The keycommand package
an easy way to define commands with optional keys.

Florent Chervet <florent.chervet@free.fr>

2009/08/26 – version 2.ζ

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 primitives \unexpanded and \protected.

Contents

1 Introduction ............................................. 2
  1.1 User interface ...................................... 2
  1.2 Error messages ..................................... 3
  1.3 Test if a key is defined ............................ 3
2 Implementation ........................................... 3
  2.1 Identification ....................................... 3
  2.2 Requirements ....................................... 4
  2.3 Syntactical enhancement ............................ 4
  2.4 Defining keys ....................................... 4
  2.5 The \keycmd prefix .................................. 5
  2.6 new key-commands .................................. 7
  2.7 new key-environments ............................... 7
  2.8 Test if keys are blank ............................. 8
3 Example ................................................. 9
4 History ................................................. 10
  [2009/08/26 v2.z] ...................................... 10
  [2009/08/04 v2.e-] ...................................... 10
  [2009/07/22 v1.0] ...................................... 10
5 References ............................................. 10
6 Index ..................................................... 11

* keycommand: CTAN:macros/latex/contrib/keycommand
This documentation is produced with the DocStrip utility.

→ To get the documentation, run (thrice): pdflatex keycommand.dtx
  To get the index, run: makeindex -s gind.ist keycommand.idx
→ To get the package, run: etex keycommand.dtx

The .dtx file is embedded into this pdf file thank to embedfile by H. Oberdiek.
The keycommand package – an easy way to define commands with optional keys.

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 keycommand.

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 optional keys 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][]{%\rule[\commandkey{raise}]{\commandkey{width}}{\commandkey{thick}}#1%}

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[width=1em]{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___

Nota bene: it is also possible to give a key a default value which is the value of another key. For example:

\newkeycommand\CmdKey[alpha=hello, beta=\commandkey{alpha}]{...}

When called as: \CmdKey[alpha=world], the key beta will then have the same value: world.
The keycommand package – an easy way to define commands with optional keys.

\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 \textit{n} is the number of mandatory other arguments (\textit{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 \texttt{\Rule} (defined in 1.1) with a key say: \texttt{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 \texttt{\Rule}!
see the definition of the keycommand.
```

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

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

1.3 Test if a key is defined

When you define a key command you may let the default value of a key empty. Then, you may wish to expand some commands only if the key has been given by the user (with a non empty value). This can be achieved using the macro \texttt{\ifcommandkey}:

\begin{verbatim}
\ifcommandkey{⟨key name⟩}{⟨commands if key is blank⟩}{⟨commands if key is NOT blank⟩}
\end{verbatim}

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 \texttt{\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 \texttt{\ProvidesPackage{keycommand}}
5 [2009/07/22 v2.e- an easy way to define commands with optional keys]
The keycommand package – an easy way to define commands with optional 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 \texttt{\LaTeX} primitives in keycommand we also load the etex package in order to get an error message if \texttt{\LaTeX} is not running.

\begin{verbatim}
6 \RequirePackage{etex,kvsetkeys}
\end{verbatim}

2.3 Syntactical enhancement

We will define a shortcut for \texttt{\expandafter\noexpand\csname ... \endcsname} all along this package.

\begin{verbatim}
7 \edef\kcmd@AtEnd{\catcode34 \the\catcode34} % "
8 \catcode34 4
\end{verbatim}

2.4 Defining keys

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

9 \def\kcmd@keydef#1#2#3#4#5{% #1=\global, #2=command, #3=family, #4=key, #5=def
10 \#1\expandafter\edef\csname kcmd@keys\string#2\endcsname{%
11 \csname kcmd@keys\string#2\endcsname,#4}%
12 \#1\@namedef{KV@#3@#4@default\expandafter}\expandafter{%
13 \csname KV@#3@#4\endcsname{#5}}%
14 \#1\@namedef{KV@#3@#4}##1}
\end{verbatim}

\begin{verbatim}
\kcmd@definekey In order to define keys, we will use the \texttt{\kv@parse} macro (kvsetkeys). Therefore, the only requirement is to define the \texttt{processor}.

15 \def\kcmd@definekey#1#2#3#4#5{%
16 \begingroup\edef\@tempa{\endgroup
17 \unexpanded{\kcmd@keydef#1##2#3#4#5}\def
18 \expandafter\noexpand\csname #3@#4\endcsname #5}{\def
19 }\@tempa}
\end{verbatim}

\begin{verbatim}
\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 \texttt{\kcmd@undefinekeys}:

20 \def\kcmd@undefinekeys#1#2{%
21 \@ifundefined{\kcmd@keys\string#2}{%\relax
22 {\\\expandafter\for\expandafter\icmp@temp
23 \expandafter:\expandafter=\csname kcmd@keys\string#2\endcsname
24 \do{\#1}\expandafter\let
25 \csname KV@kcmd@\expandafter\@gobble\string#2@default\endcsname
26 \@undefined
27 \icmp@temp
28 \#1\expandafter\let
29 \csname KV@kcmd@\expandafter\@gobble\string#2@default\icmp@temp\endcsname
30 \@undefined}%
31 \#1\@namedef{\kcmd@keys\string#2}{\@gobble}}
\end{verbatim}
The keycommand package – an easy way to define commands with optional keys.

2.5 The \keycmd prefix

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

The syntax is:

\begin{center}
\begin{tabular}{|l|l|}
\hline
\texttt{\keycmd} & \texttt{\keycmd} \parbox{5cm}{\centering
possibly \hspace{1em} \begin{itemize}
\item \texttt{\long}\hspace{1em} \texttt{\global}\hspace{1em} \texttt{\protected}\hspace{1em} \texttt{\outer}
\item \texttt{\def}\hspace{1em} \texttt{\edef}\hspace{1em} \texttt{\gdef}\hspace{1em} \texttt{\xdef}
\end{itemize}
control sequence
\texttt{\{key=value pairs\}}
Parameter string
\texttt{\langle Replacement text \rangle}
\end{tabular}
\texttt{\keycmd}\parbox{5cm}{\centering
\parbox{5cm}{\centering
\begin{itemize}
\item \texttt{\long} \hspace{1em} \texttt{\global} \hspace{1em} \texttt{\protected} \hspace{1em} \texttt{\outer}
\item \texttt{\long} \hspace{1em} \texttt{\global} \hspace{1em} \texttt{\protected} \hspace{1em} \texttt{\outer}
\end{itemize}
optional (zero or more)
required: see below
keys and default values
optional
required
}}
\hline
\end{tabular}
\end{center}

Without the star form, \texttt{\long} is assumed; but it can always be specified as \texttt{\long after \keycmd}.

Example:

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

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

\begin{verbatim}
\DeclareRobustCommand\keycmd{\@star@or@long\kcmd@prefix}
\end{verbatim}

\keycmd This is the prefixes scanner: this macro reads the prefixes one after another (including the \texttt{\def} word) and stores them in \texttt{\kcmd@prfx}. We open a group for all declarations will be local until the final definition of \texttt{\CommandWithKeys}.

\begin{verbatim}
\def\kcmd@prefix{\begingroup
\let\kcmd@gbl\@empty
\def\kcmd@prfx{\l@ngrel@x}\
futurelet\x\kcmd@@prefix}
\def\kcmd@@prefix{\let\kcmd@next@addto\kcmd@next@prefix
\ifx\x\@sptoken \let\next\kcmd@space@prefix
\else \let\next\kcmd@addto@prfx
\ifx\x\long
\else\ifx\x\outer
\else\ifx\x\protected
\def\kcmd@gbl{\global}%
\else
\def\kcmd@next@addto{\expandafter\key@cmd\noexpand}%
\ifx\x\def
\else\ifx\x\edef
\else\ifx\x\gdef \def\kcmd@gbl{\global}%
\else \let\kcmd@next@addto\kcmd@next@prefix
\ifx\y\x\kcmd@error@prefix
\else\let\y\x
\fi
\let\next\kcmd@expand@prefix
\fi\fi\fi\fi
\fi\fi\fi\fi
\fi\fi\fi\fi
\fi\next}
\def\kcmd@next@addto{\expandafter\key@cmd\noexpand}%
\futurelet\x\kcmd@prefix
\let\next\kcmd@addto\kcmd@next@prefix
\gdef\CommandWithKeys[kOne=defOne,kTwo=defTwo]#1#2{
...
}
\end{verbatim}

The keycommand package – an easy way to define commands with optional keys.
The keycommand package – an easy way to define commands with optional keys.

\keycmd, \@keycmd \keycmd 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\keycmd#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
            \"\string #1"%}
        \@testopt{"kcmd\string#1"##1[##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:

\[def\kcmd@keyerr#1#2#3{%
    \let\wheremsg=\@empty
\]
The keycommand package – an easy way to define commands with optional keys.

\newkeycommand

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

\newcommand\newkeycommand{\@star@or@long}
\newcommand\renewkeycommand{\@star@or@long}
\newcommand\providekeycommand{\@star@or@long}
\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}}
\def\provide@keycommand#1{\begingroup
\escapechar\m@ne\edef\@gtempa{{\string#1}}%
\expandafter\@ifundefined\@gtempa{\endgroup\new@keycommand#1}{\endgroup\let\kcmd@notdefinable
\renew@keycommand\kcmd@notdefinable}}

\newkeyenvironment

\newcommand\newkeyenvironment{\@star@or@long}
\newcommand\renewkeyenvironment{\@star@or@long}
\def\new@keyenvironment#1{\@testopt{\@newkeyenva{#1}}{}}
\def\@newkeyenva#1[#2]{%\ifnextchar\[\@newkeyenvb{#1}{[\{#2\}[0]\]}\else\@newkeyenv{#1}{[\{#2\][0]\][0]}\fi}
\def\@newkeyenvb#1[#2][#3]{\@newkeyenv{#1}{[\{#2\}[#3]\]}}
\def\@newkeyenv#1#2#3#4[#5]{\@ifundefined{#1}{%\let\kcmd@gbl\@empty\let\kcmd@notdefinable\relax\expandafter\@newkeycommand\csname #1\endcsname#2{#3}}{\@ifundefined{#1}{%\let\kcmd@gbl\@empty\let\kcmd@notdefinable\relax\expandafter\expandafter\expandafter\@newkeyenv\expandafter\csname #1\endcsname\kcmd@notdefinable\noexpand}}

\newkeyenvironment
The keycommand package – an easy way to define commands with optional keys.

2.8 Test if keys are blank

First we need some helper macros:

\newcommand{\kcmd@afterelse}[1]{\else#1\fi}
\def{\kcmd@afterfi}[1]{\fi#1}
\newcommand{\kcmd@expandonce}[2]{\ifx#1\kcmd@expandonce\unexpanded\expandafter{#2}}
\def{\kcmd@ifblank}[1]{\kcmd@ifblank@#1&&@secondoftwo@firstoftwo:}
\newcommand{\kcmd@AtEnd}[3]{\kcmd@expandonce{\commandkey{#1}}{#3}{#2}}

The following macros comes from the etextools\textsuperscript{1} package (by F. Chervet):

1. etextools: CTAN:macros/latex/contrib/etextools
2. etoolbox: CTAN:macros/latex/contrib/etoolbox
3. url: CTAN:macros/latex/contrib/misc/url.sty
The keycommand package – an easy way to define commands with optional keys.

3 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}
\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@}\\charleads[\{#1]\{\hfill\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 \texttt{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} \\
\verb+\Rule[thick=1pt,width=2em]{hello}+: & \Rule[thick=1pt,width=2em]{hello} \\
\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*}{.4\textwidth}{rp{.4\textwidth}}
\verb+\charfill{$\star$}+: & \charfill{$\star$} \\
\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}}
The keycommand package – an easy way to define commands with optional keys.

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

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

4 History

[2009/08/26 v2.z]

•

[2009/08/04 v2.e-]

• Fix catcode of double quote (") in case user command had a double quote in its name...
• Add History to the documentation file
• Modify the prefixes scanner (it is now the same as the one of \ltxnew\[rev.2.ζ]\). Modify the documentation (KOMA-Script classe)

[2009/07/22 v1.0]

• First version.

5 References


4. \ltxnew: CTAN:macros/latex/contrib/ltxnew
The keycommand package – an easy way to define commands with optional keys.

6 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.

<table>
<thead>
<tr>
<th>Symbols</th>
<th>\kcmd@expandnext</th>
</tr>
</thead>
<tbody>
<tr>
<td>&amp;</td>
<td>182</td>
</tr>
<tr>
<td>@argdef</td>
<td>126</td>
</tr>
<tr>
<td>@ehc</td>
<td>134, 160</td>
</tr>
<tr>
<td>@ehd</td>
<td>69, 112</td>
</tr>
<tr>
<td>@firstoftwo</td>
<td>183</td>
</tr>
<tr>
<td>@ifdefinable</td>
<td>122, 136</td>
</tr>
<tr>
<td>@undefined</td>
<td>21, 133, 140, 151, 159</td>
</tr>
<tr>
<td>@keycmd@expandnext</td>
<td>171, 174, 177</td>
</tr>
<tr>
<td>@newkeycommand</td>
<td>119, 120, 155</td>
</tr>
<tr>
<td>@newkeyenv</td>
<td>148, 149, 150</td>
</tr>
<tr>
<td>@newkeyenva</td>
<td>146, 147</td>
</tr>
<tr>
<td>@newkeyenvb</td>
<td>148, 149</td>
</tr>
<tr>
<td>@rc@ifdefinable</td>
<td>72, 136</td>
</tr>
<tr>
<td>@secondoftwo</td>
<td>183</td>
</tr>
<tr>
<td>@sptoken</td>
<td>39</td>
</tr>
<tr>
<td>@star@or@long</td>
<td>32, 113, 115, 117, 144, 145</td>
</tr>
<tr>
<td>@undefined</td>
<td>27, 30, 61</td>
</tr>
<tr>
<td>\afterassignment</td>
<td>76, 128</td>
</tr>
<tr>
<td>\catcode</td>
<td>7, 8, 182</td>
</tr>
<tr>
<td>\charfil</td>
<td>216</td>
</tr>
<tr>
<td>\charfill</td>
<td>215, 244, 247, 248, 249</td>
</tr>
<tr>
<td>\charleads</td>
<td>211, 215, 216</td>
</tr>
<tr>
<td>\cleaders</td>
<td>213</td>
</tr>
<tr>
<td>\commandkey</td>
<td>80, 97, 102, 190, 207, 209, 213, 220, 221, 222, 223</td>
</tr>
<tr>
<td>\DeclareRobustCommand</td>
<td>32, 113, 115, 117, 144, 145</td>
</tr>
<tr>
<td>\expandnext</td>
<td>168</td>
</tr>
<tr>
<td>\futurelet</td>
<td>36, 59</td>
</tr>
<tr>
<td>\ifcommandkey</td>
<td>3, 186</td>
</tr>
<tr>
<td>\kcmd@prefix</td>
<td>36, 37, 59</td>
</tr>
<tr>
<td>\kcmd@addto@prefix</td>
<td>40, 61</td>
</tr>
<tr>
<td>\kcmd@afterelse</td>
<td>166, 170, 176</td>
</tr>
<tr>
<td>\kcmd@afterfi</td>
<td>167, 172, 179</td>
</tr>
<tr>
<td>\kcmd@at@End</td>
<td>7, 193</td>
</tr>
<tr>
<td>\kcmd@def</td>
<td>75, 89, 127</td>
</tr>
<tr>
<td>\kcmd@def@</td>
<td>92, 106</td>
</tr>
<tr>
<td>\kcmd@definekey</td>
<td>15, 87, 95</td>
</tr>
<tr>
<td>\kcmd@error@prefix</td>
<td>52, 65</td>
</tr>
<tr>
<td>\kcmd@expand@prefix</td>
<td>55, 60</td>
</tr>
</tbody>
</table>

L
\l@ngrel@x | 35, 157 |

M
\meaning | 68 |

N
\new@keycommand | 114, 119, 137, 141 |
\new@keyenvironment | 144, 146, 165 |
\newkeycommand | 2, 113, 206, 211 |
\newkeyenvironment | 3, 144, 218 |

P
\protected | 43, 96 |
\provide@keycommand | 118, 138 |
\providekeycommand | 117 |

R
\renew@keycommand | 116, 131, 143 |
\renew@keyenvironment | 145, 158 |
\renewkeycommand | 115 |
\renewkeyenvironment | 145 |
\Rule | 206, 235, 238, 239, 240, 241 |
\rule | 207, 209 |

U
\unexpanded | 17, 126, 181 |

Y
\y | 52, 53, 61 |