



# The completely redesigned keycommand\* package

## key-value interface for commands and environments in L<sup>A</sup>T<sub>E</sub>X.

<florent.chervet@free.fr>

2010/04/15 – version 3.1

### Abstract

keycommand provides an easy way to define commands or environments with optional keys. `\newkeycommand` and its relatives `\renewkeycommand`, `\newkeyenvironment`, `\renewkeyenvironment` and `\providekeycommand` are defined in this package.

Keys are simply defined while defining the command itself in a very natural way. You can restrict the possible values for the keys by declaring them with a **type**. Available types for keys are : *boolean*, *enum* and *choice*.

The keycommand package requires and is based on the package xkeyval by Hendri Adriaens, and uses the `\kv@normalize` macro of kvsetkeys (Heiko Oberdiek) for robustness, as shown in 2.3).

keycommand is designed to make easier interface for user-defined commands. In particular, `\newkeycommand+` permits the use of key-commands in every context.

It works with an  $\varepsilon$ -T<sub>E</sub>X distribution of L<sup>A</sup>T<sub>E</sub>X.

## Contents

<b>1</b>	<b>User Interface</b>	<b>2</b>
1.1	General syntax	2
1.2	First example :	2
1.3	Second example : the + form	3
1.4	Explanation of the + form	3
1.5	key-environments	4
<b>2</b>	<b>Messages and more</b>	<b>4</b>
2.1	Invalid keys	4
2.2	Testing keys	4
2.3	xkeyval, keyval and kvsetkeys comparison	5
<b>3</b>	<b>Implementation</b>	<b>5</b>
3.1	Identification	5
3.2	Requirements	5
3.3	Defining (and undefining) command-keys	6
3.4	new key-commands	10
3.5	new key-environments	11
3.6	Tests on keys	12
<b>4</b>	<b>Examples</b>	<b>12</b>
<b>5</b>	<b>History</b>	<b>14</b>
	[2010/04/15 v3.1]	14
	[2010/03/28 v3.0]	14
	[2009/07/22 v1.0]	14
<b>6</b>	<b>References</b>	<b>14</b>
<b>7</b>	<b>Index</b>	<b>14</b>

\* keycommand: [CTAN:macros/latex/contrib/keycommand](http://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.

# 1 User Interface

## 1.1 General syntax

<code>\newkeycommand</code>	<code>*+[short-unexpand]</code>	<code>{\command}</code>	<code>[\keys=defaults]</code>	<code>[\OptKey]</code>	<code>[\&lt;n&gt;]</code>	<code>{\definition}</code>
	modifiers Optional	Required		Optional		Required

`\newkeycommand` will define `\command` as a new key-command! well...

Use the `*` form when you do not want it to be a `\long` macro (as for L<sup>A</sup>T<sub>E</sub>X-`\newcommand`).

The `[\keys=defaults]` argument define the keys with their default values. It is optional, but a key-command without keys seems to be useless (at least for me...). Keys may be defined as :

Type	example	value of <code>\commandkey</code>
general	<code>color=red</code>	<code>\commandkey{color}</code> is ‘red’ and may be anything (text, number, macro...)
boolean	<code>bool bold=true</code>	<code>\commandkey{bold}</code> is: 0 (for <i>false</i> ) 1 (for <i>true</i> )
enumerate	<code>enum position={left,centered,right}</code>	<code>\commandkey{position}</code> is: ‘left’ by default and can be ‘centered’ or ‘right’
	<code>enum*position={left,centered,right}</code>	This is the same, except match is case <b>ins</b> ensitive
choice	<code>choice position={left,centered,right}</code>	<code>\commandkey{position}</code> is: 0 (for <i>left</i> the default value), 1 (for <i>centered</i> ) 2 (for <i>right</i> )
	<code>choice*position={left,centered,right}</code>	This is the same, except match is case <b>ins</b> ensitive

The `OptKeys` argument is used if you wish to capture the `key=value` pairs that are not specifically defined (more on this in the examples section 4).

The key-command may have 0 up to 9 **mandatory** arguments : specify the number by `<n>` (0 if omitted).

The `+` form expands the `\commandkey` before executing the key-command itself, as explain in next section.

## 1.2 First example :

```
\newkeycommand\textrule[raise=.4ex,width=3em,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): 3em, .4pt and .4ex. Now `\textrule` can be used as follow:

```
1: \textrule[width=2em]{hello}      →    —hello—
2: \textrule[thick=5pt,width=2em]{hello} →    ■■■hello■■■
3: \textrule{hello}                →    —hello—
4: \textrule[thick=2pt,raise=1ex]{hello} →    —hello—
```

*et cætera.*

### 1.3 Second example : the + form

---

```
\newkeycommand+[\|]\myfigure[image,
    caption,
    enum placement={H,h,b,t,p},
    width=\textwidth,
    label=
][ OtherKeys]{%
|\begin{figure}|[\commandkey{placement}]
|\includegraphics|[width=\commandkey{width},\commandkey{ OtherKeys}]{%
\commandkey{image}}}%
\ifcommandkey{caption}{|\caption|{\commandkey{caption}}}{}%
\ifcommandkey{label}{|\label|{\commandkey{label}}}{}%
|\end{figure}|}
```

---

With the + form of `\newkeycommand`, the definition will be expanded (at run time). The optional `[\|]` argument means that everything inside `| ... |` is protected from expansion.

`\ifcommandkey{<name>}{<true>}{<false>}` expands `<true>` if the commandkey `<name>` is not blank.

`<Otherkeys>` captures the keys given by the user but not declared: they are simply given back to `\includegraphics` here...

### 1.4 Explanation of the + form

The `\commandkey{<name>}` stuff is expanded at run time using the following scheme:

---

```
\newkeycommand\keyMacro[A=\defA,B=\defB,C=\defC,D=\defD][1]{\begingroup
\edef\keyMacro##1{\endgroup
\noexpand\Macro{\getcommandkey{A}}
\getcommandkey{B}}
\getcommandkey{C}}
\getcommandkey{D}}
}\keyMacro{#1}}
```

---

Therefore, the arguments of `\Macro` are ready: there is no more `\commandkey` stuff, but instead the values of the keys as you gave them to the key-command. `\getcommandkey{A}` is expanded to `\defA`.

But `\defA` is not expanded of course: in the + form, `\commandkey` has the meaning of `\getcommandkey`.

As you can see, the mandatory arguments **#1**, **#2** etc. are **never expanded**: there is no need to protect them inside the special (usually `|`) character.

## 1.5 key-environments

`\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.

There is no **+** form for key-environments.

## 2 Messages and more

### 2.1 Invalid keys

If you use the command `\textule` (defined in 1.2) 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:

```
The key-value pairs "height=..."
cannot be processed for key-command \textrule!
See the definition of the keycommand!
```

The error is recoverable: the key is ignored.

If you assign a value to an *enum* or a *choice* key, which is not allowed in the definition, you will get the following message:

```
The value "..." is not allowed in key ...
for key-command \command
I'll use the default value "..." for this key instead
See the definition of the key-command!
```

The error recoverable: the key is assigned its default value.

If you use a `\commandkey{<name>}` in a key-command where  $\langle name \rangle$  is not defined as a key, you will get the T<sub>E</sub>X generic error message :

```
undefined control sequence : \keycmd->...@name.
```

### 2.2 Testing keys

`\ifcommandkey {<key name>} {<commands if key is NOT blank>} {<commands if key is blank>}`

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 `\ifcommandkey`.

## 2.3 xkeyval, keyval and kvsetkeys comparison

xkeyval: macro:->2008/08/13 v2.6a package option processing (HA)

keyval: macro:->1999/03/16 v1.13 key=value parser (DPC)

kvsetkeys: macro:->2010/03/01 v1.9 Key value parser (HO)

Example	keyval	xkeyval	kvsetkeys and keycommand
<code>\setkeys{fam}{key={{value}}}</code>	macro:->value	macro:->value	macro:->{value}
<code>\setkeys{fam}{key={{value}}}</code>	macro:->{value}	macro:->value	macro:->{{value}}
<code>\setkeys{fam}{key=_{{value}}}</code>	macro:->{{value}}	macro:->{value}	macro:->{{value}}

Table 1: Then it is clear that, at this time, kvsetkeys has the only correct behaviour...

In keycommand the key-value pairs are first normalized using kvsetkeys-`\kv@normalize`. Then braces are added around the values in order to keep the good behaviour of kvsetkeys while using xkeyval.

★                      ★  
★

## 3 Implementation

### 3.1 Identification

This package is intended to use with L<sup>A</sup>T<sub>E</sub>X 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   [2010/04/15 v3.1 - key-value interface for commands and environments in LaTeX]
```

### 3.2 Requirements

The package is based on xkeyval. However, this xkeyval is far less reliable than kvsetkeys as far as spaces and bracket (groups) are concerned, as shown in the section 2.3 of this documentation.

Therefore, we also use the macros of kvsetkeys in order to *normalize* the key=value list before setting the keys. This way, we take advantage of both xkeyval and kvsetkeys !

As long as we use  $\varepsilon$ -T<sub>E</sub>X primitives in keycommand we also load the etex package in order to get an error message if  $\varepsilon$ -T<sub>E</sub>X is not running.

The etoolbox package gives some facility to write keycommand and simplify its code much. etextools gives a few other facilities.

```

6 \def\kcmd@pkg@name{keycommand}
7 \RequirePackage{etex,kvsetkeys,xkeyval,etoolbox,etextools}
```

the `\setkeys` of xkeyval package (in case it was overwritten by a subsequent load of kvsetkeys or keyval for example :

```

8 \def\kcmd@Xsetkeys{\XKV@testopta{\XKV@testoptc\XKV@setkeys}}% in case \setkeys was overwritten
9 \@ifpackagelater{etextools}{2009/08/27}\relax
10  {\PackageError\kcmd@pkg@name{keycommand requires an implementation of\MessageBreak
11   package etextools later than 2009/08/27.\MessageBreak
12   Please update your distribution with a recent version of etextools}%
```

```

13 {The keycommand package will not be loaded.}%
14 \expandafter\endinput}

```

Some `\catcode` assertions internally used by keycommand:

```

15 \let\kcmd@AtEnd\@empty
16 \def\TMP@EnsureCode#1#2{%
17   \edef\kcmd@AtEnd{%
18     \kcmd@AtEnd
19     \catcode#1 \the\catcode#1\relax
20   }%
21   \catcode#1 #2\relax
22 }
23 \TMP@EnsureCode{32}{10}% space
24 \TMP@EnsureCode{61}{12}% = sign
25 \TMP@EnsureCode{45}{12}% - sign
26 \TMP@EnsureCode{62}{12}% > sign
27 \TMP@EnsureCode{46}{12}% . dot
28 \TMP@EnsureCode{47}{8}% / slash (etextools)
29 \AtEndOfPackage{\kcmd@AtEnd\undef\kcmd@AtEnd}

```

### 3.3 Defining (and undefining) command-keys

`\kcmd@keyfam` The macro expands to the family-name, given the keycommand name:

```

30 \def\kcmd@keyfam#1{\detokenize{keycmd->}\expandafter\@gobble\string#1}

```

`\kcmd@normalize@setkeys`

This macro assigns the values to the keys (expansion of `xkeyval-\setkeys` on the result of `kvsetkeys-\kv@normalize`). Braces are normalized too so that `key= {{{value}}}` is the same as `key={{{{value}}}}` as explained in section 2.3:

```

31 \newrobustcmd\kcmd@normalize@setkeys[4]{%
32 % #1 = key-command,
33 % #2 = family,
34 % #3 = other-key,
35 % #4 = key-values pairs
36   \kv@normalize{#4}\expandafter\let\expandafter\kv@list\expandafter\@empty
37   \expandafter\kv@parse@normalized\expandafter{\kv@list}{\kcmd@normalize@braces{#2}}%
38   \expandafter\@swaparg\expandafter{\kv@list,}{\kcmd@Xsetkeys*{#2}}%
39   \etl@nbk#3//% undeclared keys are assigned to "OtherKeys"
40   {\csedef{#2@#3}{\expandonce{\XKV@rm}}}% (if specified, ie not empty)
41   {\expandafter\etl@nbk\XKV@rm//% (otherwise a recoverable error is thown)
42     {\PackageError{kcmd@pkg@name}{The key-value pairs : \MessageBreak
43       \XKV@rm \MessageBreak
44       cannot be processed for key-command \string#1 \MessageBreak
45       See the definition of the key-command!}{}}{}}//}%
46 \long\def\kcmd@normalize@braces#1#2#3{% This is kvsetkeys processor for normalizing braces
47   \edef\kv@list{\expandonce{\kv@list,%
48     #2\if @\detokenize{#3}@\else ={{{\unexpanded{#3}}}}\fi}}

```

`\kcmd@definekey`

`\kcmd@definekey` define the keys declared for the key-command. It is used as the *processor* for the `\kv@parse` macro of `kvsetkeys`. The macro appends the key names to the key list: “*family.keylist*”.

keys are first checked for their type (bool, enum, enum\*, choice or choice\*) :

```

49 \def\kcmd@check@typeofkey#1{% expands to
50 % 0 if key has no type,
51 % 1 if boolean,

```

```

52% 2 if enum*,
53% 3 if enum,
54% 4 if choice*,
55% 5 if choice
56 \kcmd@check@typeofkey@bool#1bool // %
57 {\kcmd@check@typeofkey@enumst#1enum* // %
58 {\kcmd@check@typeofkey@enum#1enum // %
59 {\kcmd@check@typeofkey@choicest#1choice* // %
60 {\kcmd@check@typeofkey@choice#1choice // %
61 05//}4//}3//}2//}1//}
62 \def\kcmd@check@typeofkey@bool #1bool #2//{\etl@nbk#1//}
63 \def\kcmd@get@keyname@bool #1bool #2//{#2}
64 \def\kcmd@check@typeofkey@enumst #1enum* #2//{\etl@nbk#1//}
65 \def\kcmd@get@keyname@enumst #1enum* #2//{#2}
66 \def\kcmd@check@typeofkey@enum #1enum #2//{\etl@nbk#1//}
67 \def\kcmd@get@keyname@enum #1enum #2//{#2}
68 \def\kcmd@check@typeofkey@choicest #1choice* #2//{\etl@nbk#1//}
69 \def\kcmd@get@keyname@choicest #1choice* #2//{#2}
70 \def\kcmd@check@typeofkey@choice #1choice #2//{\etl@nbk#1//}
71 \def\kcmd@get@keyname@choice #1choice #2//{#2}
72 %
73 \protected\long\def\kcmd@definekey#1#2#3#4#5{% define the keys using xkeyval macros
74 % #1 = keycommand,
75 % #2 = \global,
76 % #3 = family,
77 % #4 = key (before = sign),
78 % #5 = default (after = sign)
79 \ifcase\kcmd@check@typeofkey{#4}\relax% standard
80 #2\csedef{#3.keylist}{\csname#3.keylist\endcsname,#4}%
81 \define@cmdkey{#3}[{#3@}]{#4}[{#5}]{}%
82 \or% bool
83 #2\csedef{#3.keylist}{\csname#3.keylist\endcsname,\kcmd@get@keyname@bool#4//}%
84 \kcmd@define@boolkey#1{#3}{\kcmd@get@keyname@bool#4//}{#5}%
85 \or% enum*
86 #2\csedef{#3.keylist}{\csname#3.keylist\endcsname,\kcmd@get@keyname@enumst#4//}%
87 \kcmd@define@choicekey#1*{#3}{\kcmd@get@keyname@enumst#4//}{#5}{\expandonce\val}%
88 \or% enum
89 #2\csedef{#3.keylist}{\csname#3.keylist\endcsname,\kcmd@get@keyname@enum#4//}%
90 \kcmd@define@choicekey#1{#3}{\kcmd@get@keyname@enum#4//}{#5}{\expandonce\val}%
91 \or% choice*
92 #2\csedef{#3.keylist}{\csname#3.keylist\endcsname,\kcmd@get@keyname@choicest#4//}%
93 \kcmd@define@choicekey#1*{#3}{\kcmd@get@keyname@choicest#4//}{#5}{\number\nr}%
94 \or% choice
95 #2\csedef{#3.keylist}{\csname#3.keylist\endcsname,\kcmd@get@keyname@choice#4//}%
96 \kcmd@define@choicekey#1{#3}{\kcmd@get@keyname@choice#4//}{#5}{\number\nr}%
97 \fi
98 \ifx#2\global\relax
99 #2\csletcs{KV@#3@#4}{KV@#3@#4}% globalize
100 #2\csletcs{KV@#3@#4@default}{KV@#3@#4@default}% globalize default value
101 \fi}
102 %
103 \long\def\kcmd@firstchoiceof#1,#2\@nil{\unexpanded{#1}}
104 %
105 \long\def\kcmd@define@choicekey#1#2#3#4#5#6{\begingroup\edef\kcmd@define@choicekey{\endgroup
106 \noexpand\define@choicekey#2+{#3}{#4}
107 [\noexpand\val\noexpand\nr]%
108 {\unexpanded{#5}}% list of allowed values
109 [{\kcmd@firstchoiceof#5,\@nil}]% default value
110 {\csedef{#3@#4}{\unexpanded{#6}}}% define key value if in the allowed list
111 {\kcmd@error@handler\noexpand#1{#3}{#4}{\kcmd@firstchoiceof#5,\@nil}}% error handler
112 }\kcmd@define@choicekey}
113 %

```



```

114 \def\kcmd@define@boolkey#1#2#3#4{\begingroup
115   \ettl@nbk#4//{\def\default{#4}}{\def\default{true}}//%
116   \edef\kcmd@define@boolkey{\endgroup
117     \noexpand\define@choicekey*+{#2}{#3}{\noexpand\val\noexpand\nr}%
118       {false,true}
119       [{\expandonce\default}]}%
120       {\csedef{#2@#3}{\noexpand\number\noexpand\nr}}}%
121       {\kcmd@error@handler\noexpand#1{#2}{#3}{\expandonce\default}}}%
122   }\kcmd@define@boolkey}
123 %
124 \protected\long\def\kcmd@error@handler#1#2#3#4{%
125 % #1 = key-command,
126 % #2 = family,
127 % #3 = key,
128 % #4 = default
129   \PackageError\kcmd@pkg@name{%
130     Value ‘\val\space’ is not allowed in key #3\MessageBreak
131     for key-command \string#1.\MessageBreak
132     I’ll use the default value ‘#4’ for this key.\MessageBreak
133     See the definition of the key-command!}%
134   \csdef{#2@#3}{#4}}

```

### `\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`. It uses `\csvloop` (in a `\edef` in `\kcmd@defcommand`).

```

135 \def\kcmd@undefinekeys#1#2{% #1 = global, #2 = family
136   \ifcsundef{#2.keylist}
137     {}
138     {\expandnext{\csvloop[\kcmd@undefinekey{#1}{#2}]{\csname #2.keylist\endcsname}}}%
139   \cslet{#2.keylist}\noexpand\@gobble}
140 \def\kcmd@undefinekey#1#2#3{% #1 = global, #2 = family, #3 = key
141   #1\csundef{KV@#2@#3}%
142   #1\csundef{KV@#2@#3default}%
143 }

```

### `\kcmd@def checks \@ifdefinable` and cancels definition if needed:

```

144 \protected\long\def\kcmd@def#1#2[#3][#4][#5]#6#7{%
145   \@ifdefinable#1{\kcmd@defcommand#1[#3][#4][#5]{#6}{#2}{#7}}

```

`\kcmd@defcommand` prepares (expand) the arguments before closing the group opened at the very beginning. Then it proceeds (`\kcmd@yargdef` (normal interface) or `\kcmd@yargedef` (when `\newkeycommand+` is used))

```

146 \protected\long\def\kcmd@defcommand#1[#2][#3][#4]#5#6#7{%
147   \edef\kcmd@fam{\kcmd@keyfam{#1}}\let\commandkey\relax
148   \edef\kcmd@defcommand{\endgroup
149     \kcmd@undefinekeys{\kcmd@gbl}{\kcmd@fam}% undefines all keys for this keycommand family
150     \kcmd@mount@unexpandchar{\kcmd@fam}{\expandonce\kcmd@unexpandchar}%
151     \unexpanded{\kv@parse{#2,#3}}{\kcmd@definekey\noexpand#1{\kcmd@gbl}{\kcmd@fam}}% defines key
152     \csdef{\kcmd@fam.commandkey}####1{\noexpand\csname\kcmd@fam @###1\endcsname}%
153     \csdef{\kcmd@fam.getcommandkey}####1{%
154       \unexpanded{\unexpanded\expandafter\expandafter\expandafter}{%
155         \noexpand\csname\kcmd@fam @###1\endcsname}}}%
156     \let\commandkey\noexpandcs{\kcmd@fam.\kcmd@plus get\fi commandkey}%
157     \kcmd@plus% \newkeycommand+
158     \csdef{\kcmd@fam}{\kcmd@yargedef{\kcmd@gbl}{\kcmd@long}{\noexpandcs\kcmd@fam}
159       {number#4}{#6}{\csname\kcmd@fam.unexpandchar\endcsname}}}%
160     \unless\ifx#6\relax% that means we have to define a key-environment
161     \def#6{\kcmd@yargedef{\kcmd@gbl}{\kcmd@long}{#6}%
162       0\relax{\csname\kcmd@fam.unexpandchar\endcsname}}}%

```



```

163     \fi
164     \else% \newkeycommand
165         \csdef{\kcmd@fam}{\kcmd@yargdef{\kcmd@gbl}{\kcmd@long}{\noexpandcs\kcmd@fam}
166             {\number#4}{#6}}%
167         \unless\ifx#6\relax% that means we have to define a key-environment
168             \def#6{\kcmd@yargdef{\kcmd@gbl}{\kcmd@long}{#6}\relax}%
169     \fi
170     \fi
171     \kcmd@gbl\protected\edef#1{% entry point
172         \let\noexpand\noexpand\commandkey\noexpand\noexpand\noexpandcs{%
173             \kcmd@fam.\kcmd@plus get\fi commandkey}%
174         \noexpand\csvloop[\noexpand\kcmd@resetdefault{\kcmd@fam}]{\noexpandcs{\kcmd@fam.keylist}}
175         \noexpand\noexpand\noexpand\@testopt{%
176             \kcmd@setkeys\noexpand\noexpand#1{\kcmd@fam}{\kcmd@otherkeys{#3}}}{}}%
177     \noexpandcs\kcmd@fam%
178 } \kcmd@defcommand{#5}{#7}%
179 }
180 \protected\long\def\kcmd@setkeys#1#2#3[#4]{%
181     \kcmd@normalize@setkeys{#1}{#2}{#3}{#4}\csname#2\endcsname}
182 \def\kcmd@resetdefault#1#2{\noexpandcs{KV@#1@#2@default}}
183 \long\def\kcmd@otherkeys#1{\etl@nbk#1//{\kcmd@otherkeys#1=\@nil}}{}}
184 \long\def\kcmd@@otherkeys#1=#2\@nil{#1}

```

`\kcmd@mount@unexpandchar`

This macro defines the macro `\family.unexpandchar`. `\family.unexpandchar` activates the shortcut character for `\unexpanded` and defines its meaning.

```

185 \protected\def\kcmd@mount@unexpandchar#1#2{%
186     \etl@nbk#2//{%
187         \protected\csdef{#1.unexpandchar}{\begingroup
188             \catcode'\~\active \lccode'\~#2 \lccode'#2 0\relax
189             \lowercase{\def~{\catcode'#2\active
190                 \long\def~#####1~{\unexpanded{#####1}}}%
191                 \etl@aftergroup@def~\endgroup~}}}%
192     {\cslet{#1.unexpandchar}\empty}}//%
193 }

```

`\kcmd@yargdef`      This is the `argdef` macro for the normal (non +) form:

```

194 \protected \def \kcmd@yargdef #1#2#3#4#5{\begingroup
195 % #1 = global or {}
196 % #2 = long or {}
197 % #3 = Command
198 % #4 = nr of args
199 % #5 = endenvir (or \relax if not an environment)
200 \def \kcmd@yargd@f ##1#4##2##{\afterassignment#5\endgroup
201     #1#2\expandafter\def\expandafter#3\@gobble ##1#4%
202 } \kcmd@yargd@f 0##1##2##3##4##5##6##7##8##9##4%
203 }

```

`\kcmd@yargedef`      This is the `argdef` macro for the + form:

```

204 \protected\def\kcmd@yargedef#1#2#3#4#5#6{\begingroup
205 % #1 = global or {}
206 % #2 = long or {}
207 % #3 = Command
208 % #4 = nr of args
209 % #5 = endenvir (or \relax if not an environment)
210 % #6 = unexpandchar mounting macro

```

```

211 \letcs\kcmd@nargs{kcmd@#4of9}%
212 \long\def\kcmd@yarg@edef##1##2{\afterassignment#5\endgroup
213   #1\edef#3{\begingroup #6%
214     #2\edef\unexpanded{#3##2}{\endgroup\unexpanded{##1}%
215     }\noexpand#3}%
216 }%
217 \protected\def\kcmd@yarg@body{\edef\body{\unexpanded\expandafter\expandafter\expandafter{%
218   \expandafter#3\kcmd@nargs{####1}{####2}{####3}{####4}{####5}{####6}{####7}{####8}{####9}%
219 }}\expandafter\kcmd@yarg@edef\expandafter{\body}}%
220 \def\kcmd@yarg@def@f##1#4##2##{%
221   \edef\kcmd@yarg@def@next{\kcmd@yarg@body{\expandonce{\@gobble ##1#4}}}%
222   \afterassignment\kcmd@yarg@def@next
223   \expandafter\def\expandafter#3\@gobble ##1#4%
224 }% \kcmd@yarg@def@f 0##1##2##3##4##5##6##7##8##9####4%
225 }

```

`\kcmd@nargs` Filter macros used by `\kcmd@yarg@def` to get the correct number of arguments:

```

226 \def\kcmd@XofNINE#1{%
227   \def\kcmd@XofNine##1#1##2##{%
228     \expandafter\csdef{kcmd@#1of9}#####1####2####3####4####5####6####7####8####9{%
229       \@gobble ##1#1}%
230   }% \kcmd@XofNine 0##1##2##3##4##5##6##7##8##9####1}%
231 }
232 \csvloop*[\kcmd@XofNINE]{1,2,3,4,5,6,7,8,9,0}
233 \undef\kcmd@XofNine\undef\kcmd@XofNINE

```

### 3.4 new key-commands

`\newkeycommand` Here are the entry points:

```

234 \protected\def\newkeycommand{\begingroup
235   \let\kcmd@gb1\@empty\kcmd@modifiers\new@keycommand}
236 \protected\def\renewkeycommand{\begingroup
237   \let\kcmd@gb1\@empty\kcmd@modifiers\renew@keycommand}
238 \protected\def\providekeycommand{\begingroup
239   \let\kcmd@gb1\@empty\kcmd@modifiers\provide@keycommand}

```

`\kcmd@modifiers`

This macro reads the modifiers for `\newkeycommand` (or `\newkeyenvironment`) and save them for later use. The scanning of modifiers is performed by `etextools-\futuredef`:

```

240 \protected\def\kcmd@modifiers#1{%
241   \futuredef[+]\kcmd@modifiers{% only once inside group
242     \edef\kcmd@long{\expandafter\kcmd@ifstar\kcmd@modifiers*\relax}%
243     \edef\kcmd@plus{\expandafter\kcmd@ifplus\kcmd@modifiers+\relax}%
244     \@testopt{\kcmd@unexpandchar#1}{}}
245 \def\kcmd@ifstar#1*#2\relax{\etl@nbk#2//}\long//}
246 \def\kcmd@ifplus#1+#2\relax{\etl@nbk#2//\noexpand\iftrue}\noexpand\iffalse//}

```

`\kcmd@unexpandchar` Reads the possible unexpand-char shortcut:

```

247 \def\kcmd@unexpandchar#1[#2]{\def\kcmd@unexpandchar{#2}% only once inside group...
248   \kcmd@plus \etl@nbk#2//
249   {\def\kcmd@unexpandchar@activate{\catcode'#2 \active}}%
250   {\let\kcmd@unexpandchar@activate\relax}//%
251   \else\etl@nbk#2//%
252   {\PackageError\kcmd@pkg@name{shortcut option for \string\unexpanded\MessageBreak
253     You can't use a shortcut option for \string\unexpanded\MessageBreak
254     without the \string+ form of \string\newkeycommand!}%

```

```

255      {I will ignore this option and proceed.}%
256      \let\kcmd@unexpandchar\@empty}%
257      {}//%
258      \fi#1}

```

`\new@keycommand` Reads the key-command name (cs-token):

```

259 \protected\def\new@keycommand#1{\@testopt{\@newkeycommand#1}{}}

```

`\@newkeycommand` Reads the first optional parameter (keys or number of mandatory args):

```

260 \protected\def\@newkeycommand#1[#2]{% #2 = key=values or N=mandatory args
261   \kcmd@plus \kcmd@unexpandchar@activate \fi% activates unexpand-char before reading definition
262   \ifstrnum{#2}%
263     {\@new@key@command#1[][\@testopt{\@newkeycommand#1[#2]}0]}% no kv, no optkey, number of args
264     {\@testopt{\@new@keycommand#1[#2]}0}% kv, check for optkey/nr of args

```

`\@new@keycommand` Reads the second optional parameter (opt key or number of mandatory args):

```

265 \protected\def\@new@keycommand#1[#2][#3]{%
266   \ifstrnum{#3}%
267     {\@new@key@command#1[#2][\@testopt{\@newkeycommand#1[#2]}0]}% no optkey
268     {\@testopt{\@new@keycommand#1[#2]}0}%

```

`\@new@key@command` Reads the definition of the command (`\kcmd@def` handles both cases of commands and environements).  
The so called "unexpand-char shortcut" has been activated before reading command definition:

```

269 \protected\long\def\@new@key@command#1[#2][#3][#4]#5{%
270   \kcmd@def#1\relax[\@testopt{\@newkeycommand#1[#2]}0]{\@testopt{\@newkeycommand#1[#2]}0}%

```

`\renew@keycommand`

```

271 \protected\def\renew@keycommand#1{\begingroup
272   \escapechar\m@ne\edef\@gtempa{\string#1}%
273   \expandafter\@ifundefined\@gtempa
274     {\endgroup\@latex@error{\noexpand#1undefined}\@ehc}
275   \endgroup
276   \let\@ifdefinable\@rc@ifdefinable
277   \new@keycommand#1}

```

`\provide@keycommand`

```

278 \protected\def\provide@keycommand#1{%
279   \escapechar\m@ne\edef\@gtempa{\string#1}%
280   \expandafter\@ifundefined\@gtempa
281     {\new@keycommand#1}
282     {\@testopt{\provide@key@command\kcmd@notdefinable}{}}
283 \protected\def\provide@key@command#1[#2]{\@tempwafalse\kcmd@def#1[#2]}

```

### 3.5 new key-environments

`\newkeyenvironment`

```

284 \protected\def\newkeyenvironment{\begingroup
285   \let\kcmd@gl\@empty\kcmd@modifiers\new@keyenvironment}
286 \protected\def\renewkeyenvironment{\begingroup
287   \let\kcmd@gl\@empty\kcmd@modifiers\renew@keyenvironment}

```

## `\new@keyenvironment`

```

288 \def\new@keyenvironment#1{\@testopt{\@newkeyenva{#1}}{}}
289 \def\@newkeyenva#1[#2]{%
290   \ifstrnum{#2}%
291     {\@newkeyenv{#1}{[] [] [{#2}]}}
292     {\@testopt{\@newkeyenvb{#1}[#{2}]}{}}}
293 \def\@newkeyenvb#1[#2][#3]{%
294   \ifstrnum{#3}%
295     {\@newkeyenv{#1}[#{2}] [] [{#3}]}
296     {\@testopt{\@newkeyenvc{#1}[#{2}][#{3}]}{0}}
297 \def\@newkeyenvc#1#2[#3]{\@newkeyenv{#1}{#2}[#{3}]}
298 \long\def\@newkeyenv#1#2{%
299   \kcmd@plus \kcmd@unexpandchar@activate \fi
300   \kcmd@keyenvir@def{#1}{#2}%
301 }
302 \long\def\kcmd@keyenvir@def#1#2#3#4{%
303   \cslet{end#1}\protected
304   \expandafter\kcmd@def\csname #1\expandafter\endcsname\csname end#1\endcsname#2{#3}{#4}%
305 }

```

## `\renew@keyenvironment`

```

306 \def\renew@keyenvironment#1{%
307   \@ifundefined{#1}%
308     {\@latex@error{Environment #1 undefined}\@ehc
309     }\relax
310   \cslet{#1}\relax
311   \new@keyenvironment{#1}}

```

## 3.6 Tests on keys

`\ifcommandkey {<key-name>}{<true>}{<false>}` expands `<true>` only if the value of the key is not blank:

```

312 \newcommand*\ifcommandkey[1]{\csname @\expandafter\expandafter\expandafter
313   \ettl@nbk\commandkey{#1}//{first}{second}//%
314   oftwo\endcsname}

```

`\showcommandkeys` are helper macros essentially for debugging purpose...

```

315 \newcommand\showcommandkeys[1]{\let\do\showcommandkey\docsvlist{#1}}
316 \newcommand\showcommandkey[1]{key \string"#1\string" = %
317   \expandnext\expandnext\expandnext\detokenize{\commandkey{#1}}\par}
318 \end{package}

```

## 4 Examples

```

319 <*example>
320 \ProvidesFile{keycommand-example}
321 \documentclass[a4paper]{article}
322 \usepackage[T1]{fontenc}
323 \usepackage[latin1]{inputenc}
324 \usepackage[american]{babel}
325 \usepackage{keycommand, framed, fancyvrb}
326 %
327 \makeatletter
328 \parindent\z@
329 \newkeycommand\Rule[raise=.4ex,width=1em,thick=.4pt][1]{%

```

```

330 \rule[\commandkey{raise}]{\commandkey{width}}{\commandkey{thick}}%
331 #1%
332 \rule[\commandkey{raise}]{\commandkey{width}}{\commandkey{thick}}%
333
334 \newkeycommand\charleads[sep=1][2]{%
335 \ifhmode\else\leavevmode\fi\setbox\@tempboxa\hbox{#2}\@tempdima=1.584\wd\@tempboxa%
336 \cleaders\hb@xt@\commandkey{sep}\@tempdima{\hss\box\@tempboxa\hss}#1%
337 \setbox\@tempboxa\box\voidb@x}
338 \newcommand\charfill[1][\charleads[{\#1}]{\hfill\kern\z@}]{}
339 \newcommand\charfil[1][\charleads[{\#1}]{\hfil\kern\z@}]{}
340 %
341 \newkeyenvironment{dblruled}[first=.4pt,second=.4pt,sep=1pt,left=\z@]{%
342 \def\FrameCommand{%
343 \vrule\@width\commandkey{first}%
344 \hskip\commandkey{sep}%
345 \vrule\@width\commandkey{second}%
346 \hspace{\commandkey{left}}}%
347 \parindent\z@
348 \MakeFramed {\advance\hsize-\width \FrameRestore}}
349 {\endMakeFramed}
350 %
351 \makeatother
352 \begin{document}
353 \title{This is {\tt keycommand-example.tex}}
354 \author{Florent Chervet}
355 \date{July 22, 2009}
356
357 \maketitle
358
359 {\Large Please refer to {\tt keycommand-example.tex} for definitions.}
360
361 \section{Example of a keycommand : \texttt{\string\Rule}}
362
363 \begin{tabular*}{\textwidth}{rl}
364 \verb+\Rule[width=2em]{hello}+:&\Rule[width=2em]{hello}\cr
365 \verb+\Rule[thick=1pt,width=2em]{hello}+:&\Rule[thick=1pt,width=2em]{hello}\cr
366 \verb+\Rule{hello}+:&\Rule{hello}\cr
367 \verb+\Rule[thick=1pt,raise=1ex]{hello}+:&\Rule[thick=1pt,raise=1ex]{hello}
368 \end{tabular*}
369
370 \section{Example of a keycommand : \texttt{\string\charfill}}
371
372 \begin{tabular*}{\textwidth}{rp{.4\textwidth}}
373 \verb+\charfill{\$star$}+:&\charfill{\$star$}\cr
374 \verb+\charfill[sep=2]{\$star$}+:&\charfill[sep=2]{\$star$} \\\
375 \verb+\charfill[sep=.7]{\textasteriskcentered}+:&\charfill[sep=.7]{\textasteriskcentered}
376 \end{tabular*}
377
378
379 \section{Example of a keyenvironment : \texttt{dblruled}}
380
381 \verb+\begin{dblruled}+\par
382 \verb+ test for dblrled key-environment\par+\par
383 \verb+ test for dblrled key-environment\par+\par
384 \verb+ test for dblrled key-environment+\par
385 \verb+\end{dblruled}+
386
387 \begin{dblruled}
388 test for dblrled key-environment\par
389 test for dblrled key-environment\par
390 test for dblrled key-environment
391 \end{dblruled}

```

```

392
393
394 \verb+\begin{dblruled}[first=4pt,sep=2pt,second=.6pt,left=.2em]+\par
395 \verb+   test for dblruled key-environment\par+\par
396 \verb+   test for dblruled key-environment\par+\par
397 \verb+   test for dblruled key-environment+\par
398 \verb+\end{dblruled}+
399
400 \begin{dblruled}[first=4pt,sep=2pt,second=.6pt,left=.2em]
401 test for dblruled key-environment\par
402 test for dblruled key-environment\par
403 test for dblruled key-environment
404 \end{dblruled}
405
406 \end{document}
407 \end{example}

```

## 5 History

### [2010/04/15 v3.1]

- Correction of bug in the normalization process.  
Correction of a bug in `\ifcommandkey` (undesirable space...)

### [2010/03/28 v3.0]

- Complete redesign of the implementation.  
keycommand is now based on some macros of etoolbox.
- Adding the + prefix and the ability to capture keys that where not defined.

### [2009/07/22 v1.0]

- First version.

## 6 References

- [1] Hendri Adriaens: *The xkeyval package*; 2008/08/13 v2.6a; [CTAN:macros/latex/contrib/xkeyval.dtx](#)
- [2] Heiko Oberdiek: *The kvsetkeys package*; 2007/09/29 v1.3; [CTAN:macros/latex/contrib/oberdiek/kvsetkeys.dtx](#).
- [3] David Carlisle: *The keyval package*; 1999/03/16 v1.13; [CTAN:macros/latex/required/graphics/keyval.dtx](#).

## 7 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	
<code>\@ehc</code> . . . . .	274, 308
<code>\@ifdefinable</code> . . . . .	145, 276
<code>\@ifundefined</code> . . . . .	273, 280, 307
<code>\@new@key@command</code> . . . . .	263, 267, 268, <u>269</u>
<code>\@new@keycommand</code> . . . . .	264, <u>265</u>
<code>\@newkeycommand</code> . . . . .	259, <u>260</u>
<code>\@newkeyenv</code> . . . . .	291, 295, 297, 298
<code>\@newkeyenva</code> . . . . .	288, 289
<code>\@newkeyenvb</code> . . . . .	292, 293



<code>\@newkeyenv</code> . . . . .	296, 297	<code>\kcmd@defcommand</code> . . . . .	145, 146, 148, 178
<code>\@nil</code> . . . . .	103, 109, 111, 183, 184	<code>\kcmd@define@boolkey</code> . . . . .	84, 114, 116, 122
<code>\@rc@ifdefinable</code> . . . . .	276	<code>\kcmd@define@choicekey</code> . . . . .	87, 90, 93, 96, 105, 112
<code>\@swaparg</code> . . . . .	38	<code>\kcmd@definekey</code> . . . . .	49, 151
<code>\~</code> . . . . .	188	<code>\kcmd@error@handler</code> . . . . .	111, 121, 124
<b>A</b>		<code>\kcmd@fam</code> . . . . .	147, 149, 150, 151, 152, 153, 155, 156, 158, 159, 162, 165, 173, 174, 176, 177
<code>\active</code> . . . . .	188, 189, 249	<code>\kcmd@firstchoiceof</code> . . . . .	103, 109, 111
<code>\afterassignment</code> . . . . .	200, 212, 222	<code>\kcmd@gbl</code> . . . . .	149, 151, 158, 161, 165, 168, 171, 235, 237, 239, 285, 287
<code>\AtEndOfPackage</code> . . . . .	29	<code>\kcmd@get@keyname@bool</code> . . . . .	63, 83, 84
<b>B</b>		<code>\kcmd@get@keyname@choice</code> . . . . .	71, 95, 96
<code>\body</code> . . . . .	217, 219	<code>\kcmd@get@keyname@choicest</code> . . . . .	69, 92, 93
<b>C</b>		<code>\kcmd@get@keyname@enum</code> . . . . .	67, 89, 90
<code>\catcode</code> . . . . .	19, 21, 188, 189, 249	<code>\kcmd@get@keyname@enumst</code> . . . . .	65, 86, 87
<code>\charfil</code> . . . . .	339	<code>\kcmd@ifplus</code> . . . . .	243, 246
<code>\charfill</code> . . . . .	338, 370, 373, 374, 375	<code>\kcmd@ifstar</code> . . . . .	242, 245
<code>\charleads</code> . . . . .	334, 338, 339	<code>\kcmd@keyenvir@def</code> . . . . .	300, 302
<code>\cleaders</code> . . . . .	336	<code>\kcmd@keyfam</code> . . . . .	30, 147
<code>\commandkey</code> . . . . .	147, 156, 172, 313, 317, 330, 332, 336, 343, 344, 345, 346	<code>\kcmd@long</code> . . . . .	158, 161, 165, 168, 242
<code>\csdef</code> . . . . .	134, 152, 153, 158, 165, 187	<code>\kcmd@modifiers</code> . . . . .	235, 237, 239, 240, 285, 287
<code>\csedef</code> . . . . .	40, 80, 83, 86, 89, 92, 95, 110, 120, 228	<code>\kcmd@mount@unexpandchar</code> . . . . .	150, 185
<code>\cslet</code> . . . . .	139, 192, 303, 310	<code>\kcmd@nargs</code> . . . . .	211, 218, 226
<code>\csletcs</code> . . . . .	99, 100	<code>\kcmd@normalize@braces</code> . . . . .	37, 46
<code>\csundef</code> . . . . .	141, 142	<code>\kcmd@normalize@setkeys</code> . . . . .	31, 181
<code>\csvloop</code> . . . . .	138, 174, 232	<code>\kcmd@notdefinable</code> . . . . .	282
<b>D</b>		<code>\kcmd@otherkeys</code> . . . . .	176, 183
<code>\default</code> . . . . .	115, 119, 121	<code>\kcmd@pkg@name</code> . . . . .	6, 10, 42, 129, 252
<code>\define@choicekey</code> . . . . .	106, 117	<code>\kcmd@plus</code> . . . . .	156, 157, 173, 243, 248, 261, 299
<code>\define@cmdkey</code> . . . . .	81	<code>\kcmd@resetdefault</code> . . . . .	174, 182
<code>\detokenize</code> . . . . .	30, 48, 317	<code>\kcmd@setkeys</code> . . . . .	176, 180
<code>\docsvlist</code> . . . . .	315	<code>\kcmd@undefinekey</code> . . . . .	138, 140
<b>E</b>		<code>\kcmd@undefinekeys</code> . . . . .	135, 149
<code>\etl@aftergroup@def</code> . . . . .	191	<code>\kcmd@unexpandchar</code> . . . . .	150, 244, 247
<code>\etl@nbk</code> . . . . .	39, 41, 62, 64, 66, 68, 70, 115, 183, 186, 245, 246, 248, 251, 313	<code>\kcmd@unexpandchar@activate</code> . . . . .	249, 250, 261, 299
<code>\expandnext</code> . . . . .	138, 317	<code>\kcmd@XofNINE</code> . . . . .	226, 232, 233
<code>\expandonce</code> . . . . .	40, 47, 87, 90, 119, 121, 150, 221	<code>\kcmd@XofNine</code> . . . . .	227, 230, 233
<b>F</b>		<code>\kcmd@Xsetkeys</code> . . . . .	8, 38
<code>\futuredef</code> . . . . .	241	<code>\kcmd@yarg@body</code> . . . . .	217, 221
<b>I</b>		<code>\kcmd@yarg@edef</code> . . . . .	212, 219
<code>\ifcase</code> . . . . .	79	<code>\kcmd@yargd@f</code> . . . . .	200, 202
<code>\ifcommandkey</code> . . . . .	4, 312	<code>\kcmd@yargdef</code> . . . . .	165, 168, 194
<code>\ifcsundef</code> . . . . .	136	<code>\kcmd@yarged@f</code> . . . . .	220, 224
<code>\iffalse</code> . . . . .	246	<code>\kcmd@yargedef</code> . . . . .	158, 161, 204
<code>\ifstrnum</code> . . . . .	262, 266, 290, 294	<code>\kcmd@yargedef@next</code> . . . . .	221, 222
<code>\iftrue</code> . . . . .	246	<code>\kv@list</code> . . . . .	36, 37, 38, 47
<b>K</b>		<code>\kv@normalize</code> . . . . .	36
<code>\kcmd@@therkeys</code> . . . . .	183, 184	<code>\kv@parse</code> . . . . .	151
<code>\kcmd@AtEnd</code> . . . . .	15, 17, 18, 29	<code>\kv@parse@normalized</code> . . . . .	37
<code>\kcmd@check@typeofkey</code> . . . . .	49, 79	<b>L</b>	
<code>\kcmd@check@typeofkey@bool</code> . . . . .	56, 62	<code>\Large</code> . . . . .	359
<code>\kcmd@check@typeofkey@choice</code> . . . . .	60, 70	<code>\lccode</code> . . . . .	188
<code>\kcmd@check@typeofkey@choicest</code> . . . . .	59, 68	<code>\letcs</code> . . . . .	211
<code>\kcmd@check@typeofkey@enum</code> . . . . .	58, 66	<code>\lowercase</code> . . . . .	189
<code>\kcmd@check@typeofkey@enumst</code> . . . . .	57, 64	<b>N</b>	
<code>\kcmd@def</code> . . . . .	144, 270, 283, 304	<code>\new@keycommand</code> . . . . .	235, 259, 277, 281
<code>\kcmd@defcommand</code> . . . . .	145, 146, 148, 178	<code>\new@keyenvironment</code> . . . . .	285, 288, 311
<code>\kcmd@define@boolkey</code> . . . . .	84, 114, 116, 122	<code>\newkeycommand</code> . . . . .	2, 157, 164, 234, 254, 329, 334
<code>\kcmd@define@choicekey</code> . . . . .	87, 90, 93, 96, 105, 112	<code>\newkeyenvironment</code> . . . . .	4, 284, 341
<code>\kcmd@definekey</code> . . . . .	49, 151	<code>\newrobustcmd</code> . . . . .	31
<code>\kcmd@error@handler</code> . . . . .	111, 121, 124		
<code>\kcmd@fam</code> . . . . .	147, 149, 150, 151, 152, 153, 155, 156, 158, 159, 162, 165, 173, 174, 176, 177		
<code>\kcmd@firstchoiceof</code> . . . . .	103, 109, 111		
<code>\kcmd@gbl</code> . . . . .	149, 151, 158, 161, 165, 168, 171, 235, 237, 239, 285, 287		
<code>\kcmd@get@keyname@bool</code> . . . . .	63, 83, 84		
<code>\kcmd@get@keyname@choice</code> . . . . .	71, 95, 96		
<code>\kcmd@get@keyname@choicest</code> . . . . .	69, 92, 93		
<code>\kcmd@get@keyname@enum</code> . . . . .	67, 89, 90		
<code>\kcmd@get@keyname@enumst</code> . . . . .	65, 86, 87		
<code>\kcmd@ifplus</code> . . . . .	243, 246		
<code>\kcmd@ifstar</code> . . . . .	242, 245		
<code>\kcmd@keyenvir@def</code> . . . . .	300, 302		
<code>\kcmd@keyfam</code> . . . . .	30, 147		
<code>\kcmd@long</code> . . . . .	158, 161, 165, 168, 242		
<code>\kcmd@modifiers</code> . . . . .	235, 237, 239, 240, 285, 287		
<code>\kcmd@mount@unexpandchar</code> . . . . .	150, 185		
<code>\kcmd@nargs</code> . . . . .	211, 218, 226		
<code>\kcmd@normalize@braces</code> . . . . .	37, 46		
<code>\kcmd@normalize@setkeys</code> . . . . .	31, 181		
<code>\kcmd@notdefinable</code> . . . . .	282		
<code>\kcmd@otherkeys</code> . . . . .	176, 183		
<code>\kcmd@pkg@name</code> . . . . .	6, 10, 42, 129, 252		
<code>\kcmd@plus</code> . . . . .	156, 157, 173, 243, 248, 261, 299		
<code>\kcmd@resetdefault</code> . . . . .	174, 182		
<code>\kcmd@setkeys</code> . . . . .	176, 180		
<code>\kcmd@undefinekey</code> . . . . .	138, 140		
<code>\kcmd@undefinekeys</code> . . . . .	135, 149		
<code>\kcmd@unexpandchar</code> . . . . .	150, 244, 247		
<code>\kcmd@unexpandchar@activate</code> . . . . .	249, 250, 261, 299		
<code>\kcmd@XofNINE</code> . . . . .	226, 232, 233		
<code>\kcmd@XofNine</code> . . . . .	227, 230, 233		
<code>\kcmd@Xsetkeys</code> . . . . .	8, 38		
<code>\kcmd@yarg@body</code> . . . . .	217, 221		
<code>\kcmd@yarg@edef</code> . . . . .	212, 219		
<code>\kcmd@yargd@f</code> . . . . .	200, 202		
<code>\kcmd@yargdef</code> . . . . .	165, 168, 194		
<code>\kcmd@yarged@f</code> . . . . .	220, 224		
<code>\kcmd@yargedef</code> . . . . .	158, 161, 204		
<code>\kcmd@yargedef@next</code> . . . . .	221, 222		
<code>\kv@list</code> . . . . .	36, 37, 38, 47		
<code>\kv@normalize</code> . . . . .	36		
<code>\kv@parse</code> . . . . .	151		
<code>\kv@parse@normalized</code> . . . . .	37		



<code>\noexpandcs</code> . . . . .	156, 158, 165, 172, 174, 177, 182	<b>S</b>	
<code>\nr</code> . . . . .	93, 96, 107, 117, 120	<code>\setkeys</code> . . . . .	8
<code>\number</code> . . . . .	93, 96, 120, 159, 166	<code>\showcommandkey</code> . . . . .	315, 316
		<code>\showcommandkeys</code> . . . . .	<u>315</u>
<b>P</b>		<b>T</b>	
<code>\PackageError</code> . . . . .	10, 42, 129, 252	<code>\TMP@EnsureCode</code> . . . . .	16, 23, 24, 25, 26, 27, 28
<code>\protected</code> . . . . .	73, 124, 144, 146, 171, 180, 185, 187, 194, 204, 217, 234, 236, 238, 240, 259, 260, 265, 269, 271, 278, 283, 284, 286, 303	<b>U</b>	
<code>\provide@key@command</code> . . . . .	282, 283	<code>\undef</code> . . . . .	29, 233
<code>\provide@keycommand</code> . . . . .	239, <u>278</u>	<code>\unexpanded</code> . . . . .	48, 103, 108, 110, 151, 154, 190, 214, 217, 252, 253
<code>\providekeycommand</code> . . . . .	238	<code>\unless</code> . . . . .	160, 167
<b>R</b>		<b>V</b>	
<code>\renew@keycommand</code> . . . . .	237, <u>271</u>	<code>\val</code> . . . . .	87, 90, 107, 117, 130
<code>\renew@keyenvironment</code> . . . . .	287, <u>306</u>	<b>X</b>	
<code>\renewkeycommand</code> . . . . .	236	<code>\XKV@rm</code> . . . . .	40, 41, 43
<code>\renewkeyenvironment</code> . . . . .	286	<code>\XKV@setkeys</code> . . . . .	8
<code>\Rule</code> . . . . .	329, 361, 364, 365, 366, 367	<code>\XKV@testopta</code> . . . . .	8
<code>\rule</code> . . . . .	330, 332	<code>\XKV@testoptc</code> . . . . .	8