



The completely redesigned keycommand* package



key-value interface for commands and environments in L^AT_EX.

<florent.chervet@free.fr>

2010/04/18 – version 3.14

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 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 <code>+</code> form	3
1.4 Explanation of the <code>+</code> form	3
1.5 key-environments	4
2 Messages and more	4
2.1 Invalid keys	4
2.2 Testing keys	4
2.3 <code>xkeyval</code> , <code>keyval</code> and <code>kvsetkeys</code> comparison	5
3 Implementation	5
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
4 Examples	12
5 History	14
[2010/04/18 v3.14]	14
[2010/03/28 v3.0]	14
[2009/07/22 v1.0]	14
6 References	14
7 Index	14

* 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

1 User Interface

1.1 General syntax

\newkeycommand *+[short-unexpand] {\{<command>\}} [{<keys=defaults>}][({<OptKey>})][({<n>})] {\{<definition>\}}

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

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 explained in section 1.3.

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:	<code>\textrule[width=2em]{hello}</code>	→	——hello——
2:	<code>\textrule[thick=5pt,width=2em]{hello}</code>	→	████████hello████████
3:	<code>\textrule{hello}</code>	→	——hello——
4:	<code>\textrule[thick=2pt,raise=1ex]{hello}</code>	→	████████hello████████
<i>et cetera.</i>			

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 *+[short-unexpand]	{<envir name>}	[<keys=defaults>]	[<OptKey>]	[<<n>>]	{<begin>}	{<end>}
	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 (*i.e.* 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: *<begin>* and *<end>* are then expanded, and the optional **[N]** argument protects from expansion in each of those parts.

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 <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
\setkeys{fam}{key={{value}}}	macro:->value	macro:->value	macro:->{ value }
\setkeys{fam}{key={{ {value} }}}}	macro:->{ value }	macro:->value	macro:->{ { value } }
\setkeys{fam}{key=_{{ {value} }}}}	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   [2010/04/18 v3.14 - 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 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 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   \ettl@nbk//% undeclared keys are assigned to "OtherKeys"
40   {\csedef{#2@#3}{\expandonce{\XKV@rm}}% (if specified, ie not empty)
41   {\expandafter\ettl@nbk\XKV@rm//% (otherwise a recoverable error is thrown)
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{%
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{%
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//{\ettl@nbk#1//}
63 \def\kcmd@get@keyname@bool #1bool #2//{#2}
64 \def\kcmd@check@typeofkey@enumst #1enum* #2//{\ettl@nbk#1//}
65 \def\kcmd@get@keyname@enumst #1enum* #2//{#2}
66 \def\kcmd@check@typeofkey@enum #1enum #2//{\ettl@nbk#1//}
67 \def\kcmd@get@keyname@enum #1enum #2//{#2}
68 \def\kcmd@check@typeofkey@choicest #1choice* #2//{\ettl@nbk#1//}
69 \def\kcmd@get@keyname@choicest #1choice* #2//{#2}
70 \def\kcmd@check@typeofkey@choice #1choice #2//{\ettl@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{%
136   #1 = global, #2 = family
137   \ifcsundef{#2.keylist}
138     {}
139     {\expandnext{\csvloop[\kcmd@undefinekey{#1}{#2}]}{\csname #2.keylist\endcsname}}%
140     \cslet{#2.keylist}\noexpand\gobble
141   #1\csundef{KV@#2@#3}%
142   #1\csundef{KV@#2@#3@default}%
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\let\getcommandkey\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       % \let\getcommandkey\noexpandcs{\kcmd@fam.getcommandkey}%
158     \kcmd@plus% \newkeycommand+
159     \csdef{\kcmd@fam}{\kcmd@yargedef{\kcmd@gbl}{\kcmd@long}{\noexpandcs\kcmd@fam}%
160           {\number#4}{#6}{\csname\kcmd@fam.unexpandchar\endcsname}}%
161     \unless\ifx#6\relax% that means we have to define a key-environment
162       \def#6{\kcmd@yargedef{\kcmd@gbl}{\kcmd@long}{#6}}%
```

```

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

```

188 \protected\def\kcmd@mount@unexpandchar#1#2{%
189     \etl@nbk#2//{%
190         \protected\csdef{#1.unexpandchar}{\begingroup
191             \catcode`\~\active \lccode`\~`#2 \lccode`#2 0\relax
192             \lowercase{\def~{\catcode`#2\active
193                 \long\def~#####1{\unexpanded{#####1}}}}%
194             \etl@aftergroup\def~\endgroup~}}%
195     {\cslet{#1.unexpandchar}{\empty} //%
196 }

```

\kcmd@yargdef

This is the `argdef` macro for the normal (non +) form:

```

197 \protected \def \kcmd@yargdef #1#2#3#4#5{\begingroup
198 % #1 = global or {}
199 % #2 = long or {}
200 % #3 = Command
201 % #4 = nr of args
202 % #5 = endenvir (or \relax if not an environment)
203 \def \kcmd@yargd@f ##1##2##{\afterassignment#5\endgroup
204     #1##2\expandafter\def\expandafter#3@gobble ##1##4%
205 } \kcmd@yargd@f 0##1##2##3##4##5##6##7##8##9##4%
206 }

```

\kcmd@yargedef

This is the `argdef` macro for the `+` form:

```

207 \protected\def\kcmd@yargedef#1#2#3#4#5#6{\begingroup
208 % #1 = global or {}
209 % #2 = long or {}
210 % #3 = Command

```

```

211 % #4 = nr of args
212 % #5 = endenvir (or \relax if not an environment)
213 % #6 = unexpandchar mounting macro
214   \letcs{kcmd@nargs}{kcmd@#4of9}%
215   \long\def{kcmd@yarg@edef##1##2}{\afterassignment#5\endgroup
216     #1\edef#3{\begingroup #6%
217       #2\edef\unexpanded{#3##2}{\endgroup\unexpanded{##1}%
218       }\noexpand#3}%
219   }%
220 \protected\def{kcmd@yarg@body}{\edef\body{\unexpanded\expandafter\expandafter\expandafter{%
221   \expandafter#3\kcmd@nargs{####1}{####2}{####3}{####4}{####5}{####6}{####7}{##%
222 }}\expandafter\kcmd@yarg@edef\expandafter{\body}}%
223 \def{kcmd@yarged@f##1##2##3##4##5##6##7##8##9##4%
224   \edef\kcmd@yarged@next{\kcmd@yarg@body{\expandonce{@gobble ##1##4}}}%
225   \afterassignment\kcmd@yarged@next
226   \expandafter\def\expandafter#3@gobble ##1##4%
227 }{\kcmd@yarged@f 0##1##2##3##4##5##6##7##8##9##4%
228 }

```

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

```

229 \def{kcmd@XofNINE##1}{%
230   \def{kcmd@XofNine##1##2##3##4##5##6##7##8##9##}{%
231     \expandafter\csedef{kcmd@#1of9}####1####2####3####4####5####6####7####8####9{%
232       @gobble ##1##1}%
233   }{\kcmd@XofNine 0##1##2##3##4##5##6##7##8##9##1}%
234 }
235 \csvloop*[\kcmd@XofNINE]{1,2,3,4,5,6,7,8,9,0}
236 \undef\kcmd@XofNine\undef\kcmd@XofNINE

```

3.4 new key-commands

\newkeycommand Here are the entry points:

```

237 \protected\def\newkeycommand{\begingroup
238   \let\kcmd@gb1\empty\kcmd@modifiers\new@keycommand}
239 \protected\def\renewkeycommand{\begingroup
240   \let\kcmd@gb1\empty\kcmd@modifiers\renew@keycommand}
241 \protected\def\providekeycommand{\begingroup
242   \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:

```

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

```

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

```

250 \def\kcmd@unexpandchar##1##2{\def\kcmd@unexpandchar##2% only once inside group...
251   \kcmd@plus \ettl@nbk##2//%
252     {\def\kcmd@unexpandchar@activate{\catcode`##2 \active}%
253     {\let\kcmd@unexpandchar@activate\relax}%%
254   \else\ettl@nbk##2//%

```

```

255      {\PackageError\kcmd@name{shortcut option for \string\unexpanded\MessageBreak
256      You can't use a shortcut option for \string\unexpanded\MessageBreak
257      without the \string+ form of \string\newkeycommand!}%
258      {I will ignore this option and proceed.}%
259      \let\kcmd@unexpandchar@\empty}%
260      {}//%
261      \fi#1}

```

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

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

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

```

263 \protected\def@\newkeycommand#1[#2]{% #2 = key=values or N=mandatory args
264     \kcmd@plus \kcmd@unexpandchar@activate \fi% activates unexpand-char before reading definition
265     \ifstrnum{#2}{%
266         {\@new@key@command#1[] [] [{#2}]}% no kv, no optkey, number of args
267         {\@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):

```

268 \protected\def@\new@keycommand#1[#2][#3]{%
269     \ifstrnum{#3}{%
270         {\@new@key@command#1[{#2}] [] [{#3}]}% no optkey
271         {\@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:

```
272 \protected\long\def@\new@key@command#1[#2][#3][#4]{%
273     \kcmd@def#1\relax[{#2}][{#3}][{#4}]{#5}{}}
```

\renew@keycommand

```

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

```

\provide@keycommand

```

281 \protected\def\provide@keycommand#1{%
282     \escapechar\m@ne\edef\@gtempa{{\string#1}}%
283     \expandafter\@ifundefined\@gtempa
284         {\new@keycommand#1}
285         {\@testopt{\provide@key@command\kcmd@notdefinable}{}}}
286 \protected\def\provide@key@command#1[#2]{\@tempswafalse\kcmd@def#1{#2}}

```

3.5 new key-environments

\newkeyenvironment

```

287 \protected\def\newkeyenvironment{\begingroup
288     \let\kcmd@gb1\@empty\kcmd@modifiers\new@keyenvironment}
289 \protected\def\renewkeyenvironment{\begingroup
290     \let\kcmd@gb1\@empty\kcmd@modifiers\renew@keyenvironment}

```

\new@keyenvironment

```

291 \def\new@keyenvironment#1{\@testopt{\@newkeyenva{#1}}{}}
292 \def\@newkeyenva[#2]{%
293   \ifstrnum{#2}{%
294     {\@newkeyenv{#1}{}[][\{#2\}]}}
295     {\@testopt{\@newkeyenvb{#1}[\{#2\}]}{}}}
296 \def\@newkeyenvb[#2][#3]{%
297   \ifstrnum{#3}{%
298     {\@newkeyenv{#1}{[\{#2\}][[\{#3\}]]}}
299     {\@testopt{\@newkeyenvc{#1}{[\{#2\}][[\{#3\}]]}}{0}}}
300 \def\@newkeyenvc#1#2[#3]{\@newkeyenv{#1}{#2[\{#3\}]}}
301 \long\def\@newkeyenv#1#2{%
302   \kcmd@plus \kcmd@unexpandchar@activate \fi
303   \kcmd@keyenvir@def{#1}{#2}{%
304 }
305 \long\def\kcmd@keyenvir@def#1#2#3#4{%
306   \cslet{end#1}\protected
307   \expandafter\kcmd@def\csname #1\expandafter\endcsname\csname end#1\endcsname#2{#3}{#4}{%
308 }

```

\renew@keyenvironment

```

309 \def\renew@keyenvironment#1{%
310   \@ifundefined{#1}{%
311     {\@latex@error{Environment #1 undefined}\@ehc
312   }\relax
313   \cslet{#1}\relax
314   \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:

```

315 \newcommand*\ifcommandkey[1]{\csname @\expandafter\expandafter\expandafter
316   \ettt@nbk\commandkey{#1}//{first}{second}//%
317   oftwo\endcsname}

```

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

```

318 \newcommand\showcommandkeys[1]{\let\do\showcommandkey\docslist{#1}}
319 \newcommand\showcommandkey[1]{key \string"#1\string" = %
320   \expandnext\expandnext\expandnext\detokenize{\commandkey{#1}}\par}
321 </package>

```

4 Examples

```

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

```

```
333 \rule[\commandkey{raise}]{\commandkey{width}}{\commandkey{thick}}%
334 #1%
335 \rule[\commandkey{raise}]{\commandkey{width}}{\commandkey{thick}}}
336
337 \newkeycommand\charleads[sep=1][2]{%
338   \ifhmode\else\leavevmode\fi\setbox\@tempboxa\hbox{\#2}\@tempdima=1.584\wd\@tempboxa%
339   \cleaders\hb@xt@\commandkey{sep}\@tempdima{\hss\box\@tempboxa\hss}\#1%
340   \setbox\@tempboxa\box\voidb@x}
341 \newcommand\charfill[1][]{\charleads[{\#1}]\hfill\kern\z@}
342 \newcommand\charfil[1][]{\charleads[{\#1}]\hfil\kern\z@}
343 %
344 \newkeyenvironment{dblruled}[first=.4pt,second=.4pt,sep=1pt,left=\z@]{%
345   \def\FrameCommand{%
346     \vrule@width\commandkey{first}%
347     \hskip\commandkey{sep}%
348     \vrule@width\commandkey{second}%
349     \hskip\commandkey{left}}%
350   \parindent\z@
351   \MakeFramed {\advance\hsize-\width \FrameRestore}%
352   \endMakeFramed}
353 %
354 \makeatother
355 \begin{document}
356 \title{This is {\tt keycommand-example.tex}}
357 \author{Florent Chervet}
358 \date{July 22, 2009}
359
360 \maketitle
361
362 {\Large Please refer to {\tt keycommand-example.tex} for definitions.}
363
364 \section{Example of a keycommand : \texttt{\{string\}\Rule{}}
```

365

```
366 \begin{tabular*}{\textwidth}{rl}
367 \verb+\Rule[width=2em]{hello}+:&\verb+\Rule[width=2em]{hello}\cr
368 \verb+\Rule[thick=1pt,width=2em]{hello}+:&\verb+\Rule[thick=1pt,width=2em]{hello}\cr
369 \verb+\Rule[hello]+:&\verb+\Rule[hello]\cr
370 \verb+\Rule[thick=1pt,raise=1ex]{hello}+:&\verb+\Rule[thick=1pt,raise=1ex]{hello}%
371 \end{tabular*}
```

372

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

374

```
375 \begin{tabular*}{\textwidth}{rp{.4\textwidth}}
376 \verb+\charfill{$\star$}+ & \verb+\charfill{$\star$}\cr
377 \verb+\charfill[sep=2]{$\star$}+ & \verb+\charfill[sep=2]{$\star$} \\
378 \verb+\charfill[sep=.7]{\textasteriskcentered}+ & \verb+\charfill[sep=.7]{\textasteriskcentered}%
379 \end{tabular*}
```

380

```
381
```

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

383

```
384 \verb+\begin{dblruled}+\par
385 \verb+ test for dblruled key-environment\par+\par
386 \verb+ test for dblruled key-environment\par+\par
387 \verb+ test for dblruled key-environment+\par
388 \verb+\end{dblruled}+
389
390 \begin{dblruled}
391 test for dblruled key-environment\par
392 test for dblruled key-environment\par
393 test for dblruled key-environment
394 \end{dblruled}
```

```

395
396
397 \verb+\begin{dblruled}[first=4pt,sep=2pt,second=.6pt,left=.2em]+\par
398 \verb+    test for dblruled key-environment\par+\par
399 \verb+    test for dblruled key-environment\par+\par
400 \verb+    test for dblruled key-environment+\par
401 \verb+\end{dblruled}+
402
403 \begin{dblruled}[first=4pt,sep=2pt,second=.6pt,left=.2em]
404 test for dblruled key-environment\par
405 test for dblruled key-environment\par
406 test for dblruled key-environment
407 \end{dblruled}
408
409 \end{document}
410 </example>

```

5 History

[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>	277 , 311	<code>\@new@key@command</code>	266 , 270 , 271 , 272
<code>\@ifdefinable</code>	145 , 279	<code>\@new@keycommand</code>	267 , 268
<code>\@ifundefined</code>	276 , 283 , 310	<code>\@newkeycommand</code>	262 , 263
		<code>\@newkeyenv</code>	294 , 298 , 300 , 301

\@newkeyenva	291, 292
\@newkeyenvb	295, 296
\@newkeyenvc	299, 300
\@nil	103, 109, 111, 186, 187
\@rc@ifdefinable	279
\@swaparg	38
\~	191
A	
\active	191, 192, 252
\afterassignment	203, 215, 225
\AtEndOfPackage	29
B	
\body	220, 222
C	
\catcode	19, 21, 191, 192, 252
\charfil	342
\charfill	341, 373, 376, 377, 378
\charleads	337, 341, 342
\cleaders	339
\commandkey	147, 156, 173, 316, 320, 333, 335, 339, 346, 347, 348, 349
\csdef	134, 152, 153, 159, 166, 190
\csedef	40, 80, 83, 86, 89, 92, 95, 110, 120, 231
\cslet	139, 195, 306, 313
\csletcs	99, 100
\csundef	141, 142
\csvloop	138, 177, 235
D	
\default	115, 119, 121
\define@choicekey	106, 117
\define@cmdkey	81
\detokenize	30, 48, 320
\docslist	318
E	
\ettl@aftergroup@def	194
\ettl@nbk	39, 41, 62, 64, 66, 68, 70, 115, 186, 189, 248, 249, 251, 254, 316
\expandnext	138, 320
\expandonce	40, 47, 87, 90, 119, 121, 150, 224
F	
\futuredef	244
G	
\getcommandkey	147, 157, 175
I	
\ifcase	79
\ifcommandkey	4, 315
\ifcsundef	136
\iffalse	249
\ifstrnum	265, 269, 293, 297
\iftrue	249
K	
\kcmd@@therkeys	186, 187
\kcmd@AtEnd	15, 17, 18, 29
\kcmd@check@typeofkey	49, 79
\kcmd@check@typeofkey@bool	56, 62
\kcmd@check@typeofkey@choice	60, 70
\kcmd@check@typeofkey@choicest	59, 68
\kcmd@check@typeofkey@enum	58, 66
\kcmd@check@typeofkey@enumst	57, 64
\kcmd@def	144, 273, 286, 307
\kcmd@defcommand	145, 146, 148, 181
\kcmd@define@boolkey	84, 114, 116, 122
\kcmd@define@choicekey	87, 90, 93, 96, 105, 112
\kcmd@definekey	49, 151
\kcmd@error@handler	111, 121, 124
\kcmd@fam	147, 149, 150, 151, 152, 153, 155, 156, 157, 159, 160, 163, 166, 174, 176, 177, 179, 180
\kcmd@firstchoiceof	103, 109, 111
\kcmd@gbl	149, 151, 159, 162, 166, 169, 172, 238, 240, 242, 288, 290
\kcmd@get@keyname@bool	63, 83, 84
\kcmd@get@keyname@choice	71, 95, 96
\kcmd@get@keyname@choicest	69, 92, 93
\kcmd@get@keyname@enum	67, 89, 90
\kcmd@get@keyname@enumst	65, 86, 87
\kcmd@ifplus	246, 249
\kcmd@ifstar	245, 248
\kcmd@keyenvir@def	303, 305
\kcmd@keyfam	30, 147
\kcmd@long	159, 162, 166, 169, 245
\kcmd@modifiers	238, 240, 242, 243, 288, 290
\kcmd@mount@unexpandchar	150, 188
\kcmd@nargs	214, 221, 229
\kcmd@normalize@braces	37, 46
\kcmd@normalize@setkeys	31, 184
\kcmd@notdefinable	285
\kcmd@otherkeys	179, 186
\kcmd@pkg@name	6, 10, 42, 129, 255
\kcmd@plus	156, 158, 174, 246, 251, 264, 302
\kcmd@resetdefault	177, 185
\kcmd@setkeys	179, 183
\kcmd@undefinedkey	138, 140
\kcmd@undefinedkeys	135, 149
\kcmd@unexpandchar	150, 247, 250
\kcmd@unexpandchar@activate	252, 253, 264, 302
\kcmd@XofNINE	229, 235, 236
\kcmd@XofNine	230, 233, 236
\kcmd@Xsetkeys	8, 38
\kcmd@yarg@body	220, 224
\kcmd@yarg@edef	215, 222
\kcmd@yargd@f	203, 205
\kcmd@yargdef	166, 169, 197
\kcmd@yarged@f	223, 227
\kcmd@yargedef	159, 162, 207
\kcmd@yargedef@next	224, 225
\kv@list	36, 37, 38, 47
\kv@normalize	36
\kv@parse	151
\kv@parse@normalized	37
L	
\Large	362
\lccode	191
\letcs	214
\lowercase	192
N	
\new@keycommand	238, 262, 280, 284

<pre>\new@keyenvironment 288, <u>291</u>, 314 \newkeycommand 2, 158, 165, <u>237</u>, 257, 332, 337 \newkeyenvironment 4, <u>287</u>, 344 \newrobustcmd 31 \noexpandcs 156, 157, 159, 166, 173, 175, 177, 180, 185 \nr 93, 96, 107, 117, 120 \number 93, 96, 120, 160, 167</pre>	<pre>\rule 333, 335 S \setkeys 8 \showcommandkey 318, 319 \showcommandkeys <u>318</u></pre>
P	T
<pre>\PackageError 10, 42, 129, 255 \protected 73, 124, 144, 146, 172, 183, 188, 190, 197, 207, 220, 237, 239, 241, 243, 262, 263, 268, 272, 274, 281, 286, 287, 289, 306 \provide@key@command 285, 286 \provide@keycommand 242, <u>281</u> \providekeycommand 241</pre>	<pre>\TMP@EnsureCode 16, 23, 24, 25, 26, 27, 28 U \undef 29, 236 \unexpanded 48, 103, 108, 110, 151, 154, 193, 217, 220, 255, 256 \unless 161, 168</pre>
R	V
<pre>\renew@keycommand 240, <u>274</u> \renew@keyenvironment 290, <u>309</u> \renewkeycommand 239 \renewkeyenvironment 289 \Rule 332, 364, 367, 368, 369, 370</pre>	<pre>\val 87, 90, 107, 117, 130 X \XKV@rm 40, 41, 43 \XKV@setkeys 8 \XKV@testopta 8 \XKV@testoptc 8</pre>