The completely redesigned

keycommand* package

key-value interface for commands and environments in LaTeX.

<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 ε-TeX distribution of LaTeX.

Contents

* 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

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

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 LaTeX-\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 *false*) 1 (for *true*)
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 **in**sensitive
choice	choice position={left,centered,right}	\commandkey{position} is: 0 (for *left* the default value), 1 (for *centered*) 2 (for *right*)
	choice∗position={left,centered,right}	This is the same, except match is case **in**sensitive

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 \commankey{⟨*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 ⟨*name*⟩ is not defined as a key, you will get the TeX 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 LaTeX so we don't check if it is loaded twice.

```
1 ⟨*package⟩
2 \NeedsTeXFormat{LaTeX2e}% LaTeX 2.09 can't be used (nor non-LaTeX)
3    [2005/12/01]% LaTeX must be 2005/12/01 or younger (see kvsetkeys.dtx).
4 \ProvidesPackage{keycommand}
5    [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 ε-TeX primitives in keycommand we also load the etex package in order to get an error message if ε-TeX 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}%
```

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

Some `\catcode` assertions internally used by keycommand:

```
\let\kcmd@AtEnd\@empty
\def\TMP@EnsureCode#1#2{%
  \edef\kcmd@AtEnd{%
    \kcmd@AtEnd
    \catcode#1 \the\catcode#1\relax
  }%
  \catcode#1 #2\relax
}
\TMP@EnsureCode{32}{10}% space
\TMP@EnsureCode{61}{12}% = sign
\TMP@EnsureCode{45}{12}% - sign
\TMP@EnsureCode{62}{12}% > sign
\TMP@EnsureCode{46}{12}% . dot
\TMP@EnsureCode{47}{8}% / slash (etextools)
\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:

```
\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:

```
\newrobustcmd\kcmd@normalize@setkeys[4]{%
% #1 = key-command,
% #2 = family,
% #3 = other-key,
% #4 = key-values pairs
  \kv@normalize{#4}\expandafter\let\expandafter\kv@list\expandafter\@empty
  \expandafter\kv@parse@normalized\expandafter{\kv@list}{\kcmd@normalize@braces{#2}}%
  \expandafter\@swaparg\expandafter{\kv@list,}{\kcmd@Xsetkeys*{#2}}%
  \ettl@nbk#3//% undeclared keys are assigned to "OtherKeys"
     {\csedef{#2@#3}{\expandonce{\XKV@rm}}}% (if specified, ie not empty)
     {\expandafter\ettl@nbk\XKV@rm//% (otherwise a recoverable error is thown)
        {\PackageError\kcmd@pkg@name{The key-value pairs :\MessageBreak
        \XKV@rm\MessageBreak
        cannot be processed for key-command \string#1\MessageBreak
        See the definition of the key-command!}{}{}{}//}//}
\long\def\kcmd@normalize@braces#1#2#3{% This is kvsetkeys processor for normalizing braces
  \edef\kv@list{\expandonce\kv@list,%
        #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*) :

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

```
% 2 if enum*,
% 3 if enum,
% 4 if choice*,
% 5 if choice
   \kcmd@check@typeofkey@bool#1bool //%
      {\kcmd@check@typeofkey@enumst#1enum* //%
         {\kcmd@check@typeofkey@enum#1enum //%
            {\kcmd@check@typeofkey@choicest#1choice* //%
               {\kcmd@check@typeofkey@choice#1choice //%
                  05//}4//}3//}2//}1//}
\def\kcmd@check@typeofkey@bool #1bool #2//{\ettl@nbk#1//}
\def\kcmd@get@keyname@bool #1bool #2//{#2}
\def\kcmd@check@typeofkey@enumst #1enum* #2//{\ettl@nbk#1//}
\def\kcmd@get@keyname@enumst #1enum* #2//{#2}
\def\kcmd@check@typeofkey@enum #1enum #2//{\ettl@nbk#1//}
\def\kcmd@get@keyname@enum #1enum #2//{#2}
\def\kcmd@check@typeofkey@choicest #1choice* #2//{\ettl@nbk#1//}
\def\kcmd@get@keyname@choicest #1choice* #2//{#2}
\def\kcmd@check@typeofkey@choice #1choice #2//{\ettl@nbk#1//}
\def\kcmd@get@keyname@choice #1choice #2//{#2}
%
\protected\long\def\kcmd@definekey#1#2#3#4#5{% define the keys using xkeyval macros
% #1 = keycommand,
% #2 = \global,
% #3 = family,
% #4 = key (before = sign),
% #5 = default (after = sign)
   \ifcase\kcmd@check@typeofkey{#4}\relax% standard
      #2\csedef{#3.keylist}{\csname#3.keylist\endcsname,#4}%
      \define@cmdkey{#3}[{#3@}]{#4}[{#5}]{}%
   \or% bool
      #2\csedef{#3.keylist}{\csname#3.keylist\endcsname,\kcmd@get@keyname@bool#4//}%
      \kcmd@define@boolkey#1{#3}{\kcmd@get@keyname@bool#4//}{#5}%
   \or% enum*
      #2\csedef{#3.keylist}{\csname#3.keylist\endcsname,\kcmd@get@keyname@enumst#4//}%
      \kcmd@define@choicekey#1*{#3}{\kcmd@get@keyname@enumst#4//}{#5}{\expandonce\val}%
   \or% enum
      #2\csedef{#3.keylist}{\csname#3.keylist\endcsname,\kcmd@get@keyname@enum#4//}%
      \kcmd@define@choicekey#1{}{#3}{\kcmd@get@keyname@enum#4//}{#5}{\expandonce\val}%
   \or% choice*
      #2\csedef{#3.keylist}{\csname#3.keylist\endcsname,\kcmd@get@keyname@choicest#4//}%
      \kcmd@define@choicekey#1*{#3}{\kcmd@get@keyname@choicest#4//}{#5}{\number\nr}%
   \or% choice
      #2\csedef{#3.keylist}{\csname#3.keylist\endcsname,\kcmd@get@keyname@choice#4//}%
      \kcmd@define@choicekey#1{}{#3}{\kcmd@get@keyname@choice#4//}{#5}{\number\nr}%
   \fi
   \ifx#2\global\relax
      #2\csletcs{KV@#3@#4}{KV@#3@#4}% globalize
      #2\csletcs{KV@#3@#4@default}{KV@#3@#4@default}% globalize default value
   \fi}
%
\long\def\kcmd@firstchoiceof#1,#2\@nil{\unexpanded{#1}}
%
\long\def\kcmd@define@choicekey#1#2#3#4#5#6{\begingroup\edef\kcmd@define@choicekey{\endgroup
      \noexpand\define@choicekey#2+{#3}{#4}
            [\noexpand\val\noexpand\nr]%
            {\unexpanded{#5}}% list of allowed values
            [{\kcmd@firstchoiceof#5,\@nil}]% default value
            {\csedef{#3@#4}{\unexpanded{#6}}}% define key value if in the allowed list
            {\kcmd@error@handler\noexpand#1{#3}{#4}{\kcmd@firstchoiceof#5,\@nil}}% error handler
   }\kcmd@define@choicekey}
%
```

```
\def\kcmd@define@boolkey#1#2#3#4{\begingroup
   \ettl@nbk#4//{\def\default{#4}}{\def\default{true}}//%
   \edef\kcmd@define@boolkey{\endgroup
      \noexpand\define@choicekey*+{#2}{#3}[\noexpand\val\noexpand\nr]%
            {false,true}
            [{\expandonce\default}]%
            {\csedef{#2@#3}{\noexpand\number\noexpand\nr}}%
            {\kcmd@error@handler\noexpand#1{#2}{#3}{\expandonce\default}}%
   }\kcmd@define@boolkey}
%
\protected\long\def\kcmd@error@handler#1#2#3#4{%
% #1 = key-command,
% #2 = family,
% #3 = key,
% #4 = default
   \PackageError\kcmd@pkg@name{%
      Value `\val\space' is not allowed in key #3\MessageBreak
      for key-command \string#1.\MessageBreak
      I'll use the default value `#4' for this key.\MessageBreak
      See the definition of the key-command!}{%
      \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).

```
\def\kcmd@undefinekeys#1#2{% #1 = global, #2 = family
   \ifcsundef{#2.keylist}
      {}
      {\expandnext{\csvloop[\kcmd@undefinekey{#1}{#2}]}{\csname #2.keylist\endcsname}}%
      \cslet{#2.keylist}\noexpand\@gobble}
\def\kcmd@undefinekey#1#2#3{% #1 = global, #2 = family, #3 = key
   #1\csundef{KV@#2@#3}%
   #1\csundef{KV@#2@#3@default}%
}
```

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

```
\protected\long\def\kcmd@def#1#2[#3][#4][#5]#6#7{%
   \@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))

```
\protected\long\def\kcmd@defcommand#1[#2][#3][#4]#5#6#7{%
   \edef\kcmd@fam{\kcmd@keyfam{#1}}\let\commandkey\relax
   \edef\kcmd@defcommand{\endgroup
      \kcmd@undefinekeys{\kcmd@gbl}{\kcmd@fam}% undefines all keys for this keycommand family
      \kcmd@mount@unexpandchar{\kcmd@fam}{\expandonce\kcmd@unexpandchar}%
      \unexpanded{\kv@parse{#2,#3}}{\kcmd@definekey\noexpand#1{\kcmd@gbl}{\kcmd@fam}}% defines key
      \csdef{\kcmd@fam.commandkey}####1{\noexpand\csname\kcmd@fam @####1\endcsname}%
      \csdef{\kcmd@fam.getcommandkey}####1{%
         \unexpanded{\unexpanded\expandafter\expandafter\expandafter}{%
                        \noexpand\csname\kcmd@fam @####1\endcsname}}%
      \let\commandkey\noexpandcs{\kcmd@fam.\kcmd@plus get\fi commandkey}%
      \kcmd@plus% \newkeycommand+
         \csdef{\kcmd@fam}{\kcmd@yargedef{\kcmd@gbl}{\kcmd@long}{\noexpandcs\kcmd@fam}
                           {\number#4}{#6}{\csname\kcmd@fam.unexpandchar\endcsname}}%
         \unless\ifx#6\relax% that means we have to define a key-environment
            \def#6{\kcmd@yargedef{\kcmd@gbl}{\kcmd@long}{#6}%
                           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}0\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{\ettl@nbk#1//{\kcmd@@therkeys#1=\@nil}{}//}
184 \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.

```
185 \protected\def\kcmd@mount@unexpandchar#1#2{%
186    \ettl@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             \ettl@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
```

```
    \letcs\kcmd@nargs{kcmd@#4of9}%
    \long\def\kcmd@yarg@edef##1##2{\afterassignment#5\endgroup
       #1\edef#3{\begingroup #6%
          #2\edef\unexpanded{#3##2}{\endgroup\unexpanded{##1}%
          }\noexpand#3}%
    }%
    \protected\def\kcmd@yarg@body{\edef\body{\unexpanded\expandafter\expandafter\expandafter{%
       \expandafter#3\kcmd@nargs{{####1}}{{####2}}{{####3}}{{####4}}{{####5}}{{####6}}{{####7}}{{##
    }\expandafter\kcmd@yarg@edef\expandafter{\body}}%
    \def\kcmd@yarged@f##1#4##2##{%
       \edef\kcmd@yargedef@next{\kcmd@yarg@body{\expandonce{\@gobble ##1#4}}}%
       \afterassignment\kcmd@yargedef@next
       \expandafter\def\expandafter#3\@gobble ##1#4%
    }\kcmd@yarged@f 0##1##2##3##4##5##6##7##8##9###4%
}
```

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

```
\def\kcmd@XofNINE#1{%
   \def\kcmd@XofNine##1#1##2##{%
      \expandafter\csedef{kcmd@#1of9}####1####2####3####4####5####6####7####8####9{%
                              \@gobble ##1#1}%
   }\kcmd@XofNine 0##1##2##3##4##5##6##7##8##9###1{}%
}
\csvloop*[\kcmd@XofNINE]{1,2,3,4,5,6,7,8,9,0}
\undef\kcmd@XofNine\undef\kcmd@XofNINE
```

3.4 new key-commands

\newkeycommand Here are the entry points:

```
\protected\def\newkeycommand{\begingroup
   \let\kcmd@gbl\@empty\kcmd@modifiers\new@keycommand}
\protected\def\renewkeycommand{\begingroup
   \let\kcmd@gbl\@empty\kcmd@modifiers\renew@keycommand}
\protected\def\providekeycommand{\begingroup
   \let\kcmd@gbl\@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:

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

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

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

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

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

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

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

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

```
\protected\def\@new@keycommand#1[#2][#3]{%
   \ifstrnum{#3}%
      {\@new@key@command#1[{#2}][][{#3}]}% no optkey
      {\@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 environements). The so called "unexpand-char shortcut" has been activated before reading command definition:

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

`\renew@keycommand`

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

`\provide@keycommand`

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

3.5 new key-environments

`\newkeyenvironment`

```
\protected\def\newkeyenvironment{\begingroup
   \let\kcmd@gbl\@empty\kcmd@modifiers\new@keyenvironment}
\protected\def\renewkeyenvironment{\begingroup
   \let\kcmd@gbl\@empty\kcmd@modifiers\renew@keyenvironment}
```

\new@keyenvironment

```
\def\new@keyenvironment#1{\@testopt{\@newkeyenva{#1}}{}}
\def\@newkeyenva#1[#2]{%
   \ifstrnum{#2}%
      {\@newkeyenv{#1}{[][][{#2}]}}
      {\@testopt{\@newkeyenvb{#1}[{#2}]}{}}}
\def\@newkeyenvb#1[#2][#3]{%
   \ifstrnum{#3}%
      {\@newkeyenv{#1}{[{#2}][][{#3}]}}
      {\@testopt{\@newkeyenvc{#1}{[{#2}][{#3}]}}0}}
\def\@newkeyenvc#1#2[#3]{\@newkeyenv{#1}{#2[{#3}]}}
\long\def\@newkeyenv#1#2{%
   \kcmd@plus \kcmd@unexpandchar@activate \fi
   \kcmd@keyenvir@def{#1}{#2}%
}
\long\def\kcmd@keyenvir@def#1#2#3#4{%
   \cslet{end#1}\protected
   \expandafter\kcmd@def\csname #1\expandafter\endcsname\csname end#1\endcsname#2{#3}{#4}%
}
```

\renew@keyenvironment

```
\def\renew@keyenvironment#1{%
  \@ifundefined{#1}%
     {\@latex@error{Environment #1 undefined}\@ehc
     }\relax
  \cslet{#1}\relax
  \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:

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

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

```
\newcommand\showcommandkeys[1]{\let\do\showcommandkey\docsvlist{#1}}
\newcommand\showcommandkey[1]{key \string"#1\string" = %
   \expandnext\expandnext\expandnext\detokenize{\commandkey{#1}}\par}
```

⟨/package⟩

4 Examples

```
⟨*example⟩
\ProvidesFile{keycommand-example}
\documentclass[a4paper]{article}
\usepackage[T1]{fontenc}
\usepackage[latin1]{inputenc}
\usepackage[american]{babel}
\usepackage{keycommand,framed,fancyvrb}
%
\makeatletter
\parindent\z@
\newkeycommand\Rule[raise=.4ex,width=1em,thick=.4pt][1]{%
```

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

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

\maketitle

{\Large Please refer to {\tt keycommand-example.tex} for definitions.}

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

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

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

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


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

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

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

```
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 ⟨/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