

The *free and open source*
keycommand* package
key-value interface for commands and environments in L^AT_EX.

<florent.chervet@free.fr>

2010/04/27 – version 3.1415

Abstract

keycommand provides an easy way to define commands or environments with optional keys.

`\newkeycommand` `\renewkeycommand` `\providekeycommand` and `\newkeyenvironment`,
`\renewkeyenvironment` are macros to define such commands and environments with keys.

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

Keys are defined with 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* (see 1.1).

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).

It works with an ε -T_EX distribution of L^AT_EX.

Contents

| | | |
|----------|--|-----------|
| 1 | User Interface | 2 |
| 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 |
| 1.6 | Example of a + key-environment | 4 |
| 2 | Messages and more | 4 |
| 2.1 | Invalid keys | 4 |
| 2.2 | Testing keys | 5 |
| 2.3 | xkeyval, keyval and kvsetkeys comparison | 6 |
| 3 | Implementation | 6 |
| 3.1 | Identification | 6 |
| 3.2 | Requirements | 6 |
| 3.3 | Defining (and undefining) command-keys | 7 |
| 3.4 | new key-commands | 12 |
| 3.5 | new key-environments | 14 |
| 3.6 | Tests on keys | 15 |
| 4 | Examples | 15 |
| 5 | History | 16 |
| | [2010/04/27 v3.1415] | 16 |
| | [2010/04/25 v3.141] | 17 |
| | [2010/04/18 v3.14] | 17 |
| | [2010/03/28 v3.0] | 17 |
| | [2009/07/22 v1.0] | 17 |
| 6 | References | 17 |
| 7 | Index | 17 |

* 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.

1 User Interface

1.1 General syntax

| | | | | | | |
|----------------------------------|------------------------------|------------------------------------|--|---------------------------------|-----------------------|---------------------------------------|
| $\backslash\text{newkeycommand}$ | $^*+[\text{short-unexpand}]$ | $\{\langle\text{command}\rangle\}$ | $[\langle\text{keys=defaults}\rangle]$ | $[\langle\text{OptKey}\rangle]$ | $[\langle n \rangle]$ | $\{\langle\text{definition}\rangle\}$ |
| | modifiers Optional | Required | Optional | | | Required |

$\backslash\text{newkeycommand}$ will define $\backslash\text{command}$ as a new key-command! well...

Use the $*$ form when you do not want it to be a $\backslash\text{long}$ macro (as for L^AT_EX- $\backslash\text{newcommand}$).

The $[\text{keys=defaults}]$ 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 | exemple | value of $\backslash\text{commandkey}$ |
|-----------|--|--|
| general | $\text{color}=\text{red}$ | $\backslash\text{commandkey}\{\text{color}\}$ is 'red' and may be anything (text, number, macro...) |
| boolean | $\text{bool bold}=\text{true}$ | $\backslash\text{commandkey}\{\text{bold}\}$ is: 0 (for false) 1 (for true) |
| enumerate | $\text{enum position}=\{\text{left,centered,right}\}$ | $\backslash\text{commandkey}\{\text{position}\}$ is: 'left' by default and can be 'centered' or 'right' |
| | $\text{enum}^*\text{ position}=\{\text{left,centered,right}\}$ | This is the same, except match is case insensitive |
| choice | $\text{choice position}=\{\text{left,centered,right}\}$ | $\backslash\text{commandkey}\{\text{position}\}$ is: 0 (for left the default value), 1 (for centered) 2 (for right) |
| | $\text{choice}^*\text{ position}=\{\text{left,centered,right}\}$ | This is the same, except match is case insensitive |

The OptKey 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 $\langle n \rangle$ (0 if omitted).

The $+$ form expands the $\backslash\text{commandkey}$ before executing the key-command itself, as explain in section 1.3.

1.2 First example :

```
 $\backslash\text{newkeycommand}\backslash\text{textrule}[\text{raise}=.4\text{ex},\text{width}=3\text{em},\text{thick}=.4\text{pt}][1]\{\%
\text{rule}[\backslash\text{commandkey}\{\text{raise}\}]\{\backslash\text{commandkey}\{\text{width}\}\}\{\backslash\text{commandkey}\{\text{thick}\}\}
\#1
\text{rule}[\backslash\text{commandkey}\{\text{raise}\}]\{\backslash\text{commandkey}\{\text{width}\}\}\{\backslash\text{commandkey}\{\text{thick}\}\}\}$ 
```

defines the keys width , thick and raise with their default values (if not specified): 3em , $.4\text{pt}$ and $.4\text{ex}$. Now $\backslash\text{textrule}$ can be used as follow:

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

| | | | | | | | |
|--------------------------------------|-----------------------------|---------------------------------------|--|---------------------------------|-----------------------|--|----------|
| $\backslash\text{newkeyenvironment}$ | $*+[\text{short-unexpand}]$ | $\{\langle\text{envir name}\rangle\}$ | $[\langle\text{keys=defaults}\rangle]$ | $[\langle\text{OptKey}\rangle]$ | $[\langle n \rangle]$ | $\{\langle\text{begin}\rangle\}\{\langle\text{end}\rangle\}$ | |
| | modifiers Optional | Required | Optional | | | Required | Required |

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.

Key-environments may be defined with the **+** form in the same way as `\newkeycommand` is used. Be aware that each part of the environment: $\langle\text{begin}\rangle$ and $\langle\text{end}\rangle$ are expanded at run time then, and the optional `[\]` argument protects from expansion in each of those parts.

1.6 Example of a + key-environment

```
\newkeyenvironment+[\ ]{ myfigure}[
  caption,
  enum placement={H,h,b,t,p},
  width=.5\linewidth,
  label
][ OtherKeys][1]%
{% begin part
  |\begin{figure}|[\commandkey{placement}]
  |\includegraphics|[\commandkey{ OtherKeys},width=\commandkey{width}]{#1}%
}
{% end part
  \ifcommandkey{caption}{|\caption|{\commandkey{caption} image file = #1}}{}%
  \ifcommandkey{label}{|\label|{\commandkey{label}}}{}%
  |\end{figure}|%
}
```

As you can see, `\commandkey` and mandatory arguments (**#1** here) are available both in the $\langle\text{begin}\rangle$ and in the $\langle\text{end}\rangle$ parts of the key-environment.

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 is recoverable: the key is assigned its default value.

If you use a `\commandkey{<name>}` in a key-command where *<name>* is not defined as a key, you will get the T_EX 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^AT_EX 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   [2010/04/27 v3.1415 - key-value interface for commands and environments in LaTeX]
```

3.2 Requirements

The package is based on xkeyval. However, 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 ϵ -T_EX primitives in keycommand we also load the etex package in order to get an error message if ϵ -T_EX is not running.

The etoolbox package gives some facility to write keycommand.

From version 3.141 onwards, keycommand does not load etextools anymore.

```

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

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

```

8 \protected\def\kcmd@Xsetkeys{\XKV@sttrue\XKV@plfalse\XKV@testoptc\XKV@setkeys}% in case \setkeys
9 % was overwritten
```

Some `\catcode` assertions internally used by keycommand:

```

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

```

`\kcmd@ifstrdigit` This macro is used to test the optional arguments of `\newkeycommand`, in particular, one must know in an argument is a single digit (representing the number of mandatory arguments) or anything else (representing the key=value list or the “special” OptKey key:

```

25 \iffalse%\ifdefined\pdfmatch% use \pdfmatch if present
26   \long\def\kcmd@ifstrdigit#1{\csname @\ifnum\pdfmatch
27     {\detokenize{^[[:space:]]*[[[:digit:]]][[:space:]]*$}}{\detokenize{#1}}=1 %
28     first\else second\fi oftwo\endcsname}
29 \else% use filter, very efficient !
30 \def\kcmd@ifstrdigit#1{%
31   \kcmd@nbk#1//%
32   {\expandafter\expandafter\expandafter\kcmd@ifstrdigit@i
33     \expandafter\expandafter\expandafter{\detokenize\expandafter{\number\number0#1}}}%
34   {\@secondoftwo}}//%
35 }
36 \def\kcmd@ifstrdigit@i#1{%
37   \def\kcmd@ifstrdigit@ii##1#1##2##3\kcmd@ifstrdigit@ii{%
38     \csname @\ifx##20first\else second\fi oftwo\endcsname
39     }\kcmd@ifstrdigit@ii 00 01 02 03 04 05 06 07 08 09 0#1 \relax\kcmd@ifstrdigit@ii
40 }
41 \fi

```

3.3 Defining (and undefining) command-keys

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

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

`\kcmd@nbk` is the optimized `\ifnotblank` macro of `etoolbox` (with / having a catcode of 8):

```
43 \def\kcmd@nbk#1#2/#3#4#5//{#4}%
```

`\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:

```

44 \newrobustcmd\kcmd@normalize@setkeys[4]{%
45 % #1 = key-command,
46 % #2 = family,
47 % #3 = other-key,
48 % #4 = key-values pairs
49   \kv@normalize{#4}\toks@{}}%

```

```

50 \expandafter\kv@parse@normalized\expandafter{\kv@list}{\kcmd@normalize@braces{#2}}%
51 \edef\kv@list{\kcmd@Xsetkeys{\unexpanded{#2}}{\the\toks@}}\kv@list
52 \kcmd@nbk#3//% undeclared keys are assigned to "OtherKeys"
53   {\cslet{#2@#3}\XKV@rm}% (if specified, ie not empty)
54   {\expandafter\kcmd@nbk\XKV@rm//% (otherwise a recoverable error is thown)
55     {\PackageError\kcmd@pkg@name{The key-value pairs :\MessageBreak
56       \XKV@rm\MessageBreak
57       cannot be processed for key-command \string#1\MessageBreak
58       See the definition of the key-command!}{}}{}}//}%
59 }
60 \long\def\kcmd@normalize@braces#1#2#3% This is kvsetkeys processor for normalizing braces
61   \toks@\expandafter{\the\toks@,#2}%
62   \ifx @\detokenize{#3}@ \else \toks@\expandafter{\the\toks@={{#3}}}\fi
63 }

```

\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*) :

```

64 \def\kcmd@check@typeofkey#1% expands to
65% 0 if key has no type,
66% 1 if boolean,
67% 2 if enum*,
68% 3 if enum,
69% 4 if choice*,
70% 5 if choice
71   \kcmd@check@typeofkey@bool#1bool //%
72   {\kcmd@check@typeofkey@enumst#1enum* //%
73     {\kcmd@check@typeofkey@enum#1enum //%
74       {\kcmd@check@typeofkey@choicest#1choice* //%
75         {\kcmd@check@typeofkey@choice#1choice //%
76           05//}4//}3//}2//}1//}
77 \def\kcmd@check@typeofkey@bool #1bool #2//{\kcmd@nbk#1//}
78 \def\kcmd@get@keyname@bool #1bool #2//{#2}
79 \def\kcmd@check@typeofkey@enumst #1enum* #2//{\kcmd@nbk#1//}
80 \def\kcmd@get@keyname@enumst #1enum* #2//{#2}
81 \def\kcmd@check@typeofkey@enum #1enum #2//{\kcmd@nbk#1//}
82 \def\kcmd@get@keyname@enum #1enum #2//{#2}
83 \def\kcmd@check@typeofkey@choicest #1choice* #2//{\kcmd@nbk#1//}
84 \def\kcmd@get@keyname@choicest #1choice* #2//{#2}
85 \def\kcmd@check@typeofkey@choice #1choice #2//{\kcmd@nbk#1//}
86 \def\kcmd@get@keyname@choice #1choice #2//{#2}
87%
88 \protected\long\def\kcmd@definekey#1#2#3#4#5% define the keys using xkeyval macros
89% #1 = keycommand,
90% #2 = \global,
91% #3 = family,
92% #4 = key (before = sign),
93% #5 = default (after = sign)
94   \ifcase\kcmd@check@typeofkey{#4}\relax% standard
95     #2\csedef{#3.keylist}{\csname#3.keylist\endcsname,#4}%
96     \define@cmdkey{#3}[{#3@}]{#4}[{#5}]{}%
97   \or% bool
98     #2\csedef{#3.keylist}{\csname#3.keylist\endcsname,\kcmd@get@keyname@bool#4//}%
99     \kcmd@define@boolkey#1{#3}{\kcmd@get@keyname@bool#4//}{#5}%
100  \or% enum*
101     #2\csedef{#3.keylist}{\csname#3.keylist\endcsname,\kcmd@get@keyname@enumst#4//}%
102     \kcmd@define@choicekey#1*{#3}{\kcmd@get@keyname@enumst#4//}{#5}{\expandonce\val}%
103  \or% enum
104     #2\csedef{#3.keylist}{\csname#3.keylist\endcsname,\kcmd@get@keyname@enum#4//}%

```



```

105     \kcmd@define@choicekey#1*{#3}{\kcmd@get@keyname@enum#4//}{#5}{\expandonce\val}%
106     \or% choice*
107     #2\csedef{#3.keylist}{\csname#3.keylist\endcsname,\kcmd@get@keyname@choicest#4//}%
108     \kcmd@define@choicekey#1*{#3}{\kcmd@get@keyname@choicest#4//}{#5}{\number\nr}%
109     \or% choice
110     #2\csedef{#3.keylist}{\csname#3.keylist\endcsname,\kcmd@get@keyname@choice#4//}%
111     \kcmd@define@choicekey#1*{#3}{\kcmd@get@keyname@choice#4//}{#5}{\number\nr}%
112     \fi
113     \ifx#2\global\relax
114         #2\csletcs{KV@#3@#4}{KV@#3@#4}% globalize
115         #2\csletcs{KV@#3@#4@default}{KV@#3@#4@default}% globalize default value
116     \fi
117 }
118 %
119 \long\def\kcmd@firstchoiceof#1,#2\kcmd@nil{\unexpanded{#1}}
120 %
121 \long\def\kcmd@define@choicekey#1#2#3#4#5#6{%
122     \begingroup\edef\kcmd@define@choicekey{\endgroup
123         \noexpand\define@choicekey#2+{#3}{#4}
124             [\noexpand\val\noexpand\nr]%
125             {\unexpanded{#5}}% list of allowed values
126             [{\kcmd@firstchoiceof#5,\kcmd@nil}]% default value
127             {\csedef{#3@#4}{\unexpanded{#6}}}% define key value if in the allowed list
128             {\kcmd@error@handler\noexpand#1{#3}{#4}{\kcmd@firstchoiceof#5,\kcmd@nil}}% error handl
129     }\kcmd@define@choicekey
130 }
131 \def\kcmd@define@boolkey#1#2#3#4{\begingroup
132     \kcmd@nbk#4//{\def\default{#4}}{\def\default{true}}//%
133     \edef\kcmd@define@boolkey{\endgroup
134         \noexpand\define@choicekey*+{#2}{#3}[\noexpand\val\noexpand\nr]%
135         {false,true}
136         [{\unexpanded\expandafter{\default}}]%
137         {\csedef{#2@#3}{\noexpand\number\noexpand\nr}}%
138         {\kcmd@error@handler\noexpand#1{#2}{#3}{\unexpanded\expandafter{\default}}}%
139     }\kcmd@define@boolkey
140 }
141 %
142 \protected\long\def\kcmd@error@handler#1#2#3#4{%
143 % #1 = key-command,
144 % #2 = family,
145 % #3 = key,
146 % #4 = default
147     \PackageError\kcmd@pkg@name{%
148         Value ‘\val\space’ is not allowed in key #3\MessageBreak
149         for key-command \string#1.\MessageBreak
150         I’ll use the default value ‘#4’ for this key.\MessageBreak
151         See the definition of the key-command!}%
152     \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`.

```

153 \protected\def\kcmd@undefinekeys#1#2{% #1 = global, #2 = family
154     \ifcsundef{#2.keylist}
155         {\cslet{#2.keylist}\@gobble}
156         {\expandafter\expandafter\expandafter\docsvlist
157             \expandafter\expandafter\expandafter{%
158                 \csname #2.keylist\endcsname}%
159         \cslet{#2.keylist}\@gobble}%
160 }

```

```

161 \def\kcmd@undefinekey#1#2#3{% #1 = global, #2 = family, #3 = key
162   #1\csundef{KV@#2@#3}%
163   #1\csundef{KV@#2@#3@default}%
164 }

```

`\kcmd@setdefaults` sets the defaults values for the keys at the very beginning of the keycommand:

```

165 \def\kcmd@setdefaults#1{%
166   \ifcsundef{#1.keylist}{}
167   {\expandafter\expandafter\expandafter\docsvlist
168     \expandafter\expandafter\expandafter{%
169       \csname#1.keylist\endcsname}}%
170 }

```

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

```

171 \protected\long\def\kcmd@def#1#2[#3][#4][#5]#6#7{%
172   \ifx#1\kcmd@donot@provide \endgroup
173   \else
174     \@tempswafalse\@ifdefinable#1{\@tempswattrue}%
175     \if@tempswa
176       \edef\kcmd@fam{\kcmd@keyfam{#1}}%
177       \expandafter\kcmd@defcommand\expandafter{\kcmd@fam}#1[#{3}][#{4}][#{5}]{#6}{#2}{#7}%
178     \else\endgroup
179     \fi
180   \fi
181 }

```

`\kcmd@defcommand` prepares (expands) 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))

```

182 \protected\long\def\kcmd@defcommand#1#2[#3][#4][#5]#6#7#8{%
183   \let\commandkey\relax \let\getcommandkey\relax \let#2\relax
184   \cslet{#1}\relax \cslet{#1.commankey}\relax \cslet{#1.getcommandkey}\relax
185   \def\do{\kcmd@undefinekey{\kcmd@gbl}{#1}}%
186   \edef\kcmd@defcommand{\endgroup
187     \kcmd@undefinekeys{\kcmd@gbl}{#1}% undefines all keys for this keycommand family
188     \ifx\kcmd@unexpandchar\empty\else
189       \kcmd@mount@unexpandchar{#1}{\unexpanded\expandafter{\kcmd@unexpandchar}}%
190     \fi
191     \unexpanded{\kv@parse{#3,#4}}{\kcmd@definekey\noexpand#2{\kcmd@gbl}{#1}}% defines keys
192     \csdef{#1.commankey}####1{\noexpand\csname#1@###1\endcsname}%
193     \csdef{#1.getcommandkey}####1{%
194       \unexpanded{\unexpanded\expandafter\expandafter\expandafter}{%
195         \noexpand\csname#1@###1\endcsname}}%
196     \kcmd@ifplus% \newkeycommand+ / \newkeyenvironment+
197     \protected\csdef{#1}{%
198       \kcmd@yargedef{\kcmd@gbl}{\kcmd@long}\csname#1\endcsname
199       {\number#5}{\noexpand#7}{\csname#1.unexpandchar\endcsname}}%
200     \ifx#7@gobble\else
201       \protected\def#7{\kcmd@yargedef#7}%
202     \fi
203     \else% \newkeycommand / \newkeyenvironment
204     \csdef{#1}{%
205       \kcmd@yargdef{\kcmd@gbl}{\kcmd@long}\csname#1\endcsname
206       {\number#5}{\noexpand#7}}%
207     \ifx#7@gobble\else \def#7####1{% that means we have to define a key-environment
208       \def#7{%
209         \let\getcommandkey\csname#1.getcommandkey\endcsname
210         \let\commandkey\csname#1.commandkey\endcsname
211         ####1}%
212       }%

```

```

213     \fi
214     \fi
215     \def\noexpand\do####1{\unexpanded{\expandafter\noexpand\csname}KV@#1@####1@default%
216                                         \endcsname}%
217     \let\commandkey\relax \let\getcommandkey\relax \let#2\relax
218     \kcmd@gb1\protected\edef#2{% entry point
219         \let\getcommandkey\noexpand\noexpand\csname#1.getcommandkey\endcsname
220         \kcmd@ifplus \let\commandkey\getcommandkey
221         \else \let\commandkey\noexpand\noexpand\csname#1.commandkey\endcsname
222         \fi
223         \noexpand\kcmd@setdefaults{#1}%
224         \ifx#7\@gobble \noexpand\noexpand\noexpand\@testopt
225             {\kcmd@setkeys#2{#1}{\kcmd@otherkey{#4}}}%
226         \else \noexpand\noexpand\noexpand\@testopt
227             {\kcmd@setkeys#2{#1}{\kcmd@otherkey{#4}}}%
228         \fi
229     }%
230     \csname#1\endcsname% expand \kcmd@yargedef or \kcmd@yargdef
231 } \kcmd@defcommand{#6}{#8}% #6 = definition, #8 = definition end-envir
232 }
233 \protected\long\def\kcmd@setkeys#1#2#3[#4]{% #1=key-command, #2=family, #3=otherkey, #4=key=value
234     \kcmd@normalize@setkeys{#1}{#2}{#3}{#4}\csname#2\endcsname
235 }
236 \long\def\kcmd@otherkey#1{\kcmd@nbk#1//{\kcmd@otherkey@name#1=\kcmd@nil}}//}
237 \long\def\kcmd@otherkey@name#1=#2\kcmd@nil{#1}

```

`\kcmd@mount@unexpandchar`

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

```

238 \protected \def \kcmd@mount@unexpandchar#1#2{%
239     \protected\csdef{#1.unexpandchar}{\begingroup
240         \catcode'\~\active \lccode'\~#2 \lccode'#2 0\relax
241         \lowercase{%
242             \expandafter\endgroup\expandafter\def\expandafter~{%
243                 \catcode'#2\active
244                 \long\def~#####1~{\unexpanded{#####1}}}%
245             ~}%
246     }%
247 }

```

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

```

248 \protected \def \kcmd@yargdef #1#2#3#4#5{\begingroup
249 % #1 = global or {}
250 % #2 = long or {}
251 % #3 = Command
252 % #4 = nr of args
253 % #5 = endenvir (or \@gobble if not an environment, or \relax if #3 is endenvir)
254 \def \kcmd@yargd@f ##1#4##2##{\afterassignment#5\endgroup
255     #1#2\expandafter\def\expandafter#3\@gobble ##1#4%
256 } \kcmd@yargd@f 0##1##2##3##4##5##6##7##8##9##4%
257 }

```

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

```

258 \protected \def \kcmd@yargedef#1#2#3#4#5#6{\begingroup
259 % #1 = global or {}
260 % #2 = long or {}

```

```

261 % #3 = Command
262 % #4 = nr of args
263 % #5 = endenvir (or \@gobble if not an environment, or \relax if #3 is endenvir)
264 % #6 = unexpandchar mounting macro
265 \kcmd@nargs{#4}%
266 \protected\long\def\kcmd@yarg@edef##1##2{\endgroup
267     #1\edef#3{\begingroup #6%
268         #2\edef#3\unexpanded{##2}{\endgroup\unexpanded{##1}%
269         }#3}%
270 }%
271 \protected\def\kcmd@envir##1{%
272     \edef\next{\kcmd@yarg@edef{\def\noexpand#5{\expandonce{#5##1}}\expandonce{#3##1}}\next
273     }%
274 \protected\def\kcmd@command##1{%
275     \edef\next{\kcmd@yarg@edef{\expandonce{#3##1}}\next
276     }%
277 \protected\def\kcmd@yargedef##1{%
278     \kcmd@yargedef@##1 0###1#####2###3###4###5###6###7###8###9#####4%
279     }%
280 \ifx#5\@gobble % keycommand
281     \def\next{\kcmd@command}%
282 \else % key-environment
283     \def\next{\kcmd@envir}%
284 \fi
285 \let\@next\relax
286 \def\kcmd@yargedef@##1##2#4##3###{%
287     \ifx\@next\relax
288         \edef\@next{\next{\expandonce{\kcmd@nargs}}{\expandonce{\@gobble##2#4}}}%
289         \ifx#5\@gobble \edef\@next{\expandonce\@next\noexpand#5}%
290         \else \edef\@next{\edef\noexpand\@next{\noexpand\unexpanded{\expandonce\@next}}#5}%
291         \fi
292     \fi
293     \afterassignment\@next
294     \expandafter\def\expandafter##1\@gobble##2#4%
295     }%
296 \kcmd@yargedef#3%
297 }

```

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

```

298 \def\kcmd@nargs#1{\edef\kcmd@nargs%##1##2##3##4##5##6##7##8##9%
299     {\ifnum#1>0{###1%
300         \ifnum#1>1}{###2%
301         \ifnum#1>2}{###3%
302         \ifnum#1>3}{###4%
303         \ifnum#1>4}{###5%
304         \ifnum#1>5}{###6%
305         \ifnum#1>6}{###7%
306         \ifnum#1>7}{###8%
307         \ifnum#1>8}{###9%
308         \fi\fi\fi\fi\fi\fi\fi\fi}\fi}%
309 }%

```

3.4 new key-commands

`\newkeycommand` Here are the entry points:

```

310 \newrobustcmd*\newkeycommand{\begingroup
311     \let\kcmd@gb1\@empty\kcmd@star@or@long\new@keycommand}
312 \newrobustcmd*\renewkeycommand{\begingroup
313     \let\kcmd@gb1\@empty\kcmd@star@or@long\renew@keycommand}

```

```
314 \newrobustcmd* \providekeycommand { \begingroup
315   \let \kcmd@gb1 \@empty \kcmd@star@or@long \provide@keycommand }
```

`\kcmd@star@or@long`

This is the adaptation of L^AT_EX's `\@star@or@long` macro:

```
316 \def \kcmd@star@or@long #1 { \ifstar
317   { \let \kcmd@long \@empty \kcmd@plus #1 }
318   { \def \kcmd@long { \long } \kcmd@plus #1 } }
319 \def \kcmd@@ifplus #1 { \ifnextchar + { \@firstoftwo { #1 } } } % same as LaTeX's \ifstar
320 \def \kcmd@plus #1 { \kcmd@ifplus
321   { \def \kcmd@ifplus { \iftrue } \kcmd@testopt #1 }
322   { \def \kcmd@ifplus { \iffalse } \kcmd@testopt #1 } }
323 \def \kcmd@testopt #1 { \@testopt { \kcmd@unexpandchar #1 } }
```

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

```
324 \def \kcmd@unexpandchar #1 [ #2 ] { %
325   \kcmd@ifplus
326     \kcmd@nbk #2 //
327     { \def \kcmd@unexpandchar { #2 } % only once inside group...
328       \def \kcmd@unexpandchar@activate { \catcode ' #2 \active } %
329     } { %
330       \let \kcmd@unexpandchar \@empty
331       \let \kcmd@unexpandchar@activate \relax
332     } // %
333   \else \let \kcmd@unexpandchar \@empty
334     \kcmd@nbk #2 // %
335     { \PackageError \kcmd@pkg@name { shortcut option for \string \unexpanded \MessageBreak
336       You can't use a shortcut option for \string \unexpanded \MessageBreak
337       without the \string+ form of \string \newkeycommand ! } %
338       { I will ignore this option and proceed. } %
339     } %
340     { } // %
341   \fi #1 }
```

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

```
342 \def \new@keycommand #1 { \@testopt { \newkeycommand #1 } 0 }
```

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

```
343 \long \def \@newkeycommand #1 [ #2 ] { % #2 = key=values or N=mandatory args
344   \kcmd@ifplus \kcmd@unexpandchar@activate \fi % activates unexpand-char before reading definition
345   \kcmd@ifstrdigit { #2 } %
346   { \@new@key@command #1 [ ] [ ] [ { #2 } ] } % no kv, no optkey, number of args
347   { \@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):

```
348 \long \def \@new@keycommand #1 [ #2 ] [ #3 ] { %
349   \kcmd@ifstrdigit { #3 } %
350   { \@new@key@command #1 [ { #2 } ] [ ] [ { #3 } ] } % no optkey
351   { \@testopt { \@new@key@command #1 [ { #2 } ] [ { #3 } ] 0 } }
```

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

```
352 \long \def \@new@key@command #1 [ #2 ] [ #3 ] [ #4 ] #5 { %
353   \kcmd@def #1 \@gobble [ { #2 } ] [ { #3 } ] [ { #4 } ] [ { #5 } ] }
```

`\renew@keycommand`

```

354 \def\renew@keycommand#1{\begingroup
355   \escapechar\m@ne\edef\@gtempa{\string#1}%
356   \expandafter\@ifundefined\@gtempa
357     {\endgroup\@latex@error{\noexpand#1undefined}\@ehc}
358   \endgroup
359   \let\@ifdefinable\@rc@ifdefinable
360   \new@keycommand#1%
361 }

```

`\provide@keycommand`

```

362 \def\provide@keycommand#1{\begingroup
363   \escapechar\m@ne\edef\@gtempa{\string#1}%
364   \expandafter\@ifundefined\@gtempa
365     {\endgroup\new@keycommand#1}
366     {\endgroup\def\kcmd@donot@provide{\renew@keycommand\kcmd@donot@provide
367       }\kcmd@donot@provide}%
368 }
369 \let\kcmd@donot@provide\@empty% it must not be undefined

```

3.5 new key-environments

`\newkeyenvironment`

```

370 \newrobustcmd*\newkeyenvironment{\begingroup
371   \let\kcmd@gb1\@empty\kcmd@star@or@long\new@keyenvironment}
372 \newrobustcmd\renewkeyenvironment{\begingroup
373   \let\kcmd@gb1\@empty\kcmd@star@or@long\renew@keyenvironment}

```

`\new@keyenvironment`

```

374 \def\new@keyenvironment#1{\@testopt{\@newkeyenva{#1}}{}}
375 \long\def\@newkeyenva#1[#2]{%
376   \kcmd@ifstrdigit{#2}%
377   {\@newkeyenv{#1}{[] [] [{#2}]}}
378   {\@testopt{\@newkeyenvb{#1}[#{#2}]}}{}}
379 \long\def\@newkeyenvb#1[#2][#3]{%
380   \kcmd@ifstrdigit{#3}%
381   {\@newkeyenv{#1}[#{#2}] [] [{#3}]}}
382   {\@testopt{\@newkeyenvc{#1}[#{#2}][#{#3}]0}}
383 \long\def\@newkeyenvc#1#2[#3]{\@newkeyenv{#1}{#2[#{#3}]}}
384 \long\def\@newkeyenv#1#2{%
385   \kcmd@ifplus \kcmd@unexpandchar@activate \fi
386   \kcmd@keyenvir@def{#1}{#2}%
387 }
388 \long\def\kcmd@keyenvir@def#1#2#3#4{%
389   \expandafter\let\csname end#1\endcsname\relax
390   \expandafter\kcmd@def\csname #1\expandafter\endcsname\csname end#1\endcsname#2{#3}{#4}%
391 }

```

`\renew@keyenvironment`

```

392 \def\renew@keyenvironment#1{%
393   \@ifundefined{#1}%
394     {\@latex@error{Environment #1 undefined}\@ehc
395     }\relax
396   \cslet{#1}\relax
397   \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:

```
398 \newcommand*\ifcommandkey[1]{\csname @\expandafter\expandafter\expandafter
399   \kcmd@nbk\commandkey{#1}//{first}{second}//%
400   oftwo\endcsname}
```

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

```
401 \newrobustcmd*\showcommandkeys[1]{\let\do\showcommandkey\docsvlist{#1}}
402 \newrobustcmd*\showcommandkey[1]{key \string"#1\string" = %
403   \detokenize\expandafter\expandafter\expandafter{\commandkey{#1}}\par}
404 </package>
```

4 Examples

```
405 <*example>
406 \ProvidesFile{keycommand-example}
407 \documentclass[a4paper]{article}
408 \usepackage[T1]{fontenc}
409 \usepackage[latin1]{inputenc}
410 \usepackage[american]{babel}
411 \usepackage{keycommand, framed, fancyvrb}
412 %
413 \makeatletter
414 \parindent\z@
415 \newkeycommand*\Rule[raise=.4ex,width=1em,thick=.4pt][1]{%
416   \rule[\commandkey{raise}]{\commandkey{width}}{\commandkey{thick}}%
417   #1%
418   \rule[\commandkey{raise}]{\commandkey{width}}{\commandkey{thick}}%
419 }
420 \newkeycommand*\charleads[sep=1][2]{%
421   \ifhmode\else\leavevmode\fi\setbox\@tempboxa\hbox{#2}\@tempdima=1.584wd\@tempboxa%
422   \cleaders\hb@xt@\commandkey{sep}\@tempdima{\hss\box\@tempboxa\hss}#1%
423   \setbox\@tempboxa\box\voidb@x}
424 \newcommand*\charfill[1][\charleads[#{1}]{\hfill\kern\z@}]%
425 \newcommand*\charfil[1][\charleads[#{1}]{\hfil\kern\z@}]%
426 %
427 \newkeyenvironment*{dblruled}[first=.4pt,second=.4pt,sep=1pt,left=\z@]{%
428   \def\FrameCommand{%
429     \vrule\@width\commandkey{first}%
430     \hskip\commandkey{sep}
431     \vrule\@width\commandkey{second}%
432     \hspace{\commandkey{left}}}%
433   \parindent\z@
434   \MakeFramed {\advance\hsize-\width \FrameRestore}}
435   {\endMakeFramed}
436 %
437 \makeatother
438 \begin{document}
439 \title{This is {\tt keycommand-example.tex}}
440 \author{Florent Chervet}
441 \date{July 22, 2009}
442
443 \maketitle
444
445 {\Large Please refer to {\tt keycommand-example.tex} for definitions.}
446
```

```

447 \section{Example of a keycommand : \texttt{\string\Rule}}
448
449 \begin{tabular*}\textwidth{rl}
450 \verb+\Rule[width=2em]{hello}+:\&\Rule[width=2em]{hello}\cr
451 \verb+\Rule[thick=1pt,width=2em]{hello}+:\&\Rule[thick=1pt,width=2em]{hello}\cr
452 \verb+\Rule{hello}+:\&\Rule{hello}\cr
453 \verb+\Rule[thick=1pt,raise=1ex]{hello}+:\&\Rule[thick=1pt,raise=1ex]{hello}
454 \end{tabular*}
455
456 \section{Example of a keycommand : \texttt{\string\charfill}}
457
458 \begin{tabular*}\textwidth{rp{.4\textwidth}}
459 \verb+\charfill{\$star$}+: & \charfill{\$star$}\cr
460 \verb+\charfill[sep=2]{\$star$}+: & \charfill[sep=2]{\$star$} \\
461 \verb+\charfill[sep=.7]{\textasteriskcentered}+: & \charfill[sep=.7]{\textasteriskcentered}
462 \end{tabular*}
463
464
465 \section{Example of a keyenvironment : \texttt{dblruled}}
466
467 Key environment \texttt{dblruled} uses \texttt{framed.sty} and therefore it can be used
468 even if a pagebreak occurs inside the environment:
469 \medskip
470
471 \verb+\begin{dblruled}+\par
472 \verb+  test for dblruled key-environment\par+\par
473 \verb+  test for dblruled key-environment\par+\par
474 \verb+  test for dblruled key-environment+\par
475 \verb+\end{dblruled}+
476
477 \begin{dblruled}
478 test for dblruled key-environment\par
479 test for dblruled key-environment\par
480 test for dblruled key-environment
481 \end{dblruled}
482
483
484 \verb+\begin{dblruled}[first=4pt,sep=2pt,second=.6pt,left=.2em]+\par
485 \verb+  test for dblruled key-environment\par+\par
486 \verb+  test for dblruled key-environment\par+\par
487 \verb+  test for dblruled key-environment+\par
488 \verb+\end{dblruled}+
489
490 \begin{dblruled}[first=4pt,sep=2pt,second=.6pt,left=.2em]
491 test for dblruled key-environment\par
492 test for dblruled key-environment\par
493 test for dblruled key-environment
494 \end{dblruled}
495
496 \end{document}
497 </example>

```

5 History

[2010/04/27 v3.1415]

- Key-environment can now be nested ! (it's not too late... I hope so)
- Keys and mandatory arguments as well can be used in both begin end end part of the environment.

[2010/04/25 v3.141]

- No new feature but a real improvement in optimization.
In particular, keycommand does not load etextools anymore.
- Bug fix for `\providekeycommand`.

[2010/04/18 v3.14]

- Correction of bug in the normalization process.
Correction of a bug in `\ifcommandkey` (undesirable space...)
- Modification of the pdf documentation for the `+` form of key-environments.

[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> | 357, 394 |
| <code>\@firstoftwo</code> | 319 |
| <code>\@ifdefinable</code> | 174, 359 |
| <code>\@ifnextchar</code> | 319 |
| <code>\@ifundefined</code> | 356, 364, 393 |
| <code>\@new@key@command</code> | 346, 350, 351, <u>352</u> |
| <code>\@new@keycommand</code> | 347, <u>348</u> |
| <code>\@newkeycommand</code> | 342, <u>343</u> |
| <code>\@newkeyenv</code> | 377, 381, 383, <u>384</u> |
| <code>\@newkeyenva</code> | 374, 375 |
| <code>\@newkeyenvb</code> | 378, 379 |
| <code>\@newkeyenvc</code> | 382, 383 |
| <code>\@next</code> | 285, 287, 288, 289, 290, 293 |
| <code>\@rc@ifdefinable</code> | 359 |
| <code>\@secondoftwo</code> | 34 |
| <code>\~</code> | 240 |
| A | |
| <code>\active</code> | 240, 243, 328 |
| <code>\afterassignment</code> | 254, 293 |
| <code>\AtEndOfPackage</code> | 24 |
| C | |
| <code>\catcode</code> | 14, 16, 240, 243, 328 |
| <code>\charfil</code> | 425 |
| <code>\charfill</code> | 424, 456, 459, 460, 461 |
| <code>\charleads</code> | 420, 424, 425 |
| <code>\cleaders</code> | 422 |

| | | | |
|---|--|--|---|
| <code>\commandkey</code> | 183, 210, 217, 220, 221, 399, 403, 416, 418, 422, 429, 430, 431, 432 | <code>\kcmd@long</code> | 198, 205, 317, 318 |
| <code>\csdef</code> | 152, 192, 193, 197, 204, 239 | <code>\kcmd@mount@unexpandchar</code> | 189, 238 |
| <code>\csedef</code> | 95, 98, 101, 104, 107, 110, 127, 137 | <code>\kcmd@nargs</code> | 265, 288, 298 |
| <code>\cslet</code> | 53, 155, 159, 184, 396 | <code>\kcmd@nbk</code> | 31, 43, 52, 54, 77, 79, 81, 83, 85, 132, 236, 326, 334, 399 |
| <code>\csletcs</code> | 114, 115 | <code>\kcmd@nil</code> | 119, 126, 128, 236, 237 |
| <code>\csundef</code> | 162, 163 | <code>\kcmd@normalize@braces</code> | 50, 60 |
| D | | | |
| <code>\default</code> | 132, 136, 138 | <code>\kcmd@normalize@setkeys</code> | 44, 234 |
| <code>\define@choicekey</code> | 123, 134 | <code>\kcmd@otherkey</code> | 225, 227, 236 |
| <code>\define@cmdkey</code> | 96 | <code>\kcmd@otherkey@name</code> | 236, 237 |
| <code>\detokenize</code> | 27, 33, 42, 62, 403 | <code>\kcmd@pkg@name</code> | 6, 55, 147, 335 |
| <code>\docsvlist</code> | 156, 167, 401 | <code>\kcmd@plus</code> | 317, 318, 320 |
| E | | | |
| <code>\expandonce</code> | 102, 105, 272, 275, 288, 289, 290 | <code>\kcmd@setdefaults</code> | 165, 223 |
| G | | | |
| <code>\getcommandkey</code> | 183, 209, 217, 219, 220 | <code>\kcmd@setkeys</code> | 225, 227, 233 |
| I | | | |
| <code>\ifcase</code> | 94 | <code>\kcmd@star@or@long</code> | 311, 313, 315, 316, 371, 373 |
| <code>\ifcommandkey</code> | 5, 398 | <code>\kcmd@testopt</code> | 321, 322, 323 |
| <code>\ifcsundef</code> | 154, 166 | <code>\kcmd@undefinekey</code> | 161, 185 |
| <code>\iffalse</code> | 25, 322 | <code>\kcmd@undefinekeys</code> | 153, 187 |
| <code>\ifnum</code> | 26, 299, 300, 301, 302, 303, 304, 305, 306, 307 | <code>\kcmd@unexpandchar</code> | 188, 189, 323, 324 |
| <code>\iftrue</code> | 321 | <code>\kcmd@unexpandchar@activate</code> | 328, 331, 344, 385 |
| K | | | |
| <code>\kcmd@@ifplus</code> | 319, 320 | <code>\kcmd@Xsetkeys</code> | 8, 51 |
| <code>\kcmd@AtEnd</code> | 10, 12, 13, 24 | <code>\kcmd@yarg@edef</code> | 266, 272, 275 |
| <code>\kcmd@check@typeofkey</code> | 64, 94 | <code>\kcmd@yargd@f</code> | 254, 256 |
| <code>\kcmd@check@typeofkey@bool</code> | 71, 77 | <code>\kcmd@yargdef</code> | 205, 230, 248 |
| <code>\kcmd@check@typeofkey@choice</code> | 75, 85 | <code>\kcmd@yargedef</code> | 198, 201, 230, 258 |
| <code>\kcmd@check@typeofkey@choicest</code> | 74, 83 | <code>\kcmd@yargedef@</code> | 278, 286 |
| <code>\kcmd@check@typeofkey@enum</code> | 73, 81 | <code>\kv@list</code> | 50, 51 |
| <code>\kcmd@check@typeofkey@enumst</code> | 72, 79 | <code>\kv@normalize</code> | 49 |
| <code>\kcmd@command</code> | 274, 281 | <code>\kv@parse</code> | 191 |
| <code>\kcmd@def</code> | 171, 353, 390 | <code>\kv@parse@normalized</code> | 50 |
| <code>\kcmd@defcommand</code> | 177, 182 | L | |
| <code>\kcmd@define@boolkey</code> | 99, 131, 133, 139 | <code>\Large</code> | 445 |
| <code>\kcmd@define@choicekey</code> | 102, 105, 108, 111, 121, 122, 129 | <code>\lccode</code> | 240 |
| <code>\kcmd@definekey</code> | 64, 191 | <code>\lowercase</code> | 241 |
| <code>\kcmd@donot@provide</code> | 172, 366, 367, 369 | M | |
| <code>\kcmd@envir</code> | 271, 283 | <code>\medskip</code> | 469 |
| <code>\kcmd@error@handler</code> | 128, 138, 142 | N | |
| <code>\kcmd@fam</code> | 176, 177 | <code>\new@keycommand</code> | 311, 342, 360, 365 |
| <code>\kcmd@firstchoiceof</code> | 119, 126, 128 | <code>\new@keyenvironment</code> | 371, 374, 397 |
| <code>\kcmd@gbl</code> | 185, 187, 191, 198, 205, 218, 311, 313, 315, 371, 373 | <code>\newkeycommand</code> | 2, 196, 203, 310, 337, 415, 420 |
| <code>\kcmd@get@keyname@bool</code> | 78, 98, 99 | <code>\newkeyenvironment</code> | 4, 196, 203, 370, 427 |
| <code>\kcmd@get@keyname@choice</code> | 86, 110, 111 | <code>\newrobustcmd</code> | 44, 310, 312, 314, 370, 372, 401, 402 |
| <code>\kcmd@get@keyname@choicest</code> | 84, 107, 108 | <code>\nr</code> | 108, 111, 124, 134, 137 |
| <code>\kcmd@get@keyname@enum</code> | 82, 104, 105 | <code>\number</code> | 33, 108, 111, 137, 199, 206 |
| <code>\kcmd@get@keyname@enumst</code> | 80, 101, 102 | P | |
| <code>\kcmd@ifplus</code> | 196, 220, 321, 322, 325, 344, 385 | <code>\PackageError</code> | 55, 147, 335 |
| <code>\kcmd@ifstrdigit</code> | 25, 345, 349, 376, 380 | <code>\pdfmatch</code> | 25, 26 |
| <code>\kcmd@ifstrdigit@i</code> | 32, 36 | <code>\protected</code> | 8, 88, 142, 153, 171, 182, 197, 201, 218, 233, 238, 239, 248, 258, 266, 271, 274, 277 |
| <code>\kcmd@ifstrdigit@ii</code> | 37, 39 | <code>\provide@keycommand</code> | 315, 362 |
| <code>\kcmd@keyenvir@def</code> | 386, 388 | <code>\providekeycommand</code> | 314 |
| <code>\kcmd@keyfam</code> | 42, 176 | R | |
| | | <code>\renew@keycommand</code> | 313, 354, 366 |
| | | <code>\renew@keyenvironment</code> | 373, 392 |
| | | <code>\renewkeycommand</code> | 312 |
| | | <code>\renewkeyenvironment</code> | 372 |
| | | <code>\Rule</code> | 415, 447, 450, 451, 452, 453 |
| | | <code>\rule</code> | 416, 418 |

| | | | |
|-------------------------------------|---|----------------------------------|-------------------------|
| S | | V | |
| <code>\setkeys</code> | 8 | <code>\val</code> | 102, 105, 124, 134, 148 |
| <code>\showcommandkey</code> | 401, 402 | | |
| <code>\showcommandkeys</code> | 401 | | |
| T | | X | |
| <code>\TMP@EnsureCode</code> | 11, 18, 19, 20, 21, 22, 23 | <code>\XKV@plfalse</code> | 8 |
| | | <code>\XKV@rm</code> | 53, 54, 56 |
| U | | <code>\XKV@setkeys</code> | 8 |
| <code>\undef</code> | 24 | <code>\XKV@sttrue</code> | 8 |
| <code>\unexpanded</code> | 51, 119, 125, 127, 136, 138, 189, 191, 194, 215, 244, 268, 290, 335, 336 | <code>\XKV@testoptc</code> | 8 |