOne=default value,...][n]
                  { commands at begin EnvirWithKeys }
                  { commands at end EnvirWithKeys }
```

Where n is the number of mandatory other arguments (*ie* without keys), if any.

A example of a key-environment is left in the file: `keycommand-example.tex`.

1.2 Error messages

If you use the command `\Rule` (defined in 1.1) with a key say: `height` that has not been declared at the definition of the key-command, you will get an error message like this:

```
There was no key ``height''
in the keycommand \Rule!
see the definition of the keycommand.
```

However, if you use `\commandkey{height}` **in the definition** of `\Rule` you will not have any error message: `\commandkey{height}` will just be expanded into `\relax` at `\Rule` expansion time.

To be honest, when you redefine a key-command using `\renewkeycommand` or `\renewkeyenvironment` or `\keycmd\def` the keys defined before for the old command are undefined. This way you have the expected error message in all cases.

⋆ ⋆
⋆

2 Implementation

2.1 Identification

This package is intended to use with LaTeX so we don't check if it is loaded twice.

```
1 ⟨*package⟩
2 \NeedsTeXFormat{LaTeX2e}% LaTeX 2.09 can't be used (nor non-LaTeX)
3    [2005/12/01]% LaTeX must be 2005/12/01 or younger (see kvsetkeys.dtx).
4 \ProvidesPackage{keycommand}
5    [2009/07/22 an easy way to define commands with keys]
```

2.2 Requirements

The package is based on kvsetkeys. kvsetkeys is more reliable than keyval as far as spaces and bracket (groups) are concerned. Please refer to the kvsetkeys documentation for more information.

As long as we use ε-TeX primitives in keycommand we also load the etex package in order to get an error message if ε-TeX is not running.

```
6 \RequirePackage{etex,kvsetkeys}
```

2.3 Defining keys

`\kcmd@keydef` To handle the case where the key-command was defined as `\global`, we have to define keys globally too. Therefore, we can't use the `\define@key` macro of the keyval package.

```
7 \def\kcmd@keydef#1#2#3#4#5{% #1=\global(eventually), #2=command, #3=family, #4=key, #5=def
8    #1\expandafter\edef\csname kcmd@keys\string#2\endcsname{%
9       \csname kcmd@keys\string#2\endcsname,#4}%
10   #1\@namedef{KV@#3@#4@default\expandafter}\expandafter{%
11      \csname KV@#3@#4\endcsname{#5}}%
12   #1\@namedef{KV@#3@#4}##1}
```

`\kcmd@definekey` In order to define keys, we will use the `\kv@parse` macro (kvsetkeys). Therefore, the only requirement is to define the *processor*.

```
13 \def\kcmd@definekey#1#2#3#4#5{%
14   \begingroup\edef\@tempa{\endgroup
15      \unexpanded{\kcmd@keydef{#1}{#2}{#3}{#4}{#5}}{\def
16            \expandafter\noexpand\csname #3@#4\endcsname{####1}}%
17   }\@tempa}
```

`\kcmd@undefinekeys` Now in case we redefine a key-command, we would like the old keys (*ie* the keys associated to the old definition of the command) to be cleared, undefined. That's the job of `\kcmd@undefinekeys`:

```
18 \def\kcmd@undefinekeys#1#2{%
19   \@ifundefined{kcmd@keys\string#2}
20      \relax
21      {\expandafter\@for\expandafter\kcmp@temp
22         \expandafter:\expandafter=\csname kcmd@keys\string#2\endcsname
23      \do{#1\expandafter\let
```

```
            \csname KV@kcmd@\expandafter\@gobble\string #2@\kcmp@temp @default\endcsname
            \@undefined
         #1\expandafter\let
            \csname KV@kcmd@\expandafter\@gobble\string #2@\kcmp@temp\endcsname
            \@undefined}}%
   #1\@namedef{kcmd@keys\string#2}{\@gobble}}
```

2.4 The \keycmd prefix

`\keycmd` acts just like a (expandable) prefix for `\def` or `\edef`:

The syntax is:

`\keycmd`*	* optional
(`\protected`|`\long`|`\outer`|`\global`)	optional
(`\def`|`\gdef`|`\edef`|`\xdef`)	required
`\CommandWithKeys`	Name of the command to define
`[key=value pairs]`	keys and default values
parameter string	optional (same syntax as `\def`)
{ *replacement text* }	required

Without the star form, `\long` is assumed; but it can always be specified as `\long` **after** `\keycmd`. Example:

```
\keycmd\gdef\CommandWithKeys[kOne=defOne,kTwo=defTwo]#1#2{...}
```

`\keycmd` First we have to read the prefixes, if any:

```
\DeclareRobustCommand\keycmd{\@star@or@long\kcmd@prefix}
```

`\kcmd@prefix` This macro reads the prefixes one after another (including the `\def` word) and stores them in `\kcmd@prfx`. We open a group for all declarations will be local until the final definition of `\CommandWithKeys`.

```
\def\kcmd@prefix{\begingroup
   \let\kcmd@gbl\@empty
   \def\kcmd@prfx{\l@ngrel@x}%
   \futurelet\x\kcmd@pref@x}
\def\kcmd@pref@x{%
   \ifx\x\@sptoken         \let\next\kcmd@pref@x@space
   \else                   \let\next\kcmd@addto@prfx
      \ifx\x\long
      \else\ifx\x\outer
      \else\ifx\x\protected
      \else\ifx\x\global   \def\kcmd@gbl{\global}%
      \else
         \def\kcmd@pref@x@next{\expandafter\key@cmd\noexpand}%
         \ifx\x\def
         \else\ifx\x\edef
         \else\ifx\x\gdef  \def\kcmd@gbl{\global}%
         \else\ifx\x\xdef  \def\kcmd@gbl{\global}%
         \else \def\next{\kcmd@error{a \string\def\space
                     was expected after \string\keycmd.}}%
         \fi\fi\fi\fi
      \fi\fi\fi\fi
   \fi\next}
\begingroup\def\:{\kcmd@pref@x@space}
\expandafter\gdef\: {\futurelet\x\kcmd@pref@x}
\endgroup
\def\kcmd@addto@prfx#1{\expandafter\def\expandafter
   \kcmd@prfx\expandafter{\kcmd@prfx#1}\kcmd@pref@x@next}
\def\kcmd@pref@x@next{\futurelet\x\kcmd@pref@x}
\def\kcmd@error#1{\@latex@error{#1}\@ehd}
```

`\key@cmd, \@keycmd` `\key@cmd` will take the name of the command to be defined as its first argument and checks if there are keys-values placed between brackets just after. Then, `\@keycmd`

will check if the command is definable; if it is not, then the switch \@tempswa is set to false: the definition is processed nevertheless with a basic \def, but the group (opened in \kcmd@prefix) is closed just after the assignment, canceling everything out.

```
\def\key@cmd#1{\@testopt{\expandafter\@keycmd\noexpand#1}{}}
\def\@keycmd#1[#2]{\@tempswafalse\expandafter
   \@rc@ifdefinable\noexpand#1{\@tempswatrue}%
   \if@tempswa
      \let#1=\relax
      \def\next{\kcmd@def#1{#2}}%
   \else \def\next{\afterassignment\endgroup
      \def\kcmd@notdefinable}%
   \fi\next}
```

\kcmd@relaxify We temporarily assign the value \relax to some commands in order to avoid so many \noexpand during the expanded definition of \kcmd@def@:

```
\def\kcmd@relaxify{%
   \let\commandkey\relax
   \let\kvsetkeys\relax
   \let\kv@parse\relax
   \let\@testopt\relax
   \let\kv@set@family@handler\relax
   \let\kcmd@undefinekeys\relax
   \let\kcmd@keyerr\relax
   \let\kcmd@definekey\relax
   \def\"##1"{\expandafter\noexpand\csname ##1\endcsname}}
```

\kcmd@def \kcmd@def will define the keys and the command itself:

```
\def\kcmd@def#1#2{% #1=\Command, #2=key-values
   \edef\kcmd@fam{kcmd@\expandafter\@gobble\string#1}%
   \kcmd@relaxify
   \edef\kcmd@def@##1{\endgroup
      \kv@set@family@handler{\kcmd@fam}{\kcmd@keyerr{#1}{####1}{####2}}%
      \kcmd@undefinekeys{\kcmd@gbl}{#1}%
      \kv@parse{##1}{\kcmd@definekey{\kcmd@gbl}{#1}{\kcmd@fam}}%
      \kcmd@gbl\protected\def#1{% entry point
         \protected\def\commandkey########1{\noexpand\csname\kcmd@fam @########1\endcsname}%
         \def\"kcmd\string#1"########1[########2]{%
               \kvsetkeys{\kcmd@fam}{########1,########2}%
               \"\string #1"}%
         \@testopt{\"kcmd\string#1"{##1}}{}}%
      \let\commandkey\relax
      \expandafter\expandafter\expandafter
         \expandafter\expandafter\expandafter
            \expandafter\kcmd@prfx\"\string#1"%
   }\kcmd@def@{#2}}
```

\kcmd@keyerr \kcmd@keyerr is the default handler for key-commands. It is called whenever the user wants to use a key that was not defined in the key-command:

```
\def\kcmd@keyerr#1#2#3{%
   \let\wheremsg\@empty
   \ifdefined\trcg@where\trcg@where{#1}\fi
   \@latex@error{There was no key "#2" \MessageBreak
      in the keycommand \string#1!\MessageBreak
      see the definition of the keycommand (or environment)\wheremsg}\@ehd}
```

2.5 new key-commands

\newkeycommand The \expandafter...\noexpand trick is there in case the command to (re-)define had been defined as \outer before...

```
\DeclareRobustCommand\newkeycommand{\@star@or@long
   \expandafter\new@keycommand\noexpand}
\DeclareRobustCommand\renewkeycommand{\@star@or@long
   \expandafter\renew@keycommand\noexpand}
\DeclareRobustCommand\providekeycommand{\@star@or@long
   \expandafter\provide@keycommand\noexpand}

\def\new@keycommand#1{\@testopt{\expandafter\@newkeycommand\noexpand#1}{}}
\def\@newkeycommand#1[#2]{\begingroup
   \@tempswafalse\expandafter
   \@ifdefinable\noexpand#1{\@tempswatrue}%
   \if@tempswa
      \let#1=\relax
      \let\kcmd@gbl\@empty
      \def\kcmd@prfx##1{\unexpanded{\@testopt{\@argdef{##1}}0}}%
      \def\next{\kcmd@def#1{#2}}%
   \else \def\next{\afterassignment\endgroup
      \def\kcmd@notdefinable}%
   \fi\next}

\def\renew@keycommand#1{\begingroup
   \escapechar\m@ne\edef\@gtempa{{\string#1}}%
   \expandafter\@ifundefined\@gtempa
      {\endgroup\@latex@error{\noexpand#1undefined}\@ehc}
      \endgroup
   \let\@ifdefinable\@rc@ifdefinable
   \expandafter\new@keycommand\noexpand#1}

\def\provide@keycommand#1{\begingroup
   \escapechar\m@ne\edef\@gtempa{{\string#1}}%
   \expandafter\@ifundefined\@gtempa
      {\endgroup\new@keycommand#1}
      {\endgroup\let\kcmd@notdefinable\noexpand
      \renew@keycommand\kcmd@notdefinable}}
```

2.6 new key-environments

```
\DeclareRobustCommand\newkeyenvironment{\@star@or@long\new@keyenvironment}
\DeclareRobustCommand\renewkeyenvironment{\@star@or@long\renew@keyenvironment}

\def\new@keyenvironment#1{\@testopt{\@newkeyenva{#1}}{}}
\def\@newkeyenva#1[#2]{%
   \kernel@ifnextchar [{\@newkeyenvb{#1}[{#2}]}{\@newkeyenv{#1}{[{#2}][0]}}}
\def\@newkeyenvb#1[#2][#3]{\@newkeyenv{#1}{[{#2}][#3]}}
\def\@newkeyenv#1#2#3#4{%
   \@ifundefined{#1}%
      {\expandafter\let\csname #1\expandafter\endcsname
            \csname end#1\endcsname}%
      \relax
   \expandafter\@newkeycommand
      \csname #1\endcsname#2{#3}%
   \l@ngrel@x\expandafter\def\csname end#1\endcsname{#4}}

\def\renew@keyenvironment#1{%
  \@ifundefined{#1}%
     {\@latex@error{Environment #1 undefined}\@ehc
     }\relax
  \expandafter\let\csname#1\endcsname\relax
  \expandafter\let\csname\expandafter\string\csname #1\endcsname\endcsname\relax
  \expandafter\let\csname end#1\endcsname\relax
  \new@keyenvironment{#1}}
```

2.7 kvsetkeys correction

`\kv@normalize` inserts an unwilling blank space due to a small error in the code. Mr Oberdiek has been asked for correction...

```
\def\kv@normalize#1{%
  \begingroup
    \toks@{,#1,}%
    \KVS@Comma
    \KVS@SpaceComma{ }%
    \KVS@CommaSpace
    \KVS@CommaComma
    \KVS@Equals
    \KVS@SpaceEquals{ }%
    \KVS@EqualsSpace% <original code>\KVS@EqualsSpace{ }</original code>
    \xdef\KVS@Global{\the\toks@}%
  \endgroup
  \let\kv@list\KVS@Global
}
```

⟨/package⟩

3 Example

```
⟨*example⟩
\ProvidesFile{keycommand-example}
\documentclass{article}
\usepackage[T1]{fontenc}
\usepackage[latin1]{inputenc}
\usepackage[american]{babel}
\usepackage{keycommand}
\usepackage{framed}
%
\makeatletter
\parindent\z@
\newkeycommand\Rule[raise=.4ex,width=1em,thick=.4pt][1]{%
   \rule[\commandkey{raise}]{\commandkey{width}}{\commandkey{thick}}%
   #1%
   \rule[\commandkey{raise}]{\commandkey{width}}{\commandkey{thick}}}

\newkeycommand\charleads[sep=1][2]{%
   \ifhmode\else\leavevmode\fi\setbox\@tempboxa\hbox{#2}\@tempdima=1.584\wd\@tempboxa%
   \cleaders\hb@xt@\commandkey{sep}\@tempdima{\hss\box\@tempboxa\hss}#1%
   \setbox\@tempboxa\box\voidb@x}
\newcommand\charfill[1][]{\charleads[{#1}]{\hfill\kern\z@}}
\newcommand\charfil[1][]{\charleads[{#1}]{\hfil\kern\z@}}
%
\newkeyenvironment{dblruled}[first=.4pt,second=.4pt,sep=1pt,left=\z@]{%
   \def\FrameCommand{%
      \vrule\@width\commandkey{first}%
      \hskip\commandkey{sep}
      \vrule\@width\commandkey{second}%
      \hspace{\commandkey{left}}}%
   \parindent\z@
   \MakeFramed {\advance\hsize-\width \FrameRestore}}
   {\endMakeFramed}
%
\makeatother
\begin{document}
\title{This is {\tt keycommand-example.tex}}
\author{Florent Chervet}
\date{July 22, 2009}
\maketitle

\section{Example of a keycommand : \texttt{\string\Rule}}

\begin{tabular*}\textwidth{rl}
\verb+\Rule[width=2em]{hello}+:&\Rule[width=2em]{hello}\cr
\verb+\Rule[thick=1pt,width=2em]{hello}+:&\Rule[thick=1pt,width=2em]{hello}\cr
\verb+\Rule{hello}+:&\Rule{hello}\cr
```

```
\verb+\Rule[thick=1pt,raise=1ex]{hello}+:&\Rule[thick=1pt,raise=1ex]{hello}
\end{tabular*}

\section{Example of a keycommand : \texttt{\string\charfill}}

\begin{tabular*}\textwidth{rp{.4\textwidth}}
\verb+\charfill{$\star$}+: & \charfill{$\star$}\cr
\verb+\charfill[sep=2]{$\star$}+: & \charfill[sep=2]{$\star$} \\
\verb+\charfill[sep=.7]{\textasteriskcentered}+: & \charfill[sep=.7]{\textasteriskcentered}
\end{tabular*}


\section{Example of a keyenvironment : \texttt{dblruled}}

\verb+\begin{dblruled}+\par
\verb+   test for dblruled key-environment\par+\par
\verb+   test for dblruled key-environment\par+\par
\verb+   test for dblruled key-environment+\par
\verb+\end{dblruled}+

\begin{dblruled}
 test for dblruled key-environment\par
 test for dblruled key-environment\par
 test for dblruled key-environment
\end{dblruled}


\verb+\begin{dblruled}[first=4pt,sep=2pt,second=.6pt,left=.2em]+\par
\verb+   test for dblruled key-environment\par+\par
\verb+   test for dblruled key-environment\par+\par
\verb+   test for dblruled key-environment+\par
\verb+\end{dblruled}+

\begin{dblruled}[first=4pt,sep=2pt,second=.6pt,left=.2em]
 test for dblruled key-environment\par
 test for dblruled key-environment\par
 test for dblruled key-environment
\end{dblruled}


\end{document}
⟨/example⟩
```

4 References

[1] Heiko Oberdiek: *The kvsetkeys package*; 2007/09/29 v1.3; CTAN:macros/latex/contrib/oberdiek/kvsetkeys.dtx.

[2] David Carlisle: *The keyval package*; 1999/03/16 v1.13; CTAN:macros/latex/required/graphics/keyval.dtx.

5 Index

Numbers written in italic refer to the page where the corresponding entry is described; numbers underlined refer to the code line of the definition; numbers in roman refer to the code lines where the entry is used.

Symbols